Reachability Is in DynFO

SAMIR DATTA, Chennai Mathematical Institute & UMI ReLaX
RAGHAV KULKARNI and ANISH MUKHERJEE, Chennai Mathematical Institute
THOMAS SCHWENTICK and THOMAS ZEUME, TU Dortmund University

Patnaik and Immerman introduced the dynamic complexity class DynFO of database queries that can be maintained by first-order dynamic programs with the help of auxiliary relations under insertions and deletions of edges. This article confirms their conjecture that the reachability query is in DynFO.

As a byproduct, it is shown that the rank of a matrix with small values can be maintained in DynFO. It is further shown that the (size of the) maximum matching of a graph can be maintained in non-uniform DynFO, an extension of DynFO, with non-uniform initialisation of the auxiliary relations.

CCS Concepts: • **Theory of computation → Logic and databases**; **Complexity theory and logic**; **Finite Model Theory**;

Additional Key Words and Phrases: Dynamic descriptive complexity, DynFO, reachability

ACM Reference format:
Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. 2018. Reachability Is in DynFO. *J. ACM* 65, 5, Article 33 (August 2018), 24 pages.
https://doi.org/10.1145/3212685

1 INTRODUCTION

In many data management scenarios, data is subject to frequent change. When a web server is temporarily not available, data packages have to be rerouted immediately; when a train is cancelled on short notice, travellers need to find alternative connections as fast as possible. Recomputation of a query after each small change of the data is often not possible due to the large amount of data at hand and efficiency considerations. Often, it is also not necessary: the loss of one server usually affects only a small part of the network. Therefore, it makes sense to consider incremental algorithms that use previously computed auxiliary data to answer queries faster, after a small change.

The first and third authors were partially funded by a grant from Infosys Foundation. The last two authors acknowledge financial support by DFG Grants No. SCHW 678/6-1 and No. SCHW 678/6-2.
Authors' addresses: S. Datta, Chennai Mathematical Institute & UMI ReLaX, H1, SIPCOT IT Park, Siruseri, Kelambakkam, 603103, India; email: sdatta@cmi.ac.in; R. Kulkarni, Chennai Mathematical Institute, H1, SIPCOT IT Park, Siruseri, Kelambakkam, 603103, India; email: kulraghav@gmail.com; A. Mukherjee, Chennai Mathematical Institute, H1, SIPCOT IT Park, Siruseri, Kelambakkam, 603103, India; email: anish@cmi.ac.in; T. Schwentick, TU Dortmund University, Faculty for Computer Science, Otto-Hahn-StraSSe 12, Dortmund, 44227, Germany; email: thomas.schwentick@tu-dortmund.de; T. Zeume, TU Dortmund University, Faculty for Computer Science, Otto-Hahn-StraSSe 12, Dortmund, 44227, Germany; email: thomas.zeume@tu-dortmund.de.

https://doi.org/10.1145/3212685

In this article, we do not study the dynamic scenario from the point of view of incremental algorithms, but rather from the point of view of descriptive complexity (see Reference [22]). More precisely, we use the setting of dynamic complexity theory as introduced by Patnaik and Immerman [36]. Dynamic complexity theory has its roots in theoretical investigations of the view update problem for relational databases. In a nutshell, it investigates the logical complexity of updating the result of a query under deletion or insertion of tuples into a database.

Besides possibly saving resources, a dynamic approach to query answering can increase the expressivity of database query languages. It is well-known that the relational algebra inherits the well-known expressivity limitations from first-order logic. It basically can only express local queries that do not count (see, e.g., Reference [27] for more information on the limits of first-order logic on finite structures). In the dynamic setting, first-order logic is more powerful: as a simple 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.

Starting with work by Dong et al. [9, 12] (with the name first-order incremental evaluation systems, FOIES) as well as Patnaik and Immerman [36], the power of first-order logic as an update mechanism has been studied over the last decades. In this line of work, the result of a query is updated by first-order formulas that have access to the current database and to an auxiliary database that may contain helpful information. The relations of the auxiliary database are updated by first-order formulas as well. Beyond the expressive equivalence with the relational algebra, first-order logic is also an interesting update language thanks to its correspondence to low-level circuit-based complexity classes: queries maintainable by first-order updates can also be maintained by highly parallel algorithms. We refer to the class of queries that can be updated by first-order formulas under single tuple insertions and deletions by DynFO as introduced in Reference [36].

The reachability query returns, for a given graph G, all pairs (s, t) of nodes, for which there is a path from s to t. When investigating the expressive power of DynFO, the reachability query is of particular interest, since it is one of the simplest queries that cannot be expressed (statically) in first-order logic, but rather requires recursion. Actually, it is, in a sense, prototypical due to its connection to transitive closure logic. The question whether the reachability query can be maintained by first-order update formulas has been considered as one of the central open questions in dynamic complexity. It has been studied for several restricted graph classes and variants of DynFO [6, 7, 11, 16, 17, 36, 46]. In this article, we confirm the conjecture of Patnaik and Immerman [36], that the reachability query for general directed graphs is indeed in DynFO.

Theorem 1. *Reachability is in* DynFO.

The proof of Theorem 1 relies on a known reduction from reachability to the matrix rank query and a dynamic program for a suitable restriction of the latter. This query, to which we refer as SVRank (short for *rank for small valued matrices*, cf. Section 2), is defined on quadratic matrices with integer values that are bounded by the number of rows of the matrix.

More precisely, whether there is a path from a node s to a node t in a graph G can be reduced to the question whether a certain matrix has maximal rank. Technically, the reduction thus yields a collection of matrices, one for each pair (s, t) of nodes. It has the additional property that a single change in G (deletion or insertion of an edge) only yields a single change in each of these matrices. Furthermore, the reduction can use arithmetic in a certain generic way. We formalise such reductions as *bounded expansion first-order truth-table reductions* with arithmetic (bfo$(+, \times)$-tt reductions). We refer to their arithmetic-free version as bfo-tt reductions and show that DynFO is closed under bfo-tt reductions.

We show the following result.

Theorem 2. *SVRank is in* DynFO.

Due to the use of arithmetic in the reduction from the reachability query to SVRank, the immediate implication of Theorem 2 and the reduction is that the reachability query is in DynFO(+,×), the extension of DynFO, in which dynamic programs can use an addition and a multiplication relation on their domain from the very beginning of the computation (unlike DynFO programs, where the database/graph and the auxiliary relations do not contain any tuples initially). However, we show that with respect to weakly domain independent queries (cf. Section 2), such as the reachability query, DynFO(+,×) and DynFO coincide. We also show that for weakly domain independent queries, the existence of a DynFO(+,×) program implies the existence of a FOIES.

By bfo-tt reductions to the reachability query, it is not hard to show that satisfiability of 2-CNF formulas and regular path queries for graph databases can also be maintained in DynFO.

Finally, we show that two queries that deal with matchings in graphs can be maintained in a non-uniform extension of DynFO.

Theorem 3. *PerfectMatching and MaxMatching are in non-uniform* DynFO.

Here, PerfectMatching is the Boolean query that asks for the existence of a perfect matching in a graph, whereas MaxMatching returns the size of the maximum matching (encoded as a singleton unary relation). Non-uniform DynFO is the extension of DynFO whose programs can use arbitrarily pre-defined auxiliary relations (similar to non-uniform circuit families).

Related Work. All steps of the proof of Theorem 1 benefit from previous work. The underlying correspondence between graph reachability and the inverse of its adjacency matrix has already been used long ago [5] and the precise relationship between reachability and matrix rank used in our proof has already been stated in Reference [26]. The algorithm constructed for maintaining the rank of a matrix adapts a dynamic sequential algorithm from Reference [14].[1] The third step extends the technique for maintaining arithmetic from Reference [13].

Whether the reachability query can be maintained by first-order update programs has been one of the main questions in dynamic complexity. The positive results towards the resolution of this question can be clustered in two groups:

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

In a sense, the former line of research has won this race, since the methods developed there led to the maintenance algorithm presented here.

The first result in group (1) was that, on arbitrary directed graphs, reachability can be maintained with SQL updates [7]. However, this result requires us to store all possible (simple) paths and therefore may use auxiliary data of exponential size. In addition, it also exploits the aggregation and grouping capabilities of SQL. Hesse significantly improved this result by showing that reachability is in (uniform) DynTC^0 [17] (with polynomial size auxiliary data). His technique is based on 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 Reference [6], this approach was extended to show that reachability can even be maintained in non-uniform $\mathrm{DynAC}^0[2]$. In terms of logic, DynTC^0 and $\mathrm{DynAC}^0[2]$ can be seen as the extensions of DynFO in which update formulas are allowed to use majority quantifiers and modulo-2 counting quantifiers, respectively. In the latter paper, also, the dynamic complexity of matrix rank was studied for the first time (putting it in uniform DynTC^0).

Results of group (2) showed that the reachability query can be maintained in DynFO for undirected graphs [35], directed acyclic graphs [9], and embedded planar graphs [6]. For undirected

[1] We note that the conference version of this article exhibits a different algorithm for this problem.

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) [18]. In the case of undirected graphs, spanning trees [35] or distance functions [16] can be used. The result for directed acyclic graphs is based on a smart observation that allows us to figure out whether there is a path from a to b after deleting some edge (c, d) [9].

In a third line of research on reachability, inexpressibility results have been obtained for fragments of DynFO [8, 11, 46].

As for maximum matching, in References [40, 41], a reduction from maximum matching to matrix rank has been used to construct a dynamic algorithm for maximum matching. While in this construction, the inverse of the input matrix is maintained using the Schwartz Zippel Lemma; we use the Isolation Lemma of Mulmuley et al. [33] to construct a non-uniform DynFO-algorithm for maximum matching.

Organisation

In Section 2, we fix our notation for databases, queries, and linear algebra. In Section 3, we define the dynamic complexity framework and introduce bfo-tt reductions. Section 4 is dedicated to the proof that the reachability query is in DynFO. That regular path queries and 2-SAT are also in DynFO is shown in Section 5. Section 6 presents our results on graph matchings. That the reachability query can also be maintained by FOIES is shown in Section 7. The conclusion is given in Section 8.

2 PRELIMINARIES

By $[n]$, we denote the set $\{1, \ldots, n\}$ and by $[n]_0$, the set $\{0, 1, \ldots, n\}$.

In this article, we are interested in the following algorithmic problems:

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

Problem: MaxMatching
Input: An undirected graph G
Output: The size k of a maximum matching of G

Problem: PerfectMatching
Input: An undirected graph G
Output: Does G have a perfect matching?

A *matching* M of an undirected graph G is a subset of pairwise non-adjacent edges of G. A node is matched by M if it is the endpoint of one of the edges in M. A *maximum matching* of G (also: maximum-cardinality matching) is a matching that has the largest number of edges. A *perfect matching* of G is a matching that matches all vertices.

2.1 Databases and Queries

As much of the original motivation for the investigation of dynamic complexity came from incremental view maintenance (cf. References [9, 12, 36]), it is common to consider logical structures as relational databases and to use notation from relational databases.

A *(relational) schema* τ consists of a set τ_{rel} of relation symbols, accompanied by an arity function $\text{Ar} : \tau_{\text{rel}} \to \mathbb{N}$, and a set τ_{const} of constant symbols. In this article, a *domain* is a finite set. A *database* $\mathcal{D}$ over schema τ with domain D assigns to every relation symbol $R \in \tau_{\text{rel}}$ a relation of arity $\text{Ar}(R)$ over D and to every constant symbol $c \in \tau_{\text{const}}$ an element (called a *constant*) from D. The *active domain* $\text{adom}(\mathcal{D})$ of a database $\mathcal{D}$ consists of those elements of D that either occur in some relation or as a constant.

A *τ-structure* $\mathcal{S}$ is a pair $(D, \mathcal{D})$, where D is a domain and $\mathcal{D}$ is a database with domain D over schema τ. By $\text{dom}(\mathcal{S})$, we refer to D. For a relation symbol $R \in \tau$ and a constant symbol $c \in \tau$, we denote by $R^{\mathcal{S}}$ and $c^{\mathcal{S}}$ the relation and constant, respectively, that are assigned to those symbols in $\mathcal{S}$.

The distinction between structures and databases will be relevant in the dynamic complexity framework, since there the domain D will be static, whereas the database and its active domain might change. However, we often do not keep the two formalisms too much apart and, e.g., refer by "database" to the corresponding structure, in cases where the domain is given by the context (or not important).

Often structures come with special arithmetic relations $<$, $+$, and $\times$ that are interpreted by a linear order on the domain, its induced addition, and its induced multiplication relation. When a linear order is present, we often identify the elements of D with the numbers in $\{0, \ldots, |D| - 1\}$. A pair $(a, b) \in D \times D$ then represents the number $a \times |D| + b$, where, in the latter term, a denotes the number with which the element a is identified. Likewise for tuples of higher arity. It is well known that from $<$, $+$, and $\times$, arithmetic for tuples can be defined in first-order logic.

A *k-ary query* q on τ-structures is a mapping that is closed under isomorphisms and assigns a subset of D^k to every τ-structure over domain D. The problems Reach, MaxMatching, and PerfectMatching can be represented as binary, unary, and boolean queries, respectively, on graph structures, i.e., $\{E\}$-structures where E is a binary relation. For example, the query representing Reach maps a graph structure to a binary relation that contains the transitive closure of the graph. The query representing MaxMatching assumes arithmetic relations to be present and maps a graph structure to the set $\{m\}$, where m is the size of a maximum matching of G.

We write $R \restriction A$ for the restriction of a relation R to tuples over the set A, and $\mathcal{D} \restriction A$ for the database resulting from $\mathcal{D}$ by restricting all relations to tuples over A. That is, $\mathcal{D} \restriction A$ is the substructure of $\mathcal{D}$ induced by A.

A query q is *weakly domain independent*, if $q(\mathcal{D}) \restriction \text{adom}(\mathcal{D}) = q(\mathcal{D} \restriction \text{adom}(\mathcal{D}))$, for all databases $\mathcal{D}$.

2.2 Linear Algebra and Matrices

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 $e_i^{(n)}$, we denote the n-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 matrix A are denoted by $\text{rank}(A)$ and $\det(A)$, respectively. 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).

To the best of our knowledge, computational linear algebra problems like matrix rank and matrix inverse have not been studied before in dynamic complexity (with the notable exception of Boolean matrix multiplication in Reference [19]). Therefore, there is no standard way of representing the matrix rank problem in the dynamic complexity framework. The key question is how to represent the numbers that appear in a matrix. We use a representation that does not allow matrices with large numbers but suffices for our applications in which matrix entries are not (much) larger than the number of rows in the matrix.

We say that an $(m \times m)$-matrix A over $\mathbb{Z}$ has *small values*, if for each $i, j \in \{1, \ldots, m\}$, $|A[i, j]| \leq m$.

Problem: SVRank
Input: $(m \times m)$-matrix A with small values
Output: Rank of A

For this query, $(m \times m)$-matrices A are represented by structures as follows. The domain of the structure contains $m + 1$ elements. There is a linear order $<$ that enables us to identify the universe with $[m]_0$. There are compatible $+$ and $\times$ relations as well. Furthermore, they have two ternary relations A_+, A_- to represent the entries of A. That $A[i, j] = a$, for $a \in \{1, \ldots, m\}$ is represented by a triple (i, j, a) in A_+. Similarly, if $A[i, j] = a$, for $a \in \{-m, \ldots, -1\}$, there is a triple (i, j, a) in A_-. For each i, j, there is at most one triple (i, j, a) in $A_+ \cup A_-$. If, for some i, j there is no such triple (i, j, a), then $A[i, j] = 0$.

The query result of SVRank is a unary relation Q that is supposed to contain a unique element r, the rank of A.

Change operations might insert a triple (i, j, a) to A_+ or A_- (in case no (i, j, b) is there) or delete a triple. That is, basically, single-matrix entries can be set to 0 or from 0 to some other value. However, the relations $<$, $+$, and $\times$ cannot be changed.

3 DYNAMIC COMPLEXITY

In this section, we recall the dynamic complexity framework.

3.1 Dynamic Programs and DynFO

Inputs in dynamic complexity are represented as relational structures as defined in Section 2. The domain is fixed from the beginning, but the database in the initial structure is empty. This initially empty structure is then modified by a sequence of insertions and deletions of tuples.

The goal of a dynamic program is to answer a given query for the database that results from any change sequence. To this end, the program can use an auxiliary data structure represented by an auxiliary database over the same domain. Depending on the exact setting, the auxiliary database might be initially empty or not.

We make this more precise now, closely following the exposition in Reference [42]. A dynamic program $\mathcal{P}$ works on an *input structure* $\mathcal{I}$ over a schema τ_{in} and updates an *auxiliary structure* $\mathcal{A}$ over a schema[2] 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.

The input structure 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 structure 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** $\varphi(\vec{x}, \vec{y})$,

[2]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.

where the formula φ is over $\tau_{\text{in}} \cup \tau_{\text{aux}}$. We call φ the *update formula* for T under insertions into R. 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. We refer to the insertion or deletion of a tuple together with the update operations applied by $\mathcal{P}$ as a *change step*.

The dynamic program $\mathcal{P}$ *maintains* a k-ary query q if it has a k-ary auxiliary relation Q that, after each change sequence, contains the result of q on the current input database. More precisely, for each non-empty[3] sequence α of changes and each empty input structure $\mathcal{I}_\emptyset$, relation Q in $\mathcal{P}_\alpha(\mathcal{S}_\emptyset)$ and $q(\alpha(\mathcal{I}_\emptyset))$ 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.

Several dynamic settings have been studied in the literature (see, e.g., References [13, 15, 16, 36]). Here, we concentrate on the following three dynamic complexity classes:

- DynFO is the class of all dynamic queries that can be maintained by dynamic programs with formulas from first-order logic starting from an empty database and empty auxiliary relations.
- DynFO(+,×) is defined as DynFO, but the programs have three particular auxiliary relations that are initialised as a linear order and the corresponding addition and multiplication relations. More precisely, a DynFO(+,×) program initially has a binary relation $<$, which is initially some linear order on the domain of the input database and is never changed. The relations + and × are ternary relations defined as follows: $(u, v, w) \in +$ if $\mu(u) + \mu(v) = \mu(w)$, where μ is the mapping that maps the i-th element with respect to $<$ to the number i, where we consider the smallest element as the 0-th. Likewise, $(u, v, w) \in \times$ if $\mu(u) \times \mu(v) = \mu(w)$. There might be further auxiliary relations, but they are initially empty. A query is in DynFO(+,×) if there is a DynFO(+,×)-program that maintains it, for every (finite) domain and every initial choice of $<$ on the domain.
- *Non-uniform* DynFO is defined as DynFO, but the auxiliary relations may be initialised arbitrarily. A query is in non-uniform DynFO, if there is a dynamic program such that, for every finite domain, the auxiliary relations can be chosen in some way such that the program correctly maintains the query.

It is well known that first-order logic with arithmetic is as powerful as uniform AC^0-circuits [4]. This correspondence naturally transfers to the dynamic setting. That is, a query can be maintained in DynFO(+,×) if and only if it can be maintained by uniform AC^0 circuits.

3.2 Logical Truth-Table Reductions

In this article, we use a more general notion of reductions between queries than the bounded-expansion first-order reductions (bfo-reductions) in Reference [35]. They basically compare to bfo reductions like truth-table reductions relate to many-one reductions in complexity theory.

The rough idea is to reduce the computation of a query q on a structure $\mathcal{S}$ to the computation of a query q' by

[3]This technical restriction ensures that we can handle, e.g., 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.

(1) defining in a first-order fashion, from $\mathcal{S}$, a collection of structures of the form $\mathcal{J}(\mathcal{S}, \vec{a})$, one for each tuple $\vec{a}$ of some arity m over the domain of $\mathcal{S}$,
(2) combining all query results $q'(\mathcal{J}(\mathcal{S}, \vec{a}))$ into a structure $\mathcal{S}'$, and
(3) defining $q(\mathcal{S})$ from $\mathcal{S}$ and $\mathcal{S}'$ by a first-order formula.

A technical complication arises from the fact that we need to allow the structures $\mathcal{J}(\mathcal{S}, \vec{a})$ to be defined not over the domain of $\mathcal{S}$ but over some Cartesian product over this domain. Thus, we have to deal with two "dimension parameters": d will denote the dimension of the domain of the structures $\mathcal{J}(\mathcal{S}, \vec{a})$ and m will denote the arity of the tuples $\vec{a}$.

Let, in the following σ, τ be relational schemas.

An *interpretation* $\mathcal{J}$ of dimension d and arity m from databases with schema σ to databases with schema τ consists of

— a σ-formula $\varphi_D(\vec{x}, \vec{y})$ and
— σ-formulas $\varphi_R(\vec{x}_1, \ldots, \vec{x}_{\mathrm{Ar}(R)}, \vec{y})$, for every $R \in \tau$,

where $\vec{y} = y_1, \ldots, y_m$, $\vec{x} = x_1, \ldots, x_d$ and, for every j, $\vec{x}_j = x_{j1}, \ldots, x_{jd}$.

For every (finite) σ-structure $\mathcal{S}$ and each m-tuple $\vec{a}$ over the domain S of $\mathcal{S}$, the interpretation $\mathcal{J}$ defines a structure $\mathcal{J}(\mathcal{S}, \vec{a})$ with

— domain $D^{\vec{a}} \stackrel{\text{def}}{=} \{\vec{b} \in S^d \mid \mathcal{S} \models \varphi_D(\vec{b}, \vec{a})\}$, and
— relations

$$R^{\vec{a}} \stackrel{\text{def}}{=} \{(\vec{b}_1, \ldots, \vec{b}_{\mathrm{Ar}(R)}) \in (D^{\vec{a}})^{\mathrm{Ar}(R)} \mid \mathcal{S} \models \varphi_R(\vec{b}_1, \ldots, \vec{b}_{\mathrm{Ar}(R)}, \vec{a})\},$$

for every R.

A *first-order truth-table query-to-query reduction* (fo-tt reduction) $\mathcal{R} = (\mathcal{J}, \varphi)$ from q to q' consists of an interpretation $\mathcal{J}$ and a formula φ with free variables $x_1, \ldots, x_k$, where k is the arity of q, which fulfills the following reduction property.

For every (finite) structure $\mathcal{S}$, $q(\mathcal{S})$ is the set $\{\vec{t} \mid \mathcal{S}' \models \varphi(\vec{t})\}$, where the structure $\mathcal{S}'$ with domain $\mathrm{dom}(\mathcal{S})$ is defined as follows. Let d be the dimension of $\mathcal{J}$, m its arity, ℓ the arity of q', and σ, τ the schemas of $\mathcal{J}$.

— $\mathcal{S}'$ has all relations from $\mathcal{S}$;
— Furthermore, $\mathcal{S}'$ has a relation $\hat{Q}$ of arity $m + d\ell$ that contains all tuples of the form $(\vec{a}, \vec{s})$, where $\vec{a} \in U^d$ and $\vec{s} \in q'(\mathcal{J}(\mathcal{S}, \vec{a}))$. In $(\vec{a}, \vec{s})$, the ℓ-tuple s over universe U^d is considered a $d\ell$-tuple in the obvious way.

We refer to φ as the *wrap-up formula* of the reduction.

In analogy to Reference [35], we say that an interpretation $\mathcal{J}$ has *bounded expansion* if there is a constant *expansion bound* c such that for all structures $\mathcal{S}_1, \mathcal{S}_2$ over the same domain D, which differ by exactly one tuple, and for every tuple $\vec{a}$ over D, the databases $\mathcal{J}(\mathcal{S}_1, \vec{a})$ and $\mathcal{J}(\mathcal{S}_2, \vec{a})$ differ by at most c tuples. A fo-tt reduction has bounded expansion if its underlying interpretation has bounded expansion. We refer to fo-tt reductions with bounded expansion as bfo-tt reductions.

Example 1. As an illustrating example, we show how the well-known reduction from 2-Sat to Reach can be cast as a bfo-tt reduction. The Boolean query 2-Sat asks whether a given propositional formula in 2-CNF has a satisfying assignment. Here, a propositional formula is in 2-CNF if it is in conjunctive normal form and each clause contains at most two literals.

Instances of 2-Sat can be represented as structures as follows. The domain of a structure representing a formula φ is the set of variables of φ. The clauses of φ are represented by three binary

input relations C_{TT}, C_{TF}, and C_{FF} such that a tuple $(x, y) \in C_{TT}$ corresponds to a clause $x \vee y$, a tuple $(x, y) \in C_{TF}$ to a clause $x \vee \neg y$, and a tuple $(x, y) \in C_{FF}$ to a clause $\neg x \vee \neg y$.

Thus, insertion and deletion of tuples corresponds to insertion and deletion of clauses in a natural way.

Intuitively, the reduction maps a 2-CNF-formula θ with variables V to the graph $G = (V \cup \overline{V}, E)$, where $\overline{V} = \{\neg x \mid x \in V\}$ and E contains the edges $(\neg L, L')$ and $(\neg L', L)$ if $L \vee L'$ is a clause in θ. It can be easily seen that θ is satisfiable if and only if there is no variable $x \in V$ such that there exist both a path from x to $\neg x$ and a path from $\neg x$ to x in G.

More formally, the graph G will be encoded over the set of pairs over V. For two variables $u \neq v$ from V, a pair (x, u) will represent x and (x, v) will represent $\overline{x}$. This can be achieved by a two-dimensional[4] interpretation $\mathcal{J}$ of arity 2. For each pair (u, v) of variables, $\mathcal{J}(\theta, u, v)$ is the graph defined as above with u and v indicating positive and negated literals, respectively. Thus, the formula $\varphi_D((x_1, x_2), (y_1, y_2))$ could be chosen as $(x_2 = y_1) \vee (x_2 = y_2)$, allowing only pairs in the domain of $\mathcal{J}(\theta, u, v)$ whose second entry is one of the parameters given by y_1 and y_2. The formula $\varphi_E((x_{11}, x_{12}), (x_{21}, x_{22}), (y_1, y_2))$ can be chosen as

$$\Big((C_{TT}(x_{11}, x_{21}) \vee C_{TT}(x_{21}, x_{11})) \wedge (x_{12} = y_2) \wedge (x_{22} = y_1)\Big) \vee \cdots,$$

with respective subformulas for C_{TF} and C_{FF}. Finally, the wrap-up formula φ can be chosen as

$$\exists u, v(u \neq v) \wedge \neg\exists x(\hat{Q}((x, u), (x, v), (u, v)) \wedge \hat{Q}((x, v), (x, u), (u, v))).$$

Since the modification of a single clause in θ induces only two first-order definable modifications to the edge set of each corresponding graph, the reduction is also bounded.

The relevance of bfo-tt reductions for this article stems from the following property.

Proposition 4. DynFO *is closed under bfo-tt reductions, that is, if there is a bfo-tt reduction from a query q to a query q' and $q' \in$* DynFO*, then $q \in$* DynFO.

Proof Sketch. Let $(\mathcal{J}, \varphi)$ be a bfo-tt reduction from q to q' with expansion bound c. Let σ, τ be the schemas of q and q', respectively, and let d be the dimension and m the arity of $\mathcal{J}$. Let $\mathcal{P}'$ be a dynamic program for q'.

The program $\mathcal{P}'$ can be turned into a dynamic program $\mathcal{P}$ for σ-structures that have one auxiliary relation R of arity $m + d\mathrm{Ar}(R')$, for every (input and auxiliary) relation R' of $\mathcal{P}'$. For each m-tuple $\vec{a}$, the program $\mathcal{P}$ simulates the behaviour of $\mathcal{P}'$ on $\mathcal{J}(\mathcal{S}, \vec{a})$, independently. Since each change operation for $\mathcal{S}$ translates into a sequence of at most c change operations for $\mathcal{J}(\mathcal{S}, \vec{a})$, this amounts, for every tuple $\vec{a}$, to a sequence of at most c update operations of $\mathcal{P}'$, which can be applied successively (but in parallel for different $\vec{a}$).

Since the query relation of $\mathcal{P}'$ is one of its auxiliary relations, $\mathcal{P}'$ has, in particular, the relation $\hat{Q}$ from the reduction property above available, and can therefore compute $q(\mathcal{S})$ in a first-order manner. □

We actually use slightly more powerful logical reductions when we work with DynFO(+,×). We define bfo(+, ×)-tt reductions in almost the same way as bfo-tt reductions, but they assume in $\mathcal{S}$ distinguished relations $<$, $+$, and $\times$, representing arithmetic on the universe. In such a reduction, the query q must not depend on the choice of $<$, $+$, and $\times$, but $\mathcal{J}(\mathcal{S}, \vec{a})$ of course can. By an adaptation of the proof of Proposition 4, it can be shown that if there is a bfo(+, ×)-tt reduction

[4]We emphasise that the reduction constructs one graph for each pair (u, v) with $u \neq v$. For simplicity, we ignore the case $u = v$ in the following. Graphs defined with these parameters do not contribute to the success of the reduction. We likewise ignore 2-CNF-formulas with only one variable.

from a query q to a query q', then $q \in \mathsf{DynFO}(+,\times)$ if $q' \in \mathsf{DynFO}$. Similar reductions can be defined for non-uniform DynFO as well.

4 REACHABILITY IS IN DYNFO

The goal of this section is to prove the main result of this article.

Theorem 5. *Reach* $\in$ DynFO.

The proof of Theorem 5 consists of four relatively simple steps and it involves one additional query, RankModp, that will be defined later.

(1) Reach can be reduced to SVRank by a bfo(+, ×)-tt reduction.
(2) SVRank can be reduced to RankModp by a bfo-tt reduction.
(3) RankModp $\in$ DynFO.
(4) For every weakly domain independent query q, if $q \in \mathsf{DynFO}(+,\times)$, then $q \in \mathsf{DynFO}$.

From (1)–(3), it follows that Reach $\in \mathsf{DynFO}(+,\times)$. From (4), we can conclude that Reach $\in$ DynFO, since Reach is weakly domain independent.

All four steps build, to some extent, on previous work. The basic correspondence between graph reachability and the inverse of its adjacency matrix has already been used long ago [5] and the precise relationship between reachability and matrix rank used in our proof has already been stated in Reference [26]. The algorithm constructed for the third step adapts a dynamic sequential algorithm for maintaining rank from Reference [14]. The last step extends the technique for maintaining arithmetic presented in Reference [13].

In the following, we describe the four steps separately, with each step mostly self-contained.

4.1 From Reachability to Matrix Rank

Towards the reduction from Reach to SVRank, 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 Reference [21, Theorem 6.1.10.]) is that $I - \frac{1}{n}A_G$ is invertible and

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

From the sum on the right-hand side, it can be easily concluded that the inverse of $I - \frac{1}{n}A_G$ has a non-zero entry at position (s, t) if and only if t is reachable from s.

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.

Then the following chain of equivalences holds:

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

Here, B^{+s} denotes the $((n+1)\times n)$-matrix that is obtained from B by adding an additional row $(e_s^{(n)})^\top$, and B^{+st} denotes the extension of B^{+s} by the additional column vector $e_t^{(n+1)}$.

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 bfo$(+,\times)$-tt reduction from Reach to SVRank. To this end, we observe that, for graphs with n vertices, the resulting $((n+1)\times(n+1)$-matrix B^{+st} has small values, in the sense defined in Section 2.

It is easy to check that, in the presence of arithmetic, B^{+st} can be obtained from G by a two-dimensional and binary bfo$(+,\times)$-tt reduction $(\mathcal{J},\varphi)$. For each database $\mathcal{D}$ (representing a graph G), and each pair (s,t) over the universe U of $\mathcal{D}$, $\mathcal{J}(\mathcal{D},(s,t))$ is a database that encodes B^{+st}. The interpretation $\mathcal{J}$ uses two dimensions because the universe representing B^{+st} is of size $n+2$ for graphs with n vertices.

For every pair (s,t), the result relation SVRank $(\mathcal{J}(\mathcal{D},(s,t)))$ is the set $\{(r)\}$, where r is the rank of B^{+st}. Therefore, the relation $\hat{Q}$ consists of all triples (s,t,r), for which r is the rank of B^{+st}. Thus, for each (s,t), the wrap-up formula φ only needs to check whether $r=n+1$, where n is the number of nodes of G.

Finally, each change in G results in only one change in B^{+st}, for every (s,t), and therefore the reduction actually has expansion bound 1.

4.2 From Rank to Rank Mod p

Even though input matrices for SVRank have small entries, the maintenance algorithm on which the DynFO-program for SVRank will be based needs to deal with matrices that have large entries. To avoid the complications that arise from the need to compute with large numbers, we show next that, to maintain the rank of a matrix A with small values, it suffices to maintain its rank over the field $\mathbb{Z}_p$, for sufficiently many primes p. We denote this rank by $\mathrm{rank}_p(A)$. Formally, this gives rise to a reduction from SVRank to the following query.[5]

Problem: RankModp
Input: $(m\times m)$-matrix A with values from $\{0,\ldots,p-1\}$, prime $p\le m^2$
Output: Rank of A over $\mathbb{Z}_p$

The bound m^2 might appear a bit arbitrary, but we will see that it just suffices.

The reduction from SVRank to RankModp is actually pretty simple. It is based on the fact that, for large enough m, it suffices to maintain the rank for all primes $p\le m^2$, as we will argue next. These primes indeed suffice thanks to the simple observation that $\mathrm{rank}(A)\ge k$ if and only if $\mathrm{rank}_p(A)\ge k$ for some prime $p\le m^2$. Clearly, $\mathrm{rank}(A)\ge k$ if and only if there is some $k\times k$ submatrix A' of A with $\det(A')\neq 0$. Since $\det(A')$ is bounded by $m!m^m$, its binary representation has $O(m\log m)$ digits (for sufficiently large m). Since, for large enough n, there are more than $\frac{n}{\ln n}$ prime numbers between 1 and n [39], there are more than $\frac{m^2}{2\ln m}$ prime numbers below m^2 and thus their product is clearly larger than $m!m^m$. Therefore, $\det(A')\neq 0$ if and only if there exists a prime $p\le m^2$ such that $\det(A')\not\equiv 0 \pmod p$. Thus, for large enough m, $\mathrm{rank}(A)\ge k$ if and only if there exists a prime $p\le m^2$ such that $\mathrm{rank}_p(A)\ge k$.

The actual reduction $(\mathcal{J},\varphi)$ from SVRank to RankModp is one-dimensional and binary. Each pair $\vec{i}=(i_1,i_2)$ over m is interpreted as the number $n(\vec{i})\stackrel{\text{def}}{=}(i_1-1)m+(i_2-1)$. For a database $\mathcal{D}$

[5] As a dynamic problem, we do *not* consider operations that change p. However, RankModp can also be maintained under these operations by simply always maintaining the rank over $\mathbb{Z}_p$, for every prime $p\le m^2$.

representing an input matrix A for SVRank and each pair $\vec{i} = (i_1, i_2)$, $B^{\vec{i}} \stackrel{\text{def}}{=} \mathcal{J}(\mathcal{D}, \vec{i})$ is the all-zero matrix if $n(\vec{i})$ is not a prime. If $n(\vec{i})$ is some prime p, then $\mathcal{B}^{\vec{i}}$ represents the matrix A over $\mathbb{Z}_p$. Whether a number $n(\vec{i})$ is a prime number can be tested by a first-order formula thanks to the availability of $\times$.

The wrap-up formula φ simply computes the maximum k, such that for some prime $p = n(\vec{i})$, the result relation for RankModp$(\mathcal{J}(\mathcal{D}, \vec{i}))$ contains k.

4.3 Maintaining Rank Mod p in DynFO

In this subsection, we show that the rank of a matrix modulo, some prime p, can be maintained in DynFO. This is the most important intermediate result for Theorem 5 and is also interesting in its own right. First, we give an informal description of the dynamic algorithm for matrix rank. Afterwards, we describe how it can be transformed into a DynFO program. In the following, we fix $m > 0$, a prime $p < m^2$, and only consider matrices with small values.

The algorithm is an adaptation of a dynamic algorithm that has been stated in Reference [14]. The idea is to maintain an invertible matrix B and a matrix E in reduced row-echelon form such that $BA = E$. That E is in reduced row-echelon form means that

- —the leftmost non-zero entry (the *leading entry*) in every row is 1,
- —the column of such a leading entry only contains zero entries otherwise, and
- —rows are sorted in a "diagonal" fashion; that is, the column numbers of leading entries strictly increase with the row number.

The rank of A equals the number of non-zero rows of E thanks to $\text{rank}(E) = \text{rank}(BA) = \text{rank}(A)$ and the structure of E. Thus, maintaining the matrices B and E suffices to maintain the rank of A.

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

After a change of $A[i, j]$, the product BA' differs from BA 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 BA':
   - —let some entry with a maximum number of successive zeros in its row (right after column j) 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.[6]

(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 (row) positions.

An illustrating example can be found in Figure 1.

For each of the row operations in the algorithm, the same operation is applied to the matrix B. This ensures that $B'A' = E'$. As all these row operations correspond to multiplying a suitable elementary matrix from the left, B remains invertible (see, e.g., Reference [32, p. 133]). Each of the three steps can be performed in constant parallel time.

[6]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.

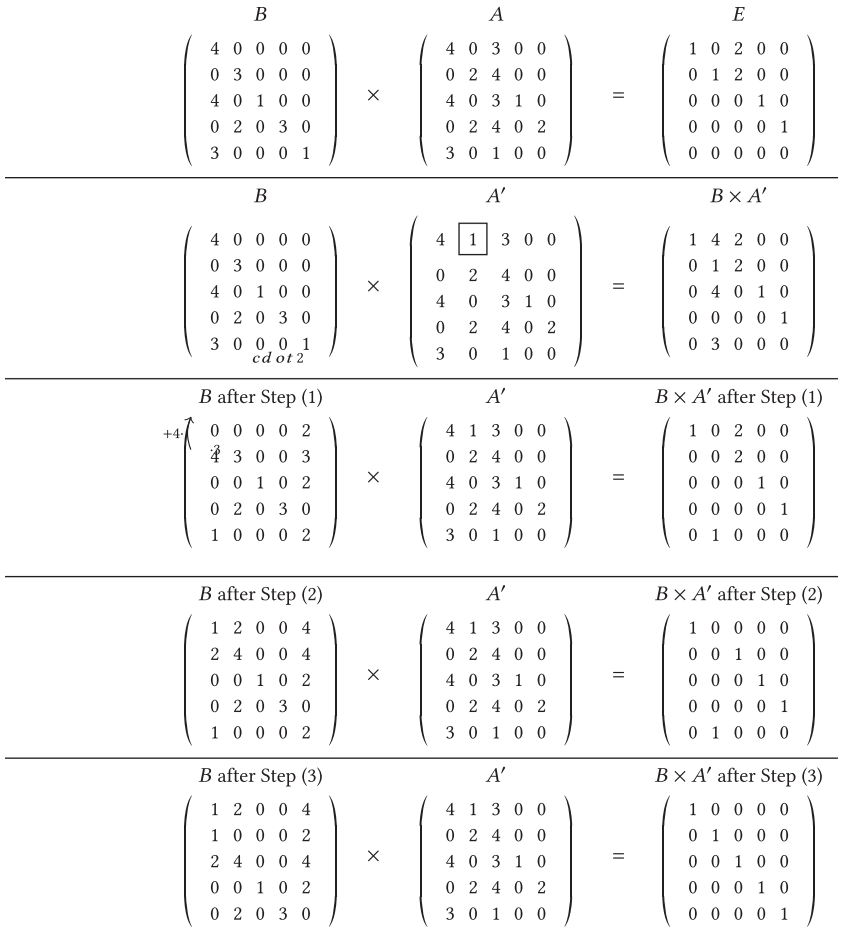

Fig. 1. Illustration of the modifications necessary for one change in matrix A for $p = 5$.

A dynamic program can be easily obtained from the informal description given above. It maintains auxiliary relations that encode the matrices B and E. As each of the steps (1)–(3) can be performed in constant parallel time and since $<$, $+$, and $\times$ are available in the logical representation of the matrix A, the steps can be translated into a DynFO update program $\mathcal{P}$ in a straightforward way.

Theorem 6. *RankModp is in* DynFO.

4.4 DynFO(+, ×) vs. DynFO

Since Reach is clearly weakly domain independent, the proof for Theorem 5 can be completed by a proof for the following result.

Proposition 7. *If a query $q \in$ DynFO(+,×) is weakly domain independent, then $q \in$ DynFO.*

Etessami already observed that DynFO programs have the same expressive power as DynFO(+,×) programs, if, before the actual change sequence starts, for each element u of the universe, the edge (u, u) is inserted and afterwards deleted [13]. He described how these preliminary changes can be used to construct a linear order and compatible + and × 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 elements, and corresponding ternary addition and multiplication relations, respectively. Here, an element u of the domain is called *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. In the following, we refer to the set of activated elements of change sequence α by $A(\alpha)$ and by A, if α is clear from the context.[7]

We point out the subtle differences between elements of the domain, elements of the *active* domain, and activated elements. The domain contains all elements that can be used in relations and does not change during a dynamic computation. The active domain $\text{adom}(\mathcal{D})$ of a database $\mathcal{D}$ that results from a change sequence α (applied to an initially empty database) consists of all elements that occur in some tuple of $\mathcal{D}$. An element is activated if it occurs somewhere in α. In particular, every element of $\text{adom}(\mathcal{D})$ is activated and every activated element is in the domain, but not necessarily vice versa. For example, adding the edges $(1, 2)$ and $(2, 3)$ to an initially empty graph over domain $\{1, 2, 3, 4\}$ and subsequently deleting the edge $(1, 2)$, yields an input database with active domain $\{2, 3\}$ and activated elements $1, 2, 3$.

We show next that DynFO programs can simulate DynFO(+,×) programs for weakly domain independent queries without any form of preprocessing.

Proof (of Proposition 7). Let q be a weakly domain independent query and $\mathcal{P}$ a DynFO(+,×) program that maintains q. For simplicity, we assume that q uses only one binary relation E as input relation, 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 first give a rough description of the construction of $\mathcal{P}'$. More details will be given below. The update program $\mathcal{P}'$ maintains a linear order $<$ on the set A of activated elements. Thanks to that linear order, we can always associate A with a set of size $m \stackrel{\text{def}}{=} |A|$ of the form $[m-1]_0$ with the natural linear order. In fact, we assume for the moment that A is always of this form. Furthermore, an addition relation and a multiplication relation on A is maintained, just as in Reference [13].

For the construction of $\mathcal{P}'$ we view each computation of $\mathcal{P}$ as a sequence of stages, based on the size of A. More precisely, we say that a computation of $\mathcal{P}$ on a universe U of size n is in stage $i < \log\log n + 1$, if more than N_i but at most N_{i+1} elements of U are activated, where $N_i \stackrel{\text{def}}{=} 2^{2^i}$, for every $i \geq 0$. We will ignore the case where $\leq N_0 = 2$ elements are activated in the following; it can be easily dealt with separately.

The basic idea of the construction of $\mathcal{P}'$ is to use different *threads* that simulate the different stages of $\mathcal{P}$ and we refer to the thread that is responsible for stage i as thread i. For each i, thread i begins as soon as $\mathcal{P}$ enters stage $i-1$ and ends at the end of stage i of $\mathcal{P}$. During stage i of $\mathcal{P}$, thread i is *in charge*. The query result for q is always provided by the thread that is in charge. See Figure 2 for an illustration.

When thread i starts, a linear order, an addition relation, and a multiplication relation over $[N_{i-1}-1]_0$ are available. From these relations, a linear order, an addition relation, and a multiplication relation on 4-tuples over $[N_{i-1}-1]_0$ can be easily defined in first-order logic.

[7] Usually, α is just the sequence of all changes of the computation at hand.

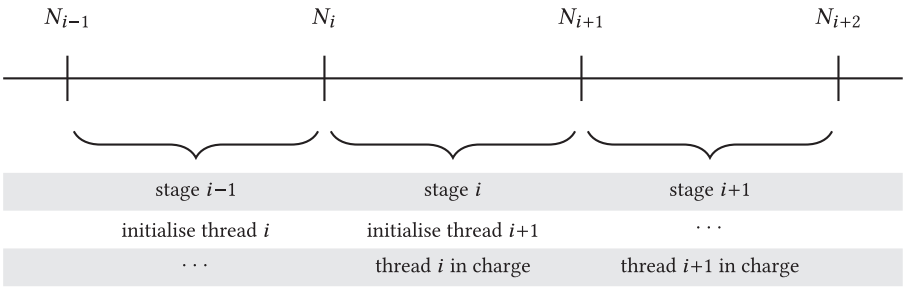

Fig. 2. Illustration of the stages used in the proof of Proposition 7.

Thread i uses the set of 4-tuples over $[N_{i-1} - 1]_0$ as universe of size N_{i+1}. It uses one $4k$-ary auxiliary relation R' for every (auxiliary or input) k-ary relation R of $\mathcal{P}$. It starts on the structure over $([N_{i-1}]_0)^4$ with empty relation E, with the linear order and the corresponding addition and multiplication relations over $([N_{i-1} - 1]_0)^4$, but with otherwise empty auxiliary relations. It is thus in the position to simulate the behaviour of $\mathcal{P}$ on an initially empty structure.

By E', we refer to the 8-ary relation of thread i corresponding to the input relation E of $\mathcal{P}$. When stage $i - 1$ starts, relation E might already contain up to (around) N_{i-1}^2 edges, whereas E' is empty, since thread i has not started yet. Therefore, thread i cannot immediately simulate $\mathcal{P}$ in a lock-step fashion, but it first has to catch up with $\mathcal{P}$. Indeed, thread i will make sure that at the end of stage $i - 1$, all tuples in E have corresponding tuples in E', so that it is prepared to be in charge.

To catch up, thread i needs to add more than one edge per step. It is not hard to figure out that it suffices to add at most four edges per step to E'. The details will be given below. During stage i, thread i can simulate $\mathcal{P}$ in a lock-step fashion, mimicking every step. The universe $([N_{i-1} - 1]_0)^4$ is large enough to represent each new element that is activated by some 4-tuple over $[N_{i-1} - 1]_0$. After stage i, thread i is abandoned and thread $i + 1$ takes over. The moment, when thread $i + 1$ has to take over can be recognised by maintaining a counter for each thread: if the counter of thread i reaches the value $(N_{i-1})^4$ then thread $i + 1$ has to take over in the next step.

It should be noted that it does not matter that the linear order used in each simulation depends on the arrival order of edges, since $\mathcal{P}$ is guaranteed to yield the same result for every linear order.

Next, we describe $\mathcal{P}'$ in more detail.

We describe first how $\mathcal{P}'$ constructs a linear order[8] $<$, an addition relation $+$ and a multiplication relation $\times$ on the set A of activated elements. This part of the simulation is just as in Reference [13]. We recall that $\mathcal{P}$ never changes its linear order, addition relation, and multiplication.

The relation $<$ orders the activated elements in the order of activation. For concreteness: if an edge (a, b) is inserted, the two elements a and b are activated and become the two largest elements of $<$ with $a < b$. The update formula for determining whether a tuple (y_1, y_2) is in the relation $<$ after inserting an edge (x_1, x_2) into E states that

— $y_1 < y_2$; or
— y_1 is already activated, $y_2 = x_1$ or $y_2 = x_2$, and is not yet activated; or
— $y_1 = x_1$, $y_2 = x_2$, $y_1 \neq y_2$, and both y_1 and y_2 are not yet activated.

That an element x is activated can be expressed by $\exists y\, x < y \vee y < x$. For deletion operations, nothing has to be changed; that is, the update formula for $<$ is $\phi^{<}_{\mathrm{DEL}E}(x_1, x_2; y_1, y_2) = y_1 < y_2$.

[8]We use infix notation for $<$, $+$ and $\times$.

We always identify activated elements with their position in $<$; that is, the minimal element in $<$ is considered as 0, the second as 1 and so on. We use numbers as constants in formulas. It is straightforward to replace these numbers by "pure" formulas. For example, the subformula $x > 1$ can be replaced by $\exists x_1 \exists x_2\ x_1 < x_2 \wedge x_2 < x$. The formulas for $+$ and $\times$ are in the same spirit and use the well-known inductive definitions of addition and multiplication, respectively.

Thread i considers 4-tuples as 4-digit base-N_{i-1} numbers and thus identifies a 4-tuple (u_1, u_2, u_3, u_4) over $[N_{i-1} - 1]_0$ with the number $u_1 \times N_{i-1}^3 + u_2 \times N_{i-1}^2 + u_3 \times N_{i-1} + u_4$. We note that $<$, $+$, $\times$ can be lifted to relations over 4-tuples in first-order logic and, therefore, they need not be maintained as separate auxiliary relations.

During stages $i-1$ and i, thread i maintains a bijection g_i between the activated elements and 4-tuples over $[N_{i-1} - 1]_0$. At the start of thread i, $g_i(k) = (0, 0, 0, k)$, for every $k \in [N_{i-1} - 1]_0$, and g_i is extended in a straightforward fashion.

As explained above, thread i has to catch up during stage $i-1$ to make sure that, at the beginning of stage i, its relation E' is isomorphic to E under g_i. To this end, thread i decreases the size of the symmetric difference $\Delta \stackrel{\text{def}}{=} E \triangle g_i^{-1}(E')$ of E and $g_i^{-1}(E')$ by three tuples, for each change step, in the following fashion. When a change δ occurs, it is first simply applied to E, without triggering the associated update operations. Afterwards, thread i identifies the lexicographically smallest (up to) four pairs e_1, e_2, e_3, e_4 over $[N_i]_0$ in Δ and sequentially applies the appropriate update operations. That is, if $e_j \in E \setminus g_i^{-1}(E')$, it simulates the operations of $\mathcal{P}$ for an insertion of e_j (and thus effectively inserts $g_i(e_j)$) and otherwise for deletion. It is easy to see that, in this way, $|\Delta|$ indeed decreases by at least three, unless $|\Delta| < 4$ already. Since $|\Delta| \leq N_{i-1}^2$ initially, $\frac{1}{3}N_{i-1}^2$ change steps suffice for thread i to catch up. Since stage $i-1$ has at least $\frac{1}{2}(N_i - N_{i-1})$ change steps and $N_i - N_{i-1} = N_{i-1}^2 - N_{i-1} \geq \frac{3}{4}N_{i-1}^2$ for $i \geq 2$, this really works out.[9]

During stage i, when a change δ modifies a tuple e, thread i of program $\mathcal{P}'$ applies δ to $g_i(e)$, and performs the necessary updates to the relations of thread i. The query result during stage i is always $g_i^{-1}(Q')$, where Q' is the auxiliary relation corresponding to $\mathcal{P}$'s query relation Q in $\mathcal{P}'$.

This completes the description of the behaviour of thread i, for each i.

Of course, it is not possible to let each thread use its own set of auxiliary relations. However, we can simply increase the arity of each relation by one and use the additional entry to indicate, for each tuple, the number of the thread, for which it is used. As an example, all relations E' are encoded into one 9-ary relation $\widehat{E}$ and the relation E' of thread i is just the set of tuples $\{\bar{t} \mid (i, \bar{t}) \in \widehat{E}\}$. Similarly, the bijections g_i can be stored in a single 6-ary relation $\widehat{G}$ that contains a tuple $(i, k, \bar{t}) \in \widehat{G}$ if $g_i(k) = \bar{t}$.

The correctness of $\mathcal{P}'$ can be shown as follows. Let α denote some change sequence and, for each i, let α_i denote the prefix of α that lasts until the end of stage i. We refer to the change sequence of $\mathcal{P}$ that arises from the process that reduces the size of Δ for thread i during stage $i-1$, as explained above, by γ_{i-1}. We argued above that $\alpha_{i-1}(\mathcal{I}_\emptyset) = \gamma_{i-1}(\mathcal{I}_\emptyset)$ holds. The program $\mathcal{P}'$ applies $g_i(\gamma_{i-1})$ in stage $i-1$, and therefore $g_i(\mathcal{P}_{\gamma_{i-1}}(\mathcal{S}_\emptyset)) = \mathcal{P}'_{g_i(\gamma_{i-1})}(g_i(\mathcal{S}_\emptyset))$ can be shown by a simple induction on the length of change sequences. That is, the state of thread i of $\mathcal{P}'$ at the end of stage $i-1$ is the isomorphic image of some state that $\mathcal{P}$ can reach for the input database $\alpha_i(\mathcal{I}_\emptyset)$.

Finally, it is easy to see that during stage i, program $\mathcal{P}'$ can correctly keep track of the changes and updates. Thanks to the weak domain independence of query q, it does not matter that the simulation uses a domain of size $N_{i+1} < n$ rather than n, since the activated part of the domain during stage i is always of size at most N_{i+1}. Therefore $\mathcal{P}'$ has a correct output, at any time. □

[9] The border case $i = 1$ and $|\delta| \leq 3$ can be handled in a straightforward, mostly analogous way.

We remark that Proposition 7 does not hold for arbitrary queries. For example, the domain-dependent boolean query q_{even}, which is true for domains of even size and false otherwise, cannot be maintained in DynFO from scratch. This is because the first-order initialisation formulas cannot tell domains of even and odd size apart for large, empty structures (see, e.g., Reference [27]).

5 SOME APPLICATIONS

From Theorem 5, the maintainability of other important queries can be inferred. The bounded first-order reduction shown in Example 1 immediately yields

Corollary 8. 2-*Sat is in* DynFO.

We next exhibit a straightforward bounded first-order reduction from regular path queries to reachability.

Graph databases have received considerable attention in the database theory community recently (see, e.g., References [1, 2, 28, 31] and References [3, 44] for surveys). Usually they contain huge amounts of data and, therefore, queries on graph databases should be evaluated in parallel and, if possible, dynamically. In the following, we show that the answer of a (fixed) regular path query can be re-evaluated dynamically after a modification of a graph database.

We make those notions more precise first. Graph databases are usually modeled by directed graphs with edge labels from a finite alphabet. A regular path query q is a regular expression over label names. Evaluating q on a graph database G yields all pairs (u, v) of nodes for which there is a (not necessarily simple) path from u to v in G whose sequence of labels is in the language specified by q.[10] We thus model graph databases as finite structures with one binary relation E_a per edge label a.

In the following, we show that regular path queries can be maintained in DynFO. To this end, we present a simple and well-known bounded first-order reduction to the reachability problem [24]. Let A be an NFA for the language of a regular path query q, let Q be its set of states, and let s_0 be the initial state and s_f the unique accepting state of A, respectively. Let the synchronised product $G \times A$ of G and A be the (directed, unlabeled) graph with node set $G \times Q$ and an edge from (u_1, p_1) to (u_2, p_2) if there is an a-labeled edge from u_1 to u_2 for some symbol a, for which there is also a transition from p_1 to p_2. Then, $(u, v) \in q(G)$ holds if and only if (v, s_f) is reachable from (u, s_0) in $G \times A$. Since each single change in G only induces at most $|Q|$ first-order definable changes in $G \times A$, the reduction is bounded and therefore, the maintainability of (fixed) regular path queries follows from Theorem 5. This easily extends to conjunctions of regular path queries since DynFO is closed under boolean operations.

Corollary 9. *Regular path queries and conjunctions thereof can be maintained in* DynFO.

Further classes of query languages for labeled graphs have been studied in the dynamic context in the literature, see References [34, 43, 45].

6 MAINTAINING THE SIZE OF MAXIMUM MATCHINGS

Matchings in graphs are one of the most studied graph-theoretical concepts in Computer Science with many applications (see, e.g., References [25, 30]). In this section, we show that the size of maximum matching and, therefore, also the existence of a perfect matching, can be maintained in non-uniform DynFO. We recall that non-uniform DynFO programs can use initial databases that can depend on the size of the universe in an arbitrary, even non-computable way.

[10] The set of labels actually need 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.

To this end, we reuse the techniques developed in Section 4 for maintaining the rank of a matrix. In previous work, a non-uniform dynamic program with TC^0-updates has been obtained for both problems [6]. It remains open whether (maximum or perfect) matching can be maintained in uniform DynFO.

The basic idea of our approach relies on a correspondence between the rank of the Tutte matrix of a graph and the size of maximum matchings. The *Tutte matrix* T_G of an undirected graph G is the $n \times n$ matrix with entries

$$t_{ij} = \begin{cases} x_{ij} & \text{if } (i,j) \in E \text{ and } i < j \\ -x_{ji} & \text{if } (i,j) \in E \text{ and } i > j, \\ 0 & \text{if } (i,j) \notin E \end{cases}$$

where the x_{ij} are indeterminates.[11]

Theorem 10 (Lovász [29]). *Let G be a graph with a maximum matching of size m. Then $rank(T_G) = 2m$.*

Unfortunately, the rank maintenance algorithm presented in Section 4.3 cannot be applied immediately as the entries of T_G are indeterminates, and applying the maintenance algorithm to a matrix with indeterminates might yield polynomials with exponentially many terms. However, the rank of T_G can be determined by computing the rank for a matrix obtained by replacing the indeterminates in T_G by well-chosen positive integer values. For a graph G, let w be a function that assigns a positive integer weight to every edge (i,j) and let $B_{G,w}$ be the integer matrix obtained from T_G by substituting x_{ij} by $2^{w(i,j)}$.

Theorem 11. *If G is a graph with a maximum matching of size m and w is a weight assignment for the edges of G, then $rank(B_{G,w}) \leq 2m$. Furthermore, if G has a maximum matching with unique minimal weight with respect to w, then $rank(B_{G,w}) = 2m$.*

This theorem is implicit in Lemma 4.1 in Reference [20]. For the sake of completeness, we give a full proof here. The proof uses the following theorem.

Theorem 12 (Mulmuley et al. [33]). *Let G be a graph with a perfect matching and w a weight assignment such that G has a unique perfect matching with minimal weight with respect to w. Then $\det(B_{G,w}) \neq 0$.*

Proof (of Theorem 11). Recall that the rank of a matrix can be defined as the size of the largest submatrix with non-zero determinant. Thus, $\text{rank}(B_{G,w}) \leq \text{rank}(T_G)$ and, therefore, $\text{rank}(B_{G,w}) \leq 2m$ by Theorem 10.

For showing $\text{rank}(B_{G,w}) \geq 2m$ when G has a maximum matching of unique minimal weight with respect to w, we adapt the proof of Theorem 10 given in Reference [37]. Let U be the set of vertices contained in the unique maximum matching of G with minimal weight, and G' the subgraph of G induced by U. Observe that G' has a unique minimal weight *perfect* matching with respect to w. Restricting $B_{G,w}$ to rows and columns labeled by elements from U yields the matrix $B_{G',w'}$, where w' is the weighting w restricted to edges from G'. However, then $\det(B_{G',w'}) \neq 0$ by Theorem 12 and, therefore, $\text{rank}(B_{G,w}) \geq 2m$. □

Using the technique implicit in Reference [38] one can find, for every $n \in \mathbb{N}$, weighting functions $w_1, \ldots, w_{n^2}$ with weights in $[4n]$, such that for every graph G over $[n]$ there is an $i \in [n^2]$ such that G has a maximum matching with unique minimal weight with respect to w_i.

[11] The rank of a matrix with indeterminates can be defined as the size of the largest quadratic submatrix with non-zero determinant.

We show how to obtain those functions. The following lemma is due to Mulmuley et al. [33], but we use the version stated in Reference [23, Lemma 11.5].

Lemma 13 (Isolation Lemma). *Let $m, M \in \mathbb{N}$ and let $\mathcal{F} \subseteq 2^{[m]}$ be a non-empty set of subsets of $[m]$. If a weight function $w \in [M]^{[m]}$ is uniformly chosen at random, then with probability at least $1 - \frac{m}{M}$, the minimum weight subset in $\mathcal{F}$ is unique; where the weight of a subset $F \in \mathcal{F}$ is $\sum_{i \in F} w(i)$.*

Lemma 14 (Non-uniform Isolation Lemma, Implicit in Reference [38]). *Let $m \in \mathbb{N}$ and $\mathcal{F}_1, \ldots, \mathcal{F}_{2^m} \subseteq 2^{[m]}$. There are weight functions $w_1, \ldots, w_m$ from $[4m]^{[m]}$ such that for any $i \in [2^m]$ with $\mathcal{F}_i \neq \emptyset$, there exists a $j \in [m]$ such that the minimum weight subset of $\mathcal{F}_i$ with respect to w_j is unique.*

Proof. The proof is implicit in the proof of Lemma 2.1 in Reference [38]. We give a self-contained presentation.

We call a collection $u_1, \ldots, u_m$ of weight functions *bad* for some $\mathcal{F}_i$ if no $F \in \mathcal{F}_i$ is a minimum weight subset with respect to any u_j. For each $\mathcal{F}_i \neq \emptyset$, the probability of a randomly chosen weight sequence $U \stackrel{\text{def}}{=} u_1, \ldots, u_m$ to be bad is at most $(\frac{1}{4})^m$ thanks to Lemma 13 (for $M \stackrel{\text{def}}{=} 4m$). Thus, the probability that such a U is bad for *some* $\mathcal{F}_i$ is at most $2^m \times (\frac{1}{4})^m < 1$. Hence there exists a sequence U which is good for all $\mathcal{F}_i$. □

We immediately get the following corollary:

Corollary 15. *Let $G_1, \ldots, G_{2^{n^2}}$ be some enumeration[12] of the graphs on $[n]$ and let $\mathcal{F}_1, \ldots, \mathcal{F}_{2^{n^2}}$ be their respective sets of perfect matchings. There is a sequence $w_1, \ldots, w_{n^2}$ of weight assignments assigning a value from $[4n^2]$ to the edges of $[n]^2$ such that for every graph G over $[n]$ there is some $i \in [n^2]$ such that if G has a perfect matching then it also has a perfect matching with unique minimal weight with respect to w_i.*

Then, to maintain the size of maximum matchings of a graph G over $[n]$, it is sufficient to maintain $\operatorname{rank}(B_{G,w_i})$ for all $i \in [n^2]$. The rank of T_G is the maximal rank among those ranks thanks to Theorem 11.

Theorem 16. *PerfectMatching and MaxMatching are in non-uniform* DynFO.

Proof Sketch. It suffices to show that MaxMatching is in non-uniform DynFO. The idea is to advise a dynamic program with the weighting functions $w_1, \ldots, w_{n^2}$ that assign weights such that for all graphs with n nodes there is a maximum matching with unique weight. The advice is given to the dynamic program via the initialisation of the auxiliary relations. The program then maintains the ranks for the matrices B_{G,w_i} and outputs the maximal such rank. We make this more precise in the following.

Recall that the weighting functions assign values of up to $4n^2$ and that, therefore, the determinant of each B_{G,w_i} can be of size up to $n!(2^{4n^2})^n \leq 2^{5n^3}$, and thus it is sufficient to maintain the rank of those matrices modulo up to $5n^3$ many primes, which are contained in the first n^4 numbers by the prime number theorem.[13]

The dynamic program computes, for each of the weighting functions w_i and each prime $p \leq n^4$, the rank of B_{G,w_i} modulo p.

This informal description can be formalised by exhibiting a non-uniform bfo-tt reduction from MaxMatching to RankModp. Such reductions are defined as bfo-tt reductions but they can assume arbitrary additional relations on the structure. By an adaptation of the proof of Proposition 4,

[12] For notational simplicity we use n^2 instead of $\binom{n}{2}$ here.

[13] We disregard small values of n as the query can be directly encoded with first-order formulas for such values.

it can be shown that if there is a non-uniform bfo-tt reduction from a query q to a query q', then q is in non-uniform DynFO if $q' \in$ DynFO.

The non-uniform reduction $(\mathcal{J}, \varphi)$ from MaxMatching to RankModp is two-dimensional and 6-ary. Two dimensions are used to encode graphs as matrices as described in Section 2.2. For a parameter $(p_1, \ldots, p_6)$, the interpretation $\mathcal{J}$ maps a given graph to an instance of RankModp that asks for the rank of B_{G,w_i} mod p where w_i is encoded by the first two parameters and p is encoded by the remaining four parameters. For converting edges to entries of B_{G,w_i} mod p, the reduction uses non-uniform relations. The wrap-up formula determines the highest rank of all those instances. □

7 FIRST-ORDER INCREMENTAL EVALUATION SYSTEMS

The dynamic complexity framework was independently formalised in the form of *first-order incremental evaluation systems* (short: *FOIES*) by Dong et al. (see References [9, 12], and also References [10, 11]). Apart from notational differences, the DynFO-setting and FOIES differ in how they treat domains. While domains in DynFO are fixed before a dynamic computation starts, the FOIES-framework allows for the domain to grow and shrink. More precisely,[14] the domain of a state of a FOIES computation is the active domain; that is, the set of elements contained in some tuple of the input database. Thus, a tuple can be inserted that contains an element that is not contained in the current domain, and the domain is extended by this element. Likewise, when a tuple is removed, and one of its elements is not contained in any other tuple afterwards, then that element is removed from the domain.[15]

Thus, in FOIES, the notions of domain, active domain, and activated elements coincide, at any point in time. Yet, FOIES have an infinite background universe U and the domain D always satisfies $D \subset U$. We show next that reachability can also be maintained by FOIES programs.

Theorem 17. *If a query $Q \in$ DynFO(+,×) is domain-independent, then Q can be maintained by a FOIES-program.*

Proof Sketch. Let Q be a domain-independent query and $\mathcal{P}$ a DynFO(+,×) program that maintains Q. We assume that Q uses only one binary relation E, the adaptation for arbitrary structures is straightforward.

We will construct a FOIES program $\mathcal{P}'$ that simulates $\mathcal{P}$. To this end, we extend the construction used in the proof of Proposition 7.

The program $\mathcal{P}'$ handles changes to the (active) domain by using the simulation technique presented in the proof of Proposition 7. As before, the computations of $\mathcal{P}$ are split into stages and $\mathcal{P}'$ uses one thread for simulating $\mathcal{P}$, for each different stage. Here, the stages are based on the size of the (active) domain A. A computation of $\mathcal{P}$ is in stage i, if more than N_i but at most N_{i+1} elements are contained in A, where $N_i \stackrel{\text{def}}{=} 2^{2^i}$. The update program $\mathcal{P}'$ uses one thread per stage of $\mathcal{P}$, the i-th thread being responsible for providing the correct query result whenever more than N_i but at most N_{i+1} elements are in the domain. See Figure 3 for an illustration.

Two issues need to be addressed when constructing the FOIES-program $\mathcal{P}'$: (1) Thread i uses arithmetic on N_{i-1} elements. When one of those elements is removed from the domain, then the arithmetic has to be adapted. (2) In contrast to the construction for Proposition 7, $\mathcal{P}$ can enter a stage multiple times because elements can be inserted *and* removed from the domain.

The first issue can be easily resolved. Using Etessami's technique, the program $\mathcal{P}'$ maintains arithmetic relations $<$, $+$, and $\times$ on the domain. When the j-th element a of the domain is removed,

[14] We do not give a formal definition of FOIES, but only describe how they differ from DynFO-programs.
[15] In some papers, FOIES may use a bounded number of elements that are not used by an input tuple [10].

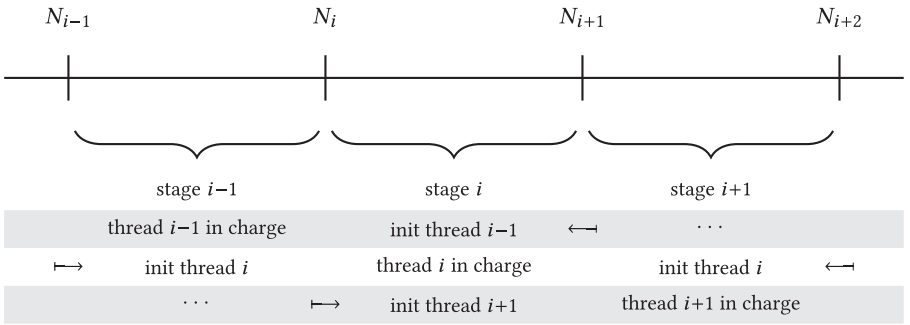

Fig. 3. Illustration of the stages used in the proof of Theorem 17.

then the arithmetic is adapted by replacing a by the current maximal element b with respect to $<$. To this end, first, all tuples containing b are removed from the auxiliary relations, and then every auxiliary tuple $\vec{d}$ containing a is substituted by the tuple $\vec{d}'$, obtained by replacing all occurrences of a in $\vec{d}$ by b. Note that after those substitutions the functions g_i are still bijections between 4-tuples over the first N_i elements of the domain and N_{i+2}.

We now address the second issue. Thread i shall provide the correct result whenever the domain contains more than N_i but at most N_{i+1} elements; also, when this interval has been reached by removing elements from the domain. To guarantee this, thread i is started either when the $(N_{i-1} + 1)$-th element is inserted or when the $(N_{i+2} + 1)$-th element is removed from the domain. It stops when the size of the domain is not in $\{(N_{i-1} + 1), \ldots, N_{i+2}\}$.

The way in which thread i acts is just as in the proof of Proposition 7. For each change operation δ, in addition to performing the updates for δ, the program $\mathcal{P}'$ inserts up to five tuples to $\widehat{E}_i$ and simulates $\mathcal{P}$ for these five insertion steps. This guarantees that the query result can be decoded from the query result of thread i, when the domain contains $n \in \{(N_i + 1), \ldots, N_{i+1}\}$ elements. This has been already shown for the case where thread i is started by the insertion of the $(N_{i-1} + 1)$-th element in the proof of Proposition 7. The argument for the case where the thread is started when the $(N_{i+2} + 1)$st element is removed from the domain is similar. At least $\frac{N_{i+2}-N_{i+1}}{2} = \frac{N_{i+1}^2-N_{i+1}}{2}$ edge deletions are needed to get from a domain of size N_{i+2} to a domain of size N_{i+1}. Thus, when arriving at an domain of size N_{i+1}, up to $2(N_{i+1}^2 - N_{i+1})$ edges are contained in $\widehat{E}_i$ since each deletion contributes four edges to $\widehat{E}_i$ as long as edges are missing in $\widehat{E}_i$ (only four, as one edge may be removed from $\widehat{E}_i$ by δ). Since there are at most $N_{i+1}^2 \geq 2(N_{i+1}^2 - N_{i+1})$ edges in a graph over an domain of size N_{i+1}, the edge transfer is completed before thread i is in charge. □

8 CONCLUSION AND FUTURE WORK

In this article, we showed that reachability can be maintained in DynFO and thereby confirmed the conjecture of Patnaik and Immerman [36]. The proof adapts and combines several known techniques in a surprisingly elementary way. One of the key ingredients, the maintainability of the rank of a matrix with first-order update formulas, is of independent interest. As an immediate consequence of those results, regular path queries and 2-satisfiability can also be maintained in DynFO. By combining the linear algebraic part of the proof that reachability is in DynFO with the Isolation Lemma, we showed how the size of a maximum matching can be maintained in DynFO with non-uniform initialisation.

Reachability is arguably one of the most important algorithmic problems in computer science, and algorithms for solving reachability are the basis for solving many other problems. For this

reason, the open status of the maintainability question for reachability has stifled progress in the study of descriptive dynamic complexity severely. The positive answer to this question raises hopes that other areas become accessible to the methods of dynamic complexity now. One example being query languages for graph databases, as is illustrated by the dynamic program for maintaining regular path queries. Other potential candidate areas are dynamic model checking and query evaluation under ontologies. However, the DynFO bound for reachability does not extend to all of NL, simply because DynFO is not known to be closed under *unbounded* first-order reductions.

Also, the basic techniques used for maintaining reachability are promising for further progress. Linear algebraic problems, such as the rank problem, have thus far been neglected in the study of dynamic complexity. Also, techniques known from the study of small static complexity classes have not been systematically tested in the dynamic framework. The application of the Isolation Lemma presented here indicates that this might be worthwhile.

We plan to explore those techniques by trying to apply them to related problems such as maintaining a reachability witness, the (shortest) distance, the number of paths, whether there is a matching (and witnesses for that), the value of the determinant, and disjoint paths.

Another interesting direction for future research is to explore whether the dynamic programs for maintaining reachability, rank, and the size of a maximum matching presented here can be generalised and optimised. We indicate some intriguing challenges in the following.

In this article, we only looked at modifications of a single tuple. A closer analysis reveals that the dynamic program for rank still works when $O(\log n)$ many entries of a column can be modified, and therefore reachability can be maintained when several incoming edges of one node can be modified at once (dually: all outgoing edges).[16] Whether reachability and rank can be maintained for other, more complex modifications remains open. We note that this is closely related to the question of which fragments of transitive closure logic can be maintained by first-order updates.

It would also be worthwhile to study whether reachability (as well as the other problems studied here) can be maintained in fragments of DynFO. Typical fragments limit the arity of the auxiliary relations or the syntactic shape of update formulas. The dynamic programs presented here have very high arity, which makes them hard to apply in practical scenarios. It remains open whether reachability can be maintained with auxiliary relations of small arity. So far, it is only known that reachability cannot be maintained using unary auxiliary relations. Another interesting question is whether reachability can be maintained by even weaker update mechanisms, e.g., NC^0updates. Lower bounds for this fragment are conceivable; yet, even for the quantifier-free fragment of DynFO, which corresponds to restricted NC^0 updates, lower bounds are nontrivial. It is only known that binary auxiliary relations are not sufficient to maintain reachability in this fragment of DynFO [46].

ACKNOWLEDGMENTS

We thank William Hesse for stimulating and illuminating discussions. We are grateful to an anonymous reviewer for pointing us to the dynamic algorithm for maintaining rank by Frandsen and Frandsen [14]. This simplified the presentation enormously. Further, we thank Nils Vortmeier for many very helpful comments on drafts of this article.

REFERENCES

[1] Serge Abiteboul and Victor Vianu. 1999. Regular path queries with constraints. *J. Comput. Syst. Sci.* 58, 3 (1999), 428–452. DOI: http://dx.doi.org/10.1006/jcss.1999.1627

[16] This observation is due to William Hesse.

[2] Marcelo Arenas, Sebastián Conca, and Jorge Pérez. 2012. Counting beyond a yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In *Proceedings of the 21st World Wide Web Conference (WWW'12)*. 629–638.

[3] Pablo Barceló Baeza. 2013. Querying graph databases. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'13)*. 175–188. DOI: http://dx.doi.org/10.1145/2463664.2465216

[4] 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

[5] Stephen A. Cook. 1985. A taxonomy of problems with fast parallel algorithms. *Inform. Control* 64, 1–3 (1985), 2–21. DOI: http://dx.doi.org/10.1016/S0019-9958(85)80041-3

[6] Samir Datta, William Hesse, and Raghav Kulkarni. 2014. Dynamic complexity of directed reachability and other problems. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP'14) (Part I)*. 356–367.

[7] Guozhu Dong, Leonid Libkin, Jianwen Su, and Limsoon Wong. 1999. Maintaining transitive closure of graphs in SQL. *Int. J. Inform. Technol.* 51, 1 (1999), 46.

[8] 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

[9] Guozhu Dong and Jianwen Su. 1993. First-order incremental evaluation of datalog queries. In *Proceedings of the 4th International Workshop on Database Programming Languages (DBPL'93) 1993*. 295–308.

[10] Guozhu Dong and Jianwen Su. 1997. Deterministic FOIES are strictly weaker. *Ann. Math. Artif. Intell.* 19, 1–2 (1997), 127–146. DOI: http://dx.doi.org/10.1023/A:1018951521198

[11] 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.

[12] Guozhu Dong and Rodney W. Topor. 1992. Incremental evaluation of datalog queries. In *Proceedings of the 4th International Conference on Database Theory (ICDT'92)*. 282–296. DOI: http://dx.doi.org/10.1007/3-540-56039-4_48

[13] Kousha Etessami. 1998. Dynamic tree isomorphism via first-order updates. In *Proceedings of the 7th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'98)*. 235–243. DOI: http://dx.doi.org/10.1145/275487.275514

[14] 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

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

[16] Erich Grädel and Sebastian Siebertz. 2012. Dynamic definability. In *Proceedings of the 15th International Conference on Database Theory (ICDT'12)*. 236–248. DOI: http://dx.doi.org/10.1145/2274576.2274601

[17] William Hesse. 2003. 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

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

[19] William Hesse and Neil Immerman. 2002. Complete problems for dynamic complexity classes. In *Proceedings of the 17th IEEE Symposium on Logic in Computer Science (LICS'02)*. 313.

[20] Thanh Minh Hoang. 2010. On the matching problem for special graph classes. In *Proceedings of the 25th Annual IEEE Conference on Computational Complexity (CCC'10)*. 139–150.

[21] Roger A. Horn and Charles R. Johnson. 2012. *Matrix Analysis*. Cambridge University Press.

[22] Neil Immerman. 1999. *Descriptive Complexity*. Springer. I–XVI, 1–268 pages.

[23] Stasys Jukna. 2001. *Extremal Combinatorics*, Vol. 2. Springer.

[24] Detlef Kähler and Thomas Wilke. 2003. Program complexity of dynamic LTL model checking. In *Proceedings of the 17th International Workshop on Computer Science Logic (CSL'03)*. 271–284. DOI: http://dx.doi.org/10.1007/978-3-540-45220-1_23

[25] Marek Karpinski and Wojciech Rytter. 1998. *Fast Parallel Algorithms for Graph Matching Problems*. Oxford University Press, Inc., New York.

[26] 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.

[27] Leonid Libkin. 2004. *Elements Of Finite Model Theory*, Texts in Theoretical Computer Science. An Eatcs Series. Springer-Verlag.

[28] Katja Losemann and Wim Martens. 2013. The complexity of regular expressions and property paths in SPARQL. *ACM Trans. Database Syst.* 38, 4 (2013), 24. DOI: http://dx.doi.org/10.1145/2494529

[29] László Lovász. 1979. On determinants, matchings, and random algorithms. In *Proceedings of the International Symposium of Fundamentals of Computation Theory (FCT) 1979*. 565–574.

[30] L. Lovász and M. D. Plummer. 1986. *Matching Theory*, Vol. 29. North-Holland Publishing Co.
[31] Alberto O. Mendelzon and Peter T. Wood. 1995. Finding regular simple paths in graph databases. *SIAM J. Comput.* 24, 6 (1995), 1235–1258.
[32] C. D. Meyer. 2000. *Matrix Analysis and Applied Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia.
[33] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. 1987. Matching is as easy as matrix inversion. *Combinatorica* 7, 1 (1987), 105–113. DOI: http://dx.doi.org/10.1007/BF02579206
[34] Pablo Muñoz, Nils Vortmeier, and Thomas Zeume. 2016. Dynamic graph queries. In *Proceedings of the 19th International Conference on Database Theory (ICDT'16)*. 14:1–14:18. DOI: http://dx.doi.org/10.4230/LIPIcs.ICDT.2016.14
[35] Sushant Patnaik and Neil Immerman. 1994. Dyn-FO: A parallel, dynamic complexity class. In *Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'94)*. 210–221.
[36] Sushant Patnaik and N. Immerman. 1997. Dyn-FO: A Parallel, dynamic complexity class. *J. Comput. Syst. Sci.* 55, 2 (Oct. 1997), 199–209.
[37] Michael O. Rabin and Vijay V. Vazirani. 1989. Maximum matchings in general graphs through randomization. *J. Algorithms* 10, 4 (1989), 557–567. DOI: http://dx.doi.org/10.1016/0196-6774(89)90005-9
[38] Klaus Reinhardt and Eric Allender. 2000. Making nondeterminism unambiguous. *SIAM J. Comput.* 29, 4 (2000), 1118–1131. DOI: http://dx.doi.org/10.1137/S0097539798339041
[39] J. Barkley Rosser and Lowell Schoenfeld. 1962. Approximate formulas for some functions of prime numbers. *Illinois J. Math.* 6, 1 (1962), 64–94.
[40] Piotr Sankowski. 2004. Dynamic transitive closure via dynamic matrix inverse (extended abstract). In *Proceedings of the 45th Symposium on Foundations of Computer Science (FOCS'04)*. 509–517. DOI: http://dx.doi.org/10.1109/FOCS.2004.25
[41] Piotr Sankowski. 2007. Faster dynamic matchings and vertex connectivity. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'07) 2007*. 118–126. http://dl.acm.org/citation.cfm?id=1283383.1283397.
[42] Thomas Schwentick and Thomas Zeume. 2016. Dynamic complexity: Recent updates. *SIGLOG News* 3, 2 (2016), 30–52. DOI: http://dx.doi.org/10.1145/2948896.2948899
[43] 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
[44] Peter T. Wood. 2012. Query languages for graph databases. *ACM SIGMOD Rec.* 41, 1 (2012), 50–60.
[45] Thomas Zeume. 2015. *Small Dynamic Complexity Classes*. Ph.D. Dissertation. TU Dortmund University.
[46] 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

Received September 2016; revised October 2017; accepted April 2018