On the quantifier-free dynamic complexity of Reachability

Thomas Zeume and Thomas Schwentick

TU Dortmund University
Germany

Abstract. The dynamic complexity of the reachability query is studied in the dynamic complexity framework of Patnaik and Immerman, restricted to quantifier-free update formulas.
It is shown that, with this restriction, the reachability query cannot be dynamically maintained, neither with binary auxiliary relations nor with unary auxiliary functions, and that ternary auxiliary relations are more powerful with respect to graph queries than binary auxiliary relations. Further results are obtained including more inexpressibility results for reachability in a different setting, inexpressibility results for some other queries and normal forms for quantifier-free update programs.

1 Introduction

In modern data management scenarios, data is subject to frequent changes. In order to avoid costly re-computations from scratch after each small update, one can try to (re-)use auxiliary data structures that has been already computed before to keep the information about the data up-to-date. However, the auxiliary data structures need to be updated dynamically whenever the data changes.

The descriptive dynamic complexity framework (short: dynamic complexity) introduced by Patnaik and Immerman [1] models this setting. It was mainly inspired by relational databases. For a relational database subject to change, auxiliary relations are maintained with the intention to help answering a query $\mathcal{Q}$. When an update to the database, an insertion or deletion of a tuple, occurs, every auxiliary relation is updated through a first-order query that can refer to the database as well as to the auxiliary relations. A particular auxiliary relation shall always represent the answer to $\mathcal{Q}$. The class of all queries maintainable in this way, and thus also in the core of SQL, is called DynFO.

Beyond query or view maintenance in databases we consider it an important goal to understand the dynamic complexity of fundamental algorithmic problems. Reachability in directed graphs is the most intensely investigated problem in dynamic complexity (and also much studied in dynamic algorithms and other dynamic contexts) and the main query studied in this paper. It is one of the simplest inherently recursive queries and thus serves as a kind of drosophila in the study of the dynamic maintainability of recursive queries by non-recursive means. It can be maintained with first-order update formulas supplemented by

counting quantifiers on general graphs [2] and with plain first-order update formulas on both acyclic graphs and undirected graphs [1]. However, it is not known whether Reachability on general graphs is maintainable with first-order updates. This is one of the major open questions in dynamic complexity.

All attempts to show that Reachability *cannot* be maintained in DynFO have failed. In fact, there are no general inexpressibility results for DynFO at all.[1] This seems to be due to a lack of understanding of the underlying mechanisms of DynFO. To improve the understanding of dynamic complexity, mainly two kinds of restrictions of DynFO have been studied: (1) limiting the information content of the auxiliary data by restricting the arity of auxiliary relations and functions and (2) reducing the amount of quantification in update formulas.

A study of bounded arity auxiliary relations was started in [3] and it was shown that unary auxiliary relations are not sufficient to maintain the reachability query with first-order updates. Further inexpressibility results for unary auxiliary relations were shown and an arity hierarchy for auxiliary relations was established. However, to separate level k from higher levels, database relations of arity larger than k were used. Thus the hierarchy has not yet been established for queries on graphs. In [4] it was shown that unary auxiliary relations are not sufficient to maintain Reachability for update formulas of any local logic. The proofs strongly use the "static" weakness of local logics and do not fully exploit the dynamic setting, as they only require update sequences of constant length.

The second line of research was initiated by Hesse [5]. He invented and studied the class DynProp of queries maintainable with quantifier-free update formulas. He proved that Reachability on deterministic graphs (i.e. graphs of unary functions) can be maintained with quantifier-free first-order update formulas.

There is still no proof that Reachability on general graphs cannot be maintained in DynProp. However, *some* inexpressibility results for DynProp have been shown in [6]: the alternating reachability query (on graphs with $\wedge$- and $\vee$-nodes) is not maintainable in DynProp. Furthermore, on strings, DynProp exactly contains the regular languages (as Boolean queries on strings).

Contributions. The high-level goal of this paper is to achieve a better understanding of the dynamic maintainability of Reachability and dynamic complexity in general. Our main result is that the reachability query cannot be dynamically maintained by quantifier-free updates with binary auxiliary relations. This result is weaker than that of [3] in terms of the logic (quantifier-free vs. general first-order) but it is stronger with respect to the information content of the auxiliary data (binary vs. unary). We establish a strict hierarchy between DynProp for unary, binary and ternary auxiliary relations (this is still open for DynFO).

We further show that Reachability is not maintainable with unary auxiliary *functions* (plus unary auxiliary relations). Although unary functions provide less information content than binary relations, they offer a very weak form of

[1] Of course, a query maintainable in DynFO can be evaluated in polynomial time and thus queries that cannot be evaluated in polynomial time cannot be maintained in DynFO either.

quantification in the sense that more elements of the domain can be taken into account by update formulas.

All these results hold in the setting of Patnaik and Immerman where update sequences start from an empty database as well as in the setting that starts from an arbitrary database, where the auxiliary data is initialized by an arbitrary function. We show that if, in the latter setting, the initialization mapping is permutation-invariant, quantifier-free updates cannot maintain Reachability even with auxiliary functions and relations of arbitrary arity. Intuitively a permutation-invariant initialization mapping maps isomorphic databases to isomorphic auxiliary data. A particular case of permutation-invariant initialization mappings, studied in [7], is when the initialization is specified by logical formulas. In this case, lower bounds for first-order update formulas have been obtained for several problems [7].

We transfer many of our inexpressibility results to the k-Clique query for fixed $k \geq 3$ and the k-Col query for fixed $k \geq 2$.

Finally, we show two normal form results: every query in DynProp is already maintainable with negation-free quantifier-free formulas only as well as with conjunctive quantifier-free formulas only. Thus, one approach to inexpressibility proofs could be to use these syntactically restricted update formulas.

Related Work. We already mentioned the related work that is most closely related to our results. As said before, the reachability query has been studied in various dynamic frameworks, one of which is the Cell Probe model. In the Cell Probe model, one aims for lower bounds for the number of memory accesses of a RAM machine for static and dynamic problems. For dynamic reachability, lower bounds of order $\log n$ have been proved [8].

Outline. In Section 2 we fix our notation and in Section 3 we define our dynamic setting more precisely. The lower bound results for Reachability are presented in Section 4 (for auxiliary relations) and in Section 5 (for auxiliary functions). In Section 6 we transfer the lower bounds to other queries. Finally, we establish the two normal forms for DynProp in Section 7. Due to the space limit, most proofs are only available in the full version of the paper [9].

Acknowledgement. We thank Ahmet Kara and Martin Schuster for careful proofreading. We acknowledge the financial support by the German DFG under grant SCHW 678/6-1.

2 Preliminaries

A *domain* is a finite set. A *schema (or signature)* τ consists of a set τ_{rel} of relation symbols and a set τ_{const} of constant symbols together with an arity function $\text{Ar} : \tau_{\text{rel}} \mapsto \mathbb{N}$. A *database* $\mathcal{D}$ of schema τ with domain D is a mapping that 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 $\mathcal{D}$ is a database with schema τ and domain D. Sometimes we omit the schema when it is clear from the context. If $\mathcal{S}$ is a structure over domain D and D' is a subset of D that contains all constants

of $\mathcal{S}$, then the substructure of $\mathcal{S}$ induced by D' is denoted by $\mathcal{S} \restriction D'$. For two structures $\mathcal{S}$ and $\mathcal{T}$ we write $\mathcal{S} \simeq_\pi \mathcal{T}$ if $\mathcal{S}$ and $\mathcal{T}$ are isomorphic via π.

The k-ary atomic type[2] $\langle \mathcal{S}, \vec{a} \rangle$ of a tuple $\vec{a} = (a_1, \ldots, a_k)$ over D with respect to a τ-structure $\mathcal{S}$ is the set of all atomic formulas $\varphi(\vec{x})$ with $\vec{x} = (x_1, \ldots, x_k)$ for which $\varphi(\vec{a})$ holds in $\mathcal{S}$, where $\varphi(\vec{a})$ is short for the substitution of $\vec{x}$ by $\vec{a}$ in φ. We note that the atomic formulas can use constant symbols.

An *s-t-graph* is a graph $G = (V, E)$ with two distinguished nodes s and t. A *k-layered s-t-graph* G is a directed graph (V, E) in which $V - \{s, t\}$ is partitioned into k layers $A_1, \ldots, A_k$ such that every edge is from s to A_1, from A_k to t or from A_i to A_{i+1}, for some $i \in \{1, \ldots, k-1\}$. The *s-t-reachability query* s-t-Reach is a Boolean query that is true for an s-t-graph G, if and only if t can be reached from s in G.

3 Dynamic Queries and Programs

The following presentation follows [10] and [11]. For a more formal introduction, see [9].

A *dynamic instance* of the s-t-reachability query is a pair (G, α), where G is an s-t-graph and α is a sequence of changes to G, i.e. a sequence of edge insertions and deletions. The dynamic s-t-reachability query Dyn(s-t-Reach) yields as result the relation that is obtained by first applying the updates from α to G and then evaluating the s-t-reachability query on the resulting graph. This setting extends to general databases and other queries in a straightforward way.

The database resulting from applying an update δ to a database $\mathcal{D}$ is denoted by $\delta(\mathcal{D})$. The result $\alpha(\mathcal{D})$ of applying a sequence of updates $\alpha = \delta_1 \ldots \delta_m$ to a database $\mathcal{D}$ is defined by $\alpha(\mathcal{D}) \stackrel{\text{def}}{=} \delta_m(\ldots(\delta_1(\mathcal{D}))\ldots)$.

Dynamic programs, to be defined next, consist of an initialization mechanism and an update program. The former yields, for every database $\mathcal{D}$ an initial state of P with initial auxiliary data (and possibly with further built-in data). The latter defines the new state, for each update in α. Built-in data never changes. In general, built-in data can be "simulated" by auxiliary data yet this does not (seem to) hold for all of the restricted kinds of auxiliary data studied in this paper.

A *dynamic schema* is a triple $(\tau_{\text{in}}, \tau_{\text{aux}}, \tau_{\text{bi}})$ of schemas of the input database, the built-in database and the auxiliary database, respectively. We always let $\tau \stackrel{\text{def}}{=} \tau_{\text{in}} \cup \tau_{\text{aux}} \cup \tau_{\text{bi}}$.

Definition 1. (Update program) An *update program* P over dynamic schema $(\tau_{\text{in}}, \tau_{\text{aux}}, \tau_{\text{bi}})$ is a set of first-order formulas (called *update formulas* in the following) that contains, for every $R \in \tau_{\text{aux}}$ and every $\delta \in \{\text{INS}_S, \text{DEL}_S\}$ with $S \in \tau_{\text{in}}$, an update formula $\phi^R_\delta(x_1, \ldots, x_l; y_1, \ldots, y_m)$ over the schema τ where l is the arity of S and m is the arity of R.

[2] As we only consider atomic types in this paper, we will often simply say type instead of atomic type.

A *program state* $\mathcal{S}$ over dynamic schema $(\tau_{\text{in}}, \tau_{\text{aux}}, \tau_{\text{bi}})$ is a structure $(D, \mathcal{I}, \mathcal{A}, \mathcal{B})$ where D is the domain, $\mathcal{I}$ is a database over the input schema (the *current database*), $\mathcal{A}$ is a database over the auxiliary schema (the *auxiliary database*) and $\mathcal{B}$ is a database over the built-in schema (the *built-in database*).

The semantics of update programs is as follows. For an update $\delta(\vec{a})$ and program state $\mathcal{S} = (D, \mathcal{I}, \mathcal{A}, \mathcal{B})$ we denote by $P_\delta(\mathcal{S})$ the state $(D, \delta(\mathcal{I}), \mathcal{A}', \mathcal{B})$, where $\mathcal{A}'$ consists of relations $R' \stackrel{\text{def}}{=} \{\vec{b} \mid \mathcal{S} \models \phi_\delta^R(\vec{a}; \vec{b})\}$. The effect $P_\alpha(\mathcal{S})$ of an update sequence $\alpha = \delta_1 \dots \delta_m$ to a state $\mathcal{S}$ is the state $P_{\delta_m}(\dots(P_{\delta_1}(\mathcal{S}))\dots)$.

Definition 2. (Dynamic program) A *dynamic program* is a triple $(P, \textsc{Init}, Q)$, where P is an update program over some dynamic schema $(\tau_{\text{in}}, \tau_{\text{aux}}, \tau_{\text{bi}})$, the tuple $\textsc{Init} = (\textsc{Init}_{\text{aux}}, \textsc{Init}_{\text{bi}})$ consists of a function $\textsc{Init}_{\text{aux}}$ that maps τ_{in}-databases to τ_{aux}-databases and a function $\textsc{Init}_{\text{bi}}$ that maps domains to τ_{bi}-databases, and $Q \in \tau_{\text{aux}}$ is a designated *query symbol*.

A dynamic program $\mathcal{P} = (P, \textsc{Init}, Q)$ *maintains* a dynamic query $\textsc{Dyn}(\mathcal{Q})$ if, for every dynamic instance $(\mathcal{D}, \alpha)$, the relation $\mathcal{Q}(\alpha(\mathcal{D}))$ coincides with the query relation $Q^{\mathcal{S}}$ in the state $\mathcal{S} = P_\alpha(\mathcal{S}_{\textsc{Init}}(\mathcal{D}))$, where $\mathcal{S}_{\textsc{Init}}(\mathcal{D})$ is the initial state, i.e. $\mathcal{S}_{\textsc{Init}}(\mathcal{D}) \stackrel{\text{def}}{=} (D, \mathcal{D}, \textsc{Init}_{\text{aux}}(\mathcal{D}), \textsc{Init}_{\text{bi}}(D))$.

Several dynamic settings and restrictions of dynamic programs have been studied in the literature [1, 12, 7, 11]. Possible parameters are, for instance:

- the logic in which update formulas are expressed;
- whether in dynamic instances $(\mathcal{D}, \alpha)$, the initial database $\mathcal{D}$ is always empty;
- whether the initialization mapping $\textsc{Init}_{\text{aux}}$ is *permutation-invariant* (short: *invariant*), that is, whether $\pi(\textsc{Init}_{\text{aux}}(\mathcal{D})) = \textsc{Init}_{\text{aux}}(\pi(\mathcal{D}))$ holds, for every database $\mathcal{D}$ and permutation π of the domain; and
- whether there are any built-in relations at all.

Definition 3. (DynFO, DynProp) DynFO is the class of all dynamic queries maintainable by dynamic programs with first-order update formulas. DynProp is the subclass of DynFO, where update formulas do not use quantifiers. A dynamic program is *k-ary* if the arity of its auxiliary relation symbols[3] is at most k. By k-ary DynProp (resp. DynFO) we refer to dynamic queries that can be maintained with k-ary dynamic programs.

Thus in our basic setting the initialization mappings can be arbitrary. We will explicitly state when we relax this most general setting. Figure 1 illustrates the relationships between the various settings for the initialization. From now on we restrict our attention to quantifier-free update programs. Next, we give a non-trivial example for such a program.

Example 1. We provide a DynProp-program P for the dynamic variant of the Boolean query NonEmptySet, where, for a unary relation U subject to insertions and deletions of elements, one asks whether U is empty. It illustrates a technique to maintain lists with quantifier-free dynamic programs, introduced in [11, Proposition 4.5], which is used in some of our upper bounds.

[3] We note that this restriction does not apply to the built-in relations.

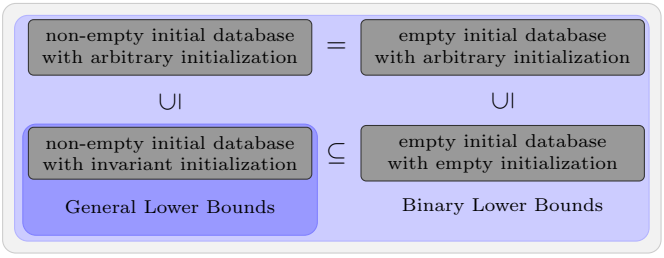

Fig. 1. Initializations considered in literature and lower bounds obtained in this paper for quantifier-free updates.

The program P is over auxiliary schema $\tau_{\text{aux}} = \{Q, \textsc{First}, \textsc{Last}, \textsc{List}\}$, where Q is the query bit (i.e. a 0-ary relation symbol), First and Last are unary relation symbols, and List is a binary relation symbol. The idea is to store in a program state $\mathcal{S}$ a list of all elements currently in U. The list structure is stored in the binary relation $\textsc{List}^{\mathcal{S}}$ such that $\textsc{List}^{\mathcal{S}}(a, b)$ holds for all elements a and b that are adjacent in the list. The first and last element of the list are stored in $\textsc{First}^{\mathcal{S}}$ and $\textsc{Last}^{\mathcal{S}}$, respectively. We note that the order in which the elements of U are stored in the list depends on the order in which they are inserted into the set.

For a given instance of NonEmptySet the initialization mapping initializes the auxiliary relations accordingly.

The update formulas for insertions are as follows:

$$\phi^{\textsc{First}}_{\text{INS}}(a; x) \stackrel{\text{def}}{=} (\neg Q \wedge a = x) \vee (Q \wedge \textsc{First}(x)) \qquad \phi^{\textsc{Last}}_{\text{INS}}(a; x) \stackrel{\text{def}}{=} a = x$$

$$\phi^{\textsc{List}}_{\text{INS}}(a; x, y) \stackrel{\text{def}}{=} \textsc{List}(x, y) \vee (\textsc{Last}(x) \wedge a = y) \qquad \phi^{Q}_{\text{INS}}(a) \stackrel{\text{def}}{=} \top$$

For deletions we only exhibit the update formula for List, the others are similar.

$$\phi^{\textsc{List}}_{\text{DEL}}(a; x, y) \stackrel{\text{def}}{=} x \neq a \wedge y \neq a \wedge \big(\textsc{List}(x, y) \vee (\textsc{List}(x, a) \wedge \textsc{List}(a, y))\big)$$

4 Lower Bounds for Dynamic Reachability

In this section we prove lower bounds for the maintainability of the dynamic s-t-reachability query Dyn(s-t-Reach).

The proofs use the following tool which is a slight variation of Lemma 1 from [11]. The intuition is as follows. When updating an auxiliary tuple $\vec{c}$ after an insertion or deletion of a tuple $\vec{d}$, a quantifier-free update formula has access to $\vec{c}$, $\vec{d}$, and the constants only. Thus, if a sequence of updates changes only tuples from a substructure $\mathcal{S}'$ of $\mathcal{S}$, the auxiliary data of $\mathcal{S}'$ is not affected by information outside $\mathcal{S}'$. In particular, two isomorphic substructures $\mathcal{S}'$ and $\mathcal{T}'$ should remain isomorphic, when corresponding updates are applied to them.

We formalize the notion of corresponding updates as follows. Let π be an isomorphism from a structure $\mathcal{S}$ to a structure $\mathcal{T}$. Two updates $\delta(\vec{a})$ on $\mathcal{S}$ and $\delta(\vec{b})$ on $\mathcal{T}$ are said to be π-*respecting* if $\vec{b} = \pi(\vec{a})$. Two sequences $\alpha = \delta_1 \cdots \delta_m$ and $\beta = \delta'_1 \cdots \delta'_m$ of updates respect π if, for every $i \leq m$, δ_i and δ'_i are π-respecting.

Lemma 2 (Substructure Lemma). *Let $\mathcal{P}$ be a* DynProp *program and $\mathcal{S}$ and $\mathcal{T}$ states of $\mathcal{P}$ with domains S and T, respectively. Further, let $S' \subseteq S$ and $T' \subseteq T$ such that $\mathcal{S} \restriction S'$ and $\mathcal{T} \restriction T'$ are isomorphic via π. Then $P_\alpha(\mathcal{S}) \restriction S'$ and $P_\beta(\mathcal{T}) \restriction T'$ are isomorphic via π for all π-respecting update sequences α, β on S' and T'.*

The Substructure Lemma can be applied along the following lines to prove that Dyn(s-t-Reach) cannot be maintained in some settings with quantifier-free updates. Towards a contradiction, assume that there is a quantifier-free program $\mathcal{P} = (P, \text{Init}, Q)$ that maintains Dyn(s-t-Reach). Then, find

- two states $\mathcal{S}$ and $\mathcal{T}$ occurring as states[4] of $\mathcal{P}$ with current graphs $G_\mathcal{S}$ and $G_\mathcal{T}$;
- substructures $\mathcal{S}'$ and $\mathcal{T}'$ of $\mathcal{S}$ and $\mathcal{T}$ isomorphic via π; and
- two π-respecting update sequences α and β such that $\alpha(G_\mathcal{S})$ is in s-t-Reach and $\beta(G_\mathcal{T})$ is not in s-t-Reach.

This yields the desired contradiction, since Q has the same value in $P_\alpha(\mathcal{S})$ and $P_\beta(\mathcal{T})$ by the Substructure Lemma.

How such states $\mathcal{S}$ and $\mathcal{T}$ can be obtained depends on the particular setting. Yet, Ramsey's Theorem and Higman's Lemma often prove to be useful for this task. Next, we present the variants of these theorems used in our proofs. We refer to [9] and [13, Proposition 2.5, page 3] for proofs.

Theorem 3 (Ramsey's Theorem for Structures). *For every schema τ and all natural numbers k and n there exists a number $R_{\tau,k}(n)$ such that, for every τ-structure $\mathcal{S}$ with domain A of size $R_{\tau,k}(n)$, every $\vec{d} \in A^k$ and every order $\prec$ on A, there is a subset B of A of size n with $B \cap \vec{d} = \emptyset$, such that, for every l, the type of $(\vec{a}, \vec{d})$ in $\mathcal{S}$ is the same, for all $\prec$-ordered l-tuples $\vec{a}$ over B.*

A word u is a *subsequence* of a word v, in symbols $u \sqsubseteq v$, if $u = u_1 \dots u_k$ and $v = v_0 u_1 v_1 \dots v_{k-1} u_k v_k$ for some words $u_1, \dots, u_k$ and $v_0, \dots, v_k$.

Theorem 4 (Higman's Lemma). *For every alphabet of size c and function $g : \mathbb{N} \to \mathbb{N}$ there is a natural number $H(c)$ such that in every sequence $(w_i)_{1 \le i \le H(c)}$ of $H(c)$ many words with $|w_i| \le g(i)$ there are l and k with $l < k$ and $w_l \sqsubseteq w_k$.*

Before turning towards lower bounds for arbitrary initialization, we state a lower bound for the restricted setting of invariant initialization. Intuitively lower bounds in this setting can be obtained easier because invariant initialization cannot generate complex initial auxiliary structures such as lists from simple-structured input databases.

Theorem 5. Dyn(s-t-Reach) *cannot be maintained in* DynProp *with invariant initialization mapping and empty built-in schema. This holds even for 1-layered s-t-graphs.*

[4] I.e. $\mathcal{S} = \mathcal{P}_\delta(\mathcal{S}_{\text{Init}}(G))$ for some s-t-graph G, and likewise for $\mathcal{T}$.

4.1 A Binary Lower Bound

As already mentioned in the introduction, the proof that Dyn(s-t-Reach) is not in unary DynFO in [3] uses constant-length update sequences, and is mainly an application of a locality-based static lower bound for monadic second order logic. This technique does not seem to generalize to binary DynFO. We prove the first unmaintainability result for Dyn(s-t-Reach) with respect to binary auxiliary relations.

Theorem 6. Dyn(s-t-Reach) *is not in binary* DynProp.

The proof of Theorem 6 will actually show that binary DynProp cannot even maintain Dyn(s-t-Reach) on 2-layered s-t-graphs. These restricted graphs will then help us to separate binary DynProp from ternary DynProp. This separation shows that the lower bound technique for binary DynProp does not immediately transfer to ternary DynProp. At the moment we do not know whether it is possible to adapt the technique to full DynProp.

The following notion of homogeneous sets is used in the proof of Theorem 6. Let $\mathcal{S}$ be a structure of some schema τ and A, B disjoint subsets of the domain of $\mathcal{S}$. We say that B is *A-$\prec$-homogeneous up to arity m*, if for every $l \leq m$, all tuples $(a, \vec{b})$, where $a \in A$ and $\vec{b}$ is an $\prec$-ordered l-tuple over B, have the same type. We may drop the order $\prec$ from the notation if it is clear from the context, and we may drop A if $A = \emptyset$. We observe that if the maximal arity of τ is m and B is A-homogeneous up to arity m, then B is A-homogeneous up to arity m' for every m'. In this case we simply say B is *A-homogeneous*.

Lemma 7. *For every schema τ and natural number n, there is a natural number $R^{hom}_\tau(n)$ such that for any two disjoint subsets A, B of the domain of a τ-structure $\mathcal{S}$ with $|A|, |B| \geq R^{hom}_\tau(n)$, there are subsets $A' \subseteq A$ and $B' \subseteq B$ such that $|A'|, |B'| = n$ and B' is A'-homogeneous in $\mathcal{S}$.*

Proof (of Theorem 6). Let us assume, towards a contradiction, that the dynamic program (P, Init, Q) over schema $\tau = (\tau_{\text{in}}, \tau_{\text{aux}}, \tau_{\text{bi}})$ with binary τ_{aux} maintains the dynamic s-t-reachability query for 2-layered s-t-graphs. We choose numbers n, n_1, n_2 and n_3 such that n_3 is sufficiently large with respect to τ, n_2 is sufficiently large with respect to n_3, n_2 is sufficiently large with respect to n_1 and n is sufficiently large with respect to n_1.

Let $G = (V, E)$ be a 2-layered s-t-graph with layers A, B, where A and B are both of size n and $E = \{(b, t) \mid b \in B\}$. Further, let $\mathcal{S} = (V, E, \mathcal{A}, \mathcal{B})$ be the state obtained by applying Init to G.

We will first choose homogeneous subsets. By Lemma 7 and because n is sufficiently large, there are subsets A_1 and B_1 such that $|A_1| = |B_1| = n_1$ and B_1 is A_1-$\prec$-homogeneous in $\mathcal{S}$, for some order $\prec$. Next, let A_2 and B_2 be arbitrarily chosen subsets of A_1 and B_1, respectively, of size $|B_2| = n_2$ and $|A_2| = 2^{|B_2|}$, respectively. We note that B_2 is still A_2-homogeneous. In particular, B_2 is still A_2-homogeneous with respect to schema τ_{bi}. We associate with every subset $X \subseteq B_2$ a unique vertex a_X from A_2 in an arbitrary fashion.

Now,we define the update sequence α as follows.

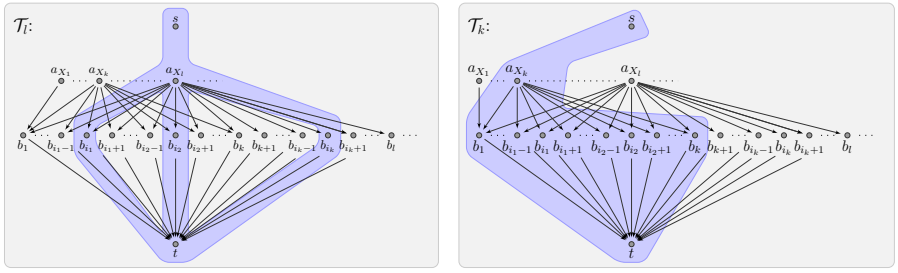

Fig. 2. The structure $\mathcal{S}'$ from the proof of Theorem 6. The isomorphic substructures $\mathcal{T}_k$ and $\mathcal{T}_l$ are highlighted in blue.

(α) For every subset X of B_2 and every $b \in X$ insert an edge (a_X, b), in some arbitrarily chosen order.

Let $\mathcal{S}' \stackrel{\text{def}}{=} (V, E', \mathcal{A}', \mathcal{B})$ be the state of $\mathcal{P}$ after applying α to $\mathcal{S}$, i.e. $\mathcal{S}' = P_\alpha(\mathcal{S})$. We observe that the built-in data has not changed, but the auxiliary data might have changed. In particular, B_2 is not necessarily A_2-homogeneous with respect to schema τ_{aux} in state $\mathcal{S}'$.

Our plan is to exhibit two sets X, X' such that $X \subsetneq X' \subseteq B_2$ such that the restriction of $\mathcal{S}'$ to $\{s, t, a_{X'}\} \cup X'$ contains an isomorphic copy of $\mathcal{S}'$ restricted to $\{s, t, a_X\} \cup X$. Then the Substructure Lemma will easily give us a contradiction.

By Ramsey's Theorem and because $|B_2|$ is sufficiently large with respect to n_2, there is a subset $B_3 \subseteq B_2$ of size n_3 such that B_3 is $\prec$-homogeneous in $\mathcal{S}'$. Let $b_1 \prec \ldots \prec b_{n_3}$ be an enumeration of the elements of B_3 and let $X_i \stackrel{\text{def}}{=} \{b_1, \ldots, b_i\}$, for every $i \in \{1, \ldots, n_3\}$.

Let $\mathcal{S}'_i$ denote the restriction of $\mathcal{S}'$ to $X_i \cup \{s, t, a_{X_i}\}$. For every i, we construct a word w_i of length i, that has a letter for every node in X_i and captures all relevant information about those nodes in $\mathcal{S}'_i$. More precisely, $w_i \stackrel{\text{def}}{=} \sigma_i^1 \cdots \sigma_i^i$, where for every i and j, σ_i^j is the binary type of (a_{X_i}, b_j).

Since B_3 is sufficiently large with respect to τ_{aux}, it follows, by Higman's Lemma, that there are k and l such that $k < l$ and $w_k \sqsubseteq w_l$, that is $w_k = \sigma_k^1 \sigma_k^2 \ldots \sigma_k^k = \sigma_l^{i_1} \sigma_l^{i_2} \ldots \sigma_l^{i_k}$ for suitable numbers $i_1 < \ldots < i_k$. Let $\vec{b} \stackrel{\text{def}}{=} (b_1, \ldots, b_k)$ and $\vec{b'} \stackrel{\text{def}}{=} (b_{i_1}, \ldots, b_{i_k})$. Further, let $\mathcal{T}_k \stackrel{\text{def}}{=} \mathcal{S}'_k \restriction T_k$ where $T_k = \{s, t, a_{X_k}\} \cup \vec{b}$, and $\mathcal{T}_l \stackrel{\text{def}}{=} \mathcal{S}'_l \restriction T_l$ where $T_l \stackrel{\text{def}}{=} \{s, t, a_{X_l}\} \cup \vec{b'}$. We refer to Figure 2 for an illustration of the substructures $\mathcal{T}_k$ and $\mathcal{T}_l$ of $\mathcal{S}'$.

It can be shown that $\mathcal{T}_k \simeq_\pi \mathcal{T}_l$, where π is the isomorphism that maps s and t to themselves, a_{X_k} to a_{X_l} and b_j to b_{i_j} for every $j \in \{1, \ldots, k\}$.

Thus, by the Substructure Lemma, application of the following two update sequences to $\mathcal{S}'$ results in the same query result:

(β_1) Deleting edges $(a_{X_k}, b_1), \ldots, (a_{X_k}, b_k)$ and adding an edge (s, a_{X_k}).
(β_2) Deleting edges $(a_{X_l}, b_{i_1}), \ldots, (a_{X_l}, b_{i_k})$ and adding an edge (s, a_{X_l}).

However, applying β_1 yields a graph in which t is not reachable from s, whereas by applying β_2 a graph is obtained in which t is reachable from s. This is the desired contradiction. □

4.2 Separating Low Arities

An arity hierarchy for DynFO was established in [3]. The dynamic queries $\mathcal{Q}_{k+1}$ used to separate k-ary and $(k+1)$-ary DynFO can already be maintained in $(k+1)$-ary DynProp, thus the hierarchy transfers to DynProp immediately. However, $\mathcal{Q}_{k+1}$ is a k-ary query and has an input schema of arity $6k+1$ (improved to $3k+1$ in [14]). Here we establish a strict arity hierarchy between unary, binary and ternary DynProp for Boolean queries and binary input schemas.

We use the problems s-t-TwoPath, where one asks whether there is a path of length two from s to t in a given s-t-graph G, and the problem s-TwoPath where one asks whether there is *any* path of length 2 starting in s.

Proposition 8. *The dynamic query* Dyn(s-t-TwoPath) *is in binary* DynProp*, but not in unary* DynProp*.*

Proposition 9. *The dynamic query* Dyn(s-TwoPath) *is in ternary* DynProp*, but not in binary* DynProp*.*

5 Lower Bounds with Auxiliary Functions

In this section we consider the extension of the quantifier-free update formalism by auxiliary functions. Recall that DynProp-update formulas have access only to the inserted or deleted tuple $\vec{a}$ and the currently updated tuple of an auxiliary relation $\vec{b}$. When auxiliary functions are allowed in update formulas, further elements of the structure can be accessed by function application. This can be seen as adding weak quantification to quantifier-free formulas. The class of dynamic queries that can be maintained with quantifier-free update formulas and auxiliary functions is denoted DynQF.

DynQF is strictly more expressive than DynProp. E.g., it contains all Dyck languages, among other non-regular languages [6]. Further, undirected reachability can be maintained in DynQF with built-in relations [5].

Lists can be represented by unary functions in a straightforward way. Therefore, it is not surprising that the upper bound of Proposition 8 already holds for unary DynProp with unary built-in functions.

Proposition 10. Dyn(s-t-Reach) *on 1-layered s-t-graphs can be maintained in unary* DynProp *with unary built-in functions.*

Yet, unary DynQF cannot maintain the reachability query. Also Theorem 5 can be extended to quantifier-free programs with auxiliary functions.

Theorem 11. Dyn(s-t-Reach) *is not in unary* DynQF*.*

Theorem 12. Dyn(s-t-Reach) *cannot be maintained in* DynQF *with invariant initialization mapping and empty built-in schema. This holds even for 1-layered s-t-graphs.*

6 Lower Bounds for Other Dynamic Queries

Lower bounds for the dynamic variants of the k-Clique and k-Col problems (where k is fixed) can be established via reductions to the dynamic s-t-reachability query for shallow graphs.

Proposition 13. *The dynamic query* Dyn(k-Clique)*, for $k \geq 3$, and the dynamic query* Dyn(k-Col)*, for $k \geq 2$, are not in binary* DynProp*.*

Proposition 14. *The dynamic query* Dyn(k-Clique)*, for $k \geq 3$, and the dynamic query* Dyn(k-Col)*, for $k \geq 2$, cannot be maintained in* DynQF *with invariant initialization mapping.*

7 Normal forms for Dynamic Programs

In this section, we give normal forms for dynamic programs. The study of normal forms has a long tradition in logics. Normal forms are often helpful in proofs based on the structure of formulas and yield insights for the construction of algorithms.

A formula is *negation-free* if it does not use negation at all. A formula is *conjunctive* if it is a conjunction of (positive or negated) literals. A dynamic program is negation-free (conjunctive, respectively) if all its update formulas are negation-free (conjunctive, respectively). Two dynamic programs $\mathcal{P}$ and $\mathcal{P}'$ are equivalent, if they maintain the same query. The results in this section allow arbitrary initialization but no auxiliary functions. The first theorem is a straightforward generalization of Theorem 6.6 from [5] which states this observation for a subclass of DynProp.

Theorem 15. *(a) Every* DynFO*-program has an equivalent negation-free* DynFO*-program.*
(b) Every DynProp*-program has an equivalent negation-free* DynProp*-program.*

Theorem 16. *Every* DynProp*-program has an equivalent conjunctive* DynProp*-program.*

8 Future Work

The question whether Reachability is maintainable with first-order updates remains one of the major open questions in dynamic complexity. Proving that Reachability cannot be maintained with quantifier-free updates with arbitrary auxiliary data seems to be a worthwhile intermediate goal, but it appears nontrivial as well.

We contributed to the intermediate goal by giving a first lower bound for binary auxiliary relations. Whether the strictness of the arity hierarchy for DynProp extends beyond arity three is another open question.

For (full) first-order updates a major challenge is the development of lower bound tools. Current techniques are in some sense not fully dynamic: either results from static descriptive complexity are applied to constant-length update sequences; or non-constant but very regular update sequences are used. In the latter case, the updates do not depend on previous changes to the auxiliary data (as, e.g., in [7] and in this paper). Finding techniques that adapt to changes could be a good starting point.

The normal forms obtained for DynProp give hope that some fragments of DynFO collapse. Therefore, we plan to study normal forms for DynFO extensively. One interesting question being which fragments of DynFO can be captured by a conjunctive query normal form.

References

[1] Patnaik, S., Immerman, N.: Dyn-FO: A parallel, dynamic complexity class. In: PODS, ACM Press (1994) 210–221
[2] Hesse, W.: The dynamic complexity of transitive closure is in DynTC^0. In: ICDT. (2001) 234–247
[3] Dong, G., Su, J.: Arity bounds in first-order incremental evaluation and definition of polynomial time database queries. J. Comput. Syst. Sci. **57**(3) (1998) 289–308
[4] Dong, G., Libkin, L., Wong, L.: Incremental recomputation in local languages. Inf. Comput. **181**(2) (2003) 88–98
[5] Hesse, W.: Dynamic Computational Complexity. PhD thesis, University of Massachusetts Amherst (2003)
[6] Gelade, W., Marquardt, M., Schwentick, T.: The dynamic complexity of formal languages. In: STACS. (2009) 481–492
[7] Grädel, E., Siebertz, S.: Dynamic definability. In: ICDT. (2012) 236–248
[8] Patrascu, M., Demaine, E.D.: Lower bounds for dynamic connectivity. In Babai, L., ed.: STOC, ACM (2004) 546–553
[9] Zeume, T., Schwentick, T.: On the quantifier-free dynamic complexity of reachability. CoRR **abs/1306.3056** (2013) `http://arxiv.org/abs/1306.3056`.
[10] Weber, V., Schwentick, T.: Dynamic complexity theory revisited. Theory Comput. Syst. **40**(4) (2007) 355–377
[11] Gelade, W., Marquardt, M., Schwentick, T.: The dynamic complexity of formal languages. ACM Trans. Comput. Log. **13**(3) (2012) 19
[12] Etessami, K.: Dynamic tree isomorphism via first-order updates. In: PODS, ACM Press (1998) 235–243
[13] Schmitz, S., Schnoebelen, P.: Multiply-recursive upper bounds with Higman's lemma. In: ICALP vol. 2. (2011) 441–452
[14] Dong, G., Zhang, L.: Separating auxiliary arity hierarchy of first-order incremental evaluation systems using (3k+1)-ary input relations. Int. J. Found. Comput. Sci. **11**(4) (2000) 573–578