The Dynamic Descriptive Complexity of k-Clique*

Thomas Zeume

TU Dortmund University
thomas.zeume@tu-dortmund.de

July 11, 2021

Abstract

In this work the dynamic descriptive complexity of the k-clique query is studied. It is shown that when edges may only be inserted then k-clique can be maintained by a quantifier-free update program of arity $k-1$, but it cannot be maintained by a quantifier-free update program of arity $k-2$ (even in the presence of unary auxiliary functions). This establishes an arity hierarchy for graph queries for quantifier-free update programs under insertions. The proof of the lower bound uses upper and lower bounds for Ramsey numbers.

1 Introduction

The k-clique query — does a given graph contain a k-clique? — can be expressed by an existential first-order formula with k quantifiers. In this work we study the descriptive complexity of the k-clique query in a setting where edges may be inserted dynamically into a graph. In particular we are interested in lower bounds for the resources necessary to express this query dynamically.

The dynamic descriptive complexity framework (short: dynamic complexity), independently introduced by Dong, Su and Topor [6, 3] and Patnaik and Immerman [16], models the setting of dynamically changing graphs. For a graph subject to changes, auxiliary relations are maintained with the intention to help answering a query $\mathcal{Q}$. When an insertion (or, in the general setting, a deletion) of an edge occurs, every auxiliary relation is updated through a first-order query that can refer to both the graph itself and the auxiliary relations. The query $\mathcal{Q}$ is maintained by such a program, if one designated auxiliary relation

*An extended abstract of this work appeared in the proceedings of the conference Mathematical Foundations of Computer Science 2014 (MFCS 2014)[19]. The author acknowledges the financial support by the German DFG under grant SCHW 678/6-1.

always stores the current query result. The class of all queries maintainable by first-order update programs is called DynFO[1].

Since k-clique can be expressed in existential first-order logic, it can be trivially maintained by a first-order update program. Therefore for characterizing the precise dynamic complexity of this query we need to look at fragments of DynFO. It turns out that k-clique can still be maintained under insertions when the update formulas are not allowed to use quantifiers at all and auxiliary relations may only have restricted arity. We obtain the following characterization.

Main result: When only edge insertions are allowed then k-clique ($k \geq 3$) can be maintained by a quantifier-free update program of arity $k-1$, but it cannot be maintained by a quantifier-free update program of arity $k-2$.

Actually we prove that every property expressible by a positive existential first-order formula with k quantifiers and, possibly, negated equality atoms can be maintained by a $(k-1)$-ary quantifier-free program under insertions.

In order to understand why the lower bound contained in the above result is interesting, we shortly discuss the status quo of lower bound methods for the dynamic complexity framework. Up to now very few lower bounds are known; all of them for fragments of DynFO obtained by either bounding the arity of the auxiliary relations or by restricting the usage of quantifiers (or by restricting both). Usually those bounds have been stated only for the setting where both insertions and deletions are allowed. We emphasize that our lower bound for the insertion-only setting immediately transfers to this more general setting.

The study of bounded arity auxiliary relations was started by Dong and Su [5]. They exhibited concrete graph queries that cannot be maintained in unary DynFO, and they showed that DynFO has an arity hierarchy for general (that is non-graph) queries. Both results rely on previously obtained static lower bounds.

Hesse started the study of the quantifier-free fragment of DynFO (short: DynProp) in [15]. Although this fragment appears to be rather weak at first glance, deterministic reachability [15] and regular languages [11] can be maintained in DynProp. In [11], Gelade et al. also provided first lower bounds. They proved that non-regular languages as well as the alternating reachability problem cannot be maintained in this fragment. The use of very restricted graphs in the proof of the latter result implies that there is a $\exists^*\forall^*\exists^*$FO-definable query that cannot be maintained in DynProp. In [21] it was shown that reachability and 3-clique cannot be maintained in the binary quantifier-free fragment of DynFO.

In general, it is a difficult task to prove lower bounds in the dynamic complexity setting; even when update formulas are restricted to the quantifier-free fragment of first-order logic. We are not at the point where we can, when given a

[1] In this work we stick to the specific framework introduced by Patnaik and Immerman. The main result also holds in the framework of Dong, Su and Topor, even though both frameworks differ in details.

query, apply a set of tools in order to prove that the query cannot be maintained in DynProp. Finding more queries that cannot be maintained in DynProp seems to be a reasonable approach towards finding more generic proof methods.

The lower bound provided by the main result follows this approach and is interesting in two ways. First, it exhibits, for every k, a query in $\exists^k$FO that cannot be maintained in $(k-2)$-ary DynProp, even when only insertions are allowed. We believe that finding simple queries that cannot be maintained will advance the understanding of dynamic complexity. Using the same proof technique as is used for the main result we also exhibit a $\exists^*\forall^*$FO-definable query that cannot be maintained in DynProp; this improves the result from [11]. Second, the main result establishes the first arity hierarchy for graph queries, although for a weak fragment of DynFO and for insertions only.

The proof of the lower bound uses upper and lower bounds for Ramsey numbers. This has been quite curious for us.

A natural question is how far this method to prove lower bounds can be pushed. As an intermediate step between the quantifier-free fragment and DynFO itself, Hesse suggested the study of quantifier-free update programs with auxiliary functions [15]. The main result can be extended as follows.

Extension of the main result: k-clique ($k \geq 3$) cannot be maintained by a quantifier-free update program of arity $k-2$ with unary auxiliary functions.

So far there have been only two lower bounds for dynamic classes with auxiliary functions. Alternating reachability was actually shown to be not maintainable in the quantifier-free fragment of DynFO even in the presence of a successor and a predecessor function [11]. Further, in [21], it was shown that reachability cannot be maintained in unary DynProp with unary auxiliary functions. Thus our extension is a first lower bound for arbitrary unary auxiliary functions and k-ary auxiliary relations, for every fixed k. We also explain why the lower bound technique does not extend to binary auxiliary functions. To this end we show that binary DynQF can maintain every boolean graph property when the domain is large with respect to the actually used domain.

A preliminary version of this work appeared in [19]. It was without many of the proofs and did not contain the lower bound for a $\exists^*\forall^*$FO-definable query.

Related work Up to now we mentioned only work immediately relevant for this work. For the interested reader we give a short list of further related work.

Further lower bounds have been shown in [1, 2, 12]. Further upper bounds have been shown in [10, 14, 18, 12]. Many other aspects such as whether the auxiliary relations are determined by the current structure (see e.g. [17, 4, 12]) and the presence of an order (see e.g. [12]) have been studied.

Outline In Section 2 we fix some of our notations and in Section 3 we recapitulate the formal dynamic complexity framework. In Sections 4 and 5 we

prove the upper and lower bound of the main result, respectively. In Section 6 we study the extension of DynProp by auxiliary functions.

Acknowledgement I am grateful to Thomas Schwentick for encouraging discussions and many suggestions for improving a draft of this work. Further I thank Samir Datta for fruitful discussions while he was visiting Dortmund. I thank Nils Vortmeier for proofreading.

2 Preliminaries

We fix some of our notations. Most notations are reused from [21]. The reader can feel free to skip this section and return when encountering unknown notations.

A *domain* D is a finite set. A (relational) *schema* 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 $c \in \tau_{\text{const}}$ an element (called *constant*) from D.

A *τ-structure* $\mathcal{S}$ is a pair $(D, \mathcal{D})$ where $\mathcal{D}$ is a database with schema τ and domain D. 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 $\mathcal{S} \restriction D'$.

A tuple $\vec{a} = (a_1, \ldots, a_k)$ is *$\prec$-ordered* with respect to a linear order[2] $\prec$ of the domain, if $a_1 \prec \ldots \prec a_k$. The *$k$-ary atomic type* $\langle \mathcal{S}, \vec{a} \rangle$ of $\vec{a}$ over D with respect to $\mathcal{S}$ 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. As we only consider atomic types here, we will often simply say type instead of atomic type.

For a set A, denote by A^k the set of all k-tuples over A and, following [13], by $[A]^k$ the set of all k-element subsets of A. A *k-hypergraph* G is a pair (V, E) where V is a set and E is a subset of $[V]^k$. If $E = [V]^k$ then G is called *complete*. An *r-coloring col* of G is a mapping that assigns to every edge in E a color from $\{1, \ldots, r\}$. A *r-colored k-hypergraph* is a pair (G, col) where G is a k-hypergraph and *col* is a r-coloring of G. If the name of the r-coloring is not important we also say *G is r-colored*.

A (directed) graph $G = (V, E)$ is in k-Clique if V contains k nodes $v_1, \ldots, v_k$ such that $(v_i, v_j) \in E$ or $(v_j, v_i) \in E$ for all $1 \leq i, j \leq k$.

3 Dynamic Setting

The following introduction to dynamic descriptive complexity is borrowed from previous work [21, 20]. Although the focus of this work is on maintaining the k-clique query under insertions, we introduce the general dynamic complexity framework in order to be able to give a broader discussion of concrete results.

[2] All linear orders in this work are strict.

A *dynamic instance* of a query $\mathcal{Q}$ is a pair $(\mathcal{D}, \alpha)$, where $\mathcal{D}$ is a database over some finite domain D and α is a sequence of modifications to $\mathcal{D}$. Here, a *modification* is either an insertion of a tuple over D into a relation of $\mathcal{D}$ or a deletion of a tuple from a relation of $\mathcal{D}$. The result of $\mathcal{Q}$ for $(\mathcal{D}, \alpha)$ is the relation that is obtained by first applying the modifications from α to $\mathcal{D}$ and then evaluating $\mathcal{Q}$ on the resulting database. We use the Greek letters α and β to denote modifications as well as modification sequences. The database resulting from applying a modification α to a database $\mathcal{D}$ is denoted by $\alpha(\mathcal{D})$. The result $\alpha(\mathcal{D})$ of applying a sequence of modifications $\alpha \stackrel{\text{def}}{=} \alpha_1 \dots \alpha_m$ to a database $\mathcal{D}$ is defined by $\alpha(\mathcal{D}) \stackrel{\text{def}}{=} \alpha_m(\dots(\alpha_1(\mathcal{D}))\dots)$.

Dynamic programs, to be defined next, consist of an initialization mechanism and an update program. The former yields, for every (input) database $\mathcal{D}$, an initial state with initial auxiliary data. The latter defines the new state of the dynamic program for each possible modification.

A *dynamic schema* is a tuple $(\tau_{\text{in}}, \tau_{\text{aux}})$ where τ_{in} and τ_{aux} are the schemas of the input database and the auxiliary database, respectively. For the moment schema τ_{aux} may not contain constant symbols. This will be adapted in Section 6. We always let $\tau \stackrel{\text{def}}{=} \tau_{\text{in}} \cup \tau_{\text{aux}}$.

Definition 1. (Update program) An *update program* $\mathcal{P}$ over a dynamic schema $(\tau_{\text{in}}, \tau_{\text{aux}})$ is a set of first-order formulas (called *update formulas* in the following) that contains, for every relation symbol R in τ_{aux} and every $\delta \in \{\text{INS}_S, \text{DEL}_S\}$ where S is a relation symbol from τ_{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 $(\tau_{\text{in}}, \tau_{\text{aux}})$ is a structure $(D, \mathcal{I}, \mathcal{A})$ where D is a finite domain, $\mathcal{I}$ is a database over the input schema (the *current database*) and $\mathcal{A}$ is a database over the auxiliary schema (the *auxiliary database*).

The *semantics of update programs* is as follows. Let P be an update program, $\mathcal{S} = (D, \mathcal{I}, \mathcal{A})$ be a program state and $\alpha = \delta(\vec{a})$ a modification where $\vec{a}$ is a tuple over D and $\delta \in \{\text{INS}_S, \text{DEL}_S\}$ for some $S \in \tau_{\text{in}}$. If P is in state $\mathcal{S}$ then the application of α yields the new state $\mathcal{P}_\alpha(\mathcal{S}) \stackrel{\text{def}}{=} (D, \alpha(\mathcal{I}), \mathcal{A}')$ where, in $\mathcal{A}'$, a relation symbol $R \in \tau_{\text{aux}}$ is interpreted by $\{\vec{b} \mid \mathcal{S} \models \phi_\delta^R(\vec{a}; \vec{b})\}$. The effect $P_\alpha(\mathcal{S})$ of applying a modification sequence $\alpha \stackrel{\text{def}}{=} \alpha_1 \dots \alpha_m$ to a state $\mathcal{S}$ is the state $P_{\alpha_m}(\dots(P_{\alpha_1}(\mathcal{S}))\dots)$.

Definition 2. (Dynamic program) A *dynamic program* is a triple (P, INIT, Q), where

- P is an update program over some dynamic schema $(\tau_{\text{in}}, \tau_{\text{aux}})$,
- INIT is a mapping that maps τ_{in}-databases to τ_{aux}-databases, and
- $Q \in \tau_{\text{aux}}$ is a designated *query symbol*.

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

Definition 3. (DynFO and DynProp) DynFO is the class of all dynamic queries that can be maintained by dynamic programs with first-order update formulas and arbitrary initialization mappings. DynProp is the subclass of DynFO, where update formulas are not allowed to use quantifiers. A dynamic program is *k-ary* if the arity of its auxiliary 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.

In the literature, classes with restricted initialization mappings have been studied as well, see [21] for a discussion. The choice made here is not a real restriction as lower bounds proved for arbitrary initialization hold for restricted initialization as well. On the other hand, our upper bounds also hold for other settings of initialization; with the single exception of Theorem 6.5, which requires arbitrary initialization. Furthermore our results also hold in the related setting where domains can be infinite.

4 k-Clique Can Be Maintained under Insertions with Arity $k-1$

In this section we prove that the k-clique query can be maintained in $(k-1)$-ary DynProp when only edge insertions are allowed. Instead of proving this result directly, we show that the class of all semi-positive existential first-order queries can be maintained in DynProp under insertions.

A *positive existential first-order query* over schema τ is a query that can be expressed by a first-order formula of the form $\varphi(\vec{y}) = \exists \vec{x} \psi(\vec{x}, \vec{y})$ where ψ is a quantifier-free formula that contains no negations. *Semi-positive existential first-order queries* may contain literals of the form $z_i \neq z_j$.

We will prove that every semi-positive existential first-order query can be maintained in DynProp when only insertions are allowed. More precisely, it will be shown that $(k-1)$-ary DynProp is sufficient for boolean queries with k existential quantifiers. In particular k-Clique can be maintained in $(k-1)$-ary DynProp. Before turning to the proof we give some intuition.

Example 1. We show how to maintain 3-Clique in binary DynProp under insertions. The very simple idea is to use an additional binary auxiliary relation R that stores all edges whose insertion would complete a 3-clique. Hence a tuple (a_1, a_2) is inserted into R as soon as deciding whether there is a 3-clique containing the nodes a_1 and a_2 only depends on those two nodes. We refer to Figure 1 for an illustration.

Thus the update formula for R is

$$\phi^R_{\text{INS}E}(u,v;x,y) \stackrel{\text{def}}{=} u \neq v \wedge x \neq y \wedge \Big(\big(E\{u,y\} \wedge v = x\big) \vee \big(E\{u,x\} \wedge v = y\big) \\ \vee \big(E\{v,y\} \wedge u = x\big) \vee \big(E\{v,x\} \wedge u = y\big) \Big)$$

where $E\{x,y\}$ is an abbreviation for $E(x,y) \vee E(y,x)$.

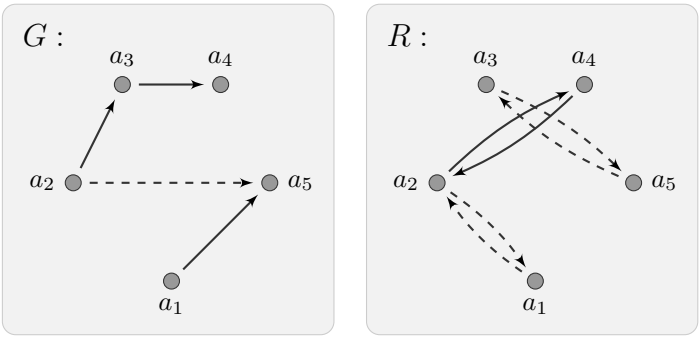

Figure 1: Illustration of the construction from Example 1. Inserting the edge (a_2, a_5) into G leads, e.g., to the insertion of (a_1, a_2) into R since inserting (a_1, a_2) into G would now complete a 3-clique. The tuple (a_1, a_2) is inserted into R by the dynamic program since chosing (u, v, x, y) as (a_2, a_5, a_1, a_2) satisfies the update formula $\phi^R_{\text{INS}E}(u, v; x, y)$.

The update formula for the query symbol Q is $\phi^Q_{\text{INS}E}(u, v; x, y) = Q \vee R(u, v)$. □

The general proof for arbitrary semi-positive existential first-order properties extends the approach from the previous example.

Theorem 4.1. *An ℓ-ary query expressible by a semi-positive existential first-order formula with k quantifiers can be maintained under insertions in $(\ell+k-1)$-ary* DynProp.

Proof. For simplicity we restrict the proof to boolean graph queries. The proof easily carries over to arbitrary semi-positive existential queries.

We give the intuition first. Basically a semi-positive existential sentence with k quantifiers can state which (not necessarily induced) subgraphs with k nodes shall occur in a graph. Therefore it is sufficient to construct a dynamic quantifier-free program that maintains whether the input graph contains a subgraph H. Such a program can work as follows. For every induced, proper subgraph $H' = \{u_1, \ldots, u_m\}$ of H, the program maintains an auxiliary relation that stores all tuples $\vec{a} = (a_1, \ldots, a_m)$ such that inserting H' into $\{a_1, \ldots, a_m\}$ (with a_i corresponding to u_i) yields a graph that contains H.

In particular, auxiliary relations have arity at most $k - 1$ (as only proper subgraphs of H have a corresponding auxiliary relation). Furthermore the graph H is contained in the input graph whenever the value of the 0-ary relation corresponding to the empty subgraph of H is true. In the example above, the relation R is the relation for the subgraph of the 3-clique graph that consists of a single edge, and the designated query relation is the 0-ary relation for the empty subgraph.

Those auxiliary relations can be updated as follows. Assume that a tuple $\vec{a} = (a_1, \ldots, a_m)$ is contained in the relation corresponding to H'. If, after the insertion of an edge with end point a_m, every edge from u_m in H' has a

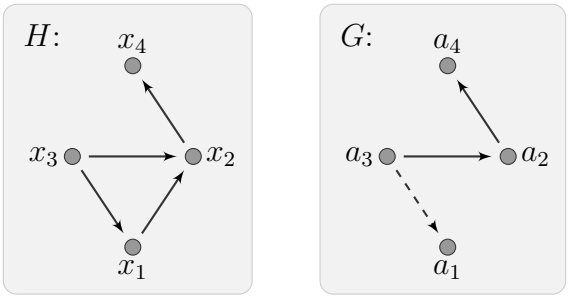

Figure 2: Illustration of the notions used in Theorem 4.1. The graph H is the graph defined by the existential semi-positive formula $\exists x_1 \exists x_2 \exists x_3 \exists x_4 \big(\bigwedge_{i \neq j} x_i \neq x_j \wedge E(x_3, x_1) \wedge E(x_1, x_2) \wedge E(x_3, x_2) \wedge E(x_2, x_4) \big)$. Before inserting the edge (a_3, a_1) into G, the tuple (a_1, a_2, a_3) can be extended to $H_{((x_1,x_2,x_3),x_4)}$, but (a_1, a_2) does not extend to $H_{((x_1,x_2),(x_3,x_4))}$. After inserting the edge (a_3, a_1), the tuple (a_1, a_2) can be extended to $H_{((x_1,x_2),(x_3,x_4))}$ as well.

corresponding edge from a_m in the graph induced by $\{a_1, \ldots, a_m\}$, then the tuple $\vec{a}' = (a_1, \ldots, a_{m-1})$ has to be inserted into the auxiliary relation for the induced subgraph $H' \restriction \{u_1, \ldots, u_{m-1}\}$. This is because inserting the graph $H' \restriction \{u_1, \ldots, u_{m-1}\}$ into $\{a_1, \ldots, a_{m-1}\}$ will now yield a graph that contains H. Observe that for those updates no quantifiers are needed.

In the following we make the intuitive idea outlined above more precise. We first show how a quantifier-free dynamic program can maintain whether the input graph contains a certain (not necessarily induced) subgraph. Afterwards we show how to combine the programs for several subgraphs in order to maintain an arbitrary semi-positive existential formula.

For the first step it will be technically easier not to speak about subgraphs H' of H (as in the intuition above) but to work with partitions of H. We introduce this notion as well as other useful notions next. Let H be a graph. A tuple $(\vec{y}, \vec{z})$ is called a *partition* of H if it contains every node of H exactly once. The subgraph of H induced by $\vec{y}$ is denoted by $H \restriction \vec{y}$; the graph obtained from H by removing the edges of $H \restriction \vec{y}$ is denoted by $H_{(\vec{y},\vec{z})}$.

Now let $G = (V, E)$ and $H = (V', E')$ be graphs, and let $(\vec{y}, \vec{z})$ be an arbitrary partition of H with $|\vec{y}| = \ell$. We say that an ℓ-ary tuple $\vec{a}$ can be extended to $H_{(\vec{y},\vec{z})}$, if there is a $|\vec{z}|$-tuple $\vec{b}$ such that the mapping π defined by $\pi(\vec{y}, \vec{z}) \stackrel{\text{def}}{=} (\vec{a}, \vec{b})$ maps edges in $H_{(\vec{y},\vec{z})}$ to edges in G. Intuitively $\vec{a}$ can be extended to $H_{(\vec{y},\vec{z})}$ when deciding whether H is a subgraph of G, where $\vec{y}$ corresponds to $\vec{a}$, depends only on $\vec{a}$ and not on nodes of G not contained in $\vec{a}$. See Figure 2 for an illustration.

Let $\vec{a} = (a_1, \ldots, a_\ell)$ be a tuple that can be extended to $H_{(\vec{y},\vec{z})}$. Then a node a_i is called *saturated* with respect to a partition $(\vec{y}, \vec{z})$ and $\vec{a}$ if (a_i, a_j) (respectively (a_j, a_i)) is an edge in G whenever (y_i, y_j) (respectively (y_j, y_i)) is an edge in H. A tuple (c, d) is *critical* for a_i with respect to a partition $(\vec{y}, \vec{z})$ and $\vec{a}$ if a_i is not saturated in G but it is saturated in $G + (c, d)$. In Figure 2, the tuple (a_3, a_1) is critical for a_3 with respect to the partition $((x_1, x_2, x_3), x_4)$

and the tuple (a_1, a_2). Observe that therefore the insertion of the edge (a_3, a_1) yields a graph where (a_1, a_2) can be extended to $H_{((x_1,x_2),(x_3,x_4))}$.

We are now ready to construct a DynProp-program $\mathcal{P}$ that maintains whether the input graph contains a graph H as (not necessarily induced) subgraph. The program $\mathcal{P}$ has an auxiliary relation $R_{(\vec{y},\vec{z})}$ of arity $|\vec{y}|$ for every partition $(\vec{y}, \vec{z})$ of H with $|\vec{z}| \geq 1$. The intention is that, for a state $\mathcal{S}$ with input graph G, a tuple $\vec{a}$ is in $R^{\mathcal{S}}_{(\vec{y},\vec{z})}$ whenever $\vec{a}$ extends to $H_{(\vec{y},\vec{z})}$ in G. Thus $R_{(\vec{y},\vec{z})}$ corresponds to the auxiliary relation for $H \restriction \vec{y}$ in the intuitive explanation above. The condition $|\vec{z}| \geq 1$ ensures that the auxiliary relations are of arity at most $|H| - 1$.

Before sketching the construction of the update formulas it is illustrative to see what happens when inserting the edge (a_3, a_1) in Figure 2. We observed above that this yields a graph where (a_1, a_2) can be extended to $H_{((x_1,x_2),(x_3,x_4))}$. Therefore (a_1, a_2) should be inserted into the auxiliary relation $R_{((x_1,x_2),(x_3,x_4))}$. However, this update of $R_{((x_1,x_2),(x_3,x_4))}$ can be made without quantifiers since it is sufficient to verify that (a_1, a_2, a_3) is already in $R_{((x_1,x_2,x_3),x_4)}$ and that (a_1, a_3) was critical. This involves the nodes a_1, a_2 and a_3 only.

In general, when an edge e is inserted, the update formulas of $\mathcal{P}$ check for which nodes and partitions the edge is critical; and adapt the auxiliary relations accordingly.

For updating a relation $R_{(\vec{y},\vec{z})}$ with $\vec{y} = (y_1, \ldots, y_\ell)$ and $\vec{z} = (z_1, \ldots, z_{k-\ell})$ the update formula $\phi^R_{\text{INS } E}(u, v; \vec{y})$ has to check whether there is some $R_{(\vec{y}',\vec{z}')}$ with $\vec{y}' = (y_1, \ldots, y_i, z_j, y_{i+1} \ldots, y_\ell)$ and $\vec{z}' = (z_1, \ldots, z_{j-1}, z_{j+1}, \ldots, z_{k-\ell})$ such that the insertion of (u, v) saturates z_j. It is also possible that the insertion of a single edge saturates two nodes, this case is very similar and will not be treated in detail here.

The formula $\phi^R_{\text{INS } E}(u, v; \vec{y})$ is a conjunction of formulas φ_u, φ_v and $\varphi_{u,v}$ responsible for dealing with the cases where u, v and both u and v are being saturated. We only exhibit φ_u:

$$\varphi_u \stackrel{\text{def}}{=} \bigvee_{\substack{\text{For all } (\vec{y}',\vec{z}') \text{ with} \\ \vec{y}'=(y_1,\ldots,y_i,z_j,y_{i+1},\ldots,y_\ell) \\ \vec{z}'=(z_1,\ldots,z_{j-1},z_{j+1},\ldots,z_{k-\ell})}} \Big(R_{(\vec{y}',\vec{z}')}(y_1, \ldots, y_i, u, y_{i+1}, \ldots, y_\ell) \wedge \bigwedge_{i'} u \neq y_{i'} \wedge \bigwedge_{(z_j,y_{i'}) \in H\restriction\vec{y}'} E(u, y_{i'}) \wedge \bigwedge_{(y_{i'},z_j) \in H\restriction\vec{y}'} E(y_{i'}, u) \Big)$$

The other formulas are very similar. This completes the construction of $\mathcal{P}$.

It remains to construct a quantifier-free dynamic program for an arbitrary semi-positive existential formula using quantifier-free programs for subgraphs. To this end let $\varphi = \exists \vec{x} \psi(\vec{x})$ be an arbitrary semi-positive existential first-order formula. We show how to translate φ into an equivalent disjunction of formulas φ_i of the form

$$\varphi_i = \exists \vec{x}_i \bigwedge_{y,y' \in \vec{x}_i} \Big(y \neq y' \wedge \psi_i(\vec{x}_i) \Big)$$

where each ψ_i is a conjunction of atoms over $\{E\}$ and $|\vec{x}_i| \leq |\vec{x}|$.

Observe that the quantifier-free part of each φ_i encodes a subgraph H_i. Hence a graph G satisfies φ if and only if one of the graphs H_i is a subgraph of G. Thus a program maintaining the query defined by φ can be constructed by combining the dynamic programs for all H_i in a straightforward way.

We now sketch how to translate φ into the form stated above. First φ is rewritten as disjunction of conjunctive queries, that is as $\bigvee_i \exists \vec{y}_i \gamma_i(\vec{y}_i)$ where each γ_i is a conjunction of positive literals and literals of the form $x \neq x'$. Afterwards each $\exists \vec{y}_i \gamma_i(\vec{y}_i)$ is rewritten into an equivalent disjunction over all equality types on the variables in $\vec{y}_i$, that is as

$$\bigvee_{\varepsilon} \exists \vec{y}_{i,\varepsilon} \Big(\bigwedge_{y,y' \in \vec{y}_{i,\varepsilon}} y \neq y' \wedge \varphi_{i,\varepsilon}(\vec{y}_{i,\varepsilon}) \Big)$$

where ε is over all equality types and $\varphi_{i,\varepsilon}$ is a conjunction of atoms over $\{E\}$. □

5 k-Clique Cannot Be Maintained with Arity $k-2$

In this section we prove that the k-clique query cannot be maintained by a $(k-2)$-ary quantifier-free update program when $k \geq 3$. The proof uses two main ingredients; the Substructure Lemma from [11, 21] and a new Ramsey-like lemma. We state those lemmas next. Towards the end of this section we apply the lower bound technique presented in this section to show that there is a first-order property expressible by a formula with only one quantifier alternation which cannot be maintained in DynProp (with arbitrary arity).

For the convenience of the reader we recall the intuition for the Substructure Lemma as presented in [21]. 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 $\mathcal{S}$, then the auxiliary data of $\mathcal{A}$ is not affected by information outside $\mathcal{A}$. In particular, two isomorphic substructures $\mathcal{A}$ and $\mathcal{B}$ remain isomorphic, when corresponding modifications are applied to them.

For stating the Substructure Lemma we need the following notion of corresponding modifications in isomorphic structures. Let π be an isomorphism from a structure $\mathcal{A}$ to a structure $\mathcal{B}$. Two modifications $\alpha = \delta(\vec{a})$ on $\mathcal{A}$ and $\alpha = \delta'(\vec{b})$ on $\mathcal{B}$ where $\delta, \delta' \in \{\text{INS}_R, \text{DEL}_R\}$ for some $R \in \tau_{\text{in}}$ are said to be *π-respecting* if $\delta = \delta'$ and $\vec{b} = \pi(\vec{a})$. Two sequences $\alpha_1 \cdots \alpha_m$ and $\beta'_1 \cdots \beta'_m$ of modifications respect π if α_i and α'_i are π-respecting for every $i \leq m$. Recall that $P_\alpha(\mathcal{S})$ denotes the state obtained by executing the dynamic program $\mathcal{P}$ for the modification sequence α from state $\mathcal{S}$.

The Substructure Lemma stated next is illustrated in Figure 3.

Lemma 5.1 (Substructure Lemma [11, 21])**.** *Let $\mathcal{P}$ be a* DynProp*-program 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 $P_\alpha(\mathcal{S}) \restriction A$ and $P_\beta(\mathcal{T}) \restriction B$ are isomorphic via π for all π-respecting modification sequences α, β*

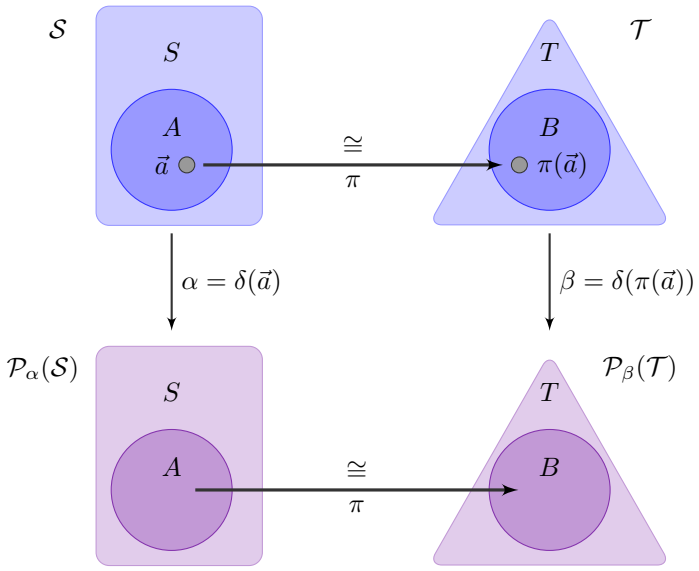

Figure 3: The statement of the Substructure Lemma.

on A and B. In particular, if the query relation of $\mathcal{P}$ is boolean, then it has the same value in $P_\alpha(\mathcal{S})$ and $P_\beta(\mathcal{T})$

The second ingredient exhibits a disparity between upper bounds for Ramsey numbers in k-ary structures and lower bounds for Ramsey numbers in $(k+1)$-dimensional hypergraphs. While the first condition in the following lemma guarantees the existence of a Ramsey clique of size $f(|A|)$ in k-ary structures over A, the second condition states that there is a 2-coloring of the complete $(k+1)$-hypergraph over A that does not contain a Ramsey clique of size $f(|A|)$. This disparity is the key to the lower bound proof.

Lemma 5.2. *Let $k \in \mathbb{N}$ be arbitrary and τ a k-ary schema. Then there is a function $f : \mathbb{N} \to \mathbb{N}$ and an $n \in \mathbb{N}$ such that for every domain A larger than n the following conditions are satisfied:*

(S1) For every τ-structure $\mathcal{S}$ over A and every linear order $\prec$ on A there is a subset A' of A of size $|A'| \geq f(|A|)$ such that all $\prec$-ordered k-tuples over A' have the same type in $\mathcal{S}$.

(S2) The set $[A]^{k+1}$ of all $(k+1)$-hyperedges over A can be partitioned into two sets B and B' such that for every set $A' \subseteq A$ of size $|A'| \geq f(|A|)$ there are $(k+1)$-hyperedges $b, b' \subseteq A'$ with $b \in B$ and $b' \in B'$.

The two lemmas above can be used to obtain the lower bound for the k-clique query as follows. The proof of Lemma 5.2 will be presented afterwards.

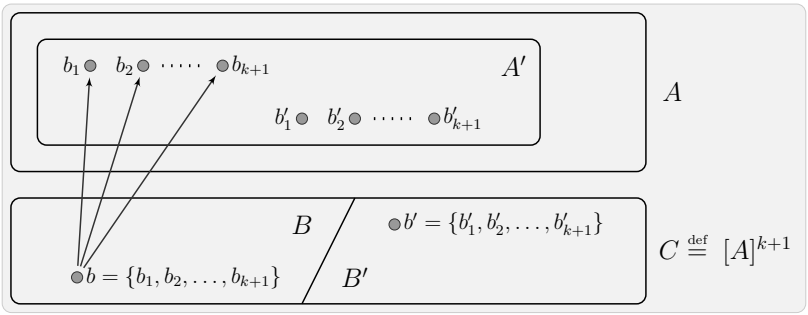

Figure 4: The construction from the proof that $(k+2)$-clique cannot be maintained in k-ary DynProp.

Theorem 5.3. *$(k+2)$-Clique $(k \geq 1)$ cannot be maintained under insertions by a k-ary DynProp-program.*

Proof. Towards a contradiction assume that there is a k-ary DynProp-program $\mathcal{P}$ over schema τ that maintains $(k+2)$-Clique. Let n and f be as in Lemma 5.2. For a set A larger than n let $\prec$ be an arbitrary order on A and let $D \stackrel{\text{def}}{=} A \uplus C$ be a domain with $C \stackrel{\text{def}}{=} [A]^{k+1}$. Further let B, B' be the partition of $[A]^{k+1}$ guaranteed to exist by (S2) in Lemma 5.2.

We consider a state $\mathcal{S}$ over domain D where the input graph G contains the following edges:

$$\{(b, b_1), (b, b_2), \ldots, (b, b_{k+1}) \mid b = \{b_1, b_2, \ldots, b_{k+1}\} \in B\}$$

See Figure 4 for an illustration.

By Condition (S1) there is a subset $A' \subseteq A$ of size $|A'| \geq f(|D|)$ such that all ordered k-tuples over A' have the same τ-type in $\mathcal{S}$. Then by (S2) there are $(k+1)$-hyperedges $b, b' \subseteq A'$ with $b \in B$ and $b' \in B'$. Without loss of generality $b = \{b_1, b_2, \ldots, b_{k+1}\}$ with $b_1 \prec \ldots \prec b_{k+1}$ and $b' = \{b'_1, b'_2, \ldots, b'_{k+1}\}$ with $b'_1 \prec \ldots \prec b'_{k+1}$. By construction of the graph G, all elements in b are connected to the node $b \in C$ while there is no node in C connected to all elements of b'. Thus applying the modification sequences

(α) Insert the edges (b_i, b_j) in lexicographic order with respect to $\prec$.

(β) Insert the edges (b'_i, b'_j) in lexicographic order with respect to $\prec$.

yields one graph with a $(k+2)$-clique and one graph without a $(k+2)$-clique, respectively. However, by the Substructure Lemma, the program $\mathcal{P}$ yields the same result since the substructures induced by $\vec{b} = (b_1, \ldots, b_{k+1})$ and $\vec{b}' = (b'_1, \ldots, b'_{k+1})$ are isomorphic. This is the desired contradiction. □

In the following we prove Lemma 5.2. The *k-dimensional Ramsey* number for r colors and clique-size l, denoted by $R_k(l; r)$, is the smallest number n such that

every r-coloring of a complete k-hypergraph with n nodes has a monochromatic clique of size l. The *tower function* $\mathrm{tow}_k(n)$ is defined by

$$\mathrm{tow}_k(n) \stackrel{\mathrm{def}}{=} 2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}$$

with $(k-1)$ many 2's. The following classical result for asymptotic bounds on Ramsey numbers due to Erdős, Hajnal and Rado is the key to prove Lemma 5.2. The concrete formulation is from [7].

Theorem 5.4. *[9, 8] Let k, ℓ and r be positive integers. Then there are positive constants c_k, $c_{k,r}$ and ℓ_k such that*

(a) $R_k(\ell; r) \leq tow_k(c_{k,r}\ell)$

(b) $R_k(\ell; 2) \geq tow_{k-1}(c_k\ell^2)$ *for all* $\ell \geq \ell_k$

The theorem immediately implies that (T1) Ramsey cliques in r-colored k-dimensional complete hypergraphs are of size at least $\Omega(\log^{(k-1)}(n))$; and that (T2) there are 2-colorings of the $(k+1)$-dimensional complete hypergraphs such that monochromatic cliques are of size $O((\log^{(k-1)}(n))^{\frac{1}{2}})$. Here $\log^{(k)}(n)$ denotes $\log(\log(\ldots(\log n)\ldots))$ with k many log's.

The conditions (T1) and (T2) are formalized and proved in the following corollary.

Corollary 5.5. *Let k and r be integers. There are functions $g \in \Omega(\log^{(k-1)}(n))$ and $h \in O(\sqrt{\log^{(k-2)}(n)})$ such that:*

(a) Every r-colored complete k-hypergraph with n nodes contains a monochromatic clique of size $g(n)$.

(b) The complete k-hypergraph with n nodes can be 2-colored such that every monochromatic clique is of size at most $h(n)$.

Proof. The corollary follows immediately from Theorem 5.4. Define g and h by

$$g(n) \stackrel{\mathrm{def}}{=} \left\lfloor \tfrac{1}{c_{k,r}} \log^{(k-1)}(n) \right\rfloor \text{ and } h(n) \stackrel{\mathrm{def}}{=} \left\lceil \sqrt{\tfrac{1}{c_k} \log^{(k-2)}(n)} \right\rceil + 1$$

where the constants $c_{k,r}$ and c_k are as in Theorem 5.4.

For proving a), consider an arbitrary hypergraph G with n nodes, and an arbitrary r-coloring of G. Then, by Theorem 5.4a), there is a monochromatic clique of size ℓ where ℓ is the maximal number such that $\mathrm{tow}_k(\ell c_{k,r}) \leq n$. The number ℓ is exactly $g(n)$.

For proving b), consider again an arbitrary hypergraph G with n nodes. By Theorem 5.4b), there is a 2-coloring without a monochromatic clique of size ℓ where ℓ is the minimal number such that $n < \mathrm{tow}_{k-1}(\ell^2 c_k)$. Thus the largest monochromatic clique of G is of size at most $h(n)$. □

The conditions (T1) and (T2) are already quite similar to the conditions (S1) and (S2). The major difference is that (T1) is about hypergraphs and not about structures with a k-ary schema.

Fortunately the upper bound from Theorem 5.4 can be generalized to Ramsey numbers for structures. To this end some notions need to be transferred from hypergraphs to structures. Let τ be a k-ary schema, let $\mathcal{S}$ be a τ-structure over domain D and let $\prec$ be a linear order on D. A subset $D' \subseteq D$ of the domain of $\mathcal{S}$ is called an *$\prec$-ordered τ-clique* if all $\prec$-ordered k-tuples $\vec{a} \in D'^k$ have the same τ-type. Recall that the type of a tuple $\vec{a}$ includes information of how $\vec{a}$ relates to the constants of the structure, and therefore all tuples over a τ-clique relate in the same way to constants as well. Denote by $R(\ell; \tau)$ the smallest number n such that every τ-structure with n elements contains an $\prec$-ordered τ-clique of size ℓ, for every order $\prec$ of the domain.

Theorem 5.6. *Let τ be a schema with maximal arity k and let ℓ be a positive integer. Then there is a constant c such that $R(\ell; \tau) \leq tow_k(\ell c)$.*

Proof. The proof of Observation 1' in [11, p. 11] yields this bound. For the sake of completeness we repeat the full construction.

Consider the schema τ and let Γ be the set of all k-ary types for τ. Let $\mathcal{S}$ be a τ-structure over domain D of size $\text{tow}_k(\ell c)$ where $c = |\Gamma|$. Further let $\prec$ be an arbitrary order on D. Define a coloring *col* of the complete k-dimensional hypergraph over domain D with colors Γ as follows. An edge $\{e_1, \ldots, e_k\}$ with $e_1 \prec \ldots \prec e_k$ is colored by the type $\langle S, e_1, \ldots, e_k \rangle$. By Theorem 5.4 there is an induced monochromatic sub-k-hypergraph with domain $D' \subseteq D$ with $|D'| \geq \ell$. By the definition of the coloring *col*, two $\prec$-ordered k-tuples over D' have the same type and therefore D' is a $\prec$-ordered τ-clique in $\mathcal{S}$ as well. □

The previous theorem implies that Ramsey cliques in k-ary structures are of size at least $\Omega(\log^{(k-1)}(n))$. The proof is analogous to the proof of Corollary 5.5.

Corollary 5.7. *Let τ be a schema with maximal arity k. There is a function $g \in \Omega(\log^{(k-1)}(n))$ such that every τ-structure with n elements contains an ordered τ-clique of size $g(n)$.*

It remains to prove Lemma 5.2.

Proof (of Lemma 5.2). Let $k \in \mathbb{N}$ be arbitrary and let τ be a k-ary schema τ. Choose $f \stackrel{\text{def}}{=} g$ where $g \in \Omega(\log^{(k-1)}(n))$ is the function from Corollary 5.7. We show that there is an n such that f satisfies the conditions (S1) and (S2) for all domains larger than n.

Let $h \in O(\sqrt{\log^{(k-1)}(n)})$ be the function guaranteed to exist for $k+1$ by Corollary 5.5b). Then $h \in o(f)$, and therefore there is an n such that $f(n') > h(n')$ for all $n' > n$. Hence for every domain larger than n condition (S1) is satisfied for f due to Corollary 5.7 and condition (S2) is satisfied due to Corollary 5.5. □

Next we apply the lower bound proof technique presented above in order to improve upon a result by Gelade et al. [11]. They provided a lower bound for the

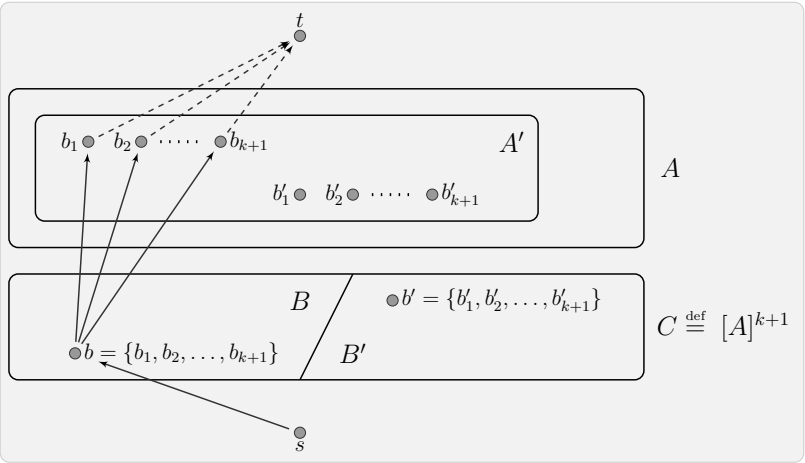

Figure 5: The construction from the proof that the $\exists^*\forall^*$FO-definable query $\varphi \stackrel{\text{def}}{=} \exists x \forall y \big(E(s,x) \wedge (E(y,t) \to E(x,y))\big)$ cannot be maintained in DynProp. The edges inserted by the modification sequence α are dashed.

alternating reachability problem. The use of very restricted graphs in the proof implies that there is a $\exists^*\forall^*\exists^*$FO-definable query that cannot be maintained in DynProp. We show that there is a first-order property expressible by a formula with only one quantifier alternation which cannot be maintained in DynProp. It remains open whether there is a $\exists^*$FO- or $\forall^*$FO-property that is not maintainable in DynProp.

Theorem 5.8. *There is a* $\exists^*\forall^*$FO*-definable query which cannot be maintained by a* DynProp*-program.*

Proof. Consider the graph schema $\{E\}$ extended by two constants s and t. We show that the query $\mathcal{Q}$ defined by $\varphi \stackrel{\text{def}}{=} \exists x \forall y \big(E(s,x) \wedge (E(y,t) \to E(x,y))\big)$ cannot be maintained by any DynProp-program. We remark that it is possible to remove the constants from the following construction by using more existential quantifiers.

The proof is an adaption of the proof of Theorem 5.3. Towards a contradiction assume that there is a k-ary DynProp-program $\mathcal{P}$ over k-ary schema τ that maintains $\mathcal{Q}$. Let n, f, A, C, B, B', $\prec$ be as in the proof of Theorem 5.3.

We consider a state $\mathcal{S}$ over domain D where the input graph G contains, as before, the edges

$$\{(b, b_1), (b, b_2), \ldots, (b, b_{k+1}) \mid b = \{b_1, b_2, \ldots, b_{k+1}\} \in B\}$$

and, additionally, the edges

$$\{(s, b) \mid b = \{b_1, b_2, \ldots, b_{k+1}\} \in B\}$$

See Figure 5 for an illustration.

By Lemma 5.2, we can find tuples $b = \{b_1, b_2, \ldots, b_{k+1}\}$ with $b_1 \prec \ldots \prec b_{k+1}$ and $b' = \{b'_1, b'_2, \ldots, b'_{k+1}\}$ with $b'_1 \prec \ldots \prec b'_{k+1}$ such that (s, t, b) and (s, t, b') have the same τ-type in $\mathcal{S}$.

However, applying the modification sequences

(α) Insert the edges (b_i, t) in lexicographic order with respect to $\prec$.

(β) Insert the edges (b'_i, t) in lexicographic order with respect to $\prec$.

yields one graph that satisfies φ and one graph that does not. However, by the Substructure Lemma, the program $\mathcal{P}$ yields the same result. This is the desired contradiction. □

6 Adding Auxiliary Functions

In quantifier-free update programs, as considered up to here, only the modified and updated tuple as well as the constants can be accessed while updating an auxiliary tuple. Since lower bounds for first-order update programs where arbitrary elements can be accessed in updates seem to be out of reach for the moment, it seems natural to look for extensions of quantifier-free update programs that allow for accessing more elements in some restricted way.

With DynQF, one such extension was proposed by Hesse. In addition to auxiliary relations, a DynQF-program may maintain auxiliary functions. Those functions are updated by update terms that may use function symbols as well as an if-then-else construct.

While Hesse obtained upper bounds for DynQF only, in subsequent work some first lower bounds have been obtained. In [11] it was shown that the alternating reachability problem cannot be maintained in DynProp extended by a fixed successor function and a fixed predecessor function. Later, in [21], the reachability problem was shown to be not maintainable in unary DynProp with additional (updatable) unary auxiliary functions.

In this section we continue the study of DynProp extended by auxiliary functions. In the first part we prove that $(k+2)$-Clique cannot be maintained by a k-ary DynProp-program with unary auxiliary functions, even if only insertions are considered. In the second part we discuss the expressiveness of binary DynQF in large domains and argue why the lower bound technique for the k-clique query does not immediately translate.

Before continuing, we repeat a toy example from [20] which is designed to give an impression of DynQF. For a more formal treatment we refer the reader to [11] and [21].

Example 2. Consider the unary graph query $\mathcal{Q}(x)$ that returns all nodes a of a given graph G with maximal outdegree .

We construct a unary DynQF-program $\mathcal{P}$ that maintains $\mathcal{Q}$ in a unary relation denoted by the designated symbol Q. The program uses two unary functions Succ and Pred that shall encode a successor and its corresponding

predecessor relation on the domain. For simplicity, but without loss of generality, we therefore assume that the domain is of the form $D = \{0, \ldots, n-1\}$. For every state $\mathcal{S}$, the 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 the standard predecessor function (with $\textsc{Pred}^{\mathcal{S}}(0) = 0$). Both functions are initialized accordingly. In the following, when we talk about a *number*, we mean the element whose position in Succ is that number. The program has constants that represent the numbers 0 and 1.

The program $\mathcal{P}$ maintains two unary functions #edges and #nodes. The function #edges counts, for every node a, the number of outgoing edges of a; more precisely $\#\textsc{edges}(a) = b$ if and only if b is the number of outgoing edges of a. The function #nodes counts, for every number a, the number of nodes with a outgoing edges; more precisely $\#\textsc{nodes}(a) = b$ if and only if b is the number of nodes with a outgoing edges. A constant Max shall always point to the number i such that i is the maximal number of outgoing edges from some node in the current graph.

When inserting an outgoing edge (u, v) for a node u that already has a outgoing edges, the counter #edges of u is incremented from a to $a+1$ and all other edge-counters remain unchanged. The counter #nodes of a is decremented, the counter of $a+1$ is incremented, and all other node-counters remain unchanged. The number Max increases if, before the insertion, u was a node with maximal number of outgoing edges. This yields the following update terms:

$$t^{\#\textsc{edges}}_{\textsc{ins}\ E}(u, v; x) \stackrel{\text{def}}{=} \textsc{ite}\Big(\neg E(u, v) \wedge x = u, \textsc{Succ}(\#\textsc{edges}(x)), \#\textsc{edges}(x)\Big)$$

$$t^{\#\textsc{nodes}}_{\textsc{ins}\ E}(u, v; x) \stackrel{\text{def}}{=} \textsc{ite}\Big(\neg E(u, v) \wedge x = \#\textsc{edges}(u), \textsc{Pred}(\#\textsc{nodes}(x)),$$
$$\textsc{ite}\big(\neg E(u, v) \wedge x = \textsc{Succ}(\#\textsc{edges}(u)), \textsc{Succ}(\#\textsc{nodes}(x)),$$
$$\#\textsc{nodes}(x)\big)\Big)$$

$$t^{\textsc{Max}}_{\textsc{ins}\ E}(u, v) \stackrel{\text{def}}{=} \textsc{ite}\Big(\textsc{Max} = \#\textsc{edges}(u) \wedge \neg E(u, v), \textsc{Succ}(u), \textsc{Max}\Big)$$

The ITE-construct chooses, depending on the predicate in its first argument, either the second or the third argument as result term.

The update formula for the designated query symbol Q is as follows:

$$\phi^{Q}_{\textsc{ins}\ E}(u, v; x) \stackrel{\text{def}}{=} t^{\#\textsc{edges}}_{\textsc{ins}\ E}(u, v; x) = t^{\textsc{Max}}_{\textsc{ins}\ E}(u, v)$$

The update terms and and the update formula for deletions are very similar. □

6.1 Lower Bounds for Unary Functions

In this section we generalize the lower bounds obtained so far as follows.

Theorem 6.1. *$(k+2)$-Clique ($k \geq 1$) cannot be maintained under insertions by a k-ary DynProp-program with unary auxiliary functions.*

Theorem 6.2. *There is a* $\exists^*\forall^*$FO*-definable query which cannot be maintained by a* DynProp*-program with unary auxiliary functions.*

The proofs are along the same lines as the proofs of Theorem 5.3 and Theorem 5.8. Instead of the Substructure Lemma for DynProp a corresponding lemma for DynQF from [11, 21] is used. This Substructure Lemma is slightly more involved as it requires to exhibit isomorphic substructures that, additionally, have similar neighbourhoods. Before stating the Substructure Lemma for DynQF and proceeding with the proof we repeat some useful notions from [11, 21].

For the following definitions we fix two structures $\mathcal{S}$ and $\mathcal{T}$ with domains S and T over schema τ. Here, and in the rest of this section, all auxiliary schemas are the disjoint union of a set τ_{rel} of relation symbols and a set τ_{fun} of function symbols. Denote by Terms_τ^m the set of terms of nesting depth at most m with function symbols from τ_{fun}.

The *m-neighborhood* $\mathcal{N}_\mathcal{S}^m(A)$ of a set $A \subseteq S$ is the set of all elements of S that can be obtained by applying a term of nesting depth at most m to a vector of elements from A. More precisely $\mathcal{N}_\mathcal{S}^m(A)$ is the set

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

While for the Substructure Lemma for DynProp it is sufficient to consider two isomorphic substructures, the Substructure Lemma for DynQF also takes their neighborhoods into account. The neighborhoods need to be similar in the following sense. Two subsets $A \subseteq S$, $B \subseteq T$ are *m-similar*, if there is a bijection $\pi : \mathcal{N}_\mathcal{S}^m(A) \to \mathcal{N}_\mathcal{T}^m(B)$ such that

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

We write $A \approx_m^{\pi,\mathcal{S},\mathcal{T}} B$ to indicate that A and B are m-similar via π in $\mathcal{S}$ and $\mathcal{T}$. Two tuples $(a_1, \ldots, a_p)$ and $(b_1, \ldots, b_p)$ are m-similar if $\{a_1, \ldots, a_p\} \approx_m^{\pi,\mathcal{S},\mathcal{T}} \{b_1, \ldots, b_p\}$ via the isomorphism π that maps a_i to b_i, for every $i \in \{1, \ldots, p\}$. Note that if $A \approx_0^{\pi,\mathcal{S},\mathcal{T}} B$, then $\mathcal{S} \restriction A$ and $\mathcal{T} \restriction B$ are τ_{rel}-isomorphic by the first and third property.

Lemma 6.3 (Substructure lemma for DynQF [11, 21])**.** *Let $\mathcal{P}$ be a* DynQF *program and let ℓ be some number. There is a number $m \in \mathbb{N}$ such that for all states $\mathcal{S}$ and $\mathcal{T}$ of $\mathcal{P}$ with domains S and T; and all subsets A and B of S and T, respectively, the following holds. If $A \approx_m^{\pi,\mathcal{S},\mathcal{T}} B$, then $A \approx_0^{\pi,P_\alpha(\mathcal{S}),P_\beta(\mathcal{T})} B$, for all π-respecting modification sequences α and β on A and B of length at most ℓ.*

The lemma is slightly rephrased in comparison to [21]. Here, the number m is independent of the states $\mathcal{S}$ and $\mathcal{T}$. However, this follows immediately from the proof of the lemma in [21].

In order to apply the Substructure Lemma, it is necessary to find similar substructures. To this end the following analogon of Corollary 5.7 for structures with unary functions can be used. Recall that $\log^{(k)}(n)$ denotes $\log(\log(\ldots(\log n)\ldots))$ with k many log's.

Lemma 6.4. *Let τ be a k-ary schema whose function symbols are of arity at most 1; and let $m \in \mathbb{N}$ be an arbitrary number. Then there is a function $g \in \Omega(\log^{(k-1)}(n))$ such that for every τ-structure $\mathcal{S}$ with domain S and every linear order $\prec$ on S, there is a subset $S' \subseteq S$ of size $g(|S|)$ such that all $\prec$-ordered k-tuples over S' are m-similar.*

Proof. The idea is to construct, from the structure $\mathcal{S}$, a purely relational structure $\mathcal{T}$ such that the type of a tuple $\vec{a}$ in $\mathcal{T}$ encodes the type of the whole m-neighborhood of $\vec{a}$ in $\mathcal{S}$. Then Corollary 5.7 is applied to the structure $\mathcal{T}$ in order to obtain S'.

For the construction of $\mathcal{T}$ we need some notions from the proof of Theorem 5.4 in [21]. Let $t_1, \ldots, t_\ell$ be the lexicographic enumeration of $\textsc{Terms}_\tau^m$ with respect to some fixed order of the function symbols. Let the *m-neighborhood vector* $\vec{\mathcal{N}}_{\mathcal{S}}^m(c)$ of an element c in $\mathcal{S}$ be the tuple $(c, t_1(c), \ldots, t_l(c))$. For a tuple $\vec{c} = (c_1, \ldots, c_p)$, the m-neighborhood vector $\vec{\mathcal{N}}_{\mathcal{S}}^m(\vec{c})$ of $\vec{c}$ is the tuple $(\vec{\mathcal{N}}_{\mathcal{S}}^m(c_1), \ldots, \vec{\mathcal{N}}_{\mathcal{S}}^m(c_p))$.

The *m-similarity type* of a k-ary tuple $\vec{a}$ is the (quantifier-free) τ-type of the m-neighborhood tuple $\vec{\mathcal{N}}_{\mathcal{S}}^m(\vec{a})$ of $\vec{a}$. Observe that for fixed m, k and τ there are only finitely many similarity types. Denote the set of all such similarity types by Γ. Further observe that two tuples $\vec{a}$ and $\vec{b}$ with the same m-similarity type are m-similar. This is certified by the bijection that maps $\vec{\mathcal{N}}_{\mathcal{S}}^m(\vec{a})$ to $\vec{\mathcal{N}}_{\mathcal{S}}^m(\vec{b})$ component-wise.

For the construction of $\mathcal{T}$ we assume, without loss of generality, that the schema of $\mathcal{S}$ contains the equality symbol $=$. The structure $\mathcal{T}$ is over the same domain as $\mathcal{S}$ and uses the schema τ_Γ which contains a k-ary relation R_γ for every k-ary similarity type $\gamma \in \Gamma$. A relation $R_\gamma^{\mathcal{T}}$ contains all tuples $\vec{a}$ whose similarity type in $\mathcal{S}$ is γ.

Then, by Corollary 5.7, $\mathcal{T}$ contains an $\prec$-ordered τ-clique S' of size $\Omega(\log^{(k-1)}(|S|))$. We show that all $\prec$-ordered k-tuples over S' are m-similar in the structure $\mathcal{S}$. Therefore let $\vec{a}$ and $\vec{b}$ be two such tuples. By definition of S' they have the same type in $\mathcal{T}$ and therefore, by definition of $\mathcal{T}$, their neighborhood vectors $\vec{\mathcal{N}}_{\mathcal{S}}^m(\vec{a})$ and $\vec{\mathcal{N}}_{\mathcal{S}}^m(\vec{b})$ have the same type in $\mathcal{S}$. Hence, by the observation from above, the tuples $\vec{a}$ and $\vec{b}$ are m-similar. □

Proof (of Theorem 6.1 and Theorem 6.2). The proofs are along the same lines as the proofs of Theorem 5.3 and Theorem 5.8. The only difference is that here we use Lemma 6.4 in order to obtain the set A'. The contradiction is then obtained by using the Substructure Lemma for DynQF and the modification sequences (α) and (β) from Theorem 5.3 and Theorem 5.8, respectively. □

6.2 Discussion of Binary Auxiliary Functions

A natural question is whether the lower bounds transfer to k-ary auxiliary functions with $k \geq 2$. We conjecture that they do, but we will argue that the techniques used so far are not sufficient for proving lower bounds for binary auxiliary functions.

The fundamental difference between unary and binary auxiliary functions is that, on the one hand, unary functions can access elements that depend either on the tuple that has been modified in the input structure or on the auxiliary tuple under consideration but not on both. On the other hand binary functions can access elements that depend on both tuples.

A consequence is that binary DynQF can maintain every boolean graph property when the domain is large with respect to the actually used domain. We make this more precise. In the following we assume that all domains D are a disjoint union of a modifiable domain D^+ and a non-modifiable domain D^-, and that modifications may only involve tuples over D^+. Auxiliary data, however, may use the full domain. A dynamic complexity class $\mathcal{C}$ *profits from padding* if every boolean graph property can be maintained whenever the non-modifiable domain is sufficiently large in comparison to the modifiable domain[3].

Above we have seen that DynProp with unary auxiliary functions does not profit from padding.

Theorem 6.5. *Binary* DynQF *profits from padding.*

Proof. First we show that ternary DynQF profits from padding. Let $\mathcal{Q}$ be an arbitrary boolean graph property. In the following we construct a ternary DynQF program $\mathcal{P}$ which maintains $\mathcal{Q}$ if $2^{|D^+|^2} = |D^-|$. The idea is to identify D^- with the set of all graphs over D^+, that is D^- contains an element c_G for every graph G over D^+. A unary relation $R_\mathcal{Q}$ stores those elements of D^- that correspond to graphs with the property $\mathcal{Q}$. Finally the program maintains a pointer p to the element in D^- corresponding to the current graph over D^+. The pointer is updated upon edge modification by using ternary functions f_{INS} and f_{DEL} initialized by the initialization mapping in a suitable way.

The program $\mathcal{P}$ is over schema $\tau = \{Q, p, f_{\text{INS}}, f_{\text{DEL}}, R_\mathcal{Q}\}$ where p is a constant, f_{INS} and f_{DEL} are ternary function symbols, $R_\mathcal{Q}$ is a unary relation symbol and Q is the designated query symbol.

We present the initialization mapping of $\mathcal{P}$ first. The initial state $\mathcal{S}$ for a graph H is defined as follows. The functions f_{INS} and f_{DEL} are independent of H and defined via

$$f^{\mathcal{S}}_{\text{INS}}(a, b, c_G) = c_{G+(a,b)}$$
$$f^{\mathcal{S}}_{\text{DEL}}(a, b, c_G) = c_{G-(a,b)}$$

for $a, b \in D^+$ and $c_G \in D^-$. For all other arguments the value of the functions is arbitrary. Here $G + (a, b)$ and $G - (a, b)$ denote the graphs obtained by adding

[3]Note that this type of padding differs from the padding technique used by Patnaik and Immerman for maintaining a PTIME-complete problem in DynFO [17].

the edge (a, b) to G and removing the edge (a, b) from G, respectively. The relation $R^{\mathcal{S}}_{\mathcal{Q}}$ contains all c_G with $G \in \mathcal{Q}$. Finally the constant $p^{\mathcal{S}}$ points to c_H.

It remains to exhibit the update formulas. After a modification, the pointer p is moved to the node corresponding to the modified graph, and the query bit is updated accordingly:

$$t^{p}_{\text{INS}}(u, v) = f_{\text{INS}}(u, v, p) \qquad t^{\mathcal{Q}}_{\text{INS}}(u, v) = R_{\mathcal{Q}}(f_{\text{INS}}(u, v, p))$$
$$t^{p}_{\text{DEL}}(u, v) = f_{\text{DEL}}(u, v, p) \qquad t^{\mathcal{Q}}_{\text{DEL}}(u, v) = R_{\mathcal{Q}}(f_{\text{DEL}}(u, v, p))$$

Now we sketch how to modify this construction for binary DynQF. The binary DynQF program maintains $\mathcal{Q}$ on an extended non-modifiable domain that contains

- an element c_G for every graph G over D^+, and
- elements $c_{G,a,\text{INS}}$ and $c_{G,a,\text{DEL}}$ for every graph G over D^+ and every $a \in D^+$.

The intuition is that when an edge (a, b) is inserted into the graph G then the pointer p is moved from c_G to the element $c_{G+(a,b)}$ using the intermediate element $c_{G,a,\text{INS}}$.

For insertion modifications the binary DynQF program maintaining $\mathcal{Q}$ uses two binary functions f_{INS} and s_{INS} that are initialized as

$$f^{\mathcal{S}}_{\text{INS}}(a, c_G) = c_{G,a,\text{INS}}$$
$$s^{\mathcal{S}}_{\text{INS}}(b, c_{G,a,\text{INS}}) = c_{G+(a,b)}$$

for $a, b \in D^+$ and $c_G, c_{G,a,\text{INS}} \in D^-$. For all other arguments the value of the functions is arbitrary.

When an insertion occurs, the pointer and the query bit are updated via

$$t^{p}_{\text{INS}}(u, v) = s_{\text{INS}}(v, f_{\text{INS}}(u, p))$$
$$t^{\mathcal{Q}}_{\text{INS}}(u, v) = R_{\mathcal{Q}}(s_{\text{INS}}(v, f_{\text{INS}}(u, p)))$$

The update formulas and terms for deletions are analogous.

□

Hence the ability to profit from padding distinguishes binary DynQF and DynProp extended by unary functions. Although the proof of the preceding theorem requires the non-modifiable domain to be of exponential size with respect to the modifiable domain, the construction also explains why the lower bound technique from the previous sections cannot be immediately applied to binary DynQF. In the lower bound construction only tuples over the set A are modified, while tuples containing elements from $C = [a]^k$ are not modified. Thus, by treating C as a non-modifiable domain, it can be used to store information as in the proof above. As the modification sequences used in the lower bounds are of length k^2, finding similar substructures in structures with binary auxiliary functions becomes much harder.

7 Conclusion and Future Work

In this work we exhibited a precise dynamic descriptive complexity characterization of the k-clique query when only insertions are allowed. The characterization implies an arity hierarchy for graph queries for DynProp under insertions. Further we exhibited a very simple $\exists^*\forall^*$FO-property which is not maintainable in DynProp. We also discussed the limit of our proof methods.

While proving lower bounds for full DynFO — a major long-term goals in dynamic descriptive complexity — might be really hard to achieve, we believe that the following goals are suitable for both developing new lower bound methods and for further improving the current methods.

Goal 1. *Prove general quantifier-free lower bounds for insertions and deletions for the reachability query and the k-clique query.*

It is known that both queries cannot be maintained in binary DynProp [21]. We conjecture that neither the 3-clique query nor the reachability query can be maintained in DynProp under deletions.

Goal 2. *Find a general framework for proving quantifier-free lower bounds.*

Goal 3. *Find a query that cannot be maintained in binary* DynQF.

In this and previous work on lower bounds in the dynamic descriptive complexity setting, the theorems of Ramsey and Higman played an important role for lower bound proofs. Therefore it appears to be promissing to study the applicability of other combinatorical tools in this context.

References

References

[1] Guozhu Dong, Leonid Libkin, and Limsoon Wong. On impossibility of decremental recomputation of recursive queries in relational calculus and SQL. In Paolo Atzeni and Val Tannen, editors, *Database Programming Languages (DBPL-5), Proceedings of the Fifth International Workshop on Database Programming Languages, Gubbio, Umbria, Italy, 6-8 September 1995*, Electronic Workshops in Computing, page 7. Springer, 1995.

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

[3] Guozhu Dong and Jianwen Su. First-order incremental evaluation of datalog queries. In *Database Programming Languages (DBPL-4), Proceedings of the Fourth International Workshop on Database Programming Languages - Object Models and Languages, Manhattan, New York City, USA, 30 August - 1 September 1993*, pages 295–308, 1993.

[4] Guozhu Dong and Jianwen Su. Deterministic FOIES are strictly weaker. *Ann. Math. Artif. Intell.*, 19(1-2):127–146, 1997.

[5] 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.

[6] Guozhu Dong and Rodney W. Topor. Incremental evaluation of datalog queries. In Joachim Biskup and Richard Hull, editors, *Database Theory - ICDT'92, 4th International Conference, Berlin, Germany, October 14-16, 1992, Proceedings*, volume 646 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 1992.

[7] Dwight Duffus, Hanno Lefmann, and Vojtech Rödl. Shift graphs and lower bounds on Ramsey numbers $r_k(l;r)$. *Discrete Mathematics*, 137(1-3):177–187, 1995.

[8] Paul Erdős, András Hajnal, and Richard Rado. Partition relations for cardinal numbers. *Acta Mathematica Hungarica*, 16(1):93–196, 1965.

[9] Paul Erdös and Richard Rado. Combinatorial theorems on classifications of subsets of a given set. *Proc. London Math. Soc. (3)*, 2:417–439, 1952.

[10] 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.

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

[12] 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.

[13] Ronald L. Graham, Bruce L. Rothschild, and Joel H. Spencer. *Ramsey Theory*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1990.

[14] William Hesse. The dynamic complexity of transitive closure is in DynTC^0. *Theor. Comput. Sci.*, 296(3):473–485, 2003.

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

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

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

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

[19] Thomas Zeume. The dynamic descriptive complexity of k-clique. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 547–558. Springer, 2014.

[20] Thomas Zeume and Thomas Schwentick. Dynamic conjunctive queries. In Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy, editors, *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014.*, pages 38–49. OpenProceedings.org, 2014.

[21] Thomas Zeume and Thomas Schwentick. On the quantifier-free dynamic complexity of reachability. *Inf. Comput.*, 240:108–129, 2015.