Dynamic Membership for Regular Languages

Antoine Amarilli
LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Louis Jachiet
LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Charles Paperman
Univ. Lille, CNRS, INRIA, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Abstract

We study the *dynamic membership problem* for regular languages: fix a language L, read a word w, build in time $O(|w|)$ a data structure indicating if w is in L, and maintain this structure efficiently under letter substitutions on w. We consider this problem on the unit cost RAM model with logarithmic word length, where the problem always has a solution in $O(\log |w| / \log \log |w|)$ per operation.

We show that the problem is in $O(\log \log |w|)$ for languages in an algebraically-defined, decidable class **QSG**, and that it is in $O(1)$ for another such class **QLZG**. We show that languages not in **QSG** admit a reduction from the prefix problem for a cyclic group, so that they require $\Omega(\log |w| / \log \log |w|)$ operations in the worst case; and that **QSG** languages not in **QLZG** admit a reduction from the prefix problem for the multiplicative monoid $U_1 = \{0, 1\}$, which we conjecture cannot be maintained in $O(1)$. This yields a conditional trichotomy. We also investigate intermediate cases between $O(1)$ and $O(\log \log |w|)$.

Our results are shown via the dynamic word problem for monoids and semigroups, for which we also give a classification. We thus solve open problems of the paper of Skovbjerg Frandsen, Miltersen, and Skyum [29] on the dynamic word problem, and additionally cover regular languages.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases regular language, membership, RAM model, updates, dynamic

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.116

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2102.07728`

Funding The authors have been partially supported by the ANR project EQUUS ANR-19-CE48-0019. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 431183758.

Acknowledgements We thank Jean-Éric Pin and Jorge Almeida for their advice on **ZG** and **SG**, and thank the ICALP referees for their helpful feedback.

1 Introduction

This paper studies how to handle letter substitution updates on a word while maintaining the information of whether the word belongs to a regular language. Specifically, we fix a regular language L – for instance $L = a^*b^*$. We are then given an input word w, e.g., $w = aaaa$. We first preprocess w in linear time to build a data structure, which we can use in particular to test if $w \in L$. Now, w is edited by letter substitutions, and we want to update w and keep track at each step of whether $w \in L$. For instance, an update can replace the third letter of w by a b, so that $w = aaba$, which is no longer in L. Then another update can replace, e.g., the fourth letter of w by a b, so that $w = aabb$, and now we have $w \in L$ again. Our problem, called *dynamic membership*, is to devise a data structure to handle such update

operations and determine whether $w \in L$, as efficiently as possible. We study this task from a theoretical angle, but it can also be useful in practice to maintain a Boolean condition (expressed as a regular language) on a user-edited word.

Dynamic membership was studied for various update operations, e.g., append operations for streaming algorithms or the sliding window model [14, 13, 15], letter substitutions for the dynamic word problem for monoids [29], or concatenations and splits [22]. It was also studied in the case of *pattern matching*, where we check if the word contains some target pattern [9], which is also assumed to be editable. It is also connected to the incremental validation problem, which has been studied for strings and for XML documents [5]. The problem was also studied from the angle of *dynamic complexity*, which does not restrict the running time but the logical language used to handle updates [16]; and very recently refined to a study of the amount of *parallel work* required [28].

Our focus in this work is to identify classes of fixed regular languages for which dynamic membership can be solved extremely efficiently, e.g., in constant time or sublogarithmic time. Our update language only allows letter substitutions to the input word, in particular the length of the input word can never be changed by updates. We make this choice because insertions and deletions already make it challenging to efficiently maintain the word itself (see Section 7). We work within the computational model of the unit-cost RAM, with logarithmic cell size.

Dynamic word problem for monoids [29]. Our problem closely relates to the work by Skovbjerg Frandsen, Miltersen, and Skyum on the *dynamic word problem for monoids* [29]: fix a finite monoid, read a *word* which is a sequence of monoid elements, and maintain under substitution updates the composition of these elements according to the monoid's internal law. Indeed, the dynamic membership problem for a language L reduces to the dynamic word problem for any monoid that recognizes L; but the converse is not true. Hence, studying the dynamic word problem for monoids is coarser than studying the dynamic membership problem for languages, although it is a natural first step and is already very challenging.

In the context of monoids, Skovbjerg Frandsen et al. [29] propose a general algorithm for the dynamic word problem that can handle each operation in time $O(\log n / \log\log n)$, for n the length of the word. This is a refinement of the elementary $O(\log n)$ algorithm that decomposes the word as a balanced binary tree whose nodes are annotated with the monoid image of the corresponding infix. They show that the $O(\log n / \log\log n)$ bound is tight for some monoids, namely noncommutative groups, and a generalization of them defined via an equation. This is obtained by a reduction from the so-called *prefix-$\mathbb{Z}_d$ problem*, for which an $\Omega(\log n / \log\log n)$ lower bound [12] is known in the cell probe model [17]. We will reuse this lower bound in our work.

They also show that the problem is easier for some monoids. For instance, commutative monoids can be maintained in $O(1)$, simply by maintaining the number of element occurrences. They also show a trickier $O(\log\log n)$ upper bound for *group-free monoids*: this is based on a so-called Krohn-Rhodes decomposition [27] and uses a predecessor data structure implemented as a van Emde Boas tree [34]. However, there are non-commutative monoids for which the problem is in $O(1)$ (as we will show), and there is still a gap between group-free monoids (with an upper bound in $O(\log\log n)$) and the monoids for which the $\Omega(\log n / \log\log n)$ lower bound applies. This was claimed as open in [29] and not addressed afterwards. While there is a more recent study by Pătraşcu and Tarniţă [23], it focuses on single-bit memory cells.

Our contributions. In this paper, we attack these problems using algebraic monoid theory. This unlocks new results: first on the dynamic word problem for monoids, where we extend the results of [29], and then on the dynamic membership problem for regular languages.

We start with our results on the dynamic word problem for monoids, which are summarized in Figure 1 along with a table of the main classes in Table 1. First, in Section 3, we show how a more elaborate $O(\log \log n)$ algorithm can cover all monoids to which the $\Omega(\log n / \log \log n)$ lower bound of [29] does not apply: we dub this class **SG** and characterize it by the equation $x^{\omega+1}yx^{\omega} = x^{\omega}yx^{\omega+1}$, for any elements x and y, where ω denotes the idempotent power. Our algorithm shares some ideas with the $O(\log \log n)$ algorithm of [29], in particular it uses van Emde Boas trees, but it faces significant new challenges. For instance, we can no longer use a Krohn-Rhodes decomposition, and proceed instead by a rather technical induction on the $\mathcal{J}$-classes of the monoid. Thus, we have an unconditional dichotomy between monoids in **SG**, which are in $O(\log \log n)$, and monoids outside of **SG**, which are in $\Theta(\log n / \log \log n)$.

Second, in Section 4, we generalize the $O(1)$ result on commutative monoids to the monoid class **ZG** [4]. This class is defined via the equation $x^{\omega+1}y = yx^{\omega+1}$, i.e., only the elements that are part of a group are required to commute with all other elements. We show that the dynamic word problem for these monoids is in $O(1)$: we use an algebraic characterization to reduce them to commutative monoids and to monoids obtained from nilpotent semigroups, for which we design a simple but somewhat surprising algorithm. We also show a conditional lower bound: for any monoid M not in **ZG**, we can reduce the *prefix-U_1* problem to the dynamic word problem for M. This is the problem of maintaining a binary word under letter substitution updates while answering queries asking if a prefix contains a 0. It can be seen as a priority queue (slightly weakened), so we conjecture that no $O(1)$ data structure for this problem exists in the RAM model. If this conjecture holds, **ZG** is exactly the class of monoids having a dynamic word problem in $O(1)$.

We then extend our results in Section 5 from monoids to the dynamic word problem for semigroups. Our results for **SG** extend directly: the upper bound on **SG** also applies to semigroups in **SG**, and semigroups not in **SG** must contain a submonoid not in **SG** so covered by the lower bound. For **ZG**, there are major complications, and we must study the class **LZG** of semigroups where all submonoids are in **ZG**. Semigroups not in **LZG** are covered by our conditional lower bound on prefix-U_1, but it is very tricky to show the converse, i.e., that imposing the condition on **LZG** suffices to ensure tractability. We do so by showing tractability for $\mathbf{ZG} * \mathbf{D}$, the semigroups generated by *semidirect products* of **ZG** semigroups and so-called *definite semigroups*, and by showing in [3] that $\mathbf{ZG} * \mathbf{D} = \mathbf{LZG}$, a *locality result* of possible independent interest.

Next, we extend our results in Section 6 from semigroups to languages. This is done using the notion of *stable semigroup* [6, 7], denoted as the **Q** operator; and specifically the class **QSG** of regular languages where the stable semigroup of the syntactic morphism in is **SG**, and the class **QLZG** where all local monoids of the stable semigroup of the syntactic morphism are in **ZG**. We obtain:

▶ **Theorem 1.1.** *Let L be a regular language, and consider the dynamic membership problem for L on the unit-cost RAM with logarithmic word length under letter substitution updates:*

- *If L is in the class* **QLZG***, then the problem is in $O(1)$.*
- *If L is not in the class* **QLZG** *but is in the class* **QSG***, then the dynamic membership problem is in $O(\log \log n)$ with n the length of the word. Further, solving it in $O(1)$ time gives an $O(1)$ implementation of a structure for the prefix-U_1 problem.*
- *If L is not in the class* **QSG***, then the dynamic membership problem is in $\Theta(\log n / \log \log n)$.*

We last present in Section 7 some extensions and questions for future work: preliminary observations on the precise complexity of languages in **QSG** $\setminus$ **QLZG** (as the $O(\log \log n)$ bound is not shown to be tight), the complexity of deciding which case of the theorem applies, the support for insertion and deletion updates, and the support for infix queries. Because of space constraints, the complete proofs of results are deferred to the full version [2].

2 Preliminaries and Problem Statement

Computation model. We work in the RAM model with unit cost, i.e., each cell can store integers of value at most polynomial in $O(|w|)$ where $|w|$ is the length of the input, and arithmetic operations (addition, successor, modulo, etc.) on two cells take unit time. As the integers have at most polynomial value, the memory usage is also polynomially bounded.

We consider *dynamic problems* where we are given an input word, preprocess it in linear time to build a data structure, and must then handle *update operations* on the word (by reflecting them in the data structure), and *query operations* on the current state of the word (using the data structure). The *complexity* of the problem is the worst-case complexity of handling an update or answering a query.

Like in [29], the lower bounds that we show actually hold in the coarser *cell probe* model, which only considers the number of memory cells accessed during a computation. Furthermore, they hold even without the assumption that the preprocessing is linear.

Given two dynamic problems A and B, we say that A has a *(constant-time) reduction* to B if we can implement a data structure for problem A having constant-time complexity when using as oracle constantly many data structures for problem B (built during the preprocessing). In other words, queries and updates on the structure for A can perform constant-time computations using its own memory, but they can also use the data structures for B as an oracle, i.e., perform a constant number of queries and updates on them, which are considered to run in $O(1)$. We similarly talk of a dynamic problem having a *(constant-time) reduction* to multiple problems, meaning we can use all of them as oracle. If problem A reduces to problems $B_1, \ldots, B_n$, then any complexity upper bound that holds on all problems $B_1, \ldots, B_n$ also holds for A, and any complexity lower bound on A extends to at least one of the B_i.

Problem statement. Our problems require some algebraic prerequisites. We refer the reader to the book of Pin [25] and his lecture notes [26] for more details. A *semigroup* is a set S equipped with an associative composition law (written multiplicatively), and a *monoid* is a semigroup M with a *neutral element*, i.e., an element 1 such that $1x = x1 = x$ for all $x \in M$; the neutral element is then unique. One example of a monoid is the *free monoid* Σ^* defined for a finite alphabet Σ and consisting of all words with letters in Σ (including the empty word), with concatenation as its law. Except for the free monoid, all semigroups and monoids considered are finite.

A semigroup element $x \in S$ is *idempotent* if $xx = x$. For $x \in S$, we denote by ω the idempotent power of x, i.e., the least positive integer such that x^ω is idempotent. A *zero* for S is an element $0 \in S$ such that $0x = x0 = 0$ for all $x \in S$: if it exists, it is unique. Given a semigroup S, we write S^1 for the monoid obtained by adding a fresh neutral element to S if it does not already have one.

A *morphism* from a semigroup S to a semigroup S' is a map $\mu \colon S \to S'$ such that for any $x, y \in S$, we have $\mu(xy) = \mu(x)\mu(y)$. A *morphism* from a monoid M to a monoid M' must additionally verify that $\mu(1) = 1'$, for 1 and $1'$ the neutral elements of M and M' respectively.

The *direct product* of two monoids M_1 and M_2 is $M_1 \times M_2$ with componentwise composition; it is also a monoid. A *quotient* of a monoid M is a monoid M' such that there is a surjective morphism from M to M'. A *submonoid* is a subset of a monoid which is also a monoid. The analogous notions for semigroups are defined in the expected way. A *pseudovariety* of monoids (resp., semigroups) is a class of monoids (resp., semigroups) closed under direct product, quotient and submonoid (resp., subsemigroup). The pseudovariety of monoids (resp., semigroups) *generated* by a class $\mathbf{V}$ of monoids (resp., of semigroups) is the least pseudovariety closed under these operations and containing $\mathbf{V}$. As we are working with finite semigroups and monoids, we refer to pseudovarieties simply as varieties.

We consider dynamic problems where we maintain a word w on a finite alphabet Σ, every letter being stored in a cell. We allow *letter substitution* updates of the form (i, a) for $1 \leq i \leq |w|$ and $a \in \Sigma$. The letter substitution update (i, a) replaces the i-th letter of w by a; the size $|w|$ of the word never changes. Given the input word w, we first preprocess it in time $O(|w|)$ to build a data structure. The data structure must then support update operations for letter substitution updates, and some query operations (to be defined below). The complexity that we measure is the worst-case complexity of an update operation or query operation, as a function of $|w|$. Our definition does not limit the memory used. However, all our upper complexity bounds actually have memory usage in $O(|w|)$. Further, all our lower bounds hold even when no assumption is made on the memory usage.

We focus on three related dynamic problems, allowing different query operations. The first is the *dynamic word problem for monoids*: fix a monoid M, the alphabet Σ is M, and the query returns the *evaluation* of the current word w, i.e., the product of the elements of w (it is an element of M). This is the problem studied in [29]. The second is the *dynamic word problem for semigroups*, which is the same but with a semigroup, and assuming that $|w| > 0$. The third is the *dynamic membership problem for regular languages*: we fix a regular language L on the alphabet Σ, and the query checks whether the current word belongs to L.

We study the data complexity of these problems in the rest of this paper, i.e., the complexity expressed as a function of w, with the monoid, semigroup, or language being fixed. Let us first observe that, for monoids and more generally for semigroups, the usual algebraic operators of quotient, subsemigroup, and direct product, do not increase the complexity of the problem:

▶ **Proposition 2.1.** *Let S and T be finite semigroups. The dynamic word problem of subsemigroups or quotients of S reduces to the same problem for S, and the dynamic word problem of $S \times T$ reduces to the same problem for S and T.*

Hard problems. All our lower bounds are obtained by reducing from the problem *prefix-M*, for M a fixed monoid. In this problem, we maintain a word of M^* under letter substitution updates, and handle *prefix queries*: given a prefix length, return the evaluation of the prefix of that length.

In particular, for $d \geq 2$, we consider the problem *prefix-$\mathbb{Z}_d$* for $\mathbb{Z}_d$ the cyclic group of order d, i.e., $\mathbb{Z}_d = \{0, \ldots, d-1\}$ with addition modulo d, where the evaluation of prefix is the sum of its elements modulo d. The following lower bound is known already in the cell probe model:

▶ **Theorem 2.2** ([12, 29])**.** *For any fixed $d \geq 2$, any structure for prefix-$\mathbb{Z}_d$ on a word of length n has complexity $\Omega(\log n / \log \log n)$.*

We also consider the problem *prefix-U_1*, where $U_1 = \{0, 1\}$ is the multiplicative monoid whose composition is the logical AND, i.e., prefix queries check if the prefix contains an occurrence of 0. Equivalently, we must maintain a subset S of a universe $\{1, \ldots, n\}$ (intuitively

n is the length of the word) under insertions and deletions, and support *threshold queries* that ask, given $0 \leq i \leq n$, whether S contains some element which is $\leq i$ (i.e., if some position before i has a 0). The prefix-U_1 problem can be solved in $O(\log \log n)$ [33] with a priority queue data structure, or even in expected $O(\sqrt{\log \log n})$ if we allow randomization [18]. Note that prefix-U_1 is slightly weaker than a priority queue as we can only *compare* the minimal value to a value given as input. We do not know of lower bounds on prefix-U_1, but conjecture [20] that it cannot be solved in $O(1)$:

▶ **Conjecture 2.3.** *There is no structure for prefix-U_1 with complexity $O(1)$.*

Note that the best algorithm for prefix-U_1 works by sorting small sets of large integers. This takes linear time in the cell probe model, so does not rule out the existence of an $O(1)$ priority queue [33]. Hence, a lower bound for prefix-U_1 would need to apply to the RAM model specifically, which would require new techniques.

Our last hard problem is prefix-U_2 where U_2 is the monoid $\{1, a, b\}$ with composition law $xy = y$ for $x, y \in \{a, b\}$, i.e., queries check if the last non-neutral element is a or b (or nothing). By adapting known results on the *colored predecessor problem* [24], we have:

▶ **Theorem 2.4** (Adapted from [24])**.** *Any structure for prefix-U_2 on a word of length n must be in $\Omega(\log \log n)$.*

General algorithms. Of course, the "hard" prefix problems, and the three problems that we study, can all be solved in $O(|w|)$ by re-reading the whole word at each update. We can improve this to $O(\log |w|)$ by maintaining a balanced binary tree on the letters of the word, with each node of the tree carrying the evaluation in the monoid of the letters reachable from that node. Any letter substitution update on the word can be propagated up to the root in logarithmic time, and the annotation of the root is the evaluation of the word. This algorithm has been implemented in practice [22]. A finer bound is given in [29] using a folklore technique of working with $(\log n)$-ary trees rather than binary trees, and using the power of the RAM model. We recall it here for monoids (but it applies to all three problems):

▶ **Theorem 2.5** ([29])**.** *For any fixed monoid M, the dynamic word problem and prefix problem for M are in $O(\log n / \log \log n)$.*

Our goal in this paper is to solve the dynamic word problem and dynamic membership problem more efficiently for specific classes of monoids, semigroups, and languages. We start our study with monoids in the next two sections, by studying the varieties **SG** and **ZG**.

3 Dynamic Word Problem for Monoids in SG

We define the class **SG** of monoids by the equation $x^{\omega+1}yx^{\omega} = x^{\omega}yx^{\omega+1}$ for all x, y. It incidentally occurs in [10, Theorem 3.1], but to our knowledge was not otherwise studied. The name **SG** means *swappable groups*. Intuitively, a monoid M is in **SG** iff, for any two elements $r, t \in M$ belonging to the same subgroup of M, we can *swap* them, i.e., $rst = tsr$ for all $s \in M$. We first recall the lower bound from [29] on the dynamic word problem for monoids *not in* **SG**, and then show an upper bound for monoids in **SG**.

Lower bound. The monoids not in **SG** are in fact those covered by the lower bound of [29]. Namely, we have the following, implying the $\Omega(\log n / \log \log n)$ lower bound by Theorem 2.2:

▶ **Theorem 3.1** ([29], Theorem 2.5.1)**.** *For any monoid M not in* **SG***, there exists $d \geq 2$ such that the prefix-$\mathbb{Z}_d$ problem reduces to the dynamic word problem for M.*

Upper bound. The rest of this section presents our upper bound on monoids in **SG**. In fact, we show a more general claim on the dynamic word problem for *semigroups* in **SG**, i.e., those satisfying the equation of **SG**. This covers in particular the monoids of **SG**:

▶ **Theorem 3.2.** *The dynamic word problem for any semigroup in* **SG** *is in* $O(\log \log n)$.

This result extends the result of [29] on group-free monoids, because **SG** contains all aperiodic monoids and all commutative monoids. Indeed, an aperiodic monoid satisfies the equation $x^{\omega+1} = x^\omega$, and then $x^{\omega+1}yx^\omega = x^\omega yx^\omega = x^\omega yx^{\omega+1}$. Besides, commutative monoids clearly satisfy the equation. Of course, **SG** captures monoids not covered by [29], e.g., products of a commutative monoid and an aperiodic monoid.

The result of [29] uses van Emde Boas trees [34], which we extend to store values in an alphabet Σ. Fixing an alphabet Σ, a *vEB tree* (or *vEB*) is a data structure parametrized by an integer n called its *span*, which stores a set $X \subseteq \{1, \ldots, n\}$ and a mapping $\mu\colon X \to \Sigma$, and supports the following operations: *inserting* an integer $x \in \{1, \ldots, n\} \setminus X$ with a label $\mu(x) := a$; *retrieving* the label of $x \in \{1, \ldots, n\}$ if $x \in X$ (or a special value if $x \notin X$); *removing* an integer $x \in X$ and its label; *finding the next integer* of X that follows an input $x \in \{1, \ldots, n\}$ (or a special value if none exists); and *finding the previous integer*.

We can implement vEBs so that these five operations run in $O(\log \log n)$ time in the worst case, and so that a vEB can be constructed in linear time from an ordered list.

We use vEBs in our inductive proof to represent words with "gaps": a vEB represents the word obtained by concatenating the labels of the elements of X in order. For a semigroup S and span $n \in \mathbb{N}$, the *dynamic word problem on vEBs* for S is to maintain a vEB T of span n on alphabet S under insertions and deletions, and to answer queries asking the evaluation in S of the word currently represented by T. As before, the complexity is the worst-case complexity of an insertion, deletion, or query, measured as a function of the span n (which never changes). The data structure for this problem must be initialized during a preprocessing phase on the initial vEB T, which must run in $O(n)$. Note that when T is empty then its evaluation is not expressible in the semigroup S: we then return a special value.

It is then clear that Theorem 3.2 follows from its generalization to vEBs, as a word in the usual sense can be converted in linear time to a vEB where $X = \{1, \ldots, n\}$:

▶ **Theorem 3.3.** *Let S be a semigroup in* **SG**. *The dynamic word problem for S on a vEB of span n is in $O(\log \log n)$.*

We show this result in the rest of the section. We assume without loss of generality that S has a zero, as otherwise we can simply add one. We first introduce some algebraic preliminaries, and then present the proof, which is an induction on $\mathcal{J}$-classes of the semigroup.

Preliminaries and proof structure. The *$\mathcal{J}$-order* of S is the preorder $\leq_\mathcal{J}$ on S defined by $s \leq_\mathcal{J} s'$ if $S^1sS^1 \subseteq S^1s'S^1$, recalling that S^1 is the monoid where we add a neutral element to S if it does not already have one. The equivalence classes of the symmetric closure of this preorder are called *$\mathcal{J}$-classes*. We lift the $\mathcal{J}$-order to $\mathcal{J}$-classes C, C' by writing $C \leq_\mathcal{J} C'$ if $u \leq_\mathcal{J} v$ for all $u \in C$ and $u' \in C'$. A $\mathcal{J}$-class is *maximal* if it is maximal for this order.

We show Theorem 3.3 by induction on the number of $\mathcal{J}$-classes of the semigroup. More precisely, at every step, we consider a maximal $\mathcal{J}$-class C, and remove it by reducing to the semigroup $S \setminus C$. Remember that the number of classes only depends on the fixed semigroup S, so it is constant. Thus, as the constant number of operations on vEBs at each class each take time $O(\log \log n)$, the overall bound is indeed in $O(\log \log n)$.

The base case of the induction is that of a semigroup with a single $\mathcal{J}$-class; from our assumption that the semigroup has a zero, that $\mathcal{J}$-class must then consist of the zero, i.e., we have the trivial monoid $\{0\}$, and the image is always 0 (or undefined if the word is empty).

We now show the induction step of Theorem 3.3. Take a semigroup S with more than one $\mathcal{J}$-class, and fix a maximal $\mathcal{J}$-class C of S: we know that $S \setminus C$ is not empty. What is more:

▷ Claim 3.4. For any x, y of S with $xy \in C$, then $x \in C$ and $y \in C$.

Thus, whenever a combination of elements "falls" outside of the maximal class C, then it remains in $S \setminus C$; and we can see $S \setminus C$ as a semigroup, which still has a zero, and has strictly less $\mathcal{J}$-classes. So we will study how to reduce to $S \setminus C$. We now make a case disjunction depending on whether C is *regular*, i.e., whether it contains an idempotent element.

Non-regular maximal classes. This case is easy, because products of two or more elements of C are never in C. To formalize this property, for a maximal $\mathcal{J}$-class C of S, we call a word w on S^* *pair-collapsing* for C if the product of any two adjacent letters of w is in $S \setminus C$. We show:

▶ **Lemma 3.5.** *Let C be a maximal $\mathcal{J}$-class. If C is non-regular, then any word is pair-collapsing: for any $x, y \in C$, we have $xy \in S \setminus C$.*

We can then show the following, which we will reuse for regular maximal classes:

▶ **Lemma 3.6.** *Let S be a semigroup and let C be a maximal $\mathcal{J}$-class of S. Consider the dynamic word problem for S on vEBs of some span n where we assume that, at every step, the represented word is pair-collapsing for C. Then that problem reduces to the dynamic word problem for $S \setminus C$ on vEBs of span n.*

Thanks to Lemma 3.5, this allows us to settle the case of a non-regular maximal $\mathcal{J}$-class, using the induction hypothesis to maintain the problem on $S \setminus C$.

Case of a regular maximal class. We now consider a maximal $\mathcal{J}$-class C that is regular. Consider the semigroup $C^0 := C \cup \{0\}$ for a fresh zero 0, i.e., the multiplication is that of C except that $x0 = 0x = 0$ for all $x \in C^0$, and that $xy = 0$ in C^0 for all $x, y \in C$ for which the product xy in S is not an element of C. Note that 0 is unrelated to the zero which S was assumed to have; intuitively, the 0 of C^0 stands for combinations of elements that are not in C. Another way to see C^0 is as the quotient of S by the ideal $S \setminus C$, i.e., we identify all elements of $S \setminus C$ to 0. By Prop. 4.35 of Chapter V of [26], we know that C^0 is a so-called *0-simple semigroup*. By the Rees-Sushkevich theorem (Theorem 3.33 of [26]), S is isomorphic to some *Rees matrix semigroup with 0*. This is a semigroup $M^0(G, I, J, P)$ with I and J two non-empty sets, G a group called the *structuring group*, and P a matrix indexed by $J \times I$ having values in G^0. The elements of the semigroup are the elements of $I \times G \times J$ and the element 0, with $x0 = 0x = 0$ for any element $x \in I \times G \times J$, and for (i, g, j) and (i', g', j') two elements of $I \times G \times J$, their product is 0 if $p_{j,i'} = 0$, and $(i, gp_{j,i'}g', j')$ otherwise.

With this representation, the idea is to collapse together the maximal runs of consecutive elements of C^0 whose product is not 0, i.e., does not "fall" outside of C. Once this is done, the product of two elements always falls in $S \setminus C$, so we can conclude using Lemma 3.6.

However, we cannot do this in a naive fashion. For instance, if we insert a letter in the vEB in the middle of such a maximal run, we cannot hope to split the run and know the exact group annotation of the two new runs – this could amount to solving a prefix-$\mathbb{Z}_d$ problem. Instead, we must now use the fact that S is in **SG**, and derive the consequences

of the equation in terms of the Rees-Sushkevich representation. Intuitively, the equation ensures that the structuring group G is commutative, and that annotations in G can "move around" relative to other elements in S without changing the evaluation. Formally:

▷ Claim 3.7. The structuring group G is commutative.

▷ Claim 3.8. Let $r, s, t \in S^*$ and $(i, g, j), (i', g', j') \in I \times G \times J$. Write $w = r(i, g, j)s(i', g', j')t$ and $w' = r(i, gg', j)s(i', 1, j')t$ where 1 is the neutral element of G. Then $\mathrm{eval}(w) = \mathrm{eval}(w')$.

This allows us to reduce the dynamic word problem on S to the same problem where we assume that the word is always pair-collapsing:

▷ Claim 3.9. The dynamic word problem for S on vEBs (of some span n) reduces to the same problem on vEBs of span n where we additionally require that, at every step, the represented word is pair-collapsing for the maximal $\mathcal{J}$-class C.

Proof sketch. We maintain a mapping where all maximal runs of word elements evaluating to C are collapsed to a single element, which we can evaluate following the Rees-Sushkevich representation. The tricky case is whenever an update breaks a maximal run into two parts: we cannot recover the G-component of the annotation of each part, but we use Claim 3.8 to argue that we can simply put it on the left part without altering the evaluation in S. ◁

This claim together with Lemma 3.6 implies that the dynamic word problem for S reduces to the same problem for $S \setminus C$, for which we use the induction hypothesis. This establishes the induction step and concludes the proof of Theorem 3.2.

4 Dynamic Word Problem for Monoids in ZG

We pursue our study of the dynamic word problem for monoids with the class **ZG**, introduced in [4] and defined by the equation $x^{\omega+1}y = yx^{\omega+1}$ for all x, y. This asserts that elements of the form $x^{\omega+1}$, which are the ones belonging to a *subgroup* of the monoid, are *central*, i.e., commute with all other elements. By the equations, and recalling that $x^{\omega+1} = x^\omega x^{\omega+1}$, clearly **ZG** $\subseteq$ **SG**. In this section, we show an $O(1)$ upper bound on the dynamic word problem for monoids in **ZG**, and a conditional lower bound for any monoid not in **ZG**.

Upper bound. Recall the result on commutative monoids from [29]:

▶ **Theorem 4.1** ([29]). *The dynamic word problem for any commutative monoid is in* $O(1)$.

Our goal is to generalize it to the following result:

▶ **Theorem 4.2.** *The dynamic word problem for any monoid in* **ZG** *is in* $O(1)$.

This generalizes Theorem 4.1 (as commutative monoids are clearly in **ZG**) and covers other monoids, e.g., the monoid $M = \{1, a, b, ab, 0\}$ with $a^2 = b^2 = ba = 0$, where it intuitively suffices to track the position of a's and b's and compare them if there is only one of each.

We now prove Theorem 4.2. A semigroup S is *nilpotent* if it has a zero and there exists $k > 0$ such that $S^k = \{0\}$, i.e., all products of k elements are equal to 0. Alternatively [26, Chapter X, Section 4], S is nilpotent iff it satisfies the equation $x^\omega y = yx^\omega = x^\omega$. We then consider the monoids of the form S^1 where S is nilpotent – an example of this is the monoid M described above. The variety generated by such monoids is called **MNil** and was studied by Straubing [30]. We can show:

▶ **Proposition 4.3.** *For any nilpotent S, the dynamic word problem for S^1 is in $O(1)$.*

Proof sketch. We maintain a (non-sorted) doubly-linked list L of the positions of the word w that contain a non-neutral element. As S is nilpotent, the evaluation of w is 0 unless constantly many non-neutral letters remain, which we can then find in $O(1)$ with L. ◀

In [3] we show that **ZG** is generated by such monoids S^1 and by commutative monoids:

▶ **Proposition 4.4** (Corollary 3.6 of [3]). *The variety* **ZG** *is generated by commutative monoids and monoids of the form S^1 for S a nilpotent semigroup.*

In view of Theorem 4.1 and Proposition 4.3, the dynamic word problem is in $O(1)$ for the semigroups that generate the variety **ZG** (Proposition 4.4). Theorem 4.2 then follows from Proposition 2.1.

Lower bound. We now show a conditional lower bound on the dynamic word problem for monoids outside of **ZG**. To do this, we will reduce from the prefix-U_1 problem:

▶ **Theorem 4.5.** *For any monoid M in $\mathbf{SG} \setminus \mathbf{ZG}$, the prefix-$U_1$ problem reduces to the dynamic word problem for M.*

Proof sketch. We consider the variety **ZE** [1] of monoids whose idempotents are central, i.e., the variety defined by the equation $x^\omega y = yx^\omega$. We show that $\mathbf{ZG} = \mathbf{SG} \cap \mathbf{ZE}$. We then show that, for any monoid not in **ZE**, we can reduce from the prefix-U_1 problem by encoding the elements 0 and 1 of U_1 using carefully chosen elements of the monoid. ◀

Using Conjecture 2.3, and together with Theorem 3.1 for the monoids not in **SG**, this implies a conditional super-constant lower bound for monoids outside **ZG**.

5 Dynamic Word Problem for Semigroups

We have classified the complexity of the dynamic word problem for monoids: it is in $O(\log \log n)$ for monoids in **SG**, in $O(1)$ for monoids in **ZG**, in $\Omega(\log n / \log \log n)$ for monoids not in **SG**, and non-constant for monoids not in **ZG** conditionally to Conjecture 2.3. In this section, we extend our results from monoids to *semigroups*.

Submonoids and local monoids. A *submonoid* of a semigroup S is a subset of the semigroup which is stable under its composition law and is a monoid. We first notice via Proposition 2.1 that a semigroup that contains a hard submonoid is also hard:

▷ Claim 5.1. The dynamic word problem for any submonoid of a semigroup S reduces to the same problem for S.

We will investigate if studying the submonoids of a semigroup suffices to understand the complexity of its dynamic word problem. To do so, we focus on a certain kind of submonoids: the *local monoids*. A submonoid N of S is a *local monoid* if there exists an idempotent element e of S such that $N = eSe$, i.e., N is the set of elements that can be written as ese for some $s \in S$. The point of local monoids is that they are maximal in the sense that every submonoid T of S is a submonoid of a local monoid: indeed, taking 1 the neutral element of T, all elements of T can be written as $1T1 \subseteq 1S1$ and $1S1$ is a local monoid. For **V** a variety of monoids, we denote by **LV** the variety of semigroups such that all local monoids are in **V**. As we explained, this is equivalent to imposing that all submonoids are in **V** (since varieties are closed under the submonoid operation).

Case of SG. We now revisit our results on monoids to extend them to semigroups, starting with **SG**. We denote by **LSG** the variety of semigroups whose local monoids are in **SG**. We show that a semigroup where all local monoids are in **SG** must itself be in **SG**:

▷ Claim 5.2. We have **LSG** = **SG** as varieties of semigroups.

Semigroups in **SG** are already covered by our upper bound (Theorem 3.2), and semigroups not in **LSG** have a submonoid not in **SG**, so we can use Claim 5.1 and Theorem 3.1. Hence:

▶ **Corollary 5.3.** *Let S be a semigroup. If S is in* **SG***, then the dynamic word problem for S is in $O(\log\log n)$. Otherwise, the dynamic word problem for S is in $\Omega(\log n/\log\log n)$.*

Case of ZG. The variety **ZG** is not equal to **LZG**. For instance, let S be the syntactic semigroup of a^*b^*, that is the semigroup $\{a, b, ab, 0\}$ defined with $a^2 = a$, $b^2 = b$ and $ba = 0$. It is not in **ZG**, since a and b are idempotents that do not commute. However, its local monoids are either trivial or U_1, so they are all in **ZG**, showing that this semigroup is in **LZG**. Still, we can extend our characterization from monoids to semigroups up to studying **LZG**:

▶ **Theorem 5.4.** *Let S be a semigroup. If S is in* **LZG***, then the dynamic word problem for S is in $O(1)$. Otherwise, unless prefix-U_1 is in $O(1)$, the dynamic word problem for S is not in $O(1)$.*

The second part of the claim is by Claim 5.1 and Theorem 3.1, but the first part is much trickier. We use a characterization of **LZG** as a *semidirect product* $\mathbf{ZG} * \mathbf{D}$, which follows from a very technical *locality result* on **ZG** [3], and then design an algorithm for the dynamic word problem for semigroups in $\mathbf{ZG} * \mathbf{D}$. We prove Theorem 5.4 in the rest of this section.

Given two semigroups S and T, a *semigroup action* of S on T is defined by a map $\mathrm{act} : S \times T \to T$ such that $\mathrm{act}(s_1, \mathrm{act}(s_2, t)) = \mathrm{act}(s_1 s_2, t)$ and $\mathrm{act}(s, t_1 t_2) = \mathrm{act}(s, t_1)\mathrm{act}(s, t_2)$. We then define the product $\circ_{\mathrm{act}}$ on the set $T \times S$ as follows: for all s_1, s_2 in S and t_1, t_2 in T, we have: $(t_1, s_1) \circ_{\mathrm{act}} (t_2, s_2) := (t_1 \mathrm{act}(s_1, t_2), s_1 s_2)$. The set $T \times S$ equipped with the product $\circ_{\mathrm{act}}$ is a semigroup called the *semidirect product* of S by T, denoted $T \circ_{\mathrm{act}} S$.

We say that a semigroup D is *definite* if there exists an integer $k \in \mathbb{N}$ such that for all $y, x_1, \ldots, x_k$ in D, we have $yx_1 \cdots x_k = x_1 \cdots x_k$. Alternatively, a semigroup is definite iff it satisfies the equation $yx^\omega = x^\omega$ [31, Proposition 2.2] for all x, y in D. In particular, every nilpotent semigroup is definite. We write **D** for the variety of definite semigroups.

Our alternative definition of **LZG** will be the variety of semigroups $\mathbf{ZG} * \mathbf{D}$ generated by semigroups that are the semidirect product of a **ZG** monoid by a definite semigroup.

The variety $\mathbf{ZG} * \mathbf{D}$ of semigroups is not immediately related to the variety **LZG** defined above. One can easily show that $\mathbf{ZG} * \mathbf{D} \subseteq \mathbf{LZG}$, but the other direction is much more challenging to establish. We show this as a so-called *locality theorem*, which we defer to a separate paper [3] because it uses different tools and is of possible independent interest:

▶ **Theorem 5.5** ([3], Theorem 1.1)**.** *We have:* $\mathbf{ZG} * \mathbf{D} = \mathbf{LZG}$.

To conclude the proof of Theorem 5.4, by the locality theorem above, it suffices to solve the dynamic word problem for semigroups in $\mathbf{ZG} * \mathbf{D}$. By Proposition 2.1, it suffices to consider the semigroups that generate the variety. We do this below, establishing Theorem 5.4:

▶ **Proposition 5.6.** *Let S be a definite semigroup, let T be a semigroup of* **ZG***, and let* act *be a semigroup action of S on T. The dynamic word problem for the semigroup $T \circ_{\mathrm{act}} S$ reduces to the same problem for T.*

Proof sketch. We express the direct product of the letters of the input word as a product involving elements of T and prefix sums of elements of S, which we can maintain in $O(1)$. ◀

6 Dynamic Word Problem for Languages

We now turn to the dynamic membership problem for regular languages, and show Theorem 1.1 using the three previous sections and some extra algebraic results.

Connection to the dynamic word problem. A regular language L is *recognized* by a finite monoid if there exists a morphism $\eta\colon \Sigma^* \to M$ such that $L = \eta^{-1}(\eta(L))$. The *syntactic congruence* of L is the equivalence relation on Σ^* where $u, v \in \Sigma^*$ are equivalent iff, for each $r, t \in \Sigma^*$, either $rut \in L$ and $rvt \in L$, or $rut \notin L$ and $rvt \notin L$. The *syntactic monoid* M of L is the quotient of Σ^* by the syntactic congruence of L, and the *syntactic morphism* is the morphism mapping Σ^* to M; the syntactic morphism witnesses that the syntactic monoid recognizes L.

The dynamic membership problem for a language clearly reduces to the dynamic word problem for its syntactic monoid. However, the converse is not true: the language $L := (aa)^*ba^*$ has a syntactic monoid M that can be shown to be outside of **SG**, but we can solve dynamic membership for L in $O(1)$ by counting the b's at even and odd positions. Intuitively, M has a neutral element 1 so that the dynamic word problem for M has a reduction from prefix-$\mathbb{Z}_2$, but 1 is not achieved by a letter of the alphabet so dynamic membership for L is easier.

We extend our results to languages using the notion of *stable semigroup* [6, 7]. This allows us to remove the neutral element (as it is a semigroup not a monoid) and ensures that all semigroup elements can be achieved by subwords of some constant length (the *stability index*).

Formally, let L be a regular language and $\eta\colon \Sigma^* \to M$ its syntactic morphism. The *powerset* of M is the monoid whose elements are subsets of M and for $E, F \subseteq M$, the product EF is $\{xy \mid x \in E, y \in F\}$. The *stability index* of L is the idempotent power s of $\eta(\Sigma)$ in the powerset monoid. Intuitively, this choice of s ensures that, for any two words $w_1, w_2 \in \Sigma^s$, the value $\eta(w_1w_2)$ of their concatenation in the monoid can be achieved by another word of Σ^s, i.e., $\eta(w_1w_2) = \eta(w)$ for some $w \in \Sigma^s$. Then $\eta(\Sigma^s)$ is a subsemigroup of M, because $(\eta(\Sigma^s))^2 = \eta(\Sigma^s)$: we call it the *stable semigroup* of L. For any class of semigroups $\mathbf{V}$, we denote by $\mathbf{QV}$ the class of languages whose stable semigroup is in $\mathbf{V}$.

Upper bounds. We first show that the dynamic membership problem for a regular language reduces more specifically to the dynamic word problem for its *stable semigroup*:

▶ **Proposition 6.1.** *Let L be a regular language. The dynamic membership problem for L reduces to the dynamic word problem for the stable semigroup of L.*

Proof sketch. We partition the word of L into chunks of size s (plus one of size $\leq s$) for s the stability index, and feed them to the data structure for the stable semigroup of L. ◀

By Corollary 5.3 and Theorem 5.4, this implies that languages in **QSG** (resp., in **QLZG**) have a dynamic membership problem in $O(\log \log n)$ (resp., in $O(1)$).

Lower bounds. We now show that languages whose stable semigroup is not in **LV** admit a reduction from the dynamic word problem of a monoid of **V**.

▶ **Proposition 6.2.** *Let $\mathbf{V}$ be a variety of monoids and let L be a regular language not in $\mathbf{QLV}$. There is a monoid not in $\mathbf{V}$ whose dynamic word problem reduces to the dynamic membership problem for L.*

Proof sketch. If L is not in **QLV**, then its stable semigroup contains a submonoid M not in **V**, and all elements can be achieved by a block of s letters for s the stability index. ◀

Again by Corollary 5.3 and Theorem 5.4, we deduce that languages outside of **QSG** have complexity at least $\Omega(\log n / \log\log n)$. Further, assuming Conjecture 2.3, and languages outside of **QLZG** do not have complexity $O(1)$.

7 Extensions, Problem Variants, and Future Work

We have presented our results on the dynamic word problem for monoids and semigroups, and on the dynamic membership problem for regular languages. We conclude the paper by a discussion of problems for further study. We first discuss the question of *intermediate complexities* between $O(1)$ and $O(\log\log n)$. We then study the complexity of *deciding which case applies* as a function of the target language, semigroup, or monoid. We last explore the issue of *handling insertions and deletions* on the input word, and of supporting *infix queries*.

Intermediate complexities. Our $O(\log\log n)$ upper bound in Theorem 3.2 and its variants may not be tight. Still, we can identify a language L_{U_2} in $\mathbf{QSG} \setminus \mathbf{QLZG}$ for which we show that the dynamic membership problem is in $\Theta(\log\log n)$ (even allowing randomization and allowing a probably correct answer), because the prefix-U_2 problem reduces to it.

We can also identify a language of $\mathbf{QSG} \setminus \mathbf{QLZG}$ that reduces to prefix-U_1 and so can be solved in expected $O(\sqrt{\log\log n})$. This shows that languages in $\mathbf{QSG} \setminus \mathbf{QLZG}$ have different complexity regimes, at least when allowing randomization.

▶ **Proposition 7.1.** *There is a language L_{U_2} in $\mathbf{QSG} \setminus \mathbf{QLZG}$ which is equivalent to prefix-U_2 under constant-time reductions, and a language L_{U_1} in $\mathbf{QSG} \setminus \mathbf{QLZG}$ which is equivalent to prefix-U_1 under constant-time reductions.*

Deciding which case applies. A natural question about our results is the question of efficiently identifying, given a regular language, which of the cases of Theorem 1.1 applies, or in particular of determining, given an input monoid or semigroup, if it is in **SG**, or in **ZG**. This depends on how the input is represented. If we are given a monoid explicitly (as a table of its operations), then the equations of **ZG** and of **SG** can be checked in polynomial time. If the monoid is represented more concisely as the transition monoid of some automaton, then the verification can be performed in PSPACE. We do not know if the problems are PSPACE-hard, though this seems likely at least for **SG** because of its proximity to aperiodic monoids [8]. We leave open the precise complexities of this task, in particular for the **L** and **Q** operators.

Handling insertions and deletions. Another natural question is to handle insertion and deletion updates, i.e., *inserting* letter a at position k transforms the word $w_1 \cdots w_{k-1} w_k \cdots w_n$ into $w_1 \cdots w_{k-1} a w_k \cdots w_n$, and *deleting* at position k does the opposite. Any regular language can be maintained under such updates in $O(\log n)$ using a Fenwick tree, but it makes the problem much harder for most languages. For example, if the alphabet has two letters a and b, just testing if the word that we maintain contains an a requires $\Omega(\log n / \log\log n)$ by [21]. This is why we do not study such updates in this work. Interestingly, notice that our algorithm in Theorem 3.3 supports insertions and deletions on words represented as vEBs, but the semantics are different (they use explicit positions in a fixed range).

Infix queries. A natural extension of dynamic membership for a regular language L is the *dynamic infix membership problem*, where we can query any *infix* of the word (identified by its endpoints) to ask whether it is in L. The $O(\log n/\log\log n)$ algorithm of Theorem 2.5 supports this, and so can the $O(\log\log n)$ algorithm of [29] for group-free monoids. However, the infix problem can be harder. Consider for instance the language L_2 on $\{a, b\}$ of words with an even number of a's. Dynamic membership has complexity $O(1)$ because L_2 is commutative, but infix queries (or even prefix queries) require $\Omega(\log n/\log\log n)$ as prefix-$\mathbb{Z}_2$ reduces to it.

We leave open the study of the complexity of the infix problem. We note, however, that this problem for a language L can be studied as the dynamic membership problem for a regular language defined from L. So our results cover the infix problem via this transformation; we leave to future work a characterization of the resulting classes.

▷ Claim 7.2. For any fixed regular language L, the dynamic infix membership problem is equivalent up to constant-time reductions to the dynamic membership problem for the language $\Sigma^* x L x \Sigma^*$ where x is a fresh letter.

Other open questions. A natural question for future work would be to study the complexity of our problems in weaker models, e.g., pointer machines [32], or machines with counters. One could also extend our study to languages that are not regular, e.g., generalizing bounds on maintaining the language of well-parenthesized strings ([19, Proposition 1] and [11]).

References

1 Jorge Almeida and Assis Azevedo. Implicit operations on certain classes of semigroups. In *Semigroups and their Applications*. Springer, 1987.

2 Antoine Amarilli, Louis Jachiet, and Charles Paperman. Dynamic membership for regular languages, 2021. `arXiv:2003.02576`.

3 Antoine Amarilli and Charles Paperman. Locality and centrality: The variety **ZG**, 2021. `arXiv:2102.07724`.

4 Karl Auinger. Semigroups with central idempotents. In *Algorithmic problems in groups and semigroups*. Springer, 2000.

5 Andrey Balmin, Yannis Papakonstantinou, and Victor Vianu. Incremental validation of XML documents. *TODS*, 29(4), 2004. URL: `http://db.ucsd.edu/wp-content/uploads/pdfs/212.pdf`.

6 David A. Mix Barrington, Kevin J. Compton, Howard Straubing, and Denis Thérien. Regular languages in NC^1. *J. Comput. Syst. Sci.*, 44(3), 1992. URL: `https://www.sciencedirect.com/science/article/pii/002200009290014A`.

7 Laura Chaubard, Jean-Eric Pin, and Howard Straubing. First order formulas with modular predicates. In *LICS*, 2006. URL: `https://hal.archives-ouvertes.fr/hal-00112846/document`.

8 Sang Cho and Dung T. Huynh. Finite-automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88(1), 1991. `doi:10.1016/0304-3975(91)90075-d`.

9 Raphael Clifford, Allan Grønlund, Kasper Green Larsen, and Tatiana Starikovskaya. Upper and lower bounds for dynamic data structures on strings. In *STACS*, 2018. `doi:10.4230/LIPIcs.STACS.2018.22`.

10 Assis de Azevedo. The join of the pseudovariety **J** with permutative pseudovarieties. In *Lattices, Semigroups, and Universal Algebra*. Springer, 1990.

11 Gudmund Skovbjerg Frandsen, Thore Husfeldt, Peter Bro Miltersen, Theis Rauhe, and Søren Skyum. Dynamic algorithms for the Dyck languages. In *WADS*, 1995. URL: `https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.57.4615&rep=rep1&type=pdf`.

12 Michael Fredman and Michael Saks. The cell probe complexity of dynamic data structures. In *STOC*, 1989. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.453.9085&rep=rep1&type=pdf.

13 Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. In *STACS*, 2018. doi:10.4230/LIPIcs.STACS.2018.31.

14 Moses Ganardi, Danny Hucke, and Markus Lohrey. Querying regular languages over sliding windows. In *FSTTCS*, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.18.

15 Moses Ganardi, Danny Hucke, Markus Lohrey, and Tatiana Starikovskaya. Sliding window property testing for regular languages. In *ISAAC*, 2019. doi:10.4230/LIPIcs.ISAAC.2019.6.

16 Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. The dynamic complexity of formal languages. *TOCL*, 13(3), 2012. arXiv:0812.1915.

17 Kasper Green Larsen, Jonathan Lindegaard Starup, and Jesper Steensgaard. Further unifying the landscape of cell probe lower bounds. In *SOSA*, 2021. URL: https://epubs.siam.org/doi/pdf/10.1137/1.9781611976472.25.

18 Yijie Han and Mikkel Thorup. Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In *FOCS*, 2002.

19 Thore Husfeldt and Theis Rauhe. Hardness results for dynamic problems by extensions of Fredman and Saks' chronogram method. In *ICALP*, 1998. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.6315&rep=rep1&type=pdf.

20 Louis Jachiet. Computing and maintaining the minimum of a set S of integers while allowing updates on S. Theoretical Computer Science Stack Exchange. URL: https://cstheory.stackexchange.com/q/47831 (version: 2020-11-08).

21 Louis Jachiet. On the complexity of a "list" datastructure in the RAM model. Theoretical Computer Science Stack Exchange. URL: https://cstheory.stackexchange.com/q/46746 (version: 2020-05-01).

22 Eugene Kirpichov. Incremental regular expressions. Code available at: https://github.com/jkff/ire, 2012. URL: http://jkff.info/articles/ire/.

23 Mihai Pătraşcu and Corina E Tarniţă. On dynamic bit-probe complexity. *Theoretical Computer Science*, 380(1-2), 2007. URL: https://www.sciencedirect.com/science/article/pii/S0304397507001624.

24 Mihai Pătraşcu and Mikkel Thorup. Randomization does not help searching predecessors. In *SODA*, 2007. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.208.6464&rep=rep1&type=pdf.

25 Jean-Éric Pin. *Varieties of formal languages*. Foundations of Computer Science. Plenum Publishing Corp., New York, 1986. With a preface by M.-P. Schützenberger, Translated from the French by A. Howie. doi:10.1007/978-1-4613-2215-3.

26 Jean-Éric Pin. Mathematical foundations of automata theory, 2019. URL: https://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf.

27 John Rhodes. The fundamental lemma of complexity for arbitrary finite semigroups. *Bulletin of the American Mathematical Society*, 74(6), 1968. doi:10.1090/s0002-9904-1968-12064-6.

28 Jonas Schmidt, Thomas Schwentick, Till Tantau, Nils Vortmeier, and Thomas Zeume. Work-sensitive dynamic complexity of formal languages, 2021. arXiv:2101.08735.

29 Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *JACM*, 44(2), 1997. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.868.4450&rep=rep1&type=pdf.

30 Howard Straubing. The variety generated by finite nilpotent monoids. *Semigroup Forum*, 24(1), 1982. doi:10.1007/bf02572753.

31 Howard Straubing. Finite semigroup varieties of the form V*D. *Journal of Pure and Applied Algebra*, 36, 1985. URL: https://www.sciencedirect.com/science/article/pii/0022404985900623.

32 Robert Endre Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of Computer and System Sciences*, 18(2), 1979. doi:10.1016/0022-0000(79)90042-4.

33 Mikkel Thorup. Equivalence between priority queues and sorting. *JACM*, 54(6), 2007.

34 Peter van Emde Boas, Robert Kaas, and Erik Zijlstra. Design and implementation of an efficient priority queue. *Mathematical systems theory*, 10(1), 1976.

Appendix

Table 1 Summary of the main classes of monoids and semigroups used in the paper.

Class	Description	Equation	References
ZE	Monoids/semigroups with central idempotents	$x^\omega y = yx^\omega$	[1]
ZG	Monoids/semigroups with central groups	$x^{\omega+1}y = yx^{\omega+1}$	[4]
SG	Monoids/semigroups with swappable groups	$x^{\omega+1}yx^\omega = x^\omega yx^{\omega+1}$	[10, 29]
A	Aperiodic semigroups/monoids	$x^{\omega+1} = x^\omega$	[26]
Com	Commutative semigroups/monoids	$xy = yx$	[26]
Nil	Nilpotent semigroups	$x^\omega y = yx^\omega$	[26]
MNil	Monoids dividing a nilpotent semigroup		[30]
D	Definite semigroups	$yx^\omega = x^\omega$	[31]

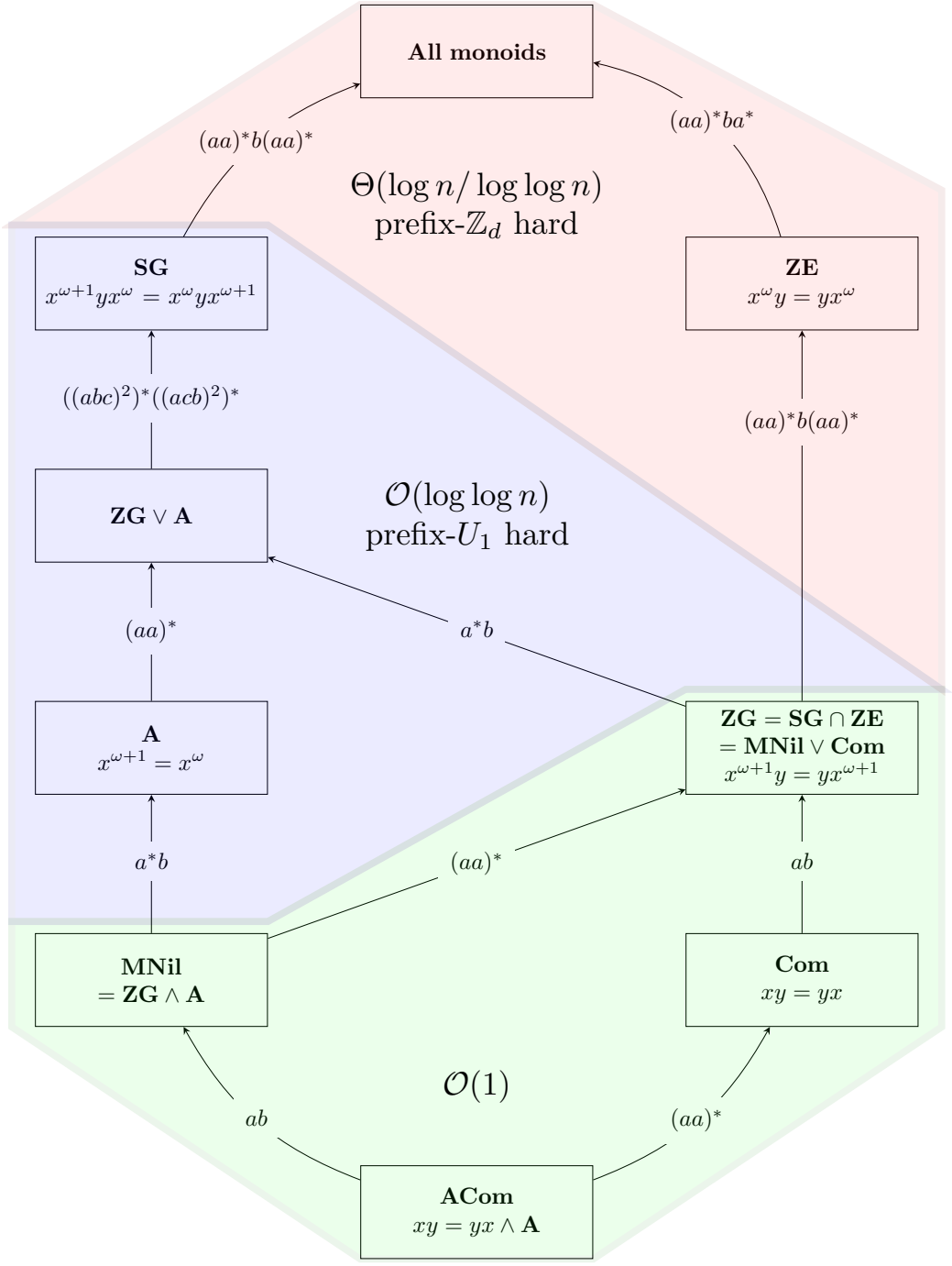

Figure 1 Complexity of the dynamic word problem for common classes of monoids. Arrows denote inclusion and are labeled with languages (with an implicit neutral letter e) whose syntactic monoids separate the classes. The classes **ZG** and **SG** are maximal for the $O(1)$ region and $O(\log\log n)$ region respectively.