Dynamic Membership for Regular Tree Languages

Antoine Amarilli
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, France

Corentin Barloy
Ruhr University Bochum, Germany

Louis Jachiet
Télécom Paris, Institut Polytechnique de Paris, France

Charles Paperman
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, France

Abstract

We study the *dynamic membership problem* for regular tree languages under relabeling updates: we fix an alphabet Σ and a regular tree language L over Σ (expressed, e.g., as a tree automaton), we are given a tree T with labels in Σ, and we must maintain the information of whether the tree T belongs to L while handling relabeling updates that change the labels of individual nodes in T.

Our first contribution is to show that this problem admits an $O(\log n / \log\log n)$ algorithm for any fixed regular tree language, improving over known $O(\log n)$ algorithms. This generalizes the known $O(\log n / \log\log n)$ upper bound over words, and it matches the lower bound of $\Omega(\log n / \log\log n)$ from dynamic membership to some word languages and from the existential marked ancestor problem.

Our second contribution is to introduce a class of regular languages, dubbed *almost-commutative* tree languages, and show that dynamic membership to such languages under relabeling updates can be decided in constant time per update. Almost-commutative languages generalize both commutative languages and finite languages: they are the analogue for trees of the *ZG languages* enjoying constant-time dynamic membership over words. Our main technical contribution is to show that this class is conditionally optimal when we assume that the alphabet features a *neutral letter*, i.e., a letter that has no effect on membership to the language. More precisely, we show that any regular tree language with a neutral letter which is not almost-commutative cannot be maintained in constant time under the assumption that the prefix-U1 problem from [4] also does not admit a constant-time algorithm.

2012 ACM Subject Classification Theory of computation → Regular languages

Keywords and phrases automaton, dynamic membership, incremental maintenance, forest algebra

Digital Object Identifier 10.4230/LIPIcs.MFCS.2025.8

Related Version *Full Version*: `https://arxiv.org/abs/2504.17536` [2]

Funding *Antoine Amarilli*: Partially supported by the ANR project EQUUS ANR-19-CE48-0019, by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 431183758, and by the ANR project ANR-18-CE23-0003-02 ("CQFD").
Corentin Barloy: Partially supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant 532727578.

1 Introduction

Regular tree languages, and the corresponding *tree automata* [11], are a way to enforce structural constraints on trees while generalizing regular languages over words. For any fixed tree language L represented as a tree automaton A, given an input tree T with n nodes, we can verify in $O(n)$ whether T belongs to L, simply by running A over T. However, in some settings, the tree T may be dynamically updated, and we want to efficiently maintain the information of whether T belongs to the language. This problem has been studied in the setting of XML documents and schemas under the name of *incremental schema validation* [6].

© Antoine Amarilli, Corentin Barloy, Louis Jachiet, and Charles Paperman;
licensed under Creative Commons License CC-BY 4.0

In this paper we study the theory of this problem in the RAM model with unit cost and logarithmic word size. We call the problem *dynamic membership* in line with earlier work on regular word languages [4]. We focus on the *computational complexity* of maintaining membership, as opposed, e.g., to *dynamic complexity* [15, 25, 22], which studies the expressive power required (in the spirit of descriptive complexity). We further focus on the measure of *data complexity* [26], i.e., we assume that the regular language is fixed, and measure complexity only as a function of the input tree. We study efficient algorithms for the dynamic membership problem, and (when possible) matching lower bounds.

The naive algorithm for dynamic membership is to re-run A over T after each change, which takes $O(n)$. However, more efficient algorithms are already known. Balmin et al. [6] show that we can maintain membership to any fixed regular tree language in time $O(\log^2 n)$ under leaf insertions, node deletions, and substitutions (also known as node relabelings). We focus in this work on the specific case of relabeling updates, and it is then known that the complexity can be lowered to $O(\log n)$: see, e.g., [3] which relies on tree decomposition balancing techniques [9]. This is slightly less favorable than the $O(\log n / \log\log n)$ complexity known for dynamic membership for regular word languages [12, 4].

In terms of lower bounds, the dynamic membership problem for trees under relabeling updates is known to require time $\Omega(\log n / \log\log n)$, already for some very simple fixed languages. These lower bounds can come from two different routes. One first route is the *existential marked ancestor problem*, where we must maintain a tree under relabeling updates that mark and unmark nodes, and efficiently answer queries asking whether a given node has a marked ancestor. This problem is known to admit an unconditional $\Omega(\log n / \log\log n)$ lower bound in the cell probe model [1], and it turns out that both the updates and the queries can be phrased as relabeling updates for a fixed regular tree language. One second route is via the dynamic membership problem in the more restricted setting of word languages, where there are known $\Omega(\log n / \log\log n)$ lower bounds for some languages [23, 4].

These known results leave a small complexity gap between $O(\log n)$ and $\Omega(\log n / \log\log n)$. Our first contribution in this work is to address this gap, by presenting an algorithm for the dynamic membership problem that achieves complexity $O(\log n / \log\log n)$, for any fixed regular tree language, and after linear-time preprocessing of the input tree. This matches the known lower bounds, and generalizes the known algorithms for dynamic membership to word languages while achieving the same complexity. Our algorithm iteratively processes the tree to merge sets of nodes into clusters and recursively process the tree of clusters; it is very related to the tree contraction technique of Gazit et al. [14] and is reminiscent of other algorithms such as the tree contraction technique of Miller and Reif [20], see also [9]. More precisely, our algorithm regroups tree nodes into clusters which have logarithmic size, ensuring that clusters fit into a memory word. This ensures that the effects of updates on clusters can be handled in constant time by the standard technique of tabulating during the preprocessing. Note that clusters may contain holes, i.e., they amount to contexts, which we handle using the notion of forest algebras. Then, the algorithm considers the induced tree structure over clusters, and processes it recursively, ensuring that the tree size is divided by $\Theta(\log n)$ at each step. The main challenge in the algorithm is to ensure that a suitable clustering of the trees at each level can be computed in linear time.

Having classified the overall complexity of dynamic membership, the next question is to refine the study and show language-dependent bounds. Indeed, there are restricted families of regular tree languages over which we can beat $O(\log n / \log\log n)$. For instance, it is easy to achieve $O(1)$ time per update when the tree language is finite, or when it is commutative (i.e., only depends on the number of label occurrences and not the tree structure). What

is more, in the setting of word languages, the complexity of dynamic membership is in $\Theta(\log n / \log\log n)$ only for a restricted subset of the word languages, namely, those outside the class QSG defined in [4] (see also [12]). Some word languages admit constant-time dynamic membership algorithms: they were classified in [4] conditional to the hypothesis that there is no data structure achieving constant time per operation for a certain problem, dubbed *prefix-U_1* and essentially amounting to maintaining a weak form of priority queue.

Our second contribution is to introduce a class of regular tree languages for which dynamic membership under relabeling updates can be achieved in constant time per update, generalizing commutative tree languages and finite tree languages. Specifically, we define the class of *almost-commutative* tree languages: these are obtained as the Boolean closure of regular tree languages which impose a commutative condition on the so-called *frequent* letters (those with more occurrences than a constant threshold) and imposing that the projection to the other letters (called *rare*) forms a specific constant-sized tree. We show that it is decidable whether a regular tree language is almost-commutative (when it is given, e.g., by a tree automaton). The motivation for almost-commutative languages is that we can show that dynamic membership to such languages can be maintained in $O(1)$ under relabeling updates, generalizing the $O(1)$ algorithm of [4] for ZG monoids.

Our third contribution is to show that, conditionally, the almost-commutative tree languages in fact *characterize* those regular tree languages that enjoy constant-time dynamic membership under relabeling updates, when we assume that the language features a so-called *neutral letter*. Intuitively, a letter e is *neutral* for a language L if it is invisible in the sense that nodes labeled with e can be replaced by the forest of their children without affecting membership to L. Neutral letters are often assumed in the setting of word languages [18, 8], and they make dynamic membership for word languages collapse to the same problem for their syntactic monoid (as was originally studied in [23], and in [4] as a first step).

When focusing on regular tree languages with a neutral letter, we show that for *any* such language which is *not* almost-commutative, dynamic membership cannot be maintained in constant time under relabeling updates, subject to the hypothesis from [4] that the prefix-U_1 problem does not admit a constant-time data structure. Thus, the $O(1)$-maintainable regular tree languages with a neutral letter can be characterized conditionally to the same hypothesis as in the case of words. We show this conditional lower bound via an (unconditional) algebraic characterization of the class of almost-commutative languages with neutral letters: they are precisely those languages with syntactic forest algebras whose vertical monoid satisfies the equation ZG ($x^{\omega+1}y = yx^{\omega+1}$, i.e., group elements are central), which was also the (conditional) characterization for $O(1)$-maintainable word languages with a neutral letter [4]. The main technical challenge to show this result is to establish that every tree language with a ZG vertical monoid must fall in our class of almost-commutative languages: this uses a characterization of ZG tree languages via an equivalence relation analogous to the case of ZG on words [5], but with more challenging proofs because of the tree structure.

Paper structure. We give preliminaries in Section 2 and introduce the dynamic membership problem. We then give in Section 3 some further preliminaries about *forest algebras*, which are instrumental to the proof of our results. We then present our first contribution in Section 4, namely, an $O(\log n / \log\log n)$ algorithm for dynamic membership to any fixed regular tree language. We then introduce almost-commutative languages in Section 5 and show that dynamic membership to these languages is in $O(1)$. In Section 6 we show our matching lower bound: in the presence of a neutral letter, and assuming that prefix-U_1 cannot be solved in constant time, then the dynamic membership problem cannot be solved in constant time for any non-almost-commutative regular language. We conclude in Section 7.

Because of space limitations, detailed proofs are deferred to the full version [2]. Note that a version of the results of Sections 5 and 6 was presented in [7] but never formally published.

2 Preliminaries

Forests, trees, contexts. We consider ordered forests of rooted, unranked, ordered trees on a finite alphabet Σ, and simply call them *forests*. Formally, we define Σ-*forests* and Σ-*trees* by mutual induction: a Σ-forest is a (possibly empty) ordered sequence of Σ-trees, and a Σ-tree consists of a *root node* with a label in Σ and with a Σ-forest of child nodes.

We assume familiarity with standard terminology on trees including parent node, children node, ancestors, descendants, root nodes, siblings, sibling sets, previous sibling, next sibling, prefix order, etc.; see [11] for details. As we work with ordered forests, we will always consider root nodes to be siblings, and define the previous sibling and next sibling relationships to also apply to roots. We will often abuse notation and identify forests with their sets of nodes, e.g., for a forest F, we define functions on F to mean functions defined on the nodes of F. We say that two forests have the *same shape* if they are identical up to changing the labels of their nodes (which are elements of Σ). The *size* $|F|$ of a Σ-forest F is its number of nodes, and we write $|F|_a$ for $a \in \Sigma$ to denote the number of occurrences of a in F.

Forest languages. A *forest language* L over an alphabet Σ is a subset of Σ-forests, and a *tree language* is a subset of Σ-trees: throughout the paper we study forest languages, but of course our results extend to tree languages as well. We write $\overline{L}$ for the complement of L. We will be specifically interested in *regular forest languages*: among many other equivalent characterizations [11], these can be formalized via finite automata, which we formally introduce below (first for words and then for forests).

A *deterministic word automaton* over a set S is a tuple $A = (Q, F, q_0, \delta)$ where Q is a finite set of *states*, $F \subseteq Q$ is a subset of *final states*, $q_0 \in Q$ is the *initial state*, and $\delta \colon Q \times S \to Q$ is a transition function. For $w = s_1 \cdots s_k$ a word over S, the *state reached by A when reading w* is defined inductively as usual: if $w = \epsilon$ is the empty word then the state reached is q_0, otherwise the state reached is $\delta(q, s_k)$ with q the state reached when reading $s_1 \cdots s_{k-1}$. The word w is *accepted* if the state reached by A when reading w is in F.

We define *forest automata* over forests, which recognize forest languages by running a word automaton over the sibling sets. These automata are analogous to the *hedge automata* of [11] (where horizontal transitions are specified by regular languages), and to the forest automata of [10] (where horizontal transitions are specified by a monoid). Formally, a *forest automaton* over Σ is a tuple $A = (Q, A', \delta)$ where Q is a finite set of states, $A' = (Q', F', q'_0, \delta')$ is a word automaton over the set Q, and $\delta \colon Q' \times \Sigma \to Q$ is a *vertical transition function* which associates to every state $q' \in Q'$ of A' and letter $a \in \Sigma$ a state $\delta(q', a) \in Q$. The *run* of $A = (Q, A', \delta)$ on a Σ-tree T is a function ρ from T to Q defined inductively as follows:

- For u a leaf of T with label $a \in \Sigma$, we have $\rho(u) = \delta(q'_0, a)$ for q'_0 the initial state of A';
- For u an internal node of T with label $a \in \Sigma$, letting $u_1, \ldots, u_k$ be its successive children, and letting $q_i := \rho(u_i)$ for all $1 \leq i \leq k$ be the inductively computed images of the run, for q the state reached in A' when reading the word $q_1 \cdots q_k$, we let $\rho(u) := \delta(q, a)$.

We say that a Σ-forest F is *accepted* by the forest automaton A if, letting $T_1, \ldots, T_k$ be the trees of F in order, and letting $q_1, \ldots, q_k$ be the states to which the respective roots of the $T_1, \ldots, T_k$ are mapped by the respective runs of A, then the word $q_1 \cdots q_k$ is accepted by the word automaton A'. (In particular, the empty forest is accepted iff $q'_0 \in F'$.) The *language* of A is the set of Σ-forests that it accepts.

Relabelings, incremental maintenance. We consider *relabeling updates* on forests F, that change node labels without changing the tree shape. A *relabeling update* gives a node u of F (e.g., as a pointer to that node) and a label $a \in \Sigma$, and changes the label of the node u to a.

We fix a regular language L of Σ-forests given by a fixed forest automaton A, and we are given as input a Σ-forest F. We can compute in linear time whether A accepts F. The *dynamic membership problem* for L is the task of maintaining whether the current forest belongs to L, under relabeling updates. We study the complexity of this problem, defined as the worst-case running time of an algorithm after each update, expressed as a function of the size $|F|$ of F. Note that the language L is assumed to be fixed, and is not accounted for in the complexity. We work in the RAM model with unit cost and logarithmic word size, and consider dynamic membership algorithms that run a linear-time *preprocessing* on the input forest F to build data structures used when processing updates. (By contrast, our lower bound results will hold without the assumption that the preprocessing is in linear time.)

As we reviewed in the introduction, for any fixed regular forest language, it is folklore that the dynamic membership problem on a tree with n nodes can be solved under relabeling updates in time $O(\log n)$ per update: see for instance [3]. Further, for some forest languages, some unconditional lower bounds in $\Omega(\log n / \log\log n)$ are known. This includes some forest languages defined from intractable word languages, for instance the forest language L_1 on alphabet $\Sigma = \{0, 1, \#\}$ where the word on Σ^* formed by the leaves in prefix order is required to fall in the word language $L_2 := (0^*10^*10^*)^*\#\Sigma^*$, i.e., a unique # with an even number of 1's before it. Indeed, dynamic membership to L_2 (under letter substitutions) admits an $\Omega(\log n / \log\log n)$ lower bound from the *prefix-*$\mathbb{Z}_2$ problem (see [13, Theorem 3], and [12, 4]), hence so does L_1. Intractable forest languages also include languages corresponding to the *existential marked ancestor* problem [1], e.g., the language L_3 on alphabet $\Sigma = \{e, m, \#\}$ where we have a unique node u labeled # and we require that u has an ancestor labeled m. Indeed, the existential marked ancestor problem of [1] allows us to mark and unmark nodes over a fixed tree (corresponding to letters e for unmarked nodes and m for marked nodes), and allows us to query whether a node (labeled #) has a marked ancestor. Thus, the existential marked ancestor problem immediately reduces to dynamic membership for L_3, which inherits the $\Omega(\log n / \log\log n)$ lower bound from [1].

3 Forest Algebras

All our results about the dynamic membership problem are in fact shown by rephrasing to the terminology of *forest algebras*. Intuitively, forest algebras give an analogue for trees to the monoids and syntactic morphisms used in the setting of words, which formed the first step of the classification of the complexity of dynamic membership on words [12, 4]. More details on forest algebras are given in the appendix of the full version [2].

Monoids. A *monoid* M is a set equipped with an associative composition law featuring a *neutral element*, that is, an element e such that $ex = xe = x$ for all x in M. We write the composition law of monoids multiplicatively, i.e., we write xy for the composition of x and y. The *idempotent power* of an element v in a finite monoid M, written v^ω, is v raised to the least integer ω such that $v^\omega = v^\omega v^\omega$: see [21, Chp. II, Prop. 6.31] for a proof that every element has an idempotent power. The *idempotent power* of M is a value $m \in \mathbb{N}$ ensuring that, for each $v \in M$, we have $v^m v^m = v^m$: this can be achieved by taking any common multiple of the exponents ω for the idempotent powers of all elements of M.

Forest algebra. A forest algebra [10] is a pair (V, H) of two monoids. The monoid V is the *vertical monoid*, with vertical composition written $\odot_{\mathrm{VV}}$ and neutral element written $\Box$. The monoid H is the *horizontal monoid*, with horizontal concatenation denoted $\oplus_{\mathrm{HH}}$, and with neutral element written ϵ. Further, we have three composition laws: $\odot_{\mathrm{VH}} \colon V \times H \to H$, $\oplus_{\mathrm{VH}} \colon V \times H \to V$, and $\oplus_{\mathrm{HV}} \colon H \times V \to V$. We require that the following relations hold:

- *Action (composition)*: for every $v, w \in V$ and $h \in H$, $(v \odot_{\mathrm{VV}} w) \odot_{\mathrm{VH}} h = v \odot_{\mathrm{VH}} (w \odot_{\mathrm{VH}} h)$.
- *Action (neutral)*: for every $h \in H$, $\Box \odot_{\mathrm{VH}} h = h$.
- *Mixing*: for every $v \in V$ and $h, g \in H$, we have $(v \oplus_{\mathrm{VH}} h) \odot_{\mathrm{VH}} g = (v \odot_{\mathrm{VH}} g) \oplus_{\mathrm{HH}} h$ and $(h \oplus_{\mathrm{HV}} v) \odot_{\mathrm{VH}} g = h \oplus_{\mathrm{HH}} (v \odot_{\mathrm{VH}} g)$.
- *Faithfulness*: for every distinct $v, w \in V$, there exists $h \in H$ such that $v \odot_{\mathrm{VH}} h \neq w \odot_{\mathrm{VH}} h$.

We explain in the appendix of the full version [2] how this formalism slightly differs from that of [10] but is in fact equivalent. We often abuse notation and write $\oplus$ to mean one of $\oplus_{\mathrm{HH}}, \oplus_{\mathrm{HV}}, \oplus_{\mathrm{VH}}$ and write $\odot$ to mean one of $\odot_{\mathrm{VH}}, \odot_{\mathrm{VV}}$.

We almost always assume that forest algebras are *finite*, i.e., the horizontal and vertical monoids H and V are both finite. The only exception is the *free forest algebra* $\Sigma^{\nabla} = (\Sigma^{\mathrm{V}}, \Sigma^{\mathrm{H}})$. Here, Σ^{H} is the set of all Σ-forests, and Σ^{V} is the set of all Σ-*contexts*, i.e., Σ-forests having precisely one node that carries the special label $\Box \notin \Sigma$: further, this node must be a leaf. The $\oplus$-laws denote horizontal *concatenation*: of two forests for $\oplus_{\mathrm{HH}}$, and of a forest and a context for $\oplus_{\mathrm{HV}}$ and $\oplus_{\mathrm{VH}}$. (Note that two contexts cannot be horizontally concatenated because the result would have two nodes labeled $\Box$.) We write the $\oplus$-laws as $+$, and remark that they are not commutative. The $\odot$-laws are the *context application operations* of plugging the forest (for $\odot_{\mathrm{VH}}$) or context (for $\odot_{\mathrm{VV}}$) on the right in place of the $\Box$-node of the context on the left. We write the $\odot$-laws functionally, i.e., for $s \in \Sigma^{\mathrm{V}}$ and for $u \in \Sigma^{\mathrm{V}}$ or $f \in \Sigma^{\mathrm{H}}$ we write $s(u)$ to mean $s \odot_{\mathrm{VV}} u$ and write $s(f)$ to mean $s \odot_{\mathrm{VH}} f$.

We write $\Box \in \Sigma^{\mathrm{V}}$ for the trivial Σ-context consisting only of a single node labeled $\Box$, we write $\epsilon \in \Sigma^{\mathrm{H}}$ for the empty forest, and for $a \in \Sigma$ we write $a_{\Box} \in \Sigma^{\mathrm{V}}$ the Σ-context which consists of a single root node labeled a with a single child labeled $\Box$. Note that Σ^{V} and Σ^{H} are spanned by ϵ and $\Box$ and by the $a_{\Box}$ via context application and concatenation.

Morphisms. A *morphism of forest algebras* [10] consists of two functions that map forests to forests and contexts to contexts and are compatible with the internal operations. Formally, a morphism from (V, H) to (V', H') is a pair of functions $\mu_{\mathrm{V}} \colon V \to V', \mu_{\mathrm{H}} \colon H \to H'$ where:

- μ_{V} is a monoid morphism: for all $v, w \in V$, we have $\mu_{\mathrm{V}}(v \cdot w) = \mu_{\mathrm{V}}(v) \cdot \mu_{\mathrm{V}}(w)$ and $\mu_{\mathrm{V}}(\Box) = \Box$ where $\cdot$ and $\Box$ are interpreted in the corresponding monoid.
- μ_{H} is a monoid morphism: for all $h, g \in H$, we have $\mu_{\mathrm{H}}(h + g) = \mu_{\mathrm{H}}(h) + \mu_{\mathrm{H}}(g)$ and $\mu_{\mathrm{H}}(\epsilon) = \epsilon$ where $+$ and ϵ are interpreted in the corresponding monoid.
- for every $v \in V$, $h \in H$, we have $\mu_{\mathrm{H}}(v \odot_{\mathrm{VH}} h) = \mu_{\mathrm{V}}(v) \odot_{\mathrm{VH}} \mu_{\mathrm{H}}(h)$.
- for every $v \in V$, $h \in H$, we have $\mu_{\mathrm{V}}(v \oplus_{\mathrm{VH}} h) = \mu_{\mathrm{V}}(v) \oplus_{\mathrm{VH}} \mu_{\mathrm{H}}(h)$ and $\mu_{\mathrm{V}}(h \oplus_{\mathrm{HV}} v) = \mu_{\mathrm{H}}(h) \oplus_{\mathrm{HV}} \mu_{\mathrm{V}}(v)$.

We often abuse notation and write a morphism μ to mean the function that applies μ_{V} to Σ-contexts and μ_{H} to Σ-forests. For any alphabet Σ and forest algebra (V, H), given a function g from the alphabet Σ to V, we extend it to a morphism $\mu_{\mathrm{V}}, \mu_{\mathrm{H}}$ from the free forest algebra to (V, H) in the only way compatible with the requirements above, i.e.:

- For a sequence of Σ-trees $F = t_1, \ldots, t_k$ where at most one t_i is a Σ-context, letting $x_i \coloneqq \mu(t_i)$ for each i be inductively defined, then $\mu(F) \coloneqq x_1 \oplus \cdots \oplus x_k$.
- For a Σ-tree or Σ-context t with root node u labeled $a \in \Sigma$, letting F be the (possibly empty) sequence of children of u (of which at most one is a context, and for which $\mu(F)$ was inductively defined), then we set $\mu(t) \coloneqq (g(a)) \odot \mu(F)$.

A forest language L over Σ is *recognized* by a forest algebra (V, H) if there is a subset $H' \subseteq H$ and a function from Σ to V defining a morphism $\mu_{\mathrm{V}}, \mu_{\mathrm{H}}$ having the following property: a Σ-forest F is in L iff $\mu_{\mathrm{H}}(F) \in H'$.

Syntactic forest algebra. Regular forest languages can be related to forest algebras via the notion of *syntactic forest algebra*. Indeed, a forest language is regular iff it is recognized by some forest algebra (see [10, Proposition 3.19]). Specifically, we will consider the *syntactic forest algebra* of a regular forest language L: this forest algebra recognizes L, it is minimal in a certain sense, and it is unique up to isomorphism. We omit the formal definition of the syntactic forest algebra (V, H) of L (see [10, Definition 3.13] for details). We will just use the fact that it recognizes L for a certain function g from Σ to V and associated morphism $\mu_{\mathrm{V}}, \mu_{\mathrm{H}}$ from Σ-forests to (V, H), called the *syntactic morphism*, and satisfying the following:

- *Surjectivity:* for any element v of V, there is a Σ-context c such that $\mu_{\mathrm{V}}(c) = v$;
- *Minimality:* for any two Σ-contexts c and c', if $\mu_{\mathrm{V}}(c) \neq \mu_{\mathrm{V}}(c')$, then there is a Σ-context r and a Σ-forest s such that exactly one of $r(c(s))$ and $r(c'(s))$ belongs to L.

Dynamic evaluation problem for forest algebras. We will study the analogue of the dynamic membership problem for forest algebras, which is that of computing the *evaluation* of an expression. More precisely, for (V, H) a forest algebra, a (V, H)*-forest* is a forest where each internal node is labeled by an element of V and where each leaf is labeled with an element of H – but there may be one leaf, called the *distinguished leaf*, which is labeled with an element of V. The *evaluation* of a (V, H)-forest F is the image of F by the morphism obtained by extending the function g which maps elements of V to themselves and maps elements f of H to the context $c_f \coloneqq f \oplus_{\mathrm{HV}} \square$ (so that $c_f \odot_{\mathrm{VH}} \epsilon = f$). Remark that the forest F evaluates to an element of V if it has a distinguished leaf, and to H otherwise.

The *(non-restricted) dynamic evaluation problem* for (V, H) then asks us to maintain the evaluation of the (V, H)-forest F under relabeling updates which can change the label of internal nodes of F (in V) and of leaf nodes of F (in H or in V, but ensuring that there is always at most one distinguished leaf in F). Again, we assume that the forest algebra (V, H) is constant, and we measure the complexity as a function of the size $|F|$ of the input forest (note that updates never change $|F|$ or the shape of F). The *restricted dynamic evaluation problem* for (V, H) adds the restriction that the label of leaves always stays in H (initially and after all updates), so that F always evaluates to an element of H.

We will use the dynamic evaluation problem in the next section for our $O(\log n / \log \log n)$ upper bound. Indeed, it generalizes the dynamic membership problem in the following sense:

▶ **Lemma 3.1.** *Let L be a fixed regular forest language, and let (V, H) be its syntactic forest algebra. Given an algorithm for the restricted dynamic evaluation problem for (V, H) under relabeling updates, we can obtain an algorithm for the dynamic membership problem for L with the same complexity per update.*

4 Dynamic Membership to Regular Languages in $O(\log n / \log \log n)$

Having defined the technical prerequisites, we now start the presentation of our technical results. In this section, we show our general upper bound on the dynamic membership problem to arbitrary regular forest languages:

▶ **Theorem 4.1.** *For any fixed regular forest language L, the dynamic membership problem to L is in $O(\log n / \log\log n)$, where n is the number of nodes of the input forest.*

Note that this upper bound matches the lower bounds reviewed at the end of Section 2. We present the proof in the rest of this section. By Lemma 3.1, we will instead show our upper bound on the restricted dynamic evaluation problem for arbitrary fixed forest algebras.

The algorithm that shows Theorem 4.1 intuitively follows a recursive scheme. For the first step of the scheme, given the input forest F_0, we compute a so-called *clustering* of F_0. This is a partition of the nodes of F_0 into subsets, called *clusters*, which are connected in a certain sense and will be chosen to have size $O(\log n)$. Intuitively, clusters are small enough so that we maintain their evaluation under updates in $O(1)$ by tabulation; note that clusters may correspond to contexts (i.e., they may have holes), so we will perform *non-restricted* dynamic evaluation for them. Further, F_0 induces a forest structure on the clusters, called the *forest of clusters* and denoted F_1, for which we will ensure that it has size $O(n/\log n)$. We note that the clustering algorithm that we propose is very similar to the one constructed in the work of Gazit et al. [14], which has similar properties and could be used as an alternative to the one we propose. However, our clustering is not constructed in the same way and the result of building it on an input tree is not identical to the clustering of [14], so we present our own construction to ensure that our work is self-contained.

Having computed the clustering of F_0 and the forest of clusters F_1, we re-apply recursively the clustering scheme on F_1, decomposing it again into clusters of size $O(\log n)$ and a forest of clusters F_2 of size $O(n/(\log n)^2)$. We recurse until we obtain a forest F_ℓ with only one node, which is the base case: we will ensure that ℓ is in $O(\log n / \log\log n)$.

To handle updates on a node u of F_0, we will apply the update on the cluster C of F_0 containing u (in $O(1)$ by tabulation), and apply the resulting update on the node C in the forest of clusters F_1. We then continue this process recursively, eventually reaching the singleton F_ℓ where the update is trivial. The main technical challenge is to bound the complexity of the preprocessing: we must show how to efficiently compute a suitable clustering on an input forest F in time $O(|F|)$. It will then be possible to apply the algorithm to $F_0, F_1, \ldots, F_{\ell-1}$, with a total complexity amounting to $O(|F_0|)$.

The section is structured as follows. First, we formally define the notion of *clusters* and *clustering* of a forest F, and we define the *forest of clusters* induced by a clustering: note that these notions only depend on the shape of F and not on the node labels. Second, we explain how the evaluation of a (V, H)-forest reduces to computing the *evaluation* of clusters along with the *evaluation* of the forest of clusters. Third, we explain how we can compute in linear time a clustering of the input forest which ensures that the forest of clusters is small enough: we show that it is sufficient to compute any *saturated* clustering (i.e., no clusters can be *merged*), and sketch an algorithm that achieves this. Fourth, we conclude the proof of Theorem 4.1 by summarizing how the recursive scheme works, including how the linear-time preprocessing can tabulate the effect of updates on small forests.

Clusters and clusterings. A *clustering* of a forest will be defined by partitioning its vertices into *connected* sets, where connectedness is defined using the sibling and first-child edges.

▶ **Definition 4.2.** *Let F be a forest. We say that two nodes of F are* LCRS-adjacent *(for* left-child-right-sibling*) if one node is the first child of the other or if they are two consecutive siblings (in particular if they are two consecutive roots). We say that a set of nodes of F is* LCRS-connected *if, for any two nodes u, u' in F, there is a sequence $u = u_1, \ldots, u_q = u'$ of nodes in F such that u_i and u_{i+1} are LCRS-adjacent for each $1 \leq i < q$.*

Note that the edges used in LCRS-adjacency are *not* the edges of F, but those of a left-child-right-sibling representation of F (hence the name). For instance, the set $\{u, u'\} \subseteq F$ formed of a node u and its parent u' is *not* connected unless u is the first child of u'. To define clusters and clusterings, we will use LCRS-adjacency, together with a notion of *border nodes* that correspond to the "holes" of clusters:

▶ **Definition 4.3.** *Given a forest F with n nodes, we say that a node u in a subset S of F is a* border node *of S if u has a child which is not in S. For $k > 0$, we then say that an equivalence relation $\equiv$ over the nodes of F is a* k-clustering *when the following properties hold on the equivalence classes of $\equiv$, called* clusters*:*

- *each cluster contains at most k nodes;*
- *each cluster is LCRS-connected;*
- *each cluster contains at most one border node.*

The roots *of S are the nodes of S whose parent is not in S (or which are roots of F): if S is a cluster, then by LCRS-connectedness its roots must be consecutive siblings in F.*

When we have defined a k-clustering, it induces a *forest of clusters* in the following way:

▶ **Definition 4.4.** *Given a k-clustering $\equiv$ of a forest F, the* forest of clusters *$F^{\equiv}$ is a forest whose nodes are the clusters of $\equiv$, and where a cluster C_1 is the child of a cluster C_2 when the roots of C_1 are children of the border node of C_2.*

We order the children of a cluster C in $F^{\equiv}$ in the following way. For each child C' of C, its root nodes are a set of consecutive siblings, and these roots are in fact consecutive children of the border node u of C. Thus, given two children C_1 and C_2 of C in $F^{\equiv}$, we order C_1 before C_2 if the roots of C_1 come before the roots of C_2 in the order in F on the children of u. Likewise, we can order the roots of $F^{\equiv}$, called the root clusters*, according to the order on the roots of F: recall that, by our definition of siblings, the root clusters are also siblings in $F^{\equiv}$.*

Remark that the trivial equivalence relation (putting each node in its own cluster) is vacuously a k-clustering in which the border nodes are precisely the internal nodes: we call this the *trivial k-clustering*, and its forest of clusters is isomorphic to F.

Evaluation of clusters. To solve the dynamic evaluation problem on F using a clustering $\equiv$, we will now explain how the evaluation of the (V, H)-forest F can reduce to the evaluation of the forest of clusters $F^{\equiv}$ with a suitable labeling. To define this labeling, let us first define the *evaluation* of a cluster in F:

▶ **Definition 4.5.** *Given a (V, H)-forest F with no distinguished leaf, a k-clustering $\equiv$ of F, and a cluster C of $\equiv$, we define the* evaluation of C *as a value in V or H in the following way. Let F^C be the sub-forest of F induced by C, i.e., the sub-forest containing only the nodes in C and the edges connecting two nodes that are both in C: note that it is a (V, H)-forest where each node has the same label as in F. When C contains a border node u, we also add a leaf u' as the last child of u in F^C and label u' with $\square \in V$.*

The evaluation *of the cluster C in F is then the evaluation of F^C as a (V, H)-forest. Note that, as F has no distinguished leaf, the evaluation is in V if C has a border node and in H otherwise; in other words it is in V exactly when C has a child in $F^{\equiv}$.*

We now see $F^{\equiv}$ as a (V, H)-forest, with each cluster labeled by its evaluation in F: Then:

▷ Claim 4.6. For any k-clustering $\equiv$ of a (V, H)-forest F, the evaluation of F is the same as the evaluation of its forest of clusters $F^{\equiv}$.

This property is what we use in the recursive scheme: to solve the dynamic evaluation problem on the input (V, H)-forest F, during the preprocessing we will compute a clustering $\equiv$ of F and compute the evaluation of clusters and of the forest of clusters $F^{\equiv}$. Given a relabeling update on F, we will apply it to its cluster C and recompute the evaluation of C; this gives us a relabeling update to apply to the node C on the forest of clusters $F^{\equiv}$, and we can recursively apply the scheme to $F^{\equiv}$ to maintain the evaluation of $F^{\equiv}$. What remains is to explain how we can efficiently compute a clustering of a forest F that ensures that the forest of clusters $F^{\equiv}$ is "small enough".

Efficient computation of a clustering. Here is what we want to establish:

▶ **Proposition 4.7.** *There is a fixed constant $c \in \mathbb{N}$ such that the following is true: given a forest F, we can compute a k-clustering $\equiv$ of F in linear time such that $|F^{\equiv}| \leq \lceil |F| \times c/k \rceil$.*

Our algorithm starts with the trivial k-clustering and iteratively merges clusters. Formally, *merging* two clusters C and C' means setting them to be equivalent under $\equiv$, and we call C and C' *mergeable* when doing so leads to a k-clustering. Of course, we will only merge mergeable clusters in the algorithm.

We say that the k-clustering is *saturated* if it does not contain two mergeable clusters. Our definition of clusters is designed to ensure that, as soon as the clustering is saturated, there are enough large clusters so that the number of clusters (i.e., the size of the forest of clusters) satisfies the bound of Proposition 4.7, no matter how clusters were merged. Namely:

▷ Claim 4.8. There is a fixed constant $c \in \mathbb{N}$ such that the following is true: given a (V, H)-forest F, any saturated k-clustering on F has at most $\lceil c \times (n/k) \rceil$ clusters.

Proof sketch. We focus on clusters with zero or one child in the forest of clusters. We consider potential merges of these clusters with their only child, with their preceding sibling, or with their parent (if they are the first child). This allows us to show that, provided that there are multiple clusters, a constant fraction of them have to contain at least $k/2$ nodes, so the number of clusters is $O(n/k)$. ◀

Thus, to show Proposition 4.7, it suffices to compute a saturated k-clustering. Namely:

▷ Claim 4.9. Given a forest, we can compute a saturated k-clustering along with its forest of clusters in linear time.

Proof sketch. We start with the trivial k-clustering, and then we saturate the clustering by exploring nodes following a standard depth-first traversal: from a node u, we proceed recursively on the first child u' of u (if it exists) and then on the next sibling u'' of u (if it exists). Then, we first try to merge the cluster of u with the cluster of the first child u', and then try to merge the cluster of u with the cluster of the next sibling u''. By "try to merge", we mean that we merge the two clusters if they are mergeable.

This algorithm clearly runs in linear time as long as the mergeability tests and the actual merges are performed in constant time, which we can ensure with the right data structures. The algorithm clearly returns a clustering, and the complicated part is to show that it is saturated. For this, assume by contradiction that the result contains two mergeable clusters with LCRS-adjacent nodes u_1 and u_2: without loss of generality u_2 is the first child or next sibling of u_1. When processing u_1, the algorithm tried to merge the cluster that u_1 was in with that of u_2, and we know that this attempt failed because we see that u_1 and u_2 are in different clusters at the end of the algorithm. This yields a contradiction with the fact that u_1 and u_2 end up in mergeable clusters at the end of the algorithm, given the order in which the algorithm considers possible merges. ◀

Description of the main algorithm. We are now ready to describe our algorithm for the dynamic evaluation problem for a fixed forest algebra (V, H). Given as input a (V, H)-forest F with n nodes (without a distinguished node), we want to maintain the evaluation of F (which is an element in H). The first step of the preprocessing is to compute in linear time an auxiliary data structure $\mathcal{S}_k$ which is used to maintain the evaluation of small forests under updates in $O(1)$, simply by tabulation. More precisely, $\mathcal{S}_k$ can be used to solve the *non-restricted* dynamic evaluation problem for forests of size at most $k+1$, as follows:

▶ **Proposition 4.10.** *Given a forest algebra (V, H) there is a constant $c_{V,H} \in \mathbb{N}$ such that the following is true. Given $k \in \mathbb{N}$, we can compute in $O(2^{k \times c_{V,H}})$ a data structure $\mathcal{S}_k$ that stores a sequence (initially empty) of (V, H)-forests $G_1, \ldots, G_q$ and supports the following:*

- *add(G): given an (V, H)-forest G with at most $k+1$ nodes, insert it into the sequence, taking time and space $O(|G|)$*
- *relabel(i, n, σ): given an integer $1 \leq i \leq q$, a node u, and a label $\sigma \in H$ or $\sigma \in V$, relabel the node u of G_i to σ, taking time $O(1)$ – as usual we require that internal nodes have labels in H and at most one leaf has label in V*
- *eval(i): given $1 \leq i \leq q$, return the evaluation of G_i, taking $O(1)$*

Letting $k := \lfloor \log n / c_{(V,H)} \rfloor$, the first step of the preprocessing builds the data structure $\mathcal{S}_k$ in time $O(n)$. We then move on to the second step, which is to compute a sequence of forests by recursively clustering in the following way. We start by letting $F_0 := F$ be the input Σ-forest. Then, we recursively do the following. If $|F_i| = 1$ the sequence stops at $\ell = i$. Otherwise, we compute a saturated k-clustering $\equiv_i$ of F_i using Claim 4.9, and we let F_{i+1} be the forest of clusters $F_i^{\equiv_i}$. We will later show that this takes time $O(n)$ overall.

We continue with a third preprocessing step that computes the evaluation of each cluster at each level: again we will later argue that this takes $O(n)$. More precisely, we consider all the clusters C of F_0 and add the sub-forest F_0^C of F_0 induced by each C to $\mathcal{S}_k$, obtaining their evaluation in time $O(|F_0|)$. We use the result of the evaluation as labels for the corresponding nodes in F_1. Then we add the sub-forests induced by all the clusters of F_1 to $\mathcal{S}_k$ to obtain their evaluation and to label F_2. We continue until we have the evaluation of F_ℓ. Note that none of the (V, H)-forests F_i has a distinguished leaf: indeed, the only place where we perform non-restricted dynamic evaluation is in Proposition 4.10, i.e., on the sub-forests induced by the clusters and added to $\mathcal{S}_k$. Further note that all these induced sub-forests have at most $k+1$ nodes by definition of a k-clustering (the $+1$ comes from the $\square$-labeled leaf which may be added for the border node in Definition 4.5). Now, by Claim 4.6 at each step, we have that the evaluation of F_ℓ is equal to the evaluation of the input (V, H)-forest F_0. This is the answer we need to maintain, and this concludes the preprocessing.

Let us now explain how we recursively handle relabeling updates. To apply an update to the node u of the input forest F_0, we retrieve its cluster C and use $\mathcal{S}_k$ to apply this update to u to the induced sub-forest F_0^C and retrieve its new evaluation, in $O(1)$. This gives us an update to apply to the node C of the forest of clusters F_1. We continue like this over the successive levels, until we have an update applied to the single node of F_ℓ, which again by Claim 4.6 is the desired answer. The update is handled overall in time $O(\ell)$, so let us bound the number ℓ of recursion steps. At every level $i < \ell$ we have $|F_{i+1}| \leq \lceil |F_i| \times (c_{V,H}/k) \rceil$ by Claim 4.8 and $|F_i| \geq k$, so we have $|F_i| \leq n \times ((c_{V,H}+1)/k))^i$ and therefore $\ell = O(\log n / \log k) = O(\log n / \log \log n)$ given that $k = \lfloor \log n / c_{V,H} \rfloor$.

The only remaining point is to bound the total time needed in the preprocessing. The data structure $\mathcal{S}_k$ is computed in linear time in the first step. Then, we spend linear time in each F_i to compute the k-clusterings at each level in the second step, and we again spend

linear time in each F_i to feed the induced sub-forests of all the clusters of F_i to $\mathcal{S}_k$ in the third step. Now, it suffices to observe that the size of each F_i decreases exponentially with i; for sufficiently large n and k we have $k \geq 2$ so $|F_i| \leq |F_0|/2^i$ and the total size of the F_i is $O(|F_0|)$. This ensures that the total time taken by the preprocessing across all levels in the second and third steps is in $O(n)$, which concludes.

5 Dynamic Membership to Almost-Commutative Languages in $O(1)$

We have shown in Section 4 our general upper bound (Theorem 4.1) on the dynamic membership problem to arbitrary forest languages. In this section, we study how we can show more favorable bounds on the complexity on dynamic membership when focusing on restricted families of languages. More precisely, in this section, we define a class of regular forest languages, called *almost-commutative* languages, and show that the dynamic membership problem to such languages can be solved in $O(1)$. We will continue studying these languages in the next section to show that non-almost-commutative languages conditionally cannot be maintained in constant time when assuming the presence of a neutral letter.

Defining almost-commutative languages. To define almost-commutative languages, we need to define the subclasses of *virtually-singleton* languages and *regular-commutative* languages. Let us first define virtually-singleton languages via the operation of *projection*:

▶ **Definition 5.1.** *The* removal *of a node u in a Σ-forest F means replacing u by the (possibly empty) sequence of its children.* Removing *a subset of nodes of F is then defined in the expected way; note that the result does not depend on the order in which nodes are removed.*

For $\Sigma' \subseteq \Sigma$ a subalphabet, given a Σ-forest F, the projection *of F over Σ' is the forest $\pi_{\Sigma'}(F)$ obtained from F when removing all nodes that are labeled by a letter of $\Sigma \setminus \Sigma'$.*

A forest language L over Σ is virtually-singleton *if there exists a subalphabet $\Sigma' \subseteq \Sigma$ and a Σ'-forest F' such that L is the set of forests whose projection over Σ' is F'.*

Note that virtually-singleton languages are always regular: a forest automaton can read an input forest, ignoring nodes with labels in $\Sigma \setminus \Sigma'$, and check that the resulting forest is exactly the fixed target forest F'.

Let us now define *regular-commutative* languages as the *commutative* regular forest languages, i.e., membership to the language can be determined from the *Parikh image*:

▶ **Definition 5.2.** *The* Parikh image *of a Σ-forest F is the vector $v \in \mathbb{N}^\Sigma$ such that for every letter $a \in \Sigma$, the component v_a is the number of nodes labelled by a in F.*

A forest language L is regular-commutative *if it is regular and there is a set $S \subseteq \mathbb{N}^\Sigma$ such that L is the set of forests whose Parikh image is in S.*

We can now define *almost-commutative* forest languages from these two classes:

▶ **Definition 5.3.** *A forest language L is* almost-commutative *if it is a finite Boolean combination of regular-commutative and virtually-singleton languages.*

Note that almost-commutative languages are always regular, because regular forest languages are closed under Boolean operations. Further, in a certain sense, almost-commutative languages generalize all word languages with a neutral letter that enjoy constant-time dynamic membership. Indeed, such languages are known by [4] and [5, Corollary 3.5] to be described by regular-commutative conditions and the presence of specific subwords on some subalphabets. Thus, letting L be such a word language, we can define a forest language L'

consisting of the forests F where the nodes of F form a word of L (e.g., when taken in prefix order), and L' is then almost-commutative. As a kind of informal converse, given a Σ-forest F, we can represent it as a word with opening and closing parentheses (sometimes called the *XML encoding*), and the set of such representations for an almost-commutative forest language L' will intuitively form a word language L' that enjoys constant-time dynamic membership except for the (non-regular) requirement that parentheses are balanced.

We also note that we can effectively decide whether a given forest language is almost-commutative, as will follow from the algebraic characterization in the next section:

▶ **Proposition 5.4.** *The following problem is decidable: given a forest automaton A, determine whether the language accepted by A is almost-commutative.*

Tractability for almost-commutative languages. We show the main result of this section:

▶ **Theorem 5.5.** *For any fixed almost-commutative forest language L, the dynamic membership problem to L is in $O(1)$.*

Proof sketch. This result is shown by proving the claim for regular-commutative languages and virtually-singleton languages, and then noticing that tractability is preserved under Boolean operations, simply by combining data structures for the constituent languages.

For regular-commutative languages, we maintain the Parikh image as a vector in constant-time per update, and we then easily maintain the forest algebra element to which it corresponds, similarly to the case of monoids (see [23] or [4, Theorem 4.1]).

For virtually-singleton languages, we use doubly-linked lists like in [4, Prop. 4.3] to maintain, for each letter a of the subalphabet Σ', the unordered set of nodes with label a. This allows us to determine in constant-time whether the Parikh image of the input forest restricted to Σ' is correct: when this holds, then the doubly-linked lists have constant size and we can use them to recover all nodes with labels in Σ'. With constant-time reachability queries, we can then test if these nodes achieve the requisite forest F' over Σ'. ◀

6 Lower Bound on Non-Almost-Commutative Languages with Neutral Letter

We have introduced in the previous section the class of *almost-commutative languages*, and showed that such languages admit a constant-time dynamic membership algorithm (Theorem 5.5). In this section, we show that this class is tight: non-almost-commutative regular forest languages cannot enjoy constant-time dynamic membership when assuming the presence of a neutral letter and when assuming the hardness of the *prefix-U_1* problem from [4]. We first present these hypotheses in more detail, and state the lower bound that we show in this section (Theorem 6.2). Second, we present the algebraic characterization of almost-commutative regular languages on which the proof of Theorem 6.2 hinges: they are precisely the regular languages whose syntactic forest algebra is in a class called ZG. Third, we sketch the lower bound showing that dynamic membership is conditionally not in $O(1)$ for languages with a neutral letter whose syntactic forest algebra is not in ZG, and conclude.

Hypotheses and result statement. The lower bound shown in this section is conditional to a computational hypothesis on the *prefix-U_1 problem*. In this problem, we are given as input a word w on the alphabet $\Sigma = \{0, 1\}$, and we must handle substitution updates to w and queries where we are given $i \in \{1, \ldots, |w|\}$ and must return whether the prefix of w of length i contains some occurrence of 1. In other words, we must maintain a subset of

integers of $\{1, \ldots, |w|\}$ under insertion and deletion, and handle queries asking whether an input $i \in \{1, \ldots, |w|\}$ is greater than the current minimum: note that this like a priority queue except we can only compare the minimum to an input value but not retrieve it directly. We will use the following conjecture from [4] as a hypothesis for our lower bound:

▶ **Conjecture 6.1** ([4, Conj. 2.3]). *There is no data structure solving the prefix-U_1 problem in $O(1)$ time per operation in the RAM model with unit cost and logarithmic word size.*

Further, our lower bound in this section will be shown for regular forest languages L over an alphabet Σ which are assumed to feature a so-called *neutral letter* for L. Formally, a letter $e \in \Sigma$ is *neutral* for L if, for every forest F, we have $F \in L$ iff $\pi_{\Sigma\setminus\{e\}}(F) \in L$, where π denotes the projection operation of Definition 5.1. In other words, nodes labeled by e can be removed without affecting the membership of F to L.

We can now state the lower bound shown in this section, which is our main contribution:

▶ **Theorem 6.2.** *Let L be a regular forest language featuring a neutral letter. Assuming Conjecture 6.1, we have that L has dynamic membership in $O(1)$ iff L is almost-commutative.*

Algebraic characterization of almost-commutative languages. The proof of Theorem 6.2 hinges on an algebraic characterization of the almost-commutative languages. Namely, we will define a class of forest algebras called *ZG*, by imposing the *ZG equation* from [4, 5]:

▶ **Definition 6.3.** *A monoid M is in the class* ZG *if it satisfies the* ZG equation*: for all $v, w \in M$ we have $v^{\omega+1}w = wv^{\omega+1}$. A forest algebra (V, H) is in the class* ZG *if its vertical monoid is in ZG: we call it a* ZG forest algebra*.*

The intuition for the ZG equation is the following. Elements of the form $x^{\omega+1}$ in a monoid are called *group elements*: they are precisely the elements which span a subsemigroup (formed of the elements $x^{\omega+1}, x^{\omega+2}, \ldots$) which has the structure of a group (with $x^\omega = x^{2\omega}$ being the neutral element). Note that the group is always a cyclic group: it may be the trivial group, for instance in an aperiodic monoid all such groups are trivial, or in arbitrary monoids the neutral element always spans the trivial group. The equation implies that all group elements of the monoid are *central*, i.e., they commute with all other elements.

The point of ZG forest algebras is that they correspond to almost-commutative languages:

▶ **Theorem 6.4.** *A regular language L is almost-commutative if and only if its syntactic forest algebra is in ZG.*

Proving this algebraic characterization is the main technical challenge of the paper. Let us sketch the proof of Theorem 6.4, with the details given in the full version [2].

Proof sketch. The easy direction is to show that almost-commutative languages have a syntactic forest algebra in ZG. We first show this for commutative languages (whose vertical monoids are unsurprisingly commutative) and for virtually-singleton languages (whose vertical monoids are nilpotent, i.e., are in the class MNil of [4, 24]). We conclude because satisfying the ZG equation is preserved under Boolean operations.

The hard direction is to show that any regular language L whose syntactic forest algebra is in ZG is almost-commutative, i.e., can be expressed as a finite Boolean combination of virtually-singleton and regular-commutative languages. For this, we show that morphisms to a ZG forest algebra must be determined by the following information on the input forest: which letters are *rare* (i.e., occur a constant number of times) and which are *frequent*; how many times the frequent letters appear modulo the idempotent power of the monoid; and

what is the projection of the forest on the rare letters. These conditions amount to an almost-commutative language, and are the analogue for trees of results on ZG languages [4]. The proof is technical, because we must show how the ZG equation on the vertical monoid implies that every sufficiently frequent letter commutes both vertically and horizontally. ◀

Hardness for syntactic forest algebras outside ZG. To show our conditional hardness result (Theorem 6.2), what remains is to show that the dynamic membership problem is hard for regular languages with a neutral letter and whose syntactic forest algebra is not in ZG:

▶ **Proposition 6.5.** *Let L be a regular forest language, and assume that it has a neutral letter and that its syntactic forest algebra is not in ZG. Subject to Conjecture 6.1, the dynamic membership problem for L cannot be solved in constant time per update.*

Proof sketch. We reduce from the case of words: from two contexts v and $w^{\omega+1}$ witnessing that the ZG equation does not hold, we consider forests formed by the vertical composition of a sequence of contexts which can be either v or $w^{\omega+1}$, with a suitable context at the beginning and end. Let L' be the word language of those sequences of contexts giving a forest in L: we show that the syntactic monoid of L' is not in ZG, and as L' features a neutral letter, we conclude by [4] that L' does not enjoy $O(1)$ dynamic membership assuming Conjecture 6.1. Further, dynamic membership to L' can be achieved using a data structure for the same problem for L on the forests that we constructed, so hardness also holds for L. ◀

Note that Proposition 6.5 is where we use the assumption that there is a neutral letter. The result is not true otherwise, as we explain below (and will discuss further in Section 7):

▶ **Example 6.6.** *Consider the language L_0 of forests over $\Sigma = \{a, b, c\}$ where there is a node labeled a whose next sibling is labeled b. Membership to L_0 can be maintained in $O(1)$, like the language of words $\Sigma^* ab \Sigma^*$. (Note that c is not a neutral letter, because ab is accepted but acb is not.) However, one can show that the syntactic forest algebra of L_0 is not in ZG. By contrast, adding a neutral letter to L_0 yields a language (with the same syntactic forest algebra) with no $O(1)$ dynamic membership algorithm under Conjecture 6.1.*

With Proposition 6.5 and Theorem 6.4, we can conclude the proof of Theorem 6.2:

Proof of Theorem 6.2. We already know that almost-commutative languages can be maintained efficiently (Theorem 5.5). Now, given a regular forest language L which is not almost-commutative and features a neutral letter, we know by Theorem 6.4 that its syntactic forest algebra is not in ZG, so we conclude by Proposition 6.5. ◀

7 Conclusions and Future Work

We have studied the problem of dynamic membership to fixed tree languages under substitution updates. We have shown an $O(\log n / \log \log n)$ algorithm for arbitrary regular languages, and introduced the class of almost-commutative languages for which dynamic membership can be decided in $O(1)$. We have shown that, under the prefix-U1 conjecture, and if the language is regular and features a neutral letter, then it must be almost-commutative to be maintainable in constant time. Our work leaves many questions open which we present below: characterizing the $O(1)$-maintainable languages without neutral letter, identifying languages with complexity between $O(1)$ and $\Theta(\log n / \log \log n)$, and other open directions.

Constant-time dynamic membership without neutral letters. Our conditional characterization of constant-time-maintainable regular languages only holds in the presence of a neutral letter. In fact, without neutral letters, it is not difficult to find non-almost-commutative forest languages with constant-time dynamic membership, e.g., the language L_0 from Example 6.6. There are more complex examples, e.g., dynamic membership in $O(1)$ is possible for the word language "there is exactly one a and exactly one b, the a is before the b, and the distance between them is even" (it is in QLZG [4]), and the same holds for the analogous forest language. We can even precompute "structural information" on forests of a more complex nature than the parity of the depths, e.g., to maintain in $O(1)$ "there is exactly one a and exactly one b and the path from a to b is a downward path that takes only first-child edges". We also expect other constant-time tractable cases, e.g., "there is exactly one node labeled a and exactly one node labeled b and their least common ancestor is labeled c" using a linear-preprocessing constant-time data structure to answer least common ancestor queries on the tree structure [16]; or "there is a node labeled a where the leaf reached via first-child edges is labeled b". These tractable languages are not almost-commutative (and do not feature neutral letters), and a natural direction for future work would be to characterize the regular forest languages maintainable in $O(1)$ without the neutral letter assumption.

Intermediate complexity for dynamic membership. We have explained in Section 2 that dynamic membership is in $\Omega(\log n/\log\log n)$ for some regular forest languages, and in Section 5 that it is in $O(1)$ for almost-commutative languages. One natural question is to study intermediate complexity regimes. In the setting of word languages, it was shown in [23] (and extended in [4]) that any aperiodic language L could be maintained in $O(\log\log n)$ per update. This implies that some forest languages can be maintained with the same complexity, e.g., the forests whose nodes in the prefix ordering form a word in an aperiodic language L.

The natural question is then to characterize which forest languages enjoy dynamic membership between $O(1)$ and the general $\Theta(\log n/\log\log n)$ bound. We leave this question open, but note a difference with the setting for words: there are some aperiodic forest languages (i.e., both monoids of the syntactic forest algebra are aperiodic) to which an $\Omega(\log n/\log\log n)$ lower bound applies, e.g., the language for the existential marked ancestor problem reviewed at the end of Section 2. An intriguing question is whether there is a dichotomy on regular forest languages, already in the aperiodic case, between those with $O(\log\log n)$ dynamic membership, and those with a $\Omega(\log n/\log\log n)$ lower bound.

Other questions. Beyond relabeling updates it would be natural to study the complexity of dynamic membership under operations that can change the shape of the forest, e.g., leaf insertion and leaf deletion. Another question concerns the support for more general queries than dynamic membership, e.g., enumerating the answers to non-Boolean queries like in [19, 3] (but with language-dependent guarantees on the update time to improve over $O(\log n)$). Last, another generalization of dynamic membership to forest languages is to consider the same problem for context-free languages, e.g., for Dyck languages ([17, Proposition 1]), or for visibly pushdown languages. Note that this is a different setting from forest languages because substitution updates may change the shape of the parse tree.

References

1 Stephen Alstrup, Thore Husfeldt, and Theis Rauhe. Marked ancestor problems. In *FOCS*, 1998. doi:10.1109/SFCS.1998.743504.

2 Antoine Amarilli, Corentin Barloy, Louis Jachiet, and Charles Paperman. Dynamic membership for regular tree languages, 2025. doi:10.48550/arXiv.2504.17536.

3 Antoine Amarilli, Pierre Bourhis, and Stefan Mengel. Enumeration on trees under relabelings. In *ICDT*, 2018. doi:10.4230/LIPIcs.ICDT.2018.5.

4 Antoine Amarilli, Louis Jachiet, and Charles Paperman. Dynamic membership for regular languages. In *ICALP*, 2021. doi:10.4230/LIPIcs.ICALP.2021.116.

5 Antoine Amarilli and Charles Paperman. Locality and centrality: The variety ZG. *LMCS*, 2023. doi:10.46298/LMCS-19(4:4)2023.

6 Andrey Balmin, Yannis Papakonstantinou, and Victor Vianu. Incremental validation of XML documents. *TODS*, 29(4), 2004.

7 Corentin Barloy. *On the complexity of regular languages*. PhD thesis, Université de Lille, 2024. URL: https://theses.hal.science/tel-04820899v1/.

8 David A Mix Barrington, Neil Immerman, Clemens Lautemann, Nicole Schweikardt, and Denis Thérien. First-order expressibility of languages with neutral letters or: The Crane Beach conjecture. *JCSS*, 70(2), 2005. doi:10.1016/j.jcss.2004.07.004.

9 Hans L Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM Journal on Computing*, 27(6), 1998. doi:10.1137/S0097539795289859.

10 Mikolaj Bojanczyk and Igor Walukiewicz. Forest algebras. *Automata and logic: history and perspectives*, 2008.

11 Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. Tree automata: Techniques and applications, 2008. URL: https://hal.science/hal-03367725.

12 Gudmund Skovbjerg Frandsen, Thore Husfeldt, Peter Bro Miltersen, Theis Rauhe, and Søren Skyum. Dynamic algorithms for the Dyck languages. In *WADS*, pages 98–108, 1995.

13 Michael Fredman and Michael Saks. The cell probe complexity of dynamic data structures. In *STOC*, 1989.

14 Hillel Gazit, Gary L Miller, and Shang-Hua Teng. Optimal tree contraction in the EREW model. In *Concurrent Computations: Algorithms, Architecture, and Technology*. Springer, 1988.

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

16 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2), 1984. doi:10.1137/0213024.

17 Thore Husfeldt and Theis Rauhe. Hardness results for dynamic problems by extensions of Fredman and Saks' chronogram method. In *ICALP*, 1998.

18 Michal Koucký, Pavel Pudlák, and Denis Thérien. Bounded-depth circuits: Separating wires from gates. In *STOC*, 2005. doi:10.1145/1060590.1060629.

19 Katja Losemann and Wim Martens. MSO queries on trees: Enumerating answers under updates. In *CSL–LICS*, 2014. doi:10.1145/2603088.2603137.

20 Gary L Miller and John H Reif. Parallel tree contraction and its application. In *FOCS*, 1985. doi:10.1109/SFCS.1985.43.

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

22 Jonas Schmidt, Thomas Schwentick, and Jennifer Todtenhoefer. On the work of dynamic constant-time parallel algorithms for regular tree languages and context-free languages. In *MFCS*, volume 272, 2023. doi:10.4230/LIPIcs.MFCS.2023.81.

23 Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *JACM*, 44(2), 1997.

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

25 Felix Tschirbs, Nils Vortmeier, and Thomas Zeume. Dynamic complexity of regular languages: Big changes, small work. In *CSL*, 2023.

26 Moshe Y Vardi. The complexity of relational query languages. In *STOC*, 1982.