On the quantifier-free dynamic complexity of Reachability*,†

Thomas Zeume and Thomas Schwentick

June 22, 2014

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 inexpressibility results are given for the reachability query in a different setting as well as for a syntactical restriction of quantifier-free update formulas. Moreover inexpressibility results for some other queries are presented.

1 Introduction

In modern data management scenarios data is subject to frequent changes. In order to avoid costly re-computations of queries from scratch after each small modification of the data, one can try to (re-)use auxiliary data structures that have been already computed before. However, these 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 [10] models this setting. It was mainly inspired by updates in relational databases. Within this framework, for a relational database subject to change, auxiliary relations are maintained with the intention to help answering a query $\mathcal{Q}$. When a modification to the database, an insertion or deletion of a tuple, occurs, every auxiliary relation is updated through a first-order query (or, equivalently, through a core SQL query) that

*An extended abstract of this article appeared in Proceedings of the conference Mathematical Foundations of Computer Science 2013 (MFCS 2013).

†Both authors acknowledge the financial support by the German DFG under grant SCHW 678/6-1.

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 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 [8] and with plain first-order update formulas on both acyclic graphs and undirected graphs [10]. 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 so far. 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.

The study of bounded arity auxiliary relations was started in [2] 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, a strict hierarchy has not yet been established for queries on graphs. In [1] it was shown that unary auxiliary relations are not sufficient to maintain Reachability for update formulas of any logic with certain locality properties. The proofs strongly use the "static" weakness of local logics and do not fully exploit the dynamic setting, as they only require modification sequences of constant length.

The second line of research was initiated by Hesse [9]. 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 [5]: the alternating reachability query (on graphs with $\wedge$- and $\vee$-nodes) is not maintainable in DynProp. Furthermore, on strings, DynProp

[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.

exactly captures 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 [2] 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 relations vs. unary relations). We establish a strict hierarchy within 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 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 modification 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 [6], 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 [6].

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

In [15] it was shown that every query in DynProp can be maintained by a program with negation-free quantifier-free formulas only as well as by a program with disjunction-free quantifier-free formulas only. Thus lower bounds for those syntactic fragments immediately yield lower bounds for DynProp itself. Here, we show that Reachability cannot be maintained by DynProp programs with update formulas that are disjunction- *and* negation-free.

A preliminary version of this work appeared in [15]. It was without most of the proofs and did not contain the lower bound for disjunction- *and* negation-free DynProp programs. The proofs of the normal form results obtained in [15] will be included in the long version of [17]. The latter work establishes normal forms for variants of dynamic conjunctive queries, complementing the normal forms for DynProp.

Related Work We already described the most closely related work. As mentioned 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 [12].

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 a lower bound for a syntactical fragment of DynProp in Section 7.

Acknowledgement We thank Ahmet Kara and Martin Schuster for careful proofreading.

2 Preliminaries

In this section, we repeat some basic notions and fix some of our notation.

A *domain* is a finite set. For k-tuples, $\vec{a} = (a_1, \ldots, a_k)$ and $\vec{b} = (b_1, \ldots, b_k)$ over some domain D, the $2k$-tuple obtained by concatenating $\vec{a}$ and $\vec{b}$ is denoted by $(\vec{a}, \vec{b})$. The tuple $\vec{a}$ is $\prec$-*ordered* with respect to an order $\prec$ of D, if $a_1 \prec \ldots \prec a_k$. If π is a function[2] on D, we denote $(\pi(a_1), \ldots, \pi(a_k))$ by $\pi(\vec{a})$. We slightly abuse set theoretic notations and write $c \in \vec{a}$ if $c = a_i$ for some $c \in D$ and some i, and $\vec{a} \cup \vec{b}$ for the set $\{a_1, \ldots, a_k, b_1 \ldots, b_k\}$.

A (relational) *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 $\text{Ar} : \tau_{\text{rel}} \to \mathbb{N}$. 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 $c \in \tau_{\text{const}}$ a single element (called *constant*) from D. The *size of a database* is the size of its domain. Unless otherwise stated (as, e.g., in Section 5), we always consider relational schemas.

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 $\mathcal{S} \restriction D'$.

Let $\mathcal{S}$ and $\mathcal{T}$ be two structures of schema τ and over domains S and T, respectively. A mapping $\pi : S \mapsto T$ *preserves* a relation symbol $R \in \tau$ of arity m, when $\vec{a} \in R^{\mathcal{S}}$ if and only if $\pi(\vec{a}) \in R^{\mathcal{T}}$, for all m-tuples $\vec{a}$. It preserves a constant symbol $c \in \tau$, if $c^{\mathcal{T}} = \pi(c^{\mathcal{S}})$. The mapping is τ-preserving, if it preserves all relation symbols and all constant symbols from τ. Two τ-structures $\mathcal{S}$ and $\mathcal{T}$ are *isomorphic via* π, denoted by $\mathcal{S} \simeq_\pi \mathcal{T}$, if π is a bijection from S to T which is τ-preserving. We define $id[\vec{a}, \vec{b}] : S \to S$ to be the bijection that maps, for every i, a_i to b_i and b_i to a_i, and maps all other elements to themselves.

An *atomic formula* is a formula of the form $R(z_1, \ldots, z_l)$ where R is a relation symbol and each z_i is either a variable or a constant symbol. The k-ary atomic

[2]Throughout this work all functions are total.

type $\langle \mathcal{S}, \vec{a} \rangle$ 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. As we only consider atomic types in this paper, we will often simply say type instead of atomic type. The σ-type $\langle \mathcal{S}, \vec{a} \rangle_\sigma$ is the set of atomic formulas of $\langle \mathcal{S}, \vec{a} \rangle$ with relation symbols from σ. If $\prec$ is a linear order on D we call a subset $D' \subseteq D$ *$\prec$-homogeneous* (or *homogeneous*, if $\prec$ is clear from the context) if, for every l, the type of all $\prec$-ordered l-tuples over D' is the same, that is if $\langle \mathcal{S}, \vec{a} \rangle = \langle \mathcal{S}, \vec{b} \rangle$ for all ordered l-tuples $\vec{a}$ and $\vec{b}$. It is easy to observe, that a set D' is already $\prec$-homogeneous if the condition holds for every l up to the maximal arity of τ.

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 *reachability query* Reach on graphs is defined as usual, that is (a, b) is in $\textsc{Reach}(G)$ if b can be reached from a in G. 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 $(s, t) \in \textsc{Reach}(G)$.

Formally, an s-t-graph is a structure over a schema with one binary relation symbol (interpreted by the set of edges E) and two constant symbols (interpreted by the two distinguished nodes s and t).

3 Dynamic Queries and Programs

The following presentation follows [14] and [5].

Informally a *dynamic instance* of a static query $\mathcal{Q}$ is a pair $(\mathcal{D}, \alpha)$, where $\mathcal{D}$ is a database and α is a sequence of modifications, i.e. a sequence of tuple insertions and deletions into $\mathcal{D}$. The dynamic query $\textsc{Dyn}(\mathcal{Q})$ yields as result the relation that is obtained by first applying the modifications from α to $\mathcal{D}$ and evaluating query $\mathcal{Q}$ on the resulting database. We formalize this as follows.

Definition 1. (Abstract and concrete modifications) The set Δ of *abstract modifications* of a schema τ contains the terms $\textsc{ins}_R$ and $\textsc{del}_R$, for every relation symbol[3] $R \in \tau$. For a database $\mathcal{D}$ over schema τ with domain D, a *concrete modification* is a term of the form $\textsc{ins}_R(\vec{a})$ or $\textsc{del}_R(\vec{a})$ where $R \in \tau$ is a k-ary relation symbol and $\vec{a}$ is a k-tuple of elements from D.

Applying a modification $\textsc{ins}_R(\vec{a})$ to a database $\mathcal{D}$ replaces relation $R^{\mathcal{D}}$ by $R^{\mathcal{D}} \cup \{\vec{a}\}$. Analogously, applying a modification $\textsc{del}_R(\vec{a})$ replaces $R^{\mathcal{D}}$ by $R^{\mathcal{D}} \setminus \{\vec{a}\}$. All other relations remain unchanged. The database resulting from applying a modification δ to a database $\mathcal{D}$ is denoted by $\delta(\mathcal{D})$. The result $\alpha(\mathcal{D})$ of applying a sequence of modifications $\alpha = \delta_1 \ldots \delta_m$ to a database $\mathcal{D}$ is defined by $\alpha(\mathcal{D}) \stackrel{\text{def}}{=} \delta_m(\ldots(\delta_1(\mathcal{D}))\ldots)$.

[3] In this work we do not allow modification of constants, for simplicity.

Definition 2. (Dynamic Query) A *dynamic instance* is a pair $(\mathcal{D}, \alpha)$ consisting of an *input database* $\mathcal{D}$ and a *modification sequence* α. For a static query $\mathcal{Q}$ with schema τ, the *dynamic query* $\textsc{Dyn}(\mathcal{Q})$ is the mapping that yields $\mathcal{Q}(\alpha(\mathcal{D}))$, for every dynamic instance $(\mathcal{D}, \alpha)$.

Our main interest in this work is the dynamic version $\textsc{Dyn}(s\text{-}t\text{-}\textsc{Reach})$ of the s-t-reachability query.

Dynamic programs, to be defined next, consist of an initialization mechanism and an update[4] program. The former yields, for every database $\mathcal{D}$ an initial state with initial auxiliary data (and possibly with further built-in data). The latter defines the new state, for each possible modification δ. The following formal definitions are illustrated in Example 1 at the end of this section.

An *dynamic schema* is a triple $(\tau_{\text{in}}, \tau_{\text{aux}}, \tau_{\text{bi}})$ of schemas of the input database, the auxiliary database, and the built-in database and respectively. We always let $\tau \stackrel{\text{def}}{=} \tau_{\text{in}} \cup \tau_{\text{aux}} \cup \tau_{\text{bi}}$. Throughout the paper, τ_{in} has to be relational. In our basic setting we also require τ_{aux} to be relational (this will be relaxed in Section 5).

A note on the role of the built-in database is in order: as opposed to the auxiliary database, the built-in database never changes throughout a "computation". Our standard classes are defined over schemas without built-in databases (that is, with empty built-in schema). Built-in databases are only used to strengthen some results in one of two possible ways, (1) by showing upper bounds in which (some) auxiliary relations or functions need not be updated or (2) by showing inexpressibility results that hold for auxiliary schemas of bounded arity but with built-in relations of unbounded arity. In general, built-in data can be "simulated" by auxiliary data. However, this need not hold, e.g., if the auxiliary schema is more restricted than the built-in schema.

Definition 3. (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 abstract modification δ of some $S \in \tau_{\text{in}}$, an update formula $\phi_\delta^R(\vec{x}; \vec{y})$ over the schema τ where $\vec{x}$ and $\vec{y}$ have the same arity as S and R, respectively.

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 a modification $\delta(\vec{a})$ and program state $\mathcal{S} = (D, \mathcal{I}, \mathcal{A}, \mathcal{B})$ we denote by $P_\delta(\mathcal{S})$ the state $(D, \delta(\mathcal{I}), \mathcal{A}', \mathcal{B})$,

[4] In previous work (by us as well as by others) there was usually no terminological distinction between the changes that are applied to the structure at hand (e.g., database or graph) and are considered as input to an update program and the changes that are applied by an update program to the auxiliary data after such a change. Both types of changes usually have been termed *updates*. In this article, we use the term *modification* for changes of the database or structure and reserve the term *update* for the respective change applied to the auxiliary data by the actual update program.

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 a modification sequence $\alpha = \delta_1 \dots \delta_m$ to a state $\mathcal{S}$ is the state $P_{\delta_m}(\dots(\mathcal{P}_{\delta_1}(\mathcal{S}))\dots)$.

Definition 4. (Dynamic program) A *dynamic program* is a triple $(P, \textsc{Init}, Q)$, where

- P is an update program over some dynamic schema $(\tau_{\text{in}}, \tau_{\text{bi}}, \tau_{\text{aux}})$,
- 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 $\mathcal{S} \stackrel{\text{def}}{=} P_\alpha(\mathcal{S}_{\textsc{Init}}(\mathcal{D}))$, 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 [10, 4, 6, 5]. 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 $\textsc{Init}$ is *permutation-invariant* (short: *invariant*) in the sense that $\pi(\textsc{Init}_{\text{aux}}(\mathcal{D})) = \textsc{Init}_{\text{aux}}(\pi(\mathcal{D}))$ and $\pi(\textsc{Init}_{\text{bi}}(D)) = \textsc{Init}_{\text{bi}}(\pi(D))$ hold, for every database $\mathcal{D}$, domain D and permutation π of the domain; and
- whether there are any built-in relations at all.

In [11], Dyn-FO is defined as the class of (Boolean) queries that can be maintained for empty initial databases with first-order update formulas, first-order definable initialization mapping and without built-in data. Furthermore, a larger class with polynomial-time computable initialization mapping was considered. Also [4] considers empty initial databases without built-in data. In [6], general instances (with non-empty initial databases) are allowed, but the initialization mapping has to be defined by logical formulas and is thus always invariant; and there is no built-in data. In [5] update formulas are restricted to be quantifier-free, the initial database is empty and a built-in order is available.

In this article, the main dynamic classes do not allow built-in data. We call a dynamic schema *normal* if it has an empty built-in schema τ_{bi}.

We consider the following basic dynamic complexity classes.

Definition 5. ($\textsc{DynFO}$, $\textsc{DynProp}$) $\textsc{DynFO}$ is the class of all dynamic queries maintainable by dynamic programs with first-order update formulas over normal dynamic schemas. $\textsc{DynProp}$ is the subclass of $\textsc{DynFO}$, where update formulas do not use quantifiers. A dynamic program is *k-ary* if the arity of its auxiliary

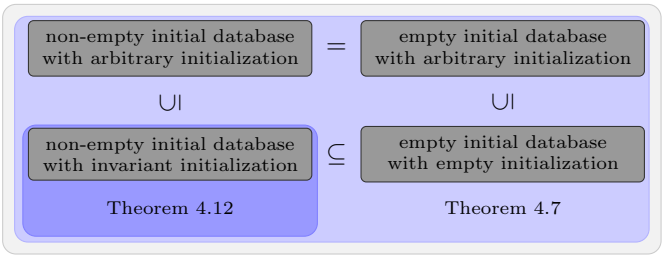

Figure 1: Relationship between different dynamic settings considered in the literature. Inclusion is with respect to the class of queries that can be maintained for a fixed (arbitrary) update language. Theorem 4.7 holds for all settings, Theorem 4.12 only for the lower left setting.

relation symbols is at most k. By k-ary DynProp (resp. DynFO) we refer to dynamic queries that can be maintained with k-ary dynamic programs.

At times we also consider dynamic programs with non-empty relational built-in schemas. We denote the extension of a dynamic class by programs with non-empty built-in schemas by a superscript *, as in DynProp*. We note that the arity restrictions in the above definition do not apply to the built-in relations.

In our basic setting the initialization mappings can be arbitrary. We will explicitly state when we relax this most general setting. Now we sketch important relaxations. Figure 1 illustrates the relationships between the various settings.

First we note that for arbitrary initialization mappings, the same queries can be maintained regardless whether one starts from an empty or from a non-empty initial database.[5] Restricting the setting for non-empty initial databases to invariant auxiliary data initialization leads to the initialization used in [6] (called *invariant initialization* in the following). For empty initial databases, allowing empty initial auxiliary data only leads to the initialization model of [11, 4] (called *empty initialization* in the following).

It is easy to see that applying an invariant initialization mapping to an empty database is pretty much useless, as, all tuples with the same constants at the same positions are treated in the same way. Therefore, queries maintainable in DynFO or DynProp with empty initial database and invariant initialization can also be maintained with empty initialization[6]. This statement also holds in the presence of arbitrary built-in relations.

From now on we restrict our attention to quantifier-free update programs. Next, we give an example of such a program.

Example 1. We provide a DynProp-program $\mathcal{P}$ 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. Of course, this query is trivially expressible in first-order logic, but not without quantifiers. The

[5]The initialization for a non-empty database can be obtained as the auxiliary relations obtained after inserting all tuples of the database into the empty one.

[6]We do not formally prove this here.

program $\mathcal{P}$ illustrates a technique to maintain lists with quantifier-free dynamic programs, introduced in [5, Proposition 4.5], which is used in some of our upper bounds.

The program $\mathcal{P}$ 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.

Insertion of a into U. A newly inserted element is attached to the end of the list[7]. Therefore the First-relation does not change except when the first element is inserted into an empty set U. Furthermore, the inserted element is the new last element of the list and has a connection to the former last element. Finally, after inserting an element into U, the query result is 'true':

$$\begin{aligned}
\phi^{\textsc{First}}_{\text{INS}}(a; x) &\stackrel{\text{def}}{=} (\neg Q \wedge a = x) \vee (Q \wedge \textsc{First}(x)) \\
\phi^{\textsc{Last}}_{\text{INS}}(a; x) &\stackrel{\text{def}}{=} a = x \\
\phi^{\textsc{List}}_{\text{INS}}(a; x, y) &\stackrel{\text{def}}{=} \textsc{List}(x, y) \vee (\textsc{Last}(x) \wedge a = y) \\
\phi^{Q}_{\text{INS}}(a) &\stackrel{\text{def}}{=} \top.
\end{aligned}$$

Deletion of a from U. How a deleted element a is removed from the list, depends on whether a is the first element of the list, the last element of the list or some other element of the list. The query bit remains 'true', if a was not the first *and* last element of the list.

$$\begin{aligned}
\phi^{\textsc{First}}_{\text{DEL}}(a; x) &\stackrel{\text{def}}{=} (\textsc{First}(x) \wedge a \neq x) \vee (\textsc{First}(a) \wedge \textsc{List}(a, x)) \\
\phi^{\textsc{Last}}_{\text{DEL}}(a; x) &\stackrel{\text{def}}{=} (\textsc{Last}(x) \wedge a \neq x) \vee (\textsc{Last}(a) \wedge \textsc{List}(x, a)) \\
\phi^{\textsc{List}}_{\text{DEL}}(a; x, y) &\stackrel{\text{def}}{=} x \neq a \wedge y \neq a \wedge \big(\textsc{List}(x, y) \vee (\textsc{List}(x, a) \wedge \textsc{List}(a, y))\big) \\
\phi^{Q}_{\text{DEL}}(a) &\stackrel{\text{def}}{=} \neg(\textsc{First}(a) \wedge \textsc{Last}(a))
\end{aligned}$$

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) with quantifier-free update formulas.

[7] For simplicity we assume that only elements that are not already in U are inserted, the formulas given can be extended easily to the general case. Similar assumptions are made whenever necessary.

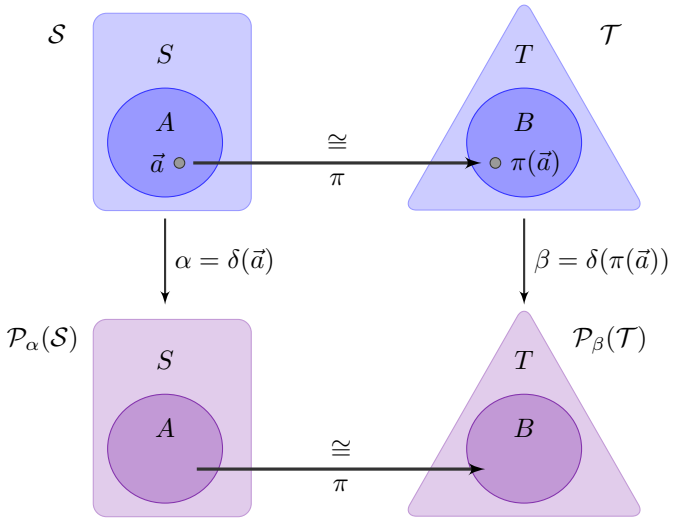

Figure 2: The statement of the substructure lemma.

First we introduce a tool for proving lower bounds for quantifier-free formulas. Afterwards we prove that

- Dyn(s-t-Reach) is not in binary DynProp*; and
- Dyn(s-t-Reach) is not in DynProp* with invariant initialization mappings.

The first result is used to obtain an arity hierarchy up to arity three for quantifier-free updates and binary queries.

The proofs use the following tool which is a slight variation of Lemma 1 from [5]. 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 modifications changes only tuples from a substructure $\mathcal{A}$ of $\mathcal{S}$, the auxiliary data of $\mathcal{A}$ is not affected by information outside $\mathcal{A}$. In particular, two isomorphic substructures $\mathcal{A}$ and $\mathcal{B}$ should remain isomorphic, when corresponding modifications are applied to them.

We formalize the notion of corresponding modifications as follows. Let π be an isomorphism from a structure $\mathcal{A}$ to a structure $\mathcal{B}$. Two modifications $\delta(\vec{a})$ on $\mathcal{A}$ and $\delta(\vec{b})$ on $\mathcal{B}$ 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 modifications respect π if, for every $i \leq m$, δ_i and δ'_i are π-respecting.

Lemma 4.1 (Substructure lemma for DynProp*)**.** *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 $A \subseteq S$ and $B \subseteq T$ such that $\mathcal{S} \restriction A$ and $\mathcal{T} \restriction B$ are isomorphic via π. Then $P_\alpha(\mathcal{S}) \restriction A$ and $P_\beta(\mathcal{T}) \restriction B$ are isomorphic via π for all π-respecting modification sequences α, β on A and B.

The substructure lemma is illustrated in Figure 2.

Proof. The lemma can be shown by induction on the length of the modification sequences. To this end, it is sufficient to prove the claim for a pair of π-respecting modifications $\delta(\vec{a})$ and $\delta(\vec{b})$ on A and B. We abbreviate $\mathcal{S} \restriction A$ and $\mathcal{T} \restriction B$ by $\mathcal{A}$ and $\mathcal{B}$, respectively.

Since π is an isomorphism from $\mathcal{A}$ to $\mathcal{B}$, we know that $R^{\mathcal{A}}(\vec{d})$ holds if and only if $R^{\mathcal{B}}(\pi(\vec{d}))$ holds, for every m-tuple $\vec{d}$ over A and every relation symbol $R \in \tau$. Therefore, $\varphi(\vec{x})$ evaluates to true in $\mathcal{A}$ under $\vec{d}$ if and only if it does so in $\mathcal{B}$ under $\pi(\vec{d'})$, for every quantifier-free formula $\varphi(\vec{x})$ over schema τ. Thus all update formulas from $\mathcal{P}$ yield the same result for corresponding tuples $\vec{d}$ and $\pi(\vec{d})$ from A and B, respectively. Hence $P_{\delta(\vec{a})}(\mathcal{S}) \restriction A$ is isomorphic to $P_{\delta(\pi(\vec{a}))}(\mathcal{S}) \restriction B$. This proves the claim. □

The following corollary is implied by Lemma 4.1, since the 0-ary auxiliary relations of two isomorphic structures coincide.

Corollary 4.2. *Let $\mathcal{P}$ be a* DynProp**-program with designated Boolean query symbol Q, and let $\mathcal{S}$ and $\mathcal{T}$ be states of $\mathcal{P}$ with domains S and T. Further let $A \subseteq S$ and $B \subseteq T$ such that $\mathcal{S} \restriction A$ and $\mathcal{T} \restriction B$ are isomorphic via π. Then Q has the same value in $P_\alpha(\mathcal{S})$ and $P_\beta(\mathcal{T})$ for all π-respecting sequences α, β of modifications on A and B.*

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[8] of $\mathcal{P}$ with current graphs $G_\mathcal{S}$ and $G_\mathcal{T}$;
- substructures $\mathcal{S} \restriction S'$ and $\mathcal{T}' \restriction T'$ of $\mathcal{S}$ and $\mathcal{T}$ isomorphic via π; and
- two π-respecting modification sequences α and β on S' and T' 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.

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

Theorem 4.3 (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.*

The proof of Theorem 4.3 uses the well-known Ramsey theorem for hypergraphs (see, e.g., [7, p. 7]) and is based on the proof of Observation 1' in [5, p. 11]. For the sake of completeness, the proof is presented in the following.

A *k-hypergraph* G is a pair (V, E) where V is a set and E is a set of k-element subsets of V. If E contains all k-element subsets of V, then G is called *complete*. A k-hypergraph $G' = (V', E')$ is a *sub-k-hypergraph* of a k-hypergraph $G = (V, E)$, if $V' \subseteq V$ and E' contains all edges $e \in E$ with $e \subseteq V'$. A *C-coloring* *col* of G, where C is a finite set of colors, is a mapping that assigns to every edge in E a color from C, that is, $col : E \to C$. A *C-colored k-hypergraph* is a pair (G, col) where G is a k-hypergraph and *col* is a C-coloring of G. If the name of the C-coloring is not important we also say *G is C-colored*.

Theorem 4.4. *(Ramsey's Theorem for Hypergraphs) For every set C of colors and natural numbers n and k there exists a number $R_C(n)$ such that, if the edges of a complete k-hypergraph of size $R_C(n)$ are C-colored, then the hypergraph contains a complete sub-k-hypergraph with n nodes whose edges are all colored with the same color.*

Proof (of Theorem 4.3). Given a schema τ and natural numbers k, n. Let $R_{\tau,k}(n)$ be chosen sufficiently large with respect to k, n, and τ such that the following argument works. Further let $\mathcal{S}$ be a τ-structure with domain A of size greater than $R_{\tau,k}(n)$ and $\prec$ an arbitrary order on A. Denote by m the maximal arity in τ and by $\vec{c}$ the constants of $\mathcal{S}$ in some order. Further denote by C the set of all constants and all elements occurring in $\vec{d}$.

Observe that proving the claim for $l \leq m$ is sufficient.

We first prove the claim for $|C| = 0$, by constructing inductively sets B_l that satisfy the condition for l with $l \leq m$. Let $B_0 = A$. The set B_l, $l \leq m$, is obtained from B_{l-1} as follows. From B_{l-1} a coloring *col* of the complete l-hypergraph G with node set B_{l-1} is constructed. The coloring *col* uses l-ary τ-types as colors. An edge $e = \{e_1, \ldots, e_l\}$ with $e_1 \prec \ldots \prec e_l$ is colored by the type $\langle \mathcal{S}, e_1, \ldots, e_l \rangle$. Because B_{l-1} is large, it has, by Ramsey's theorem, a subset B_l such that all edges $e \subseteq B_l$ of size l are colored with the same color by *col*. But then, by the definition of *col*, all $\prec$-ordered l-tuples over B_l have the same type in $\mathcal{S}$. By this construction we obtain a set B_m such that for every $l \leq m$ the type of all $\prec$-ordered l-tuples over B_m is the same. Setting $B := B_m$ proves the claim for $|C| = 0$.

The idea for the case $|C| \neq 0$ is to construct from $\mathcal{S}$ a new structure $\mathcal{S}'$ of an extended schema over domain $A' = A \setminus C$ such that $\mathcal{S}'$ encodes all information about C contained in $\mathcal{S}$ and then use the case $|C| = 0$ for $\mathcal{S}'$.

The structure $\mathcal{S}'$ is of schema $\tau \cup \tau'$, where τ' contains for every $l \leq m$ and every $(l + |C|)$-ary τ-type t, an l-ary relation symbol R_t. An l-tuple $\vec{a}$ is in $R_t^{\mathcal{S}'}$ if

and only if t is the τ-type of $(\vec{a}, \vec{C})$. Application of the case $|C| = 0$ to $\mathcal{S}'$ yields a huge homogeneous subset B' with respect to $\prec$ and schema $\tau \cup \tau'$. Then, for every $l \leq m$, the type of $(\vec{a}, \vec{C})$ in $\mathcal{S}$ is the same, for all $\prec$-ordered l-tuples $\vec{a}$ over B'. This proves the claim. □

Now we state the variant of Higman's Lemma that will be used later. 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 4.5 (Higman's Lemma)**.** *For every infinite sequence $(w_i)_{i \in \mathbb{N}}$ of words over an alphabet Σ there are l and k such that $l < k$ and $w_l \sqsubseteq w_k$.*

We will actually make use of the following stronger result. See e.g. [13, Proposition 2.5, page 3] for a proof.

Theorem 4.6. *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 $(w_i)_{1 \leq i \leq H(c)}$ of $H(c)$ many words with $|w_i| \leq g(i)$ there are l and k with $l < k$ and $w_l \sqsubseteq w_k$.*

In the following we will refer to both results as Higman's Lemma.

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 [2] uses constant-length modification 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. We recall that binary DynProp* can have built-in relations of arbitrary arity.

Theorem 4.7. Dyn(s-t-Reach) *is not in binary* DynProp*.

The proof of Theorem 4.7 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 show that binary DynProp* does not capture ternary DynProp. This separation shows that the lower bound technique for binary DynProp does not immediately transfer to ternary DynProp (or ternary DynProp*). At the moment we do not know whether it is possible to adapt the technique to full DynProp.

Before proving Theorem 4.7, we show the following corresponding result for unary DynProp* whose proof uses the same techniques in a simpler setting.

Proposition 4.8. *The dynamic s-t-reachability query is not in unary* DynProp**, not even for 1-layered s-t-graphs.*

Proof. Towards a contradiction, assume that $\mathcal{P} = (P, \text{Init}, Q)$ is a dynamic program over schema $\tau = (\tau_{\text{in}}, \tau_{\text{aux}}, \tau_{\text{bi}})$ with unary schema τ_{aux} that maintains the s-t-reachability query for 1-layered s-t-graphs. Let n' be sufficiently large[9]

[9] Explicit numbers are given at the end of the proof.

with respect to τ and n be sufficiently large with respect to n'. Further let m be the highest arity of a relation symbol from τ_{bi}.

Let $G = (V, E)$ be a 1-layered s-t-graph such that $V = \{s, t\} \cup A$ with $n = |A|$ and $E = \emptyset$. Further let $\mathcal{S} = (V, E, \mathcal{A}, \mathcal{B})$ be the state obtained by applying Init to G.

Here and in the following, we do not explicitly represent the constants s and t in $\mathcal{S}$, as they never change during the application of a modification sequence (but, of course, tuples containing constants might change in the graph and in the auxiliary relations).

First, we identify a subset of A on which the built-in relations are homogeneous. By Ramsey's Theorem for structures (choosing $\vec{d} = (s, t)$) and because $n = |A|$ is sufficiently large with respect to n' there is a set $A' \subseteq A$ of size n' and an order $\prec$ on A' such that all $\prec$-ordered m-tuples $\vec{a}_1$ and $\vec{a}_2$ over A' are of equal τ_{bi}-type.

Let $\mathcal{S}' \stackrel{\text{def}}{=} (V, E', \mathcal{A}', \mathcal{B})$ be the state of $\mathcal{P}$ that is reached from $\mathcal{S}$ after application of the following modifications to G (in some arbitrary order):

(α) For every node $a \in A'$, insert edges (s, a) and (a, t).

We observe that the built-in data has not changed, but the auxiliary data might have changed.

Let $a_1 \prec \ldots \prec a_{n'}$ be an enumeration of the elements of A'. For every $i \in \{1, \ldots, n'\}$, we define α_i to be the modification sequence that deletes the edges $(s, a_{n'}), (s, a_{n'-1}), \ldots, (s, a_{i+1})$, in this order. Let $\mathcal{S}'_i$ be the state reached by applying α_i to $\mathcal{S}'$. Thus, in state $\mathcal{S}'_i$ only nodes $a_1, \ldots, a_i$ have edges to node s. For every i, we construct a word w_i of length i, that has a letter for every node $a_1, \ldots, a_i$ and captures all relevant information about those nodes in $\mathcal{S}'_i$. The words w_i are over the set of all unary types of τ_{aux}. More precisely, the jth letter σ_i^j of w_i is the unary τ_{aux}-type of a_j in $\mathcal{S}'_i$. We recall that the unary type of a_j captures all information about the tuple (s, a_j, t).

Since $n' = |A'|$ was chosen sufficiently large with respect to τ, 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 $i_1 < \ldots < i_k$.

We argue that the structures $\mathcal{S}'_k \restriction \{s, t, a_1, \ldots, a_k\}$ and $\mathcal{S}'_l \restriction \{s, t, a_{i_1}, \ldots, a_{i_k}\}$ are isomorphic via the mapping π with $\pi(a_j) = a_{i_j}$ for all j, $\pi(s) = s$ and $\pi(t) = t$. By definition of A' and because built-in relations do not change, the mapping π preserves τ_{bi}. The schema τ_{aux} is preserved since a_j and a_{i_j} are of equal unary type, by the definition of w_k and w_l. Thus π is indeed an isomorphism. We refer to Figure 3 for an illustration.

Therefore, by Corollary 4.2, the program $\mathcal{P}$ computes the same query result for the following π-respecting modification sequences β_1 and β_2:

(β_1) Delete edges $(s, a_1), \ldots, (s, a_k)$ from $\mathcal{S}'_k$.

(β_2) Delete edges $(s, a_{i_1}), \ldots, (s, a_{i_k})$ from $\mathcal{S}'_l$.

However, applying the modification sequence β_1 yields a graph where t is not reachable from s, whereas by β_2 a graph is obtained where t is reachable from s since $k < l$, the desired contradiction.

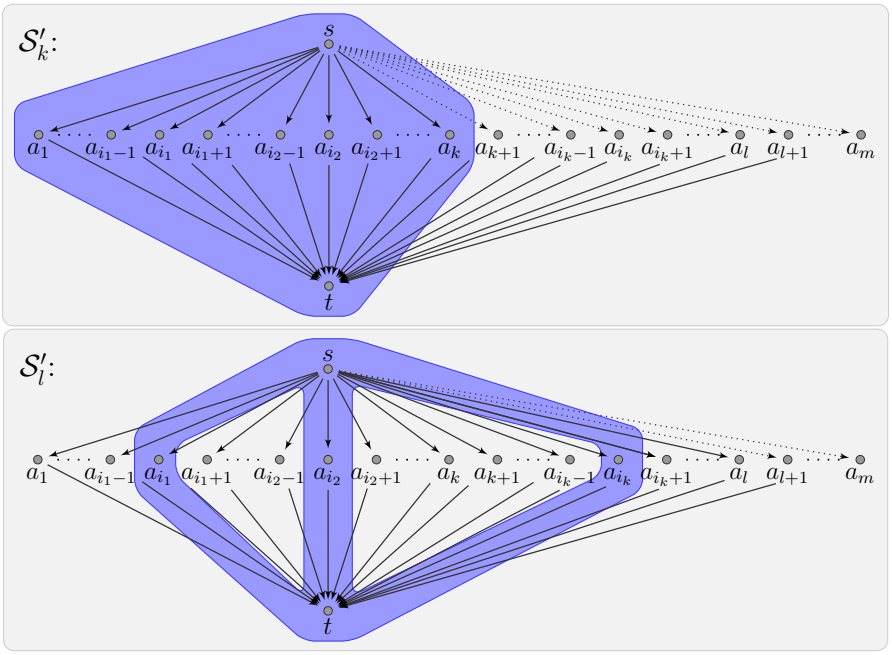

Figure 3: The structures $\mathcal{S}'_k$ and $\mathcal{S}'_l$ from the proof of Proposition 4.8. Deleted edges are dotted. The isomorphic substructures are highlighted in blue.

We now specify the numbers n and n' that were chosen in the beginning of the proof. In order to apply Higman's Lemma, the set A' needs to be of size at least $n' \stackrel{\text{def}}{=} H(|n''|)$ where n'' is the number of unary types of τ. Therefore, the set A has to be of size $n \stackrel{\text{def}}{=} R_\tau(n')$. □

Now we prove Theorem 4.7, i.e. that Dyn(s-t-Reach) is not in binary DynProp*. In the proof, we will again first choose a homogeneous subset with respect to the built-in relations. The notation introduced next and the following lemma prepare this step.

We refine the notion of homogeneous sets. 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 4.9. *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 $|A|, |B| \geq R_\tau^{hom}(n)$, there are subsets $A' \subseteq A$ and $B' \subseteq B$ such that $|A'|, |B'| = n$ and B' is A'-homogeneous in $\mathcal{S}$.

Proof. Let τ be a schema with maximal arity m. Choose k' to be a large number[10] with respect to τ and n; and let k be a large number with respect to k'. In particular k is large with respect to the number of constant symbols in τ. Further let A, B be disjoint subsets of the domain of a τ-structure $\mathcal{S}$ with $|A|, |B| > k$. Since k is large with respect to the number of constants in $\mathcal{S}$, we assume, without loss of generality, that neither A nor B contains a constant.

Fix a k'-tuple $\vec{a} = (a_1, \ldots, a_{k'})$ of A. Further let $\prec$ be an arbitrary order on B. Because $|B|$ is large with respect to k', n and τ, and by Ramsey's theorem on structures (choose $\vec{d} = \vec{a}$), there is a subset B' of B of size n such that for every $l \leq m$ the type of $(\vec{a}, \vec{b})$ in $\mathcal{S}$ is the same, for all $\prec$-ordered l-tuples $\vec{b}$ over B'.

Since k' is large with respect to τ and because there is only a bounded number of $(m+1)$-ary τ-types, there is an increasing sequence $i_1, \ldots, i_n$ such that for all $l \leq m$ the τ-types of tuples $(a_{i_j}, \vec{b})$ are equal, for all $\prec$-ordered l-tuples $\vec{b}$ over B' and $j \in \{1, \ldots, n\}$. We choose $A' := \{a_{i_1}, \ldots, a_{i_n}\}$. Then B' is A'-homogeneous up to arity m and therefore A'-homogeneous.

It remains to give explicit numbers. For the sequence $i_1, \ldots, i_n$ to exist in $1, \ldots, k'$, the number k' has to be at least $nM + 1$ where M is the number of $(m+1)$-ary τ-types. Thus k has to be at least $R_{\tau,k'}(k') + c$ where c is the number of constants in τ. Define $R_\tau^{\text{hom}}(n) \stackrel{\text{def}}{=} k$. □

Proof (of Theorem 4.7). 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 4.9 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 modification sequence α as follows.

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

[10] Again, explicit numbers can be found at the end of the proof.

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 4 for an illustration of the substructures $\mathcal{T}_k$ and $\mathcal{T}_l$ of $\mathcal{S}'$.

We show 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 $j \in \{1, \ldots, k\}$. We argue that π fulfills the requirements of an isomorphism, for every relation symbol R from $\tau_{\text{in}} \cup \tau_{\text{bi}} \cup \tau_{\text{aux}}$:

- For the input relation E this is obvious. In $\mathcal{S}'$ there are no edges from s to nodes in A_2 and all nodes from B_2 have an edge to t. Further X_l is connected to all nodes in $\vec{b}$ and X_k is connected to all nodes in $\vec{b}'$.
- For $R \in \tau_{\text{bi}}$, the requirement follows because B_2 is A_2-homogeneous for schema τ_{bi}.
- For $R \in \tau_{\text{aux}}$ of arity 2 and two 2-tuples $\vec{c}$ and $\pi(\vec{c})$ we distinguish two cases. First, if $\vec{c}$ and $\pi(\vec{c})$ contain elements from B_3 only, then $\vec{c} \in R^{\mathcal{T}_k}$ if and only if $\pi(\vec{c}) \in R^{\mathcal{T}_l}$ because B_3 is homogeneous in $\mathcal{S}'$. Second, if $\vec{c}$ contains s, t or A_{X_l}, then $\vec{c} \in R^{\mathcal{T}_k}$ if and only if $\pi(\vec{c}) \in R^{\mathcal{T}_l}$ because of the construction of w_k and w_l.

Thus, by the substructure lemma, application of the following two modification 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.

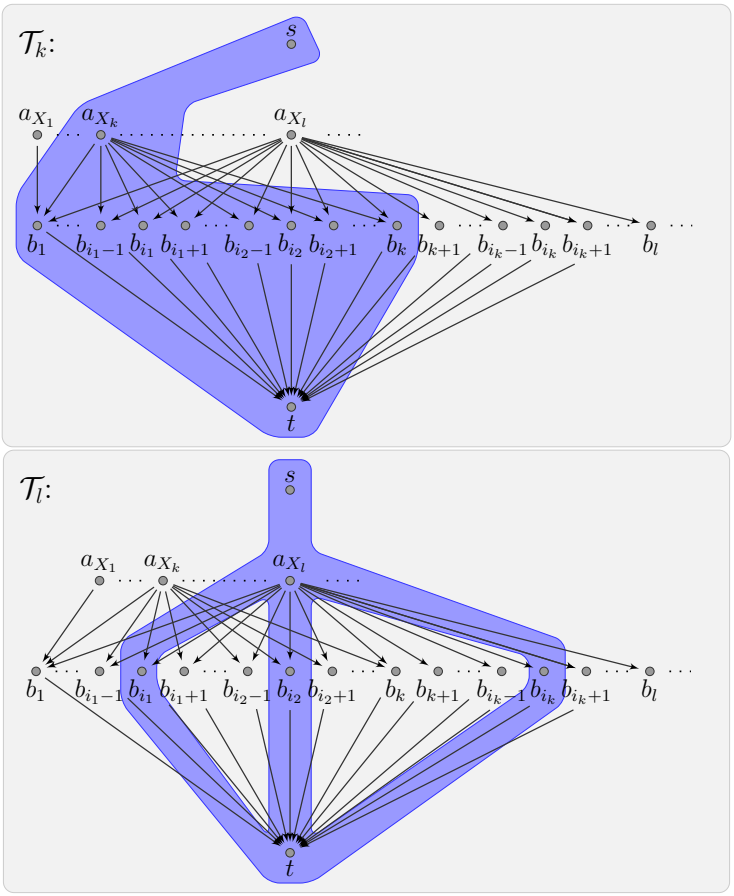

Figure 4: The structure $\mathcal{S}'$ from the proof of Theorem 4.7. The isomorphic substructures $\mathcal{T}_k$ and $\mathcal{T}_l$ are highlighted in blue.

It remains to specify the sizes of the sets. To apply Higman's lemma, $|B_3|$ has to be of size at least $n_3 \stackrel{\text{def}}{=} H(m)$ where m is the number of binary types over τ_{aux}. Hence, for applying Ramsey's theorem, $|B_2|$ has to be of size $n_2 \stackrel{\text{def}}{=} R_\tau(n_3)$. Thus it is sufficient if $|B_1|$ and $|A_1|$ contain $n_1 \stackrel{\text{def}}{=} 2^{n_2}$ elements. Therefore, by Lemma 4.9, the sets A and B can be chosen of size $n \stackrel{\text{def}}{=} R_\tau^{\text{hom}}(n_1)$. □

4.2 Separating Low Arities

An arity hierarchy for DynFO was established in [2]. 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 $6k+1$ (im-

proved to $3k+1$ in [3]). Here we establish a strict arity hierarchy between unary, binary and ternary DynProp for Boolean queries and binary input schemas.

We use the following problems s-t-TwoPath and s-TwoPath

Query:	s-t-TwoPath
Input:	An s-t-graph $G = (V, E)$.
Question:	Is there a path of length two from s to t?

Query:	s-TwoPath
Input:	A graph $G = (V, E)$ with one distinguished node $s \in V$.
Question:	Is there a path of length two starting from s?

Proposition 4.10. *The dynamic query* Dyn(s-t-TwoPath) *is in binary* DynProp*, but not in unary* DynProp**.*

Proof sketch. That Dyn(s-t-TwoPath) is not in unary DynProp* follows immediately from Proposition 4.8 as such a program would also maintain the dynamic s-t-reachability query for 1-layered graphs.

In order to prove that Dyn(s-t-TwoPath) is in binary DynProp, we sketch a DynProp-program (P, Init, Q) whose auxiliary schema contains unary relation symbols In, Out, First, and Last and a binary relation symbol List. The idea is to store, in a program state $\mathcal{S}$, a list of all nodes a such that (s, a, t) is a path in $E^{\mathcal{S}}$. The relation $\text{In}^{\mathcal{S}}$ contains all nodes with an incoming edge from s, and $\text{Out}^{\mathcal{S}}$ contains all nodes with an outgoing edge to t. The relations $\text{First}^{\mathcal{S}}$, $\text{Last}^{\mathcal{S}}$, $\text{List}^{\mathcal{S}}$ maintain the actual list, similarly to Example 1. The current query bit is maintained in $Q^{\mathcal{S}}$.

For a given instance of s-t-TwoPath the initialization mapping initializes the auxiliary relations accordingly.

Insertion of (a, b) into E. We note that edges (a, b) where $a \neq s$ and $b \neq t$ can be ignored, as they cannot contribute to any path of length 2 from s to t. Furthermore, paths of length 2 involving only nodes s and t can be easily handled by DynProp formulas, and therefore will be ignored as well.

If $a = s$ and $b \neq t$, then b is inserted into In, otherwise if $a \neq s$ and $b = t$ then a is inserted into Out.

Afterwards a or b is inserted into List, if it is now contained in both In and Out. In that case the query bit is set true.

Formally:

$$
\begin{aligned}
\phi^{\text{In}}_{\text{INS}}(a, b; x) &= \text{In}(x) \vee (x = b \wedge a = s \wedge b \neq s \wedge b \neq t) \\
\phi^{\text{Out}}_{\text{INS}}(a, b; x) &= \text{Out}(x) \vee (x = a \wedge a \neq s \wedge a \neq t \wedge b = t) \\
\phi^{\text{First}}_{\text{INS}}(a, b; x) &= \text{First}(x) \vee (\neg Q \wedge \varphi_n(x)) \\
\phi^{\text{Last}}_{\text{INS}}(a, b; x) &= (\text{Last}(x) \wedge \neg\varphi_n(a) \wedge \neg\varphi_n(b)) \vee \varphi_n(x) \\
\phi^{\text{List}}_{\text{INS}}(a, b; x, y) &= (\text{List}(x, y) \wedge \neg\varphi_n(a) \wedge \neg\varphi_n(b)) \vee (\text{Last}(x) \wedge \varphi_n(y)) \\
\phi^{Q}_{\text{INS}}(a, b) &= Q \vee \varphi_n(a) \vee \varphi_n(b)
\end{aligned}
$$

Here, $\varphi_n(x)$ is an abbreviation for

$$\phi^{\text{In}}_{\text{INS}}(a,b;x) \wedge \phi^{\text{Out}}_{\text{INS}}(a,b;x) \wedge (\neg \text{In}(x) \vee \neg \text{Out}(x))$$

expressing that x is becoming newly inserted into List.

Deletion of (a,b) from E. First, if $a = s$, then b is removed from In. Further if $b = t$ then a is removed from Out.

Afterwards a or b is removed from List, if it has been removed from In or Out. If List is empty now, then the query bit is set to false. The precise formulas are along the lines of the formulas of Example 1. □

Proposition 4.11. *The dynamic query* Dyn(s-TwoPath) *is in ternary* DynProp*, but not in binary* DynProp*.

Proof sketch. For proving that Dyn(s-TwoPath) is not in binary DynProp*, assume to the contrary that there is a binary DynProp*-program $\mathcal{P} = (P, \text{Init}, Q)$ for Dyn(s-TwoPath). With the help of $\mathcal{P}$ one can, for the graphs from the proof of Proposition 4.8, maintain whether there is a path from s to some node of B. However, this yields a correct answer for s-t-Reach for those graphs, since in the proof all nodes of B have an edge to t.

In order to prove that Dyn(s-TwoPath) is in ternary DynProp, we sketch a DynProp-program (P, Init, Q) whose auxiliary schema contains unary relation symbols In, Out, First_1, Last_1 and Empty_1, binary relation symbols List_1, First_2, Last_2 and Empty_2, and a ternary relation symbol List_2. The idea is that in a state $\mathcal{S}$, the binary relation $\text{List}_1^{\mathcal{S}}$ contains a list of all nodes a on a path (s,a,b) in $E^{\mathcal{S}}$, for some node b. The relation $\text{In}^{\mathcal{S}}$ contains all nodes with an incoming edge from s, and $\text{Out}^{\mathcal{S}}$ contains all nodes with an outgoing edge. In order to update $\text{Out}^{\mathcal{S}}$, the projection $\text{List}_2^{\mathcal{S}}(a,\cdot,\cdot)$ of the ternary relation $\text{List}_2^{\mathcal{S}}$ stores a list of nodes b with $(a,b) \in E^{\mathcal{S}}$, for every node a. The lists $\text{List}_1^{\mathcal{S}}$ and $\text{List}_2^{\mathcal{S}}(a,\cdot,\cdot)$ are maintained by using the technique from Example 1 and by using the auxiliary relations stored in $\text{First}_1^{\mathcal{S}}$, $\text{Last}_1^{\mathcal{S}}$, $\text{Empty}_1^{\mathcal{S}}$, $\text{First}_2^{\mathcal{S}}$, $\text{Last}_2^{\mathcal{S}}$ and $\text{Empty}_2^{\mathcal{S}}$. The current query bit is maintained in $Q^{\mathcal{S}}$.

For a given instance of s-TwoPath the initialization mapping initializes the auxiliary relations accordingly.

Insertion of (a,b) into E. First, if $a = s$ then b is inserted into In. Otherwise, a is inserted into Out and b is inserted into $\text{List}_2(a,\cdot,\cdot)$.

Afterwards a or b is inserted into List_1, if it is now contained in both In and Out. If one of them is inserted, then the query bit is set true.

Deletion of (a,b) from E. First, if $a = s$ then b is removed from In. Otherwise, b is removed from $\text{List}_2(a,\cdot,\cdot)$ and if $\text{List}_2(a,\cdot,\cdot)$ is empty afterwards, then a is removed from Out.

Afterwards a or b is removed from List_1, if it has been removed from In or Out. The query bit is set to false, if the list List_1 is empty now. □

4.3 Invariant Initialization

We now turn to the setting with invariant initialization. Recall that an initialization mapping INIT with $\text{INIT} = (\text{INIT}_{\text{aux}}, \text{INIT}_{\text{bi}})$ is invariant if

$$\pi(\text{INIT}_{aux}(\mathcal{D})) = \text{INIT}_{aux}(\pi(\mathcal{D})) \text{ and } \pi(\text{INIT}_{\text{bi}}(D)) = \text{INIT}_{\text{bi}}(\pi(D))$$

for every database $\mathcal{D}$, domain D and permutation π of the domain. The condition $\pi(\text{INIT}_{\text{bi}}(D)) = \text{INIT}_{\text{bi}}(\pi(D))$ implies that a built-in relation contains either all tuples or no tuple at all. Therefore DYNPROP and DYNPROP* with invariant initialization mapping coincide.

First-order logic, second-order logic and other logics considered in computer science can only define queries, i.e. mappings that are invariant under permutations. Therefore the following result applies, in particular, for all initialization mappings defined in those logics.

Theorem 4.12. DYN(s-t-REACH) *cannot be maintained in* DYNPROP *with invariant initialization mapping. This holds even for 1-layered s-t-graphs.*

Proof. Towards a contradiction, assume that the dynamic program (P, INIT, Q) with schema $\tau = \tau_{\text{in}} \cup \tau_{\text{aux}}$ and invariant initialization mapping INIT maintains the s-t-reachability query for 1-layered s-t-graphs. Let n be the number of types of tuples of arity up to m for $\tau_{\text{aux}} \cup \{E\}$ where m is the highest arity of relation symbols in $\tau_{\text{aux}} \cup \{E\}$.

We consider the 1-layered s-t-graphs $G_i = (V_i, E_i)$, for every i from $1, \ldots, n+1$, with $V_i = \{s, t\} \cup A_i$ where $A_i = \{a_0, \ldots, a_i\}$ and $E = \{s\} \times A_i \cup A_i \times \{t\}$. Further, we let $\mathcal{S}_i = (V_i, E_i, \mathcal{A}_i)$ be the state obtained by applying INIT to G_i.

Our goal is to find $\mathcal{S}_k$ and $\mathcal{S}_l$ with $k < l$ such that $\mathcal{S}_k$ is isomorphic to $\mathcal{S}_l \restriction V_k$ (see Figure 5 for an illustration). Then, by the substructure lemma, the program $\mathcal{P}$ computes the same query result for the following modification sequences:

(β_1) Delete edges $(s, a_0), \ldots, (s, a_k)$ from $\mathcal{S}_k$.

(β_2) Delete edges $(s, a_0), \ldots, (s, a_k)$ from $\mathcal{S}_l$.

However, applying the modification sequence β_1 yields a graph where t is reachable from s, whereas by β_2 a graph is obtained where t is not reachable from s, a contradiction.

Thus it remains to find such states $\mathcal{S}_k$ and $\mathcal{S}_l$. A tuple is *diverse*, if all components are pairwise different. For arbitrary $m' \leq m$, diverse tuples $\vec{a}, \vec{b} \in A^{m'}$ and $i \leq n$, we observe that $G_i \simeq_{id[\vec{a},\vec{b}]} G_i$ where $id[\vec{a}, \vec{b}]$ is the bijection that maps a_i to b_i, b_i to a_i and every other element from S to itself. Therefore $\mathcal{S}_i \simeq_{id[\vec{a},\vec{b}]} \mathcal{S}_i$ by the invariance of INIT. Thus $\langle \mathcal{S}_i, \vec{a} \rangle = \langle \mathcal{S}_i, \vec{b} \rangle$, and therefore all diverse m' tuples are of the same type in $\mathcal{S}_i$.

Since n is the number of types up to arity m, there are two states $\mathcal{S}_k$ and $\mathcal{S}_l$ such that, for every $m' \leq m$, all diverse m'-tuples are of the same type in $\mathcal{S}_k$ and $\mathcal{S}_l$. But then $\mathcal{S}_k \simeq \mathcal{S}_l \restriction V_k$. □

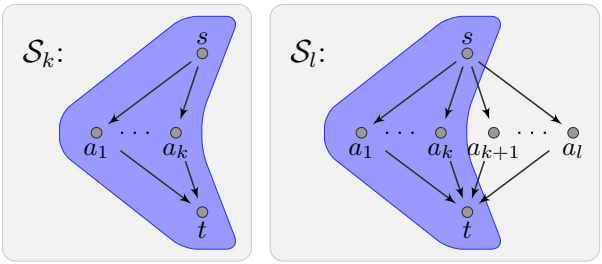

Figure 5: The structures $\mathcal{S}_k$ and $\mathcal{S}_l$ from the proof of Theorem 4.12. The isomorphic substructures are highlighted in blue.

The proof of the previous result does not extend to DynFO, since reachability in graphs of depth three is expressible even in (static) predicate logic. The proof fails, because the substructure lemma does not hold for DynFO-programs. At first glance, layered graphs with many layers look like a good candidate for proving that DynFO cannot maintain s-t-Reach in this setting. However, in [6] it is shown that DynFO with FO+TC-definable initialization mappings can express s-t-Reach for arbitrary acyclic graphs.

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 can only access the inserted or deleted tuple $\vec{a}$ and the currently updated tuple $\vec{b}$ of an auxiliary relation. With auxiliary functions further elements might be accessed via function terms over $\vec{a}$ and $\vec{b}$. Thus, in a sense, auxiliary functions 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.

After the formal definition of DynQF and adapting the substructure lemma to it, we prove that

- Dyn(s-t-Reach) is not in unary DynQF; and
- Dyn(s-t-Reach) is not in DynQF with invariant initialization.

When full first-order updates are available, auxiliary functions can be simulated in a straight forward way by auxiliary relations. However, without quantifiers this is not possible. Auxiliary functions are quite powerful. While only regular languages can be maintained in DynProp, all Dyck languages, among other non-regular languages, can be maintained in DynQF [5]. Furthermore, undirected reachability can be maintained in DynQF with built-in relations [9].

We extend our definition of schemata to allow also function symbols. Within this section, a *schema (or signature)* τ consists of a set τ_{rel} of relation symbols, a

set τ_{fun} of function symbols and a set τ_{const} of constant symbols together with an arity function $\text{Ar} : \tau_{\text{rel}} \cup \tau_{\text{fun}} \mapsto \mathbb{N}$. A schema is *relational* if $\tau_{\text{fun}} = \emptyset$. 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, to every k-ary function symbol $f \in \tau_{\text{fun}}$ a k-ary function, and to every constant symbol $c \in \tau_{\text{const}}$ a single element (called *constant*) from D.

In the following, we extend our definition of update programs for the case of auxiliary schemas with functions[11]. It is straightforward to extend the definition of update formulas for auxiliary relations: they simply can make use of function terms. However, following the spirit of DynProp, we allow a more powerful update mechanism for auxiliary functions that allows case distinctions in addition to composition of function terms.

The following definitions are adapted from [5].

Definition 6. (Update term) *Update terms* are inductively defined by the following.

(1) Every variable and every constant is an update term.

(2) If f is a k-ary function symbol and $t_1, \ldots, t_k$ are update terms, then $f(t_1, \ldots, t_k)$ is an update term.

(3) If ϕ is a quantifier-free update formula (possibly using update terms) and t_1 and t_2 are update terms, then $\text{ite}(\phi, t_1, t_2)$ is an update term.

The semantics of update terms associates with every update term t and interpretation $I = (\mathcal{S}, \beta)$, where $\mathcal{S}$ is a state and β a variable assignment, a value $[\![t]\!]_I$ from S. The semantics of (1) and (2) is straightforward. If $\mathcal{S} \models \phi$ holds, then $[\![\text{ite}(\phi, t_1, t_2)]\!]_I$ is $[\![t_1]\!]_I$, otherwise $[\![t_2]\!]_I$.

The extension of the notion of update programs for auxiliary schemas with function symbols is now straightforward. An update program still has an update formula ϕ^R_δ (possibly using terms built from function symbols) for every relation symbol $R \in \tau_{\text{aux}}$ and every abstract modification δ. Furthermore, it has, for every abstract modification δ and every function symbol $f \in \tau_{\text{aux}}$, an update term $t^f_\delta(\vec{x}; \vec{y})$. For a concrete modification $\delta(\vec{a})$ it redefines f for each tuple $\vec{b}$ by evaluating $t^f_\delta(\vec{a}; \vec{b})$ in the current state.

Definition 7. (DynQF) DynQF is the class of queries maintainable by quantifier-free update programs with (possibly) auxiliary functions. The class k-ary DynQF is defined via update programs that use auxiliary functions and relations of arity at most k.

We define DynQF* as the extension of DynQF with built-in functions and relations of arbitrary arity.

Lists can be represented by unary functions in a straightforward way. Therefore, it is not surprising that the upper bound of Proposition 4.10 already holds for unary DynProp with unary built-in functions.

[11] We also allow functions in built-in schemas. As they are not updated they do not need any further particular definitions.

Proposition 5.1. DYN(s-t-REACH) *on 1-layered s-t-graphs can be maintained in unary* DYNQF* *with relational auxiliary schema and only unary built-in functions. In particular,* DYN(s-t-REACH) *on 1-layered s-t-graphs can be maintained in unary* DYNQF*.*

Proof sketch. We construct a DYNQF*-program $\mathcal{P}$ over relational auxiliary schema $\{Q, \mathrm{ConS}, \mathrm{ConT}, C\}$ and functional built-in schema $\{\textsc{Pred}, \textsc{Succ}\}$, where Q is the query bit (i.e. a 0-ary relation symbol), ConS, ConT and C are unary relation symbols and PRED and SUCC are unary function symbols.

The basic idea is to interpret elements of D as numbers according to their position in the graph of SUCC. For simplicity, but without loss of generality, we therefore assume that the domain is of the form $D = \{0, \ldots, n-1\}$ with $s = 0$ and $t = n-1$. For every state $\mathcal{S}$, the built-in function $\textsc{Succ}^{\mathcal{S}}$ is then the standard successor function on D (with $\textsc{Succ}^{\mathcal{S}}(n-1) = n-1$) and $\textsc{Pred}^{\mathcal{S}}$ is its corresponding predecessor function (with $\textsc{Pred}^{\mathcal{S}}(0) = 0$).

The second idea is to store the current number i of vertices connected to both s and t by letting $C^{\mathcal{S}} = \{i\}$. If an edge-insertion connects an element to s and t then i is replaced by $i+1$ in $C^{\mathcal{S}}$ with the help of $\textsc{Pred}^{\mathcal{S}}$ and $\textsc{Succ}^{\mathcal{S}}$. Analogously i is replaced by $i-1$ for edge-removals that disconnect an element from s or t. The relations $\mathrm{ConS}^{\mathcal{S}}$ and $\mathrm{ConT}^{\mathcal{S}}$ store the elements currently connected to s and t, respectively.

For a given instance of the s-t-reachability query on 1-layered s-t-graphs the initialization mapping initializes the auxiliary relations accordingly.

Insertion of (a,b) into E. If $a = s$ then node b is inserted into ConS; if $b = t$ then node a is inserted into ConT. Further, if a or b is now in both S and T then the counter is incremented by 1:

$$\begin{aligned}
\phi^{\mathrm{ConS}}_{\mathrm{INS}}(a,b;x) &\stackrel{\text{def}}{=} (a = s \wedge x = b) \vee \mathrm{ConS}(x)\\
\phi^{\mathrm{ConT}}_{\mathrm{INS}}(a,b;x) &\stackrel{\text{def}}{=} (b = t \wedge x = a) \vee \mathrm{ConT}(x)\\
\phi^{C}_{\mathrm{INS}}(a,b;x) &\stackrel{\text{def}}{=} \big(a = s \wedge \mathrm{ConT}(b) \wedge C(\textsc{Pred}(x))\big)\\
&\qquad \vee \big(b = t \wedge \mathrm{ConS}(a) \wedge C(\textsc{Pred}(x))\big)\\
&\qquad \vee \big(a = s \wedge \neg\mathrm{ConT}(b) \wedge C(x)\big)\\
&\qquad \vee \big(b = t \wedge \neg\mathrm{ConS}(a) \wedge C(x)\big)\\
\phi^{Q}_{\mathrm{INS}}(a,b) &\stackrel{\text{def}}{=} \neg\phi^{C}_{\mathrm{INS}}(a,b;s)
\end{aligned}$$

Deletions can be maintained in a similar way. □

We refer to [9, Section 4.3] and [5, Sections 4 and 6] for more examples of DYNQF-programs.

In the following we work towards lower bounds for DYNQF. We first extend the substructure lemma to non-relational structures. If a modification changes a tuple from a substructure $\mathcal{A}$ of a structure $\mathcal{S}$, then the update of the auxiliary data of $\mathcal{A}$ can depend on elements obtained from applying functions to elements in $\mathcal{A}$. We formally capture these elements by the notion of neighborhood, defined next.

The *nesting depth* $d(t)$ of an update term t is its nesting depth with respect to function symbols: If t is a variable, then $d(t) = 0$; if t is of the form $f(t_1, \ldots, t_k)$ then $d(t) = \max\{d(t_1), \ldots, d(t_k)\} + 1$; and if t is of the form $\text{ITE}(\phi, t_1, t_2)$ then $d(t) = \max\{d(\phi), d(t_1), d(t_2)\}$. The nesting depth $d(\phi)$ of ϕ is the maximal nesting depth of all update terms occurring in ϕ. The *nesting depth of* $\mathcal{P}$ is the maximal nesting depth of an update term occurring in $\mathcal{P}$.

For a schema τ, let TERMS^k_τ be the set of terms of nesting depth at most k with function symbols from τ. Informally, the k-neighborhood of a set A is the set of all elements of S that can be obtained by applying a term of nesting depth at most k to a vector of elements from A.

Definition 8. (Neighborhoods) Let $\mathcal{S}$ be a state with domain S over schema τ and $k \geq 0$. The *k-neighborhood* $\mathcal{N}^k_\mathcal{S}(A)$ of a set $A \subseteq S$ is the set

$$\{[\![t]\!]_{(\mathcal{S},\beta)} \mid t \in \text{TERMS}^k_\tau \text{ and } \beta(x) \in A, \text{for every variable } x \text{ in } t\}.$$

A subset A of S is *closed* if $\mathcal{N}^1_\mathcal{S}(A) = A$.

The k-neighborhood of a tuple $\vec{a}$ or a single element a is defined accordingly. We note that for a closed set A it also holds $\mathcal{N}^k_\mathcal{S}(A) = A$, for every k.

A bijection π between (the domains S and T of) two structures $\mathcal{S}$ and $\mathcal{T}$ over $\tau = \tau_{\text{rel}} \cup \tau_{\text{fun}}$ is an *isomorphism*, if it preserves τ_{rel} and $\pi(f^\mathcal{S}(\vec{a})) = f^\mathcal{T}(\pi(\vec{a}))$ for all k-ary function symbols $f \in \tau_{\text{fun}}$ and k-tuples $\vec{a}$ over S. Two subsets $A \subseteq S$, $B \subseteq T$ are *k-similar*, if there is a bijection $\pi : \mathcal{N}^k_\mathcal{S}(A) \to \mathcal{N}^k_\mathcal{T}(B)$ such that

- the restriction of π to A is a bijection of A and B,
- π satisfies $\pi(t^\mathcal{S}(\vec{a})) := t^\mathcal{T}(\pi(\vec{a}))$ for all $t \in \text{TERMS}^k_{\tau_{\text{fun}}}$ and $\vec{a}$ over A, and
- π preserves τ_{rel} on $\mathcal{N}^k_\mathcal{S}(A)$.

We write $A \approx^{\pi,\mathcal{S},\mathcal{T}}_k B$ to indicate that A and B are k-similar via π in $\mathcal{S}$ and $\mathcal{T}$. We drop $\mathcal{S}$ and $\mathcal{T}$ from this notation if they are clear from the context, and we drop π if the name is not important. We also write $(a_1, \ldots, a_p) \approx^{\mathcal{S},\mathcal{T}}_k (b_1, \ldots, b_p)$ to indicate that $\{a_1, \ldots, a_p\} \approx^{\pi,\mathcal{S},\mathcal{T}}_k \{b_1, \ldots, b_p\}$ via the isomorphism π that maps a_i to b_i, for every $i \in \{1, \ldots, p\}$. Note that if $A \approx_0 B$, then $\mathcal{S} \restriction A$ and $\mathcal{T} \restriction B$ are τ_{rel}-isomorphic by the first and third property.

The following lemma is a slight generalization of Lemma 4 from [5] and a generalization of the substructure lemma for DynProp (Lemma 4.1) to DynQF*. Intuitively, the substructure lemma for DynQF* requires not only similarity of the substructures but of their neighborhoods as well.

Lemma 5.2 (Substructure lemma for DynQF)**.** *Let $\mathcal{P}$ be a* DynQF* *program with nesting depth k and let l be some number. Furthermore let $\mathcal{S}$ and $\mathcal{T}$ be states of $\mathcal{P}$ with domains S and T and let A and B be subsets of S and T, respectively. There is a number $m \in \mathbb{N}$ such that if $A \approx^{\pi,\mathcal{S},\mathcal{T}}_m B$, then $A \approx^{\pi,P_\alpha(\mathcal{S}),P_\beta(\mathcal{T})}_0 B$, for all π-respecting modification sequences α and β on A and B of length at most l.*

Proof. The proof is an extension of the proof of Lemma 4.1. The lemma follows by an induction over the length l of the modification sequence. For $l = 0$ there is nothing to prove. The induction step follows easily using Claim (C) below.

Let $\delta(\vec{a})$ and $\delta(\vec{b})$ be two π-respecting modifications on A and B, respectively, i.e. $\vec{b} = \pi(\vec{a})$. Let $\mathcal{S}' \stackrel{\text{def}}{=} P_{\delta(\vec{a})}(\mathcal{S})$ and $\mathcal{T}' \stackrel{\text{def}}{=} P_{\delta(\vec{b})}(\mathcal{T})$. We prove the following claims for arbitrary $r \in \mathbb{N}$:

(A) If $A \approx^{\pi,\mathcal{S},\mathcal{T}}_{r+k} B$, then[12] for all $\vec{c}$ over $\mathcal{N}^r_{\mathcal{S}}(A)$:

  (i) $\vec{c} \in R^{\mathcal{S}'}$ if and only if $\pi(\vec{c}) \in R^{\mathcal{T}'}$ for all relation symbols $R \in \tau_{\text{aux}}$.

  (ii) $f^{\mathcal{S}'}(\vec{c}) \in \mathcal{N}^{r+k}_{\mathcal{S}}(A)$ and $\pi(f^{\mathcal{S}'}(\vec{c})) = f^{\mathcal{T}'}(\pi(\vec{c}))$ for all function symbols $f \in \tau_{\text{aux}}$.

(B) If $A \approx^{\pi,\mathcal{S},\mathcal{T}}_{r\cdot k} B$, then $t^{\mathcal{S}'}(\vec{c}) \in \mathcal{N}^{r\cdot k}_{\mathcal{S}}(A)$ and $\pi(t^{\mathcal{S}'}(\vec{c})) = t^{\mathcal{T}'}(\pi(\vec{c}))$ for all terms $t \in \text{TERMS}^r_{\tau_{\text{aux}}\cup\tau_{\text{bi}}}$ and $\vec{c}$ over S.

(C) If $A \approx^{\pi,\mathcal{S},\mathcal{T}}_{r\cdot k+k} B$, then $A \approx^{\pi,\mathcal{S}',\mathcal{T}'}_{r} B$.

We prove Claim (A) first. We recall that $\vec{c} \in R^{\mathcal{S}'}$ if and only if $\mathcal{S} \models \phi^R_\delta(\vec{a};\vec{c})$, and that $f^{\mathcal{S}'}(\vec{c})$ is $[\![t^f_\delta(\vec{x};\vec{y})]\!]_{(\mathcal{S},\gamma)}$, where γ maps $(\vec{x},\vec{y})$ to $(\vec{a},\vec{c})$. Since $\vec{a}$ and $\vec{c}$ are tuples over $\mathcal{N}^r_{\mathcal{S}}(A)$ it is sufficient to prove, for every tuple $\vec{d}$ over $\mathcal{N}^r_{\mathcal{S}}(A)$, that (i) $\varphi(\vec{d})$ holds in $\mathcal{S}$ if and only if $\varphi(\pi(\vec{d}))$ holds in $\mathcal{T}$, for every quantifier-free formula φ with nesting depth at most k, and that[13] (ii) $\pi([\![t]\!]_{(\mathcal{S},\vec{d})}) = [\![t]\!]_{(\mathcal{T},\pi(\vec{d}))}$, for every update term t with nesting depth at most k.

The proof is by induction on k. We start with the base case. If $k = 0$, terms and update terms do not use any function symbols and therefore, (i) and (ii) hold trivially, because π witnesses the $(r+k)$-similarity of A and B in $\mathcal{S}$ and $\mathcal{T}$.

For the induction step, we consider update terms and update formulas with nesting depth $k' \in \{1,\dots,k\}$. If an update term t with $d(t) = k'$ is of the form $f(\vec{s})$ with $\vec{s} = (s_1,\dots,s_n)$, then, by induction hypothesis, $\pi([\![s_i]\!]_{(\mathcal{S},\vec{e}_i)}) = [\![s_i]\!]_{(\mathcal{T},\pi(\vec{e}_i))}$ and $s^{\mathcal{S}}_i(\vec{e}_i) \in \mathcal{N}^{r+k'-1}_{\mathcal{S}}(A)$ for every i and vector $\vec{e}_i$ consisting of elements from $\vec{d}$. Thus, $\pi([\![f(\vec{s})]\!]_{(\mathcal{S},\vec{d})}) = [\![f(\vec{s})]\!]_{(\mathcal{T},\pi(\vec{d}))}$ because A and B are $(r+k)$-similar and $k' \le k$. The other cases are analogous. This concludes the proof of Claim (A).

Claim (B) can be proved by an induction over the nesting depth of t. The induction step uses Claim (A ii).

For Claim (C) we have to prove that π is witnessing the r-similarity of A and B in $\mathcal{S}'$ and $\mathcal{T}'$. The first property of similarity is trivial and the second follows from Claim (B). For the third property let $\vec{c}$ be an arbitrary m-tuple over $\mathcal{N}^r_A(\mathcal{S}')$ and R some m-ary relation symbol. Then $\vec{c} = ([\![t_1]\!]_{(\mathcal{S}',\vec{c}_1)},\dots,[\![t_n]\!]_{(\mathcal{S}',\vec{c}_n)})$ with $\vec{c}_i$ over A and $t_i \in \text{TERMS}^r_{\tau_{\text{aux}}}$. Thus $\vec{c}$ is a tuple over $\mathcal{N}^{r\cdot k}_A(\mathcal{S})$, by Claim (B), and therefore $R^{\mathcal{S}'}(\vec{c})$ if and only if $R^{\mathcal{T}'}(\pi(\vec{c}))$, by Claim (A). □

[12] Of course, the following two statements also hold for relation and function symbols from τ_{bi}.

[13] Here, we use $\vec{d}$ to denote the variable assignment mapping the free variables of t to the components of $\vec{d}$.

We now prove that unary DynQF cannot maintain s-t-reachability. Intuitively, unary functions cannot store the transitive closure relation of a directed path in such a way, that the information can be extracted by a quantifier-free formula. The proof is simplified by the following observation.

Lemma 5.3. *If an l-ary query $\mathcal{Q}$ can be maintained by a* DynQF*-program, then $\mathcal{Q}$ can be maintained by a k-ary* DynQF*-program with only one l-ary auxiliary relation (used for storing the query result) on databases with at least two elements.*

The restriction to structures with at least two elements is harmless, as we only use this lemma in a context where structures indeed have at least two elements.

Proof sketch. In order to encode relations by functions, two constants (i.e., 0-ary functions) $c_\bot$ and $c_\top$ are used. Those constants are initialized by two distinct elements of the domain. Then a k-ary relation R can be easily encoded by a k-ary function f_R via $(a_1, \ldots, a_k) \in R$ if and only if $f_R(a_1, \ldots, a_k) = c_\top$. □

Theorem 5.4. Dyn(s-t-Reach) *is not in unary* DynQF.

Proof. Towards a contradiction, we assume that $\mathcal{P} = (P, \text{Init}, Q)$ is a unary DynQF-program that maintains s-t-reachability over schema $\tau = \tau_{\text{in}} \cup \tau_{\text{aux}}$ with unary τ_{aux}. By Lemma 5.3 we can assume that τ_{aux} contains only 0-ary and unary function symbols and one 0-ary relation symbol Q for storing the query result. The graphs used in this proof do not have self-loops and every node has at most one outgoing edge. Therefore we can assume, in order to simplify the presentation, that τ_{aux} contains a unary function symbol e, such that in every state $\mathcal{S}$ the function $e^{\mathcal{S}}$ encodes the edge relation E as follows. If the single outgoing edge from u is (u, v) then $e(u) = v$ and if u has no outgoing edge then $e(u) = u$.

Let k be the nesting depth of $\mathcal{P}$ and let n be chosen sufficiently large with respect to τ and k. Let $G = (V, E)$ be a graph where $V = \{s, t\} \cup A$ with $A = \{a_1, \ldots, a_n\}$ and $E = \{(a_i, a_{i+1}) \mid i \in \{1, \ldots, n-1\}\}$, i.e., $G \restriction A$ is a path of length $n - 1$ from a_1 to a_n. Further, let $\mathcal{S} = (V, E, \mathcal{A})$ be the state obtained by applying Init to G.

Our goal is to find i and j with $i < j$ such that for the two nodes $a \stackrel{\text{def}}{=} a_i$ and $b \stackrel{\text{def}}{=} a_j$ it holds $(a, b, s, t) \approx_m (b, a, s, t)$, where m is the number from the substructure lemma for auxiliary functions (Lemma 5.2), for modification sequences of length 2 and nesting depth k.

Then, by Lemma 5.2, the program $\mathcal{P}$ computes the same query result for the following two modification sequences:

(β_1) Insert edges (s, a) and (b, t).

(β_2) Insert edges (s, b) and (a, t).

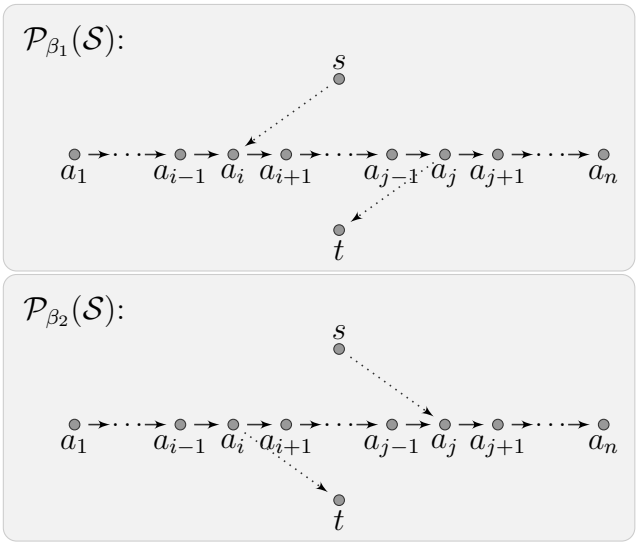

Figure 6: The structure $\mathcal{S}$ from the proof of Theorem 5.4. Edges inserted by modification sequence β_1 and modification sequence β_2, respectively, are dotted.

However, applying the modification sequence β_1 yields a graph in which t is reachable from s, whereas β_2 yields a graph in which t is not reachable from s (see Figure 6 for an illustration). This is the desired contradiction.

Thus it remains to show the existence of such i and j. To this end, let $t_1, \ldots, t_l$ be the lexicographic enumeration of $\textsc{Terms}_\tau^k$ with respect to some fixed order of the function symbols. Let the *k-neighborhood vector* $\vec{\mathcal{N}}_{\mathcal{S}}^k(c)$ of an element c in $\mathcal{S}$ be the tuple $(c, t_1(c), \ldots, t_l(c))$. For a tuple $\vec{c} = (c_1, \ldots, c_m)$, the k-neighborhood vector $\vec{\mathcal{N}}_{\mathcal{S}}^k(\vec{c})$ of $\vec{c}$ is the tuple $(\vec{\mathcal{N}}_{\mathcal{S}}^k(c_1), \ldots, \vec{\mathcal{N}}_{\mathcal{S}}^k(c_m))$. The number of equality types of such neighborhood vectors is finite and bounded by a number that only depends on m, k and τ_{aux}.

By applying Ramsey's theorem on the graph over $\{1, \ldots, n\}$, where each pair (i, j) with $i < j$ is colored by the equality type of $\vec{\mathcal{N}}_{\mathcal{S}}^{m+1}(a_i, a_j, s, t)$, we obtain numbers $i_1 < i_2 < i_3$ such that the equality types of $\vec{\mathcal{N}}_{\mathcal{S}}^{m+1}(a_{i_1}, a_{i_2}, s, t)$, $\vec{\mathcal{N}}_{\mathcal{S}}^{m+1}(a_{i_1}, a_{i_3}, s, t)$, and $\vec{\mathcal{N}}_{\mathcal{S}}^{m+1}(a_{i_2}, a_{i_3}, s, t)$ are equal. In particular, as all function symbols are unary, the equality types of $\vec{\mathcal{N}}_{\mathcal{S}}^{m+1}(a_{i_1}, s, t)$, and $\vec{\mathcal{N}}_{\mathcal{S}}^{m+1}(a_{i_2}, s, t)$ and finally those of $\vec{\mathcal{N}}_{\mathcal{S}}^{m+1}(a_{i_1}, a_{i_2}, s, t)$ and $\vec{\mathcal{N}}_{\mathcal{S}}^{m+1}(a_{i_2}, a_{i_1}, s, t)$ are equal.

For the latter conclusion, we show the following claim: if for two terms t_1 and t_2 of depth at most $m+1$ it holds $t_1(a_{i_1}) = t_2(a_{i_2})$ then also $t_1(a_{i_2}) = t_2(a_{i_1})$. We observe that if $t_1(a_{i_1}) = t_2(a_{i_2})$ then also $t_1(a_{i_1}) = t_2(a_{i_3})$ and $t_1(a_{i_2}) = t_2(a_{i_3})$ (since $\vec{\mathcal{N}}_{\mathcal{S}}^{m+1}(a_{i_1}, a_{i_2}, s, t)$, $\vec{\mathcal{N}}_{\mathcal{S}}^{m+1}(a_{i_1}, a_{i_3}, s, t)$, and $\vec{\mathcal{N}}_{\mathcal{S}}^{m+1}(a_{i_2}, a_{i_3}, s, t)$ have the same equality type). Hence, $t_1(a_{i_2}) = t_2(a_{i_2})$ and therefore $t_1(a_{i_2}) = t_2(a_{i_2}) = t_1(a_{i_1}) = t_2(a_{i_1})$. The latter equality follows as the equality types of

$\vec{\mathcal{N}}_{\mathcal{S}}^{m+1}(a_{i_1}, s, t)$, and $\vec{\mathcal{N}}_{\mathcal{S}}^{m+1}(a_{i_2}, s, t)$ are equal. This concludes the proof of the claim.

To prove $(a, b, s, t) \approx_m (b, a, s, t)$ it only remains to show that $(u, v) \in E$ if and only if $(u', v') \in E$, for two components u and v from $\vec{\mathcal{N}}_{\mathcal{S}}^{m}(a, b, s, t)$ and their corresponding components u' and v' from $\vec{\mathcal{N}}_{\mathcal{S}}^{m}(b, a, s, t)$. However, $(u, v) \in E$ if and only if $e(u) = v$, and analogously $(u', v') \in E$ if and only if $e(u') = v'$. Thus this claim follows already from the fact that $\vec{\mathcal{N}}_{\mathcal{S}}^{m+1}(a_{i_1}, a_{i_2}, s, t)$ and $\vec{\mathcal{N}}_{\mathcal{S}}^{m+1}(a_{i_2}, a_{i_1}, s, t)$ have the same equality type. □

We now extend the lower bound for invariant initialization obtained in Theorem 4.12 to quantifier-free programs with auxiliary functions. Invariant initialization is still weak in the presence of auxiliary functions in the sense, that functions initialized by invariant initialization can only point to 'distinguished' nodes, as formalized by the following lemma.

Lemma 5.5. *Let $\mathcal{P} = (P, \textsc{Init}, Q)$ be a* DynQF*-program with invariant initialization mapping* Init *and auxiliary schema τ_{aux}. Further let $\mathcal{I}$ be an input structure for $\mathcal{P}$ whose domain contains b and b' with $b \neq b'$. If $id[b, b']$ is an isomorphism of $\mathcal{I}$, then $f^{\textsc{Init}(\mathcal{I})}(\vec{a}) \neq b$ for all k-ary function symbols $f \in \tau_{aux}$ and all k-tuples $\vec{a}$.*

Proof. The claim follows immediately from the invariance of the initialization mapping. □

The following lemma will be useful for the proof of the next theorem.

Lemma 5.6. *Let $\mathcal{P}$ be a* DynQF *program and $\mathcal{S}$ and $\mathcal{T}$ be states of $\mathcal{P}$ with domains S and T. Further let $A \subseteq S$ and $B \subseteq T$ be closed. If $\mathcal{S} \restriction A$ and $\mathcal{T} \restriction B$ are isomorphic via π then $P_\alpha(\mathcal{S}) \restriction A$ and $P_\beta(\mathcal{T}) \restriction B$ are isomorphic via π for all π-respecting modification sequences α, β on A and B.*

Proof. Observe that when A and B are closed and $\mathcal{S} \restriction A$ and $\mathcal{T} \restriction B$ are isomorphic via π then A and B are k-similar via π for arbitrary k. Thus the claim follows from Lemma 5.2. □

Theorem 5.7. Dyn(s-t-Reach) *cannot be maintained in* DynQF *with invariant initialization mapping. This holds even for 1-layered s-t-graphs.*

Proof. We follow the argumentation of the proof of Theorem 4.12.

Towards a contradiction, assume that $\mathcal{P}$ is a DynQF-program with auxiliary schema τ_{aux} and invariant initialization mapping Init which maintains the s-t-reachability query for 1-layered s-t-graphs. Let m be the maximum arity of relation or function symbols in $\tau_{\text{aux}} \cup \{E\}$. Further let n be the number of isomorphism types of structures with at most $m + 2$ elements.

We consider the complete 1-layered s-t-graphs $G_i = (V_i, E_i)$, $2 \leq i \leq n + 2$, with $V_i = \{s, t\} \cup A_i$ and $A_i = \{a_1, \ldots, a_i\}$. Further let $\mathcal{S}_i = (V_i, E_i, \mathcal{A}_i)$ be the state obtained by applying Init to G_i.

We observe that $id[a, a']$ is an automorphism of G_i for all pairs (a, a') of nodes in A_i with $a \neq a'$. Thus, by Lemma 5.5, s and t are the only values that the auxiliary functions in $\mathcal{S}_i$ can assume, and therefore $\mathcal{S}_i \restriction A \cup \{s, t\}$ is closed for any subset A of A_i. Hence, by Lemma 5.6, it is sufficient to find $\mathcal{S}_k$ and $\mathcal{S}_l$ with $k < l$ such that $\mathcal{S}_k$ is isomorphic to $\mathcal{S}_l \restriction V_k$. Then, we can apply the same sequences of modifications as in Theorem 4.12 to reach a contradiction.

Recall that a tuple is diverse, if all components differ pairwise. Since $G_i \simeq_{id[\vec{a},\vec{b}]} G_i$, for two diverse m'-tuples $\vec{a}$ and $\vec{b}$ over A_i with $m' \leq m$, also $\mathcal{S}_i \simeq_{id[\vec{a},\vec{b}]} \mathcal{S}_i$ by the invariance of INIT. In particular $(s, t, \vec{a})$ and $(s, t, \vec{b})$ are of the same isomorphism type.

Since n is the number of isomorphism types of structures of at most $m + 2$ elements, there are two states $\mathcal{S}_k$ and $\mathcal{S}_l$ such that, all diverse m-tuples over A_k and A_l extended by s and t are of the same isomorphism type in $\mathcal{S}_k$ and $\mathcal{S}_l$, respectively. But then $\mathcal{S}_k \simeq \mathcal{S}_l \restriction V_k$. □

6 Lower Bounds for Other Dynamic Queries

In this section we use the lower bounds obtained for the dynamic s-t-reachability query for shallow graphs to establish lower bounds for the dynamic variants of the following Boolean queries

Query:	k-CLIQUE
Input:	A graph G
Question:	Does G contain a k-clique?

Query:	k-COL
Input:	A graph G
Question:	Is G k-colorable?

where k is a fixed natural number. Cliques are usually defined for undirected graphs only. We define a clique in a directed graph to be a set of nodes such that each pair of nodes from the set is connected by an edge. Similarly for colorability.

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 6.1. *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*.

Proof. We prove that DYN(3-CLIQUE) cannot be maintained in binary DYNPROP. Afterwards we sketch the proof for DYN(k-CLIQUE), for arbitrary $k \geq 3$. The graphs used in the proof have a k-Clique if and only if they are not $(k-1)$-colorable. Therefore it follows that DYN(k-COL) cannot be maintained in binary DYNPROP.

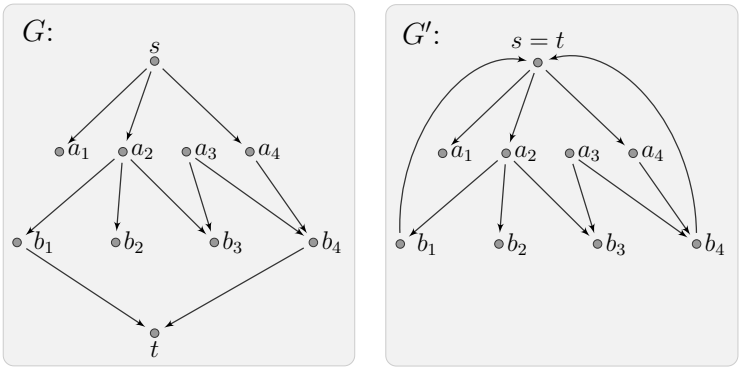

Figure 7: The construction from Proposition 6.1. The s-t-paths (s, a_2, b_1, t) and (s, a_4, b_4, t) in G correspond to the cliques $\{s, a_2, b_1\}$ and $\{s, a_4, b_4\}$ in G'.

More precisely, we show that from a binary DynProp-program $\mathcal{P}'$ for the query Dyn(3-Clique) one can construct a dynamic program $\mathcal{P}$ that maintains the s-t-reachability query for 2-layered s-t-graphs. As the latter does not exist thanks to Theorem 4.7, we can conclude that the former does not exist either.

Let us thus assume that $\mathcal{P}' = (P', \textsc{Init}', Q')$ is a dynamic program for Dyn(3-Clique) with binary auxiliary schema τ'_{aux} and built-in schema τ'_{bi}.

The reduction is very simple. For a 2-layered graph $G = (\{s, t\} \cup A \cup B, E)$, let G' be the graph obtained from G by identifying s and t. Clearly, G has a path from s to t if and only if G' has a 3-clique. See Figure 7 for an illustration.

The dynamic program $\mathcal{P}$ uses the same auxiliary schema as $\mathcal{P}'$, the same initialization mapping and the same built-in schema relations. However, edges (u, t) in E are interpreted as if they were edges (u, s) in E'. More precisely, the update formulas of $\mathcal{P}$ are obtained from those in $\mathcal{P}'$ by replacing every atomic formula $E'(x, y)$ with $(y = s \wedge E(x, t)) \vee (y \neq s \wedge E(x, y))$. Obviously, $\mathcal{P}$ is a dynamic program for s-t-reachability for 2-layered s-t-graphs if $\mathcal{P}'$ is a dynamic program for Dyn(3-Clique), as desired.

For arbitrary k, the construction is similar. The idea is that $\mathcal{P}$ simulates on a graph G the behavior of $\mathcal{P}'$ on $G \otimes K_{k-3}$, that is, the graph that results from G by adding a $(k-3)$-clique and completely connecting it with every node of G. Interestingly, the update formulas of $\mathcal{P}$ are exactly as in the previous reduction to Dyn(3-Clique), as the "virtual" additional $k-3$ nodes are never involved in changes of the graph. However, Init is not the same as $\textsc{Init}'(G)$ but rather the projection of $\textsc{Init}'(G \otimes K_{k-3})$ to the nodes of G. □

Proposition 6.2. *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.*

Proof. The proof approach is the same as for the previous proposition. We prove that DYN(3-CLIQUE) cannot be maintained in DYNQF with invariant initialization. Afterwards we sketch the proof for DYN(k-CLIQUE), for arbitrary $k \geq 3$. The graphs used in the proof have a k-Clique if and only if they are not $(k-1)$-colorable. Therefore it follows that DYN(k-COL) cannot be maintained in DYNQF with invariant initialization mapping.

More precisely, we show that from DYNQF dynamic program $\mathcal{P}'$ with invariant initialization that maintains DYN(3-CLIQUE) one can construct a dynamic program $\mathcal{P}'$ that maintains the s-t-reachability query for 1-layered s-t-graphs. As the latter does not exist thanks to Theorem 5.7, we can conclude that the former does not exist either.

Let us thus assume that $\mathcal{P}' = (P', \textsc{Init}', Q')$ is a dynamic program for DYN(3-CLIQUE) with invariant initialization mapping INIT' and auxiliary schema τ'_{aux}.

We use the following simple reduction. For a 1-layered graph $G = (\{s,t\} \cup A, E)$, let G' be the graph obtained from G by adding an edge (s,t). Clearly, G has a path from s to t if and only if G' has a 3-clique.

The dynamic program $\mathcal{P}$ uses the same auxiliary schema as $\mathcal{P}'$ and the same initialization mapping. The update formulas of $\mathcal{P}$ are obtained from those in $\mathcal{P}'$ by replacing every atomic formula $E'(x,y)$ with $(E(x,y) \vee (x = s \wedge y = t))$. Obviously, $\mathcal{P}$ is a dynamic program for s-t-reachability for 2-layered s-t-graphs if $\mathcal{P}'$ is a dynamic program for DYN(3-CLIQUE), as desired.

For arbitrary k, the construction is similar. The idea is that $\mathcal{P}$ simulates on a graph G the behavior of $\mathcal{P}'$ on $G \otimes (K_{k-3}, K_{k-3})$, that is, the graph that results from G by adding two $(k-3)$-cliques and completely connecting them with every node of G. The update formulas of $\mathcal{P}$ are exactly as in the previous reduction to DYN(3-CLIQUE). However, INIT is not the same as $\textsc{Init}'(G)$ but rather the projection of $\textsc{Init}'(G \otimes (K_{k-3}, K_{k-3}))$ to the nodes of G. By Lemma 5.5, auxiliary functions in $\textsc{Init}(G)$ do not take values from (K_{k-3}, K_{k-3}). Thus $\mathcal{P}$ is a dynamic program for s-t-reachability for 2-layered s-t-graphs if $\mathcal{P}'$ is a dynamic program for DYN(k-CLIQUE). □

7 Subclasses of DynProp

Proving that Reachability cannot be maintained in DYNPROP appears to be non-trivial. A natural question is, whether lower bounds for syntactic fragments of DYNPROP can be proved, without restrictions on the arity of auxiliary relations. Normal form results from [15] (see below) imply that lower bounds for some large fragments cannot be obtained easier than for DYNPROP. In this section we prove that Reachability cannot be maintained in the (very) weak fragment of DYNPROP where update formulas are restricted to be conjunctions of atoms.

Throughout this section we allow arbitrary initialization and no auxiliary functions.

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). We follow the naming schema from [17] and refer to the conjunctive, the negation-free and the conjunctive, negation-free fragment of DynProp as DynPropCQ$^\neg$, DynPropUCQ and DynPropCQ, respectively.

The following theorem implies that lower bounds for DynPropCQ$^\neg$ and DynPropUCQ immediately yield lower bounds for DynProp. In other words, proving lower bounds for those fragments is not easier than proving lower bounds for DynProp itself.

Theorem 7.1 ([15, 16])**.** *Let $\mathcal{Q}$ be a query. Then the following statements are equivalent:*

(a) $\mathcal{Q}$ can be maintained in DynProp*.*

(b) $\mathcal{Q}$ can be maintained in DynPropCQ$^\neg$*.*

(c) $\mathcal{Q}$ can be maintained in DynPropUCQ*.*

The rest of this section is devoted to the proof of the following theorem.

Theorem 7.2. Dyn(s-t-Reach) *is not in* DynPropCQ*.*

To this end, we first prove that the query NonEmptySet from Example 1 cannot be maintained in this fragment. Afterwards we sketch how to adapt this proof for the reachability query.

For technical reasons, the proof assumes a DynPropCQ-program in which no atom contains any variable more than once. We first illustrate by an example how this restriction can be achieved.

Example 2. We consider the following DynPropCQ-program, where, for simplicity, only update formulas for insertions are specified.

$$\phi^R_{\text{INS}}(u; x, y) = S(x, y) \wedge R(x, x)$$
$$\phi^S_{\text{INS}}(u; x, y) = S(x, y)$$

An equivalent DynPropCQ-program in which all update formulas only contain atoms with distinct variables can be obtained by replacing $R(x, x)$ by $R'(x)$ where R' is a fresh unary relation symbol. It then has to be ensured, that $R'(x) \equiv R(x, x)$. This can be achieved by updating R' with the update formula for R, in which x and y are unified.

$$\phi^R_{\text{INS}}(u; x, y) = S(x, y) \wedge R'(x)$$
$$\phi^S_{\text{INS}}(u; x, y) = S(x, y)$$
$$\phi^{R'}_{\text{INS}}(u; x) = S(x, x) \wedge R'(x)$$

Finally we apply the same construction to the atom $S(x,x)$ in $\phi^{R'}_{\text{INS}}$:

$$\begin{aligned}\phi^{R}_{\text{INS}}(u;x,y) &= S(x,y) \wedge R'(x)\\ \phi^{S}_{\text{INS}}(u;x,y) &= S(x,y)\\ \phi^{R'}_{\text{INS}}(u;x) &= S'(x) \wedge R'(x)\\ \phi^{S'}_{\text{INS}}(u;x) &= S'(x)\end{aligned}$$

The process of Example 2 necessarily terminates since there is only a finite number of equality types for the variables of each of the atoms occurring in an update formula. An *equality type* ρ of a set of variables $X = \{x_1, \ldots, x_n\}$ is simply an equivalence relation on X.

Lemma 7.3. *For every* DynPropCQ*-program there is an equivalent* DynPropCQ*-program in which no atom in any update formula contains a variable more than once.*

Proof sketch. For a given DynPropCQ-program $\mathcal{P}$ schema τ, construct an equivalent DynPropCQ-program $\mathcal{P}'$ over schema τ' where τ' contains, for every k-ary relation symbol $R \in \tau$ and every equality type ρ on k variables $x_1, \ldots, x_k$, a relation symbol R^ρ of arity k' where k' is the number of equivalence classes of ρ.

The intention is that $(\mathcal{S}, \beta) \models R(\vec{x})$, for a state R and variable assignment β respecting ρ if and only if $(\mathcal{S}, \beta^\rho) \models R^\rho(\vec{y})$, where β^ρ maps every variable y_i to the value of the i-th equivalence class of ρ under β. This can be ensured along the lines of Example 7.3. □

We prove Theorem 7.4 in a slightly more general setting. A modification α is *honest* with respect to a given state if it does not insert a tuple already present in the input database and does not delete a tuple which is not present in the database. A query is in h-Dyn$\mathcal{C}$ if it can be maintained with $\mathcal{C}$ update programs, for all sequences of honest modifications. It is easy to see that for a class $\mathcal{C}$ closed under boolean operations, the classes Dyn$\mathcal{C}$ and h-Dyn$\mathcal{C}$ coincide. However for weak classes such as DynPropCQ the restriction to honest modifications might make a difference, since update formulas cannot explicitly test (at least not in a straight forward way) whether a modification is honest. Nevertheless, all our proofs work for both kinds of types of modifications.

We prove that h-DynPropCQ (and therefore also DynPropCQ) cannot maintain the query $\exists x U(x)$ from Example 1.

Lemma 7.4. Dyn(NonEmptySet) *is neither in* DynPropCQ *nor in h-*DynPropCQ*.*

Proof. Towards a contradiction, we assume that there is a h-DynPropCQ-program $\mathcal{P} = (P, \text{Init}, Q)$ over schema τ that maintains query $\mathcal{Q}$ defined by $\exists x U(x)$ and, by Lemma 7.3, that no variable occurs more than once in any atom of an update formula of $\mathcal{P}$.

The following notions will be convenient for the proof. The *dependency graph* of a dynamic program $\mathcal{P}$ with auxiliary schema τ has vertex set $V = \tau$ and an edge (R, R') if the relation symbol R' occurs in one of the update formulas for R. The *deletion dependency graph* of $\mathcal{P}$ is defined like the dependency graph except that only update formulas for deletions are used. The *deletion depth* of a relation R is defined as the length of the shortest path from Q to R in the deletion dependency graph.

We start with a simple observation. Let $R(u)$ be a relation atom in the formula $\phi^Q_{\text{DEL}}(u)$ for the 0-ary query relation Q, that is:

$$\phi^Q_{\text{DEL}}(u) \stackrel{\text{def}}{=} \ldots \wedge R(u) \wedge \ldots$$

Further let $\mathcal{S}$ be a state in which the relation U contains two elements $a \neq b$. Then, necessarily, $R^{\mathcal{S}}$ contains a and b, as otherwise deletion of a or b would make Q empty without U becoming empty. This observation can be generalized: if a relation R has "distance k" from Q in the subgraph of the dependency graph induced by DEL-formulas and U contains at least $k+1$ elements, then R must contain all *diverse* tuples over U, that is, tuples that consist of pairwise distinct elements from U.

We prove this observation next, afterwards we look at how the statement of the lemma follows. Using our assumption on non-repeating variables, it is easy to show that the arity of relations of deletion depth k is at most k (at most one plus the arity of the updated relation).

We prove by induction on k that, for each relation R of deletion depth k, and every state $\mathcal{S}$ in which U contains at least $k+1$ elements, R has to contain all diverse tuples over U.

For $k = 0$ this is obvious as Q needs to contain the empty tuple if U is non-empty.

For $k > 0$, let $\mathcal{S}$ be a state such that $U^{\mathcal{S}}$ contains at least $k+1$ elements. Further let R be some arbitrary relation symbol of deletion distance k. Then $R(\vec{x})$ occurs in the update formula $\phi^{R'}_{\text{DEL}}(u; \vec{y})$ of some relation symbol R' of deletion depth $k-1$ for some $\vec{x} = (x_1, \ldots, x_l)$, with $\vec{x} \subseteq \{u\} \cup \vec{y}$. By the above, $l \leq k$ and $\vec{y}$ contains at most $k-1$ variables.

Towards a contradiction, let us assume that there is a diverse k-tuple $\vec{a} = (a_1, \ldots, a_k)$ over $U^{\mathcal{S}}$ that is not in $R^{\mathcal{S}}$. Let $\Theta : \{x_1, \ldots, x_l\} \to U^{\mathcal{S}}$ be the assignment with $\Theta(x_i) = a_i$ and let $\hat{\Theta}$ be some extension of Θ to an injective assignment of $\{u\} \cup \vec{y}$ to elements from $U^{\mathcal{S}}$ (such an assignment exists because $|\{u\} \cup \vec{y}| \leq k < |U|$). Then $\phi^{R'}_{\text{DEL}}(u; \vec{y})$ evaluates to false in state $\mathcal{S}$ under $\hat{\Theta}$ (since $\vec{a} \notin R^{\mathcal{S}}$ by assumption). Thus, deleting $\hat{\Theta}(u)$ from $U^{\mathcal{S}}$ yields a state $\mathcal{S}'$ with $\hat{\Theta}(\vec{y}) \notin R'^{\mathcal{S}'}$. However, $U^{\mathcal{S}'}$ still contains at least k elements and therefore, by induction hypothesis, the relation $R'^{\mathcal{S}'}$ contains every diverse tuple over $U^{\mathcal{S}'}$ and thus, in particular, $\hat{\Theta}(\vec{y})$, the desired contradiction from the assumption that $\vec{a} \notin R^{\mathcal{S}}$.

Now we can complete the proof of Lemma 7.4. Let $\mathcal{S}$ be a state in which the set U contains $m+1$ elements, where m is the maximum (finite) deletion depth of any relation symbol in $\mathcal{P}$. By the claim above, all relations whose symbols

are reachable from Q in the deletion dependency graph of $\mathcal{P}$ contain all diverse tuples over $U^{\mathcal{S}}$. Thus, all relation atoms over tuples from $U^{\mathcal{S}}$ evaluate to true. It is easy to show by induction on the length of modification sequences that this property (applied to $U^{\mathcal{S}'}$) holds for all states $\mathcal{S}'$ that can be obtained from $\mathcal{S}$ by deleting elements from $U^{\mathcal{S}}$. In particular, it holds for any such state in which $U^{\mathcal{S}'}$ contains only one element a. But then, $\phi^{Q}_{\text{DEL}}(a)$ evaluates to true in $\mathcal{S}'$ and thus Q remains true after deletion of a, the desired contradiction to the assumed correctness of $\mathcal{P}$. □

Proof sketch (of Theorem 7.2). Towards a contradiction assume that there is a DynPropCQ-program $\mathcal{P}$ for Dyn(s-t-Reach) over schema τ. We show that a DynPropCQ-program $\mathcal{P}'$ can be constructed from $\mathcal{P}$ such that $\mathcal{P}'$ maintains Dyn(NonEmptySet) under deletions. As the proof of the preceding lemma shows that Dyn(NonEmptySet) cannot be maintained in DynPropCQ even if elements are deleted from U only, this is the desired contradiction.

The intuition behind the construction of $\mathcal{P}'$ is as follows. For sets $U \subseteq A$, the 1-layered graph G with nodes $\{s,t\} \cup A$ and edges $\{(s,a) \mid a \in U\} \cup \{(a,t) \mid a \in A\}$ naturally corresponds to the instance I of Dyn(NonEmptySet) over domain A with set U. The deletion of an element a from U in I corresponds to the deletion of the edge (s,a) from G. Using this correspondence, the program $\mathcal{P}'$ essentially maintains the same auxiliary relations as $\mathcal{P}$. When a is deleted from U then $\mathcal{P}'$ simulates $\mathcal{P}$ after the deletion of (s,a).

A complication arises from the fact that Dyn(NonEmptySet) does not have constants s and t. Therefore the program $\mathcal{P}'$ encodes the relationship of s and t to elements from A by using additional auxiliary relations. More precisely, for every k-ary relation symbol $R \in \tau$ and every tuple $\rho = (\rho_1, \ldots, \rho_k)$ over $\{\bullet, s, t\}$, the program $\mathcal{P}'$ has a fresh l-ary relation symbol R^ρ where l is the number of ρ_i's with $\rho_i = \bullet$. The intention is as follows. Let $i_1 < \ldots < i_l$ such that $\rho_{i_j} = \bullet$. With every l-tuple $\vec{u} = (y_1, \ldots, y_l)$ of variables we associate the tuple $\vec{u}^\rho = (u_1^\rho, \ldots, u_k^\rho)$ of terms from $\{s, t, y_1, \ldots, y_l\}$, where (1) $u_i^\rho = s$ if $\rho_i = s$, (2) $u_i^\rho = t$ if $\rho_i = t$, and (3) $u_{i_j}^\rho = y_j$, for $j \in \{1, \ldots, l\}$. Analogously, we define $\vec{a}^\rho$ for tuples $\vec{a} = (a_1, \ldots, a_l)$ over A. Then $\mathcal{P}'$ ensures that $\vec{a} \in R^\rho$ in some state if and only if $\vec{a}^\rho \in R$ in the corresponding state of $\mathcal{P}$.

Update formulas $\phi^{R^\rho}_{\text{DEL}U}(v; x_1, \ldots, x_l)$ of $\mathcal{P}'$ are obtained from update formulas $\phi^{R}_{\text{DEL}E}(u, v; x_1, \ldots, x_k)$ of $\mathcal{P}$ in two steps. First, from $\phi^{R}_{\text{DEL}E}$ a formula ϕ' is constructed by replacing every occurrence of x_i by x_i^ρ and replacing every occurrence of u by s. Then $\phi^{R^\rho}_{\text{DEL}U}$ is obtained from ϕ' by replacing every atom $T(\vec{w})$ in $\phi^{R}_{\text{DEL}E}$ by $T^\rho(\vec{y})$, for the unique tuple $\vec{y}$ of variables and the unique tuple ρ, for which $\vec{y}^\rho = \vec{w}$.

Now, $\mathcal{P}'$ yields the same query result after deletion of elements $a_1, \ldots, a_m$ as $\mathcal{P}$ after deletion of edges $(s,a_1), \ldots, (s,a_m)$. Hence the program $\mathcal{P}'$ maintains Dyn(NonEmptySet) under deletions. This is a contradiction. □

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 non-trivial 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 modification sequences; or non-constant but very regular modification sequences are used. In the latter case, the modifications do not depend on previous changes to the auxiliary data (as, e.g., in [6] and in this paper). Finding techniques that adapt to changes could be a good starting point.

References

[1] Guozhu Dong, Leonid Libkin, and Limsoon Wong. Incremental recomputation in local languages. *Inf. Comput.*, 181(2):88–98, 2003.

[2] Guozhu Dong and Jianwen Su. Arity bounds in first-order incremental evaluation and definition of polynomial time database queries. *J. Comput. Syst. Sci.*, 57(3):289–308, 1998.

[3] Guozhu Dong and Louxin Zhang. Separating auxiliary arity hierarchy of first-order incremental evaluation systems using (3k+1)-ary input relations. *Int. J. Found. Comput. Sci.*, 11(4):573–578, 2000.

[4] Kousha Etessami. Dynamic tree isomorphism via first-order updates. In Alberto O. Mendelzon and Jan Paredaens, editors, *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, USA*, pages 235–243. ACM Press, 1998.

[5] Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. The dynamic complexity of formal languages. *ACM Trans. Comput. Log.*, 13(3):19, 2012.

[6] Erich Grädel and Sebastian Siebertz. Dynamic definability. In Alin Deutsch, editor, *15th International Conference on Database Theory, ICDT '12, Berlin, Germany, March 26-29, 2012*, pages 236–248. ACM, 2012.

[7] R.L. Graham, B.L. Rothschild, and J.H. Spencer. *Ramsey Theory*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1990.

[8] William Hesse. The dynamic complexity of transitive closure is in DynTC^0. In *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings*, pages 234–247, 2001.

[9] William Hesse. *Dynamic Computational Complexity*. PhD thesis, University of Massachusetts Amherst, 2003.

[10] Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. In *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 24-26, 1994, Minneapolis, Minnesota*, pages 210–221. ACM Press, 1994.

[11] Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. *J. Comput. Syst. Sci.*, 55(2):199–209, 1997.

[12] Mihai Patrascu and Erik D. Demaine. Lower bounds for dynamic connectivity. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 546–553. ACM, 2004.

[13] Sylvain Schmitz and Ph. Schnoebelen. Multiply-recursive upper bounds with Higman's lemma. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II*, pages 441–452, 2011.

[14] Volker Weber and Thomas Schwentick. Dynamic complexity theory revisited. *Theory Comput. Syst.*, 40(4):355–377, 2007.

[15] Thomas Zeume and Thomas Schwentick. On the quantifier-free dynamic complexity of reachability. In Krishnendu Chatterjee and Jiri Sgall, editors, *MFCS*, volume 8087 of *Lecture Notes in Computer Science*, pages 837–848. Springer, 2013.

[16] Thomas Zeume and Thomas Schwentick. On the quantifier-free dynamic complexity of reachability. *CoRR*, abs/1306.3056, 2013.

[17] Thomas Zeume and Thomas Schwentick. Dynamic conjunctive queries. In Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy, editors, *ICDT*, pages 38–49. OpenProceedings.org, 2014.