Dynamic Complexity of Regular Languages: Big Changes, Small Work

Felix Tschirbs
Ruhr-Universität Bochum, Germany

Nils Vortmeier
Ruhr-Universität Bochum, Germany

Thomas Zeume
Ruhr-Universität Bochum, Germany

Abstract

Whether a changing string is member of a certain regular language can be maintained in the DynFO framework of Patnaik and Immerman: after changing the symbol at one position of the string, a first-order update formula can express – using additionally stored information – whether the resulting string is in the regular language.

We extend this and further known results by considering changes of many positions at once. We also investigate to which degree the obtained update formulas imply work-efficient parallel dynamic algorithms.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Logic and databases

Keywords and phrases dynamic descriptive complexity, regular languages, batch changes, work

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.35

Acknowledgements We are grateful to Jonas Schmidt and Thomas Schwentick for insightful discussions.

1 Introduction

Which queries allow for efficient, parallel updates of their results under changes to their input? Dynamic descriptive complexity theory is a fundamental framework for addressing this question, which was proposed independently by Patnaik and Immerman [17] as well as by Dong and Su [8] in the early nineties. In their frameworks, dynamic update programs formalized via first-order formulas are used for updating both query results and helpful auxiliary data after changes to an input structure. Queries maintainable in this fashion are said to be in the class DynFO and, due to classical correspondences [25], can also be maintained in constant time by parallel random-access machines with polynomially many processors.

The class DynFO is surprisingly powerful. It is known, for instance, that all context-free languages [10, Theorem 4.1] as well as the reachability query [4, Theorem 1] can be maintained by first-order update programs, and that all properties expressible in monadic second-order logic can be maintained assuming that the input structure retains bounded treewidth [6, Theorem 6.1]. All these results have been stated solely for changes that modify a single tuple. Also, although these results imply that updates can be performed in constant parallel time, they usually do not consider the necessary work, that is, the total number of necessary computation steps, to perform such an update. A DynFO program that uses first-order formulas of quantifier-depth ℓ to update k-ary auxiliary relations over a domain of size n essentially requires work $n^{\ell+k}$ if it is evaluated naïvely. In fact, many DynFO programs require at least quadratic work, some updates for important queries as graph reachability have a polynomial work bound with much higher degree.

Towards practical applications, therefore two additional aspects need to be taken into account: (1) How to deal with batch changes of non-constant size?, and (2) How much work is required for updates after changes?

Several results for maintaining queries under batch changes have been obtained recently. Reachability is in DynFO under changes of $\mathcal{O}(\frac{\log n}{\log \log n})$ many edges [7, Theorem 1], and even under changes of size $\mathcal{O}(\log^c n)$, for any $c \in \mathbb{N}$, for undirected graphs or for insertions only [5, Theorems 6 and 7]. For declaratively specified batch changes, it is known that undirected reachability can be maintained under first-order defined insertions and acyclic reachability under insertions defined by quantifier-free formulas [22, Theorems 4.2 and 4.5]. Context-free languages are maintainable under Σ_1-defined changes [21, Theorem 9].

For work-efficient dynamic first-order programs, not much has been done so far. A framework for studying work-efficient DynFO is introduced in [18]. There it is shown that under changes of single positions it is possible to maintain regular languages with $\mathcal{O}(n^\epsilon)$ work for all $\epsilon > 0$, and that star-free languages and certain context-free (non-regular) languages can be maintained with polylogarithmic work.

The focus of this paper is to study these two aspects for the membership problem for regular languages, and in particular to explore how to deal with batch changes with as little work as possible. Our main contributions are:

- Regular (string) languages and regular tree languages can be maintained under changes of polylogarithmic size. For regular string languages, only work $\mathcal{O}(n^\epsilon)$ is necessary, for all $\epsilon > 0$.
- Star-free (string) languages can be maintained under changes of polylogarithmic size with polylogarithmic work. No significant improvement is possible with respect to the class of languages.

The main proof tool for the upper bounds, already used in [5], is to exploit the power of first-order formulas on small substructures. Intuitively, on small substructures, first-order quantifiers are as powerful as (restricted) second-order quantifiers. Exploiting this yields that first-order formulas are as powerful as LOGCFL on substructures of polylogarithmic size – a fact previously observed in the context of constant-depth circuits of size 2^{n^ϵ} in [1, cf. Lemma 8.1]. For maintaining queries under changes of polylogarithmic size, this can now be used by performing LOGCFL computations on structures defined on the elements affected by the change. We also investigate the expressive power of second-order quantification over relations of bounded size.

We infer the lower bounds from classical lower bounds from circuit complexity.

Parts of these results have been included in the PhD thesis of the second author [26].

Outline. After recalling essential notions from descriptive dynamic complexity theory in Section 2, we discuss how the expressivity of first-order formulas on small substructures can be exploited for dealing with large changes to a structure in Sections 3 and 6. We then introduce our techniques by establishing that regular languages can be maintained under changes of polylogarithmic size in Section 4, and discuss how to generalize these techniques to dynamic programs that use little work in Section 5.

2 The dynamic setting

We introduce the main notions of the dynamic complexity framework, following [23, 18]. We slightly adapt the framework to take into account both batch changes and the required amount of work.

Preliminaries. A schema σ is a set of relation symbols and function symbols, each with a corresponding arity. A relational structure $\mathcal{S}$ over some schema σ consists of a finite domain D, a relation $R^{\mathcal{S}} \subseteq D^k$ for each k-ary relation symbol $R \in \sigma$ and a function $f^{\mathcal{S}} \colon D^k \to D$ for every k-ary function symbol $f \in \sigma$. All structures considered in this work are equipped with a linear order $\leq$ on their domain D, so we assume that D is an initial segment $\{0, \dots, n-1\}$ of the natural numbers.

Words over an alphabet Σ are represented by structures over the schema that includes $\leq$ and a relation R_σ for every $\sigma \in \Sigma$: domain elements represent the positions of the word, and $i \in R_\sigma$ means that position i carries the symbol σ. A position i may be included in at most one relation R_σ; if it is not included in any such relation we say that it is *empty* and labeled with ϵ. Labeled trees have an additional relation E such that $(u, v) \in E$ holds exactly if v is the parent of u.

First-order logic FO is defined in the usual way, we refer to [15] for details. We allow if-then-else constructs in terms: if t_1 and t_2 are terms and φ is a formula, then $\mathrm{ITE}(\varphi, t_1, t_2)$ is a term. It evaluates to the valuation of t_1 if φ is satisfied, and to the valuation of t_2 otherwise. Throughout this work we allow first-order formulas to access the linear order $\leq$ on the structures and relations $+$ and $\times$ that encode addition and multiplication on the domain D, and write $\mathsf{FO}(\leq, +, \times)$ to make this explicit.

Circuits and complexity classes. First-order logic with the numeric relations $\leq, +, \times$ is equivalent to (first-order uniform) AC^0, the class of problems that are decided by uniform families of constant-depth circuits with polynomially many "and"-, "or"- and "not"-gates, where the former two may have unbounded fan-in. The complexity class LOGCFL contains all problems that can be reduced in logarithmic space to a context-free language. It includes NL and is included in AC^1.

Dynamic problems. In our setting, a dynamic problem is characterized by (1) the schema σ_{in} of an input structure, and (2) the set Δ of allowed change and query operations.

An example is the dynamic membership problem for some language L:

- The input structure is a word $w = w_0 \cdots w_{n-1}$ of some length n over the alphabet Σ of L.
- Change operations have the form $\textsc{set}_\sigma(P)$, for $\sigma \in \Sigma \cup \{\epsilon\}$. A concrete such change has as parameter a set P of positions and is applied by setting the symbols for all positions $i \in P$ to σ.
- The operation $\textsc{member}$ allows querying whether the current word is in L, where empty positions are ignored. So, it returns whether $w_0 \circ \cdots \circ w_{n-1} \in L$ holds.

For the purpose of this paper, a change operation in general has the form $\delta(P)$, where P is a relation variable. This definition reflects that we are interested in changes of many elements at once. We use change operations that restrict the allowed size of a parameter relation P, as allowing the input to change arbitrarily basically turns a dynamic problem into a static problem, as one needs to solve the problem from scratch any time the input is replaced. Mostly we are interested in changes that insert or delete a polylogarithmic number of tuples with respect to the domain size.

To ensure that we are able to access changed elements with little work, so, without enumerating over all elements, we demand that each change $\delta(P)$ for a unary relation P comes with a function $\textsc{chd} \colon D \to D$ that maps the i-th element of the (linearly ordered) domain D to the i-th element in P according to the order on D, for all $i \leq |P|$. If P is a k-ary relation, there are k functions $\textsc{chd}_1, \dots, \textsc{chd}_k$ that map to the first, …, k-th element in the i-th tuple in P according to the lexicographic ordering.

Query operations have the form $Q(\bar{p})$, where $\bar{p}$ is a parameter tuple of element variables. For instance, the MEMBER query from above is a query operation without parameters. An example for a query operation on words with parameters is $\text{RANGE}(\ell, r)$, which receives two positions ℓ and r as parameters and asks whether $w_\ell \circ \cdots \circ w_r$ is in L. We also consider query operations that return domain elements instead of a truth value. For example, consider a dynamic problem NEXTINSET that has as input a set $S \subseteq \{1, \ldots, n\}$, change operations $\text{INS}(P)$ and $\text{DEL}(P)$ that respectively insert a set $P \subseteq \{1, \ldots, n\}$ of numbers into S and delete such a set from S, and query operations $\text{PRED}(i)$ and $\text{SUCC}(i)$, where $\text{PRED}(i)$ returns the predecessor of i in S (so, the largest element of S that is smaller than i) and $\text{SUCC}(i)$ returns its successor (the smallest element of S that is larger than i).

Maintenance of dynamic problems. The goal of a *dynamic program* $\mathcal{P}$ is to maintain some dynamic problem Π: it must be able to answer queries to the input structure which is allowed to change according to the change operations. To do so, $\mathcal{P}$ stores and updates an auxiliary structure $\mathcal{A}$ over some schema σ_{aux} and over the same domain as the input structure. This structure $\mathcal{A}$ consists of a set of auxiliary relations and auxiliary functions.

A first-order dynamic program expresses an update of its auxiliary structure as well as the answer to a query by means of first-order formulas and terms. For each query operation Q with parameters $\bar{p}$ it has a *query rule* of the form **on query** $Q(\bar{p})$ **yield** $\varphi_Q(\bar{p})$, where φ_Q is a (first-order) *query formula* or *query term*, depending on the type of the result of Q, over the combined schema $\sigma_{\text{in}} \cup \sigma_{\text{aux}}$ of input and auxiliary structure.

For each change operation $\delta(P)$ and each auxiliary relation symbol $R \in \sigma_{\text{aux}}$ with arity k the dynamic program has an *update rule* of the form

on change $\delta(P)$ **update** R **at** $(t_1(\bar{x}), \ldots, t_k(\bar{x}))$ **as** $\varphi^R_\delta(P; \bar{x})$ **for all** $\varphi_C(\bar{x})$.

Here, $\varphi^R_\delta(P; \bar{x})$ is a (first-order) *update formula* and $t_1, \ldots, t_k$ are first-order terms, all over the schema $\sigma_{\text{in}} \cup \sigma_{\text{aux}} \cup \{P, \text{CHD}\}$, and $\varphi_C(\bar{x})$ is a *constraint formula* for the tuple $\bar{x} = x_1, \ldots, x_\ell$ of variables. The constraint formula is a conjunction of inequalities $x_i \leq f_i(n)$ for $i \in \{1, \ldots, \ell\}$, using $\mathsf{FO}(\leq, +, \times)$-definable functions $f_i : \mathbb{N} \to \mathbb{N}$.

After a change $\delta(P)$ to an input structure $\mathcal{I}$ with current auxiliary structure $\mathcal{A}$, the effect of an update on the auxiliary relation R is as follows. Let $\mathcal{I}'$ be the input structure after the change. The new relation $R^{\mathcal{A}'}$ in the updated auxiliary structure $\mathcal{A}'$ contains all (old) tuples $\bar{a} \in R^{\mathcal{A}}$ such that $\bar{a}$ is *not* equal to $(t_1(\bar{b}), \ldots, t_k(\bar{b}))$ for any tuple $\bar{b}$ that satisfies φ_C. Additionally, $R^{\mathcal{A}'}$ contains all tuples $(t_1(\bar{b}), \ldots, t_k(\bar{b}))$ such that $\bar{b}$ satisfies φ_C and $\varphi^R_\delta(\bar{b})$ holds in $(\mathcal{I}', \mathcal{A}, P, \text{CHD})$.

Phrased differently, φ_C is used to enumerate all tuples $\bar{b}$ such that containment of $(t_1(\bar{b}), \ldots, t_k(\bar{b}))$ in R may change. Whether that tuple is inserted into R (if it was not already present) or deleted from R (if it was already present) depends on the evaluation of the update formula $\varphi^R_\delta(\bar{b})$ on the changed input structure and the old auxiliary structure, given the information on the change provided by P and CHD.

Update rules for auxiliary functions are defined analogously, but use an update term (instead of $\varphi^R_\delta(P; \bar{x})$) that determines the new function value for a tuple $(t_1(\bar{b}), \ldots, t_k(\bar{b}))$ such that $\bar{b}$ satisfies the constraint.

We say that a dynamic program $\mathcal{P}$ *maintains* a dynamic problem Π, if it can always answer all queries correctly. More precisely, $\mathcal{P}$ maintains Π if applying a query operation after a sequence α of change operations on an initial structure $\mathcal{I}_0$ yields the same result as evaluating the corresponding query rule on the combined input and auxiliary structure that is obtained by applying the update rules corresponding to α to $(\mathcal{I}_0, \mathcal{A}_0)$, where $\mathcal{A}_0$ is

an initial auxiliary structure. Following Patnaik and Immerman [17], we demand that the initial input structure $\mathcal{I}_0$ is *empty*, so, has empty relations and all function values being 0. The initial auxiliary structure is over the same domain as $\mathcal{I}_0$ and is defined from $\mathcal{I}_0$ by some first-order definable initialization.

The class DynFO is the class of all dynamic problems that are maintained by some first-order dynamic program.

The work of a dynamic program. It is not immediately clear how to measure the work of a dynamic program, as first-order logic is declarative. Schmidt et al. [18] introduce a detailed framework that allows for a comparatively easy way to specify updates and queries, as well as to associate a work bound. This is done via constant-time PRAMs that implement a first-order dynamic program, for which the amount of work is well-defined as the total number of the steps of the processors.

The framework of [18] introduces more intricate syntax elements for specifying dynamic programs that allow in some cases for a translation into more efficient PRAMs. For example, in the syntax of [18] one can distinguish parts of an update formula that need to be evaluated for every tuple $(t_1(\bar{x}), \dots, t_k(\bar{x}))$ and parts that only need to be evaluated once.

In this paper, we are only interested in work bounds of the form "polylogarithmic in n" or "$\mathcal{O}(n^\epsilon)$ for arbitrary $\epsilon > 0$", so single logarithmic factors are less important. This allows us to be a bit coarser.

For us, the work of a first-order formula is just the number of wires in the AC^0 circuit that is obtained by the straight-forward translation of first-order formulas into bounded-depth circuits. We allow quantifiers of the form $\exists x \leq f(n)$ and $\forall x \leq f(n)$, for $\mathsf{FO}(\leq, +, \times)$-definable functions $f\colon \mathbb{N} \to \mathbb{N}$, to prevent that quantifiers always introduce a factor of n for the work bound.

The work of a query rule is the work of its query formula. The work of an update rule with constraint formula $\varphi_C(\bar{x}) = x_1 \leq f_1(n) \wedge \dots \wedge x_\ell \leq f_\ell(n)$ is the product of the functions f_i and the work of the update formula, as the update formula needs to be evaluated $f_1(n) \cdot \dots \cdot f_\ell(n)$ times.

An example: NextInSet. To illustrate the setting, and because we use this result in Section 5, we show that NextInSet can be maintained under changes of polylogarithmic size with a polylogarithmic amount of work. Recall that NextInSet provides two queries pred(i) and succ(i) that return the predecessor respectively the successor of i in a linearly ordered set S that is subject to insertions and deletions.

▶ **Lemma 1.** *For every $c \in \mathbb{N}$ there is a $d \in \mathbb{N}$ such that* NextInSet can be maintained in $\mathsf{DynFO}(\leq, +, \times)$ under changes of size $\log^c n$ with $\mathcal{O}(\log^d n)$ work.

Proof. We adapt the proof of [18, Lemma 4.1] that shows that NextInSet can be maintained in DynFO with work $\mathcal{O}(\log n)$ under insertions and deletions of single elements. The dynamic program maintains a complete binary tree with ordered leaves $\{1, \dots, |D|\}$, where D is the domain, including a function anc such that anc(x, k) returns the k-th ancestor of x in the tree. We associate with every tree node an interval according to the leaves in its subtree. We say that a node *covers* an element if it is in the interval associated with the node. The auxiliary structure stores for each node the minimum and maximum element of S that is covered by the node. A succ(i) or pred(i) query can be answered with $\mathcal{O}(\log n)$ work using this information (cf. [18]).

To update the auxiliary information when $\log^c n$ elements are inserted or deleted, one can enumerate the $\log^c n \cdot \log n$ tree nodes for which the information needs to be updated: all nodes that lie on the path from a leave corresponding to a changed element to the root of the tree. For each such node x, the new maximum is calculated as follows:

- **Insertions.** First, find the largest inserted element that is covered by x: the inserted element i that is covered by x and is larger than any other inserted element covered by x. This element can be identified with $\log^{2c} n$ work. Compare i with the current maximum of x and update the maximum if necessary.
- **Deletions.** If the current maximum is deleted, use $\textsc{pred}(i)$ to query the predecessor for each deleted element i covered by x, using $\mathcal{O}(\log n)$ work per query. By pairwise comparison, find the largest such predecessor that is not itself deleted. If it is covered by x, it is the new maximum, otherwise there is none.

The minimum of x is updated symmetrically. In total, an update can be performed using polylogarithmic work.

Formally, the update rule for the auxiliary function max that maps an inner node x to the maximal element of S covered by x is as follows:

$$\begin{aligned}&\textbf{on change } \textsc{ins}(P) \textbf{ update } \max \textbf{ at } \operatorname{anc}(\textsc{chd}(x_1), x_2) \textbf{ as } \varphi^{\max}_{\textsc{ins}}(x_1, x_2)\\ &\qquad\qquad \textbf{for all } x_1 \leq \log^c n \wedge x_2 \leq \log n\end{aligned}$$

Here, we address an inner node x as the x_2-th tree-ancestor of the x_1-th changed element. The update term $\varphi^{\max}_{\textsc{ins}}(x_1, x_2)$ is defined as $\mathrm{ITE}(\max_{\text{ins}} > \max_{\text{cur}}, \max_{\text{ins}}, \max_{\text{cur}})$, where $\max_{\text{cur}} = \max(\operatorname{anc}(\textsc{chd}(x_1), x_2))$ gives the current maximum for node x, and $\max_{\text{ins}}$ is the maximal changed element $\textsc{chd}(x_i)$ such that the formula $\operatorname{anc}(\textsc{chd}(x_1), x_2) = \operatorname{anc}(\textsc{chd}(x_i), x_2)$ is satisfied. ◀

3 First-order logic on substructures of polylogarithmic size

When a string is changed in polylogarithmically many positions, first-order update formulas for maintaining some language L have to be able, at a minimum, to decide for strings of this size whether they are a member of L. This is necessary for dealing with polylogarithmically many changes of subsequent positions. In the more general case that the changes are not subsequent, changes of polylogarithmic size partition a string into as many pieces. Known information about these pieces needs then to be combined by first-order update formulas to decide membership in L for the new string.

Thus, for maintaining properties under changes of polylogarithmic size, it is helpful to know the expressive power of first-order formulas on small substructures of this size.

We use the well-known fact that many complexity classes can be captured by bounded-depth circuits of subexponential size.

For the moment we simply state the result and infer a corollary suitable for our needs. Later, in Section 6, we will re-visit these results from another perspective.

▶ **Lemma 2** ([1, cf. Lemma 8.1]). *For every language $L \in \mathsf{LOGCFL}$ and every $\epsilon > 0$ there is a d such that L can be decided by depth-d circuits of size 2^{n^ϵ}.*

It follows that first-order formulas can express LOGCFL-computable queries on substructures of polylogarithmic size, which we formalize in Corollary 3. See also [5, Theorem 3] for an analogous statement regarding NL-computable queries. Corollary 3 requires that the circuits from Lemma 2 can be constructed uniformly, which is implicit in the proof given by [1] and will be made explicit in our discussion in Section 6.

▶ **Corollary 3.** *Let k and c be arbitrary natural numbers, and let Q be a k-ary,* LOGCFL*-computable query on σ-structures. There is an* FO$(\leq, +, \times)$ *formula φ over schema $\sigma \cup \{C\}$ such that for any σ-structure $\mathcal{S}$ with n elements, any subset C of its domain of size at most $\mathcal{O}(\log^c n)$, and any k-tuple $\bar{a} \in C^k$ it holds that*

$$\bar{a} \in Q(\mathcal{S}[C]) \text{ if and only if } (\mathcal{S}, C) \models \varphi(\bar{a}).$$

Here, $\mathcal{S}[C]$ denotes the substructure of $\mathcal{S}$ induced by C. Moreover, there is an equivalent AC^0 *circuit of size $\mathcal{O}(n^\epsilon)$, for every ϵ with $0 < \epsilon < 1$.*

Proof. Let Q, c and ϵ be fixed with $0 < \epsilon < 1$. The bounded-depth circuits that are guaranteed to exist by Lemma 2 for $\hat{\epsilon} = \frac{\epsilon}{c}$ have polynomial size $2^{(\log n)^\epsilon} = \mathcal{O}(n^\epsilon)$, so, are AC^0-circuits. As every AC^0-circuit has an equivalent FO$(\leq, +, \times)$-formula, the statement follows. ◀

4 Regular languages under many changes

We show in this section that regular languages of strings and trees can be maintained in DynFO. For now, we will not aim for work-efficient updates; this aspect will be treated in Section 5. We prove these intermediate results also to introduce the techniques.

For both string and tree languages our approach is similar. Every change affects a polylogarithmic number of elements of the underlying string or tree, respectively. We enrich the substructure that is induced by these affected elements by information from the maintained auxiliary relations. Thanks to Corollary 3, the first-order update formulas can express any query from LOGCFL on this substructure, which provides sufficient information to tell whether the full structure is a member of the regular language.

▶ **Theorem 4.** *One can maintain in* DynFO$(\leq, +, \times)$
(a) *membership in any regular (string) language, and*
(b) *membership in any regular tree language*
under changes of $\log^c n$ many symbols, for any fixed $c \in \mathbb{N}$, where n is the size of the underlying string respectively tree.

Proof. Towards (a), fix a regular language L over some alphabet Σ, and let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a DFA for L.

To maintain membership in L under changes of single positions, the dynamic program by Gelade et al. [10] uses auxiliary relations of the form $S_{p,q}(k, \ell)$ for all states p and q of $\mathcal{A}$. It maintains that pairs (k, ℓ) of positions are in $S_{p,q}$ if and only if $\delta^*(p, w_k \cdots w_\ell) = q$. Here, $w = w_1 \cdots w_n$ is the input word and δ^* is the extension of the transition function δ from symbols $\sigma \in \Sigma$ to words from Σ^*. Clearly, $w \in L$ if and only if $(1, n) \in S_{q_0,q_f}$ for some accepting state $q_f \in F$ of $\mathcal{A}$; and this condition can easily be expressed by a first-order formula.

We show that these auxiliary relations can be maintained in DynFO$(\leq, +, \times)$ under changes of polylogarithmic size. In a nutshell, we show that updating these auxiliary relations after such changes boils down to computing the product of polylogarithmically many elements of the transition monoid of $\mathcal{A}$. This product can be computed in FO$(\leq, +, \times)$ by Corollary 3, as it can be done in $\mathsf{NC}^1 \subseteq$ LOGCFL.

We make this approach more precise now. When $w = w_1 \cdots w_n$ is changed to $w' = w'_1 \cdots w'_n$, the auxiliary relations $S_{p,q}$ are updated as follows. An interval (k, ℓ) is in the new auxiliary relation $S_{p,q}$ if and only if $\delta^*(p, w'_k \cdots w'_\ell) = q$. The update formula, for each interval (k, ℓ),

(1) splits $w'_k \cdots w'_\ell$ into at most $\mathcal{O}(\log^c n)$ pieces $u_0, \ldots, u_m$ according to the changes;
(2) assigns to each piece u_j its corresponding element $\gamma_j : Q \to Q$ of the transition monoid of $\mathcal{A}$, i.e., the function γ_j that maps $p' \in Q$ to $q' \in Q$ if $\mathcal{A}$ transitions from p' to q' when reading u_j;
(3) computes the product $\gamma \stackrel{\text{def}}{=} \gamma'_m \circ \cdots \circ \gamma'_0$; and
(4) lets $(k, \ell) \in S_{p,q}$ if and only if $\gamma(p) = q$.

More precisely, suppose the change operation affected the positions $\{i_1, \ldots, i_m\}$ in the interval (k, ℓ). For (1), define $P \stackrel{\text{def}}{=} \{i_0, i_1, \ldots, i_m, i_{m+1}\}$, where $i_0 = k$ and $i_{m+1} = \ell + 1$. For each $0 \leq j \leq m$, the piece u_j is defined as the substring $w'_{i_j} \cdots w'_{i_{j+1}-1}$ of $w'_{i_1} \cdots w'_{i_m}$ that starts with a changed position i_j and ends just before the next changed position (except for the border cases). For (2), the element γ_j of the transition monoid for $\mathcal{A}$ on u_j can be inferred from the stored auxiliary relations. The computation in (3) can be performed in $\mathsf{FO}(\leq, +, \times)$ due to Corollary 3.

Towards (b), fix a regular tree language L. We assume that input trees T are represented as structures with relations FirstChild and NextSibling with the obvious meaning. This is without loss of generality, as these relations can be defined by first-order formulas for any ordered tree. Using these relations, we can regard T to be a binary tree. It is well-known (see for example [16, Lemma 1]) that any regular tree language over ranked or unranked trees is accepted by a (finite deterministic bottom-up) tree automaton that reads the FirstChild-NextSibling encoding of the tree. Such a tree automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, with state set Q, transition function $\delta\colon Q \times Q \times \Sigma \to Q$, initial state q_0 and set F of accepting states, assigns in a bottom-up run to a leaf with symbol σ the state $\delta(q_0, q_0, \sigma)$ and to an inner node with symbol σ and children labeled with states p, q, respectively, the state $\delta(p, q, \sigma)$. It accepts if the state assigned to the root is in F.

To maintain membership in L under changes of a single node label, Gelade et al. in [10] mainly use binary auxiliary relations that are again of the form $S_{p,q}(u, v)$, for states $p, q \in Q$. Intuitively, $(u, v) \in S_{p,q}$ means that $\mathcal{A}$ assigns the state p to the node u, if the state for node v is forced to be q – no matter which state $\mathcal{A}$ actually assigns to v based on its subtree and the transition function δ.

We show that these auxiliary relations can be updated after changes of the labels of polylogarithmically many nodes. For this we assume the existence of the descendant relation on the tree nodes, which can be made available as an auxiliary relation. Our strategy is analogous to part (a): we show that first-order formulas can define a labeled binary tree with polylogarithmically many nodes such that from the information whether this tree is a member of some other regular tree language we can infer whether the changed input tree is in L, and how the other auxiliary relations need to be updated. Thanks to Corollary 3, this approach can be implemented by $\mathsf{FO}(\leq, +, \times)$ update formulas, as regular tree language membership (for deterministic bottom-up automata) is in $\mathsf{NC}^1 \subseteq \mathsf{LOGCFL}$.

We give some more details. Let P be the set of at most $\mathcal{O}(\log^c n)$ nodes whose labels are modified by a change, resulting in an input tree T'. Without loss of generality, we assume that for two nodes $v, v' \in P$ also their lowest common ancestor is in P, and that if for a node $v \in P$ a descendant from its left or right subtree is contained in P, also some descendant from its other subtree is in P. So, with the help of the descendant relation on T, we can define in FO a binary tree T_P with node set P and edges from a node to its "nearest descendants" in T.

We assign labels to the nodes of T_P according to the behavior of $\mathcal{A}$ on the subtree of T' that is rooted in the respective node. If $v \in P$ is a leaf of T_P, then its label is just the state $q \in Q$ that $\mathcal{A}$ assigns to this node in T'. As the only difference in the subtree rooted at v

between T and T' is the label of v itself, this state can easily be expressed using the old auxiliary relations and the transition function of $\mathcal{A}$. If $v \in P$ is an inner node of T_P, its label is a function $\gamma\colon Q \times Q \to Q$. Intuitively, this labels says that if $\mathcal{A}$ assigns in T' some states p, q to the nodes u_1, u_2 that are the children of v in T_P, then it assigns the state $\gamma(p, q)$ to v in T'. This label can be expressed in first-order logic from the old auxiliary relations as follows. Let v_1, v_2 be the children of v in T such that u_1 is a descendant of v_1 and u_2 is a descendant of v_2. If the state p is assigned to u_1 in T', then the state p' is assigned to v_1 that satisfies $(u_1, v_1) \in S_{p,p'}$. Symmetrically one can determine the state q' that is assigned to v_2 in T'. The state that is assigned to v can then be read from the transition function of $\mathcal{A}$.

Another bottom-up tree automaton can "evaluate" T_P in the natural way. By Corollary 3, an $\mathsf{FO}(\leq, +, \times)$ update formula can determine the state that this automaton assigns to every inner node, and in particular the state $q' \in Q$ that it assigns to the root of T_p. From this state, the update formula can determine the state q that $\mathcal{A}$ assigns to the root of T', and therefore can check whether $T' \in L$. The auxiliary relations of the form $S_{p,q}$ can be updated similarly. ◀

5 Regular languages, big changes, small work

So far we have seen that membership in regular languages can be maintained under polylogarithmic changes. In this section, we will re-visit this result and analyze the required work. We employ the framework for work-efficient dynamic complexity introduced in Section 2.

In [18] it was shown that regular languages can be maintained with work $\mathcal{O}(n^\epsilon)$ under single-tuple changes. We first lift this result to changes of polylogarithmic size. After arguing why polylogarithmic work cannot be achieved for polylogarithmic changes, we will then take a look at first-order definable regular languages. Here, again, we lift a result from [18] and show that such languages can be maintained with polylogarithmic work under changes of polylogarithmic size. It remains open whether membership in regular languages can be maintained in DynFO with $\mathcal{O}(\log^d n)$ work for some d under single tuple changes; in Section 7 we will discuss why proving barriers likely requires new insights.

We start by analyzing the work required for maintaining membership in regular languages under polylogarithmic changes.

▶ **Theorem 5.** *One can maintain membership in regular languages in* $\mathsf{DynFO}(\leq, +, \times)$ *with work* $\mathcal{O}(n^\epsilon)$ *under changes of size* $\log^c n$*, for* $c \in \mathbb{N}$ *and all* $\epsilon > 0$.

Proof. In Theorem 4, membership of a regular language L was maintained by storing, for each pair (k, ℓ) of positions of the input string w, the behavior of a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ accepting L when reading $w_k \cdots w_\ell$. When a symbol is changed, $\Theta(n^2)$ such pairs are affected in the worst case, resulting in an at least quadratic amount of work for processing large changes.

Schmidt et al. [18] introduced a technique for decreasing the amount of work to $\mathcal{O}(n^\epsilon)$, for all $\epsilon > 0$. The technique can be lifted to polylogarithmic changes in a straightforward fashion. For the sake of completeness we sketch the idea.

The idea in Schmidt et al. [18] is to store elements of the transition monoid of the automaton only for some well-chosen subwords called special intervals, such that each position is contained in only $\mathcal{O}(n^{\epsilon'})$ special intervals, for any $\epsilon' > 0$. These special intervals are arranged in a hierarchy whose depth depends solely on ϵ'. It is then shown that the element of the transition monoid for each (not necessarily special) interval can be computed bottom-up with the help of a constant number of special intervals.

We use the same hierarchy of intervals for polylogarithmic changes and apply the strategy from the proof of Theorem 4. Changes are now distributed to intervals of the hierarchy. As each changed symbol affects $\mathcal{O}(n^{\epsilon'})$ special intervals, $\mathcal{O}(\log^c n \cdot n^{\epsilon'})$ special intervals need to be updated.

For each special interval that contains at least one changed position, we execute steps (1)–(4) from the proof of Theorem 4. Step (1) splits the interval into pieces, each containing exactly one changed position; this requires $\mathcal{O}(\log^c n)$ work. Step (2) computes the transition monoid element of each piece; this requires constant work per piece. Step (3) computes the product of up to $\mathcal{O}(\log^c n)$ monoid elements; this can be done in $\mathcal{O}(n^{\epsilon''})$ work, for any $\epsilon'' > 0$ by Corollary 3. Step (4) requires a constant amount of work. This sums up to $\mathcal{O}(n^{\epsilon''})$ work needed to update one pair (k, ℓ) in a relation $S_{p,q}$.

In total, $\mathcal{O}(\log^c n \cdot n^{\epsilon'} \cdot n^{\epsilon''}) = \mathcal{O}(n^{\epsilon})$ work is needed to process an update of size $\log^c n$, for suitable choices of ϵ', ϵ''. ◀

We now show that one cannot do much better than stated in the previous theorem: there are regular languages which cannot be maintained with polylogarithmic work under polylogarithmic changes. Even more, we will exactly characterize the languages that can be maintained with polylogarithmic work under such changes.

A language is *star-free* if it can be expressed by a regular expression without Kleene star but with negation. Star-free languages have many equivalent characterizations: A language is star-free if and only if it is first-order definable (with only $<$) if and only if its syntactic monoid is aperiodic [20]. The *syntactic monoid* of L is the monoid with the least number of elements that recognizes L. Here, a monoid M *recognizes* a language $L \subseteq \Sigma^*$ if there is a morphism $h\colon \Sigma^* \to M$ and a set $F \subseteq M$ such that $L = h^{-1}(F)$. A monoid M is *aperiodic* if there is a k such that $m^k = m^{k+1}$ for all $m \in M$.

▶ **Theorem 6.** *For a regular language L, the following are equivalent:*

(a) *L is star-free.*

(b) *For all $c \in \mathbb{N}$ and some $d \in \mathbb{N}$ depending on c, L can be maintained in* $\mathsf{DynFO}(\leq, +, \times)$ *with work $\mathcal{O}(\log^d n)$ under changes of size $\mathcal{O}(\log^c n)$.*

We first prove that non-star-free languages cannot be maintained with polylogarithmic work under polylogarithmic changes. The following lemma will be helpful.

▶ **Lemma 7.** *If a language L can be maintained in* $\mathsf{DynFO}(\leq, +, \times)$ *with $f(n)$ work under insertions of size $\log n$ then there is a constant-depth circuit of size $\mathcal{O}(f(2^n))$ that decides L (for inputs of size n).*

Proof sketch. Suppose L can be maintained in $\mathsf{DynFO}(\leq, +, \times)$ with $f(n)$ work under insertions of size $\log n$. From the update formulas for L one can construct a constant-depth circuit family which has $\mathcal{O}(f(n))$ many gates for inputs of length $\log n$, using the standard conversion from first-order formulas to circuits. Such a circuit simulates the update formulas for the insertion of the input into an empty structure. The circuit family thus has $\mathcal{O}(f(2^n))$ many gates for inputs of length n. ◀

It is well-known that for computing the number of 1s modulo a prime p occurring in a bit string of length n, a Boolean circuit of depth ℓ requires $2^{\Omega(n^{1/2\ell})}$ gates (see [24] or, for instance, [13, Theorem 12.27] for a modern exposition).

▶ **Lemma 8.** *If a regular language L is not star-free, then it cannot be maintained in* $\mathsf{DynFO}(\leq, +, \times)$ *with work $\mathcal{O}(\log^d n)$ under changes of size $\mathcal{O}(\log n)$, for any $d \in \mathbb{N}$.*

Proof. Suppose that L is a regular language which is not star-free. Then its syntactic monoid M is not aperiodic. Therefore, there must be an $m \in M$ as well as $k, \hat{p} \in \mathbb{N}$ such that $m^k = m^{k+\hat{p}}$ and $m^{k+i} \neq m^k$ for all $1 \leq i < \hat{p}$, that is, from some point onward, multiplying m with itself becomes periodic with period $\hat{p}$.

Assume, towards a contradiction, that L can be maintained with work $\mathcal{O}(\log^d n)$ under changes of size $\mathcal{O}(\log n)$, for some $d \in \mathbb{N}$. Then, by Lemma 7, there is a constant-depth circuit of size $\mathcal{O}(f(2^n)) = \mathcal{O}(\log^d 2^n) = \mathcal{O}(n^d)$ for L, and therefore also a circuit family $\mathcal{C}$ of this size for the word problem for M.

Let p be the smallest prime factor of $\hat{p}$ and let $t = \frac{\hat{p}}{p}$. From the family $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$, we construct a constant-depth circuit family $\mathcal{D} = (D_n)_{n\in\mathbb{N}}$ of size n^d which computes the number of 1's in a 0-1-string modulo p. The idea is to use the period of the element $m \in M$ for simulating "modulo p". For an input of length n, the circuit D_n with input $a_1 \dots a_n$ simulates C_{tn+k} for the input $m_1 \dots m_{tn+k}$ defined via

$$m_i = \begin{cases} m, & \text{for } i \leq k \\ m, & \text{if } i > k \text{ and } a_{(i-k) \bmod n} = 1 \\ 1, & \text{if } i > k \text{ and } a_{(i-k) \bmod n} = 0 \end{cases} .$$

So, from an input with ℓ many 1s we get an input that contains m exactly $t\ell + k$ times. So, $m_1 \cdots m_{tn+k} = m^k$ if and only if $\ell \equiv 0 \pmod{p}$.

The existence of $\mathcal{D}$ contradicts Smolensky's lower bound [24]. ◀

We now show that star-free languages can be maintained with polylogarithmic work under polylogarithmic changes. In [18], this was shown for single tuple changes.

▶ **Lemma 9.** *One can maintain membership in star-free languages in* $\mathsf{DynFO}(\leq, +, \times)$ *with work* $\mathcal{O}(\log^d n)$ *under changes of size* $\log^c n$*, for all* $c \in \mathbb{N}$ *and some* $d \in \mathbb{N}$ *depending on* c*.*

Proof. The statement follows from a detailed analysis of the dynamic programs used in [18, Theorem 5.3] (see also the long version [19, Theorem B.4]), which is based on the approach used in [9].

We sketch the main ideas and necessary adaptations. Instead of maintaining membership of a star-free language L, we aim at maintaining range queries for L's syntactic monoid M. The dynamic problem $\textsc{Range}(M)$ for a monoid M has as input sequences $m_0 \cdots m_{n-1}$ of elements of M, allows for setting positions of this sequence to some element $m \in M \cup \{\epsilon\}$, and support queries of the form $\textsc{Range}(\ell, r)$ which return the product $m_\ell \circ \cdots \circ m_r$.

Syntactic monoids M of star-free languages have one of the following four forms [14]:

(a) $M = \{1\}$
(b) $M = \{1, \sigma, \dots, \sigma^k = \sigma^{k+1}\}$ for some k
(c) $\sigma\sigma' = \sigma$ for all $\sigma, \sigma' \in M - \{1\}$
(d) $M = V \cup T$ for submonoids $V, T \subsetneq M$ such that $\sigma_M \sigma_T \in T - \{1\}$ for all $\sigma_M \in M, \sigma_T \in T - \{1\}$

The idea in [19, Theorem B.4]) and [9] is to inductively construct dynamic programs for $\textsc{Range}(M)$ for each of the Cases (a)–(d). For Case (a) this is trivial. Cases (b) and (c) can be maintained using a single NextInSet instance: For case (b), one needs to find at most k non-neutral symbols. In case (c), the product depends on the first symbol differing from 1. This instance changes only at polylogarithmic many positions and can therefore be updated with polylogarithmic work due to Lemma 1.

Case (d) is more complicated. Suppose m is the input sequence of monoid elements. The dynamic program for $\textsc{Range}(M)$ relies on queries to the following data structures:

- A RANGE(V) instance for the sequence $v = v_0 \cdots v_{n-1}$, where $v_i = m_i$ if $m_i \in V$ and else $v_i = 1$
- A RANGE(T) instance for the sequence $t = t_0 \cdots t_{n-1}$, where $t_i = m_i$ if $m_i \in T - \{1\}$ and else $t_i = 1$.
- A NEXTINSET instance for the set of *switching positions* i where $m_i \in T - \{1\}$, but $m_{i+1} \notin T - \{1\}$.
- A RANGE(T) instance maintaining the sequence $u = u_0 \cdots u_{n-1}$, where u_i is the product $m_{j+1} \dots m_i$ of the elements between the last switching position j before i and i if i is a switching position, and 1 otherwise.

All four data structures can be maintained in polylogarithmic work under changes of polylogarithmic size: NEXTINSET due to Lemma 1 and both RANGE(V) and RANGE(T) by induction.

The dynamic program for RANGE(M) can answer queries by combining the answers of constantly many queries to the above data structures (see [19, Theorem B.4]) and [9]).

Next, we describe the strategy how the dynamic program for RANGE(M) handles updates of polylogarithmic size to the input sequence m. To this end the program:

(1) Updates the sequences v and t.

(2) Computes the new switching positions: Each changed position in m possibly causes a constant number of elements to change their switching status (depending on the update and its position in its old block, see [18]). For each such position i, check whether i is a switching position after the update by checking whether m_i, m_{i+1} are in $T - \{1\}$.

(3) Computes the new block product u_i for all switching positions of affected blocks.

(4) Updates the NEXTINSET instance for the switching positions, the RANGE(T) instances for t and u, and the RANGE(V) instance for v.

Steps (1) and (2) are straightforward and only update polylogarithmically many positions. Step (3) requires only polylogarithmically many work, as only polylogarithmically many positions changed their switching status and thus the blocks can be updated by polylogarithmically many queries to the (old) RANGE(T) instance for u, each requiring polylogarithmic work. Step (4) can be done in polylogarithmic work, since for each of the instances only changes of polylogarithmic size occurred and those can be handled with polylogarithmic work. ◀

This also concludes the proof of Theorem 6.

6 Expressive power of quantification over small sets

In this section, we re-visit the power of first-order logic on small substructures. In the previous sections, via Corollary 3 and its usage, we have seen that for maintaining properties with first-order update formulas under changes of polylogarithmic size, it is essential to know the expressive power of first-order logic on substructures of polylogarithmic size. Corollary 3 relies on the existence of constant-depth circuits of subexponential size for all LOGCFL problems, see Lemma 2.

We now take a different perspective and relate the expressive power of first-order logic on small substructures to the expressive power of second-order logics where second-order quantification is restricted to relations of small size. The results of this section provide an alternative proof of Lemma 2, but also give further insights into second-order logics with size-restricted quantification, which we think is interesting in its own right.

An element of a domain D of size n carries $\log n$ bits of information. Intuitively, assuming the presence of arithmetic relations, this means that in substructures over a subdomain $D' \subseteq D$ of size $\log n$, first-order quantifiers can quantify subsets: if, e.g., an existential quantifier selects a number $k \in \{0, \dots, n-1\}$, then the subset $A \subset D'$ is selected where the i-th element of D' is in A if the i-th bit of k in its binary encoding is 1.

Similarly, first-order quantifiers over D can quantify subsets of size $\sqrt[d]{|D'|}$ over subdomains $D' \subseteq D$ of polylogarithmic size $\log^c n$, for some $d > c$. The binary string of length $\log n$ that is obtained from an existentially quantified number $k \in \{0, \dots, n-1\}$ is split into $\frac{\log n}{c \log\log n}$ segments of length $c \log\log n$ each. The quantified subset $A \subset D'$ contains the i-th element of D' if one segment is the binary encoding of i. As $\frac{\log n}{c \log\log n} \in \Omega(\sqrt[d]{|D'|}) = \Omega(\sqrt[d]{\log^c n})$ for all $d \geq c+1$, the claim follows.

The quantification of relations with arity $k \geq 1$ can be simulated as well over subdomains of size $\log^c n$: a tuple of k elements can be encoded by a bit string of length $ck \log\log n$.

This gives rise to the following variants of second-order logic, where the size of quantified sets is restricted. For $r \in \mathbb{N}$, the subset $r\mathsf{SO}$ of second-order logic allows quantification over relations of arity at most r. For a fixed function $f : \mathbb{N} \to \mathbb{N}$, we define $f(n)$-$r\mathsf{SO}$ to be the logic that has the syntax of $r\mathsf{SO}$ but restricts quantification to relations with at most $f(n)$ many tuples, where n is the size of the underlying domain. For $r = 1$, this logic is called $f(n)$-MSO.

Of course, $f(n)$-$r\mathsf{SO}$ has the same expressive power as $r\mathsf{SO}$ if $f \in \Omega(n^r)$. We will now highlight that it is still very expressive when the size of quantified sets is bounded by functions $f \in o(n)$. Our main interest is in functions $f(n) = \sqrt[c]{n}$, for arbitrary fixed c, due to the above motivation.

In the rest of this section we discuss the power of $f(n)$-$r\mathsf{SO}$ (see Subsection 6.1) and in particular of $f(n)$-MSO (see Subsection 6.2).

6.1 The power of $f(n)$-SO

The restriction of SO to quantification over relations of size $\sqrt[c]{n}$ is very expressive, especially if we allow formulas to use built-in arithmetic relations.

There are complete problems from each level of the polynomial hierarchy contained in $\sqrt[c]{n}$-MSO for each $c \in \mathbb{N}$. For instance, a padded version of 3-colorability, where positive instances with m edges come with m^c isolated nodes (and instances without such isolated nodes are negative), is still NP-complete yet expressible in $\sqrt[c]{n}$-$\exists\mathsf{MSO}$.

While some of its problems are contained, it is unlikely that all of NP is contained in $\sqrt[c]{n}$-$r\mathsf{SO}$ for some $i > 1$ and any $r \in \mathbb{N}$, as this would contradict the exponential time hypothesis: 3-SAT cannot be solved in time $2^{o(n)}$ [12]. This is due to the fact that all $\sqrt[c]{n}$-$r\mathsf{SO}$-definable problems have (bounded-depth) circuits of subexponential size.

▶ **Proposition 10.** *If a problem L can be expressed by a $\sqrt[c]{n}$-$r\mathsf{SO}$ formula, for some $c, r \in \mathbb{N}$, then there is a $d \in \mathbb{N}$ such that L can be decided by a FO-uniform family of depth-d circuits of size $2^{\mathcal{O}(\log n \sqrt[c]{n})}$.*

Proof sketch. We use the naïve translation from formulas to circuits. Suppose there is an $\sqrt[c]{n}$-$r\mathsf{SO}$ formula φ that expresses L, for some natural numbers c and r. Let d be the depth of the syntax tree of φ. Given some n, from the syntax tree of φ we construct in the natural way a depth-d Boolean circuit C_n for input strings of length n as follows. Every existential quantifier is replaced by an "or" gate, with fan-in n for a first-order quantifier and with fan-in upper-bounded by $n^{r\sqrt[c]{n}}$ for an $\sqrt[c]{n}$-$r\mathsf{SO}$ quantifier. Analogously, universal quantifiers

are replaced by "and" gates with fan-in n or $n^{r\sqrt[c]{n}}$, respectively. The obtained circuit is a tree of depth d and of degree at most $n^{r\sqrt[c]{n}}$, so the number of gates is upper-bounded by $(n^{r\sqrt[c]{n}})^{d+1} = 2^{\mathcal{O}(\log n \sqrt[c]{n})}$. ◀

▶ **Corollary 11.** *If* 3-SAT $\in \sqrt[c]{n}$-$r\mathsf{SO}$ *for some* $i, r \in \mathbb{N}$ *with* $i > 1$*, then the exponential time hypothesis fails.*

Thus, NP can likely not be captured by $\sqrt[c]{n}$-rSO. This raises the question, which complexity classes can be captured.

The following theorem shows that $\sqrt[c]{n}$-rSO captures LOGCFL. In conjunction with Proposition 10, it gives an alternative proof for Lemma 2 (which is from [1, Lemma 8.1]).

▶ **Theorem 12.** *Let* Q *be a query that is computable in* LOGCFL*. There is an* $r \in \mathbb{N}$ *such that* Q *is expressible in* $\sqrt[c]{n}$-$r\mathsf{SO}(\leq, +, \times)$*, for every* $c \in \mathbb{N}$*.*

To prove this result, we show that there is an $r \in \mathbb{N}$ such that $\sqrt[c]{n}$-$r\mathsf{SO}$ contains a problem that is LOGCFL-complete under $\mathsf{FO}(\leq, +, \times)$ reductions, for every $c \in \mathbb{N}$. It follows that for every problem $L \in \mathsf{LOGCFL}$ there is a d-dimensional $\mathsf{FO}(\leq, +, \times)$ reduction to the complete problem, for some $d \in \mathbb{N}$, and therefore that L is in $\sqrt[c]{n}$-$(dr)\mathsf{SO}(\leq, +, \times)$ for every $c \in \mathbb{N}$.

The problem we consider is the word problem for groupoids. A (finite) *groupoid* $\mathcal{G} = (G, \circ, 1)$ consists of a finite set G and a binary operation $\circ\colon G \times G \to G$ on this set with identity 1. Note that $\circ$ does not need to be associative. The F *word problem* over a groupoid $\mathcal{G}$ for a fixed set $F \subseteq G$ asks: given a sequence $w_1, \ldots, w_n$ of elements of G, can one introduce parentheses such that $w_1 \circ \cdots \circ w_n$ evaluates to an element from F?

▶ **Lemma 13** ([2, Corollary 3.4])**.** *There is a groupoid* $\mathcal{G} = (G, \circ, 1)$ *and a set* $F \subseteq G$ *such that the* F *word problem over* $\mathcal{G}$ *is* LOGCFL*-complete under* $\mathsf{FO}(\leq, +, \times)$*-reductions.*

▶ **Proposition 14.** *Let* $\mathcal{G} = (G, \circ, 1)$ *be a groupoid and let* $F \subseteq G$ *be a set of groupoid elements. The* F *word problem over* $\mathcal{G}$ *is in* $\sqrt[c]{n}$-2SO*, for every* $c \in \mathbb{N}$*.*

Proof. Let $c \in \mathbb{N}$ be fixed and let $w_1, \ldots, w_n$ be the input sequence. We first explain how the problem can be solved for sub-sequences $w_\ell, \ldots, w_r$ of length $r - \ell + 1 = \sqrt[c]{n}$ of the input. Note that $\sqrt[c]{n}$ parentheses are sufficient to fully determine the evaluation order on this sub-sequence. A $\sqrt[c]{n}$-2SO formula φ_h^1 can existentially quantify $|G|$ binary relations P_g, for all $g \in G$, with the intention that if $(i_1, i_2) \in P_g$ then there is an evaluation order such that $w_{i_1} \circ \cdots \circ w_{i_2}$ evaluates to g, and this evaluation order is encoded by further tuples in the relations $(P_g)_{g \in G}$. A first-order formula can check whether these quantified relations encode a consistent evaluation that results in a fixed element $h \in G$.

For the following formulation of φ_h^1 we assume that the input sequence is encoded by sets R_g, for each $g \in G$. The construct $z + 1$ is an abbreviation for the successor of z which can be easily expressed in FO using the linear order on the positions. For ease of presentation, we assume that parentheses are also introduced for single positions, so the evaluation order for two positions w_1, w_2 is represented as $((w_1) \circ (w_2))$. Technically, then $2\sqrt[c]{n}$ parentheses are necessary which can be encoded by quantifying two relations of size $\sqrt[c]{n}$ for every $g \in G$. We ignore this detail.

$$\varphi_h^1(\ell, r) = \exists P_{g_1} \cdots \exists P_{g_{|G|}} \Big[P_h(\ell, r) \wedge \bigwedge_{g \in G} \forall x \forall y \Big[\big(x \geq \ell \wedge y \leq r \wedge P_g(x, y) \big) \rightarrow \Big[(x = y \wedge R_g(x)) \vee \exists z \bigvee_{g_1 \circ g_2 = g} (P_{g_1}(x, z) \wedge P_{g_2}(z + 1, y)) \Big] \Big] \Big]$$

This formula φ_h^1 for sequences of length $\sqrt[c]{n}$ can be lifted to sequences of length $\sqrt[c]{n^2}$, as the evaluation tree for a sequence of this length can be decomposed into $\mathcal{O}(\sqrt[c]{n})$ subtrees of size at most $\sqrt[c]{n}$: think of deleting $\mathcal{O}(\sqrt[c]{n})$ tree edges such that the tree decomposes into a forest with components of size at most $\sqrt[c]{n}$. For each of these components, the formulas φ_h^1 describe a possible evaluation. On the basis of this evaluation, another $\sqrt[c]{n}$-2SO formula can express the possible evaluation of the whole sequence.

More formally, the formula $\varphi_h^2(\ell, r)$ that expresses whether a subsequence $w_\ell \circ \cdots \circ w_r$ of length $r - \ell + 1 = \sqrt[c]{n^2}$ of the input can be evaluated to $h \in G$ is obtained from $\varphi_h^1(\ell, r)$ by replacing the subformula $(x = y \wedge R_g(x))$ by $\varphi_g^1(x, y)$ and the atoms $P_{g_1}(x, z)$ and $P_{g_2}(z+1, y)$ by $\varphi_{g_1}^1(x, z)$ and $\varphi_{g_2}^1(z+1, y)$, respectively. Repeating this step yields a formula $\varphi_h^c(\ell, r)$ that expresses whether the full sequence $w_1 \circ \cdots \circ w_r$ can be evaluated to h. The final formula expressing the problem only needs to check that $\varphi_f^c(1, n)$ holds for some $f \in F$. ◀

6.2 The power of $f(n)$-MSO

We have seen that $\sqrt[c]{n}$-rSO is quite powerful. It turns out that the same is true already for the case $r = 1$. As an example of the expressive power of $\sqrt[c]{n}$-MSO, we outline that it captures MSO on strings and trees for all $c \in \mathbb{N}$.

We first show that $\sqrt[c]{n}$-MSO can express graph reachability. Recall that reachability is expressible in full MSO by the formula

$$\varphi_{\text{REACH}}(s,t) \stackrel{\text{def}}{=} \forall X \Big(\big(X(s) \wedge \forall x \forall y \big(\big(X(x) \wedge E(x,y) \big) \to X(y) \big) \big) \to X(t) \Big)$$

which uses that a node t can be reached from a node s if and only if all sets of nodes that include s and are closed under outgoing edges also include t.

▶ **Theorem 15.** *Graph reachability can be expressed in $\sqrt[c]{n}$-MSO, for every $c \geq 1$.*

Proof. The idea is simple and has been used similarly in Savitch's Theorem, in the context of small-depth circuits for small distance connectivity (see [3]) and for trading time with alternations (see Nepomnjascii's Theorem and [1]): For determining whether there is a path from s to t of length up to n in $\sqrt[c]{n}$-MSO, for some $c \in \mathbb{N}$, recursively decompose such paths into $\sqrt[c]{n}$ pieces until only paths of length $\sqrt[c]{n}$ remain.

More precisely, we first construct a $\sqrt[c]{n}$-MSO formula that expresses reachability via paths of length at most $\sqrt[c]{n}$ in graphs G of size n. Such a formula $\varphi_{\sqrt[c]{n}-\text{REACH}}(s,t)$ basically states that there is a subset W of nodes (of size at most $\sqrt[c]{n}+1$) such that t is reachable from s in $G[W]$. As we can only quantify over sets of size at most $\sqrt[c]{n}$, the formula asserts that there is an edge from s to a node v such that t is reachable from v via a path of length at most $\sqrt[c]{n} - 1$:

$$\varphi_{\sqrt[c]{n}-\text{REACH}}(s,t) \stackrel{\text{def}}{=} s = t \vee \exists v \, \exists W \, \forall X \Big(E(s,v) \wedge W(v) \\ \wedge \big(X \subseteq W \wedge X(v) \wedge \forall x \forall y \big(\big(X(x) \wedge W(y) \wedge E(x,y) \big) \to X(y) \big) \big) \to X(t) \Big).$$

With a Savitch-like construction, this formula can be lifted to paths of greater length. A path of length $\sqrt[c]{n^2}$ can be decomposed into $\sqrt[c]{n}$ paths of length $\sqrt[c]{n}$. So, the $\sqrt[c]{n}$-MSO formula $\varphi_{\sqrt[c]{n^2}-\text{REACH}}(s,t)$ that we obtain from $\varphi_{\sqrt[c]{n}-\text{REACH}}(s,t)$ by replacing atoms $E(x,y)$ with $\varphi_{\sqrt[c]{n}-\text{REACH}}(x,y)$ expresses reachability along paths of length $\sqrt[c]{n^2}$. Repeating this step c times results in an $\sqrt[c]{n}$-MSO formula that expresses reachability along paths of length $\sqrt[c]{n^c} = n$, that is, a formula that expresses graph reachability. ◀

With similar decompositions as used in the proof of the previous theorem, it is easy to show that $\sqrt[c]{n}$-MSO is as expressive as MSO on strings and trees.

▶ **Theorem 16.** *The following queries can be expressed in $\sqrt[c]{n}$-MSO, for every $c \geq 1$:*

(a) *membership in any regular language,*

(b) *membership in any regular tree language.*

Proof. Fix some $c \geq 1$.

Towards (a), let L be some regular language over an alphabet Σ and let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton (DFA, with state set Q, transition function δ, initial state q_0 and set F of accepting states) that decides L. We outline how an $\sqrt[c]{n}$-MSO formula for L can be constructed.

The usual approach to express membership in L in MSO is to construct a formula $\psi^{p,q}(j,k)$ for any pair p, q of states of $\mathcal{A}$ such that $\psi^{p,q}(j,k)$ is satisfied on a string $w = w_1 \cdots w_n$ if $\mathcal{A}$ goes from state p to state q while reading the word $w_j \cdots w_k$, so, if $\delta^*(p, w_j \cdots w_k) = q$. This formula existentially quantifies for all states $q \in Q$ the set of positions from w such that $\mathcal{A}$ assumes the state q after reading this position. The formula then checks whether for all pairs of neighboring positions these choices are consistent with the transition function δ.

An almost identical $\sqrt[c]{n}$-MSO-formula $\psi^{p,q}_{\sqrt[c]{n}}(j,k)$ can check whether $\delta^*(p, w_{j+1} \cdots w_k) = q$ holds, as long as $w_{j+1} \cdots w_k$ is a substring of length at most $\sqrt[c]{n}$. Similar to the proof of Theorem 15, this formula can be lifted to substrings of larger size.

Towards (b), we follow a similar approach and construct formulas for larger and larger parts of the input tree. We introduce some notation to define subtrees of a tree more easily. Let $T = (V, E, r)$ be a rooted tree, $t \in V$ and $B \subseteq V$. The tree $T(t, B)$ is the subtree of T rooted at t, but all inner nodes of this subtree that are in B become leaves. More formally, let $T(t) = (V', E')$ be the subtree of T rooted at t, and let $B' \stackrel{\text{def}}{=} B \cap V'$. The tree $T(t, B)$ results from $T(t)$ by removing all children of B' and their subtrees. Like in the proof of Theorem 4(b), we assume without loss of generality that T is a binary tree.

Let L be a regular tree language over an alphabet Σ and let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a corresponding tree automaton. Suppose we are given a set B of tree nodes and a partition $B_{q_1}, \ldots, B_{q_\ell}$ of this set that associates a state $q_i \in Q$ to every node from B. Analogously to part (a), there is an $\sqrt[c]{n}$-MSO formula $\psi^q_{\sqrt[c]{n}}(t, B_{q_1}, \ldots, B_{q_\ell})$ that expresses for trees $T(t, B)$ of size $\sqrt[c]{n}$ that $\mathcal{A}$ assigns the state q to t in $T(t, B)$ if the initial states on the leaf nodes from B are as given by the partition. This formula existentially quantifies for each state from Q a set of nodes and verifies that $\mathcal{A}$ actually assigns these states to the corresponding nodes. By using constantly many copies of each quantifier, one can obtain an analogous formula for subtrees of size $\mathcal{O}(\sqrt[c]{n})$.

A formula $\psi^q_{\sqrt[c]{n^2}}(t, B_{q_1}, \ldots, B_{q_\ell})$ for subtrees $T(t, B)$ of size $\sqrt[c]{n^2}$ existentially quantifies a set B' of nodes as well as a partition into ℓ sets $B'_{q_1}, \ldots, B'_{q_\ell}$ with the intention that for each $t' \in B' \cup \{t\}$ the subtrees $T(t', B \cup B')$ have size at most $2\sqrt[c]{n}$. It then checks that for all $t' \in B' \cup \{t\}$ the formula $\psi^q_{\sqrt[c]{n}}(t', B_{q_1} \cup B'_{q_1}, \ldots, B_{q_\ell} \cup B'_{q_\ell})$ is satisfied.

We iterate this step and obtain a formula $\psi^q_n(t)$ that expresses whether $\mathcal{A}$ assigns the state q to t in $T(t)$, and the final formula for the word problem of L only needs to check whether $\psi^f_n(r)$ holds for any accepting state $f \in F$. ◀

As mentioned before, Theorem 16 shows that $\sqrt[c]{n}$-MSO and MSO have the same expressive power on words and trees, respectively, for each $c \in \mathbb{N}$. This is not true any more if the quantifiers are further restricted.

▶ **Proposition 17.** *For $f(n) \in n^{o(1)}$ there is no $f(n)$-*MSO *formula φ that expresses the membership in the language*

$$L_{\mathtt{aa}} = \{w \in \{a,b\}^* \mid w \text{ has an even number of } a\text{'s}\}$$

Proof. Fix $f(n) \in n^{o(1)}$. We assume that there is an $f(n)$-MSO formula φ that expresses $L_{\mathtt{aa}}$ and aim for a contradiction to Håstad's lower bound on the size of constant-depth circuits for Parity [11].

Analogously to the proof of Proposition 10, there is a d such that for any n we can obtain from φ a depth-d circuit C_n for input strings of length n. Gates of this circuit have degree at most $n^{f(n)}$, so the number of gates is upper-bounded by $(n^{f(n)})^{d+1} = 2^{\log n(d+1)f(n)}$ which for $f(n) \in n^{o(1)}$ is not in $2^{\Omega(n^{\frac{1}{d-1}})}$, the lower bound for depth-d circuits that compute a parity function on n input bits [11]. ◀

7 Discussion and future directions

In this paper we have seen that regular languages can be maintained under polylogarithmic changes. We also have seen that results on work-efficiency from [18] can be transferred from single-tuple changes to polylogarithmic changes for regular languages. We further discussed the power of first-order logic on small structures, as this is an essential tool in our proofs.

We highlight three open questions. While we have seen that regular languages cannot be maintained with polylogarithmic work under polylogarithmic changes (see Lemma 8), it remains open whether polylogarithmic work suffices for single tuple changes. Lower bounds in this case seem to require new techniques, as membership in regular languages can be maintained with polylogarithmic work under single tuple changes for change sequence up to polylogarithmic length. Thus, lower bounds have to exploit long change sequences, but unfortunately most known lower bound proof techniques in dynamic complexity theory are expected to be applicable only for constant-length change sequences.

▶ **Open question 1.** Can membership in regular languages be maintained in DynFO with polylogarithmic work under single tuple changes?

Apart from knowing that membership in context-free languages is in DynFO under single tuple changes [10, Theorem 4.1], our knowledge about dynamic membership for context-free languages is very limited.

▶ **Open question 2.** Can membership in all context-free languages be maintained under changes of non-constant size?

The dynamic program used in [10, Theorem 4.1] yields a naïve work bound $\mathcal{O}(n^7)$ for membership in context-free languages as it uses 4-ary auxiliary relations which are updated using three nested existential quantifiers. Using significantly less work than $\mathcal{O}(n^{\omega-1})$, where ω is the matrix multiplication exponent, is likely not possible, since the k-Clique conjecture fails if $\mathcal{O}(n^{\omega-1-\epsilon})$ work suffices for some $\epsilon > 0$ [18, Theorem 6.4].

▶ **Open question 3.** How much work is required for maintaining membership in context-free languages (under single tuple changes)?

References

1 Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael E. Saks. Minimizing disjunctive normal form formulas and AC^0 circuits given a truth table. *SIAM J. Comput.*, 38(1):63–84, 2008. doi:10.1137/060664537.

2 François Bédard, François Lemieux, and Pierre McKenzie. Extensions to Barrington's m-program model. *Theor. Comput. Sci.*, 107(1):31–61, 1993. doi:10.1016/0304-3975(93)90253-P.

3 Xi Chen, Igor Carboni Oliveira, Rocco A. Servedio, and Li-Yang Tan. Near-optimal small-depth lower bounds for small distance connectivity. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 612–625. ACM, 2016. doi:10.1145/2897518.2897534.

4 Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in DynFO. *J. ACM*, 65(5):33:1–33:24, 2018. doi:10.1145/3212685.

5 Samir Datta, Pankaj Kumar, Anish Mukherjee, Anuj Tawari, Nils Vortmeier, and Thomas Zeume. Dynamic complexity of reachability: How many changes can we handle? In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 122:1–122:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.122.

6 Samir Datta, Anish Mukherjee, Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. A strategy for dynamic programs: Start over and muddle through. *Logical Methods in Computer Science*, 15(2), 2019. doi:10.23638/LMCS-15(2:12)2019.

7 Samir Datta, Anish Mukherjee, Nils Vortmeier, and Thomas Zeume. Reachability and distances under multiple changes. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 120:1–120:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.120.

8 Guozhu Dong and Jianwen Su. First-order incremental evaluation of datalog queries. In *Database Programming Languages (DBPL-4), Proceedings of the Fourth International Workshop on Database Programming Languages – Object Models and Languages*, pages 295–308, 1993.

9 Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *J. ACM*, 44(2):257–271, 1997. doi:10.1145/256303.256309.

10 Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. The dynamic complexity of formal languages. *ACM Trans. Comput. Log.*, 13(3):19:1–19:36, 2012. doi:10.1145/2287718.2287719.

11 Johan Håstad. Almost optimal lower bounds for small depth circuits. In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 6–20. ACM, 1986. doi:10.1145/12130.12132.

12 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

13 Stasys Jukna. *Boolean function complexity: advances and frontiers*, volume 27. Springer Science & Business Media, 2012. doi:10.1007/978-3-642-24508-4.

14 Kenneth Krohn and John Rhodes. Algebraic theory of machines. i. prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116:450–464, 1965. doi:10.1016/S0022-0000(67)80007-2.

15 Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004. doi:10.1007/978-3-662-07003-1.

16 Leonid Libkin. Logics for unranked trees: An overview. *Log. Methods Comput. Sci.*, 2(3), 2006. doi:10.2168/LMCS-2(3:2)2006.

17 Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. *J. Comput. Syst. Sci.*, 55(2):199–209, 1997. doi:10.1006/jcss.1997.1520.

18 Jonas Schmidt, Thomas Schwentick, Till Tantau, Nils Vortmeier, and Thomas Zeume. Work-sensitive dynamic complexity of formal languages. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures – 24th International Conference, FOSSACS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 – April 1, 2021, Proceedings*, volume 12650 of *Lecture Notes in Computer Science*, pages 490–509. Springer, 2021. doi:10.1007/978-3-030-71995-1_25.

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

20 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Inf. Control.*, 8(2):190–194, 1965. doi:10.1016/S0019-9958(65)90108-7.

21 Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. Dynamic complexity under definable changes. In Michael Benedikt and Giorgio Orsi, editors, *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*, volume 68 of *LIPIcs*, pages 19:1–19:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICDT.2017.19.

22 Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. Dynamic complexity under definable changes. *ACM Trans. Database Syst.*, 43(3):12:1–12:38, 2018. doi:10.1145/3241040.

23 Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. Sketches of dynamic complexity. *SIGMOD Rec.*, 49(2):18–29, 2020. doi:10.1145/3442322.3442325.

24 Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 77–82, 1987. doi:10.1145/28395.28404.

25 Larry Stockmeyer and Uzi Vishkin. Simulation of parallel random access machines by circuits. *SIAM Journal on Computing*, 13(2):409–422, 1984. doi:10.1137/0213027.

26 Nils Vortmeier. *Dynamic expressibility under complex changes*. PhD thesis, TU Dortmund University, Germany, 2019. doi:10.17877/DE290R-20434.