Dynamic Complexity under Definable Changes*

Thomas Schwentick[1], Nils Vortmeier[2], and Thomas Zeume[3]

1 **TU Dortmund University, Dortmund, Germany**
thomas.schwentick@tu-dortmund.de
2 **TU Dortmund University, Dortmund, Germany**
nils.vortmeier@tu-dortmund.de
3 **TU Dortmund University, Dortmund, Germany**
thomas.zeume@tu-dortmund.de

Abstract

This paper studies dynamic complexity under definable change operations in the DynFO framework by Patnaik and Immerman. It is shown that for changes definable by parameter-free first-order formulas, all (uniform) AC^1 queries can be maintained by first-order dynamic programs. Furthermore, many maintenance results for single-tuple changes are extended to more powerful change operations: (1) The reachability query for undirected graphs is first-order maintainable under single tuple changes and first-order defined insertions, likewise the reachability query for directed acyclic graphs under quantifier-free insertions. (2) Context-free languages are first-order maintainable under Σ_1-defined changes. These results are complemented by several inexpressibility results, for example, that the reachability query cannot be maintained by quantifier-free programs under definable, quantifier-free deletions.

1998 ACM Subject Classification F.4.1. Mathematical Logic

Keywords and phrases Dynamic descriptive complexity, SQL updates, dynamic programs

Digital Object Identifier 10.4230/LIPIcs.ICDT.2017.19

1 Introduction

In the setting of *Dynamic Complexity*, a database $\mathcal{D}$ is being changed and an update program $\mathcal{P}$ tries to answer a standing query q after each change. The program usually consists of logical formulas which can make use of additional, *auxiliary* relations which in turn need to be updated after each change. Dynamic Complexity can be seen as a logic-based counterpart of *Dynamic Algorithms*, where algorithms use auxiliary data structures to keep track of properties of structures like graphs under change operations. The Dynamic Complexity framework was introduced in [23] and a similar framework, FOIES, in [8].

In Dynamic Complexity, one usually allows first-order logic formulas as update mechanism for the auxiliary relations. This is in line with the database-oriented framework, since first-order logic correspond to database languages like relational algebra. Just as in Dynamic Algorithms, for most investigations the possible change operations are limited to insertions and deletions of single tuples. The class of queries maintainable in this fashion is called DynFO. This line of research has seen recent progress, particular with respect to the question whether the reachability query can be maintained in DynFO for directed graphs [2, 3].

* The authors acknowledge the financial support by DFG grant SCHW 678/6-2. The third author thanks the Simons Institute for the Theory of Computing for hosting him and providing excellent research conditions.

Although the restriction to single-tuple changes can be justified by the need to concentrate on the basic phenomena of dynamic maintainability of queries, it is also clear that from a more practical perspective one would be interested in more complex change operations at a time. One approach is to specify changes by "Δ-relations", e.g., by sets of tuples to be inserted or deleted. This is basically the viewpoint of *Incremental View Maintenance* (see for example [15]). However, it is clear that arbitrary Δ-relations can make the auxiliary relations useless.

In this work, we consider a different extension of the single-tuple-change paradigm that is inspired by SQL update queries (for a theoretical view at SQL updates we refer to [1]). We model such queries by *replacement queries* which can modify several relations at a time by first-order formulas that can use tuples of elements as parameters. Similar but slightly weaker frameworks were introduced in [17, 28], but these papers did not study maintainability under such complex changes.

Contributions. The generalized setting yields a huge range of research questions, e.g., all previously studied questions in Dynamic Complexity in combination with replacement queries of varying expressiveness, and this paper can only start to investigate a few of them.

We are mainly interested in positive results. In Section 4 we study first-order definable *insertion* queries (supplementing the single tuple changes). It turns out that the reachability query can still be maintained in DynFO for undirected graphs under first-order definable insertions (Theorem 3) and for directed acyclic graphs under quantifier-free insertions (Theorem 5). In Section 5, we investigate parameter-free replacement queries. We show that *all* queries that can be expressed in uniform AC^1 (and thus all queries that can be computed with logarithmic space) can be maintained in DynFO under first-order definable parameter-free replacement queries (Theorem 7). In Section 6, we show that many maintainability results for formal languages [23, 13] carry over to quantifier-free or Σ_1-definable replacement queries (Theorems 8 and 9).

It is notoriously difficult to prove inexpressibility results in Dynamic Complexity. One would expect that allowing more general change operations simplifies such results. In Section 7, we confirm this intuition to some extent and present cases where general replacement queries disable certain kinds of update programs to maintain queries that are maintainable under single-tuple changes.

Some proofs are omitted due to space constraints and can be found in the full version of this paper [24].

Related work. In addition to the related work mentioned already above, several other prior results for Dynamic Complexity under more general changes have been obtained. The reachability query for directed graphs has been studied under deletions of sets of edges and nodes that form an anti-chain in [5] and under insertions of sets of tuples that are cartesian-closed in [8]. Hesse observed that the maintenance procedure for this query under single tuple changes from [3] can deal with the replacement of the set of outgoing edges of a node (or, alternatively, the set of incoming edges). Edge contractions have been studied in [26]. Koch considered more general sets of changes in [19], though only for non-recursive queries.

Implementations of work on Dynamic Complexity are reported in [22] and [19].

2 Preliminaries

As much of the original motivation for the investigation of dynamic complexity came from incremental view maintenance (cf. [9, 6, 23]), it is common to consider logical structures as relational databases and to use notation from relational databases.

A *(relational) schema* τ consists of a set τ_{rel} of relation symbols, accompanied by an arity function $\text{Ar} : \tau_{\text{rel}} \to \mathbb{N}$, and a set τ_{const} of constant symbols. In this work, a *domain* is a finite set. A *database* $\mathcal{D}$ over schema τ with domain D assigns to every relation symbol $R \in \tau_{\text{rel}}$ a relation of arity $\text{Ar}(R)$ over D and to every constant symbol $c \in \tau_{\text{const}}$ an element (called *constant*) from D. A τ*-structure* $\mathcal{S}$ is a pair $(D, \mathcal{D})$ where D is a domain and $\mathcal{D}$ is a database with domain D over schema τ. By $\text{dom}(\mathcal{S})$ we refer to D. For a relation symbol $R \in \tau$ and a constant symbol $c \in \tau$ we denote by $R^{\mathcal{S}}$ and $c^{\mathcal{S}}$ the relation and constant, respectively, that are assigned to those symbols in $\mathcal{S}$. A k*-ary query* q on τ-structures is a mapping that assigns a subset of D^k to every τ-structure over domain D and is closed under isomorphisms.

We represent graphs as structures over a schema that contains a single binary relation E. The *reachability query* q_{Reach} maps graphs to their transitive closure relation.

In Section 6 we consider databases that represent words over some alphabet Σ. In a nutshell, the positions of a word correspond to elements of the domain and the letters at positions are indicated by unary relations. More formally, words are represented by databases with an immutable linear order on their domain and one unary relation R_σ for every $\sigma \in \Sigma$. For simplicity, we always assume that the domain of such a database is of the form $\{1, \dots, n\}$ and the linear order ist just the natural order. At any point in time, an element of the domain is allowed to be in at most[1] one relation R_σ. However, elements need not to be in any relation R_σ and, in this case, they do not correspond to a position with a symbol but rather to the empty word ϵ. Thus, we first associate with every position i an element $w_i \in \Sigma_\epsilon$, where by Σ_ϵ we denote the set $\Sigma \cup \{\epsilon\}$, and say that the database represents the string $w = w_1 \cdots w_n$. As an example, the database with domain $\{1, 2, 3, 4, 5\}$ and $R_a = \{2, 4\}, R_b = \{1\}$ represents the string *baa*. As a further convenience, we assume that databases have constants min and max that represent the smallest and the largest element, 1 and n, respectively.[2] We will not allow the linear order, min or max to be modified by change operations. In this paper, we will rarely distinguish between a database and the string it represents.

We use several notions from finite model theory (see, e.g., [20]). By $\text{qd}(\varphi)$, we denote the *quantifier-rank* of a first-order formula φ, that is, its maximum nesting depth of quantifiers. We denote the set of *rank-k types* of tuples of arity ℓ by $\text{FO}[k, \ell]$ (cf. [20, Definition 3.14]). The existential fragment of first-order logic is denoted by Σ_1.

3 Dynamic Programs with Complex Changes

In this section we lift the definitions from [25] to more general change operations. We first define (general) change operations, then we adapt the definition of dynamic programs presented in [25] to those more complex changes.

[1] There are ways to get rid of this requirement, but we keep it for simplicity.
[2] This assumption can be avoided by, e.g., using additional prefix and suffix relations in the proof of Theorem 8, in the spirit of [13].

Change Operations. The change operations that we consider in this paper are based on queries. In their most general form, they can modify a given database over schema τ by replacing some of its relations with the results of first-order-defined queries on the database. These queries are allowed to use parameters.

To this end, a *replacement rule* ρ_R for relation R is of the form **replace** R **by** $\mu_R(\bar{p};\bar{x})$. Here, R is a relation symbol and $\mu_R(\bar{p};\bar{x})$ is a first-order formula over τ, where the tuple $\bar{x}$ has the same arity as R and $\bar{p}$ is another tuple of variables, called the *parameter tuple*. A *replacement query* $\rho(\bar{p})$ is a set of replacement rules for distinct relations with the same parameter tuple $\bar{p}$. In the case of replacement queries ρ that consist of a single replacement rule, we usually do not distinguish between ρ and its single replacement formula μ_R.

For a database $\mathcal{D}$, a change operation $\delta = (\rho, \bar{a})$ consists of a replacement query and a tuple of elements of (the domain of) $\mathcal{D}$ with the same arity as the parameter tuple of ρ. We often use the more concise notation $\rho(\bar{a})$ and refer to change operations simply as *changes*.

The result $\delta(\mathcal{D})$ of an application of a change operation $\delta = (\rho, \bar{a})$ to a database $\mathcal{D}$ is defined in a straightforward way: each relation R in $\mathcal{D}$, for which there is a replacement rule ρ_R in ρ, is replaced by the relation resulting from evaluating μ_R, that is, by $\{\bar{b} \mid \mathcal{D} \models \mu_R(\bar{a};\bar{b})\}$.

If a replacement query has no parameters we say that it is *parameter-free*.

▶ **Example 1.**

(a) As a first example, we consider directed graph structures, that is, structures with a single binary relation E. Let, for some graph G, $\delta_1 = (\rho_1, u)$ be the change operation with replacement query $\mu_E(p;x,y) = E(x,y) \vee (x = p)$ and node u. Then, in $\delta_1(G)$, there is an edge from u to every node of G.

(b) We recall that words over the alphabet $\Sigma = \{a,b,c\}$ are represented by databases with a linear order on their domain and one unary relation R_σ for every $\sigma \in \Sigma$. Let $\mathcal{D}$ be a database representing a word w and i an element of $\mathcal{D}$. Let $\delta_2 = (\rho_2, i)$ be a change operation, where the replacement query ρ_2 consists of the rules **replace** R_a **by** $\mu_{R_a}(p;x)$ and **replace** R_b **by** $\mu_{R_b}(p;x)$ with $\mu_{R_a}(p;x) = \big((x<p) \wedge R_b(x)\big) \vee \big(\neg(x<p) \wedge R_a(x)\big)$ and $\mu_{R_b}(p;x) = \big((x<p) \wedge R_a(x)\big) \vee \big(\neg(x<p) \wedge R_b(x)\big)$. Then, $\delta_2(\mathcal{D})$ represents the word obtained from w by swapping a and b symbols on all positions before i and leaving all other positions unchanged.

Some of our investigations will focus on (syntactically) restricted replacement queries that either only remove or only insert tuples to relations. For an *insertion rule* ρ_R, the replacement formula $\mu_R(\bar{p};\bar{x})$ has the form $R(\bar{x}) \vee \varphi_R$. Similarly, *deletion rules* have replacement formulas $\mu_R(\bar{p};\bar{x})$ of the form $R(\bar{x}) \wedge \varphi_R$. In [1], the change operations *replace*, *insert*, *delete* and *modify* have been studied, in particular with respect to their expressive power. These operations are captured by our change operations[3].

Another syntactic restriction to be studied extensively in this work are the quantifier-free replacement queries, that allow only quantifier-free change formulas to be used. A special case of quantifier-free changes are the *single tuple changes*. We refer by **insert** $\bar{p}$ **into** R to the insertion query **replace** R **by** $\mu_R(\bar{p};\bar{x})$, where $\mu_E(\bar{p};\bar{x}) = R(\bar{x}) \vee (\bar{p} = \bar{x})$ and by **delete** $\bar{p}$ **from** R to the deletion query **replace** R **by** $\mu_R(\bar{p};\bar{x})$, where $\mu_R(\bar{p};\bar{x}) = R(\bar{x}) \wedge \neg(\bar{p} = \bar{x})$. As mentioned before, single tuple changes are the best studied change operations in previous work on dynamic complexity. To emphasize the difference we sometimes refer to arbitrary (not single-tuple) change operations as *complex changes*. For any schema τ we denote by Δ_τ the

[3] In [1] the domain of the database can be infinite.

set of single-tuple replacement queries for the relations (with symbols) in τ. In the case of graphs, we simply write Δ_E. In case of strings over some alphabet Σ, we write Δ_Σ.

Dynamic Programs. We now introduce dynamic programs, closely following the exposition in [25]. Inputs in dynamic complexity are represented as relational structures as defined in Section 2. The domain is fixed from the beginning, but the database in the initial structure is empty. This initially empty structure is then modified by a sequence of change operations.

The goal of a dynamic program is to answer a given query for the database that results from any change sequence. To this end, the program can use an auxiliary data structure represented by an auxiliary database over the same domain. Depending on the exact setting, the auxiliary database might be initially empty or not.

A dynamic program $\mathcal{P}$ operates on an *input database* $\mathcal{I}$ over a schema τ_{in} and updates an *auxiliary database* $\mathcal{A}$ over a schema[4] τ_{aux}, both sharing the same domain D which is fixed during a computation. We call $(D, \mathcal{I}, \mathcal{A})$ a *state* and consider it as one relational structure. The relations of $\mathcal{I}$ and $\mathcal{A}$ are called *input and auxiliary relations*, respectively.

A *dynamic program* has a set of update rules that specify how auxiliary relations are updated after a change. An *update rule* for updating an auxiliary relation T after a replacement query $\rho(\bar{p})$ is of the form **on change** $\rho(\bar{p})$ **update** $T(\bar{x})$ **as** $\varphi_T(\bar{p}, \bar{x})$ where the *update formula* φ_T is over $\tau_{\text{in}} \cup \tau_{\text{aux}}$.

The semantics of a dynamic program is as follows. When a change operation $\delta = \rho(\bar{a})$ is applied to the input database $\mathcal{I}$, then the new state $\mathcal{S}$ of $\mathcal{P}$ is obtained by replacing the input database by $\delta(\mathcal{I})$ and by defining each auxiliary relation T via $T \stackrel{\text{def}}{=} \{\bar{b} \mid (\mathcal{I}, \mathcal{A}) \models \varphi_T(\bar{a}, \bar{b})\}$. For a change operation δ we denote the updated state by $\mathcal{P}_\delta(\mathcal{S})$. For a sequence $\alpha = (\delta_1, \ldots, \delta_k)$ we write $\mathcal{P}_\alpha(\mathcal{S})$ for the state obtained after successively applying $\delta_1, \ldots, \delta_k$ to $\mathcal{S}$.

A *dynamic query* is a tuple (q, Δ) where q is a query over schema τ_{in} and Δ is a set of replacement queries. The dynamic program $\mathcal{P}$ *maintains* a dynamic query (q, Δ) with k-ary q if it has a k-ary auxiliary relation Q that, after each change sequence over Δ, contains the result of q on the current input database. More precisely, for each non-empty[5] sequence α of changes and each empty input structure $\mathcal{I}_\emptyset$, relation Q in $\mathcal{P}_\alpha(\mathcal{S}_\emptyset)$ and $q(\alpha(\mathcal{I}_\emptyset))$ coincide. Here, $\mathcal{S}_\emptyset = (\mathcal{I}_\emptyset, \mathcal{A}_\emptyset)$, where $\mathcal{A}_\emptyset$ denotes the empty auxiliary structure over the domain of $\mathcal{I}_\emptyset$.

The class of dynamic queries (q, Δ) that can be maintained by a dynamic program with update formulas from first-order logic is called DynFO. We also say that the query q can be maintained in DynFO under change operations Δ. The class of dynamic queries maintainable by quantifier-free update formulas is called DynProp.

The following very simple example shows how the transitive closure of a directed graph subject to single edge insertions can be maintained in this set-up.

▶ **Example 2.** Let q_{Reach} be the reachability query that returns all pairs (u, v) of a graph, for which there is a path from u to v. The dynamic query $(q_{\text{Reach}}, \{\textbf{insert } \bar{p} \textbf{ into } E\})$ can be maintained by a dynamic program that uses one auxiliary relation T, which always contains the transitive closure of the edge relation E. Its only update rule is given by the formula $\varphi_T(p_1, p_2; x, y) = T(x, y) \vee \big(T(x, p_1) \wedge T(p_2, y)\big)$. ◀

[4] To simplify the exposition, we will usually not mention schemas explicitly and always assume that all structures we talk about are compatible with respect to the schemas at hand.

[5] This technical restriction ensures that we can handle, e.g., Boolean queries with a yes-result on empty structures without initialization of the auxiliary relations. Alternatively, one could use an extra formula to compute the query result from the auxiliary (and input) structure.

Our general framework follows [23] and thus does not allow inserting new elements into or removing existing elements from the domain as in the FOIES framework [8]. The step from Dynamic Complexity to FOIES can be done by adding two more change operations, add(x) and remove(x). Our results of Section 4 easily carry over, and those of Section 6 carry over if, say, new elements are always added at the end of the string. Since add(x) and remove(x) have parameters, they do not quite fit into the parameter-free framework of Section 5. However, Theorem 7 survives if parameter-free remove queries are allowed.

Complex Change Operations and Initialization of Dynamic Programs. In the presence of complex replacement queries, the initialization of the auxiliary relations requires some attention. In the original setting of Patnaik and Immerman, the input database is empty at the beginning, and the auxiliary relations are initialized by first-order formulas evaluated on this (empty) initial input database. Since tuples can be inserted only one-by-one, the auxiliary relations can be adapted slowly and it can be ensured that, e.g., always a linear order [23] or arithmetic [10] on the active domain is available.

For complex changes, the situation is more challenging for a dynamic program: as an example, in the setting of strings, the first change could insert all positions of the domain into relation R_a and thus let the database represent the word a^n, if n is the size of the underlying domain. To enable the dynamic program to answer whether the string is in some language after this change, it needs some suitable (often: non-empty) initial values of the auxiliary relations. Since in this paper, we are mainly interested in the maintenance of queries and not so much in the specific complexity of the initialization, we do not define variants of DynFO with different power of initialization, but rather follow a pragmatic approach: whenever initialization is required, we say that the query can be maintained *with suitable initialization* and specify in the context what is actually needed. In all cases, it is easy to see that the initialization of the auxiliary relations can be computed in polynomial time.

An alternative approach would be to restrict the semantics of replacement queries to elements of the active domain of the current database and to allow the activation of elements only via tuple insertions.

4 Reachability and Definable Insertions

In this section, we study the impact of first-order definable complex change operations on the (binary) reachability query q_{Reach}. We present positive cases, where previous maintainability results survive under such stronger change operations. Negative results, where such operations destroy previous maintainability results, are given in Section 7.

In the classical DynFO setting with single-tuple change operations it was shown early on that q_{Reach} can be maintained in DynFO for two important graph classes: undirected graphs and directed, acyclic graphs (dags). It turns out that these results still hold in the presence of complex *insertions*: first-order insertions for undirected graphs and quantifier-free insertions for dags. In fact, in both cases basically the same auxiliary relations can be used as in the case of single-tuple changes.

We first show that for undirected graphs, the reachability query can be maintained in DynFO, for first-order insertions and the set Δ_E of single-edge insertions and deletions. We follow the convention from [14] that modifications for undirected graphs are symmetric in the sense that if an edge (a, b) is inserted then so is the edge (b, a) (and likewise for deletions).

▶ **Theorem 3.** *Let Δ be a finite set of first-order insertion queries. Then $(q_{Reach}, \Delta \cup \Delta_E)$ can be maintained in* DynFO *for undirected graphs.*

We use the approach for maintaining q_{Reach} for undirected graphs under single-edge insertions and deletions from [23, Theorem 4.1] and maintain a spanning forest and (essentially) its transitive closure relation. The crucial observation for extending this approach to first-order insertions is that, after such an insertion, between each pair of nodes in a (new) connected component, there is a connecting path that uses only a bounded number of newly inserted edges. This allows the update of the spanning forest and its transitive closure in a first-order definable way.

The observation is stated more precisely next. For two connected nodes u, v in a graph $G' = \delta(G)$ that is obtained from a graph G by an insertion δ, we define[6] the *bridge distance* $\text{bd}(u, v)$ as the minimal number d, such that there is a path from u to v in G' that uses d edges that were newly inserted by δ.

▶ **Lemma 4.** *For each first-order insertion query ρ there is a constant $m \in \mathbb{N}$ such that for each undirected graph G, each change $\delta = \rho(\bar{a})$ and all nodes u and v of G that are connected in $\delta(G)$ it holds $bd(u, v) \leq m$.*

We informally refer to this property as the *bridge boundedness property*.

Proof. The proof makes use of the result by Feferman-Vaught that the depth k first-order type of the disjoint union of two structures is determined by the depth k first-order types of these two structures [11, 12] (see also [21]).

Let $\mu(\bar{p}; \bar{x})$ be the first-order formula underlying ρ and k its quantifier-rank. Let ℓ be the arity of $\bar{p}$, m' the number of $\text{FO}[k, 1]$-types of undirected graphs and $m = \ell + m'$.

Let G be an undirected graph and let $\delta = \rho(\bar{a})$ for some tuple $\bar{a}$ of nodes of G. Let u, v be two nodes that are connected by some path π of the form $u = w_0, w_1, \ldots, w_r = v$ in $\delta(G)$ with q bridges, that is, edges that are not in G. Our goal is to show that there exists such a path with at most m bridges. Thus, if $q \leq m$, there is nothing to prove, so we assume $q > m$. It suffices to show that there is a path from u to v with fewer than q bridges. Let $(u_1, v_1), \ldots, (u_q, v_q)$ be the bridges in π. If for some i, the nodes u_i and v_i are in the same connected component of G (before the application of δ), we can replace the bridge (u_i, v_i) by a path of "old" edges resulting in an overall path with $q - 1$ bridges. Similarly, if u_i and u_j are in the same connected component of G, for some $i < j$, we can shortcut π by a path from u_i to u_j inside G. Therefore, we can assume that, for every i, the nodes u_i and v_i are in different connected components of G, and likewise u_i and u_j for $i < j$.

We show that in this case there are i, j with $i < j$ such that μ defines an edge between u_i and v_j, and therefore a path with fewer bridges can be constructed by shortcutting the path π with the edge (u_i, v_j). By the choice of m there must be two nodes u_i and u_j, with $i < j$, in distinct connected components of G that do not contain any element from $\bar{a}$, such that u_i and u_j have the same $\text{FO}[k, 1]$-type in their respective connected components. By Feferman-Vaught, it follows that $(u_i, v_j, \bar{a})$ and $(u_j, v_j, \bar{a})$ have the same $\text{FO}[k, \ell + 2]$-types and therefore, since μ defines an edge between u_j and v_j, it also defines one between u_i and v_j. ◀

Proof. Proof of Theorem 3 The dynamic program presented in [23, Theorem 4.1] maintains the transitive closure of undirected graphs under single-edge changes with the help of auxiliary relations F and PV. The binary relation F is a spanning forest of the input graph G and $(u, v, w) \in PV$ means that w is a node in the path from u to v in F. Observe that two nodes u and v are connected in an undirected graph if and only if $(u, u, v) \in PV$ holds.

[6] Since G and δ will be always clear from the context, we do not add them as parameters to this notation.

We show how to maintain the relation F and PV under FO insertions. For the moment we assume a predefined linear order $\leq$ on the domain to be present. Let ρ be an insertion query and m the bound on the number of bridges by Lemma 4. Let G be an undirected graph and $\delta = \rho(\bar{a})$ an insertion, F a spanning forest of G and PV as described above. We show how to FO-define the auxiliary relations F' and PV' for the modified graph $G' = \delta(G)$.

We first describe a strategy to define F' and then argue that it can be implemented by a first-order formula. Let C' be a (new) connected component in $\delta(G)$. We call the smallest node of C' with respect to $\leq$ the *queen* u_0 of C'. For each connected component C of G that is a subgraph of C', we define its *queen level* as the (unique) number $\text{bd}(u, u_0)$, for nodes $u \in C$. A bridge in C' is inserted into F' if for a connected component C of G of some level i it is the lexicographically smallest edge with respect to $\leq$ that connects C with some component of level $i-1$. This clearly defines a spanning forest. The chosen edges can be defined by a first-order formula because, for each number h, there are formulas $\theta_h(x, y)$ expressing that $\text{bd}(x, y) \leq h$.

Since the construction of F' ensures that each path in F' from a node to the queen of its connected component only contains at most m new edges, and thus each path in F' contains at most $2m$ new edges, it is straightforward to extend the update formula for PV' from [23, Theorem 4.1].

It remains to show how the assumption of a predefined linear order can be eliminated. For a change sequence α, we denote by A_α the set of parameters used in α. When applying α to an initially empty graph, a linear order on A_α can be easily constructed as in the case of single-tuple changes [10]. The remaining nodes in $V \setminus A_\alpha$ behave very similarly. More precisely, one can show by induction on $|\alpha|$, that for all nodes $a \in V$ and $b, b' \in V - A_\alpha$ it holds $(a, b) \in E \Leftrightarrow (a, b') \in E$ (and likewise for (b, a) and (b', a)).

The dynamic program for maintaining q_{Reach} for undirected graphs now maintains the relations F and PV as described above, yet restricted to the induced (and ordered) subgraph of G on A_α. The transitive closure can be defined from those relations and the edge relation by a simple case distinction.

- Two nodes $a, a' \in A_\alpha$ are connected by a path if and only if there is a path from a to a' in F or if there are nodes $b, b' \in A_\alpha$ and a node $c \in V \setminus A_\alpha$ such that there are paths from a to b and from a' to b' in F as well as edges (b, c) and (b', c).
- Two nodes $a \in A_\alpha$ and $b \in V \setminus A$ are connected by a path if and only if there is a node $a' \in A_\alpha$ such that there is a path from a to a' in F and an edge (a', b).
- Finally, two nodes $a, a' \in V \setminus A_\alpha$ are connected if and only if there is an edge (a, a') or there is an edge (a, b) for some $b \in A_\alpha$ (and therefore also an edge (a', b)). ◀

Now we turn to the other graph class, acyclic graphs, for which DYNFO maintainability under complex insertions (and single-edge deletions) is preserved; albeit (we are able to show that) only for quantifier-free insertions. In [23, Theorem 4.2], edge insertions are only allowed if they do not add cycles. Of course, given the transitive closure of the current edge relation it can be easily checked by a first-order formula (a *guard*), whether a new edge closes a cycle. We will see that this is also possible for the complex insertions we consider.

► **Theorem 5.** *Let Δ be a finite set of quantifier-free insertion queries. Then $(q_{Reach}, \Delta \cup \Delta_E)$ can be maintained in* DYNFO *for directed, acyclic graphs. Furthermore, for each quantifier-free insertion, there is a first-order guard which checks whether the insertion destroys the acyclicity of the graph.*

As in the case of undirected graphs, the proof relies on a bridge boundedness property. This property allows extending the technique for maintaining the transitive closure relation of acyclic graphs under single tuple changes used in [23] and [7] to quantifier-free insertions. As in [23] and [7] no further auxiliary relations besides the transitive closure relation are needed. In Section 7 we show that the transitive closure relation does not suffice for maintaining q_{Reach} for acyclic graphs subjected to Σ_1-definable insertions[7].

In the following lemma, the bridge distance bd is defined just as above. However, we can no longer assume that bridges connect (formerly) different connected components, therefore the lemma only holds for quantifier-free insertions. The proof can be found in the full version of this paper.

▶ **Lemma 6.** *For each quantifier-free insertion query ρ there is a constant $m \in \mathbb{N}$ such that for each directed, acyclic graph G and each change $\delta = \rho(\bar{a})$ it holds that $\delta(G)$ has a cycle with at most m bridges, or for all nodes u and v of G with a path from u to v in $\delta(G)$, it holds $bd(u, v) \leq m$.*

Proof. Proof of Theorem 5 In [23, Theorem 4.2] and [7, Theorem 3.3], dynamic programs are given that maintain the transitive closure of acyclic graphs under single-edge modification, using only the transitive closure as auxiliary relation. Thanks to Lemma 6, these programs can be easily extended. Indeed, since the number of bridges of cycles created by the insertion, and, if the graph remains acyclic, the bridge distance between two path-connected nodes are bounded by a constant, a guard formula and an update formula for the transitive closure can be constructed in a straightforward manner. ◀

5 Parameter-free Changes

In this section we consider replacement queries without parameters on ordered databases. It turns out that in this case a large class of queries can be maintained in DynFO: all queries that can be expressed in uniform AC^1 and thus, in particular, all queries that can be answered in logarithmic space. This result exploits the fact that for a fixed set of replacement queries without parameters there is only a constant number of possible changes to a structure.

An *ordered* database $\mathcal{D}$ contains a built-in linear order $\leq$ on its domain that is not modified by any changes. One might suspect that parameter-free replacement queries are not very powerful, especially when they are applied to the initially empty input database. However, thanks to the linear order, one can actually construct every finite graph with relatively simple replacement queries (and similarly for other kinds of databases). For instance, one can cycle through all pairs of nodes in lexicographic order. If (u, v) is the current maximal pair, operation *keep* can move to $(u, v+1)$ (inserting it into E) while leaving (u, v) in E and *drop* can move to $(u, v+1)$ while taking (u, v) out from E.

The update programs constructed in this section use, as additional auxiliary relation, a binary BIT-relation containing all pairs (i, j) of numbers, for which the i-th bit of the binary representation of j is 1. Here, we identify elements of an ordered database with numbers. In the following, the minimal element with respect to $\leq$ is considered as 0.

By AC^1 we denote the class of problems that can be decided by a uniform[8] family of circuits of "and", "or" and "not" gates with polynomial size, depth $\mathcal{O}(\log n)$ and unbounded fan-in. We show the following theorem.

[7] Indeed, the graphs G and G' used in the proof of Theorem 10(b) show that the following lemma fails already for Σ_1-insertions.

[8] For concreteness: first-order uniform [18].

▶ **Theorem 7.** *Let q be an AC^1 query over ordered databases and Δ a finite set of parameter-free first-order definable replacement queries. Then (q, Δ) is in* DynFO *with suitable initialization.*

Proof. We first explain the idea underlying the proof.

It uses the characterization of AC^1 by iterated first-order formulas. More precisely, we use the equality $\mathrm{AC}^1 = \mathrm{IND}[\log n]$ from [18, Theorem 5.22], where $\mathrm{IND}[t(n)]$ is the class of problems that can be expressed by applying a first-order formula $\mathcal{O}(t(n))$ times and n is the size of the domain[9]. We only give an example and refer to [18, Definition 4.16] for a formal definition. Consider the formula $\varphi_{\mathrm{TC}}(x, y) = (x = y) \vee E(x, y) \vee \exists z\big(R(x, z) \wedge R(z, y)\big)$. When applying the formula to a graph and an empty relation R it defines the relation R_1 of paths of length 1, applying it to $R \stackrel{\text{def}}{=} R_1$ defines the paths of length 2; in general applying the formula to $R \stackrel{\text{def}}{=} R_i$ defines the paths of length 2^i. Thus $\log n$-fold application of φ_{TC} defines the transitive closure relation of a graph with n vertices and therefore q_{Reach} is in $\mathrm{IND}[\log n]$.

Let q be a query in AC^1 and let k be such that q can be evaluated by $k \log n$ applications of a formula φ_q.

The program $\mathcal{P}$ uses a technique inspired from prefetching, which was called *squirrel technique* in [31]. At any point t in time[10], it starts a thread θ_β, for each possible future sequence β of $2 \log n$ change operations.

Within the next $\log n$ steps (i.e. changes), it compares whether the actual change sequence α is the prefix of β of length $\log n$. If not, thread θ_β is abandoned, as soon as α departs from β. For each of these $\log n$ steps, θ_β simulates two change operations of β and applies them to the graph G_t at time t, consecutively. After $\log n$ steps, that is, at time $t + \log n$, thread θ_β has computed $\beta(G_t)$.

During the next $\log n$ steps until time $t + 2 \log n$, θ_β evaluates q on $\beta(G_t)$ by repeatedly applying the formula φ_q, k times for each single step. Again, if the actual change sequence departs from β then θ_β is abandoned. However, if β is the actual change sequence from time t to $t + 2 \log n$, the thread θ_β does not stop and has the correct query result $q(\beta(G_t))$ at time $t + 2 \log n$.

We note that, although the time window in the above sketch stretches over $2 \log n$ change operations from time t to $t + 2 \log n$, the actual sequences whose effect on the current graph is precomputed are never longer than $\log n$. This is because the application of all $2 \log n$ operations of a sequence takes until time $t + \log n$ and by that time the first $\log n$ of these operations already lie in the past.

Of course, $\mathcal{P}$ uses a lot of prefetching. However, this is possible, because only a constant number, $d = |\Delta|$, of change operations is available at any time (and there are no parameters). Thus, there are only $d^{2 \log n} = 2^{2 \log d \log n} = n^{2 \log d}$ many different change sequences, each of which can be encoded by a tuple of arity $2 \log d$ over the domain.

This explains how $\mathcal{P}$ can give correct answers for all times $t \geq 2 \log n$. All previous time points have to be dealt with by the initialization. This initialization also equips the program with the BIT relation. Clearly, the initialization can be computed in AC^1, and therefore also in polynomial time. More details of this proof can be found in the full version of this paper. ◀

[9] In the setting of [18], first-order formulas may use built-in relations $\leq$ and BIT. The relation $\leq$ is also present here, the relation BIT can be generated by a suitable initialization, see [18, Exercise 4.18].

[10] We count the occurrence of one change as one time step.

6 Formal Languages and Σ_1-definable Change Operations

In this section, we consider the membership problem for formal languages and how it can be maintained, for regular and context-free languages, under certain kinds of complex changes.

The problem of maintaining formal languages dynamically has been studied intensely in the context of single insertions to and deletions from the relations R_σ (cf. Section 2). In that setting, the class of regular languages is exactly the class of languages maintainable in DynProp[11] and all context-free languages can be maintained in DynFO [13]. All regular, some context-free, and even some non-context-free languages can be maintained in DynFO with only unary auxiliary relations [16], but this is not possible for all context-free languages [30, 27].

Here, we consider the problem of maintaining formal languages under first-order definable change operations. We assume that only replacement queries are used whose application results in structures where each position is in at most one R_σ relation. For a given formal language L we denote the membership query for L as q_L.

We prove that regular and context-free languages can be maintained dynamically for large classes of change operations: all regular languages can be maintained in DynProp under quantifier-free change operations and all context-free languages can be maintained in DynFO under Σ_1-definable (and, dually, Π_1-definable) change operations. A setting, in which language membership can be maintained with respect to simple changes but not with respect to definable change operations is exhibited in Section 7. For quantifier-free change operations, the results are obtained by generalizations of the techniques of [13].

▶ **Theorem 8.** *Let L be a regular language and Δ a finite set of quantifier-free replacement queries. Then (q_L, Δ) can be maintained in* DynProp *with suitable initialization.*

Proof. Let L be a regular language of strings over alphabet Σ and $\mathcal{A} = (Q, \Sigma, \gamma, s, F)$ a corresponding deterministic finite automaton with set Q of states, transition function[12] γ, initial state s, and set F of accepting states. In [13, Proposition 3.3], the main auxiliary relations are of the form $S_{q,r}(i, j)$ where q, r are states of $\mathcal{A}$ and i, j are positions of the string under consideration. The intended meaning of $S_{q,r}$ is that $(i, j) \in S_{q,r}$ if and only if $\gamma^*(q, w_{i+1} \cdots w_{j-1}) = r$.[13] Notice that w_i and w_j are not relevant for determining whether $(i, j) \in S_{q,r}$.

In the presence of quantifier-free change operations it suffices to maintain binary auxiliary relations of the form $S^f_{q,r}$, where $f : \Sigma_\epsilon \to \Sigma_\epsilon$ is a *relabeling function*. The intended meaning is that $(i, j) \in S^f_{q,r}$ if and only if $\gamma^*(q, f(w_{i+1} \cdots w_{j-1})) = r$, where f is extended to strings in the straightforward way.[14] Clearly, $S_{q,r} = S^{\text{id}}_{q,r}$.

For simplicity we show how to update $S^f_{q,r}$ for replacement queries of the form $\rho(p)$ with *one* parameter p. The general case works analogously, but is notationally more involved. A replacement query with one parameter basically consists of one quantifier free formula $\mu_\sigma(p; x)$, for each element $\sigma \in \Sigma$.

[11] So, only using quantifier-free update formulas.
[12] Since in this paper δ denotes change operations, we use γ for transition functions.
[13] The relations $S_{q,r}$ were actually named $R_{q,r}$ in [13], but we want to avoid confusion with the R_σ relations. Since [13] did not use constants min and max, it used further auxiliary relations of the form I_r and F_q that contain all positions i with $\gamma^*(s, w_1 \cdots w_{i-1}) = r$, and $\gamma^*(q, w_{i+1} \cdots w_n) \in F$, respectively.
[14] It should be noted that f need not be a homomorphism since $f(\epsilon) \neq \epsilon$ is allowed.

We show how the relations $S^f_{q,r}$ can be maintained by quantifier-free update formulas $\phi^f_{q,r}(p;x,y)$. Then the (Boolean) query relation can be updated by the formula

$$\bigvee_{q,r\in Q, r'\in F} \psi_{s,q}(\min) \wedge \phi^{\text{id}}_{q,r}(p;x,y)(\min,\max) \wedge \psi_{r,r'}(\max),$$

where $\psi_{q,r}(x)$ is a formula that expresses that $\delta(q,w_x) = r$.

Intuitively, each formula $\mu_\sigma(p;x)$ determines whether position x carries σ after the change. Whether this is the case only depends on (1) the current symbol at position x, (2) the current symbol at position p, and (3) on the relative order of x and p. Thus, the impact of a change can be described as follows: some relabeling function $f_\leftarrow$ is applied at all positions $x < p$, some change might occur at position p and some relabeling function $f_\rightarrow$ is applied at all positions $x > p$. More precisely, from ρ one can derive, for each[15] $\tau \in \Sigma_\epsilon$, relabeling functions $f^\tau_\leftarrow$, $f^\tau_\rightarrow$ and a symbol $\sigma(\tau)$ such that the update formula for a relation $S^f_{q,r}$ can be described by the formula

$$\begin{aligned}\phi^f_{q,r}(p;x,y) = x<y \wedge \bigvee_{\tau\in\Sigma_\epsilon} \Big(R_\tau(p) \wedge \big(&(p \le x \wedge R^{f\circ f^\tau_\rightarrow}_{q,r}(x,y)) \vee (p \ge y \wedge R^{f\circ f^\tau_\leftarrow}_{q,r}(x,y)) \vee \\ &(x<p<y \wedge \bigvee_{q',r'} (R^{f\circ f^\tau_\leftarrow}_{q,q'}(x,p) \wedge \chi_{q',f(\sigma(\tau)),r'} \wedge R^{f\circ f^\tau_\rightarrow}_{r',r}(p,y)))\big)\Big),\end{aligned}$$

where formulas of the form $\chi_{q',a,r'}$ are defined as $\top$ if $\delta(q',a) = r'$ and $\bot$, otherwise.

The initialization of the relations $S^f_{q,r}$ is straightforward. If, for a relabeling function f, $f(\epsilon) = \sigma$ then a pair (i,j) is in $S^f_{q,r}$ if and only if $\gamma^*(q,\sigma^{j-i-1}) = r$. ◀

We next turn to context-free languages. The ideas underlying the proof of Theorem 8 can be adapted to show that the result from [13], that (membership for) context-free languages can be maintained in DynFO under simple change operations, survives under quantifier-free change operations. Through some little extra effort, this can be extended to Σ_1-definable change operations (and dually, Π_1-definable change operations).

▶ **Theorem 9.** *Let L be a context-free language and Δ a finite set of Σ_1-definable replacement queries. Then (q_L,Δ) can be maintained in* DynFO *with suitable initialization.*

The proof of Theorem 9 can be found in the full version of this article. It first shows how context-free languages can be maintained under quantifier-free changes, basically combining the idea of the proof of Theorem 8 with that of [13, Theorem 4.1]. Then it shows how the case of Σ_1-definable changes can be reduced to the quantifier-free case.

7 Inexpressibility Results

We finally turn to inexpressibility results. It is notoriously difficult to show that a query *cannot* be maintained by a DynFO program. Indeed, there are no inexpressibility results for DynFO besides those that follow from the easy observation that every query that can be maintained in DynFO under single-tuple insertions is computable in polynomial time.

We expect that it should be easier to prove inexpressibility results for DynFO in the presence of first-order definable change operations. However, we have no results of this form yet. But the following results confirm that, unsurprisingly, complex change operations can

[15] Since the schema is clear from the context, we use τ here to denote a symbol from Σ.

make it harder to maintain a query. We give two examples where allowing complex changes destroy a previous maintainability result, Theorems 10 and 13, and one example, Theorem 12 where we are able to show an inexpressibility result in the presence of complex deletions but not yet for single-tuple changes.

Towards our first result, we recall that the reachability query can be maintained under single-tuple insertions with the transitive closure of the edge relation as only auxiliary relation and that this does not hold if one allows single-tuple deletions [4]. We show next that the transitive closure also does not suffice in the presence of single-tuple insertions and one complex insertion query.

For general directed graphs, a parameter-free and quantifier-free insertion query suffices, for acyclic graphs a parameter-free insertion query defined by an existential formula suffices. The latter result should be contrasted with Theorem 5.

▶ **Theorem 10.**

(a) *There is a quantifier-free and parameter-free insertion query ρ such that $(q_{Reach}, \{\textbf{insert } \bar{p} \textbf{ into } E, \rho\})$ cannot be maintained in* DynFO *on ordered directed graphs, if all auxiliary relations besides the query relation and the linear order are unary.*

(b) *There exists an Σ_1-definable and parameter-free insertion query ρ', for which the above statement holds even restricted to* acyclic*, directed graphs.*

Proof. For ease of presentation, we first give a proof for unordered directed graphs.

The proof follows an approach that has been used often before and that was made precise in [30]. We say that a k-ary query q is expressed by a formula $\varphi(\bar{x})$ with *help relations of schema* τ, if, for every database $\mathcal{D}$, there is a τ-structure H over the same domain such that for every k-tuple $\bar{a}$ over the domain of $\mathcal{D}$ it holds[16]: $\bar{a} \in q(\mathcal{D})$ if and only if $(\mathcal{D}, H) \models \varphi(\bar{a})$.

The proof is by contradiction and proceeds in the same way in both cases, (a) and (b). Our goal is to show that, under the assumption that there is a dynamic program for (a) or (b), the transitive closure of path graphs, that is, graphs that consist of a single directed path, can be expressed with unary help relations, contradicting the following lemma from [30], which is not hard to prove with the help of locality arguments.

▶ **Lemma 11** ([30, Lemma 4.3.2])**.** *The transitive closure of path graphs cannot be expressed by a first-order formula with unary help relations.*

We refer to Figure 1 for an illustration of the following high-level sketch. We start from an arbitrary path graph G_0 and equip it with some unary relations C_0, C_1, C_2. From G_0, C_0, C_1, C_2 we define a graph G in a first-order fashion, whose simple directed paths have length at most 2, so the transitive closure relation TC of G is definable by a first-order formula. Finally, the crucial step happens: the change operation δ transforms G into a graph $G' = \delta(G)$ with the property that $q_{\text{Reach}}(G_0)$ can be defined from $q_{\text{Reach}}(G')$ by a first-order formula. We can conclude that $q_{\text{Reach}}(G_0)$ can be defined by a first-order formula with the help of suitable unary help relations, since all steps from G_0 to G' are first-order definable, TC is first-order definable from G, and we assume that there is a dynamic program that computes $q_{\text{Reach}}(G')$ from G, TC and some unary auxiliary relations. This contradicts Lemma 11.

[16] This notion should not be confused with definability of the query q in existential second-order logic. In the latter case, the relations can be chosen depending on $\bar{a}$, but here the relations need to "work" for all tuples $\bar{a}$.

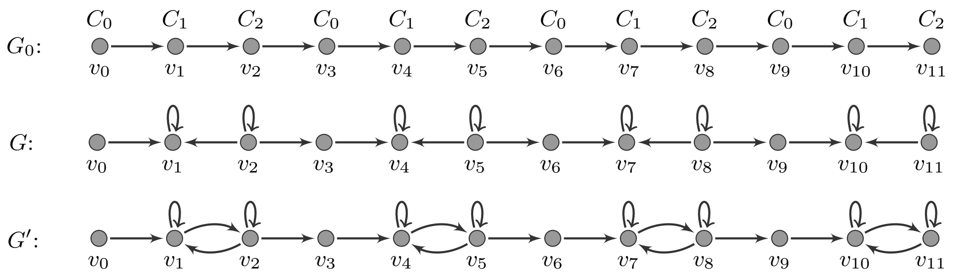
G_0:
G:
G':
C_0 C_1 C_2 C_0 C_1 C_2 C_0 C_1 C_2 C_0 C_1 C_2
v_0 v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11}

Figure 1 The graphs from the proof of Theorem 10(a).

For (a), we use the insertion query $\rho = \mu_E(x,y) \stackrel{\text{def}}{=} E(x,y) \vee \big(E(x,x) \wedge E(y,y) \wedge E(y,x)\big)$ that adds all edges (x,y) for which there is an edge (y,x) and both x and y have self-loops. We assume that there is a DynFO-program $\mathcal{P}$ that maintains the reachability query on directed graphs under insertion queries $\{\textbf{insert } \bar{p} \textbf{ into } E, \rho\}$. We further assume that $\mathcal{P}$ uses (only) unary auxiliary relations $B_1, \dots, B_k$, for some k, besides the binary relation Q intended to store the query result. We show how to construct from $\mathcal{P}$ a first-order formula φ that expresses the reachability query q_{Reach} for simple paths with unary help relations $B_1, \dots, B_k, C_0, C_1, C_2$, contradicting Lemma 11.

Let $G_0 = (V_0, E_0)$ be a simple path with $V_0 = \{v_0, \dots, v_n\}$, for which we want to define q_{Reach} using unary help relations $B_1, \dots, B_k, C_0, C_1, C_2$. Let C_0, C_1, C_2 be defined by $C_i = \{v_j \mid 0 \leq j \leq n, j \equiv_3 i\}$ where $\equiv_3$ denotes modulo 3 equivalence. From G_0 and C_0, C_1, C_2 we define the following graph G with nodes $v_0, \dots, v_n$. The graph G has an edge from vertex v to w if one of the following cases holds:

- $v \in C_0, w \in C_1$ and (v,w) is an edge in G_0,
- $v \in C_1, w \in C_2$ and (w,v) is an edge in G_0,
- $v \in C_2, w \in C_0$ and (v,w) is an edge in G_0, or
- $v \in C_1 \cup C_2$ and $v = w$.

We observe that the graph G can be first-order defined from G_0 and C_0, C_1, C_2.

Let $\delta \stackrel{\text{def}}{=} \rho$ and[17] $G' \stackrel{\text{def}}{=} \delta(G)$. The graphs G_0, G and G' for $n = 11$ are depicted in Figure 1. By our assumption, the update formula $\psi = \phi^Q_\delta(x_1, x_2)$ of $\mathcal{P}$ for the query relation Q and operation δ defines the reachability query for $\delta(G) = G'$ with the help of suitable auxiliary relations $B_1, \dots, B_k$ and the transitive closure TC of the edge relation of G.

Altogether, $G = f(G_0, C_0, C_1, C_2)$, for some first-order definable function f, TC is first-order definable from G, $G' = \delta(G)$, $q_{\text{Reach}}(G')$ is first-order definable from G, TC and $B_1, \dots, B_k$, and therefore

$$q_{\text{Reach}}(G_0) = q_{\text{Reach}}(G') \setminus \{(w,v) \mid v \in C_1, w \in C_2, (v,w) \text{ is an edge in } G_0\}$$

is first-order definable from G_0, C_0, C_1, C_2, and $B_1, \dots, B_k$, contradicting Lemma 11, as desired.

The proof for (b) and the extension to ordered graphs can be found in the full version of this paper. ◀

We now turn towards inexpressibility by quantifier-free update formulas. Very likely quantifier-free update formulas are too weak to maintain q_{Reach} even under single-tuple

[17] Since ρ is parameter-free, the insertion query ρ and its corresponding change operation δ are basically the same.

changes. Yet only restricted inexpressibility results have been obtained so far. The query q_{Reach} cannot be maintained in DynProp under single-tuple changes when the auxiliary relations are at most binary or when the initialization is severely restricted [32]. For the more general alternating reachability query quantifier-free update formulas do not suffice [13]. The next result shows that q_{Reach} cannot be maintained in DynProp, even if besides single-edge insertions only a single, very simple deletion query is allowed.

▶ **Theorem 12.** *There is a quantifier-free deletion query ρ with one parameter such that $(q_{Reach}, \Delta_E \cup \{\rho\})$ cannot be maintained in* DynProp.

Proof. For the proof, we combine the standard tool for obtaining inexpressibility results for DynProp, the Substructure Lemma [32, 13], with a combinatorial technique based on upper and lower bounds of Ramsey numbers [29].

The intuition behind the Substructure Lemma is as follows. When updating an auxiliary tuple $\bar{d}$ after a quantifier-free change parameterized by $\bar{p}$, a quantifier-free update formula only has access to $\bar{d}$ and $\bar{p}$. Thus, if a change operation changes a tuple *inside* a substructure $\mathcal{A}$ of a state $\mathcal{S}$ of a dynamic program, the auxiliary data of $\mathcal{A}$ is not affected by any information from *outside* of $\mathcal{A}$. In particular, two isomorphic substructures $\mathcal{A}$ and $\mathcal{B}$ remain isomorphic, when corresponding changes are applied to them. The Substructure Lemma is formally stated in [32, Lemma 2]. Even though the lemma is phrased for single-tuple changes only, the same proof, using the intuition explained above, extends to quantifier-free replacement queries.

For the actual proof, we assume, towards a contradiction, that there is a quantifier-free dynamic program $\mathcal{P}$ over schema τ of arity k that maintains q_{Reach} under the quantifier-free deletion $\rho(p) = \mu(p; x, y) \stackrel{\text{def}}{=} E(x, y) \wedge \neg E(p, x)$ which deletes an edge (x, y) if there is an edge (p, x). Our goal is to construct a graph G such that not all change sequences of length $k + 1$ can be maintained, no matter the initial auxiliary data.

Let n be a sufficiently large number, to be specified later. The vertex set of the graph is of the form $\{s, t\} \cup A \cup C$, for some disjoint sets A and C, with $|C| = n$. The set A contains a node for every subset of size $k + 1$ of C, that is, $A \stackrel{\text{def}}{=} \{a_X \mid X \subseteq C \text{ and } |X| = k + 1\}$. Let B be a subset of A, to be specified later.

The graph has the following edges:

(a) For each $a_X \in A$ there is an edge (s, a_X).

(b) For each $a_X \in B$ there is an edge (a_X, t).

(c) There is an edge (c, a_X) for nodes $c \in C, a_X \in A$ if $c \notin X$.

Intuitively, the nodes in C control how edges from A to t can be removed. Each node $c \in C$ is connected to a subset $A' \subseteq A$, and thus applying a change $\rho(c)$ will result in removing all edges (a_X, t) for all $a_X \in A'$. The graph is constructed in such a way that

(⋆) for a change sequence $\alpha = (\rho(c_1), \dots, \rho(c_{k+1}))$ with $|\{c_1, \dots, c_{k+1}\}| = k + 1$ it holds $(s, t) \in q_{\text{Reach}}(\alpha(G))$ if and only if $a_{\{c_1,\dots,c_{k+1}\}} \in B$.

To see this, observe that after applying changes $\rho(c_1), \dots, \rho(c_{k+1})$, all edges (a_X, t) are deleted, for which $\{c_1, \dots, c_{k+1}\} \not\subseteq X$. Thus at most the edge $(a_{\{c_1,\dots,c_{k+1}\}}, t)$ is still present. However, this edge was at all present in the graph if and only if $a_{\{c_1,\dots,c_{k+1}\}} \in B$.

For choosing the size of C and the set B, we employ the combinatorial Lemma 2 from [29]. The lemma guarantees that, depending on the schema $\tau \cup \{c_s, c_t\}$, there is an n_0 such that for every $n > n_0$ there is some m such that the following holds.

(S1) For every state $\mathcal{S}$ of the dynamic program for G, and each set C with at least n vertices of G with a linear order $<$, there is a subset C' of C of size at least m such that the k-ary auxiliary data on C' is $<$-monochromatic in the structure $(\mathcal{S}, s, t)$, i.e. all $<$-ordered

k-tuples over C' have the same quantifier-free type (including their relationships to the interpretations s, t of the constants c_s, c_t).

(S2) There is a subset B of A such that for every subset $\hat{C}$ of C of size m, there are $(k+1)$-element sets $Y = \{c_1, \ldots, c_{k+1}\}, Y' = \{c'_1, \ldots, c'_{k+1}\} \subseteq \hat{C}$ with $a_Y \in B$ and $a_{Y'} \notin B$.

We outline how the graph G is used to obtain a contradiction. Let $\mathcal{S}$ be a state of the dynamic program for the graph G with $|C| = n > n_0$ and let $<$ be a linear order. Choose B as described above and a subset C' of C of size $|C'| = m$ that is $<$-monochromatic in $(\mathcal{S}, s, t)$. Choose $Y = \{c_1, \ldots, c_{k+1}\}, Y' = \{c'_1, \ldots, c'_{k+1}\} \subseteq C'$ with $a_Y \in B$ and $a_{Y'} \notin B$. By the Substructure Lemma from [32] generalized to quantifier-free changes, the dynamic program $\mathcal{P}$ yields the same result for the tuple (s, t) for the change sequences $(\rho(c_1), \ldots, \rho(c_{k+1}))$ and $(\rho(c'_1), \ldots, \rho(c'_{k+1}))$ since C' is $<$-monochromatic. Yet the result should be different due to $(\star)$ and $a_Y \in B$, $a_{Y'} \notin B$. This is a contradiction. ◀

Finally, we turn to lower bounds for the maintenance of languages. We exhibit an example that illustrates that maintaining regular languages under full first-order replacement queries might be hard: there is a regular language L that can be maintained in DynFO under single-tuple changes with nullary auxiliary relations, but there is a relatively simple replacement query, for which this no longer holds. This is no general hardness result, as we only allow very restricted auxiliary relations, but it demonstrates the barrier of our techniques. The proof of the following result can be found in the full version of this paper.

▶ **Theorem 13.** *There is a regular language L over some alphabet Σ and a replacement query ρ, such that (q_L, Δ_Σ) can be maintained in* DynFO *with nullary auxiliary relations, but not $(q_L, \Delta_\Sigma \cup \{\rho\})$.*

8 Conclusion

In this paper, we studied the maintainability of queries in the Dynamic Complexity setting under first-order defined replacement queries. The main insight of this study is that many maintainability results carry over from the single-tuple world to settings with more general change operations. We were actually quite surprised to see that so many positive results survive this transition. However, many questions remain open, for instance: To which extent can the reachability query for (undirected or acyclic) graphs be maintained under definable deletions? What about reachability for unrestricted directed graphs under definable insertions? What about other queries? Are binary auxiliary relations sufficient in Theorem 3?

We were less surprised by the fact that stronger change operations can yield inexpressibility, but even these results required some care. Our main contribution in that respect is the proof that DynProp cannot maintain the reachability query under quantifier-free replacement queries.

From Theorem 7 about parameter-free changes and its proof, we take another insight regarding inexpressibility proofs: the squirrel technique is quite powerful to prepare an update program for a non-constant (i.e., logarithmic) number of changes. Inexpressibility proofs need to take that into account and to argue "around it".

References

1 Tom J. Ameloot, Jan Van den Bussche, and Emmanuel Waller. On the expressive power of update primitives. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-*

SIGART Symposium on Principles of Database Systems (PODS), pages 139–150, 2013. doi:10.1145/2463664.2465218.

2 Samir Datta, William Hesse, and Raghav Kulkarni. Dynamic complexity of directed reachability and other problems. In *Automata, Languages, and Programming - 41st International Colloquium (ICALP), Proceedings, Part I*, pages 356–367, 2014. doi:10.1007/978-3-662-43948-7_30.

3 Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in DynFO. In *Automata, Languages, and Programming - 42nd International Colloquium (ICALP), Proceedings, Part II*, pages 159–170, 2015. doi:10.1007/978-3-662-47666-6_13.

4 Guozhu Dong, Leonid Libkin, and Limsoon Wong. On impossibility of decremental recomputation of recursive queries in relational calculus and SQL. In *Proceedings of the Fifth International Workshop on Database Programming Languages (DBPL-5)*, page 7, 1995.

5 Guozhu Dong and Chaoyi Pang. Maintaining transitive closure in first order after node-set and edge-set deletions. *Inf. Process. Lett.*, 62(4):193–199, 1997. doi:10.1016/S0020-0190(97)00066-5.

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

7 Guozhu Dong and Jianwen Su. Incremental and decremental evaluation of transitive closure by first-order queries. *Inf. Comput.*, 120(1):101–106, 1995. doi:10.1006/inco.1995.1102.

8 Guozhu Dong, Jianwen Su, and Rodney W. Topor. Nonrecursive incremental evaluation of datalog queries. *Ann. Math. Artif. Intell.*, 14(2-4):187–223, 1995. doi:10.1007/BF01530820.

9 Guozhu Dong and Rodney W. Topor. Incremental evaluation of datalog queries. In *Proceedings of the 4th International Conference on Database Theory (ICDT)*, pages 282–296, 1992. doi:10.1007/3-540-56039-4_48.

10 Kousha Etessami. Dynamic tree isomorphism via first-order updates. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 235–243, 1998. doi:10.1145/275487.275514.

11 Solomon Feferman. Some recent work of Ehrenfeucht and Fraïssé. In *Proc. Summer Institute of Symbolic Logic*, pages 201–209, 1957.

12 Solomon Feferman and Robert L. Vaught. The first order properties of algebraic systems. *Fund. Math.*, 47:57–103, 1959.

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

14 Erich Grädel and Sebastian Siebertz. Dynamic definability. In *15th International Conference on Database Theory (ICDT)*, pages 236–248, 2012. doi:10.1145/2274576.2274601.

15 Ashish Gupta, Inderpal Singh Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 157–166, 1993. doi:10.1145/170035.170066.

16 William Hesse. *Dynamic Computational Complexity*. PhD thesis, University of Massachusetts Amherst, 2003.

17 William Hesse and Neil Immerman. Complete problems for dynamic complexity classes. In *17th IEEE Symposium on Logic in Computer Science (LICS), Proceedings*, page 313, 2002. doi:10.1109/LICS.2002.1029839.

18 Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999. doi:10.1007/978-1-4612-0539-5.

19 Christoph Koch. Incremental query evaluation in a ring of databases. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 87–98, 2010. `doi:10.1145/1807085.1807100`.

20 Leonid Libkin. *Elements of Finite Model Theory.* Springer, 2004.

21 Johann A. Makowsky. Algorithmic uses of the Feferman-Vaught Theorem. *Ann. Pure Appl. Logic*, 126(1-3):159–213, 2004. `doi:10.1016/j.apal.2003.11.002`.

22 Chaoyi Pang, Guozhu Dong, and Kotagiri Ramamohanarao. Incremental maintenance of shortest distance and transitive closure in first-order logic and SQL. *ACM Trans. Database Syst.*, 30(3):698–721, 2005. `doi:10.1145/1093382.1093384`.

23 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`.

24 Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. Dynamic complexity under definable changes. *CoRR*, abs/1701.02494, 2017. URL: `http://arxiv.org/abs/1701.02494`.

25 Thomas Schwentick and Thomas Zeume. Dynamic complexity: recent updates. *SIGLOG News*, 3(2):30–52, 2016. `doi:10.1145/2948896.2948899`.

26 Sebastian Siebertz. Dynamic definability. Diploma thesis, RWTH Aachen, 2011.

27 Nils Vortmeier. Komplexitätstheorie verlaufsunabhängiger dynamischer Programme. Master's thesis, TU Dortmund, 2013.

28 Volker Weber and Thomas Schwentick. Dynamic complexity theory revisited. *Theory Comput. Syst.*, 40(4):355–377, 2007. `doi:10.1007/s00224-006-1312-0`.

29 Thomas Zeume. The dynamic descriptive complexity of k-clique. In *Mathematical Foundations of Computer Science (MFCS) - 39th International Symposium, Proceedings, Part I*, pages 547–558, 2014. `doi:10.1007/978-3-662-44522-8_46`.

30 Thomas Zeume. *Small Dynamic Complexity Classes.* PhD thesis, TU Dortmund University, 2015.

31 Thomas Zeume and Thomas Schwentick. Dynamic conjunctive queries. In *Proc. 17th International Conference on Database Theory (ICDT)*, pages 38–49, 2014. `doi:10.5441/002/icdt.2014.08`.

32 Thomas Zeume and Thomas Schwentick. On the quantifier-free dynamic complexity of reachability. *Inf. Comput.*, 240:108–129, 2015. `doi:10.1016/j.ic.2014.09.011`.