Dynamic Complexity: Recent Updates

Thomas Schwentick
TU Dortmund University

Thomas Zeume
TU Dortmund University

1. INTRODUCTION

In many data management scenarios the data is subject to frequent modifications, and it is often essential to react to those changes quickly. When a train is canceled on short notice, travelers need to find alternative connections as fast as possible. When a web server is temporarily not available, data packages have to be rerouted immediately.

Recomputation of a query result from scratch after each small change of the data is often not possible in such scenarios due to the large amount of data at hand and efficiency considerations. Very often it is also not necessary: the breakdown of a single train does usually affect only a small fraction of the whole train network. Thus it is reasonable to try to dynamically update essential information in an incremental fashion by reusing information that has been previously computed. Ideally such a dynamic update should use less resources than recomputation from scratch.

Besides saving resources, a dynamic approach to query answering can increase the expressivity of database query languages. The relational algebra (corresponding to the core of SQL) can only express queries that can be formulated in first-order logic (aka relational calculus) and therefore inherits the well-known expressivity limitations of first-order logic. In a nutshell, it thus can only express local queries that do not count (see [Libkin 2004] for more information on the limits of first-order logic). However, when previously computed information is available after a change of the data, query results can be "built-up" over time, and therefore queries that the relational algebra cannot express might be maintainable in this dynamic setting. For example, whether the size of a set is odd or even can be easily maintained under single insertion and deletion operations with the help of a single bit of auxiliary (stored) data.

One way to model this dynamic scenario is the descriptive dynamic complexity framework (short: dynamic complexity) introduced independently by Dong, Su and Topor [Dong and Su 1993; Dong and Topor 1992] and Patnaik and Immerman [Patnaik and Immerman 1994]. It was mainly inspired by updates in relational databases. Within this framework, for a relational database subject to change, auxiliary relations are maintained to help answering a query Q. When an insertion or deletion of a tuple to the database occurs, every auxiliary relation is updated through a first-order query that can refer to the database as well as to the auxiliary relations (cf. Figure 1). The class of all queries maintainable in this way is called DYNFO.

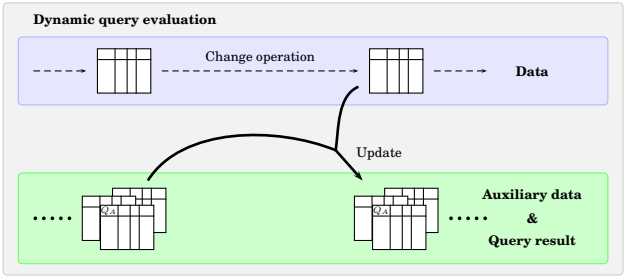

Fig. 1. The dynamic setting: after a change operation the auxiliary data is updated and one of the auxiliary relations, $\mathcal{Q}_A$, yields the result of $\mathcal{Q}$.

Example 1.1. The reachability query asks for all pairs of nodes that are connected by a path in a given graph G. A naïve approach for maintaining this query is to use a binary auxiliary relation T which represents the transitive closure of the graph. We show how to update T under *edge insertions*. When an edge (c, d) is inserted then there is a path from a vertex a to a vertex b if (1) there had been a path from a to b before the insertion, or (2) there had been a path from a to c and a path from d to b before the insertion. This can be easily specified by a first-order update rule as follows.

on insert (u, v) **into** E
update $T(x, y)$ **as** $T(x, y) \vee \big(T(x, u) \wedge T(v, y)\big)$

The new version of relation T then consists of all pairs (x, y) for which the formula $T(x, y) \vee \big(T(x, u) \wedge T(v, y)\big)$ holds, where (u, v) represents the inserted edge.

This simple example just illustrates the framework. In fact, it is not possible to update T under edge *deletions* without further auxiliary relations (see Theorem 5.1, shown in [Dong and Su 1998]).

As one of the easiest queries that requires some recursion, the reachability query has been by far the most intensely studied query in dynamic complexity. We already saw that it can be maintained when edges can only be inserted. Patnaik and Immerman conjectured that Reachability can be maintained under insertion and deletion operations with first-order update programs [Patnaik and Immerman 1997]. This has been confirmed[1] recently in [Datta et al. 2015].

The main purpose of this article is to give a high-level proof sketch of this result (in Section 4). Furthermore, it gives a detailed discussion of possible dynamic settings (Section 3), some inexpressibility results and techniques (Section 5), and some pointers to the literature for further topics (Section 6). In Section 2 we fix some notation and formally define the dynamic setting that will be used in this article. The article is not an exhaustive survey on dynamic complexity.

Readers mainly interested in the upper bound for Reachability might directly jump to Section 4 and consult Section 2 for the precise notation.

We borrow material from several talks we presented in the last few years as well as from some of our articles [Gelade et al. 2012; Zeume 2015; Zeume and Schwentick 2014, 2015]. For a more complete exposition of the current state of the art of dynamic complexity we refer to [Zeume 2015].

[1]As a side remark: the authors of this article had started a research project, funded by the German DFG, with the aim to develop lower bound methods and the (remote) ultimate goal to prove REACH $\notin$ DYNFO. Research is full of surprises.

2. BASIC DEFINITIONS

For each $m \in \mathbb{N}$, the set $\{0, \ldots, m-1\}$ is denoted by $[m]$. The reachability query REACH is defined as usual.

Problem: REACH
Input: Directed graph G
Output: Set of all pairs (u, v), for which there is a path from u to v in G

We have already seen a dynamic program in the introduction. In general, a dynamic program[2] $\mathcal{P}$ works on an *input structure* $\mathcal{I}$ over a schema τ_{inp} and updates an *auxiliary structure* $\mathcal{A}$ over a schema[3] τ_{aux}. Both structures $\mathcal{I}$ and $\mathcal{A}$ share the same domain D which does not change during a computation. We call a pair $(\mathcal{I}, \mathcal{A})$ a *state* and consider it as one relational structure. The relations of $\mathcal{I}$ and $\mathcal{A}$ are called *input and auxiliary relations*, respectively. Input relations can be *changed* by inserting or deleting a single tuple. A *change operation* is thus of the form **insert** $\vec{t}$ **into** R or **delete** $\vec{t}$ **from** R, for some tuple $\vec{t}$ and input relation R. For a sequence α of change operations and an input database $\mathcal{I}$, we denote the database resulting from applying α to $\mathcal{I}$ by $\alpha(\mathcal{I})$.

A *dynamic program* has a set of update rules that specify how auxiliary relations are updated after a change. An *update rule* for updating an auxiliary relation T after inserting a tuple into an input relation R is of the form

on insert $\vec{x}$ **into** R
update $T(\vec{y})$ **as** $\varphi(\vec{x}, \vec{y})$

where the formula φ is over $\tau_{\text{inp}} \cup \tau_{\text{aux}}$. We often refer to φ as the *update formula*.

The semantics of such an update rule is as follows. When a tuple $\vec{a}$ is inserted into input relation R, then the new state $\mathcal{S}$ of $\mathcal{P}$ is obtained by inserting $\vec{a}$ into R and by defining each auxiliary relation T via $T \stackrel{\text{def}}{=} \{\vec{b} \mid (\mathcal{I}, \mathcal{A}) \models \varphi(\vec{a}, \vec{b})\}$. Similarly for deletions. For a change operation δ we denote the updated state by $\mathcal{P}_\delta(\mathcal{S})$, and similarly for sequences of changes.

The dynamic program $\mathcal{P}$ *maintains* a k-ary query $\mathcal{Q}$ if it has a k-ary auxiliary relation Q_A that, after each change sequence, contains the result of $\mathcal{Q}$ on the current input database. More precisely, for each non-empty[4] sequence α of changes and each empty input structure $\mathcal{I}_\emptyset$, relation Q_A in $\mathcal{P}_\alpha(\mathcal{S}_\emptyset)$ and $\mathcal{Q}(\alpha(\mathcal{I}_\emptyset))$ coincide. Here, $\mathcal{S}_\emptyset = (\mathcal{I}_\emptyset, \mathcal{A}_\emptyset)$, where $\mathcal{A}_\emptyset$ denotes the empty auxiliary structure over the domain of $\mathcal{I}_\emptyset$.

The class of queries that can be maintained by a dynamic program with update formulas from first-order logic is called DYNFO.

The precise relationship between DYNFO and standard (static) complexity classes is unknown. However, a lower and an upper bound can be easily observed: every query that can be expressed in FO can also be maintained in DYNFO. On the other hand, every query in DYNFO can be evaluated in polynomial time.

We will make use of a very weak form of logically defined reductions under which DYNFO is closed. In a nutshell, a first-order reduction ρ from a Boolean query Q to a Boolean query Q' maps every structure $\mathcal{A}$ to a structure $\mathcal{B} = \rho(\mathcal{A})$ such that $Q(\mathcal{A})$ is

[2]We note that, although, in principle, we use the setting of [Patnaik and Immerman 1997], our notation considerably departs from [Patnaik and Immerman 1997]. This should not hurt much, since we try to avoid too much notational detail.

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

[4]This restriction is needed for technical reasons. Otherwise, we could not handle Boolean queries with a yes-result on empty structures. Alternatively, one could use an extra formula to compute the query result from the auxiliary (and input) structure.

true if and only if $Q'(\mathcal{B})$ is true. For the precise definition of first-order reductions we refer to [Patnaik and Immerman 1997]. A first-order reduction is[5] *bounded* if there is a constant c such that every single-tuple change in $\mathcal{A}$ yields at most c changes in $\mathcal{B}$.

3. THE SETTING

Although the principal idea of logic-based dynamic complexity is quite simple, several design choices have to be made in a formalization. We picked one particular setting in the definition of DYNFO in Section 2. However, there are many aspects of the model that can be defined differently.

We discuss some of the available design choices in this section. We first discuss alternatives with respect to the allowed change operations, and afterwards alternatives regarding the definition of update programs. Towards the end of the section, we compare the DYNFO setting with other dynamic settings.

3.1. Change Operations

The DYNFO-setting only allows single-tuple changes: in one change operation, a single tuple can be added to or deleted from the current input structure. An alternative could be to allow insertion or deletion of a set of tuples. However, allowing to insert and delete *arbitrary* sets of tuples would make the dynamic setting pointless, since in one step one could jump between two arbitrary inputs. Therefore, set-valued changes seem to make sense only if they are restricted in some way, e.g., by bounding the size of change sets or by requiring them to be *defined* by a formula or algebraic expression.

The restriction to single-tuple changes is severe from a practical point of view. However, it turns out that already this simple type of changes raises difficult research challenges. For the same reason, the restriction to single tuple changes is also common in the study of dynamic algorithms.[6]

The DYNFO setting comes with a restriction that looks unnatural to many people, when they see it for the first time: the domain (universe) of the structure at hand is fixed, that is, it is not possible to add or remove elements from the current domain. In particular, new tuples can only come with elements that are already present in the current structure.[7] However, update programs of one setting can usually be adapted to the other setting in a straightforward manner.

3.2. Update Programs

Naturally, the machinery for update programs gives rise to many alternatives. We distinguish three "sub-aspects" here:

(1) the logic used for updates,
(2) the "data structures", i.e., the signature of the auxiliary structure, and
(3) the initialization mechanism.

3.2.1. Logic for Updates. In our definition of DYNFO, the logic used for updates is first-order predicate logic, but many other logics have been considered:

— extensions of first-order logic by, e.g., counting quantifiers to be able to capture queries that seem(ed) beyond DYNFO, and
— restrictions of first-order logic, especially with the goal to prove lower bounds or to pinpoint the dynamic complexity of a given query more precisely; these restrictions

[5] In [Patnaik and Immerman 1997] these reductions were called bounded-expansion reductions, but this name is now in use for other, unrelated, concepts.

[6] Nevertheless, we plan to study simple *defined changes* in the near future.

[7] This is the main difference between the DYNFO-setting and the FOIES-framework proposed by [Dong and Su 1993; Dong and Topor 1992].

most often affect the quantifier structure of formulas (and particularly might forbid quantifiers completely).

The precise semantics of update programs is also important, e.g., whether update formulas specify the new state of the auxiliary data explicitly or as the "delta" with respect to the previous state. The definition of DYNFO uses explicit specification, but this difference obviously does not matter. One might suspect that it matters for restricted logics, but we will see in Section 6 that most often the choice of the semantics does not influence the expressive power.

3.2.2. Data Structures. The arity of the auxiliary relations (or functions) can influence the dynamic expressive power of a logic. Many lower bound results involve some arity restriction, e.g., the reachability query can not be maintained in DYNFO with (at most) unary auxiliary relations (cf. Theorem 5.1).

For DYNFO, it does not matter whether the auxiliary data is stored in functions or relations. However, for restrictions of first-order logic, this distinction might become relevant. The quantifier-free fragment of DYNFO which allows relations and functions[8] is called DYNQF and the one that only allows relations is called DYNPROP [Hesse 2003b]. Since we mainly deal with DYNFO, we assume in the following that the auxiliary data consists of relations only, unless otherwise stated.

3.2.3. Initialization Mechanism. With respect to initialization, there are many alternatives, as well. Of course, the main interest of dynamic complexity is to understand how auxiliary data can be used to *maintain* a query. However, when it comes to the formal definitions, one has to decide in which situation the dynamic computation *starts*. Two questions need to be answered here:

— what are the possible initial values for the input structure, and
— how are the initial values of the auxiliary relations determined?

To give a first glimpse of possible pitfalls, let us assume our answer is "arbitrary" for the first question and that the initial auxiliary relations are "empty". With that choice a query could be *maintained* in first-order logic (if and) only if it can be already *expressed* in first-order logic. This holds because, at the start of a dynamic computation, the dynamic program would need to be able to react to arbitrary structures without the help of auxiliary data. And thus the query would be also expressible by the first-order update formula for the query relation.

Our definition of DYNFO adopted the setting that has been used most often in the literature, in which the input structure is initially empty and the auxiliary relations are empty as well [Patnaik and Immerman 1994]. With respect to the initialization, the definition DYNFO appears very innocent at first sight: what could be wrong about initially empty auxiliary relations? However, the setting allows a dynamic program to establish a linear order on the "activated" elements of the structure, i.e., those elements that are currently in some tuple or had been in some tuple before [Patnaik and Immerman 1997]. It is even possible to establish arithmetic relations (addition and multiplication) on the activated elements [Etessami 1998]. However, from a practical point of view, this setting seems very plausible and the ability to use a linear order or arithmetical relations to maintain queries does not contradict intuition.

[8]For the update of functions, function terms that involve an if-then-else mechanism can be used [Hesse 2003b].

The DYNFO setting can be strengthened by allowing the auxiliary relations to be initially non-empty, even though the initial structure has no tuples. This setting reminds one of notions of circuit families, where, for each input length n, there is a circuit C_i that decides all strings of this length. Like in the world of circuit families one can distinguish the case of non-uniform initialization (the initial value of the auxiliary relations is just an arbitrary function of the size of the universe) and various levels of uniform initialization (where this function is somehow restricted). Restrictions that played a role in the literature include polynomial-time computable initializations [Patnaik and Immerman 1994] and just linear order, addition and multiplication on the whole universe [Etessami 1998]. We refer to the class of queries that can be maintained by dynamic programs of the latter kind as DYNFO$(+,\times)$. This class is used in the second step of the proof that the reachability query is in DYNFO (cf. Proposition 4.2). Non-uniform initialization was considered in [Datta et al. 2014] (for an extension of DYNFO).

Orthogonally, the DYNFO setting can be varied by allowing initial situations, in which the structure under consideration is non-empty. Besides the non-sensical variant with empty initial auxiliary relations, mentioned above, non-uniform and uniform initializations of the auxiliary relations can be considered here as well. It is not hard to see that for non-uniform initialization the expressive power is the same, no matter whether the initial input structure is empty or non-empty. The non-uniform setting emphasizes the *maintenance aspect* as it somehow "abstracts away" the effect of the initialization. Thus, lower bounds against this setting are guaranteed to show a lack of maintainability as opposed to proofs that might exploit the weakness of initialization. Thus, inexpressibility results for this setting are the most powerful and therefore, the most desirable ones.

There is another, more subtle, aspect that makes the setting with non-empty initial structures interesting: it allows to study the maintainabilty of properties in the context of unordered input structures. Such investigations have been carried out by Grädel and Siebertz [Grädel and Siebertz 2012]. They considered the definition of initial auxiliary relations by extensions of first-order logic like inflationary fixed-point logic with or without counting quantifiers. This form of initialization is, in general, not able to define a linear order on unordered structures (let alone arithmetic) and allows to prove inexpressibility results.

The following table gives an overview of the main settings considered in this article.

Table I. Overview of settings and dynamic classes.

Initial auxiliary relations	**Initial input structure**	
	Empty	**Arbitrary**
Empty	DYNFO	FO
$+,\times$	DYNFO$(+,\times)$	FO$(+,\times)$
Arbitrary	Non-uniform DYNFO	

The interplay of different initializations has been investigated in [Datta et al. 2015; Grädel and Siebertz 2012; Patnaik and Immerman 1997; Weber and Schwentick 2007; Zeume and Schwentick 2015].

3.3. DYNFO and dynamic algorithms

The original motivation for the introduction of the framework of dynamic complexity was to study the dynamic evaluation of relational database queries from a Logic and Complexity perspective. The framework captures a restricted form of incremental view

maintenance in which the underlying database changes by only one tuple at a time. It is therefore natural to consider first-order logic as update language, since it directly corresponds to the relational algebra which in turn is subsumed by the database query language SQL. Thus, queries in DYNFO can be maintained by SQL updates.

Dynamic problems have been studied extensively from an algorithmic point of view as well. In this area the focus is on developing algorithms that need less resources for recomputing query results after modifications than a naïve algorithm that recomputes results from scratch. A good starting point for readers interested in dynamic algorithms are [Demetrescu and Italiano 2008; Roditty and Zwick 2008] (for upper bounds) and the survey by Miltersen on cell probe complexity [Miltersen 1999] (for lower bounds). A sequential framework for studying dynamic complexity was proposed in [Miltersen et al. 1994].

The consideration of first-order logic as an update language also makes sense from an algorithmic point of view. Since the (uniform version of the) circuit complexity class AC^0 corresponds to $\mathrm{FO}(+,\times)$ [Barrington et al. 1990], $\mathrm{DYNFO}(+,\times)$ can be seen as a dynamic version of AC^0. On the other hand, circuits from this class can be simulated by parallel random access machines (short: PRAMs) with polynomially many processors in constant time (see, e.g., [Vollmer 1999]). Thus if a query can be maintained via first-order update formulas (with built-in arithmetic), it can be dynamically recomputed by a highly parallel program in constant time as well. This low parallel complexity does not necessarily translate into fast sequential algorithms in the sense of dynamic algorithms, and it is not immediately clear how to implement first-order update programs in real systems. However, results from dynamic descriptive complexity offer a foundation for future work towards fast, parallel dynamic programs for important queries.

4. THE DYNAMIC COMPLEXITY OF REACHABILITY

Before we sketch a proof for the first-order maintainability of the reachability query, we give a short account of previous dynamic expressibility results for the reachability query. For lower bound results we refer to Section 5.

Whether $\mathrm{REACH} \in \mathrm{DYNFO}$, i.e., whether the reachability query can be maintained by first-order update programs, has been one of the main questions studied in the field of dynamic complexity. The positive results that were obtained towards the resolution of this question can be clustered in two groups:

(1) results that show how to maintain REACH on restricted classes of graphs, and
(2) results that show how to maintain REACH in extensions of DYNFO.

In a sense, the latter line of research has won this race, since the methods developed there ultimately yielded a DYNFO program for REACH. We will report about the limited success of a third line of research that worked on inexpressibility results in the DYNFO setting in Section 5.

Results of group (1) showed that the reachability query can be maintained in DYNFO for undirected graphs [Patnaik and Immerman 1994], directed acyclic graphs [Dong and Su 1993], and embedded planar graphs [Datta et al. 2014]. For undirected graphs, reachability can even be maintained in DYNQF (i.e., with quantifier-free formulas using auxiliary functions) and for acyclic deterministic graphs even in DYNPROP (i.e., with quantifier-free formulas with auxiliary relations) [Hesse 2003b].

In the case of undirected graphs, spanning trees [Patnaik and Immerman 1994] or distance functions [Grädel and Siebertz 2012] can be used. In the case of directed acyclic graphs a smart observation that allows to figure out whether there is a path from a to b after deleting some edge (c, d) can be used [Dong and Su 1993].

The first result in the second group was that, on arbitrary directed graphs, Reachability can be maintained in DYNTC^0 [Hesse 2003a]. The technique was based on using generating functions for representing the number of paths of a given length from one node to another and on the observation that only paths up to length n have to be considered. In [Datta et al. 2014] it was shown by a similar approach that Reachability can be even maintained in DYNAC$^0[2]$. In terms of logic, DYNAC$^0[2]$ can be seen as the extension of non-uniform DYNFO in which update formulas are allowed to use modulo-2 counting quantifiers. This paper initiated the study of the dynamic complexity of matrix rank (putting it in DYNTC^0) which eventually led to the result that Reachability can be maintained in DYNFO [Datta et al. 2015]. We describe the surprisingly elementary proof and some consequences of this result in the remainder of this section.

THEOREM 4.1 ([DATTA ET AL. 2015]). REACH $\in$ DYNFO.

For technical simplicity, we will sketch the proof for the Boolean s-t-reachability query instead of REACH, since a program for REACH can be obtained from a program for s-t-REACH by running it in parallel, for every pair (u, v) of vertices.

Problem:	s-t-REACH
Input:	Directed graph G, nodes s, t
Question:	Is there a path from s to t in G?

Before we start the description of the proof, we first fix some notation for matrices and vectors. By $A[i, j]$ we refer to the entry in the i-th row and j-th column of a matrix A. Similarly, $x[i]$ denotes the i-th entry of vector x. By $e_i^{(m)}$ we denote the m-dimensional unit (column) vector e with $e[i] = 1$ and $e[j] = 0$ for $j \neq i$. We write $x^\top$ if we use vector x as a row vector. The rank and the determinant of A are denoted by $\det(A)$ and $\text{rank}(A)$. For a prime number p, we denote by $\text{rank}_p(A)$ the rank of A as a matrix over $\mathbb{Z}_p$ (and with entries adjusted modulo p).

The following algorithmic problem will play an important role in the proof.

Problem:	FULLMATRIXRANK
Input:	$(m \times m)$-matrix A with values from $\{0, \ldots, m\}$
Question:	Is $\text{rank}(A) = m$?

The proof consists of three relatively simple steps, all of which build to some extent on previous work.

(1) s-t-REACH can be reduced to FULLMATRIXRANK by a bounded first-order reduction (and since DYNFO$(+,\times)$ is closed under such reductions, FULLMATRIXRANK $\in$ DYNFO$(+,\times)$ implies REACH $\in$ DYNFO$(+,\times)$).
(2) FULLMATRIXRANK $\in$ DYNFO$(+,\times)$.
(3) For every domain independent query Q, if $Q \in$ DYNFO$(+,\times)$ then $Q \in$ DYNFO (and therefore s-t-REACH $\in$ DYNFO, since s-t-REACH is domain independent).

Here, a query Q is *domain independent*, if $Q(\mathcal{D}_1) = Q(\mathcal{D}_2)$ for all databases $\mathcal{D}_1$ and $\mathcal{D}_2$ that coincide in all relations and constants (but may differ in the underlying domain).

The first step is similar in spirit to reductions in [Cook 1985; Laubner 2011]. The algorithm constructed for step (2) adapts a dynamic sequential algorithm for maintaining rank from [Frandsen and Frandsen 2009]. The third step extends the technique for maintaining arithmetic presented in [Etessami 1998].

In the following we describe the three steps separately and largely self-contained.

4.1. From Reachability to Matrix Rank

Towards the reduction from s-t-reachability to matrix rank, let G be a graph with n vertices and A_G its adjacency matrix, and let s, t be vertices of G. The important observation (which can be found, e.g., in [Horn and Johnson 2012, Theorem 6.1.10.]) is that $I - \frac{1}{n}A_G$ is invertible and

$$(I - \frac{1}{n}A_G)^{-1} = I + \sum_{i=1}^{\infty} (\frac{1}{n}A_G)^i.$$

A closer inspection of the sum on the right hand side reveals that this matrix has a non-zero entry at position (s, t) if and only if t is reachable from s. Thus, the same holds for the inverse of $I - \frac{1}{n}A_G$.

For technical reasons, we prefer to deal with integer matrices and therefore rather work with the matrix $B \stackrel{\text{def}}{=} nI - A_G$, which is also invertible. We denote by B^{+s} the $((n+1) \times n)$-matrix that is obtained by extending B by an additional row $(e_s^{(n)})^\top$ and by B^{+st} the extension of B^{+s} by the additional column vector $e_t^{(n+1)}$.

Then the following chain of equivalences holds.

$$\begin{aligned}
t \text{ is reachable from } s &\iff (B^{-1})[s,t] \neq 0 \\
&\iff (B^{-1}e_t^{(n)})[s] \neq 0 \\
&\iff \text{the equation } Bx = e_t^{(n)} \text{ has } \textit{no} \text{ solution vector } x \text{ with } x[s] = 0 \\
&\iff \text{the system } \begin{array}{rcl} Bx & = & e_t^{(n)} \\ (e_s^{(n)})^\top x & = & 0 \end{array} \text{ has no solution vector } x \text{ at all} \\
&\iff e_t^{(n+1)} \text{ is not in the column space of } B^{+s} \\
&\iff B^{+st} \text{ has rank } n+1
\end{aligned}$$

The latter equivalence holds since B is invertible, and thus B and B^{+s} have rank n.

We next describe, how the above equivalence gives rise to a bounded first-order reduction from s-t-REACH to FULLMATRIXRANK. We first observe that , for graphs with n vertices the resulting matrix has only entries in $\{0, \ldots, n\}$. We can therefore represent inputs for FULLMATRIXRANK as follows by finite structures. A matrix with m rows and columns and values in $\{0, \ldots, m\}$ is represented by a structure with universe of size $m + 1$, a linear order $<$ and a ternary relation A. Thanks to $<$, the elements can be identified with the numbers $0, 1, \ldots, m$ in a natural way. A tuple $(i, j, k) \in A$ indicates that the (i, j)-entry of A is k. If there is no (i, j, k)-tuple for some i, j, then the (i, j)-entry has the value 0. We only consider changes of A that leave A in a semantically meaningful way, that is, for every i, j there is at most one k such that $(i, j, k) \in A$. We note that changes in B^{+st} that are triggered by single edge changes in G can be applied in a way that ensures this property.

It is now easy to check that, in the presence of a linear order[9], B^{+st} can be obtained from G by a first-order reduction that has the additional property that each entry in B^{+st} depends on at most one edge of G. Hence, the reduction is a bounded-first-order reduction [Patnaik and Immerman 1997] and therefore reachability can be maintained in DYNFO$(+,\times)$ if the problem FULLMATRIXRANK can be maintained in DYNFO$(+,\times)$.

[9]We can assume that the reduction has a linear order available, since it is applied in the context of DYNFO$(+,\times)$.

4.2. Matrix rank in DYNFO$(+,\times)$

Next, we show the most important intermediate result for Theorem 4.1, which is interesting also in its own right.

PROPOSITION 4.2. FULLMATRIXRANK $\in$ DYNFO$(+,\times)$.

PROOF SKETCH. We give an informal description of the dynamic algorithm for FULLMATRIXRANK. It can be easily transformed into a DYNFO$(+,\times)$ program. In the following we fix $m > 0$ and only consider $(m \times m)$-matrices with entries from $\{0, \ldots, m\}$.

We first show that in order to maintain whether $\mathrm{rank}(A) = m$ it suffices to maintain $\mathrm{rank}_p(A)$ for sufficiently many small primes p. Clearly, $\mathrm{rank}(A) = m$ if and only if $\det(A) \neq 0$. Since $\det(A)$ is bounded by $m!m^m$, its binary representation has $\mathcal{O}(m \log m)$ digits (for sufficiently large m). It follows with the help of the Prime Number Theorem[10] that $\det(A) \neq 0$ if and only if there exists a prime $p \leq m^2$ such that $\det(A) \not\equiv 0 (\bmod p)$.

Therefore, for large enough m, $\mathrm{rank}(A) = m$ if and only if there exists a prime $p \leq m^2$ such that $\mathrm{rank}_p(A) = m$.

In the following, we describe how $\mathrm{rank}_p(A)$ can be maintained, for a prime p. To this end we adapt a dynamic algorithm that has been stated in [Frandsen and Frandsen 2009]. The idea is to maintain an invertible matrix U and a matrix E in reduced row-echelon form such that $UA = E$. That E is in reduced row-echelon form means that

— the *leading entry*, i.e., the left-most non-zero entry, in every row is 1,
— the column of such a leading entry is all-zero otherwise, and
— rows are sorted in a "diagonal" fashion, that is, for larger row numbers, the column numbers of leading entries strictly increase.

Thanks to $\mathrm{rank}(E) = \mathrm{rank}(UA) = \mathrm{rank}(A)$ and the structure of E, it holds that $\mathrm{rank}(A)$ equals the number of non-zero rows of E.

We describe next, how this information can be maintained after a change of $A[i,j]$, for any $i, j \leq m$. Let A' denote the new value of matrix A after this change. We explain next, how new matrices U' and E' can be obtained such that $U'A' = E'$.

After a change of $A[i,j]$, UA' differs from UA at most in column j. Thus, to get the desired matrix E' in reduced echelon form, we can proceed as follows.

(1) If column j has more than one leading entry of UA':
   — let the entry with the maximum number of successive zeros in its row (uniquely determined) be the new leading entry,
   — set this leading entry to 1, and set all other entries of column j to 0 by appropriate row operations.
(2) If a former leading entry of a row k is lost in column j (by the change in A or by step (1)),
   — set its new leading entry (i.e., the next non-zero entry in row k and some column $\ell > j$) to 1 and set all other entries of column ℓ to 0 by appropriate row operations.[11]
(3) If needed: move the (at most two) rows, for which the position of the leading entry has changed (compared with E) to their correct positions.

An illustrating example can be found in Figure 2. The row operations mentioned above are done by suitably adapting U. Each of the three steps can be performed in constant

[10] The Prime Number Theorem yields about N primes between 1 and $N \log N$.

[11] Since all other columns with leading entries have only one non-zero entry, and row k has no non-zero entries before column ℓ, these row operations do not do any harm to the echelon structure of the rest of the matrix.

parallel time and therefore by a DYNFO$(+,\times)$ update program $\mathcal{P}_p$. It remains to combine the DYNFO$(+,\times)$ programs $\mathcal{P}_p$, for each prime $p \leq m^2$ into one update program that performs all computations simultaneously. To this end, for each relation R that is k-ary in each $\mathcal{P}_p$ there is a $(k+2)$-ary relation in the full program. Each tuple comes with two additional entries p_1, p_2, representing a prime $((p_1 - 1) \times m + p_2)$. The program can easily determine which pairs (p_1, p_2) represent actual primes, since it can use arithmetic right from the start. □

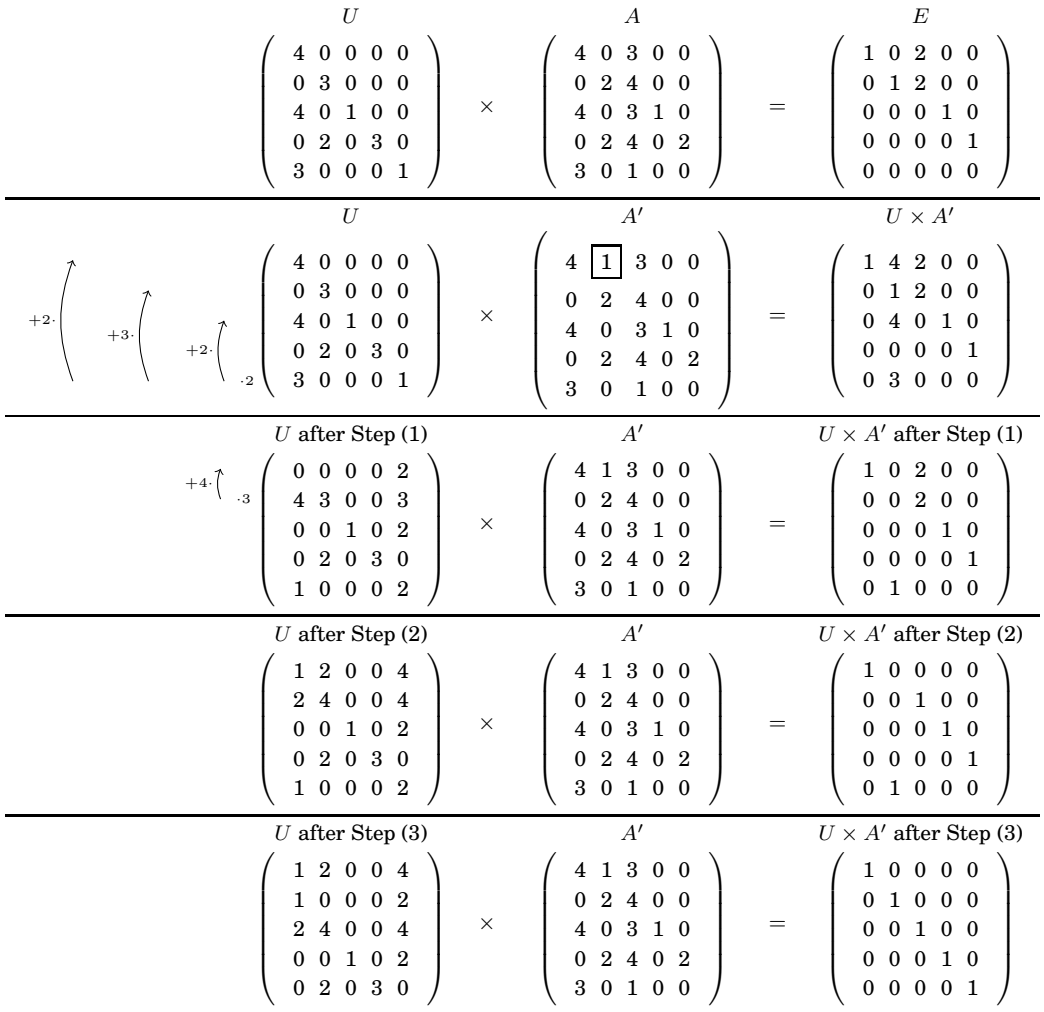

Fig. 2. Illustration of the modifications necessary for one change in matrix A for $p = 5$.

4.3. DYNFO$(+,\times)$ vs. DYNFO

Since, s-t-REACH is easily seen to be domain independent, the proof sketch for Theorem 4.1 can be completed by a proof sketch for the following result.

PROPOSITION 4.3. *If a query $Q \in$ DYNFO$(+,\times)$ is domain-independent, then $Q \in$ DYNFO.*

Etessami already observed that DYNFO programs have the same expressive power as DYNFO$(+,\times)$ programs, if, before the actual change sequence starts, for each element u of the universe, the edge (u,u) is inserted and afterwards deleted [Etessami 1998]. He described how these preliminary changes can be used to construct a linear order and compatible $+$ and $\times$ predicates on the whole universe. He also observed that, alternatively, arithmetic can be defined incrementally, so that at any point there are relations $<_{ad}$, $+_{ad}$ and $\times_{ad}$ that represent a linear order *on the activated domain*, and corresponding ternary addition and multiplication relations, respectively. We show next that by a refined technique DYNFO programs can simulate DYNFO$(+,\times)$ programs for domain-independent queries without any form of preprocessing.

PROOF (SKETCH). Let Q be a domain-independent query and $\mathcal{P}$ a DYNFO$(+,\times)$ program that maintains Q. For simplicity, we assume that Q uses only one binary relation E, the adaptation for arbitrary structures is straightforward. We recall that change sequences are applied to an initially empty structure, but that $\mathcal{P}$ uses non-empty initial relations that provide a linear order and the corresponding addition and multiplication relations on the full universe.

We will construct a DYNFO program $\mathcal{P}'$ that simulates $\mathcal{P}$. By definition of DYNFO, $\mathcal{P}'$ has to maintain Q under change sequences from an initially empty structure (just as $\mathcal{P}$) but with initially empty auxiliary relations (unlike $\mathcal{P}$). The challenge is therefore that $\mathcal{P}'$ cannot simply simulate $\mathcal{P}$ right from the beginning of the change sequence, as it does not have $<$, $+$ and $\times$ available.

We say that an element u of the universe has been *activated* by a change sequence $\alpha = \delta_1, \ldots, \delta_\ell$, if u occurs in some δ_i, no matter, whether an edge with u is still present after the whole sequence α. We refer to the set of activated elements by A. The update program $\mathcal{P}'$ maintains a linear order $<$ on A. Whenever new elements are activated by the insertion of a tuple t, the linear order is extended by these elements in a straightforward fashion. The relative order of the newly activated elements is determined by their position in t. Also an addition relation and a multiplication relation on A is maintained, just as in [Etessami 1998]. Thanks to the linear order, we can assume that A is always of the form $[m]$, for some number m, consistent with $<$.

The basic idea for the construction of $\mathcal{P}'$ is to split computations of $\mathcal{P}$ into phases, based on the size of A, and to let $\mathcal{P}'$ use different simulations for the different phases of $\mathcal{P}$. More precisely, we say that a computation of $\mathcal{P}$ on a universe U of size n is in phase $i < \sqrt{n}+1$, if more than $(i-1)^2$ but at most i^2 elements of U are activated. The update program $\mathcal{P}'$ uses one simulation per phase of $\mathcal{P}$ and we refer to the simulation that is responsible for phase i as the i-simulation.

For each i, the i-simulation begins as soon as i elements are activated.[12] The i-simulation stops as soon as A contains more than i^2 elements. For each i, the query result of $\mathcal{P}'$ is the query result of the simulation that is responsible for the current phase. That is, the query result is provided by the i-simulation if more than $(i-1)^2$ and at most i^2 elements are activated (in the simulated run of $\mathcal{P}$).

When the i-simulation starts, a linear order, an addition relation and a multiplication relation over $[i]$ are available. From these relations a linear order, an addition relation and a multiplication relation on pairs over $[i]$ can be easily defined in first-order logic.[13] The i-simulation uses the set of pairs over $[i]$ as universe of size i^2. It

[12] We note that a simulation need not start at a phase border. But each simulation is responsible for one phase.

[13] Technically, the addition relation over $[m]$ is a 6-ary relation, a tuple $(u_1,u_2,v_1,v_2,w_1,w_2)$ of which is interpreted as $(u_1,u_2)+(v_1,v_2)=(w_1,w_2)$.

maintains a bijection[14] g_i between the activated elements of the structure with respect to the computation of $\mathcal{P}$ and the pairs over $[i]^2$, that respects the linear order. At the start of the i-simulation, $g_i(k) = (0, k)$, for every $k \in [i]$.

The actual i-simulation starts on the structure over $[i]^2$ with empty input relations and with the linear order and the corresponding addition and multiplication relations over $[i]^2$. It uses one $2k$-ary auxiliary relation, for every k-ary auxiliary relation of $\mathcal{P}$, and a 4-ary auxiliary relation E' that corresponds to E.

As already stated, the input relation of the i-simulation is empty, when the simulation starts. However, at this point the "real" input relation E might have up to i^2 tuples over $[i]$, to which we refer as the *old tuples*. For each change operation δ, $\mathcal{P}'$ inserts up to four tuples to E' and simulates $\mathcal{P}$ for these four insertion steps. If δ is a deletion, then one tuple might be deleted. More precisely, for each occurring change operation $\mathcal{P}'$ determines the four lexicographically smallest tuples $t_1, \ldots, t_4$ in E, whose image under g_i is not yet in E', inserts $g_i(t_1,) \ldots, g_i(t_4)$ to E' and simulates the induced behavior of $\mathcal{P}$ on the auxiliary relations. If the change operation deletes some tuple t from E and it holds $g_i(t) \in E'$, then $g_i(t)$ is deleted from E' and the corresponding update operations are performed. It is easy to see that after $\frac{i^2}{2}$ change operations, E' will coincide with $g_i(E)$ and from this point on, the i-simulation only needs to simulate the actual current change operation. This happens before phase i starts and therefore, the query result provided by the i-simulation during phase i (translated via g_i^{-1}) will be always correct.[15]

This completes the description of the i-simulations, for each i. Of course, it is not possible to let each simulation use its own set of auxiliary relations. However, we can simply increase the arity of each relation symbol by one and use the new entry to indicate, for each tuple, the number of the simulation, for which it is used. □

4.4. Some consequences of REACH ∈ DYNFO

From Theorem 4.1, one can infer that some other queries are in DYNFO by relatively straightforward bounded first-order reductions.

As a first example we consider regular path queries (RPQs) on graph databases. A graph database is basically a directed graph with edge labels from a finite[16] alphabet. A regular path query Q is just a regular expression over label names. It yields all pairs (u, v) of a graph database for which there is a path from u to v whose sequence of labels is in the language specified by Q. The problem of maintaining such a query can be easily reduced to REACH as follows. Let A be an NFA for the language of Q with initial state q_0 and unique accepting state q_f. Then one can construct, given a graph database G, the synchronized product $G \times A$ of G and A; and $(u, v) \in Q(G)$ holds if and only if (v, q_f) is reachable from (u, q_0) in (the unlabelled graph) $G \times A$. Since each single change in G only induces at most $|A|$ changes in $G \times A$, the reduction is bounded and therefore, the maintainability of RPQs follows from Theorem 4.1. This easily transfers to conjunctions of regular path queries, CRPQs. We note that further classes of query languages for labeled graphs have been studied in the literature [Muñoz et al. 2016; Weber and Schwentick 2007].

By a standard reduction to reachability it also follows that 2-SAT is in DYNFO.

However, the consequences of Theorem 4.1 are not fully understood yet. One might be tempted to expect that DYNFO can maintain all queries Q that can be expressed by

[14]Technically, we can think of g_i as a ternary relation at this point.

[15]For careful readers we note that for very small values of i this has to be slightly adapted.

[16]The set of labels actually needs not be fixed a priori. However, given a regular expression r, only labels that occur in r are relevant for maintaining r and all other labels can be replaced by some fixed label X not occurring in r.

a formula with a unary transitive closure operator in front of a first-order formula φ. However, even though in this case φ basically specifies a first-order reduction from Q to REACH, this reduction does not need to be bounded and therefore maintainability in DYNFO does not immediately follow.

5. INEXPRESSIBILITY RESULTS

Many queries are thus in DYNFO. It is also desirable to learn what queries are *not* in DYNFO. As mentioned above, each query in DYNFO can be evaluated in polynomial time, but we do not expect that all polynomial-time queries are in DYNFO.

There are many standard methods available that can be used to prove that a query at hand is not expressible in first-order logic [Libkin 2004]. However, explicit lower bound results for FO$(+,\times)$ are much harder to achieve [Arora and Barak 2009] and there are no results of this kind for the extension of FO$(+,\times)$ by arbitrary modulo counting quantifiers. Proposition 4.3 can be interpreted as a statement that lower bounds for DYNFO are at least at hard as for FO$(+,\times)$. Indeed two prime examples of queries that cannot be expressed in first-order logic, reachability and parity, can be maintained in DYNFO. Thus the two notorious weaknesses of first-order logic, locality and the inability to count globally, are not shared by DYNFO— and thus cannot be used to prove inexpressibility results.

With current methods, explicit inexpressibility results for DYNFO seem out if reach. It is thus natural to develop such methods first for fragments of DYNFO. In this section, we give examples of such inexpressibility results for a fragment with auxiliary relations of bounded arity and for the fragment DYNPROP in which update formulas are quantifier-free.

The first result from [Dong and Su 1998] shows that the reachability query cannot be maintained with only unary auxiliary relations. Since the definition of DYNFO requires that the query result is always represented in one distinguished auxiliary relation "only unary auxiliary relations" means that all auxiliary relations *besides this one binary auxiliary relation* are unary. The result therefore also subsumes the result mentioned in the introduction, that reachability cannot be maintained without auxiliary relations (besides the query relation).

THEOREM 5.1 ([DONG AND SU 1998]). *Reachability cannot be maintained in* DYNFO *with unary auxiliary relations.*

PROOF. Towards a contradiction assume that there is a DYNFO-program $\mathcal{P}$ that maintains Reachability with one binary auxiliary relation T for storing the query result and otherwise only m unary auxiliary relations. We assume that the update rule for T for deletions is

on delete (u,v) **from** E
update $T(x,y)$ **as** $\varphi(u,v,x,y)$

for some formula φ of some quantifier-depth k. Let n be sufficiently large with respect to k and m. Let $G=(V,E)$ be a graph on n vertices which forms a directed cycle. Let $\mathcal{S}$ be the state of $\mathcal{P}$ after insertion of the edges of E in some order (see Figure 3).

Since the transitive closure of E is $V\times V$, φ is equivalent to the formulas φ' resulting from φ by replacing all occurrences of T-atoms by $\top$. Clearly, φ' only refers to E and the auxiliary relations. Since n was chosen sufficiently large, there are three disjoint paths P_1, P_2 and P_3 in C of length 2^{k+1} that are isomorphic in $\mathcal{S}$. Let (a_1,b_1), (a_2,b_2) and (a_3,b_3) be the innermost edges of P_1, P_2 and P_3, and assume that they occur in this cyclic order. When deleting the edge (a_2,b_2), there is still a path from a_3 to a_1 but no path from a_1 to a_3. Yet $(\mathcal{S},a_2,b_2,a_1,a_3)\models\varphi'$ if and only if $(\mathcal{S},a_2,b_2,a_3,a_1)\models\varphi'$ by Gaifman's Theorem [Immerman 1999]. This is the desired contradiction. □

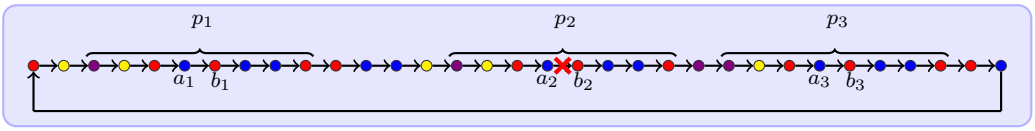

Fig. 3. Illustration of the construction in Theorem 5.1. The colors indicate the content of the unary relations.

This proof has a very "static flavor". Starting from some state of a certain shape, a contradiction is obtained by considering only one change operation and the proof argument can therefore use a standard inexpressibility method for first-order logic, locality. "Static lower bound methods" have been used in several dynamic inexpressibility results. Along similar lines as for Theorem 5.1 one can prove, e.g., that local update languages (possibly stronger than FO) cannot maintain Reachability with unary relations only [Dong et al. 2003], that there is a context-free language that is not in DYNFO with unary relations [Vortmeier 2013], and that the equal cardinality of two unary relations cannot be expressed in DYNFO in a setting with weak initializations [Grädel and Siebertz 2012]. A further lower bound from static complexity that has been reused in dynamic complexity is a theorem of Cai, stating that m parity functions cannot be computed in AC^0 using $m-1$ help bits [Cai 1990]. It has been used by Miltersen to obtain an arity hierarchy for DYNFO by exploiting a correspondence between help bits and auxiliary bits [17]. Unfortunately, "static methods" seem not sufficient (as discussed above) to prove very strong dynamic inexpressibility results. It thus seems necessary to develop more "dynamic methods" to prove inexpressibility for non-unary fragments of DYNFO.

We next sketch such a method for DYNPROP, the quantifier-free fragment of DYNFO. We give two applications, one of which yields a query that cannot be maintained in DYNPROP at all, and the other shows that REACH cannot be maintained by DYNPROP programs with binary auxiliary relations, the currently best lower bound for REACH with respect to arity.

The method uses a simple locality property of DYNPROP programs. When updating an auxiliary tuple $\vec{d}$ after a change of an edge $\vec{e}$, a quantifier-free update formula only has access to $\vec{d}$, $\vec{e}$, and the constants of the input structure. Thus, if a change operation changes a tuple *inside* a substructure $\mathcal{A}$ of a state $\mathcal{S}$, the auxiliary data of $\mathcal{A}$ is not affected by any information from *outside* of $\mathcal{A}$. In particular, two isomorphic substructures $\mathcal{A}$ and $\mathcal{B}$ remain isomorphic, when corresponding changes are applied to them.

The notion of corresponding changes is formalized as follows. Let π be an isomorphism from $\mathcal{A}$ to $\mathcal{B}$. Two changes $\delta(\vec{a})$ on $\mathcal{A}$ and $\delta'(\vec{b})$ on $\mathcal{B}$ are said to be *π-respecting* if $\delta = \delta'$ and $\vec{b} = \pi(\vec{a})$. Two sequences $\alpha = \delta_1 \cdots \delta_m$ and $\beta = \delta'_1 \cdots \delta'_m$ of changes respect π if δ_i and δ'_i are π-respecting for every $i \leq m$. We recall that $P_\alpha(\mathcal{S})$ denotes the state obtained by executing the dynamic program $\mathcal{P}$ for the change sequence α from state $\mathcal{S}$.

LEMMA 5.2 ([GELADE ET AL. 2012; ZEUME AND SCHWENTICK 2015]). *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 change sequences α, β on A and B.*

Here, $\mathcal{S} \restriction A$ denote the substructure of $\mathcal{S}$ that is induced[18] by A.

[17] [Dong and Su 1998] attributes the result to Miltersen.
[18] We note that induced substructures contain all constants of the input structure.

We refer to this lemma as the *Substructure Lemma*.

The Substructure Lemma can be applied along the following lines to prove that a (graph) query $\mathcal{Q}$ cannot be maintained in a setting with quantifier-free updates. Towards a contradiction, assume that there is a quantifier-free program $\mathcal{P}$ that maintains $\mathcal{Q}$. Then, find

— two states $\mathcal{S}$ and $\mathcal{T}$ of $\mathcal{P}$ for two graphs $G_{\mathcal{S}}$ and $G_{\mathcal{T}}$,
— substructures $\mathcal{S} \restriction A$ and $\mathcal{T} \restriction B$ of $\mathcal{S}$ and $\mathcal{T}$ isomorphic via π, and
— two π-respecting change sequences α and β on A and B,

such that there is a tuple $\vec{a}$ over A that is contained in the query result for the graph $\alpha(G_{\mathcal{S}})$ but $\pi(\vec{a})$ is not in the result of $\beta(G_{\mathcal{T}})$. This yields the desired contradiction, since either both or neither of the tuples $\vec{a}$ and $\pi(\vec{a})$ are contained in the distinguished query relations of $P_\alpha(\mathcal{S})$ and $P_\beta(\mathcal{T})$ due to the Substructure Lemma.

The first example shows that the Boolean alternating reachability query is not in DYNPROP. An *alternating graph* is represented by a binary edge relation E and a unary relation A of *universal* nodes. Given a node $t \in V$, the set of all *reachable* nodes $\text{Reach}(t)$ is defined as the smallest set satisfying[19]

— $t \in \text{Reach}(t)$;
— if $u \notin A$ and there is a $v \in \text{Reach}(t)$ such that $(u, v) \in E$, then $u \in \text{Reach}(t)$; and
— if $u \in A$ and for all $v \in V$ with $(u, v) \in E$, we have $v \in \text{Reach}(t)$, then $u \in \text{Reach}(t)$.

We consider the following P-complete problem (see, for example, [Vollmer 1999]).

Problem:	ALT-REACH
Input:	Alternating graph $G = (V, E, A)$ and two nodes s and t
Question:	Is $s \in \text{Reach}(t)$?

THEOREM 5.3 ([GELADE ET AL. 2012]). ALT-REACH $\notin$ DYNPROP.

PROOF. For each $m \in \mathbb{N}$, we define a graph $G_m = (V_m, E_m)$ (see Figure 4 for an illustration).

The vertex set V_m is the union of the following sets of nodes.

— $\{s, t\}$;
— a set P of $2m$ nodes $p_1, ..., p_{2m}$;
— a set Q, consisting of one node q_I, for each subset I of P with $|I| = m$; and
— a set R, consisting of one node r_J, for each subset J of Q.

The set A of universal nodes is just Q. The set E_m contains the following edges:

— (q_I, p), for each node q_I of Q and each $p \in I$; and
— (r_J, q), for each node r_J of R and each $q \in J$.

Towards a contradiction, we assume that there is a DYNPROP-program $\mathcal{P}$ for ALT-REACH with p auxiliary relations of maximal arity ℓ. Let m be chosen sufficiently large with respect to p and ℓ, and let A, P, Q, R be the sets of vertices of G_m as defined above.

We note that $|Q| = \binom{2m}{m} \geq 2^m$, for large enough m, and therefore $|R| = 2^{|Q|} \geq 2^{2^m}$.

For each $r \in R$, let U_r be the set $\{s, t, r\} \cup P$ of size $2m + 3$, and let O_r be the linear order on U_r defined by $s < t < r < p_1 < \cdots < p_{2m}$. The number of different structures

[19] We note that this is a backward reachability.

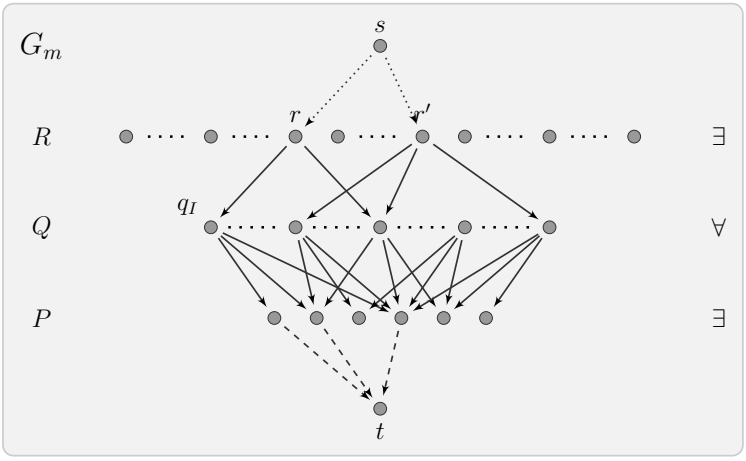

Fig. 4. An illustration of the graph G_m. Full edges represent edges from G_m; G_m^r and $G_m^{r'}$ result from G_m by inserting the dashed edges and one of the dotted edges, respectively.

(with respect to isomorphism) with a domain of size $2m+3$, a linear order and at most p relations of maximal arity $\ell \geq 2$ is bounded by $2^{(p+1)\cdot(2m+3)^\ell}$, which is strictly smaller than 2^{2^m}, for sufficiently large m.

Let $\mathcal{S}$ be the state of $\mathcal{P}$ after insertion of all edges of G_m. Due the above reasoning, since $|R| \geq 2^{2^m}$, there must be two different nodes $r, r' \in R$ for which $\mathcal{S} \restriction U_r$ and $\mathcal{S} \restriction U_{r'}$ are isomorphic under the isomorphism which maps r to r' and is the identity otherwise.[20]

Let $J, J' \subseteq Q$ with $r = r_J$ and $r' = r_{J'}$. Since $J \neq J'$ we can assume without loss of generality that there is a set $I = \{p_{i_1}, ..., p_{i_m}\}$ such that $q_I \in J$ but $q_I \notin J'$. Let G_m^r be the graph resulting from inserting the edges (s, r) and $(p_{i_1}, t), \ldots (p_{i_m}, t)$ into G_m and $G_m^{r'}$ be the graph resulting from inserting the edges (s, r') and $(p_{i_1}, t), \ldots (p_{i_m}, t)$ into G_m. Let $\mathcal{S}^r$ and $\mathcal{S}^{r'}$ denote the states of $\mathcal{P}$ after insertion of these edges, respectively. By the semantics of alternating reachability, $s \in \text{Reach}(t)$ with respect to G_m^r, but $s \notin \text{Reach}(t)$ with respect to $G_m^{r'}$. However, the Substructure Lemma guarantees that $\mathcal{S}^r \restriction U_r$ and $\mathcal{S}^{R'} \restriction U_{r'}$ are isomorphic and therefore, the update formula for the query relation of $\mathcal{P}$ yields the same query result in both states, the desired contradiction. The construction is illustrated by Figure 4. □

The proof of the following result is similar in spirit.[21]

THEOREM 5.4 ([ZEUME AND SCHWENTICK 2015]). *Reachability cannot be maintained in binary* DYNPROP.

PROOF. Towards a contradiction assume that a dynamic program $\mathcal{P}$ with quantifier-free update formulas and binary auxiliary schema maintains the dynamic s-t-reachability query. We choose numbers n and n' such that n' is sufficiently large with respect to the size of the auxiliary schema and n is sufficiently large with respect to n'. Consider the graph $G \stackrel{\text{def}}{=} (V, E)$ defined as follows. Let $V \stackrel{\text{def}}{=} \{s, t\} \uplus A \uplus B$ where B is a set of size n and A is of size 2^n. We associate with every subset $X \subseteq B$ a unique vertex

[20] The orders O_r are only needed to make sure that the isomorphism works in exactly this way.
[21] Its presentation closely follows [Zeume and Schwentick 2015].

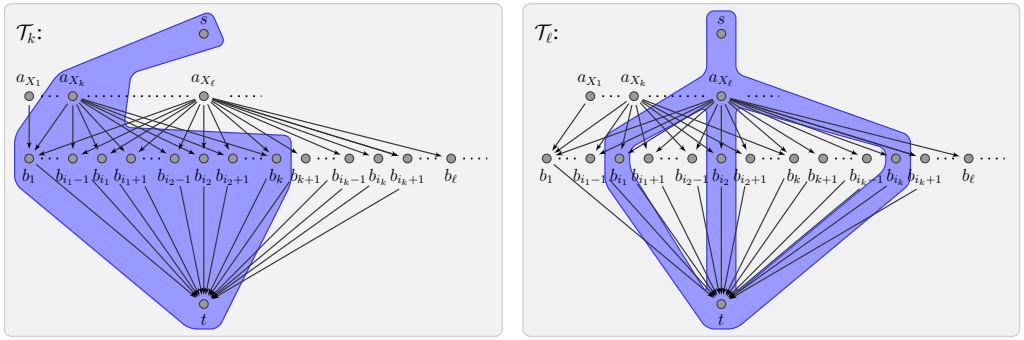

Fig. 5. The structure $\mathcal{S}'$ from the proof of Theorem 5.4 with highlighted isomorphic substructures $\mathcal{T}_k$ and $\mathcal{T}_\ell$.

a_X from A in an arbitrary fashion. The graph has an edge (b, t) for all $b \in B$ as well as an edge (a_X, b) for every subset X of B and every $b \in X$.

Let $\mathcal{S}$ be a state of $\mathcal{P}$ with G as underlying structure. Our plan is to find two sets X, X' such that $X \subsetneq X' \subseteq B$ and the restriction of $\mathcal{S}$ to $\{s, t, a_{X'}\} \cup X'$ contains an isomorphic copy of $\mathcal{S}$ restricted to $\{s, t, a_X\} \cup X$. Then the Substructure Lemma will easily give us a contradiction as follows. Consider the two change sequences β_1 and β_2:

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

Applying β_1 to G yields a graph in which t is not reachable from s, whereas applying β_2 yields a graph in which t is reachable from s. Yet, the substructures induced by $\{s, t, a_{X'}\} \cup X'$ and $\{s, t, a_X\} \cup X$ in the corresponding states are still isomorphic due to the Substructure Lemma. This is a contradiction.

For finding the sets X and X' we employ two well-known combinatorial techniques; a Ramsey-like argument and Higman's Lemma.

Let $\prec$ be an order on B. Since $|B|$ is large with respect to n', a Ramsey-like theorem for structures can be employed to obtain a subset $B' \subseteq B$ of size n' such that all tuples $(b_1, b_2) \in B' \times B'$ with $b_1 \prec b_2$ have the same type in $\mathcal{S}$. Let $b_1 \prec \ldots \prec b_{n'}$ be an enumeration of the elements of B' and let $X_i \stackrel{\text{def}}{=} \{b_1, \ldots, b_i\}$, for every $i \in \{1, \ldots, n'\}$.

Let $\mathcal{S}_i$ denote the restriction of $\mathcal{S}$ to $X_i \cup \{s, t, a_{X_i}\}$. For every i, we construct a word w_i of length i that has a letter for every node in X_i and captures all relevant information about those nodes in $\mathcal{S}_i$. More precisely, $w_i \stackrel{\text{def}}{=} \sigma_i^1 \cdots \sigma_i^i$, where for every i and j, σ_i^j is the binary atomic type of (a_{X_i}, b_j).

Since B' is sufficiently large with respect to the size of the auxiliary schema, Higman's Lemma can be employed to find k and ℓ such that $k < \ell$ and w_k is a subsequence of w_ℓ, that is $w_k = \sigma_k^1 \sigma_k^2 \ldots \sigma_k^k = \sigma_\ell^{i_1} \sigma_\ell^{i_2} \ldots \sigma_\ell^{i_k}$ for suitable numbers $i_1 < \ldots < i_k$. Define $\mathcal{T}_k \stackrel{\text{def}}{=} \mathcal{S}'_k \restriction T_k$ where $T_k = \{s, t, a_{X_k}, b_1, \ldots, b_k\}$ and $\mathcal{T}_\ell \stackrel{\text{def}}{=} \mathcal{S}'_\ell \restriction T_\ell$ where $T_\ell \stackrel{\text{def}}{=} \{s, t, a_{X_\ell}, b_{i_1}, \ldots, b_{i_k}\}$. We refer to Figure 5 for an illustration of the substructures $\mathcal{T}_k$ and $\mathcal{T}_\ell$ of $\mathcal{S}'$. It is straightforward to show that $\mathcal{T}_k \simeq_\pi \mathcal{T}_\ell$, where π is the isomorphism that maps s and t to themselves, a_{X_k} to a_{X_ℓ} and b_j to b_{i_j} for every $j \in \{1, \ldots, k\}$.

Thus X and X' can be chosen as $\{b_1, \ldots, b_k\}$ and $\{b_{i_1}, \ldots, b_{i_k}\}$, respectively. □

For inexpressibility proofs with the Substructure Lemma, the challenge is to find well-behaved isomorphic structures. Several combinatorial techniques have been employed to find such structures. A similar counting argument as in the proof of Theorem 5.3 has been used to show that Reachability is not in DYNPROP for a restricted initialization in [Zeume and Schwentick 2015]. We have seen an application of com-

bining the Substructure Lemma with Ramsey's Theorem and Higman's Lemma in the proof that Reachability is not in binary DYNPROP. Finally, in [Zeume 2014], upper and lower bounds for Ramsey numbers have been used to establish an arity hierarchy for DYNPROP for queries on graphs under insertions, by showing that $(k+2)$-Clique is not in k-ary DYNPROP. We conjecture that the same approach can be used to prove an arity hierarchy for DYNPROP for graph queries (under insertions *and* deletions).

For very small fragments also some purely dynamic ad-hoc proof techniques have been used [Hesse 2003b; Zeume and Schwentick 2014].

The biggest future challenge for determining the power of dynamic programs is to come up with new tools and methods for proving inexpressibility for larger fragments of DYNFO. The potential of the two approaches described above, the reuse of static techniques and the Substructure Lemma, seems to be exhausted. For instance, even though an extension of the Substructure Lemma can be employed to prove inexpressibility for the slight extension of DYNPROP with unary auxiliary functions, it has been argued in [Zeume 2014] that inexpressibility proofs for binary functions require new ideas. Several alternative approaches for proving non-expressibility for larger fragments of DYNFO, for instance Ehrenfeucht-Fraissé games and diagonalization-based approaches, have been explored by various researchers. Unfortunately the lower bounds proved with those approaches are not very strong so far. Yet we hope they yield better lower bounds in the future. We refer to [Zeume 2015] for a more detailed discussion. In [Grädel and Siebertz 2012] a combination of locality and dynamic techniques has been used to show inexpressibility results in a setting with arbitrary initial input structures and restricted initialization of auxiliary structures.

6. SOME FURTHER TOPICS IN DYNAMIC COMPLEXITY

As mentioned in the introduction, this article is not meant as a comprehensive survey. However, in this section, we mention a few other research directions with an emphasis on such directions to which the authors of this article contributed.

Maintainability Results. Although the reachability query has been the main object of study in Dynamic Complexity, other graph queries have been investigated as well. Previous work on undirected graph reachability lead to dynamic programs for spanning forests, 2-colorability, and the binary distance query in undirected graphs [Dong and Su 1998; Grädel and Siebertz 2012; Patnaik and Immerman 1997]. Tree isomorphism can be maintained with first-order updates as well [Etessami 1998].

The maintainability of formal languages has been studied as well. Already Patnaik and Immerman observed that regular languages and Dyck languages can be maintained[22] in DYNFO [Patnaik and Immerman 1997]. Hesse showed that the regular languages can be even maintained in DYNQF [Hesse 2003b]. In [Gelade et al. 2012] the dynamic complexity of regular language was pinpointed exactly: a formal language can be maintained in DYNPROP if and only if it is regular. We note that this is also an inexpressibility result, since it shows that every non-regular language is not in DYNPROP. They also showed that all context-free languages can be maintained in DYNFO; and that certain context-free languages can be maintained in DYNQF. The maintenance of path queries in labeled graphs is closely related.

The Fine Structure of DYNFO. As described in Section 3, various dynamic complexity classes can be obtained by varying the update formalism, the arity of the auxiliary relations and the initialization mechanism. Higher arity of auxiliary relations indeed increases the expressive power of dynamic programs with first-order updates

[22]We note that in the study of formal languages, the set of positions and its underlying order is fixed. The change operations can insert a symbol at a position or delete it (leaving the position empty).

[Dong and Su 1998] on arbitrary structures. For the quantifier-free fragment an arity hierarchy for graph queries has been established under insertions [Zeume 2014]. An extensive study of syntactical restrictions of first-order updates from a database theoretical point of view has been performed in [Zeume and Schwentick 2015]. This study considered, besides DynProp and DynQF, the following kinds of restrictions:

— Restrictions of the quantifier structure in update formulas: Dyn$\exists^*$FO and Dyn$\forall^*$FO;
— Restricted database query languages based on (unions of) conjunctive queries: DynCQ and DynUCQ;
— Their quantifier-free restrictions: DynPropCQ, DynPropUCQ;
— Classes resulting from Δ-semantics of update formulas, indicated by a prefix $\Delta-$.

Although one could have expected that these many different restrictions induce a complicated graph of relationships, it turned out that they are all neatly stacked in a linear hierarchy.

Dynamic versus Static Complexity Theory. Besides the relationship of dynamic complexity classes among each other, also the relationship to classical static complexity classes has been investigated, and several connections have been established [Etessami 1998; Patnaik and Immerman 1994]. All first-order definable properties can be maintained using existential first-order updates (though with FO-initialization only) [Zeume and Schwentick 2014] and all existential first-order properties are in DynQF. Further there are (artificial) PTIME-complete problems and natural LOGCFL-complete problems in DynFO [Patnaik and Immerman 1994; Weber and Schwentick 2007].

Analyzing dynamic programs, e.g. checking satisfiability of a dynamic program, is of course not easier than analyzing first-order formulas, and therefore undecidable in general. Many static analysis problems remain undecidable for very restricted dynamic programs [Schwentick et al. 2015].

7. PERSPECTIVES

Many questions remain open and we can only hint at some of them.

We have seen in Section 4 that the maintainability of the reachability query can be used to show other maintainability results. This might be possible for algorithmic problems from other areas as well, e.g., from Model Checking [Kähler and Wilke 2003], Program Analysis [Lev-Ami et al. 2009, 2007], or Knowledge Representation.

As mentioned before, every problem that is reducible to reachability under bounded first-order reductions can be maintained in DynFO. It is an interesting question to what extent the boundedness requirement can be weakened. This question is strongly related to the question, whether the reachability query can be maintained under more complex change operations that can change more than one tuple (or a constant number of tuples). Complex change operations were already discussed in [Patnaik and Immerman 1994], but a systematic study is still missing as of yet.

Complex change operations could help to make the algorithms found in Dynamic Complexity more applicable. Another step in that direction could be the development of a “dynamic programming language”.

Another line of research that suggests itself is which other algorithmic problems from Linear Algebra can be maintained in DynFO and in which ways they can be used to maintain other queries.

Last but not least, the machinery for inexpressibility proofs needs to be further developed. Proving that the reachability query is not in DynProp seems not completely out of reach [sic!].

Acknowledgements

We thank Samir Datta, Raghav Kulkarni and Anish Mukherjee for the excellent collaboration and Nils Vortmeier for many valuable suggestions for this article. We also thank Neil Immerman for the opportunity to describe the recent work in Dynamic Complexity in ACM SIGLOG News, for many helpful hints and, not last, for coming up with the nice title for this article. We acknowledge the financial support by DFG grant SCHW 678/6-1.

References

Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity - A Modern Approach*. Cambridge University Press.

David A. Mix Barrington, Neil Immerman, and Howard Straubing. 1990. On Uniformity within NC^1. *J. Comput. Syst. Sci.* 41, 3 (1990), 274–306. DOI:http://dx.doi.org/10.1016/0022-0000(90)90022-D

Jin-yi Cai. 1990. Lower Bounds for Constant-Depth Circuits in the Presence of Help Bits. *Inf. Process. Lett.* 36, 2 (1990), 79–83. DOI:http://dx.doi.org/10.1016/0020-0190(90)90101-3

Stephen A. Cook. 1985. A Taxonomy of Problems with Fast Parallel Algorithms. *Information and Control* 64, 1-3 (1985), 2–21. DOI:http://dx.doi.org/10.1016/S0019-9958(85)80041-3

Samir Datta, William Hesse, and Raghav Kulkarni. 2014. Dynamic Complexity of Directed Reachability and Other Problems. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Part I*. 356–367. DOI:http://dx.doi.org/10.1007/978-3-662-43948-7_30

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

Camil Demetrescu and Giuseppe F. Italiano. 2008. Mantaining Dynamic Matrices for Fully Dynamic Transitive Closure. *Algorithmica* 51, 4 (2008), 387–427. DOI:http://dx.doi.org/10.1007/s00453-007-9051-4

Guozhu Dong, Leonid Libkin, and Limsoon Wong. 2003. Incremental recomputation in local languages. *Inf. Comput.* 181, 2 (2003), 88–98. DOI:http://dx.doi.org/10.1016/S0890-5401(03)00017-8

Guozhu Dong and Jianwen Su. 1993. 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*. 295–308.

Guozhu Dong and Jianwen Su. 1998. Arity Bounds in First-Order Incremental Evaluation and Definition of Polynomial Time Database Queries. *J. Comput. Syst. Sci.* 57, 3 (1998), 289–308. DOI:http://dx.doi.org/10.1006/jcss.1998.1565

Guozhu Dong and Rodney W. Topor. 1992. Incremental Evaluation of Datalog Queries. In *Database Theory - ICDT'92, 4th International Conference*. 282–296. DOI:http://dx.doi.org/10.1007/3-540-56039-4_48

Kousha Etessami. 1998. Dynamic Tree Isomorphism via First-Order Updates. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 1998*. 235–243. DOI:http://dx.doi.org/10.1145/275487.275514

Gudmund Skovbjerg Frandsen and Peter Frands Frandsen. 2009. Dynamic matrix rank. *Theor. Comput. Sci.* 410, 41 (2009), 4085–4093. DOI:http://dx.doi.org/10.1016/j.tcs.2009.06.012

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

Erich Grädel and Sebastian Siebertz. 2012. Dynamic definability. In *15th International Conference on Database Theory, ICDT 2012,*. 236–248. DOI:http://dx.doi.org/10.1145/2274576.2274601

William Hesse. 2003a. The dynamic complexity of transitive closure is in DynTC^0. *Theor. Comput. Sci.* 296, 3 (2003), 473–485. DOI:http://dx.doi.org/10.1016/S0304-3975(02)00740-5

William Hesse. 2003b. *Dynamic Computational Complexity*. Ph.D. Dissertation. University of Massachusetts, Amherst.

Roger A Horn and Charles R Johnson. 2012. *Matrix analysis*. Cambridge university press.

Neil Immerman. 1999. *Descriptive Complexity*. Springer.

Detlef Kähler and Thomas Wilke. 2003. Program Complexity of Dynamic LTL Model Checking. In *Computer Science Logic, CSL 2003,*. 271–284. DOI:http://dx.doi.org/10.1007/978-3-540-45220-1_23

Bastian Laubner. 2011. *The structure of graphs and new logics for the characterization of Polynomial Time*. Ph.D. Dissertation. Humboldt University of Berlin. http://edoc.hu-berlin.de/dissertationen/laubner-bastian-2011-02-23/PDF/laubner.pdf

Tal Lev-Ami, Neil Immerman, Thomas W. Reps, Mooly Sagiv, Siddharth Srivastava, and Greta Yorsh. 2009. Simulating reachability using first-order logic with applications to verification of linked data structures. *Logical Methods in Computer Science* 5, 2 (2009). http://arxiv.org/abs/0904.4902

Tal Lev-Ami, Mooly Sagiv, Neil Immerman, and Thomas W. Reps. 2007. Constructing Specialized Shape Analyses for Uniform Change. In *Verification, Model Checking, and Abstract Interpretation, 8th International Conference, VMCAI 2007*. 215–233. DOI:http://dx.doi.org/10.1007/978-3-540-69738-1_16

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

Peter Bro Miltersen. 1999. Cell probe complexity-a survey. (1999). Invited talk/paper at Advances in Data Structures (Pre-conference workshop of FSTTCS'99), 1999. Never published, available at homepage.

Peter Bro Miltersen, Sairam Subramanian, Jeffrey Scott Vitter, and Roberto Tamassia. 1994. Complexity Models for Incremental Computation. *Theor. Comput. Sci.* 130, 1 (1994), 203–236. DOI:http://dx.doi.org/10.1016/0304-3975(94)90159-7

Pablo Muñoz, Nils Vortmeier, and Thomas Zeume. 2016. Dynamic graph queries. (2016). To appear in Proc. 19th International Conference on Database Theory (ICDT 2016).

Sushant Patnaik and Neil Immerman. 1994. Dyn-FO: A Parallel, Dynamic Complexity Class. In *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 1994*. 210–221. DOI:http://dx.doi.org/10.1145/182591.182614

Sushant Patnaik and Neil Immerman. 1997. Dyn-FO: A Parallel, Dynamic Complexity Class. *J. Comput. Syst. Sci.* 55, 2 (1997), 199–209. DOI:http://dx.doi.org/10.1006/jcss.1997.1520

Liam Roditty and Uri Zwick. 2008. Improved Dynamic Reachability Algorithms for Directed Graphs. *SIAM J. Comput.* 37, 5 (2008), 1455–1471. DOI:http://dx.doi.org/10.1137/060650271

Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. 2015. Static Analysis for Logic-based Dynamic Programs. In *24th EACSL Annual Conference on Computer Science Logic, CSL 2015*. 308–324.

DOI: http://dx.doi.org/10.4230/LIPIcs.CSL.2015.308

Heribert Vollmer. 1999. *Introduction to circuit complexity: a uniform approach*. Springer.

Nils Vortmeier. 2013. *Komplexitätstheorie verlaufsunabhängiger dynamischer Programme*. Master's thesis. TU Dortmund University.

Volker Weber and Thomas Schwentick. 2007. Dynamic Complexity Theory Revisited. *Theory Comput. Syst.* 40, 4 (2007), 355–377. DOI: http://dx.doi.org/10.1007/s00224-006-1312-0

Thomas Zeume. 2014. The Dynamic Descriptive Complexity of k-Clique. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014*. 547–558. DOI: http://dx.doi.org/10.1007/978-3-662-44522-8_46

Thomas Zeume. 2015. *Small Dynamic Complexity Classes*. Ph.D. Dissertation. TU Dortmund University.

Thomas Zeume and Thomas Schwentick. 2014. Dynamic Conjunctive Queries. In *Proc. 17th International Conference on Database Theory (ICDT 2014)*. 38–49. DOI: http://dx.doi.org/10.5441/002/icdt.2014.08

Thomas Zeume and Thomas Schwentick. 2015. On the quantifier-free dynamic complexity of Reachability. *Inf. Comput.* 240 (2015), 108–129. DOI: http://dx.doi.org/10.1016/j.ic.2014.09.011