Dynamic Conjunctive Queries

Thomas Zeume
TU Dortmund University
thomas.zeume@cs.tu-dortmund.de

Thomas Schwentick
TU Dortmund University
thomas.schwentick@tu-dortmund.de

ABSTRACT

The paper investigates classes of queries maintainable by conjunctive queries (CQs) and their extensions and restrictions in the dynamic complexity framework of Patnaik and Immerman. It studies the impact of union, atomic negation and quantification on the dynamic expressiveness of CQ, for the standard semantics as well as for Δ-semantics.

It turns out that, although there are many different combinations of these features, there exist basically five important fragments for the standard semantics, characterized by the addition of the following features to the possibility to build conjunctions over positive atoms: (1) arbitrary quantification and atomic negation, (2) existential quantification and atomic negation, (3) existential quantification, (4) atomic negation (but no quantification), and (5) conjunction only (and no quantification). Whether all these fragments are actually different remains mostly open, however, it is shown that (4) strictly subsumes (5). The fragments arising from Δ-semantics are also subsumed by the standard fragments (1), (2) and (4).

As a further result, all (statically) FO-definable queries are captured by fragment (2) and a complete characterization of these queries in terms of non-recursive dynamic $\text{CQ}^\neg$-programs is given.

1. INTRODUCTION

The re-evaluation of a fixed query after an update to a huge database can be a time-consuming process; in particular when it is performed from scratch. For this reason previously computed information such as the old query result and (possibly) other auxiliary information is often reused in order to speed up the process.

This maintenance of query results has attracted lots of attention over the last decades of database related research. For relational databases an algorithmic approach (see e.g. [17, 11]) and a declarative approach (see e.g. [7, 16]) have been studied. Here, we continue the study of the declarative approach where query results are updated by queries from some query language.

One possible formalization of this approach is the descriptive dynamic complexity framework (short: dynamic complexity) introduced by Patnaik and Immerman [16]. For a relational database subject to change, auxiliary relations are maintained with the intention to help answering a query $\mathcal{Q}$. When an update to the database, an insertion or deletion of a tuple, occurs, every auxiliary relation is updated through a first-order query that can refer to both, the database and the auxiliary relations. A particular auxiliary relation shall always represent the answer to $\mathcal{Q}$. The class of all queries maintainable in this way, and thus also in the core of SQL, is called DynFO.

The class DynFO is quite powerful. Many queries inexpressible in (static) first-order logic, such as the transitive closure query on undirected graphs [16] and the word problem for context-free languages [9], can be maintained in DynFO. In [8] a query inexpressible in first-order logic extended by a transitive closure operator and counting has been shown to be contained in DynFO. There are no general inexpressibility results for DynFO at all[1].

Towards a deeper understanding of the dynamic setting, two main restrictions of DynFO have been explored in the literature. Dong and Su started the study of restricted auxiliary relations [6]. They restricted the arity of auxiliary relations and obtained inexpressibility results for unary auxiliary relations. On the other hand, Hesse started the exploration of syntactic fragments of DynFO, such as the one obtained by disallowing quantification in update formulas [14]. Inexpressibility results for this particular fragment have been obtained in [9].

In this work we investigate classes of queries maintainable by conjunctive queries and extensions thereof, thus we are following the approach of Hesse.

Conjunctive queries (CQs), that is, in terms of logic, existential first-order queries whose quantifier-free part is a conjunction of atoms, are one of the most investigated query languages. Starting with Chandra and Merlin [2], who analyzed conjunctive queries for relational databases, those queries have been studied for almost every emerging new database model. Usually also the extension by unions (UCQs), by negations ($\text{CQ}^\neg$s) as well as by both unions and negations ($\text{UCQ}^\neg$s or, equivalently, $\exists^*$FO) have been studied. It is folklore that all those classes are distinct for relational databases.

In this work we aim at the following goals.

[1]Except for the trivial ones due to the fact that queries maintainable in DynFO can be computed in polynomial time.

10.5441/002/icdt.2014.08

Goal 1. *Understand the relationship between different classes of dynamic conjunctive queries as well as their relationship to static query classes.*

Our focus for relating variants of dynamic conjunctive queries is on the classes listed above. Some preliminary results have been obtained in [19] for the quantifier-free variants of conjunctive queries.

As for the relationship to static classes, it is interesting to understand whether larger static classes $\mathcal{C}$ can be captured by a dynamic class Dyn$\mathcal{C}'$, for some weaker $\mathcal{C}'$. Up to now only two such results are known, namely, that MSO can be characterized by quantifier-free DynFO on strings and that $\exists^*$FO is captured by quantifier-free DynFO extended by auxiliary functions on general structures [9].

In the framework of Patnaik and Immerman auxiliary relations are always re-defined as a whole after an update. However, in the context of query re-evaluation, it is often convenient to express the new state of an auxiliary relation R in terms of the current relation and some "Delta", that is, by specifying tuples R^+ to be inserted into R and tuples R^- to be removed from R. We refer to the former semantics as *absolute semantics* and to the latter as Δ*-semantics*. Obviously, the choice of the semantics does not affect the expressiveness of an update language that is closed under Boolean operations. However, most of the update languages in this paper lack some Boolean closure properties.

Goal 2. *Understand the relationship between absolute semantics and* Δ*-semantics for conjunctive queries and their variants.*

In this work we contribute to achieve those two goals as follows.

Contributions. For an overview of the relationship of the various dynamic classes of conjunctive queries we refer to Figure 1.

The distinctness of those classes in static relational databases does not translate into the dynamic setting:[2]

- We show that, in many cases, the addition of the union-operator does not yield additional expressive power in the dynamic setting, for example, DynUCQ$^\neg$ = DynCQ$^\neg$, DynUCQ = DynCQ, and DynPropUCQ$^\neg$ = DynPropCQ$^\neg$, where Prop indicates classes without quantifiers.
- In other cases, negation does not increase the expressive power of an update language, e.g., we have DynPropUCQ$^\neg$ = DynPropUCQ and Δ-DynCQ$^\neg$ = Δ-DynCQ.
- Further, often quantifiers can be replaced by their dual quantifiers, e.g., Dyn$\exists^*$FO(= DynUCQ$^\neg$) = Dyn$\forall^*$FO.

Whether DynCQ$^\neg$ = DynCQ remains open. However, a first-step is taken towards the separation of the remaining fragments:

- We show that dynamic conjunctive queries without negations and quantifiers are strictly weaker than the quantifier-free fragment of DynFO.

[2]The notation for classes will be formally introduced in Sections 2 and 4. In general, Δ indicates Δ-semantics, the absence of Δ indicates absolute semantics.

Furthermore, we show that dynamic conjunctive queries extended by negations capture all first-order queries:

- We characterize the class of first-order queries as the class maintainable by non-recursive dynamic Dyn$\exists^*$FO-programs with a single existential quantifier per update formula. This implies that dynamic conjunctive queries extended by negations can maintain all first-order queries.

For the second goal, the main finding is that the difference between absolute and Δ-semantics is much smaller than we had expected.

- The dynamic classes corresponding to FO, CQ$^\neg$ and Prop yield the same expressive power with respect to absolute and Δ-semantics.
- It turns out that conjunctive queries and conjunctive queries with negation coincide with respect to Δ-semantics, that is, in particular, Δ-DynCQ = Δ-DynCQ$^\neg$ and thus, also Δ-DynCQ = DynUCQ$^\neg$.

Choice of setting. The concrete settings under which dynamic complexity has been studied in the literature slightly differ in several aspects. We shortly discuss the most important aspects, what choice we took for this work and why we made this choice.

An important aspect is whether to use a finite and fixed domain, an active domain or an infinite domain. In this work, we follow the framework of Patnaik and Immerman in which the domain is finite and fixed [16]. To maintain a query, a dynamic program has to work uniformly for every domain. This fixed domain framework for a dynamic setting might appear counterintuitive at first sight. However, it allows to study the underlying dynamic mechanisms of dynamic programs, in particular when one is interested to develop lower bound methods. Fixed domains also offer a strong connection to logics and circuit complexity. In incremental evaluation systems (IES), a framework proposed by Dong and Topor [7], active domains are used. This setting is a little closer to real database systems but most results in dynamic complexity hold equally in both frameworks.

Another parameter to choose is how the auxiliary data is initialized. In the setting of Patnaik and Immerman, dynamic programs start from empty databases and the auxiliary data is either initialized by a polynomial time computation or by a formula from the same class as the update formulas. Later this was generalized by Weber and the second author by proposing that dynamic programs start from an arbitrary initial database and auxiliary data initialized by a mapping computable in some given complexity class [18].

In this work we allow for arbitrary initialization mappings. This is motivated by our long term goal to develop lower bound techniques for dynamic programs. While lower bounds in settings with restricted initialization might depend on this restriction, an inexpressibility result in the setting with arbitrary initialization, on the other hand, really shows that a query cannot be *maintained*. A result like DynUCQ = DynCQ is helpful for the development of lower bound techniques, as it shows that for proving lower bounds for DynUCQ it is sufficient to consider DynCQ programs — but also that one has to be aware that lower bounds for

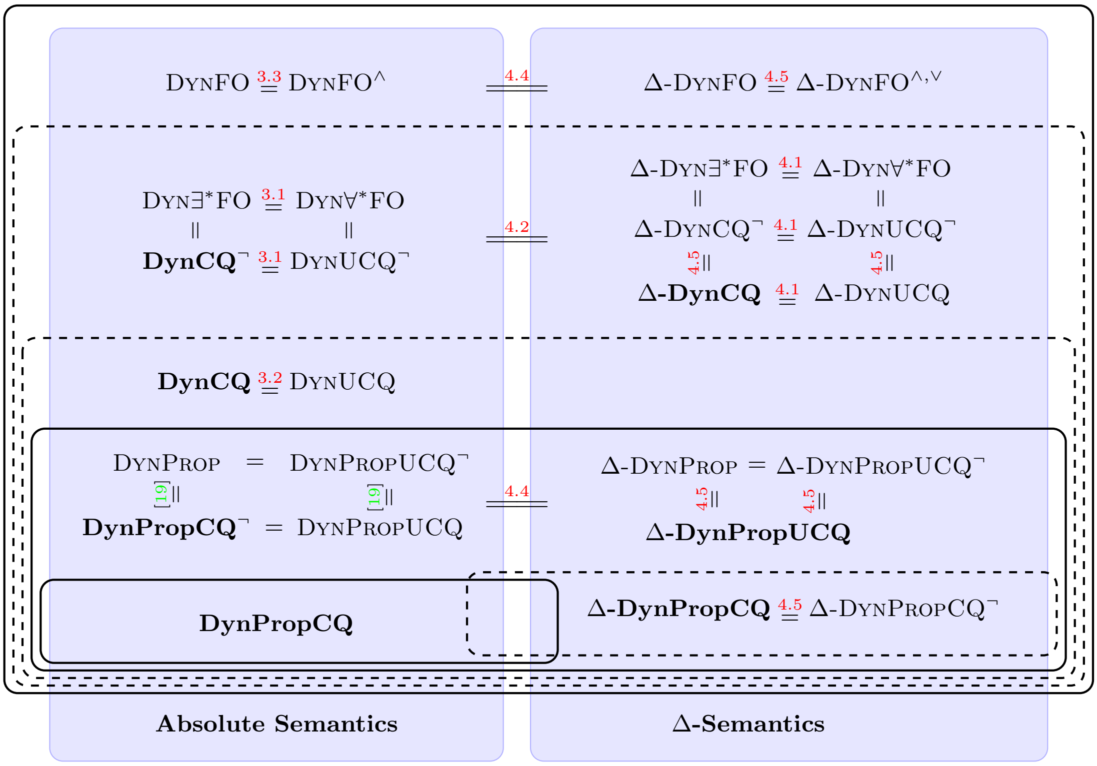

Figure 1: Hierarchy of fragments of DynFO. Solid lines are strict separations.

DynCQ are as hard as lower bounds for DynUCQ. However, though all our results are stated for arbitrary initialization mappings, they also hold in the setting with empty initial database and first-order initialization for the auxiliary data. The proofs do not carry over to the strict setting of Patnaik and Immerman where, in a dynamic class Dyn$\mathcal{C}$, only $\mathcal{C}$ initializations are allowed.

Related work. We next discuss some further related work, beyond what we already mentioned above. The expressivity of first-order logic in the dynamic complexity frameworks discussed above has been studied a lot (see e.g. [16, 6, 8, 13, 14, 18, 10, 9]). Most results focus on showing that a problem from some static complexity class can be dynamically maintained by programs of a weaker query class. Some lower bounds have been achieved as well (see e.g. [3, 4, 6, 9, 10, 19]). Many other aspects such as the arity of auxiliary relations (see e.g. [6, 14]), whether the auxiliary relations are determined by the current structure (see e.g. [16, 5, 10]), and the presence of an order (see e.g. [10]) have been studied.

An algebraic perspective of incremental view maintenance under Δ-semantics has been studied in [15]. Parts of this work have also been implemented, see, e.g., [1].

Outline. In Section 2 we define our dynamic setting more precisely. In Section 3 the results for absolute semantics are presented. The alternative Δ-semantics is introduced and studied in Section 4. In Section 5 we give the dynamic characterization of first-order logic. We conclude with a discussion and a first step towards separations in Section 6.

Acknowledgement. We thank Nils Vortmeier for careful proofreading. Further we are grateful to the anonymous reviewers for several very helpful comments. We acknowledge the financial support by the German DFG under grant SCHW 678/6-1.

2. DYNAMIC SETTING

In this section, we introduce the basic concepts and fix our notation. We mainly borrow it from our previous work [19].

A *dynamic instance* of a query $\mathcal{Q}$ is a pair $(\mathcal{D}, \alpha)$, where $\mathcal{D}$ is a database over a finite domain D and α is a sequence of updates to $\mathcal{D}$, i.e. a sequence of insertions and deletions of tuples over D. The dynamic query Dyn($\mathcal{Q}$) yields as result the relation that is obtained by first applying the updates from α to $\mathcal{D}$ and then evaluating the query $\mathcal{Q}$ on the resulting database.

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

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

A *dynamic schema* is a tuple[3] $(\tau_{\text{in}}, \tau_{\text{aux}})$ where τ_{in} and τ_{aux} are the schemas of the input database and the auxiliary

[3] In [19] a dynamic schema had an additional schema for an extra database with built-in relations. As in this paper, we do not restrict auxiliary relations in any way and allow arbitrary initialization, we do not need built-in relations.

database, respectively. In this work all schemata are purely relational, although all results also hold for input schemas with constants. We always let $\tau \stackrel{\text{def}}{=} \tau_{\text{in}} \cup \tau_{\text{aux}}$.

Definition 1. (Update program) An *update program* $\mathcal{P}$ over 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 $R \in \tau_{\text{aux}}$ and every $\delta \in \{\text{INS}_S, \text{DEL}_S\}$ with $S \in \tau_{\text{in}}$, an update formula $\phi^R_\delta(\vec{x};\vec{y})$ 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. For an update $\delta(\vec{a})$, where $\vec{a}$ is a tuple over D, and program state $\mathcal{S} = (D, \mathcal{I}, \mathcal{A})$ we denote by $P_\delta(\mathcal{S})$ the state $(D, \delta(\mathcal{I}), \mathcal{A}')$, where $\mathcal{A}'$ consists of relations $R' \stackrel{\text{def}}{=} \{\vec{b} \mid \mathcal{S} \models \phi^R_\delta(\vec{a};\vec{b})\}$. The effect $P_\alpha(\mathcal{S})$ of an update sequence $\alpha = \delta_1 \ldots \delta_m$ to a state $\mathcal{S}$ is the state $P_{\delta_m}(\ldots(\mathcal{P}_{\delta_1}(\mathcal{S}))\ldots)$.

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, i.e. $\mathcal{S}_{\text{INIT}}(\mathcal{D}) \stackrel{\text{def}}{=} (D, \mathcal{D}, \text{INIT}_{\text{aux}}(\mathcal{D}))$.

Several dynamic settings and restrictions of dynamic programs have been studied in the literature (see e.g. [16, 8, 10, 9]). 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; and
- whether the initialization mapping INIT_{aux} is *permutation-invariant* (short: *invariant*), that is, whether $\pi(\text{INIT}_{\text{aux}}(\mathcal{D})) = \text{INIT}_{\text{aux}}(\pi(\mathcal{D}))$ holds, for every database $\mathcal{D}$ and permutation π of the domain.

We refer to the introduction for a discussion of the choices made in the following definition.

Definition 3. (DYN$\mathcal{C}$) For a class $\mathcal{C}$ of formulas, let DYN$\mathcal{C}$ be the class of all dynamic queries that can be maintained by dynamic programs with formulas from $\mathcal{C}$ and arbitrary initialization mapping.

In particular DYNFO is the class of all dynamic queries that can be maintained by first-order update formulas. DYNPROP is the subclass of DYNFO, where update formulas are not allowed to use quantifiers.

We note that arbitrary, (possibly) non-uniform initialization mappings permit to maintain undecidable queries, even when the logic for expressing update formulas is very weak.

Allowing arbitrary initialization mappings in Definition 5 helps us to concentrate on the *maintenance* aspect of dynamic complexity and it helps keeping proofs short. All our results also hold for FO-definable initialization mappings on ordered domains.

3. FRAGMENTS OF DYNFO

In this section we study the relationship between the variants of dynamic conjunctive queries. This continues the study of fragments of first-order logic that has been started in [19].

We first give formal definitions of the classes of queries we are interested in:

- CQ: the class of conjunctive queries, that is, queries expressible by first-order formulas of the form $\varphi(\vec{x}) = \exists\vec{y}\psi$, where ψ is a conjunction of atomic formulas.
- UCQ: the class of all unions of conjunctive queries, that is, queries expressible by formulas of the form $\bigvee_i \exists\vec{x}\psi$, where ψ is a conjunction of atomic formulas.

We note that safety of queries is not an issue in this paper: we use queries as update formulas only and we can always assume that, for each required arity, there is an auxiliary "universal" relation containing all tuples of this arity over the active domain which could be used to make queries syntactically safe.

The classes CQ and UCQ can be extended by additionally allowing negated atoms, resulting in $\text{CQ}^\neg$ and $\text{UCQ}^\neg$, or they can be restricted by disallowing quantification. It is well known that $\text{UCQ}^\neg$ and $\exists^*\text{FO}$, the class of queries expressible by existential first-order formulas, coincide, but otherwise, all these classes are distinct. Furthermore, other quantification patterns than $\exists^*$ can be considered, like $\forall^*$ or arbitrary quantification.

The main goal of this section is to show that the relationship of these classes in the dynamic setting is much simpler than in the static setting. We prove that dynamic classes collapse as indicated in the left part of Figure 1. More precisely, we show the following two theorems regarding the second and the third fragment in the left part of Figure 1.

THEOREM 3.1. *Let $\mathcal{Q}$ be a query. Then the following statements are equivalent:*

(a) $\mathcal{Q}$ can be maintained in $\text{DYNUCQ}^\neg$.

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

(c) $\mathcal{Q}$ can be maintained in $\text{DYN}\exists^*\text{FO}$.

(d) $\mathcal{Q}$ can be maintained in $\text{DYN}\forall^*\text{FO}$.

THEOREM 3.2. *Let $\mathcal{Q}$ be a query. Then the following statements are equivalent:*

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

(b) $\mathcal{Q}$ can be maintained in DYNCQ.

Using the same technique as is used for removing unions from dynamic unions of conjunctive queries, a normal form for DYNFO can be obtained. The class $\text{DYNFO}^\wedge$ contains all queries maintainable by a program whose update formulas are in prenex normal form where the quantifier-free part is a conjunction of atoms. The following theorem improves Theorem 15 from [19].

THEOREM 3.3. *Let $\mathcal{Q}$ be a query. Then the following statements are equivalent:*

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

(b) $\mathcal{Q}$ can be maintained in DynFO$^{\wedge}$.

Before we turn to the proofs of these theorems, we discuss the proof techniques that will be used.

For showing that a class Dyn$\mathcal{C}$ of queries is contained in a class Dyn$\mathcal{C}'$, it is sufficient to construct, for every dynamic program with update queries from class $\mathcal{C}$, an equivalent dynamic program with update queries from class $\mathcal{C}'$. In cases where $\mathcal{C}' \subset \mathcal{C}$ this can also be seen as constructing a $\mathcal{C}'$-normal form for $\mathcal{C}$-programs.

Most of the proofs for the collapse of two dynamic classes in this paper are not very deep. Indeed, most of them use one or more of the following (easy) techniques.

The *replacement technique* ([19]) is used to remove subformulas of a certain kind from update formulas and to replace their "meaning" by additional auxiliary relations. In this way, we often can remove negations (choose negative literals as subformulas, see the proof of Theorem 3.5) and disjunctions (see proof of Lemma 3.7) from update formulas.

The *preprocessing technique* is used to convert (more) complicated update formulas into easier update formulas by splitting the computation performed by the complicated update formula into two parts; one of them performed by the initialization mapping and stored in an additional auxiliary relation, the other one performed by the easier update formula using the pre-computed auxiliary relation. Applications of this technique are the removal of unions from dynamic unions of conjunctive queries (see example below) as well as proving the equivalence of semantics for dynamic conjunctive queries with negations (see Lemma 4.6).

Example 1. We consider the update formula

$$\phi^R_\delta(u;x) = \exists y\big(U(x,y) \vee V(x,u)\big)$$

for a unary relational symbol R. We aim at an equivalent update formula $\psi^R_\delta(u;x)$ without disjunction. The idea is to store a 'disjunction blue print' in a precomputed auxiliary relation T and to use existential quantification to guess which disjunct becomes true.

In this example, we assume that in every state of the dynamic program on every database, both the interpretations of U and V are always non-empty sets.[4] Then, $\phi^R_\delta(u;x)$ can be replaced by

$$\exists y \exists z_1 \exists z_2 \exists z_3 \exists z_4 \big(U(z_1,z_2) \wedge V(z_3,z_4) \\ \wedge T(z_1,z_2,z_3,z_4,x,y,u)\big)$$

where T is an additional auxiliary relation symbol which is interpreted, in every state $\mathcal{S}$, by a 7-ary relation $T^{\mathcal{S}}$ containing all tuples $(a_1,\dots,a_7)$ with $(a_1,a_2) = (a_5,a_6)$ or $(a_3,a_4) = (a_5,a_7)$. Thus $T^{\mathcal{S}}$ ensures that either the values chosen for z_1,z_2 coincide with the values of x,y or the values of z_3,z_4 coincide with x,u.

Therefore, the initialization mapping initializes T with the result of the query

$$\mathcal{Q}_T(z_1,z_2,z_3,z_4,x,y,u) \stackrel{\text{def}}{=} \\ \big((z_1,z_2) = (x,y) \vee (z_3,z_4) = (x,u)\big)$$

Observe that this approach fails when U or V are interpreted by empty relations. In order to cover empty relations as well, some extra work needs to be done (see proof of Lemma 3.7).

[4]This assumption will eventually be removed in the proof.

The *squirrel technique* maintains additional auxiliary relations that reflect the state of some auxiliary relation after every possible single update (or short update sequence).[5] For example, if a dynamic program contains a relation symbol R then a fresh relation symbol R_{INS} can be used, such that the interpretation of R_{INS} contains the content of R after update INS (for every possible insertion tuple). Of course, R_{INS} has higher arity than R, as it takes the actual inserted tuple into account. Sample applications of this technique are the removal of quantifiers from *some* update formulas (see the following example and Lemma 3.4) and the maintenance of first-order queries in DynCQ$^{\neg}$ (see Theorem 5.1).

Example 2. Consider the update formula

$$\phi^Q_{\text{INS}}(u_1;x) = \exists y\big(Q(x) \vee \neg S(u_1,y)\big)$$

for the query symbol Q of some dynamic program $\mathcal{P}$. In order to obtain a quantifier-free update formula for Q after insertion of an arbitrary tuple we maintain the relation $Q_{\text{INS}}(\cdot,\cdot)$ that contains a tuple (a,b) if and only if b would be in Q in the next state, after insertion of a. Similarly for S and deletions.

Then the update formula ϕ^Q_{INS} can be replaced by the quantifier-free formula $\phi^Q_{\text{INS}}(u_1;x) \stackrel{\text{def}}{=} Q_{\text{INS}}(u_1,x)$. The relation Q_{INS} can be updated via

$$\phi^{Q_{\text{INS}}}_{\text{INS}}(u_0;u_1,x) \stackrel{\text{def}}{=} \exists y\big(Q_{\text{INS}}(u_0,x) \vee \neg S_{\text{INS}}(u_0,u_1,y)\big)$$
$$\phi^{Q_{\text{INS}}}_{\text{DEL}}(u_0;u_1,x) \stackrel{\text{def}}{=} \exists y\big(Q_{\text{DEL}}(u_0,x) \vee \neg S_{\text{DEL}}(u_0,u_1,y)\big)$$

and similarly, for the other new auxiliary relations.

We note that, in this example, the application of the technique does not eliminate all quantifiers in the program (in fact, it removes one and introduces two new formulas with quantifiers), but it removes quantification from the update formula for a *particular relation*. Removing quantification from the update formulas of the query relation will turn out to be useful in the proofs of Lemmata 3.7, 3.9 and 4.6.

The proof of the following technical lemma is based on this example. For an arbitrary quantifier prefix $\mathbb{Q} \in \{\exists,\forall\}^*$ let $\mathbb{Q}$FO be the class of queries expressible by formulas with quantifier prefix $\mathbb{Q}$. If $\mathbb{Q}$ is a substring of $\mathbb{Q}'$ and a query $\mathcal{Q}$ is in $\mathbb{Q}$FO then trivially $\mathcal{Q}$ is in $\mathbb{Q}'$FO as well.

Lemma 3.4. *Let $\mathbb{Q}$ be an arbitrary quantifier prefix. For every* Dyn$\mathbb{Q}$FO*-program there is an equivalent* Dyn$\mathbb{Q}$FO*-program $\mathcal{P}$ such that the update formulas for the designated query symbol of $\mathcal{P}$ consist of a single atom.*

Before we turn to dynamic conjunctive queries, we recall some results from [19]. We denote by PropCQ, PropUCQ, PropCQ$^{\neg}$ and PropUCQ$^{\neg}$ the classes of queries definable by conjunctive queries (and their extensions) but *without quantification*.

The first two results from [19] that we recall, are negation-free normal forms for DynFO and for DynProp, the class of queries maintainable by quantifier-free first-order formulas.

Theorem 3.5 ([19]). *(a) Every* DynFO*-program has an equivalent negation-free* DynFO*-program.*

(b) Every DynProp*-program has an equivalent* DynPropUCQ*-program.*

[5]Squirrels usually make provisions for every possible future.

Proof idea. This theorem is a generalization of Theorem 6.6 from [14]. Given a dynamic program $\mathcal{P}$, the simple idea is to maintain, for every auxiliary relation R of $\mathcal{P}$, an additional auxiliary relation $\widehat{R}$ for the complement of R. $\square$

Furthermore, there is a disjunction-free normal form for DynProp.

Theorem 3.6 ([19]). *Every* DynProp*-program has an equivalent* DynPropCQ$^\neg$*-program.*

Proof idea. The update formulas of a given DynProp-program $\mathcal{P}$ can be assumed to be in conjunctive normal form. An equivalent DynPropCQ$^\neg$-program is obtained by maintaining an additional relation $R_{\neg C}$, for every (disjunctive) clause C occurring in any update formula of $\mathcal{P}$.

With these additional relations, every update formula

$$\phi = C_1(\vec{x}_1) \wedge \ldots \wedge C_k(\vec{x}_k)$$

with clauses $C_1(\vec{x}_1), \ldots, C_k(\vec{x}_k)$ can be replaced by the conjunctive formula

$$\neg R_{\neg C_1}(\vec{x}_1) \wedge \ldots \wedge \neg R_{\neg C_k}(\vec{x}_k)$$

The new auxiliary relations $R_{\neg C}$ themselves can be maintained by viewing $\neg C$ as a conjunction of atoms and taking the conjunction of all the conjunctive update formulas for the literals[6] in $\neg C$. $\square$

The two normal forms for DynProp can be seen as collapse results for quantifier-free conjunctive queries. Together they state that the dynamic classes DynPropUCQ, DynPropCQ$^\neg$ and DynPropUCQ$^\neg$ coincide with DynProp.

Now we present new collapse results for dynamic conjunctive queries. First, we prove that in the dynamic setting disjunctions can be simulated by existential quantifiers, that is DynUCQ and DynCQ as well as DynUCQ$^\neg$ and DynCQ$^\neg$ coincide. However, we can not apply the idea of the proof of Theorem 3.6 directly for this result, since UCQ and UCQ$^\neg$ are not closed under negations.

Lemma 3.7. *(a) For every* DynUCQ$^\neg$*-program there is an equivalent* DynCQ$^\neg$*-program.*

(b) For every DynUCQ*-program there is an equivalent* DynCQ*-program.*

(c) For every DynFO*-program there is an equivalent* DynFO$^\wedge$*-program.*

Proof. We first prove the statements for domains with at least two elements and show how to drop this restriction afterwards. The construction uses the idea from Example 1. We give it for (a) but, as it does not introduce any negation operators it works for (b) as well. For (c) one starts from a negation-free DynFO-program and uses the same construction used for (a)[7].

[6] For this step it is needed that $\mathcal{P}$ contains, for every auxiliary relation R, an additional auxiliary relation $\widehat{R}$ for the complement of R. This can be ensured by the technique from Theorem 3.5.

[7] More precisely, replace the quantifier prefix $\exists\vec{y}$ used throughout the construction of (a) by a general quantifier-prefix $\exists\vec{y}_1\forall\vec{y}_2\ldots$.

Let $\mathcal{P} = (P, \text{Init}, Q)$ be a DynUCQ$^\neg$-program over schema τ. Without loss of generality, we assume that the quantifier-free parts of all update formulas of $\mathcal{P}$ are in disjunctive normal form. We convert $\mathcal{P}$ into an equivalent DynCQ$^\neg$-program $\mathcal{P}'$ whose update formulas are in prenex normalform with quantifier-free parts of the form $\bigwedge_i L_i(\vec{x}_i) \wedge T(\vec{y})$, where L_i are arbitrary literals over the modified schema $\widehat{\tau}$ and the symbols T are fresh auxiliary relation symbols.

The program $\mathcal{P}' = (P', \text{Init}', Q)$ is over schema $\tau' = \tau \cup \widehat{\tau} \cup \tau_T$, where $\widehat{\tau}$ contains a $(k+1)$-ary relation symbol $\widehat{R}$ for every k-ary relation symbol $R \in \tau$; and τ_T contains a relation symbol $T_{S,\delta}$ for every relation symbol $S \in \tau \cup \widehat{\tau}$ and every update operation δ.

The intention for relation symbols from τ' is as follows. The relation symbols $R \in \tau$ shall always be interpreted as in $\mathcal{P}$. The intention of $\widehat{R} \in \widehat{\tau}$ is, on one hand, to contain a "copy of R" (in tuples with first component c, for some fixed element c) and on the other hand, to have a guarantee that $\widehat{R}$ is non-empty. The latter is strongly ensured by enforcing all tuples that do *not* have c as first component to be in $\widehat{R}$, and by $|D| \geq 2$:

$$\widehat{R}^{\mathcal{S}} \stackrel{\text{def}}{=} \{(c, \vec{a}) \mid \vec{a} \in R^{\mathcal{S}}\} \cup \{(d, \vec{a}) \mid d \neq c \text{ and } \vec{a} \in D^k\} \quad (1)$$

The relations $T_{S,\delta}$ will be used as in Example 1.

Now we construct the update formulas for program $\mathcal{P}'$. Let $R \in \tau$ and δ be an update. Further let

$$\phi_\delta^R(\vec{u}; \vec{x}) = \exists\vec{y}(C_1(\vec{u}, \vec{x}, \vec{y}) \vee \ldots \vee C_k(\vec{u}, \vec{x}, \vec{y}))$$

be the update formula of R with respect to δ in $\mathcal{P}$, where every C_i is a conjunction of literals. For

$$C_i(\vec{u}, \vec{x}, \vec{y}) = L_1(\vec{v}_1) \wedge \ldots \wedge L_m(\vec{v}_l)$$

we define

$$\widehat{C}_i(v, \vec{u}, \vec{x}, \vec{y}) \stackrel{\text{def}}{=} \widehat{L}_1(v, \vec{v}_1) \wedge \ldots \wedge \widehat{L}_m(v, \vec{v}_l)$$

where $\widehat{L}_j = \widehat{R}$ if $L_j = R$ and $\widehat{L}_j = \neg\widehat{R}$ if $L_j = \neg R$.

The update formula $\psi_\delta^{\widehat{R}}(\vec{u}; x', \vec{x})$ for $\widehat{R} \in \widehat{\tau}$ in $\mathcal{P}'$ is

$$\begin{aligned}\psi_\delta^{\widehat{R}}(\vec{u}; x', \vec{x}) \stackrel{\text{def}}{=} & \exists\vec{y}\, \exists z_1' \exists\vec{z}_1 \ldots \exists z_k' \exists\vec{z}_k \\ & \Big(\widehat{C}_1(z_1', \vec{z}_1) \wedge \ldots \wedge \widehat{C}_k(z_k', \vec{z}_k) \\ & \quad \wedge T_{\widehat{R},\delta}(\vec{y}, z_1', \vec{z}_1, \ldots, z_k', \vec{z}_k, \vec{u}, x', \vec{x})\Big)\end{aligned}$$

The update formula $\psi_\delta^R(\vec{u}; x', \vec{x})$ for $R \in \tau$ in $\mathcal{P}'$ is

$$\begin{aligned}\psi_\delta^R(\vec{u}; \vec{x}) \stackrel{\text{def}}{=} & \exists\vec{y}\, \exists z_1' \exists\vec{z}_1 \ldots \exists z_k' \exists\vec{z}_k \\ & \Big(\widehat{C}_1(z_1', \vec{z}_1) \wedge \ldots \wedge \widehat{C}_k(z_k', \vec{z}_k) \\ & \quad \wedge T_{R,\delta}(\vec{y}, z_1', \vec{z}_1, \ldots, z_k' \vec{z}_k, \vec{u}, \vec{x})\Big)\end{aligned}$$

To ensure equivalence of this program with the original program, the relations $T_{S,\delta}$ are defined as follows.

- $T_{R,\delta}$ contains all tuples[8] $(\vec{y}, z_1', \vec{z}_1, \ldots, z_k' \vec{z}_k, \vec{u}, \vec{x})$, for which, for some i, $z_i' = c$ and $\vec{z}_i = (\vec{u}, \vec{x}, \vec{y})$.
- $T_{\widehat{R},\delta}$ contains all tuples $(\vec{y}, z_1', \vec{z}_1, \ldots, z_k', \vec{z}_k, \vec{u}, x', \vec{x})$), for which
  - $x' \neq c$, or

[8] For simplicity, we reuse variable names as element names.

– for some i, $z'_i = c$ and $\vec{z}_i = (\vec{u}, \vec{x}, \vec{y})$.

These are initialized as intended by simple quantifier-free formulas (but with disjunction). Their interpretation is never changed, that is, for every $T_{S,\delta}$, both update formulas reproduce the current value of $T_{S,\delta}$.

The initialization for relation symbols from τ and $\widehat{\tau}$ is straightforward. Auxiliary relation symbols $R \in \tau$ are initialized as in $\mathcal{P}$; and auxiliary relation symbols $\widehat{R} \in \widehat{\tau}$ are initialized by $\textsc{Init}'$ analogously to $\textsc{Init}$ but respecting Equation (1).

This concludes the proof of (a), (b) and (c) for domains with at least two elements. The restriction on the size of the domains can be dropped as follows. In all three cases the idea is to make a case distinction on the size of the domain in the update formulas of the designated query symbol.

To this end, we first construct a $\textsc{DynPropCQ}$-program $\mathcal{P}'' = (P'', \textsc{Init}'', Q'')$ over schema τ'' with $\tau' \cap \tau'' = \emptyset$ which is equivalent to $\mathcal{P}$ over databases with domains of size one. Then we construct a program $\mathcal{P}'''$ equivalent to $\mathcal{P}$ by combining $\mathcal{P}''$ and the program $\mathcal{P}'$ for domains of size at least two.

For the construction of $\mathcal{P}''$ we observe that every relation of a database over a single element domain $D = \{a\}$ contains either exactly one tuple, namely $(a, \dots, a)$, or no tuple at all. Thus every such relation R corresponds to a 0-ary relation R_0 where R_0 is true if and only if $(a, \dots, a) \in R$. Hence, by Lemma 3.8, there is a $\textsc{DynPropCQ}$-program equivalent to $\mathcal{P}$ for databases with domains of size one.

To combine $\mathcal{P}'$ and $\mathcal{P}''$ we use two different approaches, one for (a) and one for (b) and (c).

First we consider (a). To this end, we can assume, by Lemma 3.4, that the update formulas for the query relations Q' and Q'' of $\mathcal{P}'$ and $\mathcal{P}''$ consist of single atoms, respectively. We construct an intermediate program $\tilde{\mathcal{P}} = (\tilde{P}, \widetilde{\textsc{Init}}, \tilde{Q})$ over schema $\tilde{\tau} = \{\tilde{Q}, U\} \cup \tau' \cup \tau''$ where U is a fresh 0-ary relation symbol. The intention is that interpretations of symbols in τ' and τ'' are as in $\mathcal{P}'$ and $\mathcal{P}''$, respectively, and that U is interpreted by true if and only if the domain is of size one. The initializations are accordingly.

Thus all update formulas of $\tilde{\mathcal{P}}$ for relation symbols from τ' and τ'' are as in $\mathcal{P}'$ and $\mathcal{P}''$ (and thus disjunction-free). The update formula for U is $\phi^U_\delta \stackrel{\text{def}}{=} U$ and

$$\begin{aligned}\phi^{\tilde{Q}}_\delta &\stackrel{\text{def}}{=} (\phi^{Q'}_\delta \wedge \neg U) \vee (\phi^{Q''}_\delta \wedge U)\\ &\equiv (\phi^{Q'}_\delta \vee \phi^{Q''}_\delta) \wedge (\neg U \vee \phi^{Q''}_\delta) \wedge (\phi^{Q'}_\delta \vee U)\end{aligned}$$

The program $\mathcal{P}'''$ is obtained from $\tilde{\mathcal{P}}$ by removing disjunctions from $\phi^{\tilde{Q}}_\delta$ using the method[9] from the proof of Theorem 3.6. For example, the first clause is replaced by $\neg R_{\neg(Q' \vee Q'')}$ where $R_{\neg(Q' \vee Q'')}$ is a fresh auxiliary relation symbol intended to be always interpreted by the result of the query $\neg(\phi^{Q'}_\delta \vee \phi^{Q''}_\delta)$. The update formula for $R_{\neg(Q' \vee Q'')}$ after an update δ is $\neg\phi^{Q'}_\delta \wedge \neg\phi^{Q''}_\delta$; it is disjunction-free since, by our assumption, $\phi^{Q'}_\delta$ and $\phi^{Q''}_\delta$ both consist of a single atom. This concludes the proof of (a).

The program $\mathcal{P}'''$ for (b) and (c) is over schema $\tau''' = \{Q'''\} \cup \tau' \cup \tau''$. Again all update formulas of $\mathcal{P}'''$ for relation symbols from τ' and τ'' are as in $\mathcal{P}'$ and $\mathcal{P}''$ and

$$\phi^{Q'''}_\delta = \phi^{Q'}_\delta \wedge \phi^{Q''}_\delta$$

The case distinction is delegated to the initialization mapping. Recall that the size of the domain is fixed when the auxiliary relations are initialized. The initialization mapping $\textsc{Init}'''$ is as follows. If $|D| = 1$ then

$$\textsc{Init}'''(R) = \begin{cases} \textsc{Init}''(Q'') & \text{for } R = Q''', \\ D^k & \text{for } R \in \tau', \\ \textsc{Init}''(R'') & \text{for } R \in \tau'' \end{cases}$$

If $|D| \geq 2$ then

$$\textsc{Init}'''(R) = \begin{cases} \textsc{Init}'(Q') & \text{for } R = Q''', \\ \textsc{Init}'(R') & \text{for } R \in \tau', \\ D^k & \text{for } R \in \tau'' \end{cases}$$

Thus $\textsc{Init}'''$ selects either $\phi^{Q'}_\delta$ or $\phi^{Q''}_\delta$, depending on the size of the domain. If $|D| = 1$ then $\phi^{Q'}_\delta$ always evaluates to true whereas $\phi^{Q''}_\delta$ yields the same value as in $\mathcal{P}''$, and vice versa for $|D| \geq 2$. As update formulas do not use negation, all relations in the program, that is initialized to "true" ($\mathcal{P}'$ or $\mathcal{P}''$) remain "full" throughout.[10] This concludes the proof of (b). □

0-ary relations can either be true (containing the empty tuple) or false (not containing the empty tuple and thus being empty), thus 0-ary atoms are basically propositional variables. Queries on 0-ary databases are therefore basically families of Boolean functions, one for each domain size. Such queries are not very interesting from the perspective of databases, but we need to show the following lemma as we used it in the previous proof. As quantification in queries on 0-ary databases is useless, every FO query can be expressed by a quantifier-free formula and therefore can be maintained in $\textsc{DynProp}$. Yet, even more general, all queries on a 0-ary database can be maintained by even more restricted dynamic programs. Let $\textsc{DynPropCQ}$ be the class of dynamic queries definable by update programs whose update formulas are solely conjunctions of atoms (i.e. no negations, no disjunctions and no quantifiers are allowed).

Lemma 3.8. *Every query on a 0-ary database can be maintained by a* $\textsc{DynPropCQ}$*-program.*

Proof. Let τ_{in} be an input schema with 0-ary relation symbols $A_1, \dots, A_k$. Further let $\mathcal{Q}_1, \dots, \mathcal{Q}_m$ be an enumeration of all $m = 2^{2^k}$ many queries on τ_{in}. We actually show that all of them can be maintained by one $\textsc{DynPropCQ}$-program $\mathcal{P}$ with auxiliary schema $\tau_{\text{aux}} = \{R_1, \dots, R_m\}$ maintaining $\mathcal{Q}_i$ in R_i, for every $i \in \{1, \dots, m\}$.

To this end, let $\varphi_1, \dots, \varphi_m$ be propositional formulas over τ_{in} such that φ_i expresses $\mathcal{Q}_i$ and each φ_i is in conjunctive normal form. Without loss of generality, no clause contains A_l and $\neg A_l$ for any $A_l \in \tau_{\text{in}}$ and any φ_i. As τ_{aux} contains a relation symbol, for every propositional formula over $A_1, \dots, A_k$, it contains, in particular, an auxiliary relation symbol R_C, for every disjunctive clause over $A_1, \dots, A_k$.

The update formulas for R_j after changing input relation A_l can be constructed as follows. Let $\mathcal{C}$ be the set of clauses of φ_j, i.e. $\varphi_j = \bigwedge_{C \in \mathcal{C}} C$. We denote by $\mathcal{C}^+_{A_l}$, $\mathcal{C}^-_{A_l}$ and $\mathcal{C}_{A_l}$ the subsets of $\mathcal{C}$ whose clauses contain A_l, $\neg A_l$ and neither A_l nor $\neg A_l$, respectively.

[9]This method cannot be used for $\textsc{DynCQ}$ and $\textsc{DynFO}^\wedge$.

[10]This cannot be guaranteed for $\textsc{DynUCQ}^\neg$.

If A_l becomes true by an update then φ_j evaluates to true if all clauses in $\mathcal{C}_{A_l}$ and all clauses $C \setminus \{\neg A_l\}$ with $C \in \mathcal{C}^-_{A_l}$ evaluated to true before the update (clauses in $\mathcal{C}^+_{A_l}$ will evaluate to true after enabling A_l).

If A_l becomes false by an update then φ_j evaluates to true if all clauses in $\mathcal{C}_{A_l}$ and all clauses $C \setminus \{A_l\}$ with $C \in \mathcal{C}^+_{A_l}$ evaluated to true before the update (clauses in $\mathcal{C}^-_{A_l}$ will evaluate to true after disabling A_l).

Therefore the update formulas for R_j after updating A_l can be defined as follows:

$$\phi^{R_j}_{\text{INS}_{A_l}} \stackrel{\text{def}}{=} \bigwedge_{C \in \mathcal{C}_{A_l}} R_C \wedge \bigwedge_{C \in \mathcal{C}^-_{A_l}} R_{C\setminus\{\neg A_l\}}$$

$$\phi^{R_j}_{\text{DEL}_{A_l}} \stackrel{\text{def}}{=} \bigwedge_{C \in \mathcal{C}_{A_l}} R_C \wedge \bigwedge_{C \in \mathcal{C}^+_{A_l}} R_{C\setminus\{A_l\}}$$

The initialization is straightforward. The correctness of this construction can be proved by induction over the length of update sequences. □

Finally we prove that DYN$\exists^*$FO = DYN$\forall^*$FO, and therefore that unions of conjunctive queries with negation coincide with DYN$\forall^*$FO in the dynamic setting. The proof uses the replacement technique to maintain the complements of the auxiliary relations used in the DYN$\exists^*$FO-program via DYN$\forall^*$FO-formulas. A small complication arises from the fact, that the query relation (and not its complement) has to be maintained. This is solved by ensuring that the update formulas of the query relation are atomic.

A slightly more general result can be shown.

LEMMA 3.9. *Let $\mathbb{Q}$ be an arbitrary quantifier prefix. A query can be maintained in* DYN$\mathbb{Q}$FO *if and only if it can be maintained in* DYN$\overline{\mathbb{Q}}$FO.

Now Theorems 3.1 and 3.2 follow immediately from Lemma 3.7 and Lemma 3.9.

4. Δ-SEMANTICS

So far we considered a semantics where the new version of the auxiliary relations is redefined, after each update, from scratch by formulas that are evaluated on the structure with the current auxiliary relations. We refer to this as *absolute semantics* in the following.

However, in the context of view maintenance, one usually expects only few auxiliary tuples to change after an update. Therefore it is common to express the new version of the auxiliary relations in terms of the current relations and some "Delta", that is, a (small) relation R^+ of tuples to be inserted into R and a (small) relation R^- of tuples to be removed from R (with $R^+ \cap R^- = \emptyset$). The updated auxiliary relation R' is then defined by

$$R' \stackrel{\text{def}}{=} (R \cup R^+) \setminus R^-$$

We refer to this semantics as Δ-*semantics*. This is the semantics usually considered in view maintenance. As already stated in the introduction, absolute and Δ-semantics can only be different if the underlying update language is not closed under Boolean operations.

Next we formalize Δ-semantics via Δ-update programs which provide formulas defining the relations R^+ and R^-, for every auxiliary relation R.

Definition 4. (Δ-Update program) A Δ-*update program* $\mathcal{P}$ over dynamic schema $(\tau_{\text{in}}, \tau_{\text{aux}})$ is a set of first-order formulas (called Δ-*update formulas* in the following) that contains, for every $R \in \tau_{\text{aux}}$ and every $\delta \in \{\text{INS}_S, \text{DEL}_S\}$ with $S \in \tau_{\text{in}}$, two formulas $\phi^{R^+}_\delta(\vec{u};\vec{x})$ and $\phi^{R^-}_\delta(\vec{u};\vec{x})$ over the schema τ where $\vec{u}$ and S have the same arity, $\vec{x}$ and R have the same arity, and $\phi^{R^+}_\delta \wedge \phi^{R^-}_\delta$ is unsatisfiable.

The *semantics of* Δ-*update programs* is as follows. For an update $\delta = \delta(\vec{a})$ and program state $\mathcal{S} = (D, \mathcal{I}, \mathcal{A})$ we denote by $P_\delta(\mathcal{S})$ the state $(D, \delta(\mathcal{I}), \mathcal{A}')$, where the relations R' of $\mathcal{A}'$ are defined by

$$R' \stackrel{\text{def}}{=} \left(R \cup \{\vec{b} \mid \mathcal{S} \models \phi^{R^+}_\delta(\vec{a};\vec{b})\}\right) \setminus \{\vec{b} \mid \mathcal{S} \models \phi^{R^-}_\delta(\vec{a};\vec{b})\}.$$

The effect of an update sequence on a state, dynamic Δ-programs and so on are defined like their counterparts in absolute semantics except that Δ-update programs are used instead of update programs.

Definition 5. (Δ-DYN$\mathcal{C}$) For a class $\mathcal{C}$ of formulas, let Δ-DYN$\mathcal{C}$ be the class of all dynamic queries that can be maintained by dynamic Δ-programs with formulas from $\mathcal{C}$ and arbitrary initialization mapping.

We note that the definitions above do *not* require that $R^+ \cap R = \emptyset$ or $R^- \sqsubseteq R$, that is, R^+ might contain tuples that are already in R, and R^- might contain tuples that are not in R. However, in all proofs below, we construct only Δ-update formulas that guarantee these additional properties. As a consequence, for the considered fragments, the expressive power is independent of this difference.

The goal of this section is to prove the remaining results of Figure 1, that is, the collapse results depicted in the right part of the figure and the correspondences between absolute semantics and Δ-semantics.

THEOREM 4.1. *Let $\mathcal{Q}$ be a query. Then the following statements are equivalent:*

(a) $\mathcal{Q}$ can be maintained in Δ-DYNUCQ$^\neg$.

(b) $\mathcal{Q}$ can be maintained in Δ-DYNUCQ.

(c) $\mathcal{Q}$ can be maintained in Δ-DYNCQ$^\neg$.

(d) $\mathcal{Q}$ can be maintained in Δ-DYNCQ.

(e) $\mathcal{Q}$ can be maintained in Δ-DYN$\exists^*$FO.

(f) $\mathcal{Q}$ can be maintained in Δ-DYN$\forall^*$FO.

THEOREM 4.2. *Let $\mathcal{Q}$ be a query. Then the following statements are equivalent:*

(a) $\mathcal{Q}$ can be maintained in DYNUCQ$^\neg$.

(b) $\mathcal{Q}$ can be maintained in Δ-DYNUCQ$^\neg$.

The technique used for removing unions from dynamic unions of conjunctive queries under Δ-semantics can be used to obtain a Δ-DYNFO$^\wedge$ normal form for Δ-DYNFO-programs.

THEOREM 4.3. *Let $\mathcal{Q}$ be a query. Then the following statements are equivalent:*

(a) $\mathcal{Q}$ can be maintained in Δ-DYNFO.

(b) $\mathcal{Q}$ can be maintained in Δ-DynFO$^\wedge$.

We state some basic facts about dynamic programs with Δ-semantics before proving those theorems. The following lemma establishes the obvious fact that the absolute semantics and Δ-semantics coincide in expressive power for dynamic classes closed under boolean operations. We observe that the proof does not work for (extensions of) conjunctive queries. Later we will see how to extend the result to conjunctive queries.

Lemma 4.4. *Let $\mathcal{C}$ be some fragment of first-order logic closed under the boolean operations $\{\vee, \wedge, \neg\}$. Then for every query $\mathcal{Q}$ the following are equivalent:*

(a) There is a Dyn$\mathcal{C}$-program that maintains $\mathcal{Q}$.

(b) There is a Δ-Dyn$\mathcal{C}$-program that maintains $\mathcal{Q}$.

Proof. From an Dyn$\mathcal{C}$-update formula ϕ^R_δ, the Δ-Dyn$\mathcal{C}$-update formulas are defined as

$$\phi^{R^+}_\delta(\vec{u};\vec{x}) \stackrel{\text{def}}{=} \phi^R_\delta(\vec{u};\vec{x}) \wedge \neg R(\vec{x})$$
$$\phi^{R^-}_\delta(\vec{u};\vec{x}) \stackrel{\text{def}}{=} \neg\phi^R_\delta(\vec{u};\vec{x}) \wedge R(\vec{x})$$

From a Δ-Dyn$\mathcal{C}$-update formulas $\phi^{R^+}_\delta$ and $\phi^{R^+}_\delta$, an Dyn$\mathcal{C}$-update formula is obtained via

$$\phi^R_\delta(\vec{u};\vec{x}) \stackrel{\text{def}}{=} \big(R(\vec{x}) \vee \phi^{R^+}_\delta(\vec{u};\vec{x})\big) \wedge \neg\phi^{R^-}_\delta(\vec{u};\vec{x})$$

□

Removing negations in dynamic programs with Δ-semantics is straightforward using the replacement technique, since the complement $\widehat{R}$ of an auxiliary relation R can be maintained by exchanging the formulas $\phi^{R^+}_\delta$ and $\phi^{R^-}_\delta$. Observe that in contrast to absolute semantics this works for arbitrary query classes, even if they are not closed under complementation, and in particular for (extensions of) conjunctive queries.

Lemma 4.5. *Let $\mathcal{C}$ be some fragment of first-order logic. If a query $\mathcal{Q}$ can be maintained in Δ-Dyn$\mathcal{C}$ then $\mathcal{Q}$ can be maintained in negation-free Δ-Dyn$\mathcal{C}$.*

Proof. The idea is again to maintain the complements for auxiliary relations. For the sake of completeness we give a full proof.

Given a dynamic Δ-program $\mathcal{P}$ over schema τ we construct a dynamic Δ-program $\mathcal{P}'$ over schema $\tau \cup \widehat{\tau}$ where $\widehat{\tau}$ contains, for every k-ary relation symbol $R \in \tau$, a fresh k-ary relation symbol $\widehat{R}$ with the intention that $\widehat{R}$ always stores the complement of R.

The update formulas for $R \in \tau$ are as in $\mathcal{P}$. For a relation symbol $R \in \tau$ let $\phi^{R^+}_\delta(\vec{u};\vec{x})$ and $\phi^{R^-}_\delta(\vec{u};\vec{x})$ be the update formulas of R. Then the update formulas for $\widehat{R}$ can be defined as follows:

$$\phi^{\widehat{R}^+}_\delta(\vec{u};\vec{x}) = \phi^{R^-}_\delta(\vec{u};\vec{x})$$
$$\phi^{\widehat{R}^-}_\delta(\vec{u};\vec{x}) = \phi^{R^+}_\delta(\vec{u};\vec{x})$$

From $\mathcal{P}'$, a negation-free dynamic Δ-program $\mathcal{P}''$ can be constructed by replacing, for all $R \in \tau$, all occurrences of $\neg R(\vec{x})$ in update formulas of $\mathcal{P}'$ by $\widehat{R}(\vec{x})$. We omit the obvious proof of correctness. □

We now turn towards proving the main results of this section. We first prove Theorem 4.2. Afterwards we use the connection between absolute and Δ-semantics that it establishes as well as the adaption of Lemma 3.7 to Δ-semantics to prove the characterization of conjunctive queries with Δ-semantics.

The only-if-direction of Theorem 4.2 can be generalized to arbitrary quantifier prefixes. It is open whether the if-direction generalizes as well.

Lemma 4.6. *Let $\mathbb{Q}$ be an arbitrary quantifier prefix. If a query can be maintained in Dyn$\mathbb{Q}$FO then it can be maintained in Δ-Dyn$\mathbb{Q}$FO as well.*

Proof. Let $\mathcal{P} = (P, \textsc{Init}, Q)$ be a Dyn$\mathbb{Q}$FO-program with schema τ. By Lemma 3.4 we can assume, without loss of generality, that the update formulas of Q are atomic. We construct a dynamic Δ-Dyn$\mathbb{Q}$FO-program $\mathcal{P}' = (P', \textsc{Init}', Q')$.

The main challenge is to design update formulas of the kind $\phi^{R^-}_\delta$ without being able to complement the given update formulas because this would lead to $\overline{\mathbb{Q}}$FO-formulas (additionally, the disjointness requirement for formulas $\phi^{R^+}_\delta$ needs to be ensured).

The basic idea is to use two copies of the auxiliary relations, both alternating between empty and useful states, such that one copy is useful for even steps and the other one for odd steps. More precisely, for every auxiliary relation R used by $\mathcal{P}$, the program $\mathcal{P}'$ uses two auxiliary relations R_{even} and R_{odd} with the intention that after an even sequence of updates R_{even} stores the content of R after the same sequence of updates while R_{odd} is empty. After an odd sequence of updates R_{even} is empty while R_{odd} stores the content of R.

Then, for an even update, the relation R^+_{even} can be simply expressed as in absolute semantics (using "odd" relations) and R^-_{even} is empty. For an odd update R^-_{even} can be simply chosen as R_{even} and R^+_{even} is empty. Similarly for R_{odd}.

In the following we give a precise construction of $\mathcal{P}'$ over schema $\tau_{\text{even}} \cup \tau_{\text{odd}} \cup \{\textsc{Odd}, Q'\}$ where Odd is a boolean relation symbol, and τ_{even} and τ_{odd} contain, for every k-ary relation symbol $R \in \tau$, a k-ary relation symbol R_{even} and R_{odd}, respectively. The relation Odd is used to store the parity of the number of updates performed so far.

Let ϕ^R_δ be the update formula of $R \in \tau$ for an update δ in the dynamic program $\mathcal{P}$. Denote by $\phi^R_\delta[\tau \to \tau_{\text{even}}]$ the formula obtained from ϕ^R_δ by replacing every atom $S(\vec{x})$ with $S \in \tau$ by $S_{\text{even}}(\vec{x})$. Analogously for $\phi^R_\delta[\tau \to \tau_{\text{odd}}]$. Now, the update formulas for R_{odd} and R_{even} are as follows:

$$\phi^{R^+_{\text{odd}}}_\delta(\vec{u};\vec{x}) \stackrel{\text{def}}{=} \neg\textsc{Odd} \wedge \phi^R_\delta[\tau \to \tau_{\text{even}}](\vec{u};\vec{x})$$
$$\phi^{R^-_{\text{odd}}}_\delta(\vec{u};\vec{x}) \stackrel{\text{def}}{=} \textsc{Odd} \wedge R_{\text{odd}}(\vec{x})$$
$$\phi^{R^+_{\text{even}}}_\delta(\vec{u};\vec{x}) \stackrel{\text{def}}{=} \textsc{Odd} \wedge \phi^R_\delta[\tau \to \tau_{\text{odd}}](\vec{u};\vec{x})$$
$$\phi^{R^-_{\text{even}}}_\delta(\vec{u};\vec{x}) \stackrel{\text{def}}{=} \neg\textsc{Odd} \wedge R_{\text{even}}(\vec{x})$$

Observe that all those formulas can be easily converted into $\mathbb{Q}$FO-formulas. The boolean auxiliary relation Odd can be updated straightforwardly.

Now, since the update formulas of Q in $\mathcal{P}$ are quantifier-free, the relation Q' can be updated with the following quantifier-free update formulas:

$$\phi_\delta^{Q'^+}(\vec{u};\vec{x}) \stackrel{\text{def}}{=} \phi_\delta^{Q}(\vec{u};\vec{x}) \wedge \neg\Big(\big(\textsc{Odd} \wedge Q_{\text{odd}}(\vec{x})\big) \vee \big(\neg\textsc{Odd} \wedge Q_{\text{even}}(\vec{x})\big)\Big)$$

$$\phi_\delta^{Q'^-}(\vec{u};\vec{x}) \stackrel{\text{def}}{=} \neg\phi_\delta^{Q}(\vec{u};\vec{x}) \wedge \Big(\big(\textsc{Odd} \wedge Q_{\text{odd}}(\vec{x})\big) \vee \big(\neg\textsc{Odd} \wedge Q_{\text{even}}(\vec{x})\big)\Big)$$

The initialization mapping of P' is straightforward. Every $R_{\text{even}} \in \tau_{\text{even}}$ is initialized with $\textsc{Init}(R)$. All $R_{\text{odd}} \in \tau_{\text{odd}}$ are initialized with the empty relation. The relation $\textsc{Odd}$ is initialized with $\bot$, and Q' is initialized with $\textsc{Init}(Q)$. $\square$

Lemma 4.7. *(a) If a query can be maintained in* Δ-$\textsc{Dyn}$UCQ$^\neg$ *then it can be maintained in* $\textsc{Dyn}$UCQ$^\neg$ *as well.*

(b) If a query can be maintained in Δ-$\textsc{Dyn}\forall^*$FO *then it can be maintained in* $\textsc{Dyn}\forall^*$FO *as well.*

We note that the first statement could equally be expressed in terms of Δ-$\textsc{Dyn}\exists^*$FO and $\textsc{Dyn}\exists^*$FO.

Proof. We only prove (a), the proof of (b) is analogous. Let $\mathcal{P} = (P, \textsc{Init}, Q)$ be a dynamic Δ-$\textsc{Dyn}$UCQ$^\neg$-program over schema τ. By Lemma 4.5 we can assume, without loss of generality, that the update formulas of $\mathcal{P}$ are negation-free. For ease of presentation we assume that the input schema contains a single binary relation symbol E.

We construct an equivalent $\textsc{Dyn}$UCQ$^\neg$-program $\mathcal{P}'$ using the following idea. Consider some update formulas $\phi_\delta^{R^+}(\vec{u};\vec{x})$ and $\phi_\delta^{R^-}(\vec{u};\vec{x})$ of a relation $R \in \tau$ for an update δ in $\mathcal{P}$. The naïve translation into a $\textsc{Dyn}$FO-update formula $\phi_\delta^{R}(\vec{u};\vec{x})$ yields the formula

$$\phi_\delta^{R}(\vec{u};\vec{x}) = (R(\vec{x}) \vee \phi_\delta^{R^+}(\vec{u};\vec{x})) \wedge \neg\phi_\delta^{R^-}(\vec{u};\vec{x})$$

which is possibly non-UCQ$^\neg$ due to $\neg\phi_\delta^{R^-}(\vec{u};\vec{x})$. Therefore, $\mathcal{P}'$ maintains a relation R_δ^- that contains all tuples $(\vec{a},\vec{b})$ such that $\vec{a}$ would be removed from R after applying the update $\delta(\vec{b})$. Those relations are maintained using the squirrel technique.

The dynamic program $\mathcal{P}'$ is over schema $\tau \cup \tau_\Delta$ where τ_Δ contains a $(k+2)$-ary relation symbol $R_\delta^- \in \tau$ for every k-ary relation symbol $R \in \tau$ and every update $\delta \in \{\textsc{ins}, \textsc{del}\}$ of the input relation E.

The update formula for a relation symbol $R \in \tau$ is

$$\phi_\delta^{R}(\vec{u};\vec{x}) \stackrel{\text{def}}{=} (R(\vec{x}) \vee \phi_\delta^{R^+}(\vec{u};\vec{x})) \wedge \neg R_\delta^-(\vec{u},\vec{x})$$

This formula can be translated into an existential formula in a straightforward manner.

For updating a relation $R_{\delta_1}^-$ after an update δ_0, the update formula $\phi_{\delta_1}^{R^-}$ for R^- is used. However, since $R_{\delta_1}^-$ shall store tuples that have to be deleted after applying δ_1, the formula $\phi_{\delta_1}^{R^-}$ has to be adapted to use the content of relation symbols $S \in \tau$ after update δ_0 (instead, as usual, the content from before the update). For this purpose relation symbols $S \in \tau$ in $\phi_{\delta_1}^{R^-}$ need to be replaced by their update formulas as defined above.

The update formula for $R_{\delta_1}^-$ is

$$\phi_{\delta_0}^{R_{\delta_1}^-}(\vec{u}_0;\vec{u}_1,\vec{x}) \stackrel{\text{def}}{=} \phi_{\delta_0}^{R_{\delta_1}^-}[\tau \rightarrow \phi^\tau](\vec{u}_0;\vec{u}_1,\vec{x})$$

where $\phi_{\delta_0}^{R_{\delta_1}^-}[\tau \rightarrow \phi^\tau](\vec{u}_0;\vec{u}_1,\vec{x})$ is obtained from $\phi_{\delta_1}^{R^-}(\vec{u};\vec{x})$ by replacing every atom $S(\vec{z})$ by $\phi_{\delta_0}^{S}(\vec{u}_0;\vec{z})$, as constructed above. Since by our initial assumption, $\phi_{\delta_1}^{R^-}$ itself is an existential formula without negation and all update formulas $\phi_{\delta_0}^{S}$ for $S \in \tau$ are existential, the formula $\phi_{\delta_0}^{R_{\delta_1}^-}$ can be easily converted into an existential formula as well. $\square$

Lemmas 4.6 and 4.7 together yield Theorem 4.2. We now finally prove Theorem 4.1. For this we need the following adaption of Lemma 3.7 to Δ-semantics.

Lemma 4.8. *(a) For every* Δ-$\textsc{Dyn}$UCQ$^\neg$*-program there is an equivalent* Δ-$\textsc{Dyn}$CQ$^\neg$*-program.*

(b) For every Δ-$\textsc{Dyn}$FO*-program there is an equivalent* Δ-$\textsc{Dyn}$FO$^\wedge$*-program.*

Now, Theorem 4.1 follows from the Lemmata 4.5, 4.8, 4.6 and 4.7 as well as from Theorem 3.1.

5. A DYNAMIC CHARACTERIZATION OF FIRST-ORDER LOGIC

In this section we characterize first-order queries as the class of queries maintainable by non-recursive UCQ$^\neg$-programs and, equivalently, by non-recursive $\textsc{Dyn}\exists^1$FO-programs. Here $\exists^1$FO is the class of queries expressible by first-order formulas in prenex normal form with at most one existential quantifier and no universal quantifiers, and "non-recursive" is explained next. This characterization in combination with Theorem 3.1 yields that first-order queries can be dynamically maintained by CQ$^\neg$-programs.

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. A dynamic program is *non-recursive* if it has an acyclic dependency graph (as a directed graph). For every class $\mathcal{C}$, *non-recursive* $\textsc{Dyn}\mathcal{C}$ refers to the set of queries that can be maintained by non-recursive $\textsc{Dyn}\mathcal{C}$-programs.

The objective of this section is to prove the following theorem.

Theorem 5.1. *For every query* $\mathcal{Q}$ *the following statements are equivalent*

(a) $\mathcal{Q}$ *can be expressed in* FO.

(b) $\mathcal{Q}$ *can be maintained in non-recursive* $\textsc{Dyn}$FO.

(c) $\mathcal{Q}$ *can be maintained in non-recursive* $\textsc{Dyn}\exists^1$FO.

(d) $\mathcal{Q}$ *can be maintained in non-recursive* $\textsc{Dyn}\forall^1$FO.

With respect to the number of quantifiers in update formulas this result is optimal because the first-order definable alternating reachability query on graphs of bounded diameter cannot be maintained with quantifier-free update formulas [9]. Theorem 5.1 should be compared with the result of [9] that all $\exists^*$FO queries can be maintained with quantifier-free update programs extended by auxiliary *functions*.

Combining Theorem 5.1 with Theorem 3.1 immediately yields the following corollary.

Corollary 5.2. *Every first-order query can be maintained in* $\textsc{Dyn}$CQ$^\neg$.

The rest of this section is devoted to the proof of Theorem 5.1, more precisely to the equivalence of statements (a)-(c). The equivalence with (d) follows from Theorem 3.1 and the fact that its proof does not introduce recursion when applied to a non-recursive program.[11] It is obvious that (c) implies (b). For ease of presentation, we prove the remaining directions (a)⇒(c) and (b)⇒(a) for the input schema $\tau_{\text{in}} = \{E\}$ where E is a binary relation symbol. The proofs can be easily adapted to general (relational) signatures.

The following example outlines the idea of the construction for the proof of (a)⇒(c).

Example 3. Consider the query $\mathcal{Q}$ defined by

$$\begin{aligned}\varphi &= \exists x \forall y \big(E(x,x) \to E(x,y)\big)\\ &\equiv \exists x \neg \exists y \neg \big(E(x,x) \to E(x,y)\big)\end{aligned}$$

We construct a non-recursive dynamic $\text{DYN}\exists^1\text{FO}$-program $\mathcal{P}$ that maintains $\mathcal{Q}$ under deletions only (for simplicity). The construction of $\mathcal{P}$ uses the squirrel technique. It uses a separate auxiliary relation R_ψ for each subformula ψ obtained from φ by stripping off a "quantifier prefix" from the existential prefix form of φ. The relation R_ψ reflects the possible states after a sequence of changes whose length equals the number of stripped off $\neg$- and $\exists$-symbols.

In order to update the query relation after the deletion of an edge, we maintain an auxiliary ternary relation[12] R_1 that contains the result of the query $\psi_1 \stackrel{\text{def}}{=} \neg\exists y \neg\big(E(x,x) \to E(x,y)\big)$ for every choice a_1 for x and every (possibly deleted) edge $\vec{e}_1$, that is $(a_1, \vec{e}_1) \in R_1$ if and only if

$$(V, E \setminus \{\vec{e}_1\}, \{x \mapsto a_1\}) \models \forall y \big(E(x,x) \to E(x,y)\big))$$

Then we can define $\phi^{Q}_{\text{DEL}}(\vec{v}_1) \stackrel{\text{def}}{=} \exists x R_1(x, \vec{v}_1)$ and it only remains to find a way to update the relation R_1. To this end, we maintain a further relation R_2 that contains the result of $\psi_2 \stackrel{\text{def}}{=} \exists y \neg\big(E(x,x) \to E(x,y)\big)$ for every choice a_1 for x and all (possibly deleted) edges $\vec{e}_1, \vec{e}_2$, that is $(a_1, \vec{e}_1, \vec{e}_2) \in R_2$ if and only if

$$(V, E \setminus \{\vec{e}_1, \vec{e}_2\}, \{x \mapsto a_1\}) \models \exists y \neg\big(E(x,x) \to E(x,y)\big))$$

Then $\phi^{R_1}_{\text{DEL}}(\vec{v}_1; x, \vec{v}_2) \stackrel{\text{def}}{=} \neg R_2(x, \vec{v}_1, \vec{v}_2)$ and it remains to update the relation R_2. Therefore we maintain a relation R_3 that contains the result of $\psi_3 = \neg\big(E(x,x) \to E(x,y)\big)$ for every choice a_1, a_2 for x, y and all (possibly deleted) edges $\vec{e}_1, \vec{e}_2, \vec{e}_3$. Then

$$\phi^{R_2}_{\text{DEL}}(\vec{v}_1; x, \vec{v}_2, \vec{v}_3) \stackrel{\text{def}}{=} \exists y R_3(x, y, \vec{v}_1, \vec{v}_2, \vec{v}_3)$$

and it remains to update relation R_3 via

$$\begin{aligned}&\phi^{R_3}_{\text{DEL}}(\vec{v}_1; x, y, \vec{v}_2, \vec{v}_3, \vec{v}_4) \stackrel{\text{def}}{=}\\ &\qquad \neg\big(E'(x, x, \vec{v}_1, \dots, \vec{v}_4) \to E'(x, y, \vec{v}_1, \dots, \vec{v}_4)\big)\end{aligned}$$

where E' is the edge relation obtained from E by deleting $\vec{v}_1$, $\vec{v}_2$, $\vec{v}_3$ and $\vec{v}_4$, that is $E'(x, y, \vec{v}_1, \dots, \vec{v}_4)$ can be replaced by

$$E(x,y) \wedge (x,y) \neq \vec{v}_1 \wedge \ldots \wedge (x,y) \neq \vec{v}_4.$$

This completes the description of the program $\mathcal{P}$ for φ which is easily seen to be non-recursive.

[11] Alternatively, the proof of (a)⇒(c) can be easily adapted to show (a)⇒(d)

[12] For simplicity we write R_1 instead of R_{ψ_1}.

The proof of the implication (b)⇒(a) of Theorem 5.1 is based on the squirrel technique with the following idea. Given a non-recursive dynamic DYNFO-program $\mathcal{P} = (P, \text{INIT}, Q)$, we construct (again), for every update pattern $\delta = \delta_1 \ldots \delta_j$ and every auxiliary relation R, a first-order formula φ^R_δ that "precomputes" the state of R for every possible update sequence with the pattern δ. Thanks to non-recursiveness, the formula φ^R_δ can only use relations from the input schema, once δ is longer than the number of auxiliary relations, that is, it is just a first-order formula over τ_{in}. To obtain a first-order formula for Q it thus suffices to pick such a formula with an update sequence δ that does not change the structure. We now turn to the formal statement and proof of the result.

A *topological sorting* of a graph (V, E) is a sequence $v_1, \ldots, v_n$ such that every vertex from V occurs exactly once and $i > j$ for all edges $(v_i, v_j) \in E$. Every acyclic graph has a topological sorting. In particular, if $R_1, \ldots, R_m$ is a topological sorting of the dependency graph of a non-recursive dynamic program $\mathcal{P} = (P, \text{INIT}, Q)$ then update formulas for R_1 do only contain relation symbols from τ_{in}. Further we can assume, without loss of generality, that $R_m = Q$.

The following definition will be useful in the proof of Lemma 5.3. For every first-order formula φ with k free variables and every sequence $\delta = \delta_1 \ldots \delta_j$ over $\{\text{INS}, \text{DEL}\}$ let $\varphi^E_{\delta_1 \ldots \delta_j}$ be a $(k + 2j)$-ary formula such that for every graph $G = (V, E)$, every $\vec{a} \in V^k$ and every instantiation $\alpha = \delta_1(\vec{e}_1) \ldots \delta_2(\vec{e}_j)$ of δ with tuples $\vec{e}_1, \ldots, \vec{e}_j \in V^2$:

$$\alpha(G) \models \varphi \text{ if and only if } G \models \varphi_{\delta_1 \ldots \delta_j}(\vec{a}, \vec{e}_1, \ldots, \vec{e}_j).$$

It is straightforward to construct $\varphi^E_{\delta_1 \ldots \delta_j}$.

LEMMA 5.3. *If a query can be maintained in non-recursive* DYNFO*, then it can be expressed in* FO.

PROOF. Let $\mathcal{Q}$ be a query which can be maintained by a non-recursive DYNFO-program $\mathcal{P} = (P, \text{INIT}, Q)$ over schema $\tau = \tau_{\text{in}} \cup \tau_{\text{aux}}$. We assume for simplicity that $\tau_{\text{in}} = \{E\}$, for a binary symbol E. We let $R_0 \stackrel{\text{def}}{=} E$ and assume that the auxiliary relations $R_1, \ldots, R_m$ are enumerated with respect to a topological sorting of the dependency graph of $\mathcal{P}$ with $R_m = Q$.

We define inductively, by i, for every sequence $\delta_1 \ldots \delta_j$ with $j \geq i$, first-order formulas $\varphi^{R_i}_{\delta_1 \ldots \delta_j}(\vec{y}, \vec{x}_1, \ldots, \vec{x}_j)$ over schema $\tau_{\text{in}} = \{E\}$ such that $\varphi^{R_i}_{\delta_1 \ldots \delta_j}$ defines R_i after updates $\delta_1(\vec{x}_1) \ldots \delta_j(\vec{x}_j)$. More precisely $\varphi^{R_i}_{\delta_1 \ldots \delta_j}$ will be defined such that for every state $\mathcal{S} = (V, E^{\mathcal{S}}, \mathcal{A}^{\mathcal{S}})$ of $\mathcal{P}$ and every sequence $\delta = \delta_1(\vec{a}_1) \ldots \delta_j(\vec{a}_j)$ of updates the following holds:

$$\mathcal{P}_\delta(\mathcal{S}) \upharpoonright R_i = \{\vec{b} \mid (V, E) \models \varphi^{R_i}_{\delta_1 \ldots \delta_j}(\vec{b}, \vec{a}_1, \ldots, \vec{a}_j)\} \qquad (2)$$

For $R_0 = E$ the formula $\varphi^E_{\delta_1 \ldots \delta_j}$ can be defined just as in the previous lemma. For R_i with $i \geq 1$ the formula $\varphi^{R_i}_{\delta_1 \ldots \delta_j}(\vec{y}, \vec{x}_1, \ldots, \vec{x}_j)$ is obtained from the update formula $\phi^{R_i}_{\delta_j}(\vec{x}_j; \vec{y})$ of R_i by substituting all occurrences of $R_{i'}(\vec{z})$ by $\varphi^{R_{i'}}_{\delta_1 \ldots \delta_{j-1}}(\vec{x}_1, \ldots, \vec{x}_{j-1}, \vec{z})$ for all $i' < i$. Using induction over i, one can prove that the formulas $\varphi^{R_i}_{\delta_1 \ldots \delta_j}$ satisfy Equation 2. As $\mathcal{P}$ is non-recursive, each formula $\varphi^{R_i}_{\delta_1 \ldots \delta_j}$ with $j \geq i$ is over schema $\{E\}$.

The first-order formula φ for $\mathcal{Q}$ over schema $\tau_{\text{in}} = \{E\}$ can be constructed as follows. The formula "guesses" a tuple

$\vec{a} \in E$, deletes and inserts it m times and applies $\varphi^{R_m}_{(\text{DEL INS})^m}$ to the result (which is identical to the current graph), or (for the case that E is empty) it guesses a tuple $\vec{a} \notin E$, inserts and deletes it m times and applies $\varphi^{R_m}_{(\text{INS DEL})^m}$ to the result.

More precisely, φ for $\mathcal{Q}$ is defined by

$$\varphi(\vec{y}) \stackrel{\text{def}}{=} \exists \vec{x}\big((E(\vec{x}) \wedge \varphi^{R_m}_{(\text{DEL INS})^m}(\vec{y}, \underbrace{\vec{x}, \vec{x}, \ldots, \vec{x}}_{2m-\text{times}})) \vee (\neg E(\vec{x}) \wedge \varphi^{R_m}_{(\text{INS DEL})^m}(\underbrace{\vec{y}, \vec{x}, \vec{x}, \ldots, \vec{x}}_{2m-\text{times}}))\big).$$

□

6. DISCUSSION AND FUTURE WORK

In this work, we studied dynamic conjunctive queries. We have shown that, contrary to the static setting, many fragments collapse in the dynamic world. Furthermore, a close connection between absolute semantics and Δ-semantics for conjunctive queries has been established. These results were summarized in Figure 1. Finally, it has been shown that dynamic conjunctive queries with negations capture (static) first-order logic.

All results are for arbitrary initialization mappings. However, they also hold in the setting with first-order definable initialization mappings. They do not carry over when the initialization mapping and updates have to be definable in the same class.

Some first steps towards separation of the remaining classes have been taken. We only state the results; the proofs will appear in the full version of this work.

THEOREM 6.1. *The class* DynPropCQ *is a strict subclass of* DynProp.

THEOREM 6.2. *The class* DynProp *is a strict subclass of* DynCQ.

The first result requires some work. The second result relies on the observation that DynCQ captures the dynamic class DynQF, that is, the extension of DynProp by auxiliary functions; and the separation of DynQF and DynProp [9]. The even weaker dynamic class DynProjections (where update formulas are restricted to be projections) was separated from DynProp already in [12].

Whether the remaining classes DynCQ, DynCQ$^\neg$ and DynFO can be separated or collapsed remains open.

In addition to untangling the remaining variations of conjunctive queries, the dynamic quantifier hierarchy and quantifier alternation hierarchy, respectively, deserve a closer look. Lemma 3.9 shows that in the dynamic setting the Σ_i- and Π_i-fragment of first-order logic coincide. Whether there is a strict Σ_i-hierarchy remains open. Furthermore, the equivalence of $\exists^*$FO with absolute and Δ-semantics does not immediately translate to fragments of FO with alternating quantifiers (although one of the direction does, see Lemma 4.6).

Capturing first-order logic by dynamic conjunctive queries with negations does not immediately yield performance gains (since a first-order queries with k quantifiers is translated to a dynamic DynCQ$^\neg$-program of arity at least k). In future work we plan to study whether the work that has been started here can be used to improve the performance of query maintenance.

7. REFERENCES

[1] Yanif Ahmad, Oliver Kennedy, Christoph Koch, and Milos Nikolic. Dbtoaster: Higher-order delta processing for dynamic, frequently fresh views. *PVLDB*, 5(10):968–979, 2012.

[2] Ashok K Chandra and Philip M Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 77–90. ACM, 1977.

[3] Guozhu Dong, Leonid Libkin, and Limsoon Wong. On impossibility of decremental recomputation of recursive queries in relational calculus and SQL. In *DBPL*, page 7, 1995.

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

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

[6] 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.

[7] Guozhu Dong and Rodney W. Topor. Incremental evaluation of datalog queries. In *ICDT*, pages 282–296, 1992.

[8] Kousha Etessami. Dynamic tree isomorphism via first-order updates. In *PODS*, pages 235–243. ACM Press, 1998.

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

[10] Erich Grädel and Sebastian Siebertz. Dynamic definability. In *ICDT*, pages 236–248, 2012.

[11] Ashish Gupta, Inderpal Singh Mumick, and Venkatramanan Siva Subrahmanian. Maintaining views incrementally. In *ACM SIGMOD Record*, volume 22, pages 157–166. ACM, 1993.

[12] W. Hesse. *Conditional and unconditional separations of dynamic complexity classes.* Unpublished manuscript, 2003.

[13] William Hesse. The dynamic complexity of transitive closure is in DynTC^0. In *ICDT*, pages 234–247, 2001.

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

[15] Christoph Koch. Incremental query evaluation in a ring of databases. In *PODS*, pages 87–98, 2010.

[16] Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. In *PODS*, pages 210–221. ACM Press, 1994.

[17] Oded Shmueli and Alon Itai. Maintenance of views. In *SIGMOD Conference*, pages 240–255, 1984.

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

[19] Thomas Zeume and Thomas Schwentick. On the quantifier-free dynamic complexity of reachability. In *MFCS*, 2013.