Dynamic Expressibility under Complex Changes

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Nils Vortmeier

Dortmund
2019

Tag der mündlichen Prüfung: 20. November 2019

Dekan: Prof. Dr.-Ing. Gernot Fink

Gutachter:

Prof. Dr. Thomas Schwentick

Prof. Dr. Victor Vianu

Abstract

Whenever a database is changed, any previously computed and stored result of a query evaluation on that database becomes invalid and needs to be updated. This update can in principle be conducted by re-evaluating the query from scratch. However, this approach might be too inefficient, especially for large databases. Instead, it is often advantageous to make use of the old query result and possibly further stored auxiliary information.

A descriptive framework for studying this dynamic setting of query evaluation was introduced by Patnaik and Immerman in 1994, independently of a very similar formalisation by Dong, Su and Topor that was first published in 1992 and 1993. In this dynamic descriptive complexity framework, auxiliary relations are stored that hold the result of a fixed query as well as further auxiliary information. The update of these auxiliary relations after a change to the input database is specified by first-order formulas that may refer to the input database as well as to the stored auxiliary relations. The dynamic complexity class DynFO contains all queries for which the result can be maintained this way under some specified set of changes.

Most of the previous work in the DynFO framework focussed on changes that are insertions and deletions of single tuples. In this thesis we consider also more complex changes, in particular we study changes that insert or delete a set of tuples of bounded size, and changes that are specified by first-order definable queries on the input.

The main contribution of this thesis is an investigation of the expressive power of first-order update formulas in the DynFO framework with respect to these classes of complex changes. We show strong DynFO maintainability results under complex changes for queries such as the reachability query for directed and undirected graphs. We also give non-maintainability results for certain restricted dynamic settings.

Some of our maintainability results rely on a new technique of dynamic query maintenance. This technique is also used to show that all queries that are definable in monadic second-order logic are in DynFO under single-edge changes of input graphs of bounded treewidth.

We experimentally evaluate the dynamic approach and compare the performance of query evaluation from scratch with the performance of different strategies of dynamic query maintenance under complex changes.

Acknowledgements

When I started my Bachelor studies in Computer Science, I thought I would end up doing robotics. Then I took Thomas Schwentick's logics course, afterwards basically every other course he offered, and now I try to obtain a doctorate working on descriptive complexity theory. How plans can change!

Thomas accompanied me during my whole studies, and his enthusiasm and dedication for research and teaching impressed me deeply. I am indebted to him for his guidance, encouragement and constant support. During my time as a PhD student I always felt his trust, and that he is convinced I can actually finish this project, even when I was not convinced myself. I thank him for helpful criticism, honest praise, and occasional visits to the stadium. He really deserves the title *Doktorvater*.

This thesis would not exist without Thomas Zeume. Literally everything I did in research is joint work with him, and our tea breaks were the place where many ideas took shape. He is to me collaborator, advisor, critic, and a friend. Thank you very much for introducing me to the fun side of research!

The visits in Chennai were one of the most productive times during my PhD studies. I am thankful to Samir Datta for hosting Thomas and me, and for sharing his many ideas. Many thanks to him and to Anish Mukherjee for a very successful collaboration! I also thank my colleagues at Lehrstuhl 1 in Dortmund for the great teamwork, many inspirations, a delightful time, and that they always waited for me at lunch. Special thanks to Gaetano for being funny sometimes!

I am thankful and honoured that Victor Vianu agreed to review this thesis and to be part of my examination committee. I acknowledge the financial support by Deutsche Forschungsgemeinschaft under grant SCHW 678/6-2.

Many people distracted me from research and kept me kind of sane. I thank my friends for the time together: Christopher, especially for the wonderful journeys, the members of ProStuKo and Team Spieleabend, the inhabitants of Belgarien, the ASG-Helden, and all those people from Frohlinde, the MAR, Frohlinder Hopfenkinder, and Callo-Theater.

Much of what we achieve in life depends on the roots we have. I thank my family, especially my mother and my grandparents, for their love and support. Thank you for enabling me to do all this.

Contents

CHAPTER 1

Introduction

In our usual understanding, an algorithm works in three (potentially intertwined) stages: it reads its input, computes the corresponding result, which it then outputs. This way it can process a series of various unrelated inputs one after the other.

This model is often not suitable for today's data management systems, which continuously need to solve some computational problems on evolving data sets. Social networks analyse the community structure and display information based on privacy settings and user relationships, while the users frequently change their connections to other users. Web search engines and other recommender systems face a changing set of items as well as new information on their users. Routing information, for private and public transport, network traffic and other applications, needs to be updated if links are faulty, fail altogether, or are newly established.

In principle, all these challenges can of course be handled by discarding previously computed data and applying a standard algorithm after every modification of the input. This *re-computation from scratch* might however not be possible due to performance reasons, especially if the input size is huge; often it is also not necessary: small changes in a traffic network will probably only trigger very local adaptations of the routing information, and whether two users are "friends" in a social network might only affect them and their other friends. It is therefore plausible that it is sufficient to *update* a previously computed result locally to incorporate small changes of the input, and that such an update is computationally cheaper than re-computing the result from scratch.

A routine that performs such an update naturally has access to the previously computed result, the former input and its modified part. Additionally, it may use further stored *auxiliary* information if this facilitates the update. Of course, this information then needs to be updated as well. If such an update routine can continuously provide the correct answer to some query on a changing input, we say that it *maintains* this query.

There are (at least) two approaches on how such an update routine can be specified. Following the *dynamic algorithms approach*, specialised data structures are stored as auxiliary information and the goal is to design algorithms that need less time to update a result than to re-compute it from scratch. See [Demetrescu et al. 2010] for an overview on upper bounds in this field, and for example [Henzinger and Fredman 1998; Miltersen 1999; Abboud and Williams 2014] for lower bounds in different settings.

In this thesis we study the *descriptive approach*, which is mainly motivated from database applications, as discussed further below. Within this approach, the update of a query result and of the auxiliary information is declaratively defined by a set of logical formulas. We are particularly interested in updates that are specified via first-order formulas and variants thereof. Consequently, the input, the query result and the auxiliary information are encoded by relational structures, that is, plain relations over some underlying domain. The complexity measure we are interested in is, above all, the precise logic we need to express updates that are able to maintain a query result.

The dynamic descriptive complexity framework Very similar formal frameworks for studying the descriptive approach, in particular with updates that are expressible in first-order logic, were proposed independently of each other by Dong, Su and Topor [Dong and Topor 1992; Dong and Su 1993; Dong et al. 1995] and Patnaik and Immerman [Patnaik and Immerman 1994; 1997] under the names *first-order incremental evaluation system* (FOIES) and Dyn-FO, respectively.

The formalisation we use in this thesis is based on the framework of Patnaik and Immerman. In this *dynamic descriptive complexity* framework, the goal of a *dynamic program* is to maintain a fixed query under changes of the input. Such a dynamic program consists of a set of first-order *update formulas*, which define the update of a *query relation* that holds the result of the query on the input, as well as of additionally stored *auxiliary relations*. The queries that can be maintained this way constitute the dynamic complexity class DynFO.

Example 1.1. We give an example to strengthen our intuition about the descriptive approach. The *reachability query* REACH that maps a directed graph $G = (V, E)$ with node set V and edge set E to its transitive closure relation is a fundamental query that cannot be expressed by a first-order formula, see for example [Immerman 1999]. Here we present the well-known result that it can be maintained using first-order update formulas upon insertions of single edges into the graph.

Assume that in addition to the graph G a relation T is stored that equals the transitive closure of G, that is, the set of all pairs (u, v) of nodes such that there is a directed path from u to v in G. When an edge (a, b) is inserted into the graph G, forming the graph $G' = (V, E \cup \{(a, b)\})$, the updated transitive closure relation T' can be defined as follows. Firstly, all paths that exist in G continue to exist in G', and therefore $T \subseteq T'$ must hold. Secondly, every path in G' from some node u to some node v that does not already exist in G needs to use the edge (a, b). We can assume without loss of generality that such a path uses this edge only once, and conclude that the sub-paths from u to a and from b to v already exist in G.

So, a pair (u, v) is contained in T' if and only if it satisfies the first-order formula $T(u, v) \vee \big(T(u, a) \wedge T(b, v)\big)$. This formula is therefore an update formula for the query relation T with respect to changes that insert a single edge into the input graph. $\lhd$

We give two justifications for the choice of first-order logic as the means of expressing an update. As already mentioned above, the arguably strongest motivation originates from database theory. Databases are prime examples of objects that are subject to ongoing sequences of changes. The DynFO framework is particularly relevant for *relational* databases, which can me modelled as relational structures. As queries on relational databases are usually expressed in terms of the declarative query language SQL, relational database management systems support the evaluation of SQL queries and are heavily optimised to process them efficiently. The "core" of SQL corresponds to the relational algebra, which itself corresponds to first-order logic, see [Abiteboul et al. 1995]. So, a relational database management system can naturally implement first-order update formulas to update a query, and it can rely on its already existing optimiser for computing the updates quickly.

We continue Example 1.1 to illustrate this connection.

Example 1.1 (continuing from p. 2)**.** Assume that the transitive closure relation T of the directed graph G is stored in a database as a binary table `T` with attributes `u` and `v`. The update formula $T(u, v) \vee \big(T(u, a) \wedge T(b, v)\big)$ can be translated into the following SQL query, which defines the updated relation T' after the insertion of the edge (a, b) into G.

```
SELECT *
FROM   T
  UNION
SELECT T1.u, T2.v
FROM   T as T1, T as T2
WHERE  T1.v = a AND T2.u = b;
```

$\lhd$

We see from Example 1.1 that the dynamic approach enhances the expressive power of core-SQL in the dynamic setting: one can maintain the transitive closure of a binary relation if single tuples are inserted into the base relation. This is even possible if also deletions of single tuples are allowed [Datta et al. 2018a]. In the static setting, the relational algebra cannot express the transitive closure, as it cannot be expressed in first-order logic, see [Immerman 1999]. This inability actually carries over to extensions of the relational algebra that allow for aggregation, grouping and arithmetic, the most important features that distinguish SQL from the relational algebra [Libkin 2003].

Note that the SQL:1999 standard has introduced features that enable SQL to express the transitive closure of a relation and other recursively defined queries [Eisenberg and Melton 1999]. These features need customised optimisation techniques and implementation details differ among different database systems [Przymus et al. 2010; Shalygina and Novikov 2017]. The declarative dynamic approach provides an alternative way to express reachability and other recursively and non-recursively defined queries, using the core functionalities of a database management system.

We now discuss another motivation for the use of first-order definable updates. The algorithmic approach as mentioned above asks for algorithms that quickly update a previously computed query result. The algorithms usually considered there are sequential. With the widespread use of multi- and many-core processors, parallel algorithms become more and more important. In complexity theory, circuits are a widely used model for parallel computation, and first-order logic (equipped with a linear order as well as addition and multiplication relations) naturally corresponds to the circuit complexity class uniform AC^0 [Barrington et al. 1990]. This class is defined via constant-depth circuits that may use polynomially many "not"-, "or"- and "and"-gates, where "or"- and "and"-gates may have arbitrary fan-in. So, a first-order formula can be evaluated in constant time by a set of polynomially many processors working in parallel. Although this model of computation can hardly be considered practicable, we can conclude that first-order update formulas can be processed in a highly parallel fashion, and may provide insights on how one can efficiently update a query result in a parallel setting.

So far we have, rather informally, introduced the class DynFO and we have motivated the use of first-order formulas for expressing an update. However, we have not discussed in detail which kind of changes the update formulas need to be able to process. Most prior publications on the dynamic complexity framework focus on the restricted case of changes that either insert a single tuple into an input relation or delete a single tuple from such a relation. A primary goal of this thesis is to study query maintainability under classes of more *complex* changes that cannot be characterised by insertions and deletions of single (or constantly many[1]) tuples.

There are several kinds of complex changes one can consider. Note that *arbitrary changes* are not suitable for our dynamic setting, as routines that update a query result basically need to re-compute it from scratch when input relations can be replaced arbitrarily. We give some examples for more restricted complex changes. First, one can demand that the set of changed tuples satisfies some *structural property*, as for example that deleted edges are in different connected components of a graph[2]. Second, one can consider changes that are *declaratively definable*, for example as the result of a first-order definable query on the input. The study of these changes has been mentioned as an open problem by several authors [Patnaik and Immerman 1997; Etessami 1998; Grädel and Siebertz 2012; Zeume 2015]. Third, one can impose *quantitative restrictions* and demand that the number of changed tuples is bounded by some non-constant function in (for example) the input size.

Before we formulate the research goals of this thesis in detail and present its contributions, we first review previous results in dynamic complexity.

Previous work in dynamic descriptive complexity We give an overview on previous work concerning the DynFO framework. More detailed lists of references on specific

[1]If a constant number c of tuples is affected by a change, one can usually number them as the first, second, ..., c-th affected tuple. If the query at hand can be maintained by first-order formulas under single-tuple changes, these formulas can then be composed into first-order update formulas that maintain the query also under changes of c tuples at a time.

[2]A similar condition is considered in [Dong and Pang 1997]; we give more details below.

aspects of dynamic complexity will also be given in later chapters.

The query that is most studied in the dynamic complexity framework is the reachability query. It has been shown early that this query can be maintained by first-order update formulas for undirected [Patnaik and Immerman 1997; Dong and Su 1998; Grädel and Siebertz 2012] and for acyclic directed graphs [Dong and Su 1995] under single-edge changes. Quantifier-free update formulas are sufficient to maintain reachability for deterministic graphs [Hesse 2003b]. A series of results has been obtained for the reachability query in arbitrary directed graphs. This query can be maintained using uniform TC^0 update circuits [Hesse 2003a] and non-uniform $\mathsf{AC}^0[2]$ circuits [Datta et al. 2014] under single-edge changes. These circuit models are defined similarly as AC^0 circuits but additionally allow for "majority"-gates as well as "parity"-gates, respectively. Finally, Datta, Kulkarni, Mukherjee, Schwentick, and Zeume [2018a] have proven that the reachability query is in DynFO under insertions and deletions of single edges.

Other queries that can be maintained with first-order update formulas under changes of single tuples include the question whether two trees are isomorphic [Etessami 1998] and whether a string belongs to a fixed regular or context-free language [Patnaik and Immerman 1997; Hesse 2003b; Gelade et al. 2012].

Some work has been done to prove inexpressibility results for DynFO. Unfortunately, so far there is no general result that some query is not in DynFO under insertions and deletions of single tuples—apart from those that follow from the easy observation that all queries that are dynamically maintainable by first-order update formulas can be evaluated statically in polynomial time. Prior publications have however obtained inexpressibility results under single-tuple changes for restricted settings. Dong and Su [1998] restrict the arity of additionally stored auxiliary relations and show amongst other things that the reachability query cannot be maintained using only unary auxiliary relations (apart from the binary transitive closure relation). This result holds even when update formulas may use grouping and aggregation features from SQL that go beyond the relational algebra [Dong et al. 2003]. The tree isomorphism query cannot be maintained in a setting that demands that initial auxiliary relations need to be defined in some logic for arbitrary initial input structures [Grädel and Siebertz 2012]. If update formulas may not use quantifiers, one cannot maintain the alternating reachability query or any non-regular language [Gelade et al. 2012], and if additionally the initialisation is restricted or auxiliary relations are at most binary, neither can the reachability query [Zeume and Schwentick 2015]. If the auxiliary relations need to functionally depend on the input structure, then one cannot maintain whether two unordered sets are of equal size [Dong and Su 1997].

Other aspects that are studied concern the relationship of dynamic complexity classes [Zeume and Schwentick 2017], the question whether the expressive power of update formulas increases when auxiliary relations of higher arity are allowed [Dong and Su 1998; Dong and Zhang 2000; Zeume 2017], and problems that are complete for dynamic complexity classes [Hesse and Immerman 2002; Weber and Schwentick 2007].

Some work has been done on changes that go beyond single-tuple changes. Frameworks that incorporate more general changes are introduced in [Hesse and Immerman 2002; Weber and Schwentick 2007], although in these papers the actual results only concern changes that each modify the input relations by some constant number of tuples. In

[Dong et al. 1995], a class of queries including REACH is studied under insertions of *cartesian-closed* sets of tuples. A set D of binary tuples is called cartesian-closed if for each pair (a_1, b_1), (a_2, b_2) of tuples in D it also includes the tuples (a_1, b_2) and (a_2, b_1). The reachability query can be maintained in DynFO under deletions of sets of edges (nodes, strongly connected components) under the condition that for no pair e_1, e_2 of (not necessarily distinct) deleted edges (nodes, strongly connected components) there is a non-empty path from e_1 to e_2 in the original graph [Dong and Pang 1997]. Multiple simultaneous edge contractions are considered in [Siebertz 2011].

Research questions for this thesis We think that extensions of the DynFO framework that allow changes beyond single-tuple changes deserve a more thorough investigation. Such extensions are particularly motivated by practical applications: database management systems provide features to modify several rows of a table at once, power outages lead to the sudden unavailability of multiple network servers, and in large social networks many users edit their relationships at the same time. While in principle these changes can be processed as a series of insertions and deletions of single tuples, it is plausible to assume that it is more efficient to handle them directly.

We therefore aim to study the ability of dynamic programs and in particular first-order update formulas to maintain queries under classes of complex changes. A primary goal is to establish maintainability results.

Goal 1.1. Identify classes of complex changes such that DynFO maintainability results can be proven for these changes.

It is very hard to show that a query cannot be maintained dynamically, as witnessed by the lack of non-trivial general inexpressibility results for DynFO. Following the approach of previous research, we also aim for corresponding results in restricted dynamic settings.

Goal 1.2. Prove that specific queries cannot be maintained by (restricted) first-order dynamic programs under certain complex changes.

Many conceivable classes of complex changes include insertions and deletions of single tuples as special cases, and maintainability under those complex changes implies maintainability under single-tuple changes. It is therefore a reasonable first step to establish that some query can be maintained dynamically under single-tuple changes before considering more general changes. We therefore aim at providing also results of this kind, in particular for queries that have not been studied before in our dynamic setting.

Goal 1.3. Show DynFO maintainability results under single-edge changes for specific queries.

One motivation for studying complex changes mentioned above is that it might be more efficient to process complex changes directly than to handle them as a series of single-tuple changes. Of course, such a claim should be checked. We therefore set the goal of evaluating the direct treatment of complex changes in contrast to processing them as a sequence of single-tuple changes.

Goal 1.4. Experimentally evaluate the performance of first-order dynamic programs for single-tuple and complex changes.

Contributions We now summarise the contributions of this work and link them to the research goals we just stated.

This thesis is the first comprehensive study of complex changes within the dynamic complexity framework. In particular, we extend the DynFO framework towards first-order definable changes and bulk changes of a set of tuples whose number is bounded by some function in the size of the input, and study to what extent queries can be maintained by first-order update formulas under these classes of changes.

We prove surprisingly strong DynFO maintainability results under both of the major classes of complex changes we consider, as aimed for by Goal 1.1: first-order update formulas can maintain many interesting queries under non-trivial changes that modify a non-constant number of tuples. Most of the results we prove concern variants of the reachability query and the membership problem for regular and context-free languages, that is, queries for which maintainability under single-tuple changes is well understood.

More precisely, we show that reachability in undirected graphs can be maintained under changes that insert a set of edges that is defined by a first-order query on the graph. Reachability in directed acyclic graphs is in DynFO under a syntactically restricted set of first-order definable insertions of edges. We further analyse the dynamic programs for undirected reachability under different classes of first-order insertions. We identify natural and practically important fragments, as for example insertions that are defined by conjunctive queries, such that the resulting update formulas are "small". The undirected reachability query can also be maintained under insertions and deletions of a polylogarithmic number of edges with respect to the number of nodes.

A similar result is shown for the membership problem for regular languages: these problems can be maintained under changes of a polylogarithmic number of position of a string. If a change can be described by a first-order formula with restricted quantifier prefix, then also context-free languages can be maintained.

We also consider queries that have not been studied before in the DynFO framework: all queries from the class uniform AC^1, which is defined via circuits of logarithmic depth, can be maintained under first-order definable changes that do not use parameters. This weak class of changes does not include single-tuple changes.

In summary, we find that first-order update formulas are still very expressive under complex changes. Still, we are able to prove that some queries cannot be maintained with restricted dynamic programs if certain complex changes are allowed, as aimed for by Goal 1.2. For example, we show that the reachability query cannot be maintained using quantifier-free update formulas under changes that are defined by quantifier-free first-order formulas. The corresponding result is not known for single-edge changes, so our result supports the intuition that non-maintainability results should be easier to prove for stronger classes of changes.

We further show that unary auxiliary relations are not sufficient to maintain REACH with full first-order update formulas even if only very simple first-order definable insertions are allowed, although this is possible under single-edge insertions. We observe that using first-order update formulas we cannot hope for meaningful maintainability result under bulk changes of more than a polylogarithmic number of tuples with respect to the number of elements.

Although many questions remain open, these results provide a good first overview on how reachability and formal languages can be maintained under complex changes. We also study further queries that have not been considered so far in the dynamic setting, and start by providing maintainability results under single-tuple changes. Towards Goal 1.3 we show that all queries definable in monadic second-order logic MSO, the extension of first-order logic by set quantification, are in DynFO under single-tuple changes, as long as the input structure has bounded treewidth. This result is also extended to certain MSO-definable optimisation problems. It follows that one can for example maintain a minimal vertex cover for a graph with bounded treewidth, or whether such a graph is 3-colourable. To prove these results we introduce the so-called *Muddling technique*. This technique allows to obtain dynamic programs that maintain a query under arbitrary sequences of single-tuple changes from dynamic programs that do so only for a sequence of changes of bounded length. Variants of this technique are also used to prove results for complex changes.

The previously mentioned result shows maintainability in DynFO for a large class of queries at once. However, so far we have not identified any natural static complexity class above AC^0 such that all queries from this class can be maintained dynamically in DynFO under single-tuple changes. Here we present an example query with low static complexity which we so far cannot maintain dynamically, and thus forms a road-block for proving a result in that fashion. Variants of this query are used to show that dynamic programs with quantifier-free update formulas exhibit an arity hierarchy for Boolean graph queries: the expressive power of these dynamic programs increases with the arity of the auxiliary relations they may use.

The results we obtain in this thesis underscore that the DynFO framework forms a powerful theoretical approach to dynamic query maintenance. We also give an empirical evaluation of this approach, using the example of the reachability query for undirected graphs under insertions of first-order definable sets of edges. Towards Goal 1.4 we compare prototypical implementations of dynamic programs for complex and single-edge insertions as well as re-computation from scratch. We conduct three experiments and identify strengths and weaknesses of the approaches in the different scenarios. The competitive performance of the dynamic programs provides further motivation to study the DynFO framework, in particular with respect to complex changes.

Further related work We have already mentioned above related work concerning the DynFO (and FOIES) framework directly. We now discuss work in closely related domains.

In databases, *views* are derived relations that are defined as the result of a query on the base relations. Database systems may *materialise* views to allow faster access to their content. The goal of *incremental view maintenance* is to update such materialised views after changes of a base relation, see [Gupta and Mumick 1999] for an overview of this topic. Incremental view maintenance usually follows the algorithmic approach in the sense that the goal is to optimise the update time and that the update routines may be specified using general-purpose programming languages. Often, no additional auxiliary relations are used, or the only auxiliary relations are results of sub-queries of the query that defines the view. Besides recursively defined views, it also studies views that are definable in relational algebra; due to the correspondence with first-order logic, see [Abiteboul et al. 1995], these views can trivially be updated using first-order formulas.

Within the research area of incremental view maintenance, Blakeley et al. [1989] investigate whether a materialised view actually needs to be updated after a change. They give sufficient and necessary conditions for whether a change of the base relations is relevant for a view that is defined by a restricted relational algebra expression.

An algebraic approach on incremental view maintenance is studied in [Koch 2010], where a query language is presented that subsumes the relational algebra and allows for aggregates. It is shown that low-complexity update routines can be computed from view definitions from that language. Reports on implementations of this algebraic approach and experiments using changes of single tuples and sets of tuples are published as [Koch et al. 2014; Nikolic et al. 2016].

Recent work studies the question whether one can update in constant time data structures that can be used to enumerate with constant delay all result tuples of a query after single-tuple changes to the database. This was confirmed for several classes of queries, in particular for restricted classes of conjunctive queries, see e.g. [Berkholz et al. 2017; 2018a;b; Idris et al. 2017; 2018]. Further, one can enumerate the answer tuples of an MSO-definable query on trees with constant delay, while the stored data structure can be updated in logarithmic time [Amarilli et al. 2019]. It is shown that no such algorithm with constant update time exists. The number of triangles in a database can be updated in amortised time $\mathcal{O}(\sqrt{n})$, where n is the size of the database, after single-tuple changes [Kara et al. 2019]. A matching conditional lower bound is provided in the latter work.

Incremental techniques have also been studied in the context of semi-structured databases. Algorithms that incrementally maintain whether an XML document satisfies a Boolean XPath query, for different XPath fragments, are studied in [Björklund et al. 2010]. Dynamic algorithms that continuously validate an XML document against a specification are presented and evaluated in [Balmin et al. 2004; Barbosa et al. 2004].

A complexity theoretic framework for dynamic computations is introduced by Miltersen et al. [1994]. They define dynamic complexity classes *incr*-$\mathcal{C}$ based on static classes $\mathcal{C}$. For example, a problem is in the class *incr*-POLYLOGTIME if initial data structures for an input can be computed in polynomial time, and a RAM machine can update those data structures in polylogarithmic time after single-bit changes of the input. Miltersen et al. [1994] further introduce a concept of reductions that is also appropriate for dynamic complexity classes. They show for many problems that are complete for static complexity classes under LOGSPACE reductions that they are also complete under these reductions. The reductions used in the DynFO framework, see [Patnaik and Immerman 1997], use similar ideas as those introduced in [Miltersen et al. 1994].

The IES(SQL) framework of Libkin and Wong [1997] is an extension of the FOIES framework and thus closely related to our setting. It differs primarily in two aspects.

First, it uses an SQL-like update formalism, extending first-order logic with aggregation and grouping. Second, by means of aggregation and arithmetic expressions it can generate large number constants that can then be stored in the auxiliary database, although these constants are not included in the input database. Consequently, the size of the auxiliary database is not polynomially bounded in the size of the input database. In fact, the result of [Libkin and Wong 1997; Dong et al. 1999] that the reachability query for arbitrary graphs can be maintained in IES(SQL) uses auxiliary relations of potentially exponential size, as representations for all paths of the input graph are stored. Libkin and Wong [1997] show that the expressive power of IES(SQL) stays the same when only at most binary auxiliary relations are allowed. This is in contrast to the DynFO and FOIES frameworks, which both exhibit an arity hierarchy: for each $k > 0$ there are queries that can be maintained using k-ary auxiliary relations, but not using $(k-1)$-ary auxiliary relations [Dong and Su 1998].

We also refer to [Hesse 2003b] and [Zeume 2015] for further discussions of work that is related to the DynFO framework.

Outline We fix our notation, repeat necessary background and formally introduce and discuss the DynFO framework in Chapter 2. In Chapter 3 we present the Muddling technique, which will be utilised in the following chapters. Our results regarding single-tuple changes are given in Chapter 4, first-order definable and size-bounded bulk changes are defined and the corresponding results are proven in Chapter 5 and Chapter 6, respectively. We report on the experimental evaluation in Chapter 7. We conclude in Chapter 8.

Apart from the conclusion, the chapters start with an introduction that lists motivation, goals and contributions of the corresponding chapter. They close with a short outlook and with bibliographic remarks on where the presented results were published before, and who contributed to them.

CHAPTER 2

The dynamic setting

In this chapter we introduce the dynamic complexity framework as we use it in this thesis, together with several examples. Before we can do that, we repeat the necessary background, mainly from finite model theory.

Our definition of DynFO is a variation of the definitions given by Patnaik and Immerman [1997], and we discuss the differences and further variants of the framework later in this chapter. We conclude the chapter with a short outlook and further bibliographic remarks.

2.1 Preliminaries

We now fix our notation and present notions and fundamental results that are used throughout this thesis. More standard definitions will be repeated in later chapters. Nevertheless, we assume familiarity with the basic concepts of finite model theory and database theory and refer to standard texts for details [Abiteboul et al. 1995; Libkin 2004].

Basic notation We write $\mathbb{N}$ for the set of natural numbers, including 0. We denote by $[n]$ the set $\{1, \ldots, n\}$, for some $n \in \mathbb{N}$, and let $[n]_0 \overset{\text{def}}{=} \{0, \ldots, n\}$. For some set A, the set $A^k \overset{\text{def}}{=} \{\bar{a} = (a_1, \ldots, a_k) \mid a_1, \ldots, a_k \in A\}$ contains all k-tuples over A.

The logarithm of n to base 2 is denoted as $\log n$. The proofs in this thesis often assume that the logarithm is only applied to powers of 2, so, that $\log n$ always is a natural number. They can easily be adapted for the general case, then considering the number $\lceil \log n \rceil$ or $\lfloor \log n \rfloor$, depending on the context. We write $\log^d n$ for the polylogarithmic function $(\log n)^d$.

Structures, databases and queries We exclusively consider finite *relational structures* over *relational schemas* $\sigma = \{R_1, \dots, R_\ell, c_1, \dots, c_m\}$, where each R_i is a relation symbol with a corresponding *arity* $\mathrm{Ar}(R_i) \in \mathbb{N}$, and each c_j is a constant symbol. A *σ-structure* $\mathcal{D} = (D, R_1^{\mathcal{D}}, \dots, R_\ell^{\mathcal{D}}, c_1^{\mathcal{D}}, \dots, c_m^{\mathcal{D}})$ consists of a finite *domain* D, also called the *universe* of $\mathcal{D}$, relations $R_i^{\mathcal{D}} \subseteq D^{\mathrm{Ar}(R_i)}$ and constants $c_j^{\mathcal{D}} \in D$, for each $i \in \{1, \dots, \ell\}$ and each $j \in \{1, \dots, m\}$. We usually omit the superscripts and write, for example, R_1 instead of $R_1^{\mathcal{D}}$.

We use the term *(relational) σ-database* synonymously with the term σ-structure, and mainly in contexts that are more motivated by database theory applications.

The *active domain* $\mathrm{adom}(\mathcal{D})$ of a structure $\mathcal{D}$ contains all elements used in some tuple or as some constant of $\mathcal{D}$. For a set $D' \subseteq D$ and a relation R, the *restriction* $R[D']$ of R to D' is the relation $R \cap D'^{\mathrm{Ar}(R)}$. The structure $\mathcal{D}[D']$ *induced* by D' is the structure obtained from $\mathcal{D}$ by restricting the domain and all relations to D', where we assume that D' contains all constants of $\mathcal{D}$.

A k-ary *query* q on σ-structures, for some schema σ, maps each σ-structure with some domain D to a subset of D^k, and commutes with isomorphisms. For a 0-ary query we use the terms *Boolean query*, *(decision) problem* and *language* synonymously.

In this thesis we often consider the Boolean query PARITY, which is a query on structures with a unary relation U and asks whether $|U|$ is odd.

The most important class of structures we consider are *graphs*, that is, structures over the schema with a single binary relation symbol E. The domain V of a graph $G = (V, E)$ is its set of nodes, and the edge set E is an arbitrary subset of $V \times V$. Formally, all graphs in this thesis are *directed*. The *undirected graph* induced by G is the graph with the same set of nodes and edges $\{(u, v), (v, u) \mid (u, v) \in E\}$. A *path* from $u \in V$ to $v \in V$ is a sequence $u = u_0, u_1, \dots, u_n = v$ of pairwise distinct nodes from V such that $(u_i, u_{i+1}) \in E$ for all $0 \leq i < n$. An *undirected path* in G is a path in its induced undirected graph. A *(strongly) connected component* of G is a maximal connected subgraph H, i.e., for all nodes u and v of H, there is a path from u to v. A *weakly connected component* of G is a maximal subgraph whose induced undirected graph is connected.

If the formal representation does not matter, we usually speak directly of undirected graphs and their connected components.

The *reachability query* REACH selects, given a (directed) graph G, all pairs (u, v) such that there is a path from u to v in G. Similarly, the *undirected reachability query* UREACH selects (u, v) if there is an undirected path from u to v.

Logics Formulas from *first-order logic* FO over some schema σ are inductively defined as follows. Atomic formulas are of the form $R(t_1, \dots, t_k)$ or $t_1 = t_2$, where $R \in \sigma$ is a k-ary relation symbol and each t_i is either a constant symbol from σ or a variable. Formulas can be composed using the Boolean connectives $\neg, \wedge, \vee$ as well as first-order quantification $\exists x$ and $\forall x$. We use abbreviations $\rightarrow, \leftrightarrow$ as usual.

The *quantifier rank* of an FO formula is the maximal nesting depth of quantifiers in its syntax tree. An FO formula φ is a $Q_1 \cdots Q_\ell$FO formula, where each Q_i is either $\exists$ or $\forall$, if φ is of the form $Q_1 x_1 \cdots Q_\ell x_\ell \, \varphi'(x_1, \dots, x_\ell)$ with quantifier-free φ'. Similarly, $\exists^*$FO is the subset of FO with only existential quantifiers in the quantifier prefix; other subsets of FO are defined analogously.

Let φ be any FO formula over some schema σ. We write $\varphi(\bar{x})$ to denote that the free variables of φ, so, those variables that are not bound by any quantifier, are all contained in $\bar{x}$. We write $(\mathcal{D}, \bar{a}) \models \varphi$ or simply $\mathcal{D} \models \varphi(\bar{a})$ if φ holds in $\mathcal{D}$ when the free variables in $\bar{x}$ are interpreted by $\bar{a}$, where $\bar{a}$ is a tuple of elements from the domain of the σ-structure $\mathcal{D}$ and of the same length as $\bar{x}$.

Every FO formula $\varphi(\bar{x})$ naturally defines a query whose result for $\mathcal{D}$ is $\{\bar{a} \mid \mathcal{D} \models \varphi(\bar{a})\}$. A query q is *definable* (or *expressible*) in FO if there is an FO formula φ that has the same result as q for every structure over the corresponding schema.

Sometimes we allow first-order formulas to access built-in *numerical relations.* An $\mathsf{FO}(\leq, +, \times)$ formula may use a binary relation symbol $\leq$ and ternary relation symbols $+$ and $\times$, which in any structure are interpreted as a linear order on the domain and compatible addition and multiplication relations. If the structure does not already provide a linear order, we require that the result of the formula is invariant under the concrete choice of $\leq$. If a linear order on a domain D is available, we often identify D with the set $\{0, \dots, |D| - 1\}$, and do not distinguish between domain elements and the numbers they represent.

We also allow an $\mathsf{FO}(\leq, +, \times)$ formula to use the binary BIT relation, which contains a tuple (i, j) if the j-th bit in the binary representation of the number i is 1. In the presence of $\leq$, a first-order formula can express BIT given $+$ and $\times$, and vice versa [Immerman 1999].

Monadic second-order logic MSO extends FO by existential and universal quantification over sets, using quantifiers of the form $\exists X$ and $\forall X$. The notions defined above naturally translate to MSO.

While Reach cannot be expressed in FO, the following example provides an MSO formula for Reach.

Example 2.1. The following MSO formula $\varphi_{\text{Reach}}(s,t)$ expresses that there is a path from s to t, and thus expresses Reach. A path from s to t exists in a graph $G = (V, E)$ if all subsets X of V that (1) include s and (2) include all nodes that have an incoming edge from a node in X, also include t.

$$\varphi_{\text{Reach}}(s,t) \overset{\text{def}}{=} \forall X \left[\Big(X(s) \wedge \forall x \forall y \left[(X(x) \wedge E(x,y)) \rightarrow X(y) \right] \Big) \rightarrow X(t) \right] \qquad \triangleleft$$

In later chapters we consider *types* of tuples and structures, and how types compose. The *rank-k* (FO, MSO) type of a tuple $\bar{a} = (a_1, \dots, a_\ell)$ of length ℓ with respect to a structure $\mathcal{D}$ is the set of all (FO, MSO) formulas $\varphi(\bar{x})$ of quantifier rank at most k with ℓ free variables such that $\mathcal{D} \models \varphi(\bar{a})$. The rank-0 type of a tuple is also called its *atomic* type. The rank-k type of a structure $\mathcal{D}$ is the rank-k type of the empty tuple with respect to $\mathcal{D}$.

We denote the set of rank-k FO types of tuples of length ℓ by $\mathsf{FO}[k, \ell]$. As there are only finitely many semantically non-equivalent first-order formulas of quantifier-rank k with ℓ free variables, the set $\mathsf{FO}[k, \ell]$ is finite for all k and ℓ, although of non-elementary size.

A fundamental result in finite model theory [Feferman 1957; Feferman and Vaught 1959], often called Feferman-Vaught Theorem, states that the rank-k first-order type of the disjoint union of two structures is determined by the rank-k first-order types of these two structures. There are multiple extensions of this result, in particular for MSO, and we use it here in the following form.

Theorem 2.2 (Theorem of Feferman and Vaught [Feferman 1957; Feferman and Vaught 1959], see also [Makowsky 2004, Theorem 1.5])**.** *Let $\mathcal{D}$ and $\mathcal{D}'$ be two structures. Then the rank-k FO (MSO) types of a tuple $\bar{a}$ of $\mathcal{D}$ and a tuple $\bar{a}'$ of $\mathcal{D}'$ uniquely determine the rank-k FO (MSO) type of the tuple $(\bar{a}, \bar{a}')$ in the disjoint union of $\mathcal{D}$ and $\mathcal{D}'$.*

First-order formulas can be used to define a mapping from σ-structures to σ'-structures, where σ and σ' are relational schemata. A (one-dimensional) *first-order interpretation* Υ from σ to σ' consists of a first-order formula $\varphi_U(x)$ and first-order formulas $\varphi_R(x_1, \dots, x_r)$ for each r-ary relation symbol $R \in \sigma'$, each over schema σ.

The first-order interpretation Υ *interprets*, in a σ-structure $\mathcal{D}$, the σ'-structure $\Upsilon(\mathcal{D})$ with universe $U^{\Upsilon(\mathcal{D})} \overset{\text{def}}{=} \{a \mid \mathcal{D} \models \varphi_U(a)\}$ and relations

$$R^{\Upsilon(\mathcal{D})} \overset{\text{def}}{=} \{(a_1, \dots, a_r) \mid \mathcal{D} \models \varphi_R(a_1, \dots, a_r), a_1, \dots, a_r \in U^{\Upsilon(\mathcal{D})}\}$$

for each $R \in \sigma'$.

A first-order interpretation from σ to σ' not only interprets a σ'-structure in a σ-structure, it also translates σ'-formulas to σ-formulas.

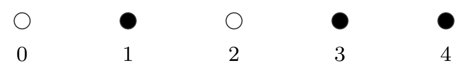
Lemma 2.3 (see [Blumensath et al. 2008, Proposition 3.2])**.** *Let Υ be a first-order interpretation from σ to σ'. For every FO (MSO) formula $\varphi(x_1, \dots, x_\ell)$ over σ' there is an FO (MSO) formula $\varphi^\Upsilon(x_1, \dots, x_\ell)$ over σ such that $\mathcal{D} \models \varphi^\Upsilon(a_1, \dots, a_\ell) \Leftrightarrow \Upsilon(\mathcal{D}) \models \varphi(a_1, \dots, a_\ell)$ for all σ-structures $\mathcal{D}$ and all elements $a_i \in U^{\Upsilon(\mathcal{D})}$.*

Analogously to one-dimensional FO interpretations one can define *d-dimensional* first-order interpretations, for a natural number $d \geq 1$, which use d-tuples of elements of $\mathcal{D}$ to represent the elements of the universe $U^{\Upsilon(\mathcal{D})}$. Lemma 2.3 holds accordingly.

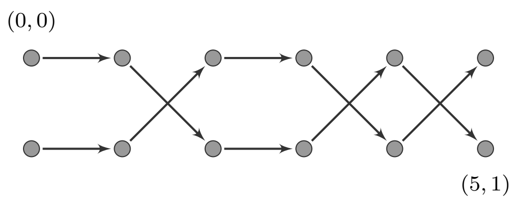
Let q and q' be Boolean queries on structures over the schemas σ and σ', respectively. A first-order interpretation Υ from σ to σ' is a *first-order reduction* from q to q' if $\mathcal{D} \in q$ holds if and only if $\Upsilon(\mathcal{D}) \in q'$, for each σ-structure $\mathcal{D}$.

Example 2.4. The query Parity can be reduced to s-t-Reach, the Boolean query that asks whether $(s, t) \in \text{Reach}(G)$, given a directed graph G and two nodes s and t of G, via a first-order reduction that has access to a linear order $\leq$ on the domain.

Given a structure $\mathcal{D}$ with a linearly ordered domain D of size n, which we identify with the set $\{0, \dots, n-1\}$, and a set $U \subseteq D$, this reduction produces a graph $G(\mathcal{D})$ with node set $\{0, \dots, n\} \times \{0, 1\}$. There are edges from a node $(i, 0)$ to a node $(i+1, 0)$ and from $(i, 1)$ to $(i+1, 1)$ if $i \notin U$, and else there are edges from $(i, 0)$ to $(i+1, 1)$ and from $(i, 1)$ to $(i+1, 0)$, for all $i \in D$. See Figure 2.1 for an example. There is path from $(0, 0)$ to $(n, 1)$ if and only if U has odd size, and the graph $G(\mathcal{D})$ can be interpreted in $\mathcal{D}$ by a



(a) A structure $\mathcal{D}$ with $n = 5$. An element $i \in U$ is represented by a black circle, an element $i' \notin U$ is represented by a white circle.



(b) The corresponding graph $G(\mathcal{D})$.

Figure 2.1: An illustration of the construction in Example 2.4.

two-dimensional $\mathsf{FO}(\leq)$ interpretation[1]. It follows that there is an $\mathsf{FO}(\leq)$ reduction from PARITY to s-t-REACH. $\lhd$

Circuit complexity The complexity class AC^0 contains all languages that are accepted by families of circuits that consist of "and"- and "or"-gates with unbounded fan-in as well as "not"-gates, and have polynomial size and constant depth. We often abuse notation and denote by AC^0 (and analogously for other complexity classes) also the class of functions that can be computed by circuits of this form.

In this thesis we only consider uniform families of circuits, and more specifically, first-order uniform families of circuits. A family $(C_n)_{n \in \mathbb{N}}$ of circuits is *first-order uniform* if the mapping $n \mapsto C_n$ can be defined by an $\mathsf{FO}(\leq, +, \times)$ interpretation, see [Immerman 1999, Definition 5.16] for details. It is well-known that a language is in (first-order uniform) AC^0 if and only if it can be expressed in $\mathsf{FO}(\leq, +, \times)$ [Barrington et al. 1990].

Analogously to AC^0 one can define the class AC^1 via circuits of depth $\mathcal{O}(\log n)$, and more generally the class $\mathsf{AC}[f(n)]$ that allows for circuits of depth $\mathcal{O}(f(n))$, for any function $f \colon \mathbb{N} \to \mathbb{N}$. The class AC^1 is a superset of, for example, $\mathsf{LOGSPACE}$ and NL.

The class $\mathrm{IND}[f(n)]$ contains queries that can be defined by *inductive applications* of first-order formulas. A query q is in $\mathrm{IND}[f(n)]$ if it can be defined by a first-order formula ψ_q that may use a relation R obtained by $\mathcal{O}(f(n))$-fold iteration of an $\mathsf{FO}(\leq, +, \times)$ formula φ_R, where n is the size of the domain. We refer to [Immerman 1999, Definition 4.16] for a formal definition, and only provide an example.

Example 2.5. We show that REACH is in $\mathrm{IND}[\log n]$. Consider the formula $\varphi_{\mathrm{R}}(x, y) = (x = y) \vee E(x, y) \vee \exists z (R(x, z) \wedge R(z, y))$. When applying the formula to a graph and an empty relation R it defines the relation R_1 of paths of length 1, applying it to $R \stackrel{\text{def}}{=} R_1$ defines the paths of length 2; in general applying the formula to $R \stackrel{\text{def}}{=} R_i$ defines the paths of length 2^i. Thus $\log n$-fold application of φ_{R} defines the transitive closure relation of a graph with n nodes. The formula ψ_{REACH} is just $R(x, y)$. $\lhd$

For polynomially bounded and first-order constructible functions $f \colon \mathbb{N} \to \mathbb{N}$, the classes $\mathsf{AC}[f(n)]$ and $\mathrm{IND}[f(n)]$ are equal [Immerman 1999, Theorem 5.22], and in particular $\mathsf{AC}^1 = \mathrm{IND}[\log n]$. Here, a function $f \colon \mathbb{N} \to \mathbb{N}$ is *first-order constructible*, if there is an

[1]The latter statement is true under the condition that D contains at least three elements. Structures with less than three elements can be dealt with separately.

$\mathsf{FO}(\leq, +, \times)$ formula $\psi_f(\bar{x})$ such that $\mathcal{D} \models \psi_f(\bar{a})$ if and only if $\bar{a}$ is a base-n representation of $f(n)$, for any ordered structure $\mathcal{D}$ with domain $\{0, \ldots, n-1\}$.

2.2 The dynamic complexity framework

In the section we present the dynamic complexity framework formally. Before we define what it means for a query to be maintained under changes to the input, we have to discuss the kind of changes we want to consider.

In this work, we study three classes of changes.

a) As already mentioned, *single-tuple changes* are most studied in the dynamic complexity literature. A single-tuple change modifies one input relation by inserting or deleting one tuple.
b) *Bulk changes* generalise insertions and deletions of single tuples to insertions and deletions of sets of tuples.
c) A *definable change* of a k-ary relation R is, in its simplest form, given by a logical formula with k free variables, possibly using additional parameters. The formula defines a k-ary query over the input structure; the change replaces R by the result of the query.

Formal definitions for the different classes of changes will be given below and in later chapters, respectively. Before, we develop an abstract, generalised notion that is also used for the definition of the framework.

A *change operation* $\delta(\bar{P}, \bar{p})$ has two tuples of variables: the *relation parameter tuple* $\bar{P}$, where each variable P_i has an associated arity $\text{Ar}(P_i)$, and the *element parameter tuple* $\bar{p}$. A *change* $\alpha = (\delta, \bar{A}, \bar{a})$ over a domain D consists of a change operation δ, a tuple $\bar{A}$ of relations over D, and a tuple $\bar{a}$ of elements of D; the tuples $\bar{A}$ and $\bar{a}$ are of the same length as $\bar{P}$ and $\bar{p}$, respectively, and each relation A_i has arity $\text{Ar}(P_i)$. We often use the notation $\alpha = \delta(\bar{A}, \bar{a})$.

The structure that results from applying α to a structure $\mathcal{D}$ is denoted by $\alpha(\mathcal{D})$. For sequences $\beta = \alpha_1 \cdots \alpha_m$ of changes, $\beta(\mathcal{D})$ is defined as $\alpha_m(\cdots(\alpha_1(\mathcal{D}))\cdots)$. The actual definition of $\alpha(\mathcal{D})$, that is, the effect of applying α to a structure, will be formally defined separately for different change operations δ.

A concrete instance of the change operations we just defined are *single-tuples changes*. Let σ be a schema. For a k-ary relation symbol $R \in \sigma$, we define single-tuple *insertions* **insert** $\bar{p}$ **into** R and *deletions* **delete** $\bar{p}$ **from** R, where $\bar{p}$ has length k. The change operations are usually more concisely noted as $\textsc{ins}_R(\bar{p})$ and $\textsc{del}_R(\bar{p})$, respectively. Given a σ-structure $\mathcal{D}$ and a change $\alpha = \textsc{ins}_R(\bar{a})$ over its domain, the structure $\alpha(\mathcal{D})$ consists of the relations

$$S^{\alpha(\mathcal{D})} \overset{\text{def}}{=} \begin{cases} S^{\mathcal{D}} & S \neq R \\ S^{\mathcal{D}} \cup \{\bar{a}\} & S = R \end{cases}$$

for each $S \in \sigma$. Symmetrically, for a change $\alpha' = \textsc{del}_R(\bar{a})$, the structure $\alpha'(\mathcal{D})$ has the relations

$$S^{\alpha'(\mathcal{D})} \stackrel{\text{def}}{=} \begin{cases} S^{\mathcal{D}} & S \neq R \\ S^{\mathcal{D}} \setminus \{\bar{a}\} & S = R \end{cases}.$$

For a schema σ we usually write Δ_σ for the set of all single-tuple change operations for σ-structures. For graphs (and similar for other structures over a singleton schema), we write Δ_E instead of $\Delta_{\{E\}}$.

We now introduce dynamic programs and dynamic complexity classes, based on the framework proposed by Patnaik and Immerman [1997], closely following the expositions of Schwentick and Zeume [2016] and Schwentick et al. [2018].

The goal of a dynamic program is to maintain the answer to a fixed query q when the underlying structure changes, where the changes are based on a fixed set Δ of change operations. We call the pair (q, Δ) a *dynamic query* and denote the schema of q by σ_{in}. To maintain a dynamic query, a dynamic program stores a set of *auxiliary relations* over a schema σ_{aux}. We demand that σ_{aux} always contains a relation symbol $\textsc{Ans}$ with the same arity as q. The pair $(\sigma_{\text{in}}, \sigma_{\text{aux}})$ is called *dynamic schema*.

The update of an auxiliary relation S after a change based on an operation $\delta(\bar{P}, \bar{p})$ is specified by an *update rule* **on change** $\delta(\bar{P}, \bar{p})$ **update** S **as** $\varphi_\delta^S(\bar{P}, \bar{p}; \bar{x})$, where the length of $\bar{x}$ equals the arity of S. The formula φ_δ^S is called the *update formula*.

Definition 2.6 (Dynamic $\mathcal{L}$-program)**.** Let $\mathcal{L}$ be a logic. A *dynamic $\mathcal{L}$-program $\mathcal{P}$* for a dynamic query (q, Δ) over dynamic schema $(\sigma_{\text{in}}, \sigma_{\text{aux}})$ contains, for every $\delta(\bar{P}, \bar{p}) \in \Delta$ and every $S \in \sigma_{\text{aux}}$, an update rule **on change** $\delta(\bar{P}, \bar{p})$ **update** S **as** $\varphi_\delta^S(\bar{P}, \bar{p}; \bar{x})$, where φ_δ^S is a formula from $\mathcal{L}$ over schema $\sigma_{\text{in}} \cup \sigma_{\text{aux}}$.

Usually, the logic $\mathcal{L}$ is first-order logic FO or some other logic clear from the context. We then do not mention $\mathcal{L}$ explicitly and just speak of *dynamic programs*.

We now define the semantics of a dynamic program. Let $\mathcal{P}$ be a dynamic program for a dynamic query (q, Δ) over dynamic schema $(\sigma_{\text{in}}, \sigma_{\text{aux}})$. A *state* $\mathcal{S} = (D, \mathcal{I}, Aux)$ of $\mathcal{P}$ consists of an *input structure* $\mathcal{I}$ and an *auxiliary structure* Aux over a common domain D. The schemata of $\mathcal{I}$ and Aux are σ_{in} and σ_{aux}, respectively. We consider $\mathcal{S}$ as one relational structure. For a change operation $\delta \in \Delta$ and a change $\alpha = \delta(\bar{A}, \bar{a})$ over domain D, the application of α to $\mathcal{S}$ yields the state $\mathcal{P}_\alpha(\mathcal{S}) = (D, \alpha(\mathcal{I}), Aux')$, where for each $S \in \sigma_{\text{aux}}$ the relation $S^{Aux'}$ in Aux' is equal to $\{\bar{b} \mid \mathcal{S} \models \varphi_\delta^S(\bar{A}, \bar{a}; \bar{b})\}$. For sequences $\beta = \alpha_1 \cdots \alpha_m$ of changes, $\mathcal{P}_\beta(\mathcal{S})$ is the state $\mathcal{P}_{\alpha_m}(\cdots(\mathcal{P}_{\alpha_1}(\mathcal{S}))\cdots)$.

The dynamic program $\mathcal{P}$ *maintains* (q, Δ), if for every sequence of changes over Δ the auxiliary relation $\textsc{Ans}$ of $\mathcal{P}$ coincides with the result of q on the current input structure. More formally, we demand that, for every non-empty domain D and every non-empty sequence β of changes over Δ and D, the relation $\textsc{Ans}$ in $\mathcal{P}_\beta(\mathcal{S}_0)$ equals $q(\beta(\mathcal{I}_0))$. Here, $\mathcal{S}_0 \stackrel{\text{def}}{=} (D, \mathcal{I}_0, Aux_0)$ and $\mathcal{I}_0$ and Aux_0 are the empty input and auxiliary structures over D, respectively. Because the relation $\textsc{Ans}$ contains the query result, we also call it the *query relation* of $\mathcal{P}$.

Dynamic complexity classes consist of dynamic queries that can be maintained by (restricted) dynamic programs. The most important class for this thesis is the class DynFO which allows for dynamic programs with full first-order logic update formulas.

Definition 2.7 (DynFO)**.** The class DynFO is the class of dynamic queries that can be maintained by dynamic FO-programs.

Instead of $(q, \Delta) \in$ DynFO we also often say that (q, Δ) *can be maintained in* DynFO, or that q is in DynFO under Δ changes, and similar for other dynamic complexity classes.

Example 2.8. As already noted in the introduction, the reachability query $\textsc{Reach}$ can be maintained dynamically under edge insertions. We can now formalise this insight as statement $(\textsc{Reach}, \{\textsc{ins}_E\}) \in$ DynFO. The dynamic program that maintains the dynamic query $(\textsc{Reach}, \{\textsc{ins}_E\})$ stores and updates a single auxiliary relation $\textsc{Ans}$ that always contains the transitive closure of the edge relation E. Its update formula is given by $\varphi^{\textsc{Ans}}_{\textsc{ins}_E}(p_1, p_2; x, y) \stackrel{\text{def}}{=} \textsc{Ans}(x, y) \vee (\textsc{Ans}(x, p_1) \wedge \textsc{Ans}(p_2, y))$. $\triangleleft$

For some dynamic queries (q, Δ) we are only interested in maintaining (q, Δ) as long as the input structure is from a certain class $\mathcal{C}$ of structures. Examples for these restrictions are acyclic graphs or structures with bounded treewidth (which will be studied in Section 4.1). We say that a dynamic program $\mathcal{P}$ maintains (q, Δ) *for* $\mathcal{C}$ if $\mathcal{P}$ maintains (q, Δ) for every change sequence $\beta = \alpha_1 \cdots \alpha_m$ over Δ such that every structure $\mathcal{I}_i \stackrel{\text{def}}{=} \alpha_1 \cdots \alpha_i(\mathcal{I}_\emptyset)$ is a structure from $\mathcal{C}$. So, we demand that every prefix of β transforms an empty structure into a structure from $\mathcal{C}$. This definition just disallows any change sequence that results in a structure outside of $\mathcal{C}$. In some cases, a dynamic $\mathcal{L}$-program can also set up a *guard*: an $\mathcal{L}$ formula that detects, using the auxiliary relations, when a change would lead to a structure outside of $\mathcal{C}$.

We say that (q, Δ) is in DynFO *for* $\mathcal{C}$ if there is a dynamic FO-program that maintains (q, Δ) for $\mathcal{C}$.

Example 2.9 ([Patnaik and Immerman 1997])**.** We show that the reachability query can be maintained under edge insertions and deletions for acyclic graphs, so, that $(\textsc{Reach}, \Delta_E)$ is in DynFO for acyclic graphs. As in Example 2.8, the dynamic program that maintains $(\textsc{Reach}, \Delta_E)$ for acyclic graphs only needs a single auxiliary relation $\textsc{Ans}$ that contains the query result. The update formula for insertions $\textsc{ins}_E$ is already given in Example 2.8.

The update formula $\varphi^{\textsc{Ans}}_{\textsc{del}_E}(p_1, p_2; x, y)$ for deletions $\textsc{del}_E(\bar{p})$ is constructed using the following observations. Let G be an acyclic graph and let G' be the graph that is obtained from G by deleting an edge (a, b). There is a path from a node u to a node v in G' if and only if (1) such a path exists in G and (2a) no path from u to v in G uses the edge (a, b) or (2b) there is a path ρ in G that uses an edge $(c, c') \neq (a, b)$ with the property that a is reachable from c but not from c'.

We do not formally prove that claim here, but give an informal correctness argument. Consider a graph G such that there is a path from u to v via (a, b) in G, no path ρ as in Condition (2b) exists, and v is reachable from u in G' via a path ρ'. As ρ' does not satisfy Condition (2b), there is in particular a path from v to a in G. As there also is a path from a to v in G, this graph is not acyclic.

We obtain the update formula as follows:

$$\begin{aligned}\varphi^{\textsc{Ans}}_{\textsc{del}_E}(p_1, p_2; x, y) \overset{\text{def}}{=} \ &\textsc{Ans}(x, y) \wedge \Big((\neg\textsc{Ans}(x, p_1) \vee \neg\textsc{Ans}(p_2, y)) \\ &\vee \exists z \exists z' (\textsc{Ans}(x, z) \wedge E(z, z') \wedge \textsc{Ans}(z', y) \wedge (z \neq p_1 \vee z' \neq p_2) \\ &\qquad \wedge \textsc{Ans}(z, p_1) \wedge \neg\textsc{Ans}(z', p_1))\Big)\end{aligned}$$

Observe that there is a first-order guard that checks whether an insertion $\textsc{ins}_E(a, b)$ leads to a cyclic graph: this is the case if and only if there already is a path from b to a in the graph at hand, so, if $(b, a) \in \textsc{Ans}$ holds. ◁

We already see from these examples that dynamic FO-programs are surprisingly expressive: reachability in acyclic graphs is an NL-complete problem, and it is one of the prime examples of a query that cannot be expressed in first-order logic statically.

To understand better the expressive power of dynamic programs, we study restricted dynamic programs that only allow for syntactically restricted update formulas. As we see in Example 2.8, even update formulas without quantifiers can be used to maintain non-trivial queries. This insight leads to the definition of the class DynProp.

Definition 2.10 (DynProp)**.** The class DynProp is the class of dynamic queries that can be maintained by dynamic FO-programs whose update formulas are quantifier-free.

Example 2.11. The query Parity is not in AC^0 and therefore not expressible even in full FO [Ajtai 1983; Furst et al. 1984]. Dynamically, under single-tuple changes, the query $(\textsc{Parity}, \Delta_U)$ can be maintained easily. Each change transforms a relation with odd parity into a relation with even parity, and vice versa, provided that the change actually alters U. The update formulas for the only auxiliary relation Ans are as follows.

$$\begin{aligned}\varphi^{\textsc{Ans}}_{\textsc{ins}_U}(p) &\overset{\text{def}}{=} (\textsc{Ans} \wedge U(p)) \vee (\neg\textsc{Ans} \wedge \neg U(p)) \\ \varphi^{\textsc{Ans}}_{\textsc{del}_U}(p) &\overset{\text{def}}{=} (\textsc{Ans} \wedge \neg U(p)) \vee (\neg\textsc{Ans} \wedge U(p))\end{aligned}$$ ◁

In the beginning of Chapter 4 we review in more detail previous result on which dynamic queries can be maintained in DynFO and in DynProp, respectively.

Further restrictions of DynFO are studied in the literature [Hesse 2003b; Zeume and Schwentick 2017]. In general, we denote by Dyn$\mathcal{L}$ the class of dynamic queries that can be maintained by dynamic $\mathcal{L}$-programs, for any logic $\mathcal{L}$.

The class DynFO, as well as all other dynamic complexity classes we consider, is not known to be closed under first-order reductions, which are often considered in static complexity theory. For the dynamic complexity framework, Patnaik and Immerman [1997] introduced *bounded* first-order reductions. Similar notions of reductions are defined by other authors [Hesse and Immerman 2002; Grädel and Siebertz 2012; Datta et al. 2018a]. They all have in common that whenever a reduction maps an instance $\mathcal{I}$ of some dynamic query to an instance $\mathcal{I}'$ of some other dynamic query, then every change of $\mathcal{I}$ induces at most c changes in the instance $\mathcal{I}'$, for some global constant c.

More formally, for Boolean dynamic queries (q, Δ) and (q', Δ') we say that a first-order reduction Υ is a *bounded first-order reduction*, if there is a $c \in \mathbb{N}$ such that for all input structures $\mathcal{I}$ for q and all changes α of $\mathcal{I}$ over Δ there is a first-order definable sequence β of changes that consists of at most c changes over Δ' such that $\Upsilon(\alpha(\mathcal{I})) = \beta(\Upsilon(\mathcal{I}))$. We refer to [Datta et al. 2018a] for a more general notion of reductions that also can be used for non-Boolean dynamic queries.

Note that the reduction from Example 2.4 is a bounded first-order reduction, as each change in $\mathcal{D}$ induces the insertion and deletion of two edges in $G(\mathcal{D})$, respectively.

2.3 Variants of the setting

The definitions of the previous section are not without alternatives. In this section we discuss some of the choices made for the definitions and point to articles that further research the alternative settings. This in particular concerns the choice of the domain and the initial state of a dynamic program. Already Zeume [2015] and Schwentick and Zeume [2016] compare the different variants of dynamic complexity thoroughly, therefore we here keep the discussion short and refer to the mentioned work for more details.

Domain One of the possibly unintuitive properties of the dynamic complexity framework, as defined in the previous section, is that the domain of the database is fixed throughout the dynamic process: changes cannot introduce new elements, and no element can be removed from the domain. Note that although the domain is fixed, dynamic programs need to work uniformly for all domains of any size.

The framework of *first order incremental evaluation systems* (FOIES) [Dong et al. 1995; Dong and Su 1995] has its roots in database theory and allows the domain to grow and shrink depending on the changes. Apart from this feature, it is basically equivalent to the DynFO setting. In the FOIES framework, databases have an infinite background domain, and the update formulas are evaluated on its active domain. Changes can add elements to the active domain by tuple insertions, or remove them if all tuples that contain them are deleted. Sometimes, for example in [Dong and Su 1997], elements may not be used any more for auxiliary relations if they are removed from the active domain.

The setting of DynFO as we use it follows the formalisation by Patnaik and Immerman [1997], which originates from (descriptive) complexity theory and finite model theory, and therefore only considers fixed and finite domains[2]. There are several advantages in using this arguably simpler framework. At first, many interesting phenomena of dynamic computations already arise for fixed domains, dealing with a varying domain might only impose technical difficulties that we are not primarily interested in. Secondly, the connections between logics and circuit complexity allow us to re-use results obtained from the latter research field. A disadvantage might be that the FOIES setting is more inclined to practice. However, result on whether a dynamic query can be maintained usually carry over from DynFO to FOIES. This is immediate for lower bounds: if a dynamic query cannot be maintained in DynFO, it also cannot be maintained under additional changes of the domain. Even for upper bounds this is true, at least for single-tuple changes:

[2]In [Patnaik and Immerman 1997], an input structure has a unary relation that holds its active domain, which seems similar to the FOIES framework. It does not seem to be actually used.

for a large class of natural σ-queries q, $(q, \Delta_\sigma) \in \mathsf{DynFO}$ implies that (q, Δ_σ) can also be maintained in the FOIES setting (see [Datta et al. 2018a, Theorem 17] for a precise statement).

We exclusively prove results for the DynFO framework in this thesis. However, most of our results transfer to a FOIES-like extension of the DynFO framework that allows for change operations $\text{add}(x)$ and $\text{remove}(x)$ that add an element x to the domain or remove it, respectively. Whenever results do not transfer immediately, we discuss issues and possible adaptations towards the end of the forthcoming chapters.

Initial input and auxiliary structures In our definition, a dynamic programs maintains a query starting from a state $\mathcal{S}_0 = (D, \mathcal{I}_0, Aux_0)$ with empty initial and auxiliary relations. This is also the standard setting in [Gelade et al. 2012; Datta et al. 2018a] and is called DynFO *from scratch* in [Zeume 2015]. Several different choices are made in the literature. Our setting is equivalent to the Dyn-FO setting of Patnaik and Immerman [1997], which is re-used by Etessami [1998] and Hesse [2003b], that considers empty initial input structures and first-order definable initial auxiliary relations (the equivalence is clear, see also [Gelade et al. 2012, Lemma 2.2]). The extension of this setting that allows polynomial time computable initialisation of auxiliary relations is called $\mathsf{Dyn\text{-}FO}^+$ in [Patnaik and Immerman 1997].

Alternative settings allow arbitrary initial input structures. In this case, an initialisation of the auxiliary relations becomes necessary: if only initially empty auxiliary relations were allows, all queries maintainable by dynamic FO-programs with arbitrary initial input structure are already expressible in FO. In its strongest form, the auxiliary relations can be initialised arbitrarily [Zeume 2015; 2017; Zeume and Schwentick 2015; 2017]. In [Datta et al. 2018a], this variant is called *non-uniform* DynFO. A dynamic query (q, Δ) is in non-uniform DynFO, if for each input structure $\mathcal{I}$ with domain D there is an auxiliary structure Aux such that (q, Δ) can be maintained in DynFO starting from the initial state $(D, \mathcal{I}, Aux)$.

A more fine-grained framework is introduced in [Weber and Schwentick 2007]. For any complexity class $\mathcal{C}$, the class $\mathsf{Dyn}(\mathcal{C}, \mathsf{FO})$ contains the dynamic queries that can be maintained in DynFO after a $\mathcal{C}$-computable initialisation. The setting of [Grädel and Siebertz 2012] looks similar at first sight. The authors consider classes $\mathsf{Dyn}(\mathcal{L}, \mathsf{FO})$, for any logic $\mathcal{L}$, that contain dynamic queries maintainable in DynFO after an $\mathcal{L}$-expressible initialisation. As a striking difference to the variants mentioned before, these initialisations are *permutation-invariant* in the sense of [Zeume and Schwentick 2015]. Let $\textsc{Init}$ be the mapping from input structures to auxiliary structures defined by the $\mathcal{L}$-expressible initialisation. For every input structure $\mathcal{I}$ and every permutation π of the domain of $\mathcal{I}$, $\textsc{Init}$ satisfies $\pi(\textsc{Init}(\mathcal{I})) = \textsc{Init}(\pi(\mathcal{I}))$. A permutation-invariant initialisation cannot define a linear order on the domain, amongst others. This is the main reason why there are general lower bound results for the setting of [Grädel and Siebertz 2012]: for example the query $\textsc{TreeIsomorphism}$ is not in $\mathsf{Dyn}(\mathcal{L}, \mathsf{FO})$ for any logic $\mathcal{L}$, but it is known to be in DynFO under single-edge changes [Etessami 1998].

In the FOIES framework, similar choices are made regarding initialisation, although less emphasis is put on this aspect. For example, [Dong and Su 1998] does not formally specify the initialisation, but only mentions that arbitrary initialisation as well as empty initialisation with empty initial input databases are possible choices. Other articles, for example [Dong and Su 1995], opt for arbitrary initialisation. In [Dong et al. 1995], the initial auxiliary relations are computed by Datalog programs.

How can we motivate the different settings? Much like fixing the domain, allowing arbitrary initialisation lets us focus on the maintenance aspect of dynamic queries (cf. [Zeume and Schwentick 2017]): it is a useful first step to prove that a dynamic query can be maintained “at all”, without worrying about initialisation. Results that certain dynamic queries cannot be maintained are strongest if they still hold with arbitrary initialisation. On the other hand, maintainability results are stronger, the less initialisation they require. Especially for practical applications it is important that the initial auxiliary relations can actually be computed, preferably with low complexity.

In this thesis we follow a pragmatic approach. The class DynFO does not allow initialisation, so results that a dynamic query is in DynFO hold for all standard variants, with the exception of the setting of [Grädel and Siebertz 2012]. If a result requires some form of initialisation, we explicitly mention this and say that a dynamic query is in DynFO *with suitable initialisation*, and specify the precise initialisation we use. For results that certain dynamic queries cannot be maintained in DynFO, we also specify whether this holds even when certain classes of initialisations are allowed.

Linear order and arithmetic relations An important special case of initialisations is whether certain *built-in numeral relations* are assumed to be present. A numerical relation only depends on the domain and does not change in the dynamic process. Examples for such relations are a linear order $\leq$ and compatible relations $+$ and $\times$ expressing addition and multiplication. These three relations are particularly important, as they allow to connect first-order logics with circuit complexity classes: $\mathsf{FO}(\leq, +, \times)$ is equal to AC^0 [Barrington et al. 1990].

Whether built-in relations are allowed differs among the various settings. The setting of extended dynamic programs of [Gelade et al. 2012] allows in principle arbitrary built-in relations. In [Hesse 2003b; Patnaik and Immerman 1997], all of $\leq, +, \times$ are assumed to be present, whereas the conference version [Patnaik and Immerman 1994] only assumes $\leq$. The update programs in [Pang et al. 2005] use $\leq$ and $+$.

In this thesis we are transparent and write $\mathsf{DynFO}(\leq, +, \times)$ when we allow update formulas to use relations $\leq$, $+$ and $\times$. In the realm of single-tuple changes, the difference between DynFO and $\mathsf{DynFO}(\leq, +, \times)$ is rather small: even when $\leq$, $+$ and $\times$ are not present from the beginning, the restriction of these relations to the active domain[3] can be maintained [Patnaik and Immerman 1997; Etessami 1998]. As a result, for a large class of natural queries, membership in $\mathsf{DynFO}(\leq, +, \times)$ implies membership in DynFO under single-tuple changes [Datta et al. 2018a, Proposition 7], see also Proposition 3.2. This is a major difference to the unordered setting of [Grädel and Siebertz 2012].

[3]More precisely, the restriction to the *activated* domain can be maintained. An element is called activated, if it was part of the active domain at some point during the dynamic process.

The difference between DynFO and DynFO$(\leq, +, \times)$ becomes more visible under complex changes. A linear order on the active domain can be maintained under single-tuple changes because elements enter the active domain "one at a time": each change uses only constantly many elements as parameters, and the parameter tuple orders them linearly. New elements are appended in that order to the end of the existing linear order.

Under complex changes, a non-constant number of elements can enter the active domain in one step, and in general FO cannot define a linear order on them. We will discuss in Chapter 6 the differences between DynFO$(\leq, +, \times)$ and DynFO for certain complex changes.

2.4 Outlook and bibliographic remarks

In this chapter we formalised the dynamic complexity framework and discussed several variants. The precise relationship between these variants often stays an open problem. Thanks to [Datta et al. 2018a, Proposition 7 and Theorem 17], we have a good understanding on the relationship between DynFO, DynFO$(\leq, +, \times)$ and FOIES for a large natural class of queries under single-tuple changes. For some specific dynamic queries we will discuss the differences between these frameworks towards the end of Chapter 6, but a general result for complex changes in the spirit of [Datta et al. 2018a, Proposition 7 and Theorem 17] is still missing.

The DynFO framework and the setting of Grädel and Siebertz [2012] differ considerably, which can be seen by means of the TreeIsomorphism query. Although Grädel and Siebertz [2012] relate subclasses from their framework to subclasses of FOIES, no general result is known that, for example, allows to transfer maintainability result for classes of dynamic queries from DynFO to the framework of [Grädel and Siebertz 2012].

Bibliographic remarks

The setting of dynamic complexity as introduced in this chapter follows the exposition of Schwentick and Zeume [2016], which is based on the setting of Dyn-FO by Patnaik and Immerman [1994; 1997]. The class DynProp is introduced in [Hesse 2003b]. The definition of change operations is a generalisation of the definition in [Schwentick et al. 2018]. Note that terms vary in the literature: what we call "change operation" and "change" is called "operation symbol" and "operation" in [Weber and Schwentick 2007], and "abstract modification" and "concrete modification" in [Zeume and Schwentick 2015], respectively. In [Schwentick et al. 2018], "change operation" is used as a synonym for "change", and the term "replacement query" is used instead.

The presented examples are standard in the dynamic complexity literature. Example 2.8 appeared first in (a conference version of) [Dong et al. 1995], Example 2.9 is from [Patnaik and Immerman 1997], which in turn is a simplification of a result of [Dong and Su 1995]. Example 2.11 appears amongst others in [Patnaik and Immerman 1997; Etessami 1998; Weber and Schwentick 2007]. The examples from Section 2.1 are folklore.

CHAPTER 3

The Muddling technique: Bounding the number of changes

Whenever a long-standing open problem is solved, for example when for some problem a new algorithm is presented that beats all previously known algorithms in terms of worst-case complexity, or when a complexity theoretic lower bound is established, researchers in computer science not only care about the result, but are also interested in the algorithmic techniques or proof techniques used. Can these techniques be applied to other problems, similar ones or in different domains? Can they be strengthened?

This also applies for proofs showing that some dynamic query can be maintained in DynFO. The constructed dynamic programs often maintain particular data structures, which are encoded within the auxiliary database, or show that logical primitives can be simulated. For example, Gelade et al. [2012] show how to maintain a certain query in DynProp using a list structure, a technique which is re-used by Zeume and Schwentick [2015] (and will be used later in Section 4.2). Datta et al. [2018a] show that for maintaining queries with some particular property, one can assume that a linear order on the domain as well as arithmetic relations $+, \times$ are available.

One goal in dynamic complexity research, and of this chapter in particular, is to expand the "tool box" of techniques to maintain a query dynamically.

Goal 3.1. Identify high-level strategies of dynamic query maintenance.

The purpose of this chapter is to introduce a technique that bounds the number of consecutive changes a dynamic program needs to support.

In Section 2.3 we discussed several details of the definitions of the dynamic complexity framework. However, we did not question our understanding that a dynamic program only maintains a dynamic query if it can update the query result under change sequences of arbitrary length. Of course, this requirement might be hard or even impossible to meet. In practical scenarios of dynamic query maintenance it is conceivable that a query can be maintained for some time but not indefinitely, as the quality of the updated answer decreases after each update. For example, the update might yield only approximate solutions after a change, or numerical errors in arithmetic computations might sum up. In our dynamic complexity setting, intuition tells us that it should be easier to construct a dynamic program that maintains a query for few change steps than to find a dynamic program that maintains it "forever".

In contrast to this intuition, we will prove in this chapter that we can actually transform dynamic programs for maintenance under short change sequences into dynamic programs for maintenance under change sequences of unbounded length. For this result we introduce a restricted notion of query maintenance, that in particular only asks for maintaining a query under change sequences of bounded length. When the specified number of changes have occurred (and intuitively the quality of the auxiliary relations is too poor for them to be useful any more), the query result and the auxiliary relations have to be recomputed from scratch, using an algorithm of higher complexity than the updates. Afterwards, the query result can again be maintained for some time, and so on. We will then show that dynamic queries that can be maintained in (some instances of) this restricted sense can also be maintained for arbitrary long change sequences.

Contributions We introduce the notion of $(\mathcal{C}, f)$-maintainability, for a complexity class $\mathcal{C}$ and a function $f\colon \mathbb{N} \to \mathbb{N}$, as the ability to maintain a dynamic query for $f(n)$ many changes, starting from an arbitrary initial input structure and with the help of auxiliary relations initialised by a $\mathcal{C}$-algorithm. Here, n is the size of the input structure's domain. We prove that in the context of single-tuple changes, $(\mathcal{C}, f)$-maintainable dynamic queries that meet a natural technical restriction are in DynFO, for some specific classes $\mathcal{C}$ and functions f. Consequently, we obtain a new strategy for proving membership in DynFO, which we call the *Muddling technique*: show that a dynamic query is $(\mathcal{C}, f)$-maintainable and by the results of this chapter conclude that it is also in DynFO.

This chapter presents joint work with Samir Datta, Anish Mukherjee, Thomas Schwentick and Thomas Zeume. Detailed bibliographic references are given at the end of the chapter.

Outline We formally introduce the notion of $(\mathcal{C}, f)$-maintainability in Section 3.1, together with a natural property of queries that we need to assume for our main contribution. We also repeat a result from [Datta et al. 2018a] that allows dynamic programs to use a linear order and arithmetic relations on the whole domain of the input structure. The main result of this chapter, that allows to transform dynamic programs maintaining a dynamic query for a bounded number of single-tuple changes into dynamic programs that maintain the dynamic query for arbitrarily many changes, is stated and proven in Section 3.2. We close with a short outlook and bibliographic remarks in Section 3.3.

3.1 Notions used for the Muddling technique

We formalise the notion of query maintenance we sketched in the introduction of this chapter. We say that a dynamic query (q, Δ) is *$(\mathcal{C}, f)$-maintainable*, for some complexity class $\mathcal{C}$ and some function $f : \mathbb{N} \to \mathbb{N}$, if there is a dynamic program $\mathcal{P}$ and a $\mathcal{C}$-algorithm $\mathcal{A}$ such that for each input structure $\mathcal{I}$ over a domain of size n, each linear order $\leq$ on the domain, and each change sequence β over Δ of length $|\beta| \leq f(n)$, the relation ANS in $\mathcal{P}_\beta(\mathcal{S})$ and $q(\beta(\mathcal{I}))$ coincide, where $\mathcal{S} = (\mathcal{I}, \mathcal{A}(\mathcal{I}, \leq))$. So, a dynamic program needs to maintain (q, Δ) for $f(n)$ changes after a $\mathcal{C}$-computable initialisation. We refer to Subsection 4.1.2 for an example for this notion.

In this thesis we do not study $(\mathcal{C}, f)$-maintainability in its own right, but use it as a tool to prove that dynamic queries are actually maintainable in DynFO and its variants. For the corresponding proof we need the technical assumption that the query under consideration does not depend "considerably" on the size of the underlying domain. We motivate this restriction with the example of the Boolean EVENDOMAIN query that asks whether the underlying domain of the input structure is of even size. Standard arguments show that first-order logic cannot express this query, and this inability extends to DynFO: without additional initialisation, no first-order update formula can give the query result after the first change. On the other hand, EVENDOMAIN is trivially $(\mathsf{AC}^0[2], f)$-maintainable under all finite sets of change operations and for all functions f, as the underlying domain is immutable and the fixed query answer can easily be computed by an $\mathsf{AC}^0[2]$ initialisation[1]. So, $(\mathcal{C}, f)$-maintainability alone cannot imply membership in DynFO, for any interesting choice of $\mathcal{C}$ and f.

In the following, we restrict ourselves to queries that have some natural property. A query q is called *domain independent* in [Datta et al. 2015], if the addition of isolated elements to the domain does not change the query result, that is, if $q(\mathcal{D}) = q(\mathcal{D}[\text{adom}(\mathcal{D})])$ for all structures $\mathcal{D}$. A *weakly domain independent* query q just satisfies $q(\mathcal{D})[\text{adom}(\mathcal{D})] = q(\mathcal{D}[\text{adom}(\mathcal{D})])$ for all structures $\mathcal{D}$, so, the query result restricted to the original domain stays the same when isolated elements are added. These two notions are equivalent for Boolean queries.

Here we introduce a slight generalisation of this notion and call a query q *almost domain independent* if there is a $c \in \mathbb{N}$ such that, for every structure $\mathcal{D}$ with domain D and every set $D' \subseteq D \backslash \text{adom}(\mathcal{D})$ with $|D'| \geq c$ it holds $q(\mathcal{D})[(\text{adom}(\mathcal{D}) \cup D')] = q(\mathcal{D}[(\text{adom}(\mathcal{D}) \cup D')])$. Informally this means that there is a constant c such that if a structure already has at least c "non-active" elements, adding more "non-active" elements does not change the query result with respect to the original elements. A query is weakly domain independent if and only if it is almost domain independent with $c = 0$.

Example 3.1. We give some examples for the notions we just defined.

1. The PARITY query is clearly domain independent.
2. The binary reachability query REACH is weakly domain independent: adding any set V' of isolated nodes to a graph does not create or destroy paths in the original

[1] An $\mathsf{AC}^0[2]$-circuit is defined similarly to an AC^0-circuit, but additionally may use "modulo 2"-gates with unbounded fan-in that determine whether an odd number of its inputs is evaluated to 1.

graph. Note that for each node $v \in V$ the tuple (v, v) is part of the query result $\textsc{Reach}(G)$, so $\textsc{Reach}(G) \neq \textsc{Reach}(G[\text{adom}(G)])$ in general and therefore $\textsc{Reach}$ is not domain independent.

3. The FO-definable Boolean query $\textsc{TwoIsolatedElements}$, which is true if and only if exactly two elements are not in the active domain, is almost domain independent with $c = 3$.
4. The Boolean query $\textsc{EvenDomain}$ is not almost domain independent. $\triangleleft$

We call a dynamic query (q, Δ) (weakly, almost) domain independent, if q is (weakly, almost) domain independent.

For weakly domain independent queries under single-tuple changes, maintainability in DynFO and $\mathsf{DynFO}(\leq, +, \times)$ is equivalent [Datta et al. 2018a]. This is a remarkable result, as FO is strictly weaker than $\mathsf{FO}(\leq, +, \times)$ in the static setting. One witnessing query is of course $\textsc{EvenDomain}$, which is easily expressible in $\mathsf{FO}(\leq, +, \times)$ by the formula $\varphi \stackrel{\text{def}}{=} \exists i \exists j (i + i = j \land \forall j'\, j' \leq j)$, but no equivalent FO formula exists. As mentioned above, $\textsc{EvenDomain}$ is not almost domain independent and separates DynFO and $\mathsf{DynFO}(\leq, +, \times)$. However, there is also an almost domain independent query that statically separates FO and $\mathsf{FO}(\leq, +, \times)$ [Otto 2000].

In this thesis we use a slightly more general formulation of the result by Datta et al. [2018a].

Proposition 3.2 (Adaptation of [Datta et al. 2018a, Proposition 7])**.** *Let q be an almost domain independent query and let Δ be a set of single-tuple change operations. If $(q, \Delta) \in \mathsf{DynFO}(\leq, +, \times)$ then also $(q, \Delta) \in \mathsf{DynFO}$.*

The proof of this proposition is very similar to the proof of [Datta et al. 2018a, Proposition 7], the necessary adaptation is discussed in [Datta et al. 2019].

3.2 From short to arbitrary long change sequences

We now present and prove the main result of this section, that for certain combinations of complexity classes $\mathcal{C}$ and functions $f\colon \mathbb{N} \to \mathbb{N}$, all almost domain independent $(\mathcal{C}, f)$-maintainable dynamic queries are in DynFO. After the title of the first publication using this result [Datta et al. 2017], we call it the *Muddling Theorem*, and similar call the technique of employing this theorem to show membership in DynFO the *Muddling technique.*

Theorem 3.3 (Muddling Theorem for single-tuple changes)**.** *Let $f : \mathbb{N} \to \mathbb{N}$ be a first-order constructible function with $f \in \mathcal{O}(n)$ and Δ a set of single-tuple change operations. If an almost domain independent dynamic query (q, Δ) is $(\mathsf{AC}[f(n)], f(n))$-maintainable then also $(q, \Delta) \in \mathsf{DynFO}$.*

In this thesis we mainly use a special case of Theorem 3.3, which we state as the following corollary.

Corollary 3.4. *Let Δ be a set of single-tuple change operations. If an almost domain independent dynamic query (q, Δ) is $(\mathsf{AC}^1, \log n)$-maintainable then also $(q, \Delta) \in \mathsf{DynFO}$.*

Before we give a proof, we outline its main ideas, for ease of presentation based on the statement of Corollary 3.4. Suppose we have a dynamic program $\mathcal{P}$ that maintains some dynamic query (q, Δ) for $\log n$ changes, starting from an arbitrary input instance and auxiliary relations that are computed by an AC^1 algorithm. How can we construct a dynamic program $\mathcal{P}'$ that maintains this dynamic query for change sequences of arbitrary length, and without initialisation (but from initially empty input structures)?

In a first step we can construct from $\mathcal{P}$ a dynamic program $\mathcal{P}^*$ that does not use any initialisation, but is only able to answer the query after the last of the $\log n$ changes happened. Observe that the AC^1 initialisation can be computed by applying a first-order formula $\log n$ times, thanks to the equality $\mathsf{AC}^1 = \text{IND}[\log n]$. Also, any update formula of $\mathcal{P}$ is applied $\log n$ times while processing the change sequence of length $\log n$. The idea is now to distribute the work of the initialisation among the $\log n$ updates.

The program $\mathcal{P}^*$ proceeds as follows. During the first half of the changes, it simulates the initialisation by applying a corresponding first-order formula twice at each step. During the second half of the changes, it simulates $\mathcal{P}$ to process both the changes of the first and of the second half, two in each step. After $\log n$ changes, it simulated $\mathcal{P}$ for all changes after it computed the initial auxiliary data, so it can return the correct query answer.

The dynamic program $\mathcal{P}'$ now simulates multiple instances of $\mathcal{P}^*$, similarly as it was already done for [Datta et al. 2018a, Proposition 7] and [Schwentick et al. 2018, Theorem 8.1]. We call such a simulation a *thread*. After each change, $\mathcal{P}'$ starts a new thread. After $\log n$ changes occurred, and $\mathcal{P}'$ started $\log n$ more threads, the first thread finishes its computation and terminates, and $\mathcal{P}'$ can output the thread's query result. Like this, $\mathcal{P}'$ can answer the query from the time the very first thread finishes its computation, which is after the change number $\log n$ happened, and then arbitrarily long.

It remains to explain how $\mathcal{P}'$ gives the query result for the first $\log n$ changes. At this point we use that the initial input structure is empty, and that the underlying query is weakly domain independent. While processing the first $\log n$ single-tuple changes, the active domain of the input structure is very small. In other words, it contains many isolated elements, and these are not relevant for the query result. So, $\mathcal{P}'$ can actually simulate $\mathcal{P}^*$ on a much smaller input instance, for which the initialisation can be computed in far less time than for the full instance, and hence $\mathcal{P}^*$ can give the query answer way earlier than after its $\log n$-th change.

With these explanations in mind, we turn to the formal proof of Theorem 3.3.

Proof (of Theorem 3.3). Let $f : \mathbb{N} \to \mathbb{N}$ be first-order constructible with $f \in \mathcal{O}(n)$ and assume that an $\mathsf{AC}[f(n)]$ algorithm $\mathcal{A}$ and a dynamic program $\mathcal{P}$ witness that an almost domain independent dynamic query (q, Δ) is $(\mathsf{AC}[f(n)], f(n))$-maintainable. Using Proposition 3.2, it is sufficient to construct a dynamic program $\mathcal{P}'$ that witnesses $(q, \Delta) \in \mathsf{DynFO}(\leq, +, \times)$. For simplicity we restrict ourselves to graph queries and assume that $\Delta = \Delta_E$. The extension for the general case is straightforward.

The overall idea is to use a simulation technique similar to the ones used in [Datta et al. 2018a, Proposition 7] and in [Schwentick et al. 2018, Theorem 8.1]. We first present the computations performed by $\mathcal{P}'$ intuitively and later explain how they can be expressed in first-order logic. We consider each application of one change as a *time step* and refer to the graph after time step t as $G_t = (V, E_t)$. After each time step t, $\mathcal{P}'$ starts a thread that is in charge of answering the query at time point $t + f(n)$. Each thread works in two phases, each lasting $\frac{f(n)}{2}$ time steps. Roughly speaking, the first phase is in charge of simulating $\mathcal{A}$ and in the second phase $\mathcal{P}$ is used to apply all changes that occur from time step $t + 1$ to time step $t + f(n)$. Using $f(n)$ many threads, $\mathcal{P}'$ is able to answer the query from time point $f(n)$ onwards.

We now give more details on the two phases and describe afterwards how to deal with time points earlier than $f(n)$. For the first phase, we make use of the equality $\mathsf{AC}[f(n)] = \text{IND}[f(n)]$ [Immerman 1999]. Let ψ be an inductive formula that is applied $df(n)$ times, for some d, to get the auxiliary relations $\mathcal{A}(G, \leq)$ for a given graph G and the given order $\leq$. The program $\mathcal{P}'$ simply applies ψ to G_t, $2d$ times during each time step, and thus the result of $\mathcal{A}$ on $(G_t, \leq)$ is obtained after $\frac{f(n)}{2}$ steps. The changes that occur during these steps are not applied to G_t directly but rather stored in additional relations. We also store the order in which the changes occur.[2]

During the second phase the $\frac{f(n)}{2}$ already stored changes and the $\frac{f(n)}{2}$ changes that happen during the next $\frac{f(n)}{2}$ steps are applied to the state after phase 1, in the order of occurrence. To this end, it suffices for $\mathcal{P}'$ to apply two changes during each time step by simulating two update steps of $\mathcal{P}$. Since $\mathcal{P}$ can maintain q for $f(n)$ changes, at the end of phase 2, at time point $t + f(n)$, $\mathcal{P}'$ can give the correct query answer for q about $G_{t+f(n)}$.

The following auxiliary relations are used by thread i:

- a binary relation $\hat{E}_i$ that contains the edges currently considered by the thread,
- relations D_i^{INS} and D_i^{DEL} that cache edge insertions and deletions, respectively, in the order they occur,
- a relation $\hat{R}_i$ for each auxiliary relation R of $\mathcal{P}$ with the same arity,
- and a relation C_i that is used as a counter: it contains exactly one tuple which is interpreted as the counter value, according to its position in the lexicographic order induced by $\leq$.

When thread i starts its first phase at time point t, it sets $\hat{E}_i$ to E_t and the counter C_i to 0; its other auxiliary relations are empty in the beginning. Whenever a change $\textsc{ins}_E(\bar{a})$ (or $\textsc{del}_E(\bar{a})$) occurs, the thread updates its auxiliary relations as follows. The change is not applied directly, so $\hat{E}_i$ is not changed. Instead, the thread needs to store the change and remember the time step it occurred. To do so, the tuple $(\bar{c}, \bar{a})$ is inserted into D_i^{INS} (or D_i^{DEL}), where $\bar{c} \in C_i$ is the current counter value. Additionally, the counter C_i is incremented by one and the relations $\hat{R}_i$ are replaced by the result of applying their defining first-order formulas $2d$ times, as explained above.

When the counter value is at least $\frac{f(n)}{2}$, the thread is in its second phase and proceeds as follows. When a change occurs, it is stored exactly as in the first phase. Additionally, the two oldest stored changes $\delta_1(\bar{a}_1)$ and $\delta_2(\bar{a}_2)$ that are not already processed are applied, in that order. These changes are identified by the two lexicographically smallest tuples $\bar{c}_1, \bar{c}_2$

[2]This seems to be a bit exaggerated here, as it would be sufficient to store the difference between the actual and the considered edge relation. In later chapters we generalise this proof to "non-commutating" change operations where the order needs to be preserved, so we already proceed this way here.

such that tuples $(\bar{c}_1, \bar{a}_1)$ and $(\bar{c}_2, \bar{a}_2)$ exist in D_i^{INS} or D_i^{DEL}. To apply these changes, the thread simulates $\mathcal{P}$ on the edge set $\hat{E}_i$ and auxiliary relations $\hat{R}_i$, replaces the auxiliary relations accordingly and adjusts $\hat{E}_i$. Also, the tuples containing $(\bar{c}_1, \bar{a}_1)$ and $(\bar{c}_2, \bar{a}_2)$ are deleted from D_i^{INS} and D_i^{DEL}. Again, the counter C_i is incremented. If the counter value is $f(n)$, the thread's query result is used as the query result of $\mathcal{P}'$, and the thread stops. All steps are easily seen to be first-order expressible.

So far we have seen how $\mathcal{P}'$ can give the query answer from time step $f(n)$ onwards. For time steps earlier than $f(n)$ the approach needs to be slightly adapted as the program does not have enough time to simulate $\mathcal{A}$. The idea is that for time steps $t < f(n)$ the active domain is small and, exploiting the almost domain independence of q, it suffices to compute the query result with respect to this small domain extended by c isolated elements, where c is the constant from almost domain independence. The result on this restricted domain can afterwards be used to define the result for the whole domain.

Towards making this idea more precise, let n_0, b be such that $bn \geq f(n)$ for all $n \geq n_0$. We focus on explaining how $\mathcal{P}'$ handles structures with $n \geq n_0$, as small graphs with less than n_0 nodes can be dealt with separately.

The program $\mathcal{P}'$ starts a new thread at time $\frac{t}{2}$ for the graph $G_{\frac{t}{2}}$ with at most $\frac{t}{2}$ edges. Such a thread is responsible for providing the query result after t time steps, and works in two phases that are similar to the phases described above. It computes relative to a domain D_t of size $\min\{2t + c, n\}$, where c is the constant from (almost) domain independence. The size of D_t is large enough to account for possible new nodes used in edge insertions in the following $\frac{t}{2}$ change steps. The domain D_t is chosen as the first $|D_t|$ elements of the full domain (with respect to $\leq$). The program $\mathcal{P}'$ maintains a bijection π between the active domain D_G of the current graph G and the first $|D_G|$ elements of the domain to allow a translation between D_G and D_t.

The first phase of the thread for t starts at time point $\frac{t}{2} + 1$ and applies ψ for $8bcd$ times during each of the next time steps. This simulation of $\mathcal{A}$ is finished after at most $\frac{(2t+c)b}{8bc} \leq \frac{t}{4}$ time steps, and therefore the auxiliary relations are properly initialised at time point $\frac{3t}{4}$.

In the second phase, starting at time step $\frac{3t}{4} + 1$ and ending at time step t, the changes that occurred in the first phase are applied, two at a time. The thread is then ready to answer q at time point t. Since at time t at most $2t$ elements are used by edges, the almost domain independence of q guarantees that the result computed by the thread relative to D_t coincides with the D_t-restriction of the query result for $\pi(G_t)$. The query result for G_t is obtained by translating the obtained result according to π^{-1}, and extending it to the full domain. More precisely, a tuple $\bar{t}$ is included in the query result, if it can be generated from a tuple $\bar{t}'$ of the restricted query result by replacing elements from $\pi^{-1}(D_t) \setminus \text{adom}(G_t)$ by elements from $V \setminus \text{adom}(G_t)$ (under consideration of equality constraints among these elements). Again, all steps are easily seen to be first-order definable using the auxiliary relations from above.

The above presentation assumes a separate thread for each time point and each thread uses its own relations. These threads can be combined into one dynamic program as follows. Since at each time point at most $f(n)$ threads are active, we can number them in a round robin fashion with numbers $1, \dots, f(n)$ that we can encode by tuples of constant arity. The arity of all auxiliary relations is incremented accordingly and the additional dimensions are used to indicate the number of the thread to which a tuple belongs. $\square$

3.3 Outlook and bibliographic remarks

The main contribution of this chapter is the introduction of the Muddling technique: in order to show membership in DynFO for almost domain independent dynamic queries, it is sufficient to show $(\mathsf{AC}[f(n)], f(n))$-maintainability for some at most linear function f, so, that the dynamic query can be maintained for $f(n)$ many changes after an $\mathsf{AC}[f(n)]$ initialisation. This technique will be used in Section 4.1 to show that all MSO-definable queries are in DynFO for graphs with bounded treewidth, and adapted to different classes of complex changes in Section 5.5 and Section 6.3. The results of [Datta et al. 2018b], which are partly presented in Section 6.3, rely on the insight that the dynamic program from [Datta et al. 2018a], proving that REACH is in DynFO under single-edge changes, can be conceptually simplified by utilising the Muddling technique.

We believe that the technique will find further applications, both for the class DynFO but also in similar settings. The main result of Section 4.1, that MSO-definable queries can be maintained in DynFO for graphs with bounded treewidth, is used in preliminary work together with Samir Datta, Siddharth Iyer, Anish Mukherjee and Thomas Zeume to show that *approximate* solutions to certain MSO-definable optimisation problems can be maintained for planar graphs and extensions thereof. Other preliminary work together with Samir Datta, Anish Mukherjee, Shreyas Srinivas and Thomas Zeume suggests that the Muddling Theorem can be adapted for the class DynQF, introduced by Hesse [2003b], which is defined via dynamic programs with quantifier-free update formulas that may use functions. A variant of the Muddling Theorem is used in [Schmidt et al. 2020] to obtain maintainability results in a dynamic setting of parameterised complexity.

Whenever one studies a particular setting within or extending the dynamic complexity framework, it might be worthwhile to check whether a variation of the Muddling Theorem holds in this setting. In particular, the Muddling Theorem can be adapted for the FOIES framework that allows changes that modify the domain by one element, using the result of Datta et al. [2018a] that also Proposition 3.2 holds in this setting.

We sketch the necessary adaptations towards the proof of Theorem 3.3. At first, the threads need to simulate the dynamic program on a slightly larger domain to be prepared for a growing domain. They can use a unary relation to differentiate between their own domain and the domain of the simulation. Secondly, observe that a thread might need to give the query result for more than one time step. If at time point i the input structure has a domain of size n and at time point $i+1$ the domain is of size $n+1$, then the thread started at step i gives the answer after $f(n)$ further steps, while the thread from step $i+1$ needs $f(n+1)$ steps. As $f \in \mathcal{O}(n)$, the difference between these numbers is just some constant d. One can therefore run $d+1$ copies of each thread such that the j-th copy compresses the first j steps of the original thread into one step and by this achieves the necessary speed-up.

The impact of the Muddling Theorem as well as of Proposition 3.2, originally from [Datta et al. 2018a], stating that $\mathsf{DynFO}(\leq, +, \times) = \mathsf{DynFO}$ when only considering almost domain independent queries, suggests that we need more results of the form "all dynamic queries in some dynamic complexity class $\mathcal{C}$ are already in a seemingly weaker class $\mathcal{C}'$". As other examples of such results, Zeume and Schwentick [2017] show for DynFO and some of its subclasses that restricting the syntax of update formulas, like disallowing disjunction or negation, often does not change the expressive power of the corresponding dynamic programs.

An important open problem with respect to high-level maintenance strategies is to determine how dynamic programs can be composed. There is a dynamic program that maintains Reach; if another dynamic program maintains some binary relation R, under which conditions is there a dynamic program that maintains the transitive closure of R?

Bibliographic remarks

The Muddling Theorem was first published, in the form of Corollary 3.4, in [Datta et al. 2017] as joint work with Samir Datta, Anish Mukherjee, Thomas Schwentick and Thomas Zeume. It was developed from a strategy to prove that all AC^1 queries are in DynFO under change operations that are defined by parameter-free first-order formulas [Schwentick et al. 2017a]; in this thesis, that result is included as Corollary 5.20 and proved using the (appropriately adapted) Muddling technique. The idea of simulating multiple instances of a dynamic program in parallel originates from [Datta et al. 2015; 2018a].

A more general version of Corollary 3.4 was published in [Datta et al. 2018b], in the present form of Theorem 3.3 it is published in [Datta et al. 2019].

CHAPTER 4

Expressibility under single-tuple changes

If we asked the user of a database management system how he or she would like to be able to modify a table of a database, a first, maybe baffled, answer would probably be: "I need to add tuples. And later on I need to remove some of them."

The insertion or deletion of a single tuple is the smallest possible change to a database, but these change operations come, at least in principle, with universal expressibility: having a database schema fixed, every database instance can be transformed into every other instance by a series of insertions and deletions of single tuples.[1]

So, single-tuple changes are a natural starting point for studying maintenance of query results under modifications of the input database, and consequently the vast majority of research articles in the field of dynamic complexity concentrates on this model. Sometimes the possible changes are even further restricted to only single-tuple insertions, or, less frequently, to only single-tuple deletions.

We first summarise known results on the expressive power of DynFO and its subclasses under single-tuple changes. Then we formulate the goals and the contributions of this chapter, and present its outline.

The prototypical queries that cannot be expressed in first-order logic are Parity and Reach. These queries are therefore natural candidates to study in a dynamic setting. Maintenance of Parity under single-tuple changes is rather trivial (cf. Example 2.11), as observed already by Patnaik and Immerman [1997]. For the reachability query it took a longer time until maintainability was established.

One line of research studied maintainability of restrictions of the reachability query,

[1]Of course, a database system that only supported insertions and deletions of single tuples would neither be efficient nor comfortable to use, and therefore a typical database management system supports more succinct means of data manipulation. Some of them will be captured by the change operations we consider in Chapter 5 and Chapter 6.

with the result that reachability in directed acyclic graphs [Dong and Su 1995; Patnaik and Immerman 1997] as well as undirected graphs [Patnaik and Immerman 1997; Dong and Su 1998; Hesse 2003b; Grädel and Siebertz 2012] is in DynFO or even weaker dynamic settings, under single-edge changes. Reachability in deterministic graphs can even be maintained with quantifier-free dynamic programs [Hesse 2003b]. Orthogonally, it was shown that the general reachability query for arbitrary directed graphs can be maintained in extensions of DynFO [Hesse 2003a; Datta et al. 2014]. Finally, Datta et al. [2015; 2018a] proved that $(\textsc{Reach}, \Delta_E)$ is in DynFO.

Several other queries have been studied in the dynamic setting. One notable result concerns the query TreeIsomorphism that asks whether two trees given as input are isomorphic. Etessami [1998] showed that this query can be maintained in DynFO under single-edge changes. Further results, often obtained by reductions to other queries or variations of already known dynamic programs, can be found for example in [Patnaik and Immerman 1997; Grädel and Siebertz 2012].

In addition to the NL-complete query Reach, some queries that are complete for (presumably) larger complexity classes can be maintained in DynFO under single-edge changes. One example is the query D2LReach on acyclic graphs [Weber and Schwentick 2007], which asks whether in an acyclic, labelled graph with edge labels from $\{(,),[,]\}$ there is a (not necessarily simple) path between two given nodes such that the concatenated labels on the edges of this path spell a word from the Dyck language D_2, that is, the set of well-parenthesized strings with two types of parentheses. This query is complete for the class LOGCFL, which lies between NL and AC^1. Another example is a padded variant of the P-complete query AlternatingReach [Patnaik and Immerman 1997].

Such results do not directly imply that all queries from complexity classes as NL, LOGCFL or even P can be maintained dynamically under single-tuple changes, because DynFO is not known to be closed under the usually used reductions. However, there are some results that whole classes of queries can be maintained in DynFO under single-tuple changes. Examples on strings include all regular languages [Patnaik and Immerman 1997], which can already be maintained using only unary auxiliary relations [Hesse 2003b] or with quantifier-free dynamic programs [Gelade et al. 2012], and all context-free languages [Gelade et al. 2012].

A number of results were obtained for path queries on labelled graphs. These queries ask, similar as the query D2LReach mentioned above, whether there is a path between two nodes such that its edge labels form a word from a given target language. If the target language is regular, we call such a query a *regular path query*. Regular path queries can be maintained under insertions [Dong et al. 1995], and also under deletions of labelled edges [Datta et al. 2018a]. Some extensions of regular path queries that can be maintained in DynFO, at least for restricted classes of graphs, are identified by Zeume [2015] and Muñoz et al. [2016].

The first goal of this chapter is to extend these maintainability results.

Goal 4.1. Show that specific (classes of) queries can be maintained dynamically under single-tuple changes.

To fully understand the expressive power of DynFO we are interested in proving that some dynamic query cannot be maintained in DynFO. Unfortunately, results of this kind are rare. Of course, every query q that is in DynFO under single-tuple insertions is contained in P, as from a dynamic FO-program $\mathcal{P}$ one can obtain a polynomial-time algorithm that for any input instance $\mathcal{I}$ simulates $\mathcal{P}$ under the change sequence that inserts all tuples from $\mathcal{I}$. Therefore, no EXPTIME-hard query is in DynFO under single-tuple changes.

Besides this trivial lower bound, no inexpressibility results for the full class of DynFO are known. Grädel and Siebertz [2012] prove that in their unordered setting of DynFO, where the initial input database is non-empty and the initial auxiliary relations are defined in some logic, TreeIsomorphism cannot be maintained, and neither whether two unary relations have the same size, but these results rely on the weakness of the initialisation. This is underscored by the fact that (TreeIsomorphism, Δ_E) *is* in DynFO as to our definition that only allows initially empty databases [Etessami 1998].

There are two further prominent restrictions of DynFO that allow for lower bound proofs: restricting the arity of the auxiliary relations and restricting the update formulas syntactically.

When only unary auxiliary relations are allowed, several queries cannot be maintained in DynFO under single-tuple changes, including Reach [Dong and Su 1998; Dong et al. 2003] and some non-regular languages [Zeume 2015; Vortmeier 2013].

The syntactically restricted class DynProp, containing dynamic queries that can be maintained by dynamic programs with quantifier-free update formulas, was introduced by Hesse [2003b], who also shows that such programs are able to maintain the reachability query on deterministic graphs under single-edge changes. We believe that (Reach, Δ_E) is not in DynProp, but so far this is only known for the special case of at most binary auxiliary relations, and in the setting with non-empty initial graphs and logically defined initialisation [Zeume and Schwentick 2015]. The alternating reachability query cannot be maintained, not even for layered graphs[2] with a constant number of layers [Gelade et al. 2012], which is a first-order definable query.

On strings, the expressive power of DynProp is characterised completely: exactly the regular languages can be maintained [Gelade et al. 2012]. While several proof techniques and inexpressibility results for DynProp are already available [Gelade et al. 2012; Zeume and Schwentick 2015; Zeume 2017], there is so far no characterisation of the dynamic graph queries in DynProp.

Goal 4.2. Understand the expressive power of DynProp on graphs.

Contributions Towards Goal 4.1 we prove that the class of MSO-definable queries can be maintained under single-edge changes on graphs of bounded treewidth, that is, we give a dynamic version of Courcelle's Theorem which states that these queries can statically be evaluated in linear time. This result is obtained independently and exceeds the result of Bouyer-Decitre et al. [2017], who show that MSO-definable queries can be

[2]This means that the vertex set V is partitioned into layers $V_1, \dots, V_d$. Edges $(u, v) \in E$ are only between adjacent layers, so if $u \in V_i$ for some $i < d$, then $v \in V_{i+1}$.

maintained under the condition that an initially computed tree decomposition stays valid throughout the dynamic process.

Our result depends on the Muddling technique as introduced in Chapter 3, as well as on a constructive Feferman-Vaught-type composition theorem for relational structures that have an intersection of logarithmic size. This composition theorem might also be useful for other applications.

Later on, we recapitulate the proof technique of Zeume [2017] for proving inexpressibility results for DynProp and encapsulate its essence in a technical lemma. This is then used, together with a variation of the list technique for DynProp from Gelade et al. [2012], to show that DynProp has a strict arity hierarchy for Boolean graph queries: for each k there is a Boolean graph query that cannot be maintained under single-edge changes in DynProp with k-ary auxiliary relations, but that can be maintained with $(k+1)$-ary auxiliary relations. This is a step towards Goal 4.2. As a corollary, we identify a $\mathsf{AC}^0[2]$ query that cannot be maintained in DynProp under single-edge changes.

The results presented in this chapter are joint work with Thomas Schwentick, Thomas Zeume, Samir Datta and Anish Mukherjee. Detailed bibliographic remarks are given at the end of this chapter.

Outline This chapter consists of two main parts. In Section 4.1 we prove the dynamic version of Courcelle's Theorem: every MSO-definable query can be maintained in DynFO under single-edge changes on graphs of bounded treewidth. The arity hierarchy for DynProp is shown in Section 4.2. We conclude with an outlook and bibliographic comments in Section 4.3.

4.1 A dynamic version of Courcelle's Theorem

In the introduction of this chapter we have seen that a wide range of queries can be maintained in DynFO under single-tuple changes, including Reach and TreeIsomorphism. These dynamic queries serve as showcase examples for the expressive power of DynFO and of the dynamic approach in general: in the static setting, these problems are complete for classes that are much stronger than FO or AC^0, namely NL for Reach and NC^1 or LOGSPACE for TreeIsomorphism, depending on the input representation [Jenner et al. 1998]. In the dynamic setting, the complexity to update a result drops to FO.

For these impressive maintainability results the corresponding dynamic programs are constructed in rather complicated, single-purpose proofs. Ideally, one would have a kind of compiler that automatically generates a dynamic program given a formal definition of a query, without the need of problem-specific "manual" work. This goal calls for proofs that whole classes of dynamic programs can be maintained in DynFO. In algorithmic research, results of this form are also called *meta-theorems*, see [Grohe 2014] for some examples. The result that all regular languages can be maintained in DynProp [Gelade et al. 2012] can be seen as such a meta-theorem. The class of regular languages is exactly the class of languages expressible in MSO, so, for every Boolean query on strings that is defined by an MSO sentence we can automatically deduce a quantifier-free dynamic program that maintains this query under changes of single positions of a string. A similar result holds for MSO-definable decision problems on trees, because all regular tree languages are in DynFO [Gelade et al. 2012], and MSO on trees coincides with regular tree languages.

The "archetypal" [Grohe 2014, p. 16] example of a meta-theorem is Courcelle's Theorem [Courcelle 1990], stating that all MSO-definable decision problems for graphs of bounded treewidth can be decided in linear time. The treewidth of a graph will be formally introduced next, in Subsection 4.1.1, for now it suffices to think of graphs that are "close" to being a tree. An algorithm for an MSO-definable problem P for graphs of bounded treewidth basically decomposes the input graph G into a tree T whose nodes are labelled with small subgraphs of G, called a tree decomposition, and then applies an appropriately extended version of the algorithm for P for trees.

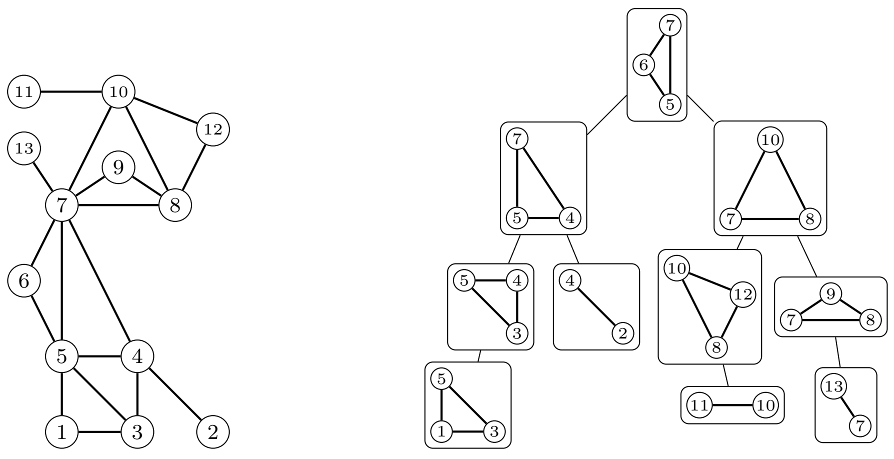
There is also a parallel version of Courcelle's Theorem, proven by Elberfeld et al. [2010], that states that all MSO-definable problems on graphs of bounded treewidth can be solved with logarithmic space.

The main contribution of this section is a dynamic version of Courcelle's Theorem: we will prove that all MSO-definable queries can be maintained under single-tuple changes in DynFO, for graphs of bounded treewidth. The generalisation from trees to graphs of bounded treewidth is more complex in the dynamic setting than in the static setting, and in particular this result is not an easy extension of the result by Gelade et al. [2012] that MSO properties on trees can be maintained in DynFO under single-edge changes. This is because a tree decomposition might change drastically under edge insertions, and it is unclear how (1) one can maintain a tree decomposition, and (2) how auxiliary information regarding an old tree decomposition can be used for a heavily changed new decomposition. To deal with this problem, we show that a once computed tree decomposition can be used for a certain period of time, even if it is not a valid tree decomposition any more for the current input graph, and then apply the Muddling technique presented in Chapter 3.

The outline of this section is as follows. We start by defining the notion of treewidth as well as tree decompositions in Subsection 4.1.1. Then we introduce the main ideas for maintaining MSO-definable queries by means of the example problem 3Col in Subsection 4.1.2. Afterwards, in Subsection 4.1.3, we give the proof for all MSO-definable queries. This subsection is again divided into three parts. At first we give a Feferman-Vaught-like composition theorem that allows us to infer the MSO type of a graph given the MSO types of a set of its subgraphs that have a logarithmic number of nodes in common. Afterwards we first show the result for MSO-definable decision problems, then we adapt the result towards optimisation problems.

4.1.1 Treewidth and tree decompositions

Many problems that are NP-complete for arbitrary graphs, like 3Col and VertexCover, are very easy when the input graphs are restricted to be trees. Moreover, it was observed that the corresponding algorithms can be generalised for classes of graphs that are "close" to being trees (cf. Arnborg et al. [1991]). The treewidth of a graph, introduced by Robertson and Seymour [1986], is a parameter that formalises how "tree-like" a certain



(a) The graph G. (b) The tree decomposition (T, B).

Figure 4.1: A graph G and a tree decomposition (T, B) of G of width 2.

graph is. Intuitively, a graph G has low treewidth if one can decompose G into subgraphs $G_1, \dots, G_m$ of small size and arrange these subgraphs as a tree T such that if a node of G appears in two subgraphs G_i, G_j, then it also appears in every subgraph on the unique path in T that connects G_i and G_j.

We make this more formal now. Let $G = (V, E)$ be a graph. A *tree decomposition* (T, B) of G consists of a (rooted, directed) tree $T = (I, F, r)$, with (tree) nodes I, (tree) edges F, a distinguished root node $r \in I$, and a function $B \colon I \to 2^V$ such that

(1) the set $\{i \in I \mid v \in B(i)\}$ is non-empty for each node $v \in V$,
(2) there is an $i \in I$ with $\{u, v\} \subseteq B(i)$ for each edge $(u, v) \in E$, and
(3) the subgraph $T[\{i \in I \mid v \in B(i)\}]$ is connected for each node $v \in V$.

We refer to the number of children of a node i of T as its *degree*, and to the set $B(i)$ as its *bag*. The *width* of a tree decomposition is defined as the maximal size of a bag minus 1. The *treewidth* of a graph G is the minimal width among all tree decompositions of G. An example of a tree decomposition is depicted in Figure 4.1.

It is in general NP-complete to determine whether some graph G has treewidth k, given both G and k as input [Arnborg et al. 1987]. However, if k is a fixed constant, on input G one can compute in linear time a tree decomposition of width k for G (or report that no such decomposition exists) [Bodlaender 1996], and then answer any problem definable in (certain extensions of) MSO in linear time as well [Courcelle 1990]. The latter result, known as Courcelle's Theorem, is a prime example of an algorithmic meta-theorem, and the main purpose of this section is to prove its dynamic counterpart in the setting of DynFO.

For our purposes we will need tree decompositions of a certain shape. We say that a tree decomposition (T, B) of a graph G with n nodes is *d-nice*, for some $d \in \mathbb{N}$, if

(1) T has depth at most $d \log n$,
(2) the degree of the nodes is at most 2, and

(3) all bags are distinct.

Often we do not make the constant d explicit and just speak of *nice* tree decompositions.

We will use that every graph of treewidth k has a nice tree decomposition of width slightly more than k. This is formalised in the following lemma, which builds on [Elberfeld et al. 2010, Lemma 3.1].

Lemma 4.1. *For every $k \in \mathbb{N}$ there is a constant $d \in \mathbb{N}$ such that for every graph of treewidth k, a d-nice tree decomposition of width $4k+5$ can be computed in logarithmic space.*

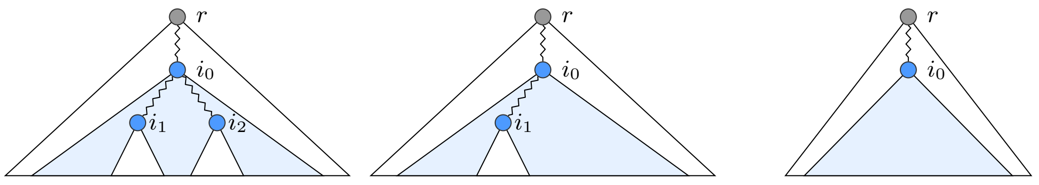
Proof. Let G be a graph of treewidth k. By [Elberfeld et al. 2010, Lemma 3.1] a tree decomposition (T, B) of width $4k+3$ can be computed in logarithmic space, such that each non-leaf node has degree 2 and the depth is at most $d \log n$, for a constant d that only depends on k. To obtain a tree decomposition with distinct bags, we compose this algorithm with three further algorithms, each reading a tree decomposition (T, B) and transforming it into a tree decomposition (T', B') with a particular property. Since each of the four algorithms requires only logarithmic space, the same holds for their composition.

The first transformation algorithm removes unnecessary leaf nodes, that is, produces a tree decomposition in which for each leaf node i and its parent $p(i)$ it holds $B(i) \not\subseteq B(p(i))$. In particular, after this transformation, each bag of a leaf node i contains some graph node, denoted $u(i)$, that does not appear in any other bag. This transformation inspects each node i separately in a bottom-up fashion, and removes it if (1) $B(i) \subseteq B(p(i))$ and (2) i is a leaf or all descendants of i are to be removed as well. Clearly, logarithmic space suffices for this.

The first transformation might introduce inner nodes of degree 1. The second transformation (inductively) removes such nodes i with parent node $p(i)$ and child i' and inserts an edge between $B(p(i))$ and $B(i')$, whenever $B(i) \subseteq B(p(i))$ or $B(i) \subseteq B(i')$ holds. For this transformation, only one linear chain of nodes in T has to be considered at any time and therefore logarithmic space suffices again. Clearly, the connectivity property is not affected by these contractions.

The third transformation adds to every bag of an inner node i the nodes $u(i_1)$ and $u(i_2)$, guaranteed to exist by the first transformation, of the leftmost and rightmost leaf nodes i_1 and i_2 of the subtree rooted at i, respectively. Here, we assume the children of every node to be ordered by the representation of T as input to the algorithm. After this transformation, each node of degree 2 has a different bag than its two children thanks to the addition of $u(i_1)$ and $u(i_2)$. Each node of degree 1 has a different bag than its child, since this was already the case before (and to both of them the same two nodes might have been added). Altogether, all bags are pairwise distinct and the bag sizes have increased by at most 2.

We emphasise that, whenever a leftmost graph node $u(i_1)$ is added to $B(i)$, it is also added to all bags of nodes on the path from i to i_1 and therefore the connectivity property is not corrupted. It is easy to see that the third transformation can also be carried out in logarithmic space.



(a) A proper triangle. (b) A unary triangle. (c) An open triangle.

Figure 4.2: Illustration of (a) a proper triangle (i_0, i_1, i_2), (b) a unary triangle (i_0, i_1, i_1), and (c) an open triangle (i_0, i_0, i_0). The blue shaded area is the part of the tree contained in the triangle.

The three presented algorithms never increase the depth of a tree decomposition, so the final result is a d-nice tree decomposition for G of width $4k + 5$. $\square$

We only consider nice tree decompositions, and due to property (3) of these decompositions we can identify bags with nodes from I. So, formally, a nice width-k tree decomposition can be encoded as a $(2k+2)$-ary relation T that contains a tuple $(b_1^1, \dots, b_1^{k+1}, b_2^1, \dots, b_2^{k+1})$ if and only if there are nodes $i_1, i_2 \in I$ with $(i_1, i_2) \in F$ such that $B(i_1) = \{b_1^1, \dots, b_1^{k+1}\}$ and $B(i_2) = \{b_2^1, \dots, b_2^k\}$.

For two nodes i, i' of I, we write $i \preceq i'$ if i' is in the subtree of T rooted at i and $i \prec i'$ if, in addition, $i' \neq i$. We will often consider subtrees of a tree decomposition that are determined by up to three tree nodes. A *triangle* δ of T is a triple (i_0, i_1, i_2) of nodes from I such that $i_0 \preceq i_1$, $i_0 \preceq i_2$, and (1) $i_1 = i_2$ or (2) neither $i_1 \preceq i_2$ nor $i_2 \preceq i_1$. In case of (2) we call the triangle *proper*, in case of (1) *unary*, unless $i_0 = i_1 = i_2$ in which we call it *open* (see Figure 4.2 for an illustration). The subtree $T(\delta)$ *induced* by a triangle consists of all nodes j of T for which the following holds:

(i) $i_0 \preceq j$,
(ii) if $i_0 \prec i_1$ then $i_1 \not\preceq j$, and
(iii) if $i_0 \prec i_2$ then $i_2 \not\preceq j$.

That is, for a proper or unary triangle, $T(\delta)$ contains all nodes of the subtree rooted at i_0 which are not below i_1 or i_2. For an open triangle $\delta = (i_0, i_0, i_0)$, $T(\delta)$ is just the subtree rooted at i_0.

Each triangle δ induces a subgraph $G(\delta)$ of G as follows: $V(\delta)$ is the union of all bags of $T(\delta)$. By $B(\delta)$ we denote the set $B(i_0) \cup B(i_1) \cup B(i_2)$ of *interface nodes* of $V(\delta)$. All other nodes in $V(\delta)$ are called *inner nodes*. The edge set of $G(\delta)$ consists of all edges of G that involve at least one inner node of $V(\delta)$.

4.1.2 Showcase 3-colourability for graphs of bounded treewidth

Before we present our main result of this section, we introduce the main proof ideas by means of a specific problem and show that the 3-colourability problem 3Col for graphs of bounded treewidth can be maintained in DynFO under single-edge changes. Given an undirected graph, 3Col asks whether its nodes can be coloured with three colours such that adjacent nodes have different colours. More precisely, we show the following result.

Theorem 4.2. *For every k, $(3\textsc{Col}, \Delta_E)$ is in* DynFO *for graphs with treewidth at most k.*

To prove this theorem, thanks to Corollary 3.4 and the fact that 3Col is almost domain independent, it suffices to show that $(3\textsc{Col}, \Delta_E)$ is $(\mathsf{AC}^1, \log n)$-maintainable for graphs with treewidth at most k. In a nutshell, our approach can be summarised as follows.

The AC^1 initialisation computes a nice tree decomposition (T, B) of width at most $4k+5$ and maximum bag size $\ell \overset{\text{def}}{=} 4k+6$, as well as information about the 3-colourability of certain subgraphs of G. More precisely, it computes, for each triangle δ of T and each 3-colouring C of the interface nodes $B(\delta)$, whether there exists a colouring C' of the inner nodes of $G(\delta)$ such that all edges involving at least one inner node are consistent with $C \cup C'$.

To determine whether the input graph is 3-colourable for the next $\log n$ changes, the dynamic program only maintains a set S of *special* bags: for each *affected* graph node v that is an endpoint of a changed (i.e. deleted or inserted) edge, S contains one bag in which v occurs. Also, if two bags are special, their least common ancestor is considered special and is included in S. After $\log n$ changes there are at most $4 \log n$ special bags. With the help of the auxiliary information computed by the initialisation, a first-order formula φ can test whether G is 3-colourable as follows. By existentially quantifying 8ℓ variables and interpreting the valuation as a bit string of length $8\ell \log n$ using BIT, the formula can choose two bits of information for each of the at most $4\ell \log n$ nodes in special bags. For each such node, these two bits are interpreted as encoding of one of three colours. The formula φ can check that this colouring of the special bags can be extended to a colouring of G, because G is essentially a union of subgraphs induced by triangles defined by special bags, and the auxiliary relations provide the information whether a colouring of the interface nodes can be extended to the graph induced by any triangle.

Before we give a detailed proof, we need some more notation. Let $G = (V, E)$ be a graph and (T, B) with $T = (I, F, r)$ a nice tree decomposition with bags of size at most ℓ. A *colouring* of a set U of nodes is a mapping from U to $\{1, 2, 3\}$. An edge (u, v) is *properly coloured* if u and v are mapped to different colours. For a triangle δ, we say that a colouring C of the set $B(\delta)$ of interface nodes is *consistent*, if there exists a colouring C' of the inner vertices of $G(\delta)$ such that all edges of $G(\delta)$ are properly coloured by $C \cup C'$. Recall that $G(\delta)$ only contains edges that involve at least one inner vertex.

We say that a tuple $\bar{v}(i) = (v_1, \ldots, v_\ell)$ *represents* a tree node $i \in I$ (or, the bag $B(i)$) if $B(i) = \{v_1, \ldots, v_\ell\}$. A tuple $\bar{v}(\delta) = (\bar{v}(i_0), \bar{v}(i_1), \bar{v}(i_2))$ *represents* the triangle $\delta = (i_0, i_1, i_2)$. If $\bar{v}(\delta) = (v_1, \ldots, v_{3\ell})$ represents a triangle δ and $\bar{c}$ is a tuple from $\{1, 2, 3\}^{3\ell}$ such that $c_j = c_{j'}$ whenever $v_j = v_{j'}$ for $j, j' \in \{1, \ldots, 3\ell\}$, we write $C_{\bar{c},\bar{v}}$ for the colouring of $B(\delta)$ defined by $C_{\bar{c},\bar{v}}(v_j) = c_j$, for every $j \in \{1, \ldots, 3\ell\}$.

Proof (of Theorem 4.2). Let $G = (V, E)$ be a graph of treewidth at most k. The AC^1 initialisation first computes a d-nice tree decomposition (T, B) with bags of size at most $\ell \overset{\text{def}}{=} 4k + 6$, for the constant d guaranteed to exist by Lemma 4.1, and the predecessor

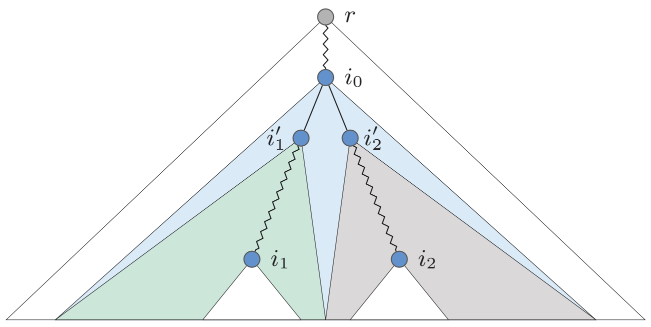


Figure 4.3: Illustration of the inductive step in the computation of colourability information for triangles in the proof of Theorem 4.2.

relation $\preceq$ of T. Also, it initialises the relations $\leq$ and BIT. Next, it computes the following auxiliary relations in a bottom-up fashion with respect to $T = (I, F, r)$. For each tuple $\bar{c} \in \{1, 2, 3\}^{3\ell}$ the auxiliary relation $R_{\bar{c}}$ contains all tuples $\bar{v}(\delta)$ from $V^{3\ell}$ that represent some triangle δ such that $C_{\bar{c},\bar{v}}$ is a consistent colouring of $B(\delta)$.

The auxiliary relations are computed inductively and bottom-up, that is, the auxiliary information for a tuple representing a triangle (i_0, i_1, i_2) is computed by using the information for the triangles rooted at the two children of i_0. It will be easy to see that each inductive step can be defined by a first-order formula and, since T has depth $d \log n$, the induction reaches a fixpoint after $d \log n$ iterations. Therefore the initialisation is in $\text{IND}[\log n] = \mathsf{AC}^1$. We recall that triangles can be open, unary or proper, depending on whether they are induced by a single bag, by two, or by three bags.

For the base case, a tuple $\bar{v}$ representing an open triangle corresponding to a leaf of T is in $R_{\bar{c}}$ if and only if, for each $j \in \{1, \ldots \ell\}$, $c_j = c_{\ell+j} = c_{2\ell+j}$, since the graph induced by this triangle has no inner nodes and therefore contains no edges.

The inductive cases are straightforward. We only describe in detail the case of a proper triangle $\delta = (i_0, i_1, i_2)$ where i_1 and i_2 are in different subtrees of i_0; the other cases are similar. Let i'_1 and i'_2 be the two children of i_0 such that $i'_1 \preceq i_1$ and $i'_2 \preceq i_2$. Figure 4.3 illustrates this situation. By the induction hypothesis, the auxiliary information for all triangles rooted at i'_1 and i'_2 has already been computed. A tuple $\bar{v} \overset{\text{def}}{=} \bar{v}(\delta)$ is in some $R_{\bar{c}}$, if there are tuples $\bar{d}, \bar{e} \in \{1, 2, 3\}^{3\ell}$ such that $\bar{u} \overset{\text{def}}{=} \bar{v}((i'_1, i_1, i_1)) \in R_{\bar{d}}$, $\bar{w} \overset{\text{def}}{=} \bar{v}((i'_2, i_2, i_2)) \in R_{\bar{e}}$ and it holds that

- $C_{\bar{c},\bar{v}}$ and $C_{\bar{d},\bar{u}}$ coincide on $B(i_0) \cap B(i'_1)$ and on $B(i_1)$,
- $C_{\bar{c},\bar{v}}$ and $C_{\bar{e},\bar{w}}$ coincide on $B(i_0) \cap B(i'_2)$ and on $B(i_2)$, and
- all edges over $B(i_0) \cup B(i'_1) \cup B(i'_2) \cup B(i_1) \cup B(i_2)$ of which at least one node is not in $B(i_0) \cup B(i_1) \cup B(i_2)$ are properly coloured by $C_{\bar{c},\bar{v}} \cup C_{\bar{d},\bar{u}} \cup C_{\bar{e},\bar{w}}$.

We next describe how a dynamic program can maintain 3-colourability for $\log n$ change steps, starting from the above initial auxiliary relations, with the help of an additional ℓ-ary relation S and another binary relation N. Whenever an edge (u, v) is inserted into or deleted from E, we consider both u and v as *affected*. With each affected graph node v

we associate a bag $B(i)$ of the tree decomposition such that $v \in B(i)$. We call such a bag *special*, as well as all graph nodes it contains. Furthermore, if the tree node i is the least common ancestor of two nodes i_1, i_2 such that $B(i_1)$ and $B(i_2)$ are special, then the bag $B(i)$ becomes special as well. We call a triangle $\delta = (i_0, i_1, i_2)$ of T *clean* if no tree node i in $T(\delta)$ is associated with a special bag $B(i)$, apart from i_0, i_1, i_2. The dynamic program keeps track of all special bags using the relation S which contains all tuples $\bar{v}$ that represent some special bag. Additionally, it maintains a bijection between the first $\ell|S|$ nodes of V with respect to the linear order $\leq$ and the graph nodes in S, encoded in the relation N. The updates of the auxiliary relations S and N are clearly first-order expressible.

It remains to describe how a first-order formula can check 3-colourability of G given the relation S, N and $R_{\bar{c}}$. Within $\log n$ changes at most $2 \log n$ graph nodes can be affected, resulting in at most $4 \log n$ special bags altogether, since for each new special bag at most one other bag can become special as the least common ancestor of special bags. That means that the set Z of graph nodes occurring in some tuple of S contains at most $4\ell \log n$ nodes.

A colouring of Z can be represented by $8\ell \log n$ bits and can thus be "guessed" by a first-order formula, by existentially quantifying over 8ℓ variables $x_1, \dots, x_{4\ell}, y_1, \dots, y_{4\ell}$. More precisely, the j-th bits of x_r and y_r together represent the colour of the special node at position $(r-1)\log n + j$ with respect to the linear order represented by N. The first-order formula can easily check that the colouring C of S represented by $x_1, \dots, x_{4\ell}, y_1, \dots, y_{4\ell}$ is consistent for edges between special nodes and that for each clean triangle of T induced by special bags it can be extended to a consistent colouring of the inner nodes. The latter information is available in the relations $R_{\bar{c}}$. □

4.1.3 MSO-definable queries on graphs of bounded treewidth

In this subsection we generalise the proof strategy from Theorem 4.2 and prove the main result of this section, a dynamic version of Courcelle's Theorem: all MSO-definable properties can be maintained in DynFO for graphs with bounded treewidth. More precisely, we start with the model checking problem MC_φ that asks whether a given graph G satisfies some fixed MSO sentence φ, that is, whether $G \models \varphi$ holds. So, we show that MSO-definable decision problems can be maintained in DynFO, for graphs with bounded treewidth.

Theorem 4.3. *For every* MSO *sentence φ and every k,* $(\mathrm{MC}_\varphi, \Delta_E)$ *is in* DynFO *for graphs with treewidth at most k.*

In addition to decision problems, we also prove that MSO-definable optimisation problems can be maintained in DynFO for graphs with bounded treewidth. An MSO-definable optimisation problem OPT_φ is induced by an MSO formula $\varphi(X_1, \dots, X_m)$ with free set variables $X_1, \dots, X_m$. Given a graph G with node set V, it asks for sets $A_1, \dots, A_m \subseteq V$ of minimal[3] size $\sum_{i=1}^{m} |A_i|$ such that $G \models \varphi(A_1, \dots, A_m)$. Examples

[3]We only deal with minimisation problems here. The adaptation to maximisation problems is straightforward.

(with $m = 1$) for such problems are the vertex cover problem and the dominating set problem.

We require from a dynamic program for such a problem that it maintains unary query relations $\text{Ans}_1, \ldots, \text{Ans}_m$ that store, at any time, an optimal solution for the current graph.

Theorem 4.4. *For every* MSO *formula* $\varphi(X_1, \ldots, X_m)$ *and every* k*,* $(\text{OPT}_\varphi, \Delta_E)$ *is in* DynFO *for graphs with treewidth at most* k*.*

Although we, for simplicity, state and prove these theorems only for graphs, they can be generalised for arbitrary input schemas. They can also be proven for certain extensions of MSO logic. On particular extension of MSO is the logic *guarded second-order logic* (GSO), which extends MSO by *guarded* second-order quantification. This logic syntactically allows to quantify over non-unary relation variables, which, however, is semantically restricted: a tuple $\bar{t} = (a_1, \ldots, a_m)$ can only occur in a quantified relation, if all elements from $\{a_1, \ldots, a_m\}$ occur together in some tuple of the structure in which the formula is evaluated. In the context of graphs, this just means that GSO allows quantification over edge sets in addition to quantification over node sets. In this context, GSO is sometimes also called MSO_2; then, "regular" MSO is often denoted MSO_1.

In general, GSO is more expressive than MSO. For example, the property that a directed graph contains a Hamiltonian cycle can be expressed by a GSO formula that (1) existentially quantifies a set C of edges and (2) checks that the edges in C constitute a Hamiltonian cycle, by checking that (2a) every node has exactly on incoming and one outgoing edge in C, and that (2b) the graph with edge set C is connected. The condition (2a) is easily seen to be expressible, the latter condition (2b) can be expressed using the MSO formula for graph reachability, see Example 2.1. On the other hand, Hamiltonicity cannot be expressed in MSO [Courcelle and Engelfriet 2012, Proposition 5.13].

The situation is different when we only consider graphs of bounded treewidth. For every k, GSO has the same expressive power as MSO on graphs with treewidth at most k [Courcelle 1994, Theorem 2.2]. So, for every GSO formula φ and every k, there is an MSO formula ψ_k that is equivalent to φ on graphs with treewidth k. Moreover, if $\varphi = \exists X_1 \cdots \exists X_m \, \varphi'$, then ψ_k is of the form $\exists X_1^1 \cdots \exists X_1^\ell \cdots \exists X_m^1 \cdots \exists X_m^\ell \psi'$, for some natural number ℓ. So, the next corollaries follow immediately.

Corollary 4.5. *For every* GSO *sentence* φ *and every* k*,* $(\text{MC}_\varphi, \Delta_E)$ *is in* DynFO *for graphs with treewidth at most* k*.*

Corollary 4.6. *For every* GSO *formula* $\varphi(X_1, \ldots, X_m)$ *and every* k*,* $(\text{OPT}_\varphi, \Delta_E)$ *is in* DynFO *for graphs with treewidth at most* k*.*

In the remainder of this section, we work out the proofs of Theorem 4.3 and Theorem 4.4. To give an overview of the outline, we sketch the main proof ideas for Theorem 4.3.

Let φ be a fixed MSO formula of quantifier rank d and k a treewidth. We show that $(\text{MC}_\varphi, \Delta_E)$ is $(\text{AC}^1, \log n)$-maintainable for graphs with treewidth at most k. The construction of a dynamic program that maintains MC_φ is similar to the one in the proof for 3Col (Theorem 4.2). At each point, the program needs to evaluate φ on a graph $G' = (V, (E \setminus E^-) \cup E^+)$ with n nodes, where $|E^- \cup E^+| = \mathcal{O}(\log n)$, using a nice tree decomposition (T, B) of width $4k + 5$ for the initial graph $G = (V, E)$ and auxiliary information on the rank-d MSO type for each triangle of T.

The graph G' can be viewed as having a *center* $C \subseteq V$ of logarithmic size that contains the nodes with edges in $E^- \cup E^+$ and, additionally, for each of these nodes v all nodes of one bag that contains v. Furthermore, there are node sets $D_1, \dots, D_\ell$ that together contain all nodes from V, such that the sets $D_i - C$ are pairwise disjoint and disconnected in G', and each set $D_i \cap C$ has size $\mathcal{O}(1)$ (see already Figure 4.4 for an illustration). From the type information for the triangles, the dynamic program can infer the rank-d MSO type of each $G'[D_i]$. The main task now is to determine the rank-d MSO type of G' from this information.

Elberfeld, Grohe and Tantau prove a *composition theorem* for a similar setting [Elberfeld et al. 2016, Theorem 3.8]. In their setting, the center C is of constant size and there might be arbitrarily many, even infinitely many, sets D_i. They show that for every MSO formula φ one can compute a first-order formula that, when evaluated on an expansion of $G[C]$ by information on the types of the $G[D_i]$, yields the same result as φ on G.

We show for our setting that one can construct an MSO formula ψ such that $G' \models \varphi$ if and only if $\mathcal{B} \models \psi$, where $\mathcal{B}$ is a structure that extends $G'[C]$ by type information about the $G'[D_i]$. This construction is detailed next. It uses in particular a composition theorem by Shelah (cf. Theorem 4.9).

Afterwards we explain how a dynamic program makes use of this formula ψ. The main idea is that from ψ on can obtain a first-order formula that also determines whether G' satisfies φ by replacing the second-order quantification over C in ψ by first-order quantification over V. This is possible, since sets of size $\mathcal{O}(\log n)$ over C can be encoded by constantly many elements from V. Altogether, we obtain a proof for Theorem 4.3.

In the last part of this subsection we extend this proof towards optimisation problems, and prove Theorem 4.4.

A Feferman–Vaught-type composition theorem

We now present a Feferman–Vaught-type composition theorem for MSO, similar to the composition theorem by Elberfeld et al. [2016]. We prove this theorem for arbitrary input schemas, but first introduce the main concept using graphs.

We consider graphs $G = (V, E)$ with a *center* $C \subseteq V$, such that for some sets $D_1, \dots, D_\ell$ and some $w > 0$ the following conditions hold.

- $\bigcup_{i=1}^{\ell} D_i = V$.
- For all $i \neq j$, $D_i \cap D_j \subseteq C$.
- All edges in E have both end nodes in C or in some D_i.
- For every i, $|D_i \cap C| \leq w$.
- For each i there is some element $v_i \in D_i \cap C$ that is not contained in any D_j, for $j \neq i$.

In this case, we say that C has *connection width* w in G. See Figure 4.4 for an illustration.

We refer to the sets $D_1, \dots, D_\ell$ as *petals* and the nodes $v_1, \dots, v_\ell$ as *identifiers* of

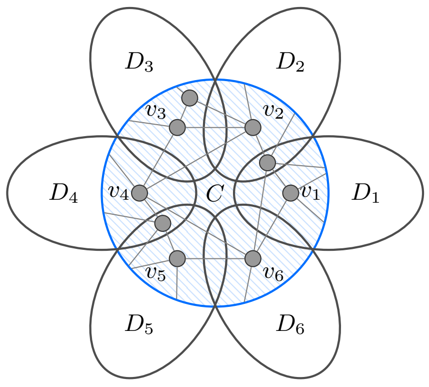


Figure 4.4: Sketch of a graph with center C of connection width 2, highlighted in blue, and petals $D_1, \dots, D_6$.

their respective petals. We emphasise that ℓ is bounded by $|C|$, but not assumed to be bounded by a constant. Readers who have read the proof of Theorem 4.2 can roughly think of C as the set of vertices from special bags.

Our goal is to show the following. If a graph G has a center C of connection width w, for come constant w, then $G[C]$ can be extended by the information about the MSO types of its petals in a suitable way, resulting in a structure $\mathcal{B}$ with universe C, such that MSO formulas over G have equivalent MSO formulas over $\mathcal{B}$.

In the following, we work out the above plan in more detail. We fix some relational schema σ and assume that it contains a unary relation symbol C.

The definition of the connection width of sets C easily carries over to σ-structures. In particular, tuples need to be entirely in C or in some petal D_i, and all constants of the structure need to be included in C. For every i, we call the set $I_i \overset{\text{def}}{=} D_i \cap C$ the *interface* of D_i and the nodes of $D_i - C$ *inner elements* of D_i.

Let $\mathcal{A}$ be a σ-structure, and C a center of connection width w with petals $D_1, \dots, D_\ell$. For every i, let $\bar{u}^i = (u^i_1, \dots, u^i_w)$ be a tuple of elements from the interface I_i of D_i such that u^i_1 is an identifier of its petal D_i and every node from I_i occurs in $\bar{u}^i$. By $(\mathcal{A}_i, \bar{u}^i)$ we denote the substructure of $\mathcal{A}$ induced by D_i with $u^i_1, \dots, u^i_w$ as constants but *without* all tuples over C, i.e., $(\mathcal{A}_i, \bar{u}^i)$ only contains tuples with at least one inner element of D_i.

Let $d > 0$. The *rank d, width w* MSO *indicator structure* of $\mathcal{A}$ relative to C and tuples $\bar{u}^i$ is the unique structure $\mathcal{B}$ which expands $\mathcal{A}[C]$ by the following relations:

- a w-ary relation J that contains all tuples $\bar{u}^i$, and
- for every rank-d MSO type τ over $\sigma \cup \{c_1, \dots, c_w\}$, a unary relation R_τ containing those identifier nodes u^i_1 for which the rank-d MSO type of $(\mathcal{A}_i, \bar{u}^i)$ is τ.

We note that different choices of the tuples $\bar{u}^i$ result in different indicator structures and we denote the set of all indicator structures of $\mathcal{A}$ relative to C by $\mathcal{S}(\mathcal{A}, C, w, d)$.

We are now ready to formulate the desired composition theorem.

Theorem 4.7. *For each $d > 0$, every* MSO *sentence φ with depth d, and each w, there is a number d' and a* MSO *sentence ψ such that for every σ-structure $\mathcal{A}$, every center C*

of $\mathcal{A}$ with connection width w and every $\mathcal{B} \in \mathcal{S}(\mathcal{A}, C, w, d')$ it holds $\mathcal{A} \models \varphi$ if and only if $\mathcal{B} \models \psi$.

The proof of Theorem 4.7 uses Shelah's *generalised sums* [Shelah 1975]. We follow the exposition from Blumensath et al. [2008]. In a nutshell, a generalised sum is a composition of several disjoint *component structures* along an *index structure*. Shelah's composition theorem states that MSO sentences on a generalised sum can be translated to MSO sentences on the index structure enriched by MSO type information on the components.

We apply the composition theorem on the basis of the following ideas, illustrated in Figure 4.5. From the center C and the petals D_i we define an index structure $\mathcal{I}$ and component structures $\mathcal{D}_i$, respectively. In the generalised sum, these disjoint structures are again composed into a structure that is very similar to $\mathcal{A}$. More precisely, $\mathcal{A}$ can be defined in the generalised sum by a first-order interpretation, and thus, thanks to Lemma 2.3, we can translate the formula φ for $\mathcal{A}$ into a formula φ' on the generalised sum. Shelah's composition theorem then provides a translation of φ' into an MSO formula ψ' on the structure $\mathcal{I}$ enriched with type information on the structures $\mathcal{D}_i$. This enriched index structure is again very similar to an MSO indicator structure $\mathcal{B}$: there is a first-order interpretation that defines the enriched index structure in $\mathcal{B}$. As a consequence, by Lemma 2.3 again, the formula ψ' can be translated into a formula φ on $\mathcal{B}$.

Before we proceed to the proof of Theorem 4.7, we formally introduce the notion of a generalised sum and state the corresponding result on translations of MSO formulas.

Definition 4.8 (Shelah [1975], formulation following Blumensath et al. [2008])**.** Let $\mathcal{I} = (I, S_1, \dots, S_r)$ be a structure and $(\mathcal{D}_i)_{i \in I}$ a sequence of structures $\mathcal{D}_i = (D_i, R^i_1, \dots, R^i_t)$ indexed by elements i of $\mathcal{I}$. The *generalised sum* of $(\mathcal{D}_i)_{i \in I}$ is the structure

$$\sum_{i \in I} \mathcal{D}_i \overset{\text{def}}{=} (U, \sim, R'_1, \dots, R'_t, S'_1, \dots, S'_r)$$

with universe $U \overset{\text{def}}{=} \{\langle i, a\rangle \mid i \in I, a \in D_i\}$ and relations

- $\langle i, a\rangle \sim \langle i', a'\rangle$ if and only if $i = i'$
- $R'_j \overset{\text{def}}{=} \{(\langle i, a_1\rangle, \dots, \langle i, a_\ell\rangle) \mid i \in I, (a_1, \dots, a_\ell) \in R^i_j\}$
- $S'_j \overset{\text{def}}{=} \{(\langle i_1, a_1\rangle, \dots, \langle i_\ell, a_\ell\rangle) \mid (i_1, \dots, i_\ell) \in S_j, a_k \in D_{i_k}$ for all $k \in \{1, \dots, \ell\}\}$

The structures $\mathcal{I}$ and $\mathcal{D}_i$ in this definition are also referred to as *index structure* and *component structures*, respectively.

Theorem 4.9 (Shelah [1975], formulation from [Blumensath et al. 2008, Theorem 3.16])**.** *From every* MSO *sentence φ, a finite sequence $\chi_1, \dots, \chi_s$ of* MSO *formulas and an* MSO *formula ψ can be constructed such that*

$$\sum_{i \in I} \mathcal{D}_i \models \varphi \;\Leftrightarrow\; (\mathcal{I}, [\![\chi_1]\!], \dots, [\![\chi_s]\!]) \models \psi$$

for all index structures $\mathcal{I}$ and component structures $\mathcal{D}_i$, where $[\![\chi]\!] \overset{\text{def}}{=} \{i \in I \mid \mathcal{D}_i \models \chi\}$.

$$\begin{tikzcd}
& \mathcal{I}, \mathcal{D}_1, \dots, \mathcal{D}_{|I|} \arrow[r, "\text{generalised sum}"] \arrow[d, dotted, no head] & \sum_{v \in I} \mathcal{D}_v \arrow[r, "\mathsf{FO}\text{ interpretation}", "\Upsilon"'] \arrow[ddd, dotted, no head, "\models" description] & \mathcal{A} \arrow[ddd, dotted, no head, "\models" description] \\
& \chi_1, \dots, \chi_s \arrow[d, dotted, no head] & & \\
\mathcal{B} \arrow[r, "\mathsf{FO}\text{ interpretation}", "\Upsilon'"'] \arrow[d, dotted, no head, "\models" description] & (\mathcal{I}, [\![\chi_1]\!], \dots, [\![\chi_s]\!]) \arrow[d, dotted, no head, "\models" description] & & \\
\psi & \psi' \arrow[l, "\text{Lemma 2.3}"'] & \varphi' \arrow[l, "\text{Theorem 4.9}"'] \arrow[uul] & \varphi \arrow[l, "\text{Lemma 2.3}"']
\end{tikzcd}$$

Figure 4.5: Overview of the proof strategy of Theorem 4.7.

Intuitively, the formulas χ_i encode the MSO type information on the component structures of the generalised sum.

With the necessary notions in place, we can now prove Theorem 4.7.

Proof (of Theorem 4.7). Suppose that $\mathcal{A}$ is a σ-structure with center C of connection width w, tuples $\bar{u}^i$ collecting the interface nodes for each petal D_i as described above, and let φ be an MSO formula over schema σ. The proof is in three steps, depicted in Figure 4.5:

(A) We present an index structure $\mathcal{I}$ and component structures $\mathcal{D}_i$, and show that there is an FO-interpretation Υ that interprets the structure $\mathcal{A}$ in the generalised sum $\sum_{i \in I} \mathcal{D}_i$. Thus there is an MSO formula φ' such that $\mathcal{A} \models \varphi$ if and only if $\sum_{i \in I} \mathcal{D}_i \models \varphi'$ by Lemma 2.3.

(B) From Theorem 4.9 we obtain formulas ψ' and $\chi_1, \dots, \chi_s$ such that $\sum_{i \in I} \mathcal{D}_i \models \varphi'$ if and only if $(\mathcal{I}, [\![\chi_1]\!], \dots, [\![\chi_s]\!]) \models \psi'$.

(C) We show that there is an FO-interpretation Υ' that interprets $(\mathcal{I}, [\![\chi_1]\!], \dots, [\![\chi_s]\!])$ in each MSO indicator structure $\mathcal{B} \in \mathcal{S}(\mathcal{A}, C, w, d')$ with appropriate d'. Thus there is an MSO formula ψ that satisfies $\mathcal{B} \models \psi$ if and only if $(\mathcal{I}, [\![\chi_1]\!], \dots, [\![\chi_s]\!]) \models \psi'$ by Lemma 2.3.

Combining these three steps allows us to conclude

$$\mathcal{A} \models \varphi \quad \overset{(A)}{\Longleftrightarrow} \quad \sum_{i \in I} \mathcal{D}_i \models \varphi' \quad \overset{(B)}{\Longleftrightarrow} \quad (\mathcal{I}, [\![\chi_1]\!], \dots, [\![\chi_s]\!]) \models \psi' \quad \overset{(C)}{\Longleftrightarrow} \quad \mathcal{B} \models \psi$$

for any MSO indicator structure $\mathcal{B} \in \mathcal{S}(\mathcal{A}, C, w, d')$.

It remains to prove (A) and (C), since (B) is a direct application of Theorem 4.9.

Towards proving (A) we construct structures $\mathcal{I}$ and $\mathcal{D}_i$ which are closely related to the substructure $\mathcal{A}[C]$ of $\mathcal{A}$ and the structures $(\mathcal{A}_i, \bar{u}^i)$, respectively. Let $R_1, \dots, R_q$ be the relation symbols of σ and let $\sigma' = \{S_1, \dots, S_q\}$. The structure $\mathcal{I}$ has universe C, for each $j \in \{1, \dots, q\}$ the σ'-relation S_j, defined as the restriction $R_j[C]$ of the respective σ-relation R_j of $\mathcal{A}$ to C, and the relation J as described above. Each structure $\mathcal{D}_v$ for $v \in C$ is over schema $\sigma \cup \{U_1 \dots, U_w\}$ and defined as follows. If v is an identifier u_1^i

of a petal, then $\mathcal{D}_v = (\mathcal{A}_i, \{u_1^i\}, \dots, \{u_w^i\})$, that is, the restriction of $\mathcal{A}$ to the elements from D_i, without any tuples consisting only of elements from $C \cap D_i$, and with additional unary, singleton relations $U_1, \dots, U_w$ such that U_j includes only the j-th interface node u_j^i. If v is no identifier node then $\mathcal{D}_v$ is the structure with universe $\{v\}$ and empty relations.

In the generalised sum $\mathcal{S} \overset{\text{def}}{=} \sum_{v \in I} \mathcal{D}_v = (U, \sim, R'_1, \dots, R'_q, U'_1, \dots U'_w, S'_1, \dots, S'_q, J')$, the universe U consists of elements of the form $\langle u_1^i, w\rangle$, where $w \in D_i$, and of the form $\langle u, u\rangle$ where $u \in C$ is not an identifier of any petal. We emphasise that the formulas of the interpretation that defines (a copy of) $\mathcal{A}$ in $\mathcal{S}$ can *not* access the components u and v of an element $\langle u, v\rangle \in U$. However, U can be partitioned into four kinds of elements, each of which can easily be distinguished from the others in a first-order fashion:

(i) Elements of the form $\langle u, u\rangle$, for which $u \in C$ is not an identifier of any petal. They can be identified since they form an equivalence class of size 1 with respect to $\sim$;
(ii) elements of the form $\langle u_1^i, u_1^i\rangle$, for an identifier $u_1^i \in C$. These are precisely the elements in U'_1;
(iii) elements of the form $\langle u_1^i, w\rangle$, where $w \in C$, $w \neq u_1^i$ and w is in the petal D_i. These elements occur in some U'_j, for $j > 1$ (but not in U'_1);
(iv) elements of the form $\langle u_1^i, w\rangle$, where $w \notin C$ is in the petal D_i. They do not occur in any U'_j and are not of type (i).

Elements of type (iv) are in one-to-one correspondence with the inner elements of petals D_i. Elements from C might have several copies in U, but only one of the types (i) or (ii). Thus, the formula that defines the universe for the first-order interpretation Υ of $\mathcal{A}$ in $\sum_{i \in I} \mathcal{D}_i$ simply drops all elements of type (iii). Tuples of $\mathcal{A}$ of a relation R_j that entirely consist of nodes from C (that is, elements of type (i) or (ii) in $\mathcal{S}$) are directly induced by the corresponding relation S'_j.

Tuples of a relation R_j in $\mathcal{A}$ that contain at least one inner node, that is, at least one node of type (iv), are defined on the basis of the relation R'_j of the generalised sum. Note however that these relations R'_j in $\mathcal{S}$ contain elements of type (iii) which are not in the universe of the interpreted structure. In a corresponding tuple of the relation R_j, these elements are substituted by their copy of type (i).

We make this more precise. Observe that it can be expressed in a first-order fashion whether for a type (iii) element $\langle v_1, u_1\rangle$ and a type (i) element $\langle v_2, v_2\rangle$ it holds $u_1 = v_2$, i.e., that, intuitively, $\langle v_2, v_2\rangle$ is the copy representing u_1 in $\mathcal{S}$ that survives in the universe of the interpretation. We claim that this condition holds, if and only if $\langle v_2, v_2\rangle$ occurs as the i-th entry in some tuple of J' with first entry $\langle v_1, u_1\rangle$, where i is the unique number such that $\langle v_1, u_1\rangle \in U'_i$. From this claim, first-order expressibility follows instantly. The "only if"-part of the claim is straightforward. For the "if"-part, it follows from the latter condition that there is a tuple with first entry v_1 and i-th entry v_2 in J, by the definition of J'. Since $\langle v_1, u_1\rangle \in U'_i$, there is also a tuple in J with v_1 as first entry and u_1 as i-th entry. However, since J has at most one tuple with any given value as first entry, $u_1 = v_2$ follows, as claimed.

A tuple with some element $\langle v, u\rangle$ of type (iv) is now in a relation R_j of the interpretation, if it can be transformed into a tuple of R'_j by replacing some elements $\langle w, w\rangle$ of type (i) with $\langle v, w\rangle$.

It follows from the construction that $\Upsilon(\mathcal{S})$ is isomorphic to $\mathcal{A}$ and therefore $\mathcal{A} \models \varphi$ if and only if $\Upsilon(\mathcal{S}) \models \varphi$. We obtain the formulas φ', $\chi_1, \dots, \chi_s$ and ψ' as explained above. Let d' be the maximal quantifier rank of any formula χ_j. This concludes step (A) of the proof.

Towards proving (C), recall that we need to show that there is a first-order interpretation Υ' which interprets $(\mathcal{I}, [\![\chi_1]\!], \dots, [\![\chi_s]\!])$ in $\mathcal{B}$, for any $\mathcal{B} \in \mathcal{S}(\mathcal{A}, C, w, d')$. Let $\mathcal{B}$ be such a structure. The universe of $\mathcal{I}$ is C, that is, the same as the universe of $\mathcal{B}$. Thus the formula of Υ' that defines the universe is trivial.

For the definitions of the relations $[\![\chi_j]\!]$, the idea is as follows. If v is *not* an identifier u_1^i of a petal, then $v \in [\![\chi_j]\!]$ if and only if χ_j holds in the structure consisting of only one element and with empty relations, which can be hard-coded in the defining formula. Otherwise, if $v = u_1^i$ for some i, we need to determine whether χ_j holds in the structure $\mathcal{D}_v = (\mathcal{A}_i, \{u_1^i\}, \dots, \{u_w^i\})$, which is a structure over schema $\sigma \cup \{U_1, \dots, U_w\}$. The structure $\mathcal{B}$ contains information about the MSO types of the structures $(\mathcal{A}_i, \bar{u}^i)$, but $(\mathcal{A}_i, \bar{u}^i)$ is a structure over schema $\sigma \cup \{c_1, \dots, c_w\}$. Yet it is easy to see that for the formula χ_j' that is obtained from χ_j by replacing every atom $U_k(x)$ by $x = c_k$ it holds $(\mathcal{A}_i, \bar{u}^i) \models \chi_j'$ if and only if $\mathcal{D}_v \models \chi_j$. So, in this case $v \in [\![\chi_j]\!]$ if and only if $v \in R_\tau^{\mathcal{B}}$ for a rank-d' MSO type τ with $\chi_j' \in \tau$. All these conditions can even be expressed by quantifier-free first-order formulas for fixed formulas χ_j. As a result, by Lemma 2.3 we obtain from Υ' and ψ' a formula ψ with $\mathcal{B} \models \psi \Leftrightarrow \mathcal{A} \models \varphi$. □

Maintaining MSO-definable decision problems

We proceed to show that every MSO-definable decision problem can be maintained in DynFO, and thus prove Theorem 4.3. As we want to apply Corollary 3.4, we need to assure that these queries are almost domain independent. This is easy to see and given by the following proposition.

Proposition 4.10 ([Datta et al. 2019, Proposition 3.2]). *Every* MSO*-definable query is almost domain independent.*

For the proof of Theorem 4.3 it consequently suffices to show that $(\text{MC}_\varphi, \Delta_E)$ is $(\mathsf{AC}^1, \log n)$-maintainable for graphs G with treewidth at most k.

The idea for our dynamic program is similar to the idea for maintaining 3Col in Theorem 4.2: during its initialisation the program constructs a tree decomposition and computes the MSO type of appropriate rank for all triangles (instead of partial colourings as in the proof of Theorem 4.2). During the change sequence, a set C of nodes is defined that contains, for each graph node v affected by a change, all nodes of at least one *special* bag containing v. The set C has connection width w for some constant w and the dynamic program basically maintains an MSO indicator structure $\mathcal{B}$ for G relative to C. As there are only $\log n$ many change steps, the size of C is bounded by $\mathcal{O}(\log n)$.

By Theorem 4.7 there is an MSO formula ψ with the property that $G \models \varphi$ if and only if $\mathcal{B} \models \psi$. Although the dynamic program maintains $\mathcal{B}$, it cannot directly evaluate ψ, as it is restricted to use first-order formulas. For this reason we first show that second-order quantification over sets of size $\mathcal{O}(\log n)$ can be simulated in first-order logic, if a particular relation is present. Afterwards we present the details of the dynamic program.

We call an MSO formula *C-restricted*, if all its quantified subformulas are of one of the following forms.

- $\exists x\ (C(x) \wedge \varphi)$ or $\forall x\ (C(x) \to \varphi)$,
- $\exists X\ (\forall x(X(x) \to C(x)) \wedge \varphi)$ or $\forall X\ (\forall x(X(x) \to C(x)) \to \varphi)$.

Let $\mathcal{A}$ be a structure with a unary relation C and a $(k+1)$-ary relation Sub, for some k. We say that Sub *encodes subsets of C* if, for each subset $C' \subseteq C$, there is a k-tuple $\bar{t}$ such that, for every element $c \in C$ it holds $c \in C'$ if and only if $(\bar{t}, c) \in \text{Sub}$. Clearly, such an encoding of subsets only exists if $|V|^k \geq 2^{|C|}$ and thus if $|C| \leq k \log |V|$.

Proposition 4.11. *For each C-restricted* MSO *sentence ψ over a schema σ (containing C) and every k there is a first-order sentence χ over $\sigma \cup \{S\}$ where S is a $(k+1)$-ary relation symbol such that, for every σ-structure $\mathcal{A}$ and $(k+1)$-ary relation Sub that encodes subsets of C (of $\mathcal{A}$), it holds $\mathcal{A} \models \psi$ if and only if $(\mathcal{A}, Sub) \models \chi$.*

Proof. The proof is straightforward. Formulas $\exists X\ (\forall x(X(x) \to C(x)) \wedge \varphi)$ are translated into formulas $\exists \bar{x}\ \varphi'$, where $\bar{x}$ is a tuple of k variables and φ' results from φ by simply replacing every atomic formula $X(y)$ by $\text{Sub}(\bar{x}, y)$. Universal set quantification is translated analogously. □

We now turn to the proof of Theorem 4.3.

Proof (of Theorem 4.3). Thanks to Corollary 3.4 and Proposition 4.10 it suffices to show that $(\text{MC}_\varphi, \Delta_E)$ is $(\mathsf{AC}^1, \log n)$-maintainable in DynFO for graphs with treewidth at most k. Let d be the quantifier rank of φ and let d' and ψ be the number and the MSO sentence guaranteed to exist by Theorem 4.7.

Given a graph $G = (V, E)$, the AC^1 initialisation first ensures that relations $\leq, +, \times$ and BIT are available. Then it computes a d_{tree}-nice tree decomposition (T, B) with $T = (I, F, r)$ with bags of size at most $\ell \overset{\text{def}}{=} 4k + 6$, for the constant d_{tree} guaranteed to exist by Lemma 4.1, together with the predecessor order $\preceq$ on I.

With each node $i \in I$, we associate a tuple $\bar{v}(i) = (v_1, \ldots, v_m, v_1, \ldots, v_1)$ of length ℓ, where $B(i) = \{v_1, \ldots, v_m\}$ and $v_1 < \cdots < v_m$. That is, if the bag size of i is ℓ, this tuple just contains all graph nodes of the bag in increasing order. If the bag size is smaller, the smallest graph node is repeated. Similarly, with each triangle $\delta = (i_0, i_1, i_2)$ such that the subgraph $G(\delta)$ has at least one inner node, we associate a tuple $\bar{v}(\delta) = (v(\delta), \bar{v}(i_0), \bar{v}(i_1), \bar{v}(i_2))$, where $v(\delta)$ denotes the smallest inner node of $G(\delta)$ with respect to $\leq$.

The dynamic program further uses auxiliary relations S, C, N, and D_τ, for each rank-d' MSO type τ over the schema that consists of the binary relation symbol E and $3\ell + 1$ constant symbols $c_1, \ldots, c_{3\ell+1}$. The intended meaning is that C is a center of G with connection width $3\ell + 1$ and that from these relations an MSO indicator structure $\mathcal{B}$ relative to C can be defined in first-order.

The relation S stores tuples $\bar{v}(i)$ representing special bags, as in the proof of Theorem 4.2. The relations D_τ provide MSO type information for all triangles. More precisely, for

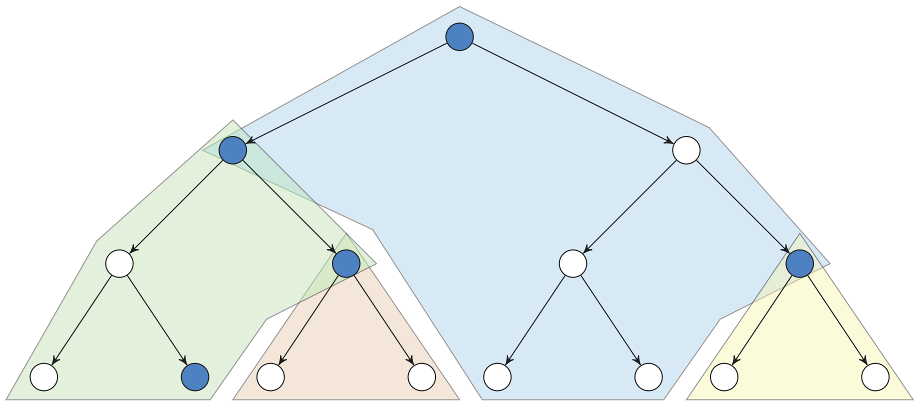


Figure 4.6: Tree of a tree decomposition. Tree nodes representing special bags are coloured blue, the corresponding maximal clean triangles are indicated as coloured areas. The union of all special bags are a center of the graph, with the graphs induced by the maximal clean triangles as petals.

each triangle $\delta = (i_0, i_1, i_2)$ for which the subgraph $G(\delta)$ has at least one inner node, D_τ contains the tuple $\bar{v}(\delta)$ if and only if the rank-d' MSO type of $(G(\delta), \bar{v}(\delta))$ is τ.

The set C always contains all graph nodes that occur in special bags (and thus in S). Additionally, for each maximal clean triangle δ with at least one inner node, C contains the inner node $v(\delta)$. The relation N defines a bijection between C and an initial segment of $\leq$.

We observe that C is a center of G and that the petals induced by C correspond to the maximal clean triangles, with respect to the special nodes stored in S, with at least two inner nodes. The interface $I(\delta)$ of a petal corresponding to a maximal clean triangle $\delta = (i_0, i_1, i_2)$ contains the nodes from $B(i_0), B(i_1)$, and $B(i_2)$ as well as the node $v(\delta)$, so C has connection width $w \overset{\text{def}}{=} 3\ell + 1$. Figure 4.6 gives an illustration.

Now, an indicator structure $\mathcal{B} \in \mathcal{S}(G, C, w, d')$ can be first-order defined as follows. Clearly, maximal clean triangles can be easily first-order defined from the relation S. For each maximal clean triangle $\delta = (i_0, i_1, i_2)$ with at least two inner nodes, the relation J contains the tuple $\bar{v}(\delta)$, and the relation R_τ contains $v(\delta)$ if and only if is $\bar{v}(\delta) \in D_\tau$. We translate the MSO formula ψ to a C-restricted MSO formula χ' such that $\mathcal{B} \models \psi \Leftrightarrow (G, Aux) \models \chi'$, where Aux is the auxiliary database stored by the dynamic program. This translation is basically as described by Lemma 2.3. The formula χ' results from ψ by

- C-restricting every quantified subformula, so, for example, replacing every quantified subformula $\exists X\ \theta$ by $\exists X\ (\forall x(X(x) \to C(x)) \wedge \theta)$ and every quantified subformula $\forall X\ \theta$ by $\forall X\ (\forall x(X(x) \to C(x)) \to \theta)$, and
- replacing every atom $A(\bar{x})$ by $\theta_A(\bar{x})$, where θ_A is the first-order formula that defines A in (G, Aux).

It clearly holds that $(G, Aux) \models \chi' \Leftrightarrow \mathcal{B} \models \psi$, and by Theorem 4.7 also $(G, Aux) \models \chi' \Leftrightarrow G \models \varphi$.

We now define a relation Sub that encodes subsets of C. We observe that C is of size at most $b \log n$ for some $b \in \mathbb{N}$. Thus a subset C' of C can be represented by a tuple $(a_1, \dots, a_b)$ of nodes, where an element $c \in C$ is in C' if and only if c is the m-th element of C with respect to the mapping defined by N, $m = (\ell - 1)\log n + j$ and the j-th bit of a_ℓ is one. By Proposition 4.11 we finally obtain a first-order formula χ such that $(G, Aux, \mathrm{Sub}) \models \chi \Leftrightarrow (G, Aux) \models \chi' \Leftrightarrow G \models \varphi$. That means that a dynamic program that maintains the auxiliary relations as intended can maintain the dynamic query $(\mathrm{MC}_\varphi, \Delta_E)$.

It thus remains to describe how the auxiliary relations can be initialised and updated. The set C is initially the bag $B(r)$ of the root of T plus one inner node $v((r, r, r))$, and S contains the tuple $\bar{v}(r)$.

The relations D_τ are computed in $d_{\text{tree}} \log n$ inductive steps, each of which can be defined in first-order logic, and therefore this computation can be carried out in AC^1, thanks to $\mathrm{IND}[\log n] = \mathsf{AC}^1$. More precisely, the computation of the relations D_τ proceeds inductively in a bottom-up fashion. It starts with triangles $\delta = (i_0, i_1, i_2)$ for which $T(\delta)$ has exactly one or two inner tree nodes (i.e., nodes different from i_0, i_1, i_2). Since such graphs $G(\delta)$ have at most 5ℓ nodes, their type can be determined by a first-order formula. Basically, all isomorphism types of such graphs and their respective MSO types can be directly encoded into first-order formulas.

For larger triangles, several cases need to be distinguished. Here we explain the case of a triangle $\delta = (i_0, i_1, i_2)$, for which i_0 has child nodes i_1' and i_2' such that $i_1' \preceq i_1$ and $i_2' \preceq i_2$ (cf., Figure 4.3). In this case, the type τ of $(G(\delta), \bar{v}(\delta))$ can be determined from the types τ_1 of $(G(\delta_1), \bar{v}(\delta_1))$ and τ_2 of $(G(\delta_2), \bar{v}(\delta_2))$, where $\delta_1 = (i_1', i_1, i_1)$ and $\delta_2 = (i_2', i_2, i_2)$, and the type τ_0 of the graph G_0 that includes all edges of $G(\delta)$ that are not already in $G(\delta_1)$ or $G(\delta_2)$. More precisely, τ_0 is the type of $(G_0, \bar{v}(i_0), \bar{v}(i_1), \bar{v}(i_2), \bar{v}(i_1'), \bar{v}(i_2'))$ and G_0 is the subgraph of G with node set $V_0 = \bigcup\{B(i_0), B(i_1), B(i_2), B(i_1'), B(i_2')\}$ and all edges from $G[V_0]$ that have at least one endpoint in $B(i_1') \cup B(i_2')$. These types are either already computed or the graphs are of size at most 5ℓ and their type can therefore be determined by a first-order formula as before.

We make this more precise. We observe that $(G(\delta), \bar{v}(\delta))$ can be composed from the graphs $(G_0, \bar{v}(i_0), \bar{v}(i_1), \bar{v}(i_2), \bar{v}(i_1'), \bar{v}(i_2'))$, $(G(\delta_1), \bar{v}(\delta_1))$ and $(G(\delta_2), \bar{v}(\delta_2))$ by first taking the disjoint union of these graphs and afterwards fusing nodes according to the identities induced by the additional constants. For both operations, the rank-d MSO type of the resulting structure only depends on the rank-d MSO type(s) of the original structure(s), see Theorem 2.2 and [Makowsky 2004, Proposition 3.6]. The type τ of $(G(\delta), \bar{v}(\delta))$ is therefore determined by a finite function f as $\tau = f(\tau_0, \tau_1, \tau_2)$, which can be directly encoded into first-order formulas.

Finally, we describe how a dynamic program can maintain S, C, and N for $\log n$ many changes. The relation D_τ is not adapted during the changes. Whenever an edge (u, v) is inserted to or deleted from G, the nodes u and v are viewed as *affected*. For every affected node u that is not yet in a bag stored in S, a *special* tree node is selected (in some canonical way, e.g. always the smallest node with respect to $\leq$ is selected) such that $u \in B(j)$. Furthermore, if node i is the least common ancestor of j and another special node it becomes special, as well. It is easy to see that when selecting j as a special node, at most one further node becomes special. The tuples $\bar{v}(j)$ and $\bar{v}(i)$ (if i exists) are added to S, their elements are added to C, the identifier nodes in C for maximal clean triangles are corrected, and N is updated accordingly. □

Extension to optimisation problems

We build on the techniques introduced for the proof of Theorem 4.3 to show that also MSO-definable optimisation problems can be maintained for graphs of bounded treewidth, that is, to show Theorem 4.4. Again, it suffices to show $(\mathsf{AC}^1, \log n)$-maintainability of such a problem under single-edge changes. As before, a dynamic program that maintains a dynamic optimisation problem $(\mathrm{OPT}_\varphi, \Delta_E)$ defines a center C of the graph that includes one special bag for each node affected by the $\log n$ changes. For each petal D_i associated with the center, the dynamic program not only stores whether the subgraph induced by the petal has some MSO type τ, but it maintains for each such type τ a collection $(B_1, \ldots, B_m)$ of subsets of $D_i \setminus C$ such that (1) the subgraph induced by the petal and expanded by the relations $B_1, \ldots, B_m$ has type τ, and (2) this collection is minimal with this property with respect to the sum $\sum_{i=1}^{m} |B_i|$.

A first-order formula can then compute, given a target MSO type for each petal that together imply that the expanded graph satisfies φ, the minimum achievable overall sum of the sizes $|B_i|$, and give the corresponding answer for the choice of MSO types that gives the optimal result.

Proof (of Theorem 4.4). We only prove the special case of $m = 1$, the extension to the general case is straightforward. Let $\varphi(X)$ be an MSO formula of quantifier rank d. The proof of Theorem 4.3 shows how one can obtain a dynamic program that $(\mathsf{AC}^1, \log n)$-maintains under single-edge changes the model checking problem MC_ψ for $\psi \overset{\text{def}}{=} \exists X\, \varphi$. We adapt this proof, and reuse its notation, in order to obtain such a dynamic program for $(\mathrm{OPT}_\varphi, \Delta_E)$, using almost the same auxiliary relations. Together with Corollary 3.4 and Proposition 4.10, the result follows.

In the following, we sketch the proof idea. We consider φ to be an MSO sentence over the signature $\{E, X\}$. Let $G^{+X} = (V, E, X)$ be an arbitrary expansion of a graph G with a center C of connection width w, for some constant w. By Theorem 4.7 there is a number d' and an MSO sentence ψ such that for every $\mathcal{B}^{+X} \in \mathcal{S}(G^{+X}, C, w, d')$ it holds that $G^{+X} \models \varphi$ if and only if $\mathcal{B}^{+X} \models \psi$. So, the formula ψ uses the type information on the petals provided by $\mathcal{B}^{+X}$ as well as $G^{+X}[C]$ directly to check whether the relation X represents a *feasible* solution of the problem OPT_φ. In the proof of Theorem 4.3 we explained how to obtain a first-order formula χ from ψ such that $(G, \mathit{Aux}, \mathrm{Sub}) \models \chi \Leftrightarrow \mathcal{B} \models \psi$, for the auxiliary database Aux maintained by the dynamic program constructed in the proof of Theorem 4.3 and a relation Sub encoding subsets of C.

Let $\mathcal{B} \in \mathcal{S}(G, C, w, d'+1)$ be an MSO indicator structure for G. Our goal is to maintain some relations that augment the type information provided by $\mathcal{B}$ such that a formula χ' similar to χ can "guess" a relation X, check that it is a feasible solution, compute its size and verify that no feasible solution of smaller size exists. Of course, a relation X of unrestricted size cannot be quantified in first-order logic, even in the presence of Sub, but we will see that the restriction of X to C and the type information on the petals can be quantified, which is sufficient for our purpose.

We now give the details of the construction. The structure $\mathcal{B}$ contains relations R_τ such that $u_1^i \in R_\tau$ if and only the rank-$(d'+1)$ MSO type of $(\mathcal{A}_i, \bar{u}^i)$ over schema $\sigma = \{E, c_1, \ldots, c_{3\ell+1}\}$ is τ, where the subgraph $\mathcal{A}_i$ over universe D_i and the tuple $\bar{u}^i$ are defined as before in this subsection. We say that a rank-d' type τ' over schema $\sigma^{+X} \overset{\text{def}}{=} \sigma \cup \{X\}$ *can be realised* in $(\mathcal{A}_i, \bar{u}^i)$ by a set $A_i \subseteq D_i$, if τ' is the rank-d' MSO type of $(\mathcal{A}_i, \bar{u}^i, A_i)$. If $(\mathcal{A}_i, \bar{u}^i)$ has rank-$(d'+1)$ MSO type τ, the existence of such a set is equivalent to the statement $\exists X\, \alpha_{\tau'} \in \tau$. We note that τ' already determines whether $u_j^i \in A_i$ shall hold, for each constant u_j^i from the tuple $\bar{u}^i$.

The dynamic program maintains relations $\#R_{\tau'}$ and $Q_{\tau'}$, for each rank-d' MSO type over σ^{+X}. The relations $\#R_{\tau'}$ give the minimal size of a set that realises the type τ'. So, if τ' can be realised in $(\mathcal{A}_i, \bar{u}^i)$ by some set A, and s is the minimal size of such a set, then $\#R_{\tau'}$ shall contain the tuple (u_1^i, v_s), where v_s is the $(s+1)$-th element[4] with respect to $\leq$. Furthermore, for the lexicographically minimal set A of this kind and size s, $Q_{\tau'}$ shall contain all tuples (u_1^i, a), where $a \in A$.

We construct a first-order formula χ' from χ that is able to define an *optimal* solution X for OPT_φ from (G, Aux, Sub) expanded by the relations $\#R_{\tau'}$ and $Q_{\tau'}$. First, this formula quantifies for each rank-d' MSO type τ' the set of identifiers u_1^i such that X realises τ' in $(\mathcal{A}_i, \bar{u}^i)$ and checks consistency: as for each node $v \in C$ that appears in $\bar{u}^i$ the type τ' already determines whether $v \in X$ shall hold, the respective types need to agree for nodes that appear in multiple petals. For each u_1^i the assigned type also needs to be realisable in the respective substructure $(\mathcal{A}_i, \bar{u}^i)$, which can be checked using the relations R_τ of $\mathcal{B}$. Using this information, χ' can apply χ to check that the implied set X is a feasible solution. With the help of $\#R_\tau$ it can compute the size of X, as $\mathsf{FO}(\leq, +, \times)$ is able to add up logarithmically many numbers [Vollmer 1999, Theorem 1.21] and C is only of logarithmic size in $|V|$. Also χ' checks that no other assignment of types τ' to identifier nodes results in feasible solutions of smaller size. Finally, χ' uses the relations $Q_{\tau'}$ to actually return an optimal solution X.

Building on the proof of Theorem 4.3, it remains to show that the additional auxiliary relations $\#R_{\tau'}$ and $Q_{\tau'}$ can be initialised and maintained. Actually, we maintain similarly defined relations $\#D_{\tau'}$ and $F_{\tau'}$, the relations $\#R_{\tau'}$ and $Q_{\tau'}$ are then first-order definable by the dynamic program using these relations.

Let δ be a triangle such that $G(\delta)$ has at least one inner node. Similar to the relations D_τ used in the proof of Theorem 4.3, here a relation $\#D_{\tau'}$ contains the tuple $(\bar{v}(\delta), u)$ if and only if (1) the rank-d' MSO type τ' is realisable in $(G(\delta), \bar{v}(\delta))$, and (2) u is the $(s+1)$-th element with respect to $\leq$, where s is the minimal size of a set that realises this type. Furthermore, for the lexicographically minimal set A of this kind and size s, $F_{\tau'}$ contains all tuples $(\bar{v}(\delta), a)$, where $a \in A$. It is clear that these relations suffice to define the relations $\#R_{\tau'}$ and $Q_{\tau'}$, given the other relations of the proof of Theorem 4.3.

[4]We ignore the case that the size could be as large as $|V|$, which can be handled by some additional encoding.

The proof of Theorem 4.3 can be extended to show that the initial versions of these auxiliary relations can be computed in AC^1. For the inductive step of this computation, a type τ' realisable in a structure $(G(\delta), \bar{v}(\delta))$ might be achievable by a finite number of combinations of types of its substructures. Here, the overall size of the realising set for X needs to be computed and the minimal solution needs to be picked. This is possible by an $\mathsf{FO}(\leq, +, \times)$-formula since the number of possible combinations is bounded by a constant depending only on d and k.

The updates of the auxiliary relations are exactly as in the proof of Theorem 4.3. Since D_τ needs no updates there, neither $\#D_{\tau'}$ nor $F_{\tau'}$ do, here. $\square$

From the proof it is easy to see that a dynamic program can also maintain the *size s* of an optimal solution, either implicitly as $\sum_{j=1}^{m} |Q_j|$ for distinguished relations Q_j, or as $\{v_s\}$. Additionally, it can easily be adapted for optimisation problems on *weighted* graphs, where nodes and edges have polynomial weights in n.

4.2 An arity hierarchy for DynProp

Dynamic programs with quantifier-free first-order update formulas are surprisingly expressive, as we have already noted in the introduction of this chapter: the query Parity is already in DynProp under changes of single bits, and many first-order expressible queries can be maintained dynamically without the need of quantification. We start this section with an example of such a query, partly because we want to construct a non-trivial quantifier-free dynamic program explicitly and by this improve our intuition on the mechanics of DynProp, and partly because we will need this particular result later on.

Example 4.12. For fixed $k \in \mathbb{N}$ let in-deg-k be the unary query that, given a graph G, returns the set of nodes with in-degree k. This query is easily definable in FO for each k. We show here that in-deg-k can be maintained by a DynProp-program $\mathcal{P}$ under single-edge changes of G.

The dynamic program we construct uses *k-lists*, a slight extension of the list technique introduced in [Gelade et al. 2012]. The list technique was used in [Zeume and Schwentick 2015] to maintain emptiness of a unary relation U under insertions and deletions of single elements with quantifier-free formulas. To this end a binary relation List which encodes a linked list of the elements in U in the order of their insertion is maintained. Additionally, two unary relations mark the first and the last element of the list. The key insight is that a quantifier-free formula can figure out whether the relation U becomes empty when an element a is deleted by checking whether a is both the first *and* the last element of the list.

To maintain in-deg-k, the quantifier-free dynamic program $\mathcal{P}$ stores, for every node $v \in V$, a list of all nodes u with $(u, v) \in E$, using a ternary relation $\textsc{List}_1$. More precisely, if $u_1, \dots, u_m$ are the in-neighbours of v then $\textsc{List}_1$ contains the tuples $(v, u_{i_j}, u_{i_{j+1}})$ where $j_1, \dots, j_m$ is some permutation of $\{1, \dots, m\}$. Additionally, the program uses ternary relations $\textsc{List}_2, \dots, \textsc{List}_k$ such that $\textsc{List}_i$ describes paths of length i in the linked list $\textsc{List}_1$. For example, if $(v, u_1, u_2), (v, u_2, u_3)$ and (v, u_3, u_4) are tuples in $\textsc{List}_1$, then $(v, u_1, u_4) \in \textsc{List}_3$. The list $\textsc{List}_1$ comes with $2k$ binary relations $\textsc{First}_1, \dots, \textsc{First}_k, \textsc{Last}_1, \dots, \textsc{Last}_k$ that mark, for each $v \in V$, the first and the last k elements of the list of in-neighbours of v, as well as with $k+2$ unary relations $\textsc{Is}_0, \dots, \textsc{Is}_k, \textsc{Is}_{>k}$ that count the number of in-neighbours for each $v \in V$ up to k. We call nodes u with $(v, u) \in \textsc{First}_i$ or $(v, u) \in \textsc{Last}_i$ the i-first or the i-last element for v, respectively.

Using these relations, the query can be answered easily: the result is the set of nodes v with $v \in \textsc{Is}_k$. We show how to maintain the auxiliary relations under insertions and deletions of single edges, and assume for ease of presentation of the update formulas that if a change $\textsc{ins}_E(u, v)$ occurs then $(u, v) \notin E$ before the change, and a change $\textsc{del}_E(u, v)$ only happens if $(u, v) \in E$ before the change.

Insertions of edges When an edge (u, v) is inserted, then the node u needs to be inserted into the list of v. This node u also becomes the last element of the list (encoded by a tuple $(v, u) \in \textsc{Last}_1$), and the i-last node u' for v becomes the $(i+1)$-last one, for each $i < k$. If only i elements are in the list for v before the change, for some $i < k$, then u becomes the $(i+1)$-first element for v. The number of elements in the list for v is updated appropriately. The update formulas are as follows:

$$\begin{aligned}
\varphi^{\textsc{List}_i}_{\textsc{ins}_E}(u, v; x, y, z) &\overset{\text{def}}{=} \textsc{List}_i(x, y, z) \lor (v = x \land \textsc{Last}_i(x, y) \land u = z) && \text{for } i \in \{1, \dots, k\} \\
\varphi^{\textsc{Last}_1}_{\textsc{ins}_E}(u, v; x, y) &\overset{\text{def}}{=} (v \neq x \land \textsc{Last}_1(x, y)) \lor (v = x \land u = y) \\
\varphi^{\textsc{Last}_i}_{\textsc{ins}_E}(u, v; x, y) &\overset{\text{def}}{=} (v \neq x \land \textsc{Last}_i(x, y)) \lor (v = x \land \textsc{Last}_{i-1}(y)) && \text{for } i \in \{2, \dots, k\} \\
\varphi^{\textsc{First}_i}_{\textsc{ins}_E}(u, v; x, y) &\overset{\text{def}}{=} \textsc{First}_i(x, y) \lor (v = x \land u = y \land \textsc{Is}_{i-1}(x)) && \text{for } i \in \{1, \dots, k\} \\
\varphi^{\textsc{Is}_0}_{\textsc{ins}_E}(u, v; x) &\overset{\text{def}}{=} (v \neq x \land \textsc{Is}_0(x)) \\
\varphi^{\textsc{Is}_i}_{\textsc{ins}_E}(u, v; x) &\overset{\text{def}}{=} (v \neq x \land \textsc{Is}_i(x)) \lor (v = x \land \textsc{Is}_{i-1}(x)) && \text{for } i \in \{1, \dots, k\} \\
\varphi^{\textsc{Is}_{>k}}_{\textsc{ins}_E}(u, v; x) &\overset{\text{def}}{=} \textsc{Is}_{>k}(x) \lor (v = x \land \textsc{Is}_k(x))
\end{aligned}$$

Deletions of edges When an edge (u, v) is deleted, the hardest task for quantifier-free update formulas is to determine whether, if the in-degree of v was *at least* $k+1$ before the change, the in-degree of v is now *exactly* k. To decide whether this is the case we use that if an element u is the j-first and at the same time the j'-last element for v, then the list for v contains exactly $j + j' - 1$ elements. If u is removed from the list, $j + j' - 2$ elements remain. So, using the relations $\textsc{First}_j$ and $\textsc{Last}_{j'}$, the exact number m of elements after the change can be determined, if $m \leq 2k - 2$.

The relations $\textsc{First}_i$ (and, symmetrically, the relations $\textsc{Last}_i$) can be maintained using the relations $\textsc{List}_j$: if the i'-first element u is removed from the list for v, u' becomes the i-first element for $i' \leq i \leq k$ if $(v, u, u') \in \textsc{List}_{i-i'+1}$. The update formulas exploit these insights:

$$
\begin{aligned}
\varphi^{\textsc{List}_i}_{\textsc{del}_E}(u,v;x,y,z) \overset{\text{def}}{=}\ & (v \neq x) \wedge \textsc{List}_i(x,y,z) \\
& \vee (v = x \wedge u \neq y \wedge \bigwedge_{i' \leq i} \neg\textsc{List}_{i'}(x,y,u) \wedge \textsc{List}_i(x,y,z)) \\
& \vee (v = x \wedge \bigvee_{\substack{j,j' \\ j+j'=i+1}} \textsc{List}_j(x,y,u) \wedge \textsc{List}_{j'}(x,u,z)) && \text{for } i \in \{1,\dots,k\} \\
\varphi^{\textsc{Last}_i}_{\textsc{del}_E}(u,v;x,y) \overset{\text{def}}{=}\ & (v \neq x \wedge \textsc{Last}_i(x,y)) \\
& \vee \Big(v = x \wedge \bigwedge_{i' \leq i} \neg\textsc{Last}_{i'}(u) \wedge \textsc{Last}_i(y)\Big) \\
& \vee \Big(v = x \wedge \bigvee_{i' \leq i} \big(\textsc{Last}_{i'}(u) \wedge \textsc{List}_{i-i'+1}(x,y,u)\big)\Big) && \text{for } i \in \{1,\dots,k\} \\
\varphi^{\textsc{First}_i}_{\textsc{del}_E}(u,v;x,y) \overset{\text{def}}{=}\ & (v \neq x \wedge \textsc{First}_i(x,y)) \\
& \vee \Big(v = x \wedge \bigwedge_{i' \leq i} \neg\textsc{First}_{i'}(u) \wedge \textsc{First}_i(y)\Big) \\
& \vee \Big(v = x \wedge \bigvee_{i' \leq i} \big(\textsc{First}_{i'}(u) \wedge \textsc{List}_{i-i'+1}(x,u,y)\big)\Big) && \text{for } i \in \{1,\dots,k\} \\
\varphi^{\textsc{Is}_i}_{\textsc{del}_E}(u,v;x) \overset{\text{def}}{=}\ & (v \neq x \wedge \textsc{Is}_i(x)) \\
& \vee (v = x \wedge \bigvee_{\substack{j,j' \\ j+j'-2=i}} \textsc{First}_j(x,u) \wedge \textsc{Last}_{j'}(x,u)) && \text{for } i \in \{0,\dots,k\} \\
\varphi^{\textsc{Is}_{>k}}_{\textsc{del}_E}(u,v;x) \overset{\text{def}}{=}\ & (v \neq x \wedge \textsc{Is}_{>k}(x)) \\
& \vee \Big(v = x \wedge \textsc{Is}_{>k}(x) \wedge \bigwedge_{\substack{j,j' \\ j+j'-2=k}} \big(\neg\textsc{First}_j(x,u) \vee \neg\textsc{Last}_{j'}(x,u)\big)\Big) && \triangleleft
\end{aligned}
$$

While DynProp includes many interesting dynamic queries, it still allows for general inexpressibility results. With a result that combines both aspects, Gelade et al. [2012] were able to characterise completely the expressive power of DynProp with respect to Boolean queries on strings, that is, formal languages, under single-tuple changes: a language L can be maintained in DynProp under changes of single positions precisely if L is regular. It follows from the construction that quantifier-free dynamic programs only need binary auxiliary relations to maintain a Boolean query on strings, simply because every regular language can be maintained in binary DynProp under single-tuple changes.

In this section we show that a similar result for DynProp on graphs is not possible and identify for every $k \geq 1$ a Boolean graph query that can be maintained with k-ary auxiliary relations in DynProp under single-tuple changes, but not with $(k-1)$-ary auxiliary relations. We say that DynProp has a strict arity hierarchy for this class of dynamic queries.

Theorem 4.13. DynProp *has a strict arity hierarchy with respect to Boolean graph queries under single-tuple changes.*

This theorem generalises multiple earlier results. Already Dong and Su [1998] prove that DynFO has an arity hierarchy, and this result translates to DynProp, but the query that separates k-ary DynProp from $(k-1)$-ary DynProp is a k-ary query on structures with $(6k+1)$-ary input relations.[5] Dong and Zhang [2000] reduce the arity of the input relations to $3k+1$. An arity hierarchy for Boolean graph queries up to $k=3$ was observed by Zeume and Schwentick [2015], and we re-use this result for the proof of Theorem 4.13. Zeume [2017] proves an arity hierarchy for DynProp if only insertions are allowed, based on the query k-CLIQUE, for different values of k. However, it is unknown whether k-CLIQUE can be maintained in DynProp if also deletions are allowed, for any $k \geq 3$.

The family of queries that witnesses the hierarchy for large arities consists of queries that are evaluated over *coloured graphs* with schema $\sigma \overset{\text{def}}{=} \{E, R\}$, that is, directed graphs (V, E, R) with an additional unary relation R that encodes a set of (red-)*coloured* nodes.[6] A node w of such a graph is said to be *covered* if there is a coloured node $v \in R$ with $(v, w) \in E$. The query ODDCOVEREDNODES asks, given a coloured graph, whether the number of covered nodes is odd.

We show that ODDCOVEREDNODES cannot be maintained with quantifier-free update formulas under single-tuple changes. A closer examination reveals the connection between variants of this query and the arity structure of DynProp. Let k be a natural number. The variant $\text{ODDCOVEREDNODES}_{\deg \leq k}$ of ODDCOVEREDNODES asks whether the number of covered nodes that additionally have in-degree at most k is odd. Note that $\text{ODDCOVEREDNODES}_{\deg \leq k}$ is a query on general coloured graphs, not only on graphs with bounded degree.

In the following subsections we show that $(\text{ODDCOVEREDNODES}_{\deg \leq k}, \Delta_\sigma)$[7] witnesses a strict arity hierarchy for DynProp, at least for arity at least 3. For this we first show in Subsection 4.2.1 that $\text{ODDCOVEREDNODES}_{\deg \leq k}$ can actually be maintained with k-ary auxiliary relations, for $k \geq 3$. Then, in Subsection 4.2.2 we introduce the technique that we will use in Subsection 4.2.3 to prove that $\text{ODDCOVEREDNODES}_{\deg \leq k}$ cannot be maintained in $(k-1)$-ary DynProp. In Subsection 4.2.4, we put the parts together.

We discuss in Subsection 4.2.5 how $\text{ODDCOVEREDNODES}_{\deg \leq k}$ can be maintained by dynamic programs that are allowed to use quantifiers. This is in particular interesting because ODDCOVEREDNODES seems to be a minor generalisation of the PARITY query, and statically both queries can be expressed by first-order formulas that are allowed to use a single parity quantifier. In the dynamic setting, ODDCOVEREDNODES is provably

[5] In the FOIES setting of Dong and Su [1998] the query relation is not considered to be an auxiliary relation and its arity is ignored for determining the maximal arity of a dynamic program. In our setting of DynFO, it is a trivial statement that a k-ary query cannot be maintained in $(k-1)$-ary DynFO, because the k-ary query relation needs to be stored as an auxiliary relation. However, Zeume [2015, Theorem 4.3.7] shows that a Boolean query that is very similar to the k-ary query used by Dong and Su [1998], and in particular is also over a $(6k+1)$-ary input schema, separates k-ary DynFO (DynProp) from $(k-1)$-ary DynFO (DynProp).

[6] We note that the additional relation R is for convenience of exposition. All our results are also valid for pure graphs: instead of using the relation R one could consider a node v coloured if it has a self-loop $(v, v) \in E$.

[7] Remember that Δ_σ denotes the set of all single-tuple changes over the schema $\sigma = \{E, R\}$.

harder than Parity, and it remains open whether it can be maintained in DynFO.

4.2.1 Expressibility results for the arity hierarchy

We start by proving that $\textsc{OddCoveredNodes}_{\text{deg}\leq k}$ can be maintained in DynProp using k-ary auxiliary relations. In Subsection 4.2.3 we show that this arity is optimal.

Proposition 4.14. *For every $k \geq 3$, $(\textsc{OddCoveredNodes}_{\text{deg}\leq k}, \Delta_\sigma)$ is in k-ary DynProp.*

Proof. Let $k \geq 3$ be some fixed natural number. We show how a DynProp-program $\mathcal{P}$ can maintain $(\textsc{OddCoveredNodes}_{\text{deg}\leq k}, \Delta_\sigma)$ using at most k-ary auxiliary relations.

The idea is as follows. Whenever a formerly uncoloured node v gets coloured, a certain number $c(v)$ of nodes become covered: v has edges to all these nodes, but no other coloured node has. Because the number $c(v)$ can be arbitrary, the program $\mathcal{P}$ necessarily has to store for each uncoloured node v the parity of $c(v)$ to update the query result. But this is not sufficient. Suppose that another node v' is coloured by a change and that, as a result, a number $c(v')$ of nodes become covered, because they have an edge from v' and so far no incoming edge from another coloured neighbour. Some of these nodes, say, $c(v, v')$ many, also have an incoming edge from v. Of course these nodes do not *become* covered any more when afterwards v is coloured, because they *are* already covered then. So, whenever a node v' gets coloured, the program $\mathcal{P}$ needs to update the (parity of the) number $c(v)$, based on the (parity of the) number $c(v, v')$. In turn, the (parity of the) latter number needs to be updated whenever another node v'' is coloured, using the (parity of the) analogously defined number $c(v, v', v'')$, and so on.

It seems that this reasoning does not lead to a construction idea for a dynamic program, as information for more and more nodes needs to be stored, but observe that only those covered nodes are relevant for the query that have in-degree at most k. So, a number $c(v_1, \ldots, v_k)$ does not need to be updated when some other node v_{k+1} gets coloured, because no relevant node has edges from all nodes $v_1, \ldots, v_{k+1}$.

We now present the construction in more detail. A node w is called *active* if its in-degree in-deg(w) is at most k. Let $A = \{a_1, \ldots, a_\ell\}$ be a set of coloured nodes and let $B = \{b_1, \ldots, b_m\}$ be a set of uncoloured nodes, with $\ell + m \leq k$. By $\mathcal{N}^{\bullet\circ}_G(A, B)$ we denote the set of active nodes w of the coloured graph G whose coloured (in-)neighbours are exactly the nodes in A and that have (possibly amongst others) the nodes in B as uncoloured (in-)neighbours. So, $w \in \mathcal{N}^{\bullet\circ}_G(A, B)$ if (1) in-deg$(w) \leq k$, (2) $(v, w) \in E$ for all $v \in A \cup B$, and (3) there is no edge $(v', w) \in E$ from a coloured node $v' \in R$ with $v' \notin A$. We omit the subscript G and just write $\mathcal{N}^{\bullet\circ}(A, B)$ if the graph G is clear from the context. The dynamic program $\mathcal{P}$ maintains the parity of $|\mathcal{N}^{\bullet\circ}_G(A, B)|$ for all such sets A, B.

Whenever a change $\alpha = \textsc{ins}_R(v)$ colours a node v of G, the update is as follows. We distinguish the three cases (1) $v \in A$, (2) $v \in B$ and (3) $v \notin A \cup B$. In case (1), the set $\mathcal{N}^{\bullet\circ}_{\alpha(G)}(A, B)$ equals the set $\mathcal{N}^{\bullet\circ}_G(A \setminus \{v\}, B \cup \{v\})$, and the existing auxiliary information can be copied. In case (2), actually $\mathcal{N}^{\bullet\circ}_{\alpha(G)}(A, B) = \emptyset$, as B contains a coloured node. The parity of the cardinality 0 of $\emptyset$ is even. For case (3) we distinguish two further cases.

If $|A \cup B| = k$, no active node w can have incoming edges from every node in $A \cup B \cup \{v\}$ as w has in-degree at most k, so $\mathcal{N}^{\bullet\circ}_{\alpha(G)}(A,B) = \mathcal{N}^{\bullet\circ}_{G}(A,B)$ and the existing auxiliary information is taken over. If $|A \cup B| < k$, then $\mathcal{N}^{\bullet\circ}_{\alpha(G)}(A,B) = \mathcal{N}^{\bullet\circ}_{G}(A,B) \setminus \mathcal{N}^{\bullet\circ}_{G}(A, B \cup \{v\})$ and $\mathcal{P}$ can combine the existing auxiliary information.

When a change $\alpha = \textsc{del}_R(v)$ uncolours a node v of G, the necessary updates are symmetrical. The case $v \in A$ is similar to case (2) above: $\mathcal{N}^{\bullet\circ}_{\alpha(G)}(A,B) = \emptyset$, because A contains an uncoloured node. The case $v \in B$ is handled similarly as case (1) above, as we have $\mathcal{N}^{\bullet\circ}_{\alpha(G)}(A,B) = \mathcal{N}^{\bullet\circ}_{G}(A \cup \{v\}, B \setminus \{v\})$. The third case $v \notin A \cup B$ is treated analogously as case (3) above, but in the sub-case $|A \cup B| < k$ we have that $\mathcal{N}^{\bullet\circ}_{\alpha(G)}(A,B) = \mathcal{N}^{\bullet\circ}_{G}(A,B) \cup \mathcal{N}^{\bullet\circ}_{G}(A \cup \{v\}, B)$.

Edge insertions and deletions are conceptionally easy to handle, as they change the sets $\mathcal{N}^{\bullet\circ}(A,B)$ by at most one element. Given all nodes of A and B and the endpoints of the changed edge as parameters, quantifier-free formulas can easily determine whether this is the case for specific sets A, B.

We now present $\mathcal{P}$ formally. For every $\ell \leq k+1$ the program maintains unary relations N_ℓ and $N^\bullet_\ell$ with the indented meaning that for a node w it holds $w \in N_\ell$ if in-deg$(w) = \ell$ and $w \in N^\bullet_\ell$ if w has exactly ℓ coloured in-neighbours. These relations can be maintained as presented in Example 4.12, requiring some additional, ternary auxiliary relations. We also use a relation $\textsc{Active} \overset{\text{def}}{=} N_1 \cup \cdots \cup N_k$ that contains all active nodes with at least one edge.

For every $\ell, m \geq 0$ with $1 \leq \ell + m \leq k$ the programs maintains $(\ell+m)$-ary auxiliary relations $P_{\ell,m}$ with the intended meaning that a tuple $(a_1, \dots, a_\ell, b_1, \dots, b_m)$ is contained in $P_{\ell,m}$ if and only if

- the nodes $a_1, \dots, a_\ell, b_1, \dots, b_m$ are pairwise distinct,
- $a_i \in R$ and $b_j \notin R$ for $i \in [\ell], j \in [m]$, and
- the set $\mathcal{N}^{\bullet\circ}(A,B)$ has an odd number of elements, where $A = \{a_1, \dots, a_\ell\}$ and $B = \{b_1, \dots, b_m\}$.

The following formula $\theta_{\ell,m}$ checks the first two conditions:

$$\theta_{\ell,m}(x_1, \dots, x_\ell, y_1, \dots, y_m) \overset{\text{def}}{=} \bigwedge_{i \neq j \in [\ell]} x_i \neq x_j \wedge \bigwedge_{i \neq j \in [m]} y_i \neq y_j \wedge \bigwedge_{i \in [\ell]} R(x_i) \wedge \bigwedge_{i \in [m]} \neg R(y_i)$$

Of course, $\mathcal{P}$ also maintains the Boolean query relation $\textsc{Ans}$.

We now describe the update formulas of $\mathcal{P}$ for the relations $P_{\ell,m}$ and $\textsc{Ans}$, assuming that each change actually alters the input graph, so, for example, no changes $\textsc{ins}_E(v,w)$ occur such that the edge (v,w) already exists.

Let $\varphi \oplus \psi \overset{\text{def}}{=} (\varphi \wedge \neg\psi) \vee (\neg\varphi \wedge \psi)$ denote the Boolean exclusive-or connector.

Colouring a node v A change $\textsc{ins}_R(v)$ increases the total number of active, covered nodes by the number of active nodes that have so far no coloured in-neighbour, but an edge from v. That is, this number is increased by $|\mathcal{N}^{\bullet\circ}(\emptyset, \{v\})|$. The update formula for $\textsc{Ans}$ is therefore

$$\varphi^{\textsc{Ans}}_{\textsc{ins}_R}(v) \overset{\text{def}}{=} \textsc{Ans} \oplus P_{0,1}(v).$$

We only spell out the more interesting update formulas for the relations $P_{\ell,m}$, for different values of ℓ, m. These formulas list the conditions for tuples $\bar{a} = a_1, \ldots, a_\ell$ and $\bar{b} = b_1, \ldots, b_m$ that $\mathcal{N}^{\bullet\circ}(\{a_1, \ldots, a_\ell\}, \{b_1, \ldots, b_m\})$ is of odd size after a change. The other update formulas are simple variants.

$$\begin{aligned}
&\varphi^{P_{\ell,m}}_{\text{INS}_R}(v; x_1, \ldots, x_\ell, y_1, \ldots, y_m) \stackrel{\text{def}}{=} \\
&\qquad \bigvee_{i \in [\ell]} \left(v = x_i \wedge P_{\ell-1,m+1}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_\ell, \bar{y}, v)\right) \\
&\quad \vee \Big(\bigwedge_{i \in [\ell]} v \neq x_i \wedge \bigwedge_{i \in [m]} v \neq y_i \wedge \big(P_{\ell,m}(\bar{x}, \bar{y}) \oplus P_{\ell,m+1}(\bar{x}, \bar{y}, v)\big)\Big) \qquad \text{for } \ell \geq 1, \ell + m < k \\
&\varphi^{P_{\ell,m}}_{\text{INS}_R}(v; x_1, \ldots, x_\ell, y_1, \ldots, y_m) \stackrel{\text{def}}{=} \\
&\qquad \bigvee_{i \in [\ell]} \left(v = x_i \wedge P_{\ell-1,m+1}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_\ell, \bar{y}, v)\right) \\
&\quad \vee \Big(\bigwedge_{i \in [\ell]} v \neq x_i \wedge \bigwedge_{i \in [m]} v \neq y_i \wedge P_{\ell,m}(\bar{x}, \bar{y})\Big) \qquad \text{for } \ell \geq 1, \ell + m = k
\end{aligned}$$

Uncolouring a node v The update formulas for a change $\text{DEL}_R(v)$ are analogous to the update formulas for a change $\text{INS}_R(v)$ as seen above. Again we only present a subset of the update formulas, the others are again easy variants.

$$\begin{aligned}
&\varphi^{\text{ANS}}_{\text{DEL}_R}(v) \stackrel{\text{def}}{=} \text{ANS} \oplus P_{1,0}(v) \\
&\varphi^{P_{\ell,m}}_{\text{DEL}_R}(v; x_1, \ldots, x_\ell, y_1, \ldots, y_m) \stackrel{\text{def}}{=} \\
&\qquad \bigvee_{i \in [m]} \left(v = y_i \wedge P_{\ell+1,m-1}(\bar{x}, v, y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_m)\right) \\
&\quad \vee \Big(\bigwedge_{i \in [\ell]} v \neq x_i \wedge \bigwedge_{i \in [m]} v \neq y_i \wedge \big(P_{\ell,m}(\bar{x}, \bar{y}) \oplus P_{\ell+1,m}(\bar{x}, v, \bar{y})\big)\Big) \qquad \text{for } m \geq 1, \ell + m < k \\
&\varphi^{P_{\ell,m}}_{\text{DEL}_R}(v; x_1, \ldots, x_\ell, y_1, \ldots, y_m) \stackrel{\text{def}}{=} \\
&\qquad \bigvee_{i \in [m]} \left(v = y_i \wedge P_{\ell+1,m-1}(\bar{x}, v, y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_m)\right) \\
&\quad \vee \Big(\bigwedge_{i \in [\ell]} v \neq x_i \wedge \bigwedge_{i \in [m]} v \neq y_i \wedge P_{\ell,m}(\bar{x}, \bar{y})\Big) \qquad \text{for } m \geq 1, \ell + m = k
\end{aligned}$$

Inserting an edge (v, w) When an edge (v, w) is inserted, the number of active, covered nodes can change at most by one. At first, a covered node w might become inactive. This happens when w had in-degree k before the insertion. Or, an active node w becomes covered. This happens if v is coloured and w had no coloured in-neighbour and in-degree at most $k-1$ before the change. The update formula for $\textsc{Ans}$ is accordingly

$$\varphi^{\textsc{Ans}}_{\textsc{ins}_E}(v,w) \stackrel{\text{def}}{=} \textsc{Ans} \oplus \Big(\big(N_k(w) \wedge \bigvee_{i\in[k]} N^\bullet_i(w)\big) \vee \big(R(v) \wedge N^\bullet_0(w) \wedge \bigvee_{i\in[k]} N_{i-1}(w)\big)\Big).$$

The necessary updated for relations $P_{\ell,m}$ are conceptionally very similar. We list the conditions that characterise whether the membership of w in $\mathcal{N}^{\bullet\circ}(A,B)$ changes, for a set $A=\{x_1,\ldots,x_\ell\}$ of coloured nodes and a set $B=\{y_1,\ldots,y_m\}$ of uncoloured nodes.

- Before the change, $w\in\mathcal{N}^{\bullet\circ}(A,B)$ holds, but not afterwards. This is either because w becomes inactive or because the new edge (v,w) connects w with another coloured node v. This case is expressed by the formula

$$\psi_1 \stackrel{\text{def}}{=} \bigwedge_{i\in[\ell]} E(x_i,w) \wedge N^\bullet_\ell(w) \wedge \bigwedge_{i\in[m]} E(y_i,w) \wedge \big(N_k(w) \vee R(v)\big).$$

- Before the change, $w\in\mathcal{N}^{\bullet\circ}(A,B)$ does not hold, but it does afterwards. Then w needs to be active and to have an incoming edge from all but one node from $A\cup B$, and v is that one node. Additionally, w has no other coloured in-neighbours. The following formulas ψ_2,ψ_3 express these conditions for the cases $v\in A$ and $v\in B$, respectively.

$$\psi_2 \stackrel{\text{def}}{=} \bigvee_{i\in[\ell]} \big(v=x_i \wedge \bigwedge_{j\in[\ell]\setminus\{i\}} E(x_j,w) \wedge \bigwedge_{j\in[m]} E(y_j,w) \wedge N^\bullet_{\ell-1}(w) \wedge \bigvee_{j\in[k]} N_{j-1}(w)\big)$$

$$\psi_3 \stackrel{\text{def}}{=} \bigvee_{i\in[m]} \big(v=y_i \wedge \bigwedge_{j\in[\ell]\setminus\{i\}} E(y_j,w) \wedge \bigwedge_{j\in[\ell]} E(x_j,w) \wedge N^\bullet_{\ell}(w) \wedge \bigvee_{j\in[k]} N_{j-1}(w)\big)$$

The update formula for $P_{\ell,m}$ is then

$$\varphi^{P_{\ell,m}}_{\textsc{ins}_E}(v,w;x_1,\ldots,x_\ell,y_1,\ldots,y_m) \stackrel{\text{def}}{=} \theta_{\ell,m}(\bar{x},\bar{y}) \wedge \big(P_{\ell,m}(\bar{x},\bar{y}) \oplus (\psi_1\vee\psi_2\vee\psi_3)\big).$$

Deleting an edge (v,w) The ideas to construct the update formulas for changes $\textsc{del}_E(v,w)$ are symmetrical to the constructions for changes $\textsc{ins}_E(v,w)$. When an edge (v,w) is deleted, the node w becomes active if its in-degree before the change was $k+1$. It is (still) covered, and then is a new active and covered node, if it has coloured in-neighbours other than v. This is the case if w has at least two coloured in-neighbours before the change, or if it has at least one coloured in-neighbour and v is not coloured.

On the other hand, if v was the only coloured in-neighbour of an active node w, this node is not covered any more. The update formula for the query relation $\textsc{Ans}$ is therefore

$$\begin{aligned}\varphi^{\textsc{Ans}}_{\textsc{del}_E}(v,w) \stackrel{\text{def}}{=} \textsc{Ans} \oplus \Big(&\big(N_{k+1}(w) \wedge \bigvee_{i\in[k+1]} N^\bullet_i(w) \wedge (\neg R(v) \vee \neg N^\bullet_1(w))\big) \\ &\vee \big(R(v) \wedge N^\bullet_1(w) \wedge \bigvee_{i\in[k]} N_i(w)\big)\Big)\end{aligned}$$

Regarding the update of relations $P_{\ell,m}$, we distinguish the same cases as for insertions $\textsc{ins}_E(v,w)$ for a set $A=\{x_1,\ldots,x_\ell\}$ of coloured nodes and a set $B=\{y_1,\ldots,y_m\}$ of uncoloured nodes.

- Before the change, $w \in \mathcal{N}^{\bullet\circ}(A, B)$ holds, but not afterwards. That means the active node w has incoming edges from all nodes from $A \cup B$, has no coloured in-neighbours apart from the nodes in A, and $v \in A \cup B$. This is expressed by the formula

$$\psi'_1 \stackrel{\text{def}}{=} \bigwedge_{i\in[\ell]} E(x_i, w) \wedge N^\bullet_\ell(w) \wedge \bigwedge_{i\in[m]} E(y_i, w) \wedge \bigvee_{i\in[k]} N_i(w) \\ \wedge \Big(\bigvee_{i\in[\ell]} x_i = v \vee \bigvee_{i\in[m]} y_i = v\Big).$$

- Before the change, $w \in \mathcal{N}^{\bullet\circ}(A, B)$ does not hold, but it does afterwards. Then w already is and stays connected to all nodes from $A \cup B$, and either have degree $k + 1$ and become active and/or loose an additional coloured in-neighbour v. The following formula ψ'_2 lists the possible combinations.

$$\psi'_2 \stackrel{\text{def}}{=} \bigwedge_{i\in[\ell]} (v \neq x_i \wedge E(x_i, w)) \wedge \bigwedge_{i\in[m]} (v \neq y_i \wedge E(y_i, w)) \\ \wedge \Big((N_{k+1}(w) \wedge \neg R(v) \wedge N^\bullet_j(w)) \\ \vee (N_{k+1}(w) \wedge R(v) \wedge N^\bullet_{j+1}(w)) \\ \vee \big(\bigvee_{i\in[k]} N_i(w) \wedge R(v) \wedge N^\bullet_{j+1}(w)\big)\Big)$$

Finally, the update formula for $P_{\ell,m}$ is

$$\varphi^{P_{\ell,m}}_{\text{DEL}_E}(v, w; x_1, \ldots, x_\ell, y_1, \ldots, y_m) \stackrel{\text{def}}{=} \theta_{\ell,m}(\bar{x}, \bar{y}) \wedge (P_{\ell,m}(\bar{x}, \bar{y}) \oplus (\psi'_1 \vee \psi'_2)). \qquad \square$$

4.2.2 Proving inexpressibility results for DynProp

Before we prove the inexpressibility results for the claimed arity hierarchy, we introduce the technique we use in this thesis to prove that some query cannot be maintained in DynProp using only k-ary auxiliary relations, for some k. This technique is a reformulation of the proof technique of Zeume [2017], which combines techniques from Gelade et al. [2012] and Zeume and Schwentick [2015] with insights regarding upper and lower bounds for Ramsey numbers.

We state a sufficient condition under which a query q cannot be maintained in k-ary DynProp under single-tuple changes. In Chapter 5, this condition will be generalised for certain more complex change operations.

For a Boolean query q, the condition basically requires that for each collection $\mathcal{B}$ of subsets of size $k + 1$ of a set $\{1, \ldots, n\}$ (for an arbitrary n larger than some n_0), there is a structure $\mathcal{D}$ and a change sequence $\beta(x_1, \ldots, x_{k+1})$ such that (1) the elements $1, \ldots, n$ cannot be distinguished by quantifier-free formulas, and (2) $\beta(i_1, \ldots, i_{k+1})(\mathcal{D})$ is a positive instance for q exactly if $\{i_1, \ldots, i_{k+1}\} \in \mathcal{B}$.

The actual statement of the condition is more general and allows non-Boolean queries, tuples $\bar{p}_1, \ldots, \bar{p}_n$ instead of elements $1, \ldots, n$, and additional parameters for the change sequence β.

The statement of the condition is also quite technical. Before we actually formulate the general statement in Lemma 4.18, we introduce the proof technique of Zeume [2017] with a rather simple example and re-prove that AlternatingReach cannot be maintained in DynProp, even when an arbitrary initialisation is allowed. This result appears already in [Gelade et al. 2012], although with a different construction. AlternatingReach is a query on *alternating graphs* $G = (V, X, U, E)$, that is, graphs (V, E) such that the node set $V = X \uplus U$ is partitioned into a set X of *existential* and a set U of *universal* nodes. The set AltReach(s) of nodes reachable from a node $s \in V$ is the smallest set that contains (1) the node s, (2) an existential node $u \in X$ if there is a node $v \in$ AltReach(s) with $(u, v) \in E$, and (3) a universal node $v \in U$ if $v \in$ AltReach(s) holds for all nodes v with $(u, v) \in E$. The query result AlternatingReach(G) contains all tuples (s, t) such that $t \in$ AltReach(s). We re-prove the following result, using the proof technique of Zeume [2017, Theorem 5.3].

Proposition 4.15 (cf. [Gelade et al. 2012, Proposition 6.2])**.** *Let k be any natural number.* (AlternatingReach, Δ_E) *is not in k-ary* DynProp*, even with arbitrary initialisation.*

In the following, we denote the set of all subsets of size k of a set A by $\binom{A}{k}$. The proof technique of Zeume [2017] uses the following combinatorial result which combines Ramsey upper and lower bounds.

Lemma 4.16 ([Zeume 2017, Lemma 5.2])**.** *Let $k \in \mathbb{N}$ be arbitrary and σ a k-ary schema. Then there is a monotone and unbounded function $f\colon \mathbb{N} \mapsto \mathbb{N}$ and an $n_0 \in \mathbb{N}$ such that for every domain A larger than n_0 the following conditions are satisfied:*

- *(S1) For every σ-structure $\mathcal{S}$ over A and every linear order $\prec$ on A there is a subset A' of A of size $|A'| \geq f(|A|)$ such that all $\prec$-ordered k-tuples over A' have the same quantifier-free type in $\mathcal{S}$.*
- *(S2) The set $\binom{A}{k+1}$ can be partitioned into two subsets $\mathcal{B}$ and $\mathcal{C}$ such that for every set $A' \subseteq A$ of size $|A'| \geq f(|A|)$ there are $B, C \in \binom{A'}{k+1}$ with $B \in \mathcal{B}$ and $C \in \mathcal{C}$.*

Intuitively speaking, this lemma says that for every large enough domain A there is a 2-colouring of the $(k+1)$-element subsets of the domain such that (1) in every structure $\mathcal{S}$ over A one can find a "large" sub-domain A' such that all (appropriately ordered) k-tuples over A' cannot be distinguished in $\mathcal{S}$ by quantifier-free formulas, but (2) for each colour there is a $(k+1)$-element subset of that colour that only contains elements from A'. So, while k-tuples over A' cannot be distinguished using the whole structure, subsets of A' of size $k+1$ can be distinguished using a pre-defined 2-colouring.

We remark that the unboundedness of f was not explicitly stated in [Zeume 2017] but it readily follows from its proof in [Zeume 2017], where f is chosen in $\Omega(\log^{(k-1)}(n))$. Monotonicity can be easily achieved.

Additionally, the proof technique utilises the *Substructure Lemma* as introduced by Gelade et al. [2012] and further developed by Zeume and Schwentick [2015]. This lemma formalises the insight that for DynProp programs, whether a tuple $\bar{c}$ is included in an auxiliary relation after a (single-tuple) change of a tuple $\bar{d}$ only depends on the elements contained in these two tuples. Suppose, we have two isomorphic states $\mathcal{S}$ and

$\mathcal{S}'$ of a DynProp program, so, input and auxiliary structures are isomorphic via some isomorphism π. A consequence of this insight is that after applying some changes $\delta(\bar{d})$ and $\delta(\pi(\bar{d}))$, respectively, the updated states are still isomorphic vie π.

We make this more formal now. Let π be an isomorphism from a structure $\mathcal{D}$ to a structure $\mathcal{D}'$. Two changes $\alpha = \delta(\bar{a})$ on $\mathcal{D}$ and $\alpha' = \delta'(\bar{b})$ on $\mathcal{D}'$ are said to be *π-respecting* if $\delta = \delta'$ and $\bar{b} = \pi(\bar{a})$. Two sequences $\beta = \alpha_1 \cdots \alpha_m$ and $\beta' = \alpha'_1 \cdots \alpha'_m$ of changes are π-respecting if α_i and α'_i are, for every $i \leq m$.

Lemma 4.17 (Substructure Lemma, [Zeume and Schwentick 2015])**.** *Let $\mathcal{P}$ be a* DynProp*-program and let $\mathcal{S}$ and $\mathcal{S}'$ be states of $\mathcal{P}$ with domains S and S'. Further let $A \subseteq S$ and $A' \subseteq S'$ such that $\mathcal{S}[A]$ and $\mathcal{S}'[A']$ are isomorphic via π. Then $\mathcal{P}_\beta(\mathcal{S})[A]$ and $\mathcal{P}_{\beta'}(\mathcal{S}')[A']$ are isomorphic via π for all π-respecting sequences β, β' of single-tuple changes on A and A'.*

Having the prerequisites in place, we now prove Proposition 4.15, using a construction very similar to the proof of [Zeume 2015, Proposition 4.1.19].

Proof (of Proposition 4.15). Assume, towards a contradiction, that there is a DynProp program $\mathcal{P}$ with k-ary auxiliary relations over schema σ_{aux} that maintains ALTERNATING-REACH. Let $\sigma = \sigma_{\text{aux}} \cup \{c_1, c_2\}$, where c_1, c_2 are additional constant symbols, and let f and n_0 be as guaranteed by Lemma 4.16 applied to k and σ. Let n be a number with $n \geq n_0$, let $A = \{1, \ldots, n, s, t\}$ and let $\prec$ be the order $1 \prec \cdots \prec n \prec s \prec t$ on A. Let $\mathcal{B}, \mathcal{C}$ be the partition of $\binom{A}{k+1}$ guaranteed to exist by Property (S2) of Lemma 4.16.

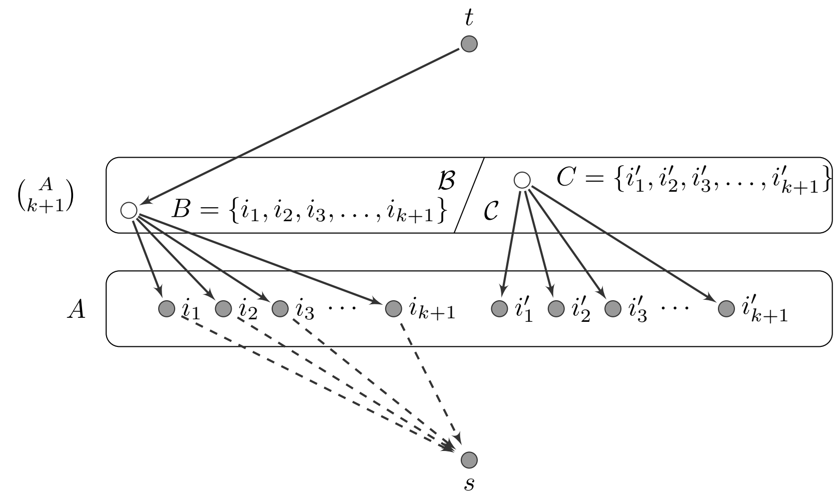
We consider an alternating graph $G = (V, X, U, E)$ with node set $V = A \cup \binom{A}{k+1} \cup \{s, t\}$. The set U of universal nodes consists of the elements from $\binom{A}{k+1}$, the remaining nodes constitute the set X of existential nodes. Let $\mathcal{S} = (\mathcal{D}, \mathit{Aux})$ be the state of $\mathcal{P}$ that is obtained by inserting the edges

- (S, i) for each set $S \in \binom{A}{k+1}$ and each $i \in S$ and
- (t, B) for each $B \in \mathcal{B}$

in some arbitrary order, starting from an empty input structure and an arbitrarily initialised auxiliary structure. See Figure 4.7 for a sketch.

Let Aux' be the σ-expansion of Aux where the constant symbols c_1, c_2 are interpreted by s, t, respectively. By Condition (S1) of Lemma 4.16 there is a subset $A' \subseteq A$ of size $|A'| \geq f(|A|)$ such that all $\prec$-ordered k-tuples over A' have the same type in Aux'. It must hold $s, t \notin A'$, as tuples involving constants cannot have the same type as tuples without constants. By Condition (S2), there are $B, C \in \binom{A'}{k+1}$ with $B \in \mathcal{B}$ and $C \in \mathcal{C}$. Let $B = \{i_1, \ldots, i_{k+1}\}$ and $C = \{i'_1, \ldots, i'_{k+1}\}$, with $i_j \prec i_{j+1}$ and $i'_j \prec i'_{j+1}$ for all $j \leq k$, and let β and β' be the change sequences that insert the edges $(i_1, s), \ldots, (i_{k+1}, s)$ and $(i'_1, s), \ldots, (i'_{k+1}, s)$ in that order, respectively.

Observe that the tuple (s, t) is in the query result for $\beta(\mathcal{D})$, but not in the query result for $\beta'(\mathcal{D})$. However, $\mathcal{P}$ gives the same answer in both cases by Lemma 4.17, as the substructures induced by $(i_1, \ldots, i_{k+1}, s, t)$ and by $(i'_1, \ldots, i'_{k+1}, s, t)$ are clearly isomorphic and the change sequences β and β' respect this isomorphism. So, contradicting the assumption, $\mathcal{P}$ does not maintain ALTERNATINGREACH. □



t
$\binom{A}{k+1}$
$\mathcal{B}$
$B = \{i_1, i_2, i_3, \dots, i_{k+1}\}$
$\mathcal{C}$
$C = \{i'_1, i'_2, i'_3, \dots, i'_{k+1}\}$
A
i_1 i_2 i_3 $\cdots$ i_{k+1} i'_1 i'_2 i'_3 $\cdots$ i'_{k+1}
s

Figure 4.7: A sketch of the graph from the proof of Proposition 4.15. Existential nodes are filled gray, universal nodes are white. The edges inserted by the change sequence β are dashed.

We now state and prove a lemma that serves as an abstraction for proofs like the proof of Proposition 4.15. It encapsulates the applications of Lemma 4.16 and Lemma 4.17, which is the same for every proof of this kind, from the problem specific construction of the input structure $\mathcal{D}$ and the change sequences β, β' as in the proof of Proposition 4.15 and sketched in Figure 4.7.

In the following, we denote the set $\{1, \dots, n\}$ by $[n]$ and write $(\mathcal{D}, \bar{a}) \equiv_0 (\mathcal{D}, \bar{b})$ if $\bar{a}$ and $\bar{b}$ have the same length and agree on their quantifier-free type in $\mathcal{D}$.

Lemma 4.18. *Let q be an m-ary σ-query. Then (q, Δ_σ) is not in k-ary* DynProp*, even with arbitrary initialisation, if there are natural numbers ℓ, r and, for each $n_0 \in \mathbb{N}$, a natural number $n \geq n_0$, such that for all subsets $\mathcal{B} \subseteq \binom{[n]}{k+1}$ there exist*

- *a σ-structure $\mathcal{D}$, a set $P = \{p_1^1, \dots, p_1^\ell, \dots, p_n^1, \dots, p_n^\ell\}$ of distinct elements, and tuples $\bar{a} = a_1, \dots, a_m$ and $\bar{u} = u_1, \dots, u_r$, such that*
  - *P and $\{a_1, \dots, a_m, u_1, \dots, u_r\}$ are disjoint and contained in the domain of $\mathcal{D}$,*
  - *$(\mathcal{D}, \bar{a}, \bar{u}, \bar{p}_{i_1}, \dots, \bar{p}_{i_{k+1}}) \equiv_0 (\mathcal{D}, \bar{a}, \bar{u}, \bar{p}_{j_1}, \dots, \bar{p}_{j_{k+1}})$ for all strictly increasing sequences $i_1, \dots, i_{k+1}$ and $j_1, \dots, j_{k+1}$ over $[n]$, where $\bar{p}_i = (p_i^1, \dots, p_i^\ell)$, and*
- *a sequence $\beta(x_1^1, \dots, x_1^\ell, \dots, x_{k+1}^1, \dots, x_{k+1}^\ell, y_1, \dots, y_r) = \alpha_1(\bar{x}_{i_1}, \bar{y}) \cdots \alpha_s(\bar{x}_{i_s}, \bar{y})$ of Δ_σ-changes, where no change α_j uses variables $x_{i_j}^t$ and $x_{i_{j'}}^{t'}$ with $i_j \neq i_{j'}$ as parameters,*

such that for all strictly increasing sequences $i_1, \dots, i_{k+1}$ over $[n]$ it holds that

$$\bar{a} \in q(\beta(\bar{p}_{i_1}, \dots, \bar{p}_{i_{k+1}}, \bar{u})(\mathcal{D})) \iff \{i_1, \dots, i_{k+1}\} \in \mathcal{B}.$$

It should be noted that, although $\mathcal{D}$ can contain further elements, the parameters in the considered change sequences are only instantiated by elements from $P \cup \{u_1, \dots, u_r\}$.

Proof (of Lemma 4.18). Let (q, Δ_σ) be a dynamic query where q is an m-ary σ-query. Suppose q satisfies the antecedent of the lemma, and let ℓ and r be numbers such that the conditions can be satisfied. Towards a contradiction, let us assume that there is a DynProp program $\mathcal{P}$ over a k-ary auxiliary schema σ_{aux} that maintains (q, Δ_σ).

Let σ be the schema $\{R_w \mid R \in \sigma_{\text{aux}}, w \in \{0, 1, \ldots, \ell\}^{\text{Ar}(R)}\} \cup \{s_1, \ldots, s_m\} \cup \{t_1, \ldots, t_r\}$, where the s_i and t_j are additional constant symbols and the arity of each relation symbol R_w equals the arity of R. Let f and n_0 be as guaranteed to exist by Lemma 4.16 applied to k and σ, and let n with $n \geq n_0$ be a number such that the conditions of the lemma can be satisfied.

Let $A \overset{\text{def}}{=} [n] \cup \{a_1, \ldots, a_m, u_1, \ldots, u_r\}$ and let $\mathcal{B}, \mathcal{C} \subseteq \binom{A}{k+1}$ be the partition guaranteed to exist by property (S2) of Lemma 4.16.

Let $\mathcal{D}, P, \bar{a}, \bar{u}$ and β be as in the condition of the statement of the lemma. Let Aux be the auxiliary structure over schema σ_{aux} that is obtained by $\mathcal{P}$ when the tuples of $\mathcal{D}$ are inserted in some arbitrary order to an initially empty structure, after an arbitrary initialisation. Our goal is to identify sequences $\bar{p}_1, \ldots, \bar{p}_{k+1}$ and $\bar{p}'_1, \ldots, \bar{p}'_{k+1}$ of tuples over P such that Lemma 4.16 and Lemma 4.17 imply that $\mathcal{P}$ gives the same answer when $\beta(\bar{p}_1, \ldots, \bar{p}_{k+1}, \bar{u})$ and $\beta(\bar{p}'_1, \ldots, \bar{p}'_{k+1}, \bar{u})$ are applied starting from the state $(\mathcal{D}, Aux)$, but the actual query results differ on the tuple $\bar{a}$. As a consequence, $\mathcal{P}$ cannot maintain (q, Δ_σ).

To be able to apply Lemma 4.16, we now construct a σ-structure $\mathcal{S}$ over domain A that encodes the restriction of Aux to domain $D \overset{\text{def}}{=} P \cup \{a_1, \ldots, a_m, u_1, \ldots, u_r\}$, intuitively by encoding the tuples $\bar{p}_1, \ldots, \bar{p}_n$ by numbers $1, \ldots, n$. We encode in the relation names which element of a tuple is actually represented by the number. Let $R \in \sigma_{\text{aux}}$ and let $(d_1, \ldots, d_{k'}) \in R^{Aux}$ be a tuple with elements from D. We let $\textsc{flat}(d_1, \ldots, d_{k'}) = (d', \ldots, d'_{k'})$ be the tuple with $d'_i = d_i$ if $d_i \in D \setminus P$ and $d'_i = j$ if $d_i = p^e_j$ for some $e \leq \ell$. We set $\textsc{flat}(d_1, \ldots, d_{k'}) \in R^{\mathcal{S}}_w$ if and only if for $w \overset{\text{def}}{=} w_1 \cdots w_{k'}$ it holds $w_i = 0$ if $d_i \in D \setminus P$ and $w_i = e$ if $d_i = p^e_j$ for some $j \leq n$. The additional constant symbols $s_1, \ldots, s_m, t_1, \ldots, t_r$ of $\mathcal{S}$ are interpreted by $a_1, \ldots, a_m, u_1, \ldots, u_r$.

Let A be ordered in the natural way by the linear order $\prec$ such that all elements from $[n]$ precede all other elements. Let A' be the subset of A as guaranteed by property (S1) of Lemma 4.16. Since all ordered tuples of A' have the same quantifier-free type in $\mathcal{S}$, they cannot involve constants and thus $A' \subseteq [n]$. Now property (S2) of Lemma 4.16 guarantees that there exist sets $B, C \in \binom{A'}{k+1}$ with $B \in \mathcal{B}$ and $C \in \mathcal{C}$. Let $i_1 < \cdots < i_{k+1}$ and $j_1 < \cdots < j_{k+1}$ be such that $B = \{i_1, \ldots, i_{k+1}\}$ and $C = \{i'_1, \ldots, i'_{k+1}\}$. Let β_1 and β_2 be the change sequences $\beta(\bar{p}_{i_1}, \ldots, \bar{p}_{i_{k+1}}, \bar{u})$ and $\beta(\bar{p}_{i'_1}, \ldots, \bar{p}_{i'_{k+1}}, \bar{u})$, respectively. It follows from the construction that $\bar{a} \in q(\beta_1(\mathcal{D}))$, but $\bar{a} \notin q(\beta_2(\mathcal{D}))$.

To reach the desired contradiction, we now prove that $\mathcal{P}$ gives the same query answer for both change sequences, starting from the state $(\mathcal{D}, Aux)$. For this, using Lemma 4.17 it is sufficient to observe that the substructures of $(\mathcal{D}, Aux)$ induced by $\bar{v}_1 \overset{\text{def}}{=} (\bar{a}, \bar{u}, \bar{p}_{i_1}, \ldots, \bar{p}_{i_{k+1}})$ and by $\bar{v}_2 \overset{\text{def}}{=} (\bar{a}, \bar{u}, \bar{p}_{i'_1}, \ldots, \bar{p}_{i'_{k+1}})$ are isomorphic via the canonical isomorphism π that maps the i-th component of $\bar{v}_1$ to the i-th component of $\bar{v}_2$, for every i. The change sequences clearly respect this isomorphism.

The substructures of $\mathcal{D}$ induced by $\bar{v}_1$ and $\bar{v}_2$ are isomorphic, as $(\mathcal{D}, \bar{v}_1) \equiv_0 (\mathcal{D}, \bar{v}_2)$

holds by the choice of the objects. It remains to show that also $(Aux, \bar{v}_1) \equiv_0 (Aux, \bar{v}_2)$ holds. Suppose, towards a contradiction, that this is not the case. As the $\{=\}$-type of the tuples is clearly the same, that means there is a relation R^{Aux} and a tuple $\bar{w}_1$ over elements of $\bar{v}_1$ such that $\bar{w}_1 \in R^{Aux} \Leftrightarrow \pi(\bar{w}_1) \notin R^{Aux}$. Because R is at most k-ary, $\bar{w}_1$ contains at most k elements $a_1, \ldots, a_k$. Observe that if $a_i = p^e_j$ for some i, j, e, then $\pi(a_i) = p^e_{j'}$ for some j'. It follows that there is a $w \in \{0, \ldots, \ell\}^{\text{Ar}(R)}$ such that $\textsc{flat}(\bar{w}_1) \in R^{\mathcal{S}}_w \Leftrightarrow \textsc{flat}(\pi(\bar{w}_1)) \notin R^{\mathcal{S}}_w$, and consequently there are $\prec$-ordered k-tuples over $[n]$ that have different quantifier-free types in $\mathcal{S}$, contradicting Property (S1) of Lemma 4.16. This concludes the proof. □

4.2.3 Inexpressibility results for the arity hierarchy

In this subsection we prove that k-ary auxiliary relations are not sufficient to maintain $\textsc{OddCoveredNodes}_{\text{deg}\leq k+1}$.

Proposition 4.19. *For every $k \geq 0$, $(\textsc{OddCoveredNodes}_{\text{deg}\leq k+1}, \Delta_\sigma)$ is not in k-ary* DynProp*, even with arbitrary initialisation.*

In the following, for a graph $G = (V, E)$ and some set $X \subseteq V$ of nodes we write $\mathcal{N}^{\rightarrow}(X)$ for the set $\{v \mid \exists u \in X\colon E(u, v)\}$ of out-neighbours of nodes in X. For singleton sets $X = \{x\}$ we just write $\mathcal{N}^{\rightarrow}(x)$ instead of $\mathcal{N}^{\rightarrow}(\{x\})$.

Proof. Let $k \geq 0$ be a fixed natural number. We apply Lemma 4.18 to show that $(\textsc{OddCoveredNodes}_{\text{deg}\leq k+1}, \Delta_\sigma)$ is not in k-ary DynProp.

The basic proof idea is simple. Given a collection $\mathcal{B} \subseteq \binom{[n]}{k+1}$, we construct a graph $G = (V, E)$ with distinguished nodes $P = \{p_1, \ldots, p_n\} \subseteq V$ such that (1) each node has in-degree at most $k + 1$ and (2) for each $B \in \binom{[n]}{k+1}$ the set $\mathcal{N}^{\rightarrow}(\{p_i \mid i \in B\})$ is of odd size if and only if $B \in \mathcal{B}$. Then applying a change sequence α which colours all nodes p_i with $i \in B$ to G results in a positive instance of $\textsc{OddCoveredNodes}_{\text{deg}\leq k+1}$ if and only if $B \in \mathcal{B}$. An invocation of Lemma 4.18 yields the intended lower bound.

It remains to construct the graph G. Let S be the set of all non-empty subsets of $[n]$ of size at most $k+1$. We choose the node set V of G as the union of P and S. Only nodes in P will be coloured, and only nodes from S will be covered. A first attempt to realise the idea mentioned above might be to consider an edge set $E_{k+1} \overset{\text{def}}{=} \{(p_i, B) \mid B \in \mathcal{B}, i \in B\}$: then, having fixed some set $B \in \mathcal{B}$, the node B becomes covered whenever the nodes p_i with $i \in B$ are coloured. However, also some other nodes $B' \neq B$ will be covered, namely if $B' \cap B \neq \emptyset$, and the number of these nodes influences the query result. We ensure that the set of nodes $B' \neq B$ that are covered by $\{p_i \mid i \in B\}$ is of even size, so that the parity of $|\mathcal{N}^{\rightarrow}(\{p_i \mid i \in B\})|$ is determined by whether $B \in \mathcal{B}$ holds. This will be achieved by introducing edges to nodes $\binom{[n]}{i} \in S$ for $i \leq k$ such that for every subset P' of P of size at most k the number of nodes from S that have an incoming edge from all nodes from P' is even. It follows that whenever $k + 1$ nodes $p_{i_1}, \ldots, p_{i_{k+1}}$ are marked, the number of covered nodes is odd precisely if there is one node in S that has an edge from *all* nodes $p_{i_1}, \ldots, p_{i_{k+1}}$, which is the case exactly if $\{i_1, \ldots, i_{k+1}\} \in \mathcal{B}$.

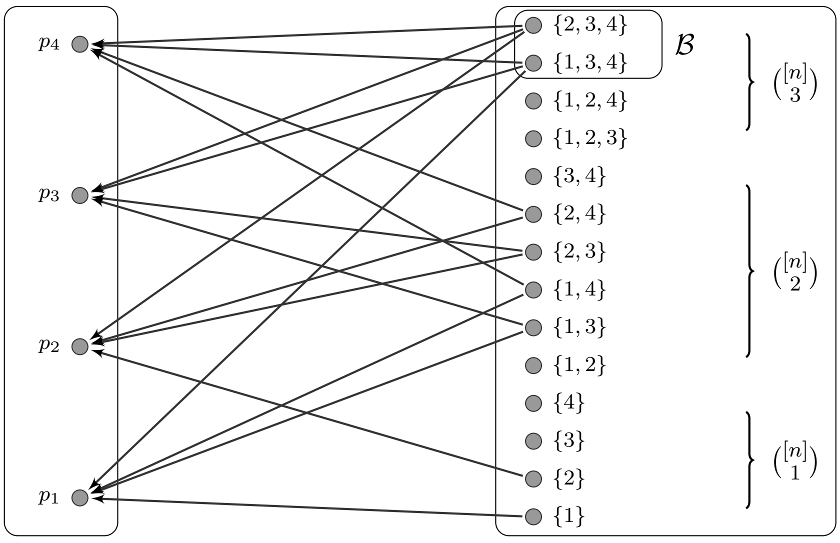


Figure 4.8: Example for the construction in the proof of Proposition 4.19, with $k = 2$ and $n = 4$.

We now make this precise. We have $m = 0$ and choose $\ell = 1, r = 0$ in the statement of Lemma 4.18. Let n be arbitrary and let $P = \{p_1, \dots, p_n\}$. For a set $X \subseteq [n]$ we write P_X for the set $\{p_i \mid i \in X\}$.

The structure $\mathcal{D}$ we construct consists of a coloured graph $G = (V, E, R)$ with nodes $V \stackrel{\text{def}}{=} P \cup S$, where $S \stackrel{\text{def}}{=} \binom{[n]}{1} \cup \dots \cup \binom{[n]}{k+1}$, and initially empty set $R \stackrel{\text{def}}{=} \emptyset$ of coloured nodes. The edge set $E = E_1 \cup \dots \cup E_{k+1}$ is constructed iteratively in $k + 1$ steps. We first define the set E_{k+1} and define the set E_j based on the set $E_{>j} \stackrel{\text{def}}{=} \bigcup_{j'=j+1}^{k+1} E_{j'}$.

The set E_{k+1} consists of all edges (p_i, B) such that $B \in \mathcal{B}$ and $i \in B$. For the construction of the set E_j with $j \in \{1, \dots, k\}$ we assume that all sets $E_{j'}$ with $j' > j$ have already been constructed. Let $X \in \binom{[n]}{j}$ be a set and let m be the number of nodes $Y \in S$ for which there are already edges $(p_i, Y) \in E_{>j}$ for all nodes p_i in P_X. If m is odd, then there is so far an odd number of nodes from S that have an incoming edge from all $p_i \in P_X$. As we want this number to be even, we let E_j contain edges (p_i, X) for all $i \in X$. If m is even, no edges are added to E_j. See Figure 4.8 for an example of this construction. Note that for each $X \in \binom{[n]}{i}$, for $i \in \{1, \dots, k+1\}$, the degree of X in G is at most i, and therefore also at most $k + 1$.

We now show that for a set $B \in \binom{[n]}{k+1}$ the cardinality of $\mathcal{N}^{\rightarrow}(P_B)$ is indeed odd if and only if $B \in \mathcal{B}$. This follows by an inclusion-exclusion argument. For a set $X \subseteq [n]$ the set $\mathcal{N}^{\rightarrow}(P_X)$ contains all nodes with an incoming edge from a node in P_X. It is therefore equal to the union $\bigcup_{i \in X} \mathcal{N}^{\rightarrow}(p_i)$. When we sum up the cardinalities of these sets $\mathcal{N}^{\rightarrow}(p_i)$, any node in $\mathcal{N}^{\rightarrow}(P_X)$ with edges to both p_i and p_j, for numbers $i, j \in X$, is accounted for twice. Continuing this argument, the cardinality of $\mathcal{N}^{\rightarrow}(X)$ can be computed as follows.

$$|\mathcal{N}^{\to}(P_X)| = \sum_{i \in X} |\mathcal{N}^{\to}(p_i)| - \sum_{\substack{i,j \in X \\ i<j}} |\mathcal{N}^{\to}(p_i) \cap \mathcal{N}^{\to}(p_j)| + \dots + (-1)^{|X|-1} |\bigcap_{i \in X} \mathcal{N}^{\to}(p_i)|$$

By construction of G, the set $\bigcap_{i \in Y} \mathcal{N}^{\to}(p_i)$ is of even size, for all sets $Y \subseteq [n]$ of size at most k. Consequently, for each $X \in \binom{[n]}{k+1}$ the parity of $|\mathcal{N}^{\to}(P_X)|$ is determined by the parity of $|\bigcap_{i \in X} \mathcal{N}^{\to}(p_i)|$, the last term in the above equation. Only the node X can possibly have incoming edges from all nodes p_i in P_X, and these edges exist if and only if $X \in \mathcal{B}$.

Let $\alpha(x_1), \dots, \alpha(x_{k+1})$ be the change sequence $\text{INS}_R(x_1), \dots, \text{INS}_R(x_{k+1})$ that colours the nodes $x_1, \dots, x_{k+1}$. Let $B \in \binom{[n]}{k+1}$ be of the form $\{i_1, \dots, i_{k+1}\}$ with $i_1 < \dots < i_{k+1}$. The change sequence $\alpha_B \overset{\text{def}}{=} \alpha(p_{i_1}) \cdots \alpha(p_{i_{k+1}})$ results in a graph where the set of coloured nodes is exactly P_B. As all nodes in $\mathcal{N}^{\to}(P_B)$ have degree at most $k+1$ and the set $\mathcal{N}^{\to}(P_B)$ is of odd size exactly if $B \in \mathcal{B}$, we have that $\alpha_B(\mathcal{D})$ is a positive instance of $\text{ODDCOVEREDNODES}_{\deg \le k+1}$ if and only if $B \in \mathcal{B}$.

It follows from Lemma 4.18 that $(\text{ODDCOVEREDNODES}_{\deg \le k+1}, \Delta_\sigma)$ cannot be maintained in k-ary DynProp. □

Immediately from Proposition 4.19 we get the following general inexpressibility result.

Corollary 4.20. $(\text{ODDCOVEREDNODES}, \Delta_\sigma)$ *is not in* DynProp.

4.2.4 Obtaining the arity hierarchy

We can now combine the results of the previous subsections and prove the main theorem of this section.

Proof (of Theorem 4.13). For every $k \ge 1$ we give a Boolean graph query that can be maintained in k-ary DynProp but not in $(k-1)$-ary DynProp under single-tuple changes.

For $k \ge 3$, we choose the dynamic query $(\text{ODDCOVEREDNODES}_{\deg \le k}, \Delta_\sigma)$ which can be maintained in k-ary DynProp but not in $(k-1)$-ary DynProp, by Proposition 4.14 and Proposition 4.19.

For $k = 2$, already Zeume and Schwentick [2015] prove that the query S-T-TWOPATH which asks whether there exists a path of length 2 between some distinguished vertices s and t separates unary DynProp from binary DynProp [Zeume and Schwentick 2015, Proposition 4.10].

For $k = 1$, we consider the Boolean graph query PARITYDEGREEDIV3 that asks whether the number of nodes whose degree is divisible by 3 is odd. This query can easily be maintained in DynProp using only unary auxiliary relations. In a nutshell, a dynamic program can maintain for each node v the degree of v modulo 3. So, it maintains three unary relations M_0, M_1, M_2 with the intention that $v \in M_i$ if the degree of v is congruent to i modulo 3. These relations can easily be updated under edge insertions and deletions. Similar as for PARITY, a bit P that gives the parity of $|M_0|$ can easily be maintained.

On the other hand, ParityDegreeDiv3 cannot be maintained in DynProp using nullary auxiliary relations. Suppose, towards a contradiction, that it can be maintained by some dynamic program $\mathcal{P}$ that only uses nullary auxiliary relations, and consider an input instance that contains five node $V = \{u_1, u_2, v_1, v_2, v_3\}$ as well as edges $E = \{(u_1, v_1), (u_1, v_2), (u_2, v_1)\}$. No matter the auxiliary structure, $\mathcal{P}$ needs to give the same answer after the changes $\alpha_1 \overset{\text{def}}{=} \textsc{ins}_E(u_1, v_3)$ and $\alpha_2 \overset{\text{def}}{=} \textsc{ins}_E(u_2, v_3)$, as it cannot distinguish these tuples using quantifier-free first-order formulas. But α_1 leads to a yes-instance for ParityDegreeDiv3, and α_2 does not. So, $\mathcal{P}$ does not maintain ParityDegreeDiv3. □

4.2.5 Allowing quantification

We have already mentioned that DynFO also has a strict arity hierarchy, witnessed by a family of queries over input schemas of growing arity [Dong and Su 1998; Zeume 2015]. It is an open question whether DynFO has an arity hierarchy with respect to graph queries. In the following we see that the queries OddCoveredNodes$_{\text{deg}\le k}$ do not witness an arity hierarchy for dynamic programs that are allowed to use quantification: we prove that (OddCoveredNodes$_{\text{deg}\le k}$, Δ_σ) is in binary DynFO for any k. We actually prove that a linear order on the active domain, which can be maintained easily in DynFO [Etessami 1998], is the only binary auxiliary relation we need.

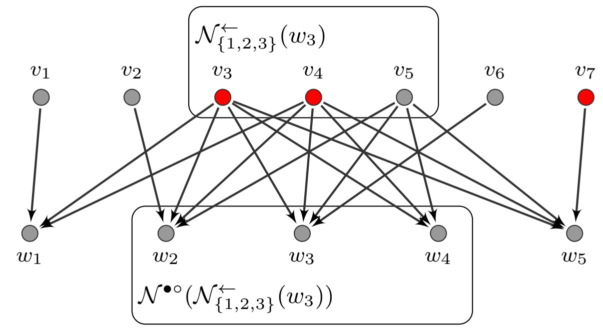
Proposition 4.21. *In the presence of a linear order,* (OddCoveredNodes$_{\text{deg}\le k}$, Δ_σ) *is in unary* DynFO*, for every $k \in \mathbb{N}$.*

An intuitive reason why quantifier-free dynamic programs need auxiliary relations of growing arity to maintain OddCoveredNodes$_{\text{deg}\le k}$ is that for checking whether some change, for instance the colouring of a node v, is "relevant" for some node w, it needs to have access to all of w's "important" neighbours. Without quantification, the only way to do this is to explicitly list them as elements of the tuple for which the update formula decides whether to include it in the auxiliary relation.

With quantification and a linear order, sets of neighbours can be defined more easily, if the total number of neighbours is bounded by a constant. Let us fix a node w with at most k (in-)neighbours, for some constant k. Thanks to the linear order, the neighbours can be distinguished as first, second, …, k-th neighbour of w, and any subset of these nodes is uniquely determined and can be defined in FO by the node w and a set $I \subseteq \{1, \dots, k\}$ that *indexes* the neighbours. With this idea, the proof of Proposition 4.14 can be adjusted appropriately for Proposition 4.21.

Proof sketch (of Proposition 4.21). Let $k \in \mathbb{N}$ be some constant. Again, we call a node *active* if its in-degree is at most k. We sketch a dynamic program that uses a linear order on the nodes and otherwise at most unary auxiliary relations.

Let I be a non-empty subset of $\{1, \dots, k\}$, and let w be an active node with at least $\max(I)$ in-neighbours. The set $\mathcal{N}_I^{\leftarrow}(w)$ of *I-indexed in-neighbours* of w includes a node v if and only if (v, w) is an edge in the input graph and v is the i-th in-neighbour of w with respect to the linear order, for some $i \in I$. The following notation is similar as in



$\mathcal{N}^{\leftarrow}_{\{1,2,3\}}(w_3)$
v_1
v_2
v_3
v_4
v_5
v_6
v_7
w_1
w_2
w_3
w_4
w_5
$\mathcal{N}^{\bullet\circ}(\mathcal{N}^{\leftarrow}_{\{1,2,3\}}(w_3))$

Figure 4.9: An illustration of the notation used in the proof of Proposition 4.21. The set $\mathcal{N}^{\bullet\circ}_G(\mathcal{N}^{\leftarrow}_{\{1,2,3\}}(w_3))$ does not include w_1, as there is no edge (v_5, w_1), and it does not include w_5, as there is an edge (v_7, w_5) for a coloured node $v_7 \notin \mathcal{N}^{\leftarrow}_{\{1,2,3\}}(w_3)$.

the proof of Proposition 4.14. For a graph G and an arbitrary set C of (coloured and uncoloured) nodes, we denote the set of active nodes that have an incoming edge from every node in C and no coloured in-neighbour that is not in C by $\mathcal{N}^{\bullet\circ}_G(C)$. An example for these notions is depicted in Figure 4.9.

For every $I \subseteq \{1, \dots, k\}$ with $I \neq \emptyset$ we introduce an auxiliary relation P_I with the following intended meaning. An active node w with at least $\max(I)$ neighbours is in P_I if and only if (1) w has no coloured in-neighbours that are not contained in $\mathcal{N}^{\leftarrow}_I(w)$, and (2) the set $\mathcal{N}^{\bullet\circ}_G(\mathcal{N}^{\leftarrow}_I(w))$ has odd size. Note that (1) implies that $w \in \mathcal{N}^{\bullet\circ}_G(\mathcal{N}^{\leftarrow}_I(w))$.

An auxiliary relation P_I basically replaces the relations $P_{\ell,m}$ with $\ell + m = |I|$ from the proof of Proposition 4.14, and the updates are mostly analogous.

We explain how the query relation Ans and the relations P_I are updated when a modification to the input graph occurs. When a node v is coloured, the query relation is only changed if v becomes the only coloured neighbour of an odd number of active nodes. This is the case if and only if there actually is an active and previously uncovered node w that v has an edge to and if $w \in P_I$ for the set $I \overset{\text{def}}{=} \{i\}$, where i is the number such that v is the i-th in-neighbour of w with respect to the linear order.

The update of a relation P_I after the colouring of a node v is as follows. Let G be the graph before the change is applied, and G' the changed graph. Let w be any active node. If v is an I-indexed in-neighbour of w, no change regarding $w \in P_I$ is necessary. Otherwise, some nodes in $\mathcal{N}^{\bullet\circ}_G(\mathcal{N}^{\leftarrow}_I(w))$ might now have a coloured neighbour v that is not contained in $\mathcal{N}^{\leftarrow}_I(w)$, and therefore are not contained in $\mathcal{N}^{\bullet\circ}_{G'}(\mathcal{N}^{\leftarrow}_I(w))$. Let w' be such a node, that is, a node with an edge from v and every node in $\mathcal{N}^{\leftarrow}_I(w)$, and let I' be such that $\mathcal{N}^{\leftarrow}_{I'}(w') = \mathcal{N}^{\leftarrow}_I(w) \cup \{v\}$. The parity of the number of nodes in $\mathcal{N}^{\bullet\circ}_G(\mathcal{N}^{\leftarrow}_I(w)) \setminus \mathcal{N}^{\bullet\circ}_{G'}(\mathcal{N}^{\leftarrow}_I(w))$ is odd if and only if $w' \in P_{I'}$. This can be used to update P_I.

We do not present the updates for the remaining changes as they can be easily constructed along the same lines. □

It is easy to maintain a linear order on the non-isolated nodes of an input graph

[Etessami 1998], which is all that is needed for the proof of Proposition 4.21. So, $\textsc{OddCoveredNodes}_{\deg\leq k}$ can also be maintained in DynFO without a predefined linear order, at the expense of binary auxiliary relations.

Corollary 4.22. *For every $k \in \mathbb{N}$, $(\textsc{OddCoveredNodes}_{\deg\leq k}, \Delta_\sigma)$ is in binary* DynFO*.*

Unfortunately we cannot generalise the proof technique from Proposition 4.21 for $\textsc{OddCoveredNodes}_{\deg\leq k}$ to $\textsc{OddCoveredNodes}$, but only to a similarly defined variant $\textsc{OddCoveredNodes}_{\deg\leq \log n}$ that asks for the parity of the number of covered nodes with in-degree at most $\log n$. Here, n is the number of nodes of the graph.

Proposition 4.23. $(\textsc{OddCoveredNodes}_{\deg\leq \log n}, \Delta_\sigma)$ *is in binary* DynFO*, in the presence of a linear order and* BIT*.*

Proof sketch. With the help of the linear order we identify the node set V of size n of the input graph with the numbers $\{0, \dots, n-1\}$, and use BIT to access the bit encoding of these numbers. Any node $v \in V$ then naturally encodes a set $I(v) \subseteq \{1, \dots, \log n\}$: $i \in \{1, \dots, \log n\}$ is contained in $I(v)$ if and only if the i-th bit in the bit encoding of v is 1.

The proof of Proposition 4.21 constructs a dynamic program that maintains unary relations P_I with $I \subseteq \{1, \dots, k\}$, and $w \in P_I$ holds if $w \in \mathcal{N}_G^{\bullet\circ}(\mathcal{N}_I^{\leftarrow}(w))$ and if $|\mathcal{N}_G^{\bullet\circ}(\mathcal{N}_I^{\leftarrow}(w))|$ is odd. We replace these relations by a single binary relation P, with the intended meaning that $(v, w) \in P$ if $w \in \mathcal{N}_G^{\bullet\circ}(\mathcal{N}_{I(v)}^{\leftarrow}(w))$ and if $|\mathcal{N}_G^{\bullet\circ}(\mathcal{N}_{I(v)}^{\leftarrow}(w))|$ is odd.

A dynamic program that maintains $\textsc{OddCoveredNodes}_{\deg\leq \log n}$ can then be constructed along the same lines as in the proof of Proposition 4.21. □

In addition to a linear order, Etessami [1998] also shows how corresponding relations addition and multiplication can be maintained for the active domain of a structure. As BIT is first-order definable in the presence of addition and multiplication, and vice versa [Immerman 1999], both a linear order and BIT on the active domain can be maintained, still using only binary auxiliary relations. So, the variant of $\textsc{OddCoveredNodes}_{\deg\leq \log n}$ that considers n to be the number of non-isolated nodes, instead of the number of all nodes, can be maintained in binary DynFO without assuming built-in relations. It is an open problem whether $\textsc{OddCoveredNodes}$ can be maintained in DynFO.

4.3 Outlook and bibliographic remarks

We briefly summarise the contributions of this chapter and state some open questions.

In Section 4.1 we have seen that a large class of queries, namely all MSO-definable queries on graphs of bounded treewidth, can be maintained in DynFO under single-tuple changes. Although this shows that a large subset of LOGSPACE-queries [Elberfeld et al. 2010] is in DynFO under single-tuple changes, it is still unknown whether all (some variant of) domain independent queries from a complexity class $\mathcal{C}$ with $\mathsf{AC}^0 \subsetneq \mathcal{C}$ can be maintained.

The $\mathsf{AC}^0[2]$ query $\textsc{OddCoveredNodes}$ introduced in Section 4.2 seems to be a good candidate for research in this direction. We showed that $\textsc{OddCoveredNodes}_{\deg\leq k}$ for each k and even OddCoveredNodes$_{\text{deg}\leq\log n}$ can be maintained in DynFO with auxiliary relations of small arities, but for the general query OddCoveredNodes, maintainability is still unknown. This query demonstrates that static complexity and dynamic complexity may behave very differently. Statically, Parity and OddCoveredNodes are very similar, they ask for the parity of the size of a set that is definable by a quantifier-free first-order formula and a formula with one existential quantifier, respectively. Both are natural problems for the complexity class $\mathsf{AC}^0[2]$ and can be solved by $\mathsf{AC}^0[2]$ circuits with a single "modulo 2" gate. Dynamically, the difference in complexity is striking. Parity can be maintained easily in DynProp, while we do not even know membership in DynFO for OddCoveredNodes.

If OddCoveredNodes cannot be maintained in DynFO under single-tuple changes, and if we can actually prove this, then we can conclude that DynFO does not capture any static complexity class $\mathcal{C}$ that includes $\mathsf{AC}^0[2]$, which does not leave much open. On the other hand, if we can actually maintain OddCoveredNodes, hopefully we can extract from the dynamic program a strategy to dynamically maintain some greater class of $\mathsf{AC}^0[2]$ queries. Note also the relationship of this question to Chapter 5: OddCoveredNodes under single-tuple changes can be interpreted as Parity under certain first-order definable change operations.

We state some more open problems. For maintaining MSO-definable queries, the Muddling technique helps to provide a tree decomposition, although a slightly outdated one and with non-optimal width. Is it possible to maintain an up-to-date tree decomposition for the input graph at hand? Note that the dynamic programs from Section 4.1 can trivially be extended to maintain a tree decomposition of width $\mathcal{O}(\log n)$: during the $\log n$ changes that need to be processed, for each insertion of an edge (u, v) the dynamic program selects bags B_u and B_v that contain u and v, respectively, adds u to all bags on the path between B_u and the lowest common ancestor of B_u and B_v in the tree decomposition, and analogously for v and B_v. Can we maintain for every graph with treewidth k a tree decomposition of width ck, for some constant c? Even of width k? Can we do this if k is restricted to 2?

A related question is whether one can maintain whether the input graph has treewidth at most k, for a fixed k. Note that this is possible under the assumption that the input graph at all times has treewidth at most k', for some fixed constant $k' > k$. This is because for every k the class of graphs of treewidth at most k is characterised by a finite list of forbidden minors[8] [Robertson and Seymour 1990], one can express in MSO whether a graph contains a fixed minor, and we can maintain this MSO property for graphs of treewidth at most k' by Theorem 4.3. However, we cannot actually construct a dynamic program for arbitrary values of k, because the result of Robertson and Seymour [1990] is non-constructive.

For the previous approach and also for the results in Section 4.1 we assume that the treewidth of the input graph never exceeds a certain bound. Another open question is whether this assumption can be weakened: can a dynamic program maintain MSO

[8]A graph G is a *minor* of a graph H if G can be constructed from H by deleting nodes and edges and contracting edges.

properties in the sense that it gives the correct answer whenever the input graph's treewidth is below a certain threshold, and does not give an answer otherwise? In particular, can the dynamic program "recover" when the treewidth of the input graph becomes "small" again? Note that the dynamic program from Theorem 4.3 achieves this goal with a "delay" of $\log n$ changes: it can maintain an MSO-definable property for $\log n$ changes after the input graph's treewidth exceeds the bound, and it can answer the query again when the treewidth has been below the bound for the last $\log n$ changes.

We have seen that MSO-definable queries can be maintained for graphs of bounded treewidth. Can the Muddling technique be used to maintain interesting queries for other classes of graphs, for example planar graphs? As already mentioned in Section 3.3, preliminary results, obtained together with Samir Datta, Siddharth Iyer, Anish Mukherjee and Thomas Zeume, suggest that an extension of Theorem 4.3 yields dynamic programs that can maintain approximate solutions to certain MSO-definable optimisation problems on planar graphs. The problems considered there are NP-hard, even when restricted to planar graphs, so they probably cannot be maintained exactly in DynFO. Other problems for planar graphs, like isomorphism [Datta et al. 2009], are in LOGSPACE; as $\mathsf{LOGSPACE} \subseteq \mathsf{AC}^1$, it is possible that the concept of $(\mathsf{AC}^1, \log n)$-maintainability can be applied.

The just mentioned graph isomorphism query is interesting also for other classes of graphs. It can be maintained in DynFO under single-edge changes for trees [Etessami 1998]. Can this result be extended to graph classes of treewidth $k \geq 2$?

The main result of the second part of this chapter is Theorem 4.13, an arity hierarchy for DynProp under Boolean graph queries. This result underscores that Goal 4.2 might be very hard to achieve in its full generality. In particular, we require stronger proof techniques to show that certain dynamic graph queries cannot be maintained in DynProp. Lemma 4.18 provides a relatively easy way of utilising the proof technique from Zeume [2017], and it will in particular be used in Chapter 5 to show that Reach is not in DynProp under certain first-order definable change operations (see Theorem 5.11), but so far we cannot show the corresponding result for single-edge changes. The current frontier is ternary DynProp (c.f. [Zeume and Schwentick 2015]): can $(\textsc{Reach}, \Delta_E)$ be maintained in DynProp with at most ternary auxiliary relations?

Bibliographic remarks

The results of Section 4.1 were published first as [Datta et al. 2017]. They are joint work with Samir Datta, Anish Mukherjee, Thomas Schwentick and Thomas Zeume. The presentation in this thesis, in particular the proof of Theorem 4.7, follows the full version [Datta et al. 2019]. The arity hierarchy for DynProp as presented in Section 4.2 is joint work with Thomas Zeume and accepted for publication [Vortmeier and Zeume 2020]. The proof strategy for DynProp lower bound results was introduced by Zeume [2017], its reformulation as Lemma 4.18 is published in a slightly weaker form in [Schwentick et al. 2018] as joint work with Thomas Schwentick and Thomas Zeume. The result of Proposition 4.15 is originally published in [Gelade et al. 2012, Proposition 6.2], and its proof in this thesis is an adaptation of the proof of [Zeume 2015, Proposition 4.1.19].

CHAPTER 5

Expressibility under definable changes

The SQL standard allows for several ways to alter the contents of a database table beyond single-tuple changes. We give an example for these capabilities.

Example 5.1. Suppose our database models a street network. One table `streets` with attributes `start` and `to` represents the existing road segments that go from one crossing to another one. Another table `usable_streets` with the same attributes represents those road segments that can actually be used by vehicles. There is a third table `open_crossings` with one attribute `name` that lists crossings that can be passed at the moment.

Suppose now that some construction works at the crossing `university` are finished, so all street segments that go from this crossing to another open crossing are now usable. The corresponding change of the table `usable_streets` can be implemented by the following SQL statement.

```
INSERT INTO usable_streets (start, to)
  SELECT start, to
  FROM   streets, open_crossings
  WHERE  start = 'university' AND to = name;
```

On the other hand, the following statement removes all tuples from `usable_streets` that mention a crossing that is not listed in `open_crossings`.

```
DELETE FROM usable_streets
WHERE start NOT IN (SELECT name FROM open_crossings)
   OR to    NOT IN (SELECT name FROM open_crossings);
```

◁

So, SQL offers a declarative formalism to change table contents, allowing to insert or to delete a set of tuples that is specified as the result of some query on the current database. In this chapter we study the question which queries can be maintained in DynFO and its subclasses under such declaratively defined changes. For this we allow changes that replace each input relation by the result of a first-order query on the input structure, possibly using elements as parameters. Again employing the equivalence between first-order logic and relational algebra, this is a natural formalisation of the update mechanism of SQL[1], and several authors posed the study of such changes as an open question [Patnaik and Immerman 1997; Etessami 1998; Grädel and Siebertz 2012; Zeume 2015].

Consequently, the main goal of this chapter is to lift dynamic programs for single-tuple changes to dynamic programs that support changes defined by first-order queries.

Goal 5.1. Show maintainability in DynFO under (classes of) first-order definable changes for queries that are in DynFO under single-tuple changes.

Intuitively, it should be harder for a dynamic program to maintain a query under definable changes than under single-tuple changes, as single-tuple changes are a special case of first-order definable changes. On the one hand, this makes it more difficult to show membership in DynFO, but on the other hand, non-maintainability proofs should become easier, which leads to the formulation of another goal for this chapter.

Goal 5.2. Show for specific queries that they cannot be maintained dynamically in DynFO under certain first-order definable changes.

Lower bound proofs of this kind come in (at least) two flavours. Firstly, they can provide a barrier for maintainability, when they state that a query cannot be maintained under first-order definable changes, although it can be maintained under some smaller set of changes, for example single-tuple changes. Secondly, they can be a first step towards a proof that the query in question cannot even be maintained under single-tuple changes.

Contributions The main contributions of this chapter concern maintainability of the reachability query under first-order definable changes. We show that reachability in undirected graphs can still be maintained in DynFO under first-order defined *insertions*, that is, definable changes that are guaranteed to preserve all existing edges, while still allowing single-edge deletions. For acyclic graphs, we show that reachability can be maintained in DynFO under insertions that are defined by quantifier-free first-order formulas, and single-edge deletions.

These results rely on the so-called *bounded bridge property*: we show that in each graph obtained by an insertion as specified above, each reachable pair of nodes is connected by a path that only uses a constant number of newly inserted edges, where this constant only depends on the first-order formula defining the change. Updating the transitive closure of a graph then only requires to concatenate a constant number of already existing paths along newly inserted edges, which is easily doable in first-order logic. The size of the

[1]Of course, SQL sub-queries that go beyond relational algebra cannot be expressed in our setting. See also [Ameloot et al. 2013] for a theoretical investigation of the expressive power of SQL update primitives.

resulting dynamic programs are consequently proportional to the constant given by the bounded bridge property, which can be huge. We therefore identify natural classes of first-order definable insertions that yield small dynamic programs maintaining undirected reachability. This investigation is the theoretical basis of the experimental evaluation of the dynamic approach in Chapter 7.

So far, we have no results that reachability can be maintained for general directed graphs under definable insertions, or for some non-trivial graph class under definable deletions. We however provide barriers for this kind of results. Firstly, we show that very easy first-order definable insertions for general directed graphs as well as for acyclic directed graphs do not have the bounded bridge property, so the technique we introduce for the maintainability results cannot be used in these settings. Building on this insight, we show that arbitrary unary auxiliary relations and the transitive closure relation are not sufficient to maintain REACH in these cases. Also, these relations do not suffice to maintain reachability for directed graphs, even acyclic ones, under definable deletions.

We still do not know whether REACH is in DynProp under single-edge changes. In this chapter we prove that this is not the case if we allow insertions or deletions definable by quantifier-free first-order formulas.

Apart from the reachability query, we also study maintenance of formal languages under definable changes. One result states that all regular languages can still be maintained in DynProp under quantifier-free changes. Additionally, all context-free languages are in DynFO under changes that are defined by first-order formulas without quantifier alternation. We briefly discuss why changes that are defined via first-order formulas *with* quantifier alternation seem to be harder to handle.

Intuition (as well as the formulation of Goal 5.1) suggests that all queries that are in DynFO under some class of definable changes need to be in DynFO under single-tuple changes. We show that this is not necessarily the case and study parameter-free changes, that is, changes that are defined by first-order formulas that do not use parameters. In SQL terminology, we can think of sub-queries that do not mention constants, and refer to the second expression of Example 5.1 for an example for such a change. This class does not subsume single-tuple changes and therefore maintainability result do not have direct implications for maintainability under single-tuple changes. The last result of this chapter states that all AC^1 queries can be maintained in DynFO under parameter-free first-order changes.

The results presented in this chapter are obtained jointly with Thomas Schwentick and Thomas Zeume. More detailed bibliographic remarks are given at the end of the chapter.

Outline We formally introduce definable changes in Section 5.1. The next two sections are devoted to the reachability query: Section 5.2 gives the positive results, Section 5.3 the negative results regarding maintainability of REACH under various classes of definable insertions and deletions. We continue with the investigation of formal languages under changes with restricted quantification in Section 5.4. In Section 5.5, we study AC^1 queries under parameter-free changes. We give an outlook and further bibliographic comments in Section 5.6.

5.1 Logically definable change operations

In this section we introduce first-order definable change operations. Intuitively, these operations replace some of the input relations by the result of first-order queries on the input structure. Before me make this approach more formal, we illustrate this idea by a few examples.

Example 5.2.

(a) Actually, a single-tuple insertion is just a special case of a first-order definable change operation. It only affects one relation of the input structure. Consider the change operation $\text{INS}_E(p_1, p_2)$ that allows to insert edges into the edge relation E of a graph. The formula $\mu(p_1, p_2; x, y) \stackrel{\text{def}}{=} E(x, y) \vee (x = p_1 \wedge y = p_2)$ with free variables x, y and parameters p_1, p_2 defines a relation that includes all previous edges and (if that edge was not already present before) one additional new edge (p_1, p_2). An edge insertion $\text{INS}_E(a, b)$ can be expressed by instantiating the parameters of μ with a and b and replacing E with the result of $\mu(x, y)$.

(b) Suppose that a user wants to connect some node of a directed graph to every other node of this graph. This can be achieved by a change operation that replaces the edge relation with the relation defined by the formula $\mu(p; x, y) \stackrel{\text{def}}{=} E(x, y) \vee (x = p)$ with parameter p.

(c) The following example allows to change two relations by one operation. Here, we consider directed graphs with two additional unary relations C_1, C_2, which we interpret as a colouring of the nodes. Let $G = (V, E, C_1, C_2)$ be such a coloured graph and suppose a user wants to swap the colours C_1, C_2 for all nodes that have an edge to a node $u \in V$. This can be achieved by simultaneously replacing C_1 with the relation defined by $\mu_{C_1}(p; x) \stackrel{\text{def}}{=} (\neg E(x, p) \wedge C_1(x)) \vee (E(x, p) \wedge C_2(x))$ and replacing C_2 with the relation defined by $\mu_{C_2}(p; x) \stackrel{\text{def}}{=} (\neg E(x, p) \wedge C_2(x)) \vee (E(x, p) \wedge C_1(x))$, where for both formulas the parameter p is instantiated by the same node u. We emphasise that these two formulas specify *one* change operation.

(d) Finally, we consider an example of a parameter-free insertion. It states that, in a graph, all nodes u and v that are connected by a path of length 2 should be connected directly. The new edge relation is defined by the formula $\mu(x, y) \stackrel{\text{def}}{=} E(x, y) \vee \exists z (E(x, z) \wedge E(z, y))$. $\lhd$

We now formally define first-order definable change operations. A *replacement rule* ρ_R for a relation symbol R is of the form **replace** R **by** $\mu_R(\bar{p}; \bar{x})$. Here, $\mu_R(\bar{p}; \bar{x})$ is a first-order formula, the length of the tuple $\bar{x}$ equals the arity of R and $\bar{p}$ is a tuple of element parameters. A *replacement query* $\rho(\bar{p})$ is a *set* of replacement rules for distinct relation symbols with the same parameter tuple $\bar{p}$. In the case of replacement queries ρ that consist of a single replacement rule, we usually do not distinguish between ρ and its single replacement formula μ_R.

We define the semantics of replacement queries. Let $\mathcal{D}$ be a structure and $\alpha = \rho(\bar{a})$ a change over the domain of $\mathcal{D}$. The result $\alpha(\mathcal{D})$ of the application of α to $\mathcal{D}$ is defined in a straightforward way: each relation $R^{\mathcal{D}}$ in $\mathcal{D}$, for which there is a replacement rule ρ_R in ρ, is replaced by the relation resulting from evaluating μ_R, that is, by $\{\bar{b} \mid \mathcal{D} \models \mu_R(\bar{a}; \bar{b})\}$. Other relations that are not explicitly replaced by ρ remain unchanged.

Some of our investigations will focus on (syntactically) restricted replacement queries that either only remove or only insert tuples to relations. For an *insertion rule* ρ_R, the replacement formula $\mu_R(\bar{p}; \bar{x})$ has the form $R(\bar{x}) \vee \varphi_R$. Similarly, *deletion rules* have replacement formulas $\mu_R(\bar{p}; \bar{x})$ of the form $R(\bar{x}) \wedge \neg\varphi_R$.

Other syntactic restrictions to be studied in this chapter are *parameter-free* replacement queries (that allow no parameters in change formulas) and *quantifier-free* replacement queries (that allow only quantifier-free change formulas). A special case of quantifier-free changes are the single-tuple changes. As already sketched in Example 5.2, an insertion $\textsc{ins}_R(\bar{p})$ can be expressed by the insertion query **replace** R **by** $\mu_R(\bar{p}; \bar{x})$, where $\mu_R(\bar{p}; \bar{x}) \overset{\text{def}}{=} R(\bar{x}) \vee (\bar{p} = \bar{x})$. Analogously, a deletion $\textsc{del}_R(\bar{p})$ is expressed by the deletion query **replace** R **by** $\mu_R(\bar{p}; \bar{x})$, where $\mu_R(\bar{p}; \bar{x}) \overset{\text{def}}{=} R(\bar{x}) \wedge \neg(\bar{p} = \bar{x})$.

5.2 Maintaining Reachability under first-order definable insertions

The reachability query is arguably the most studied query in the field of dynamic complexity, and it was shown very early that in the setting of single-tuple changes it can be maintained in DynFO on undirected as well as directed acyclic graphs [Patnaik and Immerman 1997; Dong and Su 1995; 1998]. Here, we extend these results to first-order definable *insertions*, while still allowing deletions of single edges.

These results rely on the so-called *bounded bridge property.* In a nutshell, a change operation has the bounded bridge property, if whenever a graph resulting from such an operation has a path from some node u to some node v, it also has such a path that uses only a bounded number of newly inserted edges.

The remainder of this section consists of three parts. In the first part we consider the undirected reachability query UREACH and show that this query can be maintained under first-order definable insertions (and single-edge deletions).

The dynamic programs we construct are unfortunately rather impractical, as their size is non-elementary in the size of the first-order insertions, due to the fact that the constant given by the bounded bridge property, the *bridge bound*, might be huge in general. Therefore we identify in the second part subclasses of first-order definable insertions that result in smaller programs: insertions that are defined by (unions of) conjunctive queries, with and without negation. We show that these change operations have small bridge bounds. These results motivate our experiments with a prototypical implementation of the resulting dynamic programs in Chapter 7.

We do not know whether also the directed reachability query REACH can be maintained under general first-order insertions. In the third part of this section we present the solution for a special case: a dynamic program for REACH under quantifier-free insertions on directed acyclic graphs.

All the programs we construct in this section basically use the same auxiliary relations as in the case of single-tuple changes. In Section 5.3 we show that extending the dynamic program for single-tuple insertions (see Example 2.8) with arbitrary unary auxiliary relations is not sufficient to maintain REACH under first-order insertions in cyclic graphs. There we also present barriers for maintainability under first-order deletions and lower bounds for quantifier-free dynamic programs.

5.2.1 Undirected Reachability

In this subsection, we show that the undirected reachability query can be maintained in DynFO under single-edge changes and any finite set of first-order definable insertions. The respective dynamic programs rely on the fact that for every undirected path between two nodes u, v in the graph after the insertion there is an undirected path between u and v that only uses a bounded number of newly inserted edges.

We formalise this idea with the following notion. Let G' be a graph that is obtained from a graph G by an insertion of edges. For two nodes u, v of G', the *undirected bridge distance* $\text{ubd}_{G,G'}(u, v)$ is the minimal number d such that there is an undirected path from u to v in G' that uses at most d edges that are not in G. We will refer to the new edges in such paths as *bridges*. Since the two graphs will always be clear from the context, we usually simply write $\text{ubd}(u, v)$ instead of $\text{ubd}_{G,G'}(u, v)$. We say that an insertion query ρ has the *undirected bounded bridge property* if there is a constant c such that, for every graph G' resulting from a graph G by applying ρ, and all nodes u, v of G, $\text{ubd}_{G,G'}(u, v) \leq c$. The smallest such c is called the *undirected bridge bound* of ρ.

The undirected bridge bound of a single-edge insertion is of course 1, but we now prove that each first-order insertion also has a constant undirected bridge bound.

Proposition 5.3. *Every first-order definable insertion query has the undirected bounded bridge property.*

Proof. We show that each first-order definable insertion operation ρ with underlying first-order formula $\mu(\bar{p}; \bar{x})$ of quantifier rank k has the undirected bounded bridge property. Let ℓ be the length of $\bar{p}$ and c' the number of FO$[k, 1]$-types of directed[2] graphs. We will bound the number of bridges on undirected paths created by ρ by $c \overset{\text{def}}{=} \ell + c'$. For this we use Theorem 2.2, which implies that if a change, defined by a first-order formula of quantifier rank k, inserts edges (a, b) and (c, d) for nodes a, b, c, d from different weakly connected components such that a and c as well as b and d have the same rank-k type, respectively, then the change also inserts the edges (a, d) and (c, b).

Let G be a graph and let $\alpha = \rho(\bar{a})$ for some tuple $\bar{a}$ of nodes of G. Let u, v be two nodes that are connected by some undirected path π of the form $u = w_0, w_1, \ldots, w_r = v$ in $\alpha(G)$. Let q be the number of bridges of π, that is, edges that are not in G. Our goal is to show that there exists such an undirected path with at most c bridges. For $q \leq c$, there is nothing to prove, so we assume $q > c$. It suffices to show that there is a path from u to v with fewer than q bridges. Let $(u_1, v_1), \ldots, (u_q, v_q)$ be the bridges in π. If for some i, the nodes u_i and v_i are in the same weakly connected component of G (before the application of α), we can replace the bridge (u_i, v_i) by a path of "old"

[2]Remember that we formally consider undirected paths in directed graphs.

edges resulting in an overall path with $q - 1$ bridges. Similarly, if u_i and u_j are in the same weakly connected component of G, for some $i < j$, we can shortcut π by a path from u_i to u_j inside G. Therefore, we can assume that, for every i, the nodes u_i and v_i are in different weakly connected components of G, and likewise u_i and u_j for $i < j$.

We show that in this case there are i, j with $i < j$ such that μ defines an edge between u_i and v_j, and therefore a path with fewer bridges can be constructed by shortcutting the path π with the edge (u_i, v_j). Indeed, by the choice of m there must be two nodes u_i and u_j, with $i < j$, in distinct weakly connected components of G that do not contain any element from $\bar{a}$, such that u_i and u_j have the same $\mathsf{FO}[k, 1]$-type in their respective connected components. By Theorem 2.2 it follows that $(u_i, v_j, \bar{a})$ and $(u_j, v_j, \bar{a})$ have the same $\mathsf{FO}[k, \ell + 2]$-types and therefore, since μ defines an edge between u_j and v_j, it also defines an edge between u_i and v_j. Clearly, the path $u, \dots, u_i, v_j, \dots, v$ has fewer bridges than π. □

With the help of Proposition 5.3 we can show the following result.

Theorem 5.4. *Let Δ be a finite set of first-order insertion queries. Then* $(\text{UR}\textsc{each}, \Delta \cup \Delta_E)$ *can be maintained in* DynFO.

For the proof, we use the approach for maintaining UREACH under single-edge insertions and deletions from [Dong and Su 1998, Theorem 4.3] and maintain a directed spanning forest and its transitive closure relation. The undirected bounded bridge property allows the update of the spanning forest and its transitive closure in a first-order definable way.

Proof. The dynamic program presented in [Dong and Su 1998, Theorem 4.3] maintains the undirected transitive closure of graphs under single-edge changes with the help of auxiliary relations H and TC_H. The binary relation H is a directed forest, whose undirected version is a spanning forest of the undirected version of the input graph G, and TC_H is its transitive closure.[3] Observe that two nodes u and v are in the same weakly connected component if and only if $\{(w, u), (w, v)\} \subseteq TC_H$ holds, for some node w.

We show how to maintain the relations H and TC_H for a single first-order insertion query ρ. Since Δ is finite, the update program for Δ is just the union of the update programs for each $\rho \in \Delta$. For the moment we assume a predefined linear order $\leq$ on the domain to be present; later we show how this assumption can be dropped. Let G be a graph and $\alpha = \rho(\bar{a})$ an insertion, H a directed spanning forest of G and TC_H its transitive closure. We show how to define in first-order logic the updated auxiliary relations H' and TC'_H for the modified graph $G' = \alpha(G)$.

We first describe a strategy to define H' and then argue that it can be implemented by a first-order formula. We call the smallest node of a weakly connected component C' of G' with respect to $\leq$ the *queen* u_0 of C'. For each weakly connected component C of G that is a subgraph of C', we define its *queen level* as the (unique) number $\text{ubd}(u_0, u)$, for

[3]In [Dong and Su 1998], undirected graphs are considered but it is easy to see that UREACH on directed graphs can be basically maintained in the same way. Furthermore, the relation H is used slightly differently, but the adjustments are straightforward.

nodes $u \in C$. Thanks to Proposition 5.3 the queen level of each component is bounded by a constant c.

The spanning tree of a weakly connected component C' is constructed by inserting bridges into the spanning forest of the subgraph of G that is induced by C'. More precisely, a directed edge (u, v) is inserted into the spanning forest if the weakly connected components of u and v in G have queen levels ℓ and $\ell + 1$ with respect to their queen u_0 in G', for some $\ell < c$, and (u, v) is the lexicographically smallest pair with respect to $\leq$ for which α inserts (u, v) or (v, u). The lexicographically minimal bridges can be defined by a first-order formula because the undirected bridge distance between any two nodes in any component of $\alpha(G)$ is at most c and because there are formulas $\theta_h(x, y)$ expressing that $\text{ubd}(x, y) \leq h$, for each number h. Such a formula θ_h basically quantifies h bridges and checks, using the stored result of UReach for G, that these bridges connect x and y.

Observe that an edge (u, v) might be inserted into the spanning tree such that v is not the root of its spanning tree in H. To obtain a directed spanning forest H', the direction of some edges in the directed spanning tree of the component of v needs to be flipped accordingly, as described in [Dong and Su 1998, Lemma 4.2].

The construction of H' ensures that along each directed path in H' at most c new edges are inserted, therefore it is straightforward to extend the update formula of [Dong and Su 1998, Lemma 4.2] for $TC_{H'}$. The update formulas for the deletion of edges are the same as in the single-edge modification case.

So far we have seen how the auxiliary relations can be updated in first-order logic using $\leq$. It remains to show how the assumption of a predefined linear order can be eliminated. For a change sequence β, we denote by A_β the set of parameters used in β. When applying β to an initially empty graph, a linear order on A_β can be easily constructed as in the case of single-tuple changes [Etessami 1998]. The crucial observation is that for each node $b \in V$, either all the remaining nodes in $V \setminus A_\beta$ have an edge to b or none of them. More precisely, one can show by induction on $|\beta|$ that for all nodes $a \in V$ and $b, b' \in V \setminus A_\beta$ it holds $(a, b) \in E \Leftrightarrow (a, b') \in E$ (and likewise $(b, a) \in E$ if and only if $(b', a) \in E$). The proof uses a similar idea as the proof of Proposition 5.3: one can show that the insertion query cannot distinguish b from b', therefore an edge (a, b) is inserted if and only if an edge (a, b') is inserted.

The dynamic program for maintaining UReach without a predefined linear order maintains the relations H and TC_H as described above, yet restricted to the induced (and ordered) subgraph G_β of G on A_β. Whether two nodes are in the same weakly connected component of G_β can thus be inferred from H and TC_H. Whether two nodes are in the same weakly connected component of G can be easily determined as follows.

- Two nodes $a, a' \in A_\beta$ are in the same weakly connected component of G if and only if they are in the same weakly connected component of G_β or there is a node $d \in V \setminus A_\beta$ such that there are edges (b, d) (or (d, b)) and (b', d) (or (d, b')) connecting d with some node b in the weakly connected component of a in G_β and some node b' in the weakly connected component of a' in G_β.
- Two nodes $a \in A_\beta$ and $b \in V \setminus A_\beta$ are connected by a path if and only if there is an edge between b and some node in the weakly connected component of a in G_β.

- Finally, two nodes $a, a' \in V \setminus A_\beta$ are connected if and only they are connected by an edge or they are both connected by an edge to some node $b \in A_\beta$. $\square$

5.2.2 Towards efficient dynamic programs

The size of the dynamic programs from the proof of Theorem 5.4 depends on the undirected bridge bounds of the insertion queries in Δ. Unfortunately, in the worst case the bridge bound grows non-elementarily in the quantifier rank, that is, its growth cannot be bounded by a fixed-height tower of exponential functions. We therefore turn next to insertion queries with more desirable bridge bounds (and consequently reasonably small dynamic programs). More precisely, we consider insertion queries definable by variants of conjunctive queries. We will see that for all these variants the bridge bounds are bounded by the size s of the insertion queries. Therefore, the size of the dynamic update programs grows at most linearly in s.

Conjunctive queries correspond to select-project-join queries in the relational algebra and to select-from-where queries with conjunctions of atoms in where-clauses in SQL [Abiteboul et al. 1995], and constitute one of the most investigated query languages for relational databases.

More formally, the class UCQ contains all *unions of conjunctive queries* (short: *UCQs*), that is, queries expressible by formulas of the form $\varphi(\bar{w}) = \bigvee_{i=1}^{\ell} \exists \bar{z} \psi_i$, where each ψ_i is a conjunction of atomic formulas. A *conjunctive query* is a UCQ for which $\ell = 1$ holds. The class of conjunctive queries is denoted by CQ. The extensions of CQ and UCQ by allowing negated atoms are denoted by $\mathsf{CQ}^\neg$ and $\mathsf{UCQ}^\neg$, respectively. In the following, a UCQ-*definable insertion query* is a replacement query $\rho(\bar{p})$ with a formula of the form $E(x, y) \vee \varphi(\bar{p}, x, y)$, where φ is a union of conjunctive queries. We will refer to $\varphi(\bar{p}, x, y)$ as *insertion formula*. Analogously, we define CQ-, $\mathsf{CQ}^\neg$- and $\mathsf{UCQ}^\neg$-definable insertion queries.

We will see that the undirected bridge bound of CQ-definable insertions is at most two and, more generally, for UCQ-definable insertions based on the union of ℓ conjunctive queries it is at most 2ℓ. There is no constant undirected bridge bound for $\mathsf{CQ}^\neg$-definable insertions, but it is still linear in the size of the query.

We first state the result for UCQ-definable insertions.

Proposition 5.5.

(a) For each UCQ*-definable insertion query ρ that is a union of ℓ conjunctive queries, the undirected bridge bound of ρ is at most 2ℓ.*
(b) The bound of Statement (a) is tight.

Before we prove the statements, we introduce some further notation. In the following, we associate a (hyper-)graph H with a conjunctive query ψ as it is common in the literature on CQs [Gottlob et al. 2001]: the nodes of H_ψ are the variables of ψ and each atom of ψ becomes a (hyper-)edge in H_ψ. Note that H_ψ does not need to be connected. Each maximal weakly connected component induces a conjunctive query $\theta = \exists \bar{y}\, \theta'$, where θ' is the conjunction of all atoms of the component and $\bar{y}$ contains only the quantified variables of the component. We call these formulas θ the *patterns* of ψ and denote for each variable x the unique pattern in which x occurs by θ^x.

Proof (of Proposition 5.5). (a) Let us assume, towards a contradiction, that there is a UCQ-definable insertion query ρ with insertion formula $\varphi(\bar{a}; x, y)$, a graph G, a change $\alpha = \rho(\bar{a})$ and two nodes u, v such that $\text{ubd}_{G,\alpha(G)}(u, v) \geq 2\ell + 1$, where ℓ is the number of conjunctive queries that constitute $\varphi = \bigvee_{i=1}^{\ell} \exists \bar{z}\, \psi_i$. That is, there is an undirected path π in $\alpha(G)$ from u to v with at least $2\ell + 1$ bridges, but no such path with fewer bridges. By the pigeonhole principle, at least three (not necessarily consecutive) bridges $(u_1, v_1), (u_2, v_2), (u_3, v_3)$ from π are inserted by the same CQ ψ_i. In particular, $G \models \exists \bar{z}\, \psi_i(\bar{a}, u_1, v_1)$ and $G \models \exists \bar{z}\, \psi_i(\bar{a}, u_3, v_3)$. Without loss of generality, we assume that u_1 is the first of these six nodes that occurs on the path from u to v and that (u_3, v_3) is the last of the three edges on this path. We however do not specify whether u_3 is visited before or after v_3 in the undirected path π. We show that α also inserts the edge (u_1, v_3) that can be used to shortcut π, resulting in an undirected path from u to v with fewer bridges, which is the desired contradiction.

We examine the patterns of ψ_i. Observe that the patterns θ_i^x and θ_i^y are different: if x and y were in the same connected component of ψ_i, then every assignment η that satisfies ψ_i needs to map x and y to nodes that are connected by an undirected path, and thus ψ_i only inserts edges inside weakly connected components of a graph, contradicting the assumption that ψ_i inserts at least three bridges. Now, let η_1 and η_3 be assignments witnessing that ψ_i inserts the edges (u_1, v_1) and (u_3, v_3), respectively. So, η_j maps (x, y) to (u_j, v_j), $\bar{p}$ to $\bar{a}$ and the quantified variables $\bar{z}$ to some nodes $\bar{b}_j$, and all patterns of ψ_i are satisfied under both η_1 and η_3. Let η' be the assignment that agrees with η_1 on the variables of the pattern θ_i^x and with η_3 on the remaining variables. In particular, η' agrees with η_3 on the variables of the pattern θ_i^y. Observe that η' satisfies every pattern of ψ_i and therefore proves that $G \models \exists \bar{z}\, \psi_i(\bar{a}, u_1, v_3)$ and that α inserts the edge (u_1, v_3), as desired.

(b) We now show that the bound of Statement (a) is actually tight. For the construction we use, for natural numbers q, conjunctive queries $\chi_q(z) \overset{\text{def}}{=} \exists z_1, \ldots, z_{q-1}\, E(z, z_1) \wedge E(z_1, z_2) \wedge \cdots \wedge E(z_{q-1}, z)$ expressing that z lies on a (not necessarily simple) cycle of length q.

We warm up with the case $\ell = 1$ and show that there is a CQ-definable insertion with bridge bound 2. For this, we consider the insertion formula $\psi_1(x, y) = \chi_2(x) \wedge \chi_3(y)$ and the graph G that consists of three disjoint cycles, two of length 2 and one of length 3. Let u, v be two nodes from the two different cycles of length 2. It is easy to see that ψ_1 introduces an edge from every 2-cycle node to every 3-cycle node and thus produces an undirected path with 2 bridges from u and v, but there is no such path with fewer bridges.

Now we generalise this approach for arbitrary $\ell > 1$. Let $Q = \{q_1, \ldots, q_{2\ell}\}$ be a set of 2ℓ prime number $q_1 < \cdots < q_{2\ell}$ such that $q_{2\ell} < 2q_1$. Such a set Q exists by the Prime Number Theorem (see for example Selberg [1949]). For every $i \leq \ell$, let $\psi_i(x, y)$ be the CQ $\chi_{q_{2i-1}}(x) \wedge \chi_{q_{2i}}(y)$. That is, an edge (u, v) is inserted due to ψ_i whenever u is part of a (not necessarily simple) cycle of length q_{2i-1} and v is part of a (not necessarily simple)

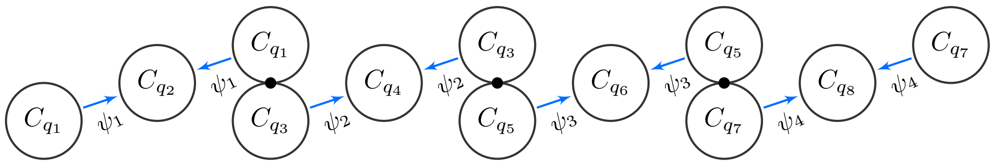


Figure 5.1: Illustration of the graph from the proof of Proposition 5.5(b) for $\ell = 4$. Each C_{q_i} depicts a cycle of length q_i. Bridges are highlighted in blue and annotated by the conjunct used for their insertion.

cycle of length q_{2i}. Furthermore, for every prime q, let C_q be a cycle graph with q nodes.

We now show that the insertion formula $\varphi(x,y) = \bigvee_{i=1}^{\ell} \exists \bar{z} \psi_i(x,y)$ constitutes an insertion query with undirected bridge bound 2ℓ. To this end, for numbers m_1, m_2, let D_{m_1,m_2} be a graph consisting of a cycle of length m_1 and a cycle of length m_2, which have one node in common. We consider the graph G that is a disjoint union of

- $\ell + 2$ cycles $C_{q_1}, C_{q_2}, C_{q_4} \ldots, C_{q_{2\ell-2}}, C_{q_{2\ell-1}}, C_{q_{2\ell}}$, and
- $\ell - 1$ graphs $D_{q_1,q_3}, D_{q_3,q_5}, \ldots, D_{q_{2\ell-3},q_{2\ell-1}}$.

Let u be some node from C_{q_1} and v some node from $C_{q_{2\ell-1}}$. It is easy to see that the application of φ yields an undirected path from u to v through the subgraphs $C_{q_2}, D_{q_1,q_3}, C_{q_4}, D_{q_3,q_5}, C_{q_6}, \ldots, D_{q_{2\ell-3},q_{2\ell-1}}, C_{q_{2\ell}}$, in that order (see Figure 5.1). This path contains 2ℓ bridges.

There is no undirected path with fewer bridges, because edges are only inserted between consecutive subgraphs in the above sequence. This is because $G \models \chi_q(u)$ holds only for a node u that is part of a simple cycle of length q, for some $q \in Q$, as the length of all non-simple cycles in G is larger than any prime number $q_i \in Q$. However, for each $i \in \{1, \ldots, \ell\}$, simple cycles of length q_{2i-1} and q_{2i}, respectively, only occur in consecutive subgraphs in the above sequence. □

We now proceed to the undirected bridge bounds for $\mathsf{UCQ}^{\neg}$-definable insertions. For the statement we need to extend the notion of a pattern slightly. We view a conjunctive query ψ with negation as a formula of the form $\exists \bar{z}\, \psi_1 \wedge \psi_2$, where ψ_1 is a conjunction of positive atoms and ψ_2 a conjunction of negated atoms. We recall that each variable of a CQ with negation needs to occur in some positive atom. The *positive patterns* of ψ are the patterns of the CQ $\exists \bar{z}\, \psi_1$, that is, the weakly connected components of ψ when only the positive atoms are considered.

Proposition 5.6.

(a) For each $\mathsf{UCQ}^{\neg}$-defined insertion query ρ that is a union of ℓ conjunctive queries with negations with at most k positive patterns each, the undirected bridge bound of ρ is at most $2(k-1)\ell$.

(b) For each $n \in \mathbb{N}$ there is a $\mathsf{CQ}^{\neg}$-defined insertion query ρ with undirected bridge bound $\geq n$.

Proof.

(a) We only need to extend the argument of Proposition 5.5(a) slightly. Towards a contradiction, we now assume that there is a $\mathsf{UCQ}^{\neg}$-definable insertion query ρ with insertion formula $\varphi(\bar{p}; x, y) = \bigvee_{i=1}^{\ell} \exists \bar{z}\, \psi_i(\bar{p}; x, y)$ with at most k positive patterns in each CQ ψ_i, a graph G, a change $\alpha = \rho(\bar{a})$ and two nodes u, v such that $\text{ubd}_{G,\alpha(G)}(u, v) \geq 2(k-1)\ell + 1$, as witnessed by an undirected path π in $\alpha(G)$ from u to v with at least $2(k-1)\ell + 1$ bridges, and the absence of a path with fewer bridges.

By the pigeonhole principle, there is a $\mathsf{CQ}^{\neg}$ ψ_i that inserts at least $2k-1$ (not necessarily consecutive) bridges $(u_1, v_1), \ldots, (u_{2k-1}, v_{2k-1})$ of π. Let u_1 be the first of these nodes in π and let η be an assignment that witnesses $G \models \exists \bar{z}\, \psi_i(\bar{a}, u_1, v_1)$. In particular, all positive patterns of ψ_i are satisfied under η. To satisfy the at most $k-2$ positive patterns of ψ_i which do not contain x or y, the variables of $\bar{z}$ that appear in these patterns are mapped into at most $k-2$ weakly connected components $C_1, \ldots, C_{k-2}$ of G.

If some weakly connected component of G would contain more than two of the nodes $v_3, \ldots, v_{2k-1}$, π could be shortcut yielding a path from u to v with fewer bridges. Likewise, none of them can be from the components of u_1 or v_1. Therefore, at least one of the nodes $v_3, \ldots, v_{2k-1}$ must be from a weakly component component other than $C_1, \ldots, C_{k-2}$. Let v_j be such a node and let η' be an assignment that witnesses $G \models \exists \bar{z}\, \psi_i(\bar{a}, u_j, v_j)$. Since η' maps the variables of θ_i^y, the positive pattern of ψ_i that contains y, into a component of G that is not in the range of η, the valuation η'' that coincides with η' on the variables of θ_i^y and with η everywhere else witnesses $G \models \psi_i(\bar{a}, u_1, v_j)$. Note that we can be sure that for all negated atoms of the form $\neg E(z_1, z_2)$, where z_1 is from θ_i^y and z_2 from some other positive pattern of ψ_i, there is no edge from $\eta''(z_1)$ to $\eta''(z_2)$, since these nodes are in different weakly connected components of G (and likewise for atoms $\neg E(z_2, z_1)$). It follows that there is a shortcut edge from u_1 to v_j and consequently an undirected path from u to v with fewer bridges than π, the desired contradiction.

(b) Let $n \in \mathbb{N}$ be some natural number. Our approach is to construct a $\mathsf{CQ}^{\neg}$-definable insertion query ρ as well as a graph G consisting of connected components $D_0, \ldots, D_n$, each D_i with a distinguished node v_i, such that the application of ρ inserts exactly the edges $(v_0, v_1), \ldots (v_{n-1}, v_n)$. Then the undirected bridge distance between v_0 and v_n will be n.

The nodes v_i will be distinguished by having an edge and a path of length 2 to some other node. The remaining challenge is to construct the insertion query and the components of G such that no edge between components D_i and D_j is inserted if $|i - j| > 1$.

Let $Q = \{q(i,j) \mid 0 \leq i < j \leq n, j - i > 1\}$ be a set of distinct prime numbers $q(0,2) < q(0,3) < \cdots < q(n-2, n)$ such that $q(n-2, n) < 2q(0,2)$. Such a set Q exists by the Prime Number Theorem (see Selberg [1949]). For each $i \in \{0, \ldots, n\}$, let Q_i be the set $Q_i \overset{\text{def}}{=} \{q(i,j) \mid q(i,j) \in Q, i < j \leq n\} \cup \{q(j,i) \mid q(j,i) \in Q, 0 \leq j < i\}$.

For a finite set N of numbers, let D_N be a graph with one distinguished node v (so, v has an edge and a path of length 2 to some node v'), and one directed cycle of length q, for each $q \in N$, all of them containing the node v and otherwise being disjoint. Finally, the graph G is the disjoint union of graphs $D_{Q_0}, \ldots, D_{Q_n}$, where in each D_{Q_i} the distinguished node is denoted by v_i (see Figure 5.2).

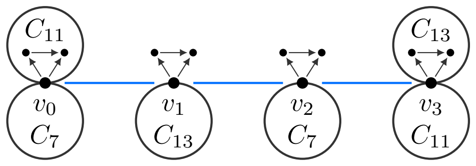


Figure 5.2: Illustration of the graph from the proof of Proposition 5.6(b) for $n = 3$ with $Q = \{7, 11, 13\}$, $q(0, 2) = 7$, $q(0, 3) = 11$, and $q(1, 3) = 13$. Each C_i depicts a cycle of length i and bridges are highlighted in blue. As an example, the edge (v_0, v_2) is not inserted because all cycles of length 7 contain either v_0 or v_2.

Now we construct a $\mathsf{CQ}^\neg$-definable insertion query that inserts exactly the edges $(v_0, v_1), \dots, (v_{n-1}, v_n)$ (and the edges in reverse direction) into G. To this end, it inserts an edge (u, v) if (1) u and v have each have an edge and a path of length 2 to some other node, and (2) for each $q \in Q$ there is some (not necessarily simple) cycle of length q in G which avoids u and v. For each k, let

$$\chi_k(x, y, x_1, \dots, x_k) \stackrel{\text{def}}{=} \bigwedge_{i=1}^{k-1} E(x_i, x_{i+1}) \wedge E(x_k, x_1) \wedge \bigwedge_{i=1}^{k} (\neg E(x, x_i) \wedge \neg E(y, x_i)).$$

That is, χ_k expresses that the variables $x_1, \dots, x_k$ are mapped to a (not necessarily simple) cycle of length k and that x and y do not have an edge to any node of that cycle, so in particular they are not part of that cycle. Finally, the insertion query ρ is induced by the $\mathsf{CQ}^\neg$ formula

$$\varphi(x, y) \stackrel{\text{def}}{=} \exists z_1, z_2, z_3, z_4\; E(x, z_1) \wedge E(z_1, z_2) \wedge E(x, z_2) \wedge E(y, z_3) \wedge E(z_3, z_4) \wedge E(y, z_4) \wedge \bigwedge_{q \in Q} \exists x_1, \dots, x_q\; \chi_q(x, y, x_1, \dots, x_q).$$

We claim that the undirected bridge distance between v_0 and v_n in G is n with respect to the insertion ρ. It is at most n since the edges $(v_0, v), \dots (v_{n-1}, v_n)$ are inserted by ρ. On the other hand, no edge between components D_i and D_j is inserted if $|i - j| > 1$. To see this, we observe first that φ only allows to insert edges between distinguished nodes v_i and v_j due to the required edge and path of length 2 to some other node. Towards a contradiction, let us assume that ρ inserts an edge between nodes v_i and v_j with $j - i > 1$. Let $q = q(i, j)$. Simple cycles of length q only occur in D_i and D_j and can thus not be used to satisfy $\exists x_1, \dots, x_q\; \chi_q(x, y, x_1, \dots, x_q)$. All non-simple cycles in G have length larger than q by choice of Q and construction of G, and can therefore also not be used to satisfy $\exists x_1, \dots, x_q\; \chi_q(x, y, x_1, \dots, x_q)$. So, $\exists x_1, \dots, x_q\; \chi_q(x, y, x_1, \dots, x_q)$ is not satisfied, the desired contradiction. □

With the bridge bounds of Proposition 5.5(a) and Proposition 5.6(a) the dynamic program constructed in the proof of Theorem 5.4 becomes actually usable. Indeed, Statement (a) of Proposition 5.5 is the basis for our implementation and experiments that are described in Chapter 7.

5.2.3 Reachability in acyclic graphs

Now we turn to the other restriction for which DynFO maintainability under complex insertions (and single-edge deletions) is preserved: directed reachability on directed, acyclic graphs. However, we are only able to show this result for quantifier-free insertions. In [Patnaik and Immerman 1997, Theorem 4.2], edge insertions are only allowed if they do not add cycles. Of course, given the transitive closure of the current edge relation it can be easily checked by a first-order formula (a *guard*) whether a new edge closes a cycle. We will see that this is also possible for the complex insertions we consider.

As in Subsection 5.2.1, we begin by stating a bounded bridge property. However, we have to modify it a bit to cope with the fact that insertion queries can introduce cycles into acyclic graphs. The *directed bridge distance* bd is defined as the undirected bridge distance ubd, but with respect to directed paths. Let G' be a graph that is obtained from a graph G by inserting edges. For two nodes u, v of G', $\text{bd}_{G,G'}(u, v)$ is the minimal number d such that there is a *directed* path from u to v in G' that uses d edges that are not in G.

We say that an insertion query ρ has the *bounded bridge property* on directed acyclic graphs if there is a constant c such that, for every graph G' resulting from a directed acyclic graph G by applying ρ, G' either contains a directed cycle with at most c bridges, or G' is acyclic and $\text{bd}_{G,G'}(u, v) \leq c$ for all nodes u, v of G'. We call the smallest such c the *directed bridge bound* of ρ on acyclic graphs.

Proposition 5.7. *Every quantifier-free insertion query has the bounded bridge property on directed acyclic graphs.*

Proof. We show that each quantifier-free insertion operation ρ with underlying formula $\mu(\bar{p}; \bar{x})$ has the bounded bridge property on acyclic graphs. Let ℓ be the length of the parameter tuple of ρ and c' the number of $\mathsf{FO}[0, \ell + 1]$-types of graphs. We will bound the number of bridges on paths created by ρ by $c \overset{\text{def}}{=} c' + 1$.

Let G be a directed, acyclic graph, let $\alpha = \rho(\bar{a})$ be a change, and let u, v be nodes of G. As in the proof of Proposition 5.3, we show that each path π from u to v with $q > c'$ bridges can be transformed into a path with fewer bridges, unless a cycle with at most c bridges is introduced.

To this end, let $(u_1, v_1), \dots, (u_q, v_q)$ be the bridges in π. Since q is larger than the number c' of $\mathsf{FO}[0, \ell + 1]$-types, there are i, j with $1 \leq i < j \leq c' + 1$ such that $(v_i, \bar{a})$ and $(v_j, \bar{a})$ have the same $\mathsf{FO}[0, \ell + 1]$-types. We distinguish three cases. In case (1), the edge (v_i, u_i) is in G and thus α introduces a cycle of length 2. In case (2), the edge (v_j, u_i) is in G and, together with the sub-path from u_i to v_j, constitutes a cycle with at most c bridges. In case (3), u_i is neither connected to v_i nor to v_j by an edge. Therefore $(u_i, v_i, \bar{a})$ and $(u_i, v_j, \bar{a})$ have the same $\mathsf{FO}[0, \ell + 2]$-types by Theorem 2.2, and α inserts an edge (u_i, v_j) as well, the desired shortcut. $\square$

This property allows extending the technique for maintaining the transitive closure relation of acyclic graphs under single-tuple changes (see Example 2.9 together with Example 2.8) to quantifier-free insertions. As in the single-edge case, no further auxiliary relations besides the transitive closure relation are needed. In Section 5.3 we show that the transitive closure relation does *not* suffice for maintaining REACH for acyclic graphs under insertions definable with existential quantifiers.

Theorem 5.8. *Let Δ be any finite set of quantifier-free insertion queries. Then* $(\text{REACH}, \Delta \cup \Delta_E)$ *can be maintained in* DynFO *for directed acyclic graphs. Furthermore, for each quantifier-free insertion, there is a first-order guard which checks whether the insertion destroys the acyclicity of the graph.*

Proof. In [Patnaik and Immerman 1997, Theorem 4.2], a dynamic program is given that maintains the transitive closure of acyclic graphs under single-edge changes, using only the transitive closure as auxiliary relation (see Examples 2.9 and 2.8). Thanks to Proposition 5.7, this program can be easily extended. Let ρ be a quantifier-free insertion query with replacement formula $\mu_E(x, y)$, and let c be the directed bridge bound of ρ on acyclic graphs. To update the transitive closure relation Q of the input graph, it suffices to existentially quantify up to c newly inserted edges and check that they connect x and y using already existing paths. The corresponding update formula is thus given by the formula

$$\varphi_\rho^Q(x,y) \overset{\text{def}}{=} Q(x,y) \vee \bigvee_{1 \le c' \le c} \exists u_1 \exists v_1 \cdots \exists u_{c'} \exists v_{c'} \Big(Q(x, u_1) \wedge Q(v_1, u_2) \wedge \cdots \wedge Q(v_{c'}, y) \\ \wedge \mu_E(u_1, v_1) \wedge \cdots \wedge \mu_E(u_{c'}, v_{c'}) \Big).$$

Analogously, a first-order guard formula can be constructed that checks whether a cycle with at most c newly inserted edges is created. □

5.3 Inexpressibility results for Reachability

Until now we have seen that (different variants of) the reachability query can still be maintained under some classes of definable insertions and single-tuple deletions. However, we have not seen so far a result that the reachability query can be maintained under definable insertions *and* definable deletions, even for restricted graph classes. On the other hand we have no proof that REACH cannot be maintained in DynFO under all FO-definable changes.

We provide preliminary results from two directions. First, in Subsection 5.3.1, we prove that REACH cannot be maintained in DynFO under complex changes with arity-restricted auxiliary relations. Then, in Subsection 5.3.2, we show that reachability cannot be maintained in DynProp under changes defined by quantifier-free formulas.

5.3.1 Inexpressibility of Reachability with Restricted Auxiliary Relations

We show that, in the presence of complex changes, the transitive closure relation as the only binary auxiliary relation does not suffice to maintain reachability. For this result to hold, it suffices to allow single-tuple insertions and one complex change query, which can be chosen either as an insertion or a deletion query. This change query can be restricted in several ways and in most cases the result even holds for acyclic graphs. These results should be contrasted with Theorem 5.8. By $\exists^*\mathsf{FO}$ we denote the existential fragment of first-order logic, that is, the set of first-order formulas in prenex normal form with only existential quantification.

Theorem 5.9.

(a) There is an insertion query ρ_1 such that $(\textsc{Reach}, \{\textsc{ins}_E, \rho_1\})$ *cannot be maintained in* DynFO*, if all auxiliary relations besides the query relation are unary. The insertion query ρ_1 can be chosen as quantifier-free and parameter-free.*
The result even holds on acyclic graphs, in which case ρ_1 can be chosen as $\exists^*\mathsf{FO}$*-definable and parameter-free.*

(b) There is a deletion query ρ_2 such that $(\textsc{Reach}, \{\textsc{ins}_E, \rho_2\})$ *cannot be maintained in* DynFO*, if all auxiliary relations besides the query relation are unary. The deletion query ρ_2 can be chosen as quantifier-free and parameter-free.*
The result also holds on acyclic graphs, even with a quantifier-free ρ_2.

Theorem 5.9(a) even holds in the case where the nodes of G are linearly ordered, but the proof becomes considerably more involved. It can be found in [Schwentick et al. 2017b, Theorem 10].[4]

The proof of Theorem 5.9 makes use of suitable static lower bounds. This approach has often been used before and was made precise by Zeume [2015].

We say that a k-ary query q is expressed by a formula $\varphi(\bar{x})$ with *help relations of schema τ*, if for every structure $\mathcal{D}$ there is a τ-structure H over the same domain such that for every k-tuple $\bar{a}$ it holds: $\bar{a} \in q(\mathcal{D})$ if and only if $(\mathcal{D}, H) \models \varphi(\bar{a})$. This notion should not be confused with definability of the query q in existential second-order logic. In the latter case, the relations can be chosen depending on $\bar{a}$, but here the relations need to "work" for all tuples $\bar{a}$.

The proof of Theorem 5.9 relies on the fact that unary help relations do not suffice to express the transitive closure for path graphs, i.e., graphs consisting of just one path, and for unions of paths even if a certain special binary relation is present. Before stating this formally, we define the latter class more precisely. For $n, m \in \mathbb{N}$, let $G_{n,m} = (V_{n,m}, E_{n,m}, \text{pos})$ where $(V_{n,m}, E_{n,m})$ is the disjoint union of n paths of m nodes each, that is, $V_{n,m} = \{v_{i,j} \mid i \in \{1, \dots, n\}, j \in \{1, \dots, m\}\}$, and $E_{n,m} = \{(v_{i,j}, v_{i,j+1}) \mid i \in \{1, \dots, n\}, j \in \{1, \dots, m-1\}$. The binary relation pos contains tuples $(v_{i,j}, v_{i',j'})$ with $j < j'$, that is, $v_{i,j}$ is at a smaller position than $v_{i',j'}$ in its path. We denote the class of all structures of the form $G_{n,m}$, for arbitrary $n, m \in \mathbb{N}$, by $\mathcal{G}_{\text{up}}$.

Lemma 5.10. *The query* $\textsc{Reach}$ *cannot be expressed by a first-order formula with unary help relations on (a) path graphs and (b) structures from $\mathcal{G}_{up}$.*

Statement (a) follows from an easy locality-based argument and was proved in [Zeume 2015, Lemma 4.3.2]. Statement (b) follows from a standard Ehrenfeucht-Fraïssé argument.

[4]We strongly conjecture that Theorem 5.9(b) holds in the case with a linear order as well.

Proof (of Theorem 5.9). In order to prove (a), let us assume, towards a contradiction, that for every quantifier-free and parameter-free insertion query ρ_1, $(\textsc{Reach}, \{\textsc{ins}_E, \rho_1\})$ can be maintained in DynFO on directed graphs with k unary auxiliary relations $B_1, \ldots, B_k$.

Our goal is to show that then the transitive closure of path graphs *could* be expressed by a first-order formula with unary help relations $B_1, \ldots, B_k$ and C_0, C_1, C_2.

We refer to Figure 5.3 for an illustration of the following. Let G be some arbitrary path graph. Let C_0, C_1, C_2 be unary relations as indicated in Figure 5.3: the relation C_i contains all nodes whose position in the path is i modulo 3. Let G_1 be derived from G by reversing all edges from C_1-nodes to C_2-nodes and by adding loops to all C_1-nodes and C_2-nodes. Let G_2 be the result of applying the insertion query $\rho_1 = \mu_E(x, y;) \overset{\text{def}}{=} E(x, y) \vee (E(x, x) \wedge E(y, y) \wedge E(y, x))$ to G_1. Intuitively, ρ_1 adds all edges (x, y) for which there is an edge (y, x) and both x and y have self-loops.

By our assumption, $\textsc{Reach}(G_2)$ can be defined by a first-order formula ψ_2 using $\textsc{Reach}(G_1)$ and unary auxiliary relations $B_1, \ldots, B_k$. We next show why this implies that the transitive closure of path graphs G can be expressed by a first-order formula with $B_1, \ldots, B_k$ and C_0, C_1, C_2 as unary help relations.

First, there is a simple formula φ_1 which defines the edge relation E_1 of G_1 in terms of the edge relation E of G and C_0, C_1, C_2. Since all directed paths in G_1 have length at most 2, $\textsc{Reach}(G_1)$ can be defined by a first-order formula ψ_1 on G_1.

By combining φ_1, ψ_1, ρ_1, and ψ_2 in a suitable way it is straightforward to construct a first-order formula $\theta(x, y)$ which defines $\textsc{Reach}(G_2)$ on $(G, C_0, C_1, C_2, B_1, \ldots, B_k)$. It is also straightforward to define $\textsc{Reach}(G)$ from $\textsc{Reach}(G_2)$ in a first-order fashion, when C_0, C_1, C_2 and $\textsc{Reach}(G_2)$ are given (basically ignore all pairs (u, v), for which there is an edge (v, u) in G).

Altogether, we can conclude that the transitive closure query can be defined on G with the help of unary help relations $C_0, C_1, C_2, B_1, \ldots, B_k$, the desired contradiction.

We observe that both graphs G_1 and G_2 in the above proof are not acyclic. Yet a slight modification of the construction yields acyclic graphs G'_1 and G'_2. However, it uses existential quantifiers in the definition of the change operation. The graphs are also depicted in Figure 5.3. The graph G'_2 is obtained by applying the operation $\rho'_1 = \mu_E(x, y;) \overset{\text{def}}{=} E(x, y) \vee (\exists z(E(x, z) \wedge E(y, z)) \wedge \exists z' E(z', x))$ to G'_1. The proof is now analogous to (a) except that G'_1 has to be first-order interpreted into the path graph G, as it uses a slightly larger domain. The rest of the argument for acyclic graphs is analogous.

For (b), a similar approach is used, this time starting from a structure $G'' = G_{n,m}$ from $\mathcal{G}_{\text{up}}$. The construction is illustrated in Figure 5.4. It only involves acyclic graphs. Let G''_1 be a graph derived from G'' by adding $n(m-1)$ nodes $\{w_{i,j} \mid i \in \{1, \ldots, n\}, j \in \{1, \ldots, m-1\}\}$ that are used to connect the paths: for each $i, i' \in \{1, \ldots, n\}$ and each $j \in \{1, \ldots, m-1\}$, the edges $(v_{i,j}, w_{i',j})$ and $(w_{i',j}, v_{i,j+1})$ are added. Notice that in G''_1 there is a path from a node $v_{i,j}$ to a node $v_{i',j'}$ with $(i, j) \neq (i', j')$ if and only if $j < j'$. Additionally, we add a node u and edges $(w_{i,j}, u)$ for all $i \in \{1, \ldots, n\}, j \in \{1, \ldots, m-1\}$. This node will be used as the parameter for the deletion. Let G''_2 be the result of applying the deletion query $\rho_2(p) = \mu_E(p; x, y;) \overset{\text{def}}{=} E(x, y) \wedge \neg E(x, p) \wedge \neg E(y, p) \wedge y \neq p$ with parameter u, which deletes all edges that are incident to a node that has an edge to u,

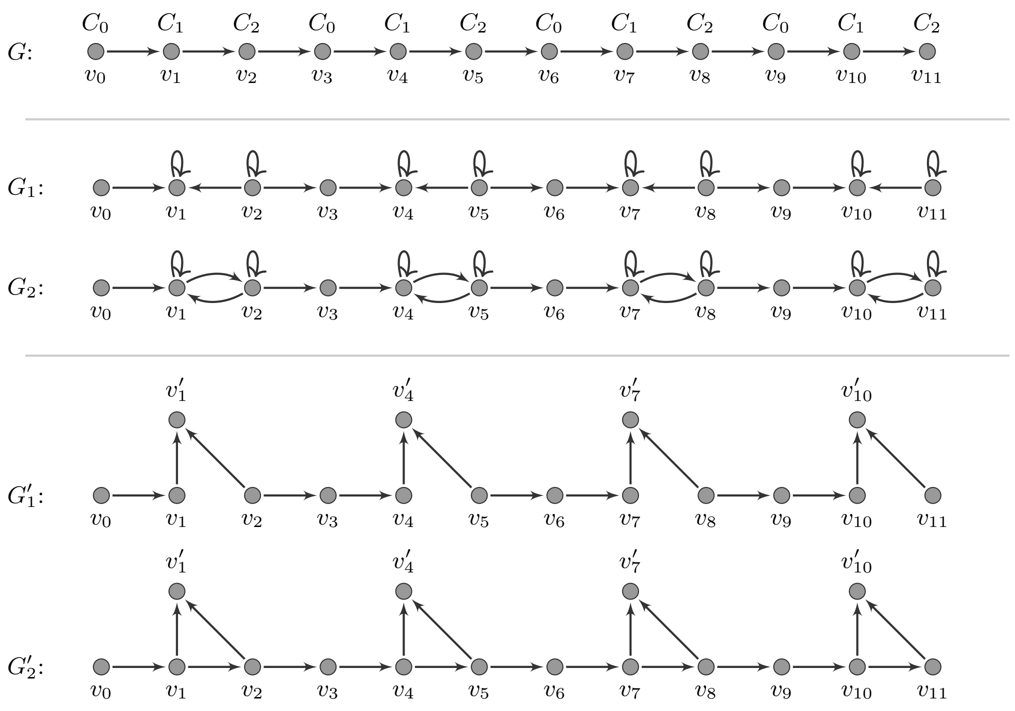

G:
G_1:
G_2:
G'_1:
G'_2:
C_0 C_1 C_2 C_0 C_1 C_2 C_0 C_1 C_2 C_0 C_1 C_2
v_0 v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11}
v'_1 v'_4 v'_7 v'_{10}

Figure 5.3: The graphs from the proof of Theorem 5.9(a).

as well as all edges to u.

The argument is now analogous to the previous proof. To this end we observe that the graph G''_1 can be first-order interpreted into G'' using pos[5]. Also, Reach(G''_1) can be expressed by a first-order formula with the help of pos. Hence the combination of the interpretation formulas and an assumed formula for updating the transitive closure yields a formula expressing the transitive closure on $\mathcal{G}_{\text{up}}$. This proves the stated result for acyclic graphs.

In order to show that the result even holds for quantifier-free *and* parameter-free deletion queries on general graphs, the nodes $w_{i,j}$ can be equipped with self-loops, and the construction hence does not need the parameter node u any more. □

5.3.2 Inexpressibility of Reachability in the Quantifier-Free Fragment

Although we believe that Reach cannot be maintained in DynProp using only quantifier-free update formulas even under single-edge changes, we have so far no proof for this conjecture. It is only known that binary auxiliary relations are not sufficient in this setting [Zeume and Schwentick 2015]. There the authors also show that certain restricted classes of initialisation procedures cannot be used to maintain Reach whenever arbitrary graphs are allowed as initial databases.

[5]Because of this step, G''_1 contains $n(m-1)$ nodes of the form $w_{i,j}$. All other steps would also work using $m-1$ nodes of the form w_j.

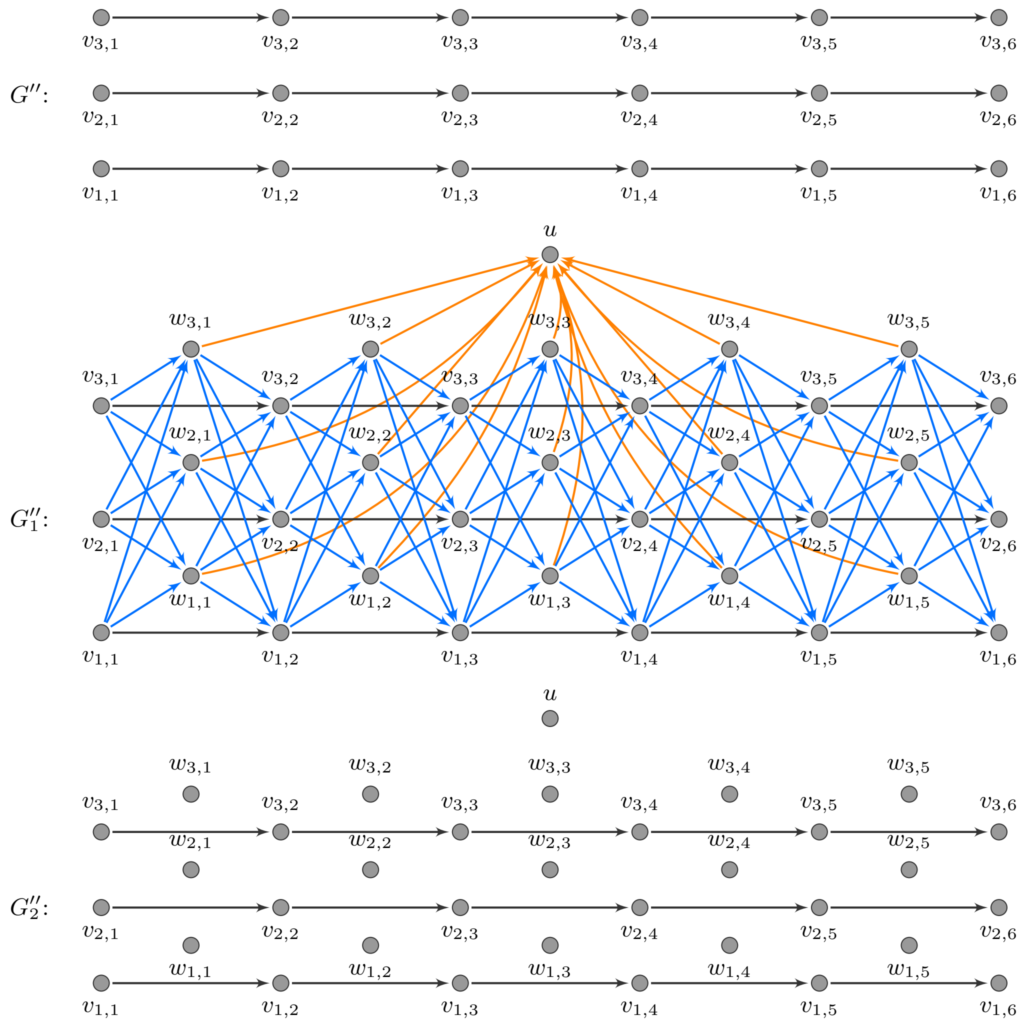


Figure 5.4: The graphs from the proof of Theorem 5.9(b) (for $n = 3$ and $m = 6$)

Here we provide a general unmaintainability result for REACH with quantifier-free update formulas in the presence of definable changes. More precisely, we give simple quantifier-free change queries (a deletion or two insertions), such that REACH and even UREACH cannot be maintained in DynProp under these changes.

Theorem 5.11.

(a) *There is a quantifier-free deletion query ρ such that* (REACH, $\{\text{INS}_E, \rho\}$) *cannot be maintained in* DynProp, *even with arbitrary initialisation.*

(b) *There are quantifier-free insertion queries ρ_1, ρ_2 such that* (REACH, $\{\text{INS}_E, \rho_1, \rho_2\}$) *cannot be maintained in* DynProp, *even with arbitrary initialisation.*

The results also hold for the reachability query REACH *over acyclic graphs and for the undirected reachability query* UREACH.

For the proof of Theorem 5.11 we lift the proof technique for DynProp inexpressibility results formalised by Lemma 4.18 from single-tuple changes to definable changes. The only difference in the statements of Lemma 4.18 and the following Lemma 5.12 is the class of change operations we consider.

Lemma 5.12. *Let q be an m-ary σ_{in}-query and let Δ be a set of quantifier-free replacement queries that includes all single-tuple insertions $\textsc{ins}_R$ for $R \in \sigma_{in}$. Then (q, Δ) is not in k-ary* DynProp*, even with arbitrary initialisation, if there are natural numbers ℓ, r and, for each $n_0 \in \mathbb{N}$, a natural number $n \geq n_0$, such that for all subsets $\mathcal{B} \subseteq \binom{[n]}{k+1}$ there exist*

- *a σ_{in}-structure $\mathcal{D}$, a set $P = \{p_1^1, \ldots, p_1^\ell, \ldots, p_n^1, \ldots, p_n^\ell\}$ of distinct elements, and tuples $\bar{a} = a_1, \ldots, a_m$ and $\bar{u} = u_1, \ldots, u_r$, such that*
  - *P and $\{a_1, \ldots, a_m, u_1, \ldots, u_r\}$ are disjoint and contained in the domain of $\mathcal{D}$,*
  - *$(\mathcal{D}, \bar{a}, \bar{u}, \bar{p}_{i_1}, \ldots, \bar{p}_{i_{k+1}}) \equiv_0 (\mathcal{D}, \bar{a}, \bar{u}, \bar{p}_{j_1}, \ldots, \bar{p}_{j_{k+1}})$ for all strictly increasing sequences $i_1, \ldots, i_{k+1}$ and $j_1, \ldots, j_{k+1}$ over $[n]$, where $\bar{p}_i = (p_i^1, \ldots, p_i^\ell)$, and*
- *a sequence $\beta(x_1^1, \ldots, x_1^\ell, \ldots, x_{k+1}^1, \ldots, x_{k+1}^\ell, y_1, \ldots, y_r) = \alpha_1(\bar{x}_{i_1}, \bar{y}) \cdots \alpha_s(\bar{x}_{i_s}, \bar{y})$ of Δ-changes, where no change α_j uses variables $x_{i_j}^t$ and $x_{i_{j'}}^{t'}$ with $i_j \neq i_{j'}$ as parameters,*

such that for all strictly increasing sequences $i_1, \ldots, i_{k+1}$ over $[n]$ it holds that

$$\bar{a} \in q(\beta(\bar{p}_{i_1}, \ldots, \bar{p}_{i_{k+1}}, \bar{u})(\mathcal{D})) \quad \iff \quad \{i_1, \ldots, i_{k+1}\} \in \mathcal{B}.$$

Proof idea. The proof of Lemma 4.18 uses only once that the change operations are limited to single-tuple changes: when it applies Lemma 4.17, the Substructure Lemma from [Zeume and Schwentick 2015]. The proof of the Substructure Lemma for single-tuple changes as presented in [Zeume and Schwentick 2015] immediately carries over to change operations definable by quantifier-free first-order formulas. Then Lemma 5.12 is a simple corollary of Lemma 4.18. □

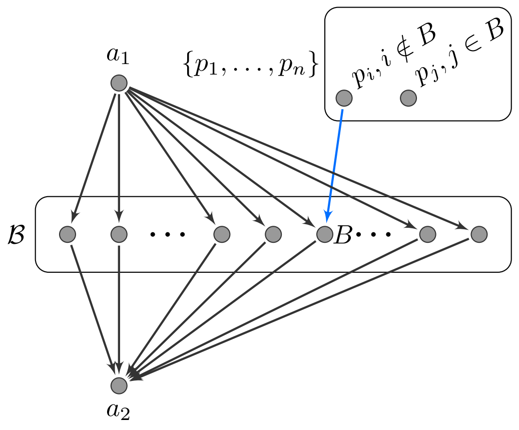
Now we can prove Theorem 5.11.

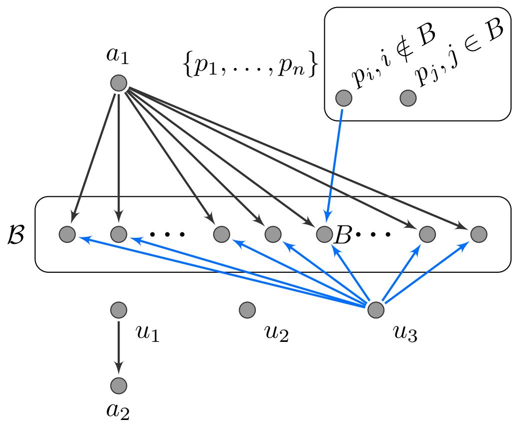
Proof (of Theorem 5.11).

(a) We apply Lemma 5.12 and prove that for some quantifier-free deletion query ρ the dynamic query $(\textsc{Reach}, \{\textsc{ins}_E, \rho\})$ is not in k-ary DynProp with arbitrary initialisation, for any k.

The idea for the construction of $\mathcal{D}$ and β, given some collection $\mathcal{B} \subseteq \binom{[n]}{k+1}$, is to start from a graph consisting of paths of length 2 of the form a_1, B, a_2, for every $B \in \mathcal{B}$. The sequence β induced by a sequence $i_1, \ldots, i_{k+1}$ deletes all edges (Y, a_2) for which some i_j is not in Y. The only path from a_1 to a_2 that can possibly remain is the path $a_1, \{i_1, \ldots, i_{k+1}\}, a_2$, which is present from the start if and only if $\{i_1, \ldots, i_{k+1}\} \in \mathcal{B}$, just as required by Lemma 5.12.

We make this more precise now. Let $\rho(p)$ be the quantifier-free deletion defined as $\rho(p) = \mu(p; x, y) \stackrel{\text{def}}{=} E(x, y) \land \neg E(p, x)$. So, $\rho(p)$ deletes an edge (x, y) if there is an edge (p, x); this is because an edge (x, y) is *not* deleted only if *no* edge (p, x) exists.





(a) The graph for the proof of Theorem 5.11(a). (b) The graph for the proof of Theorem 5.11(b).

Figure 5.5: The graphs from the proof of Theorem 5.11. Edges used for controlling the changes are highlighted in blue.

Towards a contradiction, let us assume that Reach can be maintained in k-ary DynProp under changes from $\{\text{INS}_E, \rho\}$ for some k. We choose $m = 2, \ell = 1$ and $r = 0$ in the statement of Lemma 5.12. Let $n \geq k+1$ and $\mathcal{B} \subseteq \binom{[n]}{k+1}$ be arbitrary.

The structure $\mathcal{D}$ that we construct (see Figure 5.5(a) for an illustration) is a graph with node set $\{p_1, \dots, p_n\} \cup \{a_1, a_2\} \cup \mathcal{B}$. The graph has the following edges:

- For each $B \in \mathcal{B}$ there are edges (a_1, B) and (B, a_2).
- For each $i \in [n]$ and for each $B \in \mathcal{B}$ there is an edge (p_i, B) if $i \notin B$.

Since there are no edges between vertices from $\{a_1, a_2, p_1, \dots, p_n\}$, the requirements of Lemma 5.12 on $\mathcal{D}$ are fulfilled.

Let $\beta(x_1, \dots, x_{k+1})$ be the change sequence $\rho(x_1) \cdots \rho(x_{k+1})$. For a strictly increasing sequence $i_1, \dots, i_{k+1}$ over $[n]$, let $Y = \{i_1, \dots, i_{k+1}\}$ and let β_Y be the change sequence $\beta(p_{i_1}, \dots, p_{i_{k+1}})$. This change sequence removes all edges (X, a_2) with $i_j \notin X$, for some $j \leq k+1$. Therefore, the only edge of the form (X, a_2) that might remain after applying β_Y is (Y, a_2) which only exists if $Y \in \mathcal{B}$. Since there is a path from a_1 to a_2 in $\beta_Y(\mathcal{D})$ if and only if such an edge remains, we conclude that $\bar{a} \in \text{Reach}(\beta_Y(\mathcal{D}))$ if and only if $Y = \{i_1, \dots, i_{k+1}\} \in \mathcal{B}$. By Lemma 5.12 we conclude that $(\text{Reach}, \{\text{INS}_E, \rho\})$ cannot be maintained in k-ary DynProp, the desired contradiction.

(b) The proof is very similar to the proof of part (a). Now, in the structure $\mathcal{D}$ that we construct, the possible paths from a node a_1 to a node a_2 are of the form a_1, B, u_1, a_2, for some other node u_1 and some $B \in \mathcal{B}$. However, initially there are no edges of the form (B, u_1). The first $k+1$ changes of the to-be-constructed change sequence β, induced by a sequence $i_1, \dots, i_{k+1}$, will insert edges (Y, u_2) for some particular node u_2 if some i_j is not in Y. Therefore, after these changes there is at most one node Y that is *not* connected to u_2, namely $Y = \{i_1, \dots, i_{k+1}\}$, if this node exists. A last change inserts an edge (B, u_1) for all B that are not connected to u_2. Such a node exists, and consequently there will again be a path from a_1 to a_2, if and only if $\{i_1, \ldots, i_{k+1}\} \in \mathcal{B}$. Lemma 5.12 then says that no k-ary DynProp program can maintain the query result.

We choose the insertion queries ρ_1 and ρ_2 as follows.

- $\rho_1(p, p') = \mu_1(p, p'; x, y) \overset{\text{def}}{=} E(x, y) \lor \big(y = p \land E(p', x)\big)$ and
- $\rho_2(p, p', p'') = \mu_2(p, p', p''; x, y) \overset{\text{def}}{=} E(x, y) \lor \big(y = p \land E(p', x) \land \neg E(x, p'')\big)$.

The query ρ_1 inserts edges (x, p) for all x with an edge (p', x), whereas ρ_2 inserts edges (x, p) for all x with an edge (p', x) but *without* an edge (x, p'').

Towards a contradiction, let us assume that $(\textsc{Reach}, \{\textsc{ins}_E, \rho_1, \rho_2\})$ can be maintained in k-ary DynProp, for some k. Again, we aim to use Lemma 5.12 and we let $m = 2, \ell = 1$ and $r = 3$ in the statement of the lemma. Let $n \geq k + 1$ and $\mathcal{B} \subseteq \binom{[n]}{k+1}$ be arbitrary.

The database $\mathcal{D}$ is a graph with node set $\{p_1, \ldots, p_n\} \cup \{a_1, a_2, u_1, u_2, u_3\} \cup \mathcal{B}$ and the following edges:

- For each $B \in \mathcal{B}$ there are edges (a_1, B) and (u_3, B).
- For each $i \in [n]$ and for each $B \in \mathcal{B}$ there is an edge (p_i, B) if $i \notin B$.
- There is an edge (u_1, a_2).

It can be easily verified that the conditions imposed by Lemma 5.12 are satisfied.

Let $\beta(x_1, \ldots, x_{k+1})$ be the change sequence $\rho_1(u_2, x_1) \cdots \rho_1(u_2, x_{k+1})\rho_2(u_1, u_3, u_2)$. For any set $Y = \{i_1, \ldots, i_{k+1}\}$ with $1 \leq i_1 < \cdots < i_{k+1} \leq n$ we define $\beta_Y \overset{\text{def}}{=} \beta(p_{i_1}, \ldots p_{i_{k+1}})$. After the first $k + 1$ changes of β_Y, a node $B \in \mathcal{B}$ is connected to u_2 unless $B = Y$. Therefore, the last change $\rho_2(u_1, u_3, u_2)$ inserts at most one edge, (Y, u_1), but only if $Y \in \mathcal{B}$. Thus, in $\beta_Y(\mathcal{D})$ there is a path from a_1 to a_2 if and only if $Y = \{i_1, \ldots, i_{k+1}\} \in \mathcal{B}$.

An application of Lemma 5.12 again yields the desired contradiction.

An inspection of the proofs shows that the used graphs are acyclic and that both constructions immediately work for undirected reachability. $\square$

We note that Theorem 5.11(b) can also by proved using only one definable insertion ρ instead of two insertions ρ_1, ρ_2. This insertion ρ could use two parameters, in addition to the parameters of ρ_1 and ρ_2, behave as ρ_1 if the parameter values are equal and else behave as ρ_2.

5.4 Maintaining formal languages

Similar to the reachability query for different graph classes, the membership problem for formal languages under changes of single positions is widely studied in the dynamic complexity literature. Already Patnaik and Immerman [1997] observed that in this setting all regular languages can be maintained in DynFO using the BIT predicate; Hesse [2003b] showed that DynFO with unary auxiliary relations suffices. Later, Gelade et al. [2012] proved that the class of regular languages is exactly the class of languages maintainable in DynProp. All context-free languages can be maintained in DynFO [Gelade et al. 2012], some of them and even some non-context-free languages even with only unary auxiliary relations (cf. [Gelade et al. 2012, Proposition 4.2] and [Dong and Su 1997, Example 5]).

This is not possible for all context-free languages [Zeume 2015; Vortmeier 2013].

Numerous researchers studied the question which classes of path queries to graphs can be maintained in DynFO and its subclasses [Weber and Schwentick 2007; Datta et al. 2018a; Zeume 2015; Muñoz et al. 2016; Bouyer and Jugé 2017].

In this section, we show that the membership problem for regular and context-free languages can still be maintained under certain kinds of definable changes. We first introduce some notation.

A word over an alphabet Σ is encoded by an *ordered* structure, that is, a structure $\mathcal{D}$ with a built-in linear order $\leq$ on its domain that is not modified by any change. We use this order to identify the domain with the set $\{1, \dots, n\}$. Intuitively, the positions of the word correspond to the elements of the structure's domain. For every $\sigma \in \Sigma$ the structure has a unary relation R_σ that encodes all positions of the input word that carry the letter σ.

We demand that at any point of time an element of the domain is in at most one relation R_σ. If an element i is in no relation R_σ, then we consider the position i to be labelled with the empty word ϵ. A structure of the presented form with domain $\{1, \dots, n\}$ therefore encodes a word $w = w_1 \cdots w_m$ of length $m \leq n$. As an example, the structure with domain $\{1, 2, 3, 4, 5\}$ and $R_{\mathtt{a}} = \{2, 4\}$ and $R_{\mathtt{b}} = \{1\}$ represents the word baa.

We assume that structures have constants min and max that represent the smallest and the largest element, 1 and n, respectively. This assumption is not necessary for our results but facilitates the proofs: these constants can of course be quantified in FO; for Theorem 5.13, the only result of this section concerning DynProp, the assumption can be avoided by using additional auxiliary relations for prefixes and suffixes, just as in [Gelade et al. 2012].

In the following, we will not distinguish between a structure and the word it represents.

We consider the problem of maintaining (the membership problem for) formal languages under first-order definable change operations. We assume that only replacement queries are used whose application results in structures where each position is in at most one set R_σ. For a given formal language L we denote the membership query for L as Member(L).

Under definable changes we cannot hope for strong maintenance result for DynFO without initialisation. Even the simple regular language $L((\mathtt{aa})^*)$ of words with an even number of as cannot be maintained under a change operation that inserts all positions into the relation $R_{\mathtt{a}}$: a dynamic program would need to determine the parity of the domain, which is not possible in first-order logic. In this section we consequently allow non-empty initialisations of the auxiliary relations and state their complexity along with the results.

As an alternative to initialisations one could restrict the replacement queries in a way that only a constant number of elements may enter the active domain by every change. For example, one could demand that only elements that are parameters to the change might change their label from ϵ to some symbol $\sigma \in \Sigma$. However, we are more interested in maintaining queries under stronger change operations than in optimising the complexity of the initialisation, so we do not consider this alternative approach.

We prove that regular and context-free languages can be maintained dynamically for large classes of change operations: all regular languages can be maintained in DynProp under quantifier-free change operations and all context-free languages can be maintained in DynFO under $\exists^*$FO-definable (and, dually, $\forall^*$FO-definable) change operations. So far we cannot deal in general with definable changes that utilise quantifier alternation. Towards the end of this section we discuss the challenges using an example.

For quantifier-free change operations, the results are obtained by generalisations of the techniques of [Gelade et al. 2012]. The most important observation is that large parts of an input word change rather uniformly under the changes we consider: the label of a position before the change essentially implies the label after the change, so the transformation of the input can be described by a *relabelling function* $f : \Sigma_\epsilon \to \Sigma_\epsilon$, where for an alphabet Σ we denote by Σ_ϵ the set $\Sigma \cup \{\epsilon\}$.

Theorem 5.13. *Let L be a regular language and Δ a finite set of quantifier-free replacement queries. Then* $(\textsc{Member}(\mathrm{L}), \Delta)$ *can be maintained in* DynProp *with* NC^1 *initialisation.*

Proof. Let L be a regular language over alphabet Σ and let $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ a corresponding deterministic finite automaton with set Q of states, transition function δ, initial state s, and set F of accepting states. In [Gelade et al. 2012, Proposition 3.3], the main auxiliary relations are of the form $S_{q,r}(i, j)$, where q, r are states of $\mathcal{A}$ and i, j are positions of the string under consideration. The intended meaning of $S_{q,r}$ is that $(i, j) \in S_{q,r}$ if and only if $\delta^*(q, w_{i+1} \cdots w_{j-1}) = r$, where δ^* is the extension of δ to strings[6]. Notice that w_i and w_j are not relevant for determining whether $(i, j) \in S_{q,r}$.

We show how to extend this approach to replacement queries of the form $\rho(p)$ with *one* parameter p. The general case of more parameters works analogously, but is notationally more involved. A replacement query with one parameter consists of one quantifier free formula $\mu_\sigma(p; x)$ for each symbol $\sigma \in \Sigma$. Each formula $\mu_\sigma(p; x)$ determines whether position x carries σ after the change. Whether this is the case only depends on (1) the current symbol at position x, (2) the current symbol at position p, and (3) on the relative order of x and p. Thus, the impact of a change can be described as follows: some relabelling function $f_\leftarrow$ is applied at all positions $x < p$, some relabelling occurs at position p, and some relabelling function $f_\rightarrow$ is applied at all positions $x > p$.

We generalise the approach of [Gelade et al. 2012] and maintain binary auxiliary relations $S^f_{q,r}$ for each relabelling function f and each pair $q, r \in Q$ of states. The intended meaning is that $(i, j) \in S^f_{q,r}$ if and only if $\delta^*(q, f(w_{i+1} \cdots w_{j-1})) = r$, where f is extended to strings in the straightforward way. Clearly, $S_{q,r} = S^{\mathrm{id}}_{q,r}$. When a change $\rho(i)$ occurs, an update formula $\varphi^f_{q,r}(p; x, y)$ has to select the relabelling functions $f^\tau_\leftarrow$ and $f^\tau_\rightarrow$ as well as the symbol σ^τ that characterise the effect of the change, based on the symbol τ at position i. When some relabelling f' is applied to a subword, whether a relabelling f yields a subword w with $\delta^*(q, w) = r$ is given by the auxiliary relation $S^{f \circ f'}_{q,r}$

[6]The relations $S_{q,r}$ are named $R_{q,r}$ in [Gelade et al. 2012], but we want to avoid confusion with the relations R_σ. Since [Gelade et al. 2012] does not use constants min and max, it uses further auxiliary relations of the form I_r and F_q that contain all positions i with $\delta^*(s, w_1 \cdots w_{i-1}) = r$, and $\delta^*(q, w_{i+1} \cdots w_n) \in F$, respectively.

for the relabelling function $f \circ f'$. So, the quantifier-free update formula $\varphi^f_{q,r}(p; x, y)$ can be chosen as

$$\varphi^f_{q,r}(p;x,y) \overset{\text{def}}{=} x < y \wedge \bigvee_{\tau \in \Sigma_\epsilon} \Big(R_\tau(p) \wedge \Big((p \le x \wedge R^{f\circ f^\tau_\rightarrow}_{q,r}(x,y)) \vee (y \le p \wedge R^{f\circ f^\tau_\leftarrow}_{q,r}(x,y)) \vee$$
$$(x < p < y \wedge \bigvee_{\substack{q',r' \in Q \\ \delta(q',\sigma^\tau)=r'}} (R^{f\circ f^\tau_\leftarrow}_{q,q'}(x,p) \wedge R^{f\circ f^\tau_\rightarrow}_{r',r}(p,y))) \Big) \Big).$$

The (Boolean) query relation Ans can be updated by the formula

$$\varphi(p) \overset{\text{def}}{=} \bigvee_{\substack{q,r \in Q, \sigma,\sigma' \in \Sigma_\epsilon \\ \delta(s,\sigma)=q, \delta(r,\sigma') \in F}} R_\sigma(\min) \wedge \varphi^{\text{id}}_{q,r}(p; \min, \max) \wedge R_{\sigma'}(\max).$$

The initialisation of the relations $S^f_{q,r}$ is straightforward. If $f(\epsilon) = \sigma$ for a relabelling function f, then a pair (i, j) with $i < j$ is in $S^f_{q,r}$ if and only if $\delta^*(q, \sigma^{j-i-1}) = r$, where σ^m is the word of length m that only contains the letter σ. If $f(\epsilon) = \epsilon$, then (i, j) is in $S^f_{q,r}$ if and only if $q = r$. This initialisation can be computed in NC^1, as the membership problem for regular languages is in NC^1, see [Ladner and Fischer 1980]. □

We next turn to context-free languages. Using the ideas of the proof of Theorem 5.13 we can extend the result of [Gelade et al. 2012] that context-free languages can be maintained in DynFO under changes of single positions [Gelade et al. 2012, Theorem 4.1] to quantifier-free changes. In a second step we extend this result to $\exists^*$FO-definable change operations, and dually, $\forall^*$FO-definable change operations.

Proposition 5.14. *Let L be a context-free language and Δ a finite set of quantifier-free replacement queries. Then* (Member(L), Δ) *can be maintained in* DynFO *with* P *initialisation.*

Proof. We adapt the proof of [Gelade et al. 2012, Theorem 4.1], using the same ideas as in the proof of Theorem 5.13. Let L be a context-free language over alphabet Σ and G a corresponding context-free grammar in Chomsky normal form, that is, only with rules of the form $X \to YZ$ and $X \to \sigma$, for non-terminals X, Y, Z and symbols $\sigma \in \Sigma$. We extend G, slightly violating the Chomsky normal form, with a new non-terminal E and a new rule $E \to \epsilon$, as well as with new rules $X \to XE, X \to EX$, for every non-terminal X of G. Clearly, this extension does not change the language generated by G, but simplifies for example the update formulas for deletions, see [Gelade et al. 2012] for details.

The proof from [Gelade et al. 2012] uses (a slight variation of the) auxiliary relations $S_{X,Y}(i_1, j_1, i_2, j_2)$, for each pair X, Y of non-terminals of the grammar G, with the intention that $(i_1, j_1, i_2, j_2) \in S_{X,Y}$ if and only if $i_1 \le j_1 < i_2 \le j_2$ and $X \Rightarrow^* w_{i_1} \cdots w_{j_1} Y w_{i_2} \cdots w_{j_2}$ holds. The latter statement means that from X one can derive the sentential form $w_{i_1} \cdots w_{j_1} Y w_{i_2} \cdots w_{j_2}$ using rules of G. In particular, if $(i_1, j_1, i_2, j_2) \in S_{X,Y}$ and $Y \Rightarrow^* w_{j_1+1} \cdots w_{i_2-1}$, then $X \Rightarrow^* w_{i_1} \cdots w_{j_2}$. If $(\min, i, i+1, \max) \in S_{X,E}$ for some position i, then X generates the input word w.

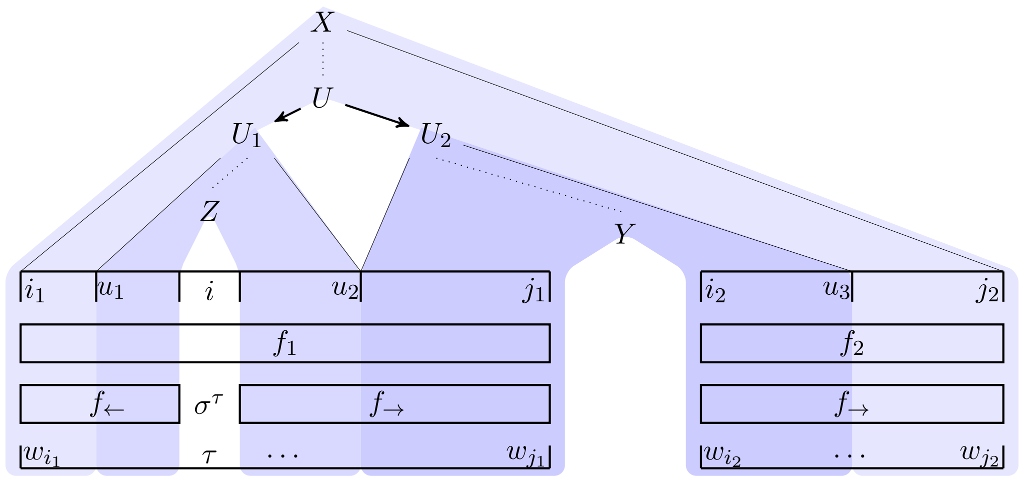


Figure 5.6: Conditions for the update of $S^{f_1,f_2}_{X,Y}$ in the proof of Proposition 5.14 (see also [Gelade et al. 2012, Fig. 2]). The bottom contains the input word string before the change. The layer above depicts the effect of the change, and the next layer shows the relabelling assumed by the auxiliary relation. On top, we see the partial derivation tree checked by the update formula.

We adapt this approach as in the proof of Theorem 5.13 and use auxiliary relations of the form $S^{f_1,f_2}_{X,Y}$ for relabelling functions f_1, f_2. The intention is that $(i_1, j_1, i_2, j_2) \in S^{f_1,f_2}_{X,Y}$ if and only if $i_1 \leq j_1 < i_2 \leq j_2$ and $X \Rightarrow^* f_1(w_{i_1} \cdots w_{j_1}) Y f_2(w_{i_2} \cdots w_{j_2})$.

We show next how to maintain these relations under quantifier-free replacement queries $\rho(p)$ with one parameter p. As in the proof of Theorem 5.13, the generalisation for any number of parameters is straightforward, but tedious.

Let $\alpha = \rho(i)$ be a change to the input word w. Similarly as in the proof of Theorem 5.13, from ρ and the symbol τ at position i one can derive relabelling functions $f_\leftarrow$, $f_\rightarrow$ and a symbol σ^τ that characterise the effect of α on the positions smaller and larger than i, and on the position i itself, respectively.

We explain how an update formula can determine whether a tuple (i_1, j_1, i_2, j_2) shall be included in $S^{f_1,f_2}_{X,Y}$ after a change $\rho(i)$ with $i_1 < i < j_1$. The case $i_2 < i < j_2$ is symmetric, all other cases are also very similar. We also omit several border sub-cases.

The idea behind the update formula is illustrated in Figure 5.6. It essentially "guesses" parts of a derivation tree that witnesses that X can produce the sentential form $f_1(\alpha(w)_{i_1} \cdots \alpha(w)_{j_1})) Y f_2(\alpha(w)_{i_2} \cdots \alpha(w)_{j_2}))$ and checks consistency of that guess using the old auxiliary relations. More formally, it iterates over all rules $U \to U_1 U_2$ of G and all non-terminals Z with a rule $Z \to f_1(\sigma^\tau)$ and checks whether there are positions u_1, u_2, u_3 with $i_1 \leq u_1 < i < u_2 < j_1$ and $i_2 \leq u_3 \leq j_2$ such that

- $(i_1, u_1 - 1, u_3 + 1, j_2) \in S^{f_1 \circ f_\leftarrow, f_2 \circ f_\rightarrow}_{X,U}$,
- $(u_1, i - 1, i + 1, u_2) \in S^{f_1 \circ f_\leftarrow, f_1 \circ f_\rightarrow}_{U_1,Z}$, and
- $(u_2 + 1, j_1, i_2, u_3) \in S^{f_1 \circ f_\rightarrow, f_2 \circ f_\rightarrow}_{U_2,Y}$ hold.

The initialisation of the relations $S^{f_1,f_2}_{X,Y}$ is straightforward. If $f_1(\epsilon) = \sigma_1$ or $f_2(\epsilon) = \sigma_2$ holds for relabelling functions f_1, f_2 and $\sigma_1, \sigma_2 \in \Sigma$, then a tuple (i_1, j_1, i_2, j_2) is in $S^{f_1,f_2}_{X,Y}$ if and only if $X \Rightarrow^* \sigma_1^{j_1-i_1+1} Y \sigma_2^{j_2-i_2+1}$. This can clearly be checked in polynomial time. $\square$

We now lift this result to $\exists^*$FO-definable replacement queries and essentially prove that every $\exists^*$FO-definable replacement query for words can be simulated by a quantifier-free replacement query. Intuitively this means that an $\exists^*$FO-definable replacement query splits the input word into constantly many subwords, and each subword changes with respect to some relabelling function.

Theorem 5.15. *Let L be a context-free language and Δ a finite set of $\exists^*$FO-definable replacement queries. Then* $(\textsc{Member}(\mathrm{L}), \Delta)$ *can be maintained in* DynFO *with* P *initialisation.*

Proof. We show that for each $\exists^*$FO-definable change operation $\rho(\bar{p})$ there exists a quantifier-free change operation $\rho'(\bar{p}, \bar{q})$, using (many) additional parameters $\bar{q}$, such that for each string w and each change $\alpha = \rho(\bar{a})$ there is a change $\alpha' = \rho'(\bar{a}, \bar{c})$ such that $\alpha(w) = \alpha'(w)$. Moreover, we show that the tuple $\bar{c}$ can be determined in FO. As by Proposition 5.14 we can maintain $\textsc{Member}(\mathrm{L})$ under quantifier-free replacement queries, the result follows: an update formula for ρ only has to determine the tuple $\bar{c}$ and then simulate the corresponding update formula for ρ'.

We construct the change operation ρ' from ρ. Let $\mu_\sigma(\bar{p}; x)$ be the replacement formula of ρ for some relation R_σ, that is, the $\exists^*$FO-formula that expresses whether after a change $\rho(\bar{a})$ the element at position x carries the symbol σ. Without loss of generality, $\mu_\sigma(\bar{p}; x) = \bigvee_{i=1}^m \theta_i(\bar{p}; x)$ where each $\theta_i(\bar{p}; x)$ is of the form $\exists y_1 \ldots, y_k\, \psi_i(\bar{y}, \bar{p}; x)$ with quantifier-free ψ_i, and ψ_i describes a full atomic type over $\bar{p}, \bar{y}, x$ with respect to the linear order and the relations R_σ. So, ψ_i is a conjunction of literals such that for any pair z, z' of variables from $\bar{p}, \bar{y}, x$ either the literal $z \leq z'$ or $\neg(z \leq z')$ and either $R_\sigma(z)$ or $\neg R_\sigma(z)$ is present in ψ_i.

Towards the construction of a replacement formula μ'_σ for ρ', we claim that for every i, every string w and every tuple $\bar{a}$ of position there exists a tuple $\bar{c} = (c_1, \ldots, c_k)$ of positions in w such that for every position b it holds $w \models \theta_i(\bar{a}, b)$ if and only if $w \models \psi_i(\bar{a}, \bar{c}; b)$.

Let $y_1 < \cdots y_\ell < x < y_{\ell+1} < \cdots < y_k$ be the order among the quantified variables $\bar{y}$ and x that is enforced by ψ_i. If $w \models \theta_i(\bar{a}, b)$ holds, let positions $(c_1, \ldots, c_\ell)$ be the lexicographically minimal and $(c_k, \ldots, c_{\ell+1})$ the lexicographically maximal positions such that $w \models \psi_i(\bar{a}, \bar{c}, b)$ holds. Intuitively, this assignment for the quantified variables $\bar{y}$ picks the positions that are as far away as possible from the position b. This tuple satisfies the condition of the claim above, that is, for every position b', the statements $w \models \theta_i(\bar{a}, b')$ and $w \models \psi_i(\bar{a}, \bar{c}; b')$ are equivalent. We call $\bar{c}$ the *canonical tuple* for θ_i with respect to w and $\bar{a}$.

As the other direction is trivial, we only argue the direction $w \models \theta_i(\bar{a}, b') \Rightarrow w \models \psi_i(\bar{a}, \bar{c}; b')$. We assume that $w \models \theta_i(\bar{a}, b')$ holds for an arbitrary position b' and let $\bar{c}'$ be a tuple of positions such that $w \models \psi_i(\bar{a}, \bar{c}'; b')$ holds. We need to show that also $w \models \psi_i(\bar{a}, \bar{c}; b')$ holds. Because $w \models \psi_i(\bar{a}, \bar{c}; b)$ holds, we only have to show that all statements $c_j \leq b'$ and $b' \leq c_j$ hold exactly if $c'_j \leq b'$ and $b' \leq c'_j$ hold, respectively. We show that $c_j \leq b'$ holds if and only if $c'_j \leq b'$, the other case is symmetric.

From the order enforced by ψ_i we see that $c'_j \leq b'$ holds if and only if $j \leq \ell$. If $j \leq \ell$, then $c_j \leq c'_j$ by the choice of $\bar{c}$ and therefore also $c_j \leq b'$ holds. If $j > \ell$, then $c_j \geq c'_j$ and $c_j \leq b'$ does not hold, either. This completes the proof of the claim.

Let $\mu'_\sigma(\bar{p}, \bar{q}_1, \ldots, \bar{q}_m; x)$ be the quantifier-free replacement formula $\bigvee_{i=1}^m \psi'_i(\bar{q}_i, \bar{p}; x)$, where ψ'_i results from ψ_i by renaming of the variables $\bar{y}$ to $\bar{q}_i$. It is clear from the observations above that, for each word w and each tuple $\bar{a}$ of positions, $w \models \mu'_\sigma(\bar{a}, \bar{c}_1, \ldots, \bar{c}_m; b)$ if and only if $w \models \mu_\sigma(\bar{a}; b)$, when the tuples c_i are chosen as the canonical tuples for θ_i with respect to w and $\bar{a}$. Note that these tuples can easily be determined in FO, given w and $\bar{a}$.

The change $\rho'(\bar{p}, (\bar{q}_\sigma)_{\sigma \in \Sigma})$ is the composition of all replacement formulas $\mu'_\sigma(\bar{p}, \bar{q}_\sigma; x)$. □

Because every $\forall^*$FO-formula is just the negation of an $\exists^*$FO-formula, the next corollary follows immediately.

Corollary 5.16. *Let L be a context-free language and Δ a finite set of $\forall^*$FO-definable replacement queries. Then* (MEMBER(L), Δ) *can be maintained in* DynFO *with* P *initialisation.*

Proof. This can be proven along the same lines as Theorem 5.15, as every $\forall^*$FO replacement formula can be written in the form $\neg \bigvee_{i=1}^m \theta_i$ with θ_i as in the previous proof. □

Wrapping up the results we have seen so far, we can maintain context-free languages in DynFO under changes that are definable by FO formulas without quantifier alternation. So far we do not know whether regular or context-free languages can be maintained under definable changes that use quantifier alternation. We exemplify the challenges imposed by these changes with the parameter-free replacement query $\rho_{\mathtt{ab}\to\mathtt{aa}}$ that intuitively replaces every subword ab in the input word with aa. It is defined by the replacement formulas $\mu_{R_\mathtt{a}}(x) \stackrel{\text{def}}{=} R_\mathtt{a}(x) \vee \big(R_\mathtt{b}(x) \wedge R_\mathtt{a}(x-1)\big)$ and $\mu_{R_\mathtt{b}}(x) \stackrel{\text{def}}{=} R_\mathtt{b}(x) \wedge \neg R_\mathtt{a}(x-1)$, where -1 denotes the $\exists\forall$FO-definable predecessor function.

When we consider the effect of this change operation to input words of the form abbabb$\cdots$abb we see that a reduction to quantifier-free replacement queries is not possible any more: the change cannot be characterised by dividing the input string into constantly many parts and applying some relabelling function to each part independently.

This change operation can indeed raise the complexity of maintaining a regular language in DynFO, at least to a small extent. We show this with the example of the regular language $L_{\mathtt{aa}} \stackrel{\text{def}}{=} L((\mathtt{b}^*\mathtt{ab}^*\mathtt{ab}^*)^*)$ over alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ that contains all strings with an even number of a's. Analogously to Example 2.11 one can show that MEMBER($L_{\mathtt{aa}}$) can be maintained in DynFO under change operations Δ_Σ that allow to set or erase the label of a single position, using only nullary auxiliary relations, that is, auxiliary bits. This is not the case any more when the change operation $\rho_{\mathtt{ab}\to\mathtt{aa}}$ is available.

Proposition 5.17. *The dynamic query* (MEMBER($L_{\mathtt{aa}}$), $\Delta_\Sigma \cup \{\rho_{\mathtt{ab}\to\mathtt{aa}}\}$) *cannot be maintained in* DynFO *using only nullary auxiliary relations and arbitrary initialisation.*

Proof sketch. To obtain a contradiction, we assume that there is a dynamic program $\mathcal{P}$ with m auxiliary bits that maintains this dynamic query. Let k be the maximal quantifier rank of any update formula of $\mathcal{P}$.

We consider strings of the form $(ab)^{n_1}(abb)^{n_2}\cdots(ab^{m+1})^{n_{m+1}}$ with $n_i \in \mathbb{N}$. For $I \subseteq \{1,\dots,m+1\}$, let $w(I)$ be the string of this form such that $n_i = 2^k$ if $i \in I$ and $n_i = 2^k + 1$ otherwise. There are 2^m different valuations of m auxiliary bits and 2^{m+1} different strings $w(I)$, so there are different index sets $I_1, I_2 \subseteq \{1,\dots,m+1\}$ such that the dynamic program stores the same auxiliary structure *Aux* for both strings $w(I_1)$ and $w(I_2)$ after an arbitrary initialisation and after applying the canonical change sequences that yield $w(I_1)$ and $w(I_2)$, respectively.

We argue that $\mathcal{P}$ cannot maintain the query result when $\rho_{\texttt{ab}\to\texttt{aa}}$ is applied repeatedly. This is because the update formulas cannot distinguish the two input strings, even after an arbitrary number of applications of $\rho_{\texttt{ab}\to\texttt{aa}}$, and therefore $\mathcal{P}$ assigns the same auxiliary relations after each change step. However, after some number ℓ of applications of $\rho_{\texttt{ab}\to\texttt{aa}}$, exactly one of the strings $\rho^{\ell}_{\texttt{ab}\to\texttt{aa}}(w(I_1))$ and $\rho^{\ell}_{\texttt{ab}\to\texttt{aa}}(w(I_2))$ is in $L_{\texttt{aa}}$. Because $\mathcal{P}$ gives the same query result for both of them, it cannot correctly maintain MEMBER$(L_{\texttt{aa}})$.

For the first part, observe that for each number i of applications of $\rho_{\texttt{ab}\to\texttt{aa}}$ the strings $\rho^{i}_{\texttt{ab}\to\texttt{aa}}(w(I_1))$ and $\rho^{i}_{\texttt{ab}\to\texttt{aa}}(w(I_2))$ only differ in the number of substrings of the form $\texttt{ab}^j$, and both numbers n^1_j and n^2_j are at least 2^k. By a standard Ehrenfeucht-Fraïsse argument, these numbers cannot be distinguished by first-order formulas of quantifier rank at most k.

We argue the second part. By applying $\rho_{\texttt{ab}\to\texttt{aa}}$, substrings of the form $\texttt{b}^*$ get smaller and eventually disappear. When by an application of $\rho_{\texttt{ab}\to\texttt{aa}}$ an even (odd) number of positions change their label from b to a and an even number of $\texttt{b}^*$-substrings disappears, then also the next application of $\rho_{\texttt{ab}\to\texttt{aa}}$ changes the label of an even (odd) number of positions from b to a. If an odd number of $\texttt{b}^*$-substrings disappears, the parity of the number of changed positions is different between the two applications of the change.

Let ℓ' be the smallest index on which I_1 and I_2 differ, that is $\ell' \in I_1 \Leftrightarrow \ell' \notin I_2$. Without loss of generality we assume that $\ell' \in I_1$ and $\ell' \notin I_2$. The string $w(I_1)$ has an even number of substrings $\texttt{ab}^{\ell'}$ (that are not succeeded by another b), and $w(I_2)$ has an odd number of these substrings. That means that with the ℓ'-th application of $\rho_{\texttt{ab}\to\texttt{aa}}$ an even number of $\texttt{b}^*$-substrings disappears in the first string, and an odd number of $\texttt{b}^*$-substrings disappears in the second string. Assuming that $\mathcal{P}$ gave the correct query answers up to the ℓ'-th application of $\rho_{\texttt{ab}\to\texttt{aa}}$ for both input strings, for $\ell \overset{\text{def}}{=} \ell + 1$ the parity of the number of a's is different among $\rho^{\ell}_{\texttt{ab}\to\texttt{aa}}(w(I_1))$ and $\rho^{\ell}_{\texttt{ab}\to\texttt{aa}}(w(I_2))$, as claimed. $\square$

Note that (MEMBER$(L_{\texttt{aa}}), \Delta_\Sigma \cup \{\rho_{\texttt{ab}\to\texttt{aa}}\}$) can be maintained in DynFO$(\leq, +, \times)$ with one unary auxiliary relation (in addition to the ternary arithmetic relations). For a word with n positions, it suffices to store the parity of the number of occurrences of substrings $\texttt{abb}\cdots\texttt{b}$ with exactly ℓ b's, for every $\ell \leq n$. We omit the details. Whether (MEMBER$(L_{\texttt{aa}}), \Delta_\Sigma \cup \{\rho_{\texttt{ab}\to\texttt{aa}}\}$) can be maintained in DynFO with only unary auxiliary relations is open, as well as whether this dynamic query can be maintained under all change operations definable by quantifier-free formulas that may use the predecessor and successor functions.

5.5 AC^1 queries under parameter-free changes

In the last sections we extended known maintainability results for DynFO under single-tuple changes to more expressive change operations, characterised by fragments of first-order logic. We extended not only the results but the known dynamic programs for the single-tuple cases: basically, the techniques we used for the single-tuple case still work for the complex changes we consider.

Remarkably, in this section we prove a maintainability result under a fragment of FO-definable changes that we do not know to hold under single-tuple changes. Of course this means that single-tuple changes cannot be expressed directly by these change operations.

We now study replacement queries without parameters, or equivalently, changes expressible as relational algebra queries that do not use constant values. One might suspect that parameter-free replacement queries are not very powerful, especially when they are applied to an initially empty input structure. However, when the input structure comes with a built-in linear order, one can actually construct every finite graph with relatively simple replacement queries (and similarly for other kinds of structures). For instance, one can cycle through all pairs of nodes in lexicographic order. If (u, v) is the current maximal edge, operation *keep* can move to $(u, v + 1)$, that is, insert this edge into E, while leaving (u, v) in E, and *drop* can move to $(u, v + 1)$ while deleting (u, v) from E. For this reason we include a linear order into the input structure and yield stronger change operations, and therefore stronger results. Note that in practical database scenarios a linear order on the domain is often available.

It turns out that in the setting of ordered structures under parameter-free changes a large class of queries can be maintained in DynFO: all queries that can be expressed in (uniform) AC^1 and thus, in particular, all queries that can be answered in logarithmic space. This result exploits the fact that for a fixed set of replacement queries without parameters there is only a constant number of possible changes to a structure, in each step.

We prove this result in two steps. First we show that the dynamic queries of interest are $(\mathsf{AC}^1, \log n)$-maintainable as defined in Section 3.1. Then we adapt the Muddling technique for single-tuple changes (Theorem 3.3) to prove an analogue of Corollary 3.4 for parameter-free replacement queries.

We start with the proof that AC^1 queries are $(\mathsf{AC}^1, \log n)$-maintainable under parameter-free replacement queries.

Theorem 5.18. *Let q be an AC^1 query over ordered structures and Δ a finite set of parameter-free first-order definable replacement queries. Then (q, Δ) is $(\mathsf{AC}^1, \log n)$-maintainable.*

Proof. Let q be an r-ary AC^1 query over ordered structures and let Δ be a finite set of parameter-free first-order definable replacement queries. We construct an AC^1 algorithm $\mathcal{A}$ and a dynamic program $\mathcal{P}$ such that $\mathcal{P}$ maintains (q, Δ) for $\log n$ many changes starting from auxiliary relations initialised by $\mathcal{A}$. We use a technique that is inspired from the *squirrel technique* of [Zeume and Schwentick 2017] and equip $\mathcal{P}$ with the necessary information to answer q after all change sequences of length $\log n$.

For a fixed set Δ of parameter-free change operations there is only a polynomial number of change sequences of length at most $\log n$ based on these operations. Therefore the algorithm $\mathcal{A}$ can compute the effect of applying each of these change sequences to the input structure, evaluate q on the resulting structure and store the query result in the auxiliary relations. The program $\mathcal{P}$ then only needs to record the change sequence that is actually applied and return the corresponding query result.

We make this more formal now but, for simplicity, only consider the case of ordered graphs ($\sigma_{\text{in}} = \{E, \leq\}$) and assume that $\Delta = \{\rho_0, \rho_1\}$ contains only two change operations.

Let $G = (V, E, \leq)$ be an ordered graph with n vertices. The initialisation $\mathcal{A}$ first computes the predicate BIT that is compatible to the linear order $\leq$. With this predicate, bit sequences of length $\log n$ can be represented by one node of G. We use nodes of the graph to encode change sequences as follows: A sequence $\beta = \alpha_1 \cdots \alpha_m$ with $m \leq \log n$ is encoded by the node v_β whose bit string representation has 1 at position $i \leq m$ if and only if $\alpha_i = \rho_1$, and has 0 at all other positions.

The algorithm $\mathcal{A}$ initialises an $(r+2)$-ary auxiliary relation R as follows. For each $m \leq \log n$ and each $v \in V$ such that $v = v_\beta$ for a change sequence β of length m, $\mathcal{A}$ computes the query result $\textsc{Ans}_v = q(\beta(G))$. Note that $\mathcal{A}$ can compute the graph $\beta(G)$ because the iterated evaluation of $\log n$ first-order formulas is clearly in AC^1. For every tuple $\bar{q} \in \textsc{Ans}_v$, $\mathcal{A}$ inserts the tuple $(v, m, \bar{q})$ in R.

The dynamic program $\mathcal{P}$ only stores the number m of changes that are already applied and the node v that encodes these changes. The updates for these auxiliary relations are clear. The program then only needs to output the relation $\{\bar{q} \mid (v, m, \bar{q}) \in R\}$. $\square$

We now show the analogue of Corollary 3.4 for parameter-free replacement queries. Contrary to Corollary 3.4 we here do not assume that the query to be maintained is almost domain independent: the following theorem is valid for all AC^1 queries. However, this comes with the price that we need an initialisation of the auxiliary relations.

Theorem 5.19 (Muddling for parameter-free definable changes)**.** *Let q be an AC^1 query over ordered structures and Δ a finite set of parameter-free first-order definable replacement queries. If (q, Δ) is $(\mathsf{AC}^1, \log n)$-maintainable, then (q, Δ) is in DynFO with AC^1 initialisation.*

Proof sketch. Our goal is to construct a dynamic program $\mathcal{P}$ that maintains q after an AC^1 initialisation. The construction of this program follows the construction in the proof of Theorem 3.3 that implies membership in $\mathsf{DynFO}(\leq, +, \times)$ for $(\mathsf{AC}^1, \log n)$-maintainable dynamic queries (q', Δ') where q' is almost domain independent and Δ' is a set of single-tuple changes.

In the proof of Theorem 3.3, the constructed program uses threads that work slightly differently depending on whether they are supposed to give the query result for a time point $t \geq \log n$ or for an earlier time point. Observe that only for time points before $\log n$ the assumptions that q' is almost domain independent and that Δ' is a set of single-tuple changes are actually used. This means that the same construction as in the proof of Theorem 3.3 gives a dynamic $\mathsf{FO}(\leq, +, \times)$-program $\mathcal{P}$ without initialisation that can give the query result for (q, Δ) from time point $\log n$ onwards.

It only remains to show that the query result can also be obtained for the first $\log n$ time steps. By Theorem 5.18 there is an AC^1 initialisation $\mathcal{A}'$ and a dynamic program $\mathcal{P}'$ that maintains (q, Δ) for $\log n$ many steps from auxiliary relations initialised by $\mathcal{A}'$. So, the initialisation algorithm $\mathcal{A}''$ that first initialises the relations $+$ and $\times$ and then executes $\mathcal{A}'$ together with the dynamic program $\mathcal{P}''$ that simulates in parallel $\mathcal{P}$ and $\mathcal{P}'$ and outputs the result of $\mathcal{P}'$ until time point $\log n$ and then outputs the result of $\mathcal{P}$ witness that (q, Δ) is in DynFO with AC^1 initialisation. $\square$

The main theorem of this section is now an immediate corollary of the preceding theorems.

Corollary 5.20. *Let q be an AC^1 query over ordered structures and Δ a finite set of parameter-free first-order definable replacement queries. Then (q, Δ) is in* DynFO *with* AC^1 *initialisation.*

5.6 Outlook and bibliographic remarks

In this chapter we studied three classes of queries and three classes of first-order definable changes. We saw that undirected reachability as well as reachability in acyclic directed graphs can be maintained under (restricted) first-order insertions, extended the dynamic programs for regular and context-free languages to changes defined by quantifier-restricted first-order formulas, and obtained membership in DynFO for all AC^1 queries under parameter-free first-order changes.

We comment on the remark in Section 2.3 that the results in this thesis also hold in the variant of the FOIES framework of Dong et al. [1995] obtained by extending the DynFO framework with change operations add(x) and remove(x) that change the underlying domain by one element. These operations do not fit in the parameter-free setting of Section 5.5, but we can allow similar change operations `add` and `remove` that add some new element to the domain as the largest element of $\leq$ or remove that element, respectively. The other results transfer immediately.

We give some pointers for future research. It is still unclear whether the general Reach query can be maintained under non-trivial first-order definable insertions, we could only show that unary auxiliary relations are not sufficient, see Theorem 5.9(a). We have also no results so far regarding the undirected reachability query under first-order deletions. One problem for proving membership in DynFO is that one deletion could render the auxiliary information basically useless in the following sense. Suppose the input graph contains a node v that has an edge to every other node, and the spanning tree stored as auxiliary information consists precisely of these edges. A first-order deletion can remove all of v's edges at once, and the update formulas need to define a spanning tree for the remaining graph from scratch. It therefore seems that a dynamic program that maintains UReach under such first-order deletions needs to store a spanning tree with small, ideally constant, node degrees, as long as this is possible. Unfortunately, the dynamic program for first-order *insertions* from Theorem 5.4 constructs spanning trees with high degree. Possibly this observation can be used to show that UReach cannot be maintained under both first-order insertions *and* deletions, at least if only the auxiliary relations from the proof of Theorem 5.4 are allowed.

The positive result for reachability rely on the bounded bridge property: it is only necessary to concatenate a constant number of old paths to connect a reachable pair of nodes in the changed graph.[7] We relied on a similar property for maintaining formal languages in Section 5.4. The quantifier-restricted change operations studied there split the input string into a constant number of fragments, and in each fragment the positions are relabelled according to the same function. First-order formulas with $\exists\forall$ quantifier prefix can express the successor function on strings, and changes defined using this function do not have the property stated above, as already discussed towards the end of Section 5.4. There we have already sketched that we *can* maintain some regular language in DynFO under some $\exists\forall$FO-definable change operation. The approach used there is tailored towards the specific language and the specific changes, generalisations in any direction are left open. Can this language be maintained under all changes that only use the successor function and no quantifier? Are there regular languages that are not in unary DynFO under $\exists\forall$FO-definable changes?

The state of art regarding lower bound techniques and results is still unsatisfactory. We were able to prove unmaintainability of REACH in DynProp under quantifier-free first-order insertions and deletions, but the proof crucially relies on the fact that the update formulas do not have access to the endpoints of the changed edges. It is therefore unclear how this result could be generalised to show that REACH is not in DynProp under single-edge changes. Further research is necessary to answer this question.

In this chapter we focussed on reachability and on formal language recognition. Of course there are a lot more interesting queries for which we know maintainability in DynFO under single-tuple changes. For all of them we can now ask the question: “Can we do this under first-order changes?” As we already remarked several times, for example when we compared the queries PARITY and ODDCOVEREDNODES in Section 4.3, results of this form may also have consequences for query maintenance under single-tuple changes.

Bibliographic remarks

The results of this chapter are obtained jointly with Thomas Schwentick and Thomas Zeume and are published as [Schwentick et al. 2017a] and [Schwentick et al. 2018]. The detailed distribution is as follows. The results from Section 5.2 appear in both versions, except of the findings of Subsection 5.2.2, which are only contained in [Schwentick et al. 2018]. Theorem 5.9(a) and Theorem 5.11(a) are again published in both versions, Theorem 5.9(b) and Theorem 5.11(b) are only in [Schwentick et al. 2018]. The presentation of the proof of Theorem 5.11 follows the presentation of [Schwentick et al. 2018]. The

[7]Actually, the bounded bridge property is the only necessary assumption for the dynamic programs to work. These dynamic programs are therefore also able to maintain reachability under other, not necessarily first-order definable, changes with the bounded bridge property. For example, we could also allow changes that insert plain sets of edges with the structural restriction that the associated bridge bound is bounded by some constant. See Chapter 6 for further results concerning changes that insert or delete sets of edges.

results concerning formal languages from Section 5.4 only appear in [Schwentick et al. 2017a]. The result of Corollary 5.20 is published in both [Schwentick et al. 2017a] and [Schwentick et al. 2018], although with a different proof.

CHAPTER 6

Expressibility under size-restricted bulk changes

A database table can be modified in ways that cannot be captured by a fixed set of first-order definable change operations. For example, SQL's `INSERT` statement can add a list of tuples without a pre-determined size bound. Specific database systems support non-standard instructions, like for example PostgreSQL's `COPY` command[1] which similarly can insert a long list of tuples. Additionally, if for performance reasons it is not possible to update the auxiliary data after every change but this can only be done periodically, then many changes may accumulate between the updates, even if only single-tuple changes are allowed.

These aspects lead to the requirement that dynamic programs need to be able to update a query result when a set of tuples is changed that does not adhere to any structural limitations. In this chapter we study these *bulk* changes that may insert and delete sets of tuples. Of course we cannot allow these changes to replace the input structure with an arbitrary different one, as a dynamic program then essentially needs to re-compute the query result from scratch. Therefore we restrict these changes quantitatively and demand that the size of the changed sets of tuples is bounded by a function in the domain size. For example, we study changes that allow to insert up to $\log n$ edges into a graph with n nodes.

The goals of this chapter are basically analogous to the goals of Chapter 5. Immediately the question comes up which maintainability results for single-tuple changes survive when bulk changes are allowed, and how large the change may be.

Goal 6.1. Show maintainability in DynFO under bulk changes of non-constant size for queries that can be maintained under single-tuple changes.

[1]see `https://www.postgresql.org/docs/current/sql-copy.html`

On the other hand, as more complex changes make query maintenance more difficult, we want to prove that certain queries cannot be maintained under changes that exceed some size bound.

Goal 6.2. Show for specific queries that they cannot be maintained dynamically under bulk changes of a certain size.

Contributions As in the previous chapter we mostly consider the reachability query for directed and undirected graphs and the membership problem for formal languages. We first show a barrier for these and other queries: we can only hope to maintain them under changes of polylogarithmic size with respect to the size of the domain, at least if we only allow $\mathsf{FO}(\leq, +, \times)$ update formulas. Consequently, we then try to prove matching maintainability results. We succeed for reachability in undirected graphs as well as for the membership problems of regular languages on words and trees. These results rely on an investigation of the expressive power of a restricted version of MSO logic where second-order quantification is restricted to sets of small size, which might be of independent interest.

For Reach on general directed graphs we only show maintainability under insertions of polylogarithmic size. It is known that Reach can be maintained under simultaneous insertions and deletions of $\frac{\log n}{\log \log n}$ edges [Datta et al. 2018b]. We extend this result, which uses an appropriate extension of the Muddling technique from Chapter 3, to more general change operations. Then we proceed to study the Distance query which asks for the length of the shortest path between two graph nodes. We show that this query can be maintained under changes of polylogarithmic size by $\mathsf{FO{+}Maj}(\leq, +, \times)$ update formulas, that is, $\mathsf{FO}(\leq, +, \times)$ formulas that additionally are allowed to use majority quantifiers.

It is still unknown whether Distance is in DynFO under single-tuple changes. We prove as an intermediate result that $\mathsf{DynFO}(\leq, +, \times)$ can maintain auxiliary relations under changes of size $\frac{\log n}{\log \log n}$ such that the query result can be extracted from these relations by an $\mathsf{FO{+}Maj}(\leq, +, \times)$ formula.

These results are joint work with Thomas Zeume, Samir Datta and Anish Mukherjee. Again, more detailed bibliographic remarks are given at the end of the chapter.

Outline We start by extending the dynamic complexity framework to bulk changes in Section 6.1, together with a discussion on principal limits. In Section 6.2 we introduce a restriction of MSO that still expresses Reach and the membership problem for regular languages on words and trees. We show that $\mathsf{FO}(\leq, +, \times)$ can simulate this restricted logic on substructures of polylogarithmic size in the size of the domain, which we use to show that reachability in undirected graphs as well as regular (tree) languages can be maintained under changes of polylogarithmic size. Afterwards, in Section 6.3, we formulate a variant of the Muddling theorem for bulk changes and provide the results we mentioned above regarding Reach and Distance. These results build on the techniques from [Datta et al. 2018a] and we introduce the necessary background from linear algebra before. We conclude in Section 6.4 by giving an outlook and further bibliographic remarks.

6.1 Changing sets of tuples

In this section, we extend our framework to change operations that insert or delete sets of tuples, and discuss the principal limits of query maintenance under these changes.

A single-tuple change α as defined in Chapter 2 consists of a change operation δ that specifies which relation is modified and whether the tuple is inserted or deleted, and the actual tuple that is inserted or deleted. The update formulas access that tuple via explicit first-order parameters. We generalise this approach to *size-restricted bulk changes* that provide a set of changed tuples via an explicit second-order parameter, and introduce the change operations **insert** $f(n)$ **tuples** P **into** R and **delete** $f(n)$ **tuples** P **from** R, for a relations symbol P and a function $f\colon \mathbb{N} \to \mathbb{N}$. We usually denote these operations as $\textsc{ins}_R[f(n)]$ and $\textsc{del}_R[f(n)]$, respectively. Given a structure $\mathcal{D}$ over some domain D and an input schema that contains a k-ary relation symbol R, a *bulk insertion* $\alpha = (\textsc{ins}_R[f(n)], A)$ for $\mathcal{D}$ provides a k-ary set A of size $|A| \leq f(|D|)$. The result $\alpha(\mathcal{D})$ is defined according to intuition: the relation $R^{\alpha(\mathcal{D})}$ is chosen as $R^{\mathcal{D}} \cup A$, all other relations remain unchanged. *Bulk deletions* are defined analogously.

For any set Δ of single-tuple change operations, we write $\Delta[f(n)]$ for the set $\{\delta[f(n)] \mid \delta \in \Delta\}$ of bulk change operations that lift the operations from Δ to changes of size $f(n)$. If $(q, \Delta[f(n)]) \in \mathsf{DynFO}$, we also say that q can be maintained in DynFO under $f(n)$ changes from Δ. Note that in all our subsequent maintainability results stating that some query can be maintained under $f(n)$ changes, we can replace $f(n)$ by $\mathcal{O}(f(n))$, as each change of size $\mathcal{O}(f(n))$ can be replaced by $\mathcal{O}(1)$ changes of size $f(n)$.

In the remainder of this section we discuss which kind of maintainability results we can aim for. For reasons that we discuss later, we mainly consider $\mathsf{FO}(\leq, +, \times)$ update formulas that use built-in arithmetic relations. Much of our reasoning is based on the expressive power of $\mathsf{FO}(\leq, +, \times)$ on small structures. For example, although $\mathsf{FO}(\leq, +, \times)$ cannot express $\textsc{Parity}$ [Ajtai 1983; Furst et al. 1984], it can determine whether a sufficiently small relation has odd size.

Theorem 6.1 ([Denenberg et al. 1986; Fagin et al. 1985]).

(a) *For every $d \in \mathbb{N}$ there is a $\mathsf{FO}(\leq, +, \times)$ formula $\varphi(A)$ that determines whether a set A with $|A| \leq \log^d n$ has odd size, where n is the size of the domain.*

(b) *Such a formula does not exist for sets A that may be as large as $f(n)$, for any function $f(n) \in \log^{\omega(1)} n$.*

Part (a) of this result enables us to extend Example 2.11 and show that the query $\textsc{Parity}$ can be maintained in $\mathsf{DynFO}(\leq, +, \times)$ under polylogarithmically many insertions and deletions of elements into and from the unary relation U.

Proposition 6.2. *The dynamic query* $(\textsc{Parity}, \Delta_U[\log^d n])$ *is in* $\mathsf{DynFO}(\leq, +, \times)$*, for every* $d \in \mathbb{N}$.

Proof sketch. Suppose a bulk change inserts or deletes a set A of elements to or from U. Without loss of generality, we assume that $A \cap U = \emptyset$ in case of an insertion, and $A \subseteq U$ in case of a deletion. The $\mathsf{FO}(\leq, +, \times)$ update formula determines the parity of the number of elements in A, which is possible thanks to Theorem 6.1(a), and updates the query answer accordingly. $\square$

Part (b) of Theorem 6.1 provides a barrier for maintaining Parity under bulk changes.

Proposition 6.3. *Let $f(n) \in \log^{\omega(1)} n$ be an arbitrary function. The dynamic query $(\textsc{Parity}, \Delta_U[f(n)])$ is not in $\mathsf{DynFO}(\leq, +, \times)$.*

Proof sketch. Let $f(n) \in \log^{\omega(1)} n$ be fixed. The assumption $(\textsc{Parity}, \Delta_U[f(n)]) \in \mathsf{DynFO}(\leq, +, \times)$ leads to a contradiction with Theorem 6.1(b). Let φ be the $\mathsf{FO}(\leq, +, \times)$ formula for the change operation $\textsc{ins}_U[f(n)]$ that updates the query result. When the first change $\textsc{ins}_U[f(n)](A)$ is applied starting from an empty input structure, the auxiliary relations are empty and the formula φ needs to determine directly whether A is of odd size. It is therefore easy to obtain from φ a formula that expresses whether a set of size at most $f(n)$ has odd size, contradicting Theorem 6.1(b). $\square$

In light of Proposition 6.3 we see that we cannot hope for meaningful results that a query can be maintained in $\mathsf{DynFO}(\leq, +, \times)$ under more than polylogarithmically many changes. This in particular concerns formal language and graph reachability maintenance, as the membership problem for the regular language of strings with an odd number of a symbols is equivalent to Parity, and the latter problem can also be reduced easily to graph reachability, see Example 2.4.

The previous propositions are stated in terms of $\mathsf{DynFO}(\leq, +, \times)$, not DynFO. For single-tuple changes, the difference is neglectable for most interesting queries, see Proposition 3.2, but this is not true any more for bulk changes. Using standard arguments one can show that FO without built-in linear order and arithmetic can only determine whether a set has odd size if the size of this set is bounded by some fixed constant. The same way we concluded Proposition 6.3 from Theorem 6.1(b) it follows that DynFO without any built-in relations cannot maintain Parity under bulk changes of non-constant size, and consequently neither regular languages nor reachability. Therefore, all results in this chapter will be stated with respect to $\mathsf{DynFO}(\leq, +, \times)$ and its generalisations.

If the size of bulk changes is given as a function in the size of the *active* domain instead of the whole domain and we make sure that a linear order and arithmetic is always available on the active domain, then some results of this chapter transfer to DynFO without built-in arithmetic. This will be discussed in more detail in Section 6.4.

6.2 Utilising second-order logics with restricted quantifiers

An important insight we use in Section 4.1 is that MSO formulas can be evaluated via $\mathsf{FO}(\leq, +, \times)$ formulas on substructures of logarithmic size, as second-order quantification over sets of size $\log n$ can be simulated by first-order quantification over sets of size n. This is possible because each subset of a set with $\log n$ elements can be represented by one of n nodes.

In this chapter's setting, a change might insert many edges into a graph. If we, for example, want to maintain the transitive closure of the graph, we necessarily need to be able to *express* the transitive closure of the graph that only consists of the just inserted edges. If $\log n$ edges are inserted into a graph with n nodes, the same reasoning as above gives that an $\mathsf{FO}(\leq, +, \times)$ update formula can express the transitive closure of the inserted edges, as Reach is MSO-expressible, see Example 2.1. Of course, in a second step, this information needs to be combined with the maintained transitive closure of the graph before the change.

But what happens if polylogarithmically many edges are inserted? We cannot simulate MSO on the set of nodes that is affected by such a change, simply because we cannot represent all of its subsets by constant-sized tuples over all nodes, as there are super-polynomially many of these subsets. We circumvent this problem and show that we can represent all subsets of affected nodes up to a certain size, and even relations of arbitrary arity over this set, as long as they contain only few tuples. It follows that we can simulate a weak form of second-order quantification that is only able to quantify relations of small size. We demonstrate that this restricted form of second-order logic is still very expressive, and deduce that $\mathsf{FO}(\leq, +, \times)$ update formulas can express reachability and regular languages on words and trees on the substructures that are affected by a change of polylogarithmic size. Then we use this ability to show that the corresponding query results can be updated for the whole structure.

6.2.1 The expressive power of restricted second-order quantification

In this section we are particularly interested in second-order logics with size-restricted quantification. In model theory, the restriction of MSO that allows only quantification over *finite* sets is called *weak* MSO. Over finite structures, in contrast to arbitrary structures, this restriction of course does not change the expressive power of MSO. Here we introduce a more restricted variant of MSO that only allows quantification over sets of bounded size with respect to the underlying domain.

Let $f\colon \mathbb{N} \to \mathbb{N}$ be a fixed function. The syntax of $f(n)$-MSO over a given schema σ is defined exactly as for MSO. We do not give a formal definition of its semantics, but rather explain the difference to regular MSO. Intuitively, an $f(n)$-MSO formula $\exists X\, \varphi(X)$ is satisfied in a σ-structure $\mathcal{D}$ with universe D, if there is a set $D' \subseteq D$ with $|D'| \leq f(|D|)$ such that $\mathcal{D} \models \varphi(D')$; a formula $\forall X\, \varphi(X)$ is satisfied if for all sets $D' \subseteq D$ with $|D'| \leq f(|D|)$ it holds that $\mathcal{D} \models \varphi(D')$.

In the following we list results highlighting that MSO is still very expressive when quantified sets are bounded by $\sqrt[i]{n}$, for arbitrary i. We first recall some definitions. Remember that words over an alphabet Σ are encoded as ordered structures with unary relations R_σ for every $\sigma \in \Sigma$, see also Section 5.4. To define languages of trees instead of words, we consider *ranked* alphabets Σ, that is, every symbol $\sigma \in \Sigma$ comes with an associated natural number, called its *rank*. A *tree language* over a ranked alphabet Σ is a set of rooted, Σ-labelled, ranked and ordered trees that we represent by relations FirstChild and NextSibling with the obvious meaning, with the property that the number of children of a node equals the rank of its symbol $\sigma \in \Sigma$. A tree language is *regular* if it is accepted by a deterministic finite (bottom-up) tree automaton (see c.f. [Comon et al. 2007]).

Theorem 6.4. *The following queries can be expressed in* $\sqrt[i]{n}$-MSO*, for every* $i \geq 1$*.*

(a) Reachability in directed graphs,

(b) the word problem for any regular language,

(c) the word problem for any regular tree language.

Proof. Let $i \geq 1$ be fixed. We prove the parts separately, but with similar proof ideas.

(a) We use an approach that is well-known in the context of small-depth circuits computing small distance connectivity, c.f. [Chen et al. 2016]. From the MSO formula for Reach from Example 2.1 we can easily construct an $\sqrt[i]{n}$-MSO formula that expresses reachability via paths of length at most $\sqrt[i]{n}$ in a graph G of size n. Such a formula $\varphi_{\sqrt[i]{n}}(s,t)$ basically states that there is a subset V' of nodes (of size at most $\sqrt[i]{n}+1$) such that t is reachable from s in $G[V']$. As we cannot quantify over sets of size $\sqrt[i]{n}+1$, the formula[2] asserts that there is an edge from s to a node v such that t is reachable from v via a path of length at most $\sqrt[i]{n}-1$.

$$\begin{aligned}\varphi_{\sqrt[i]{n}}(s,t) \overset{\text{def}}{=} s = t \vee \exists V' \, \exists v \in V' \, \forall X \subseteq V' \\ \Big(E(s,v) \wedge \Big(X(u) \wedge \forall x \in X \, \forall y \in V' \, \big(E(x,y) \to X(y)\big)\Big) \to X(t)\Big)\end{aligned}$$

With a Savich-like construction, this formula can be lifted to paths of greater length. A path of length $\sqrt[i]{n^2}$ can be decomposed into $\sqrt[i]{n}$ paths of length $\sqrt[i]{n}$. So, the $\sqrt[i]{n}$-MSO formula $\varphi_{\sqrt[i]{n^2}}(s,t)$ that we obtain from $\varphi_{\sqrt[i]{n}}(s,t)$ by replacing atoms $E(x,y)$ with $\varphi_{\sqrt[i]{n}}(x,y)$ expresses reachability along paths of length $\sqrt[i]{n^2}$. Repeating this step i times results in an $\sqrt[i]{n}$-MSO formula for Reach.

(b) Let L be some regular language over an alphabet Σ and let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton (DFA, with state set Q, transition function δ, initial state q_0 and set F of accepting states) that decides L. We give an $\sqrt[i]{n}$-MSO formula for Member(L), the word problem for L.

The usual approach to express Member(L) in MSO is to construct a formula $\psi^{p,q}(j,k)$ for any pair p,q of states of $\mathcal{A}$ such that $\psi^{p,q}(j,k)$ is satisfied on a string $w = w_1 \cdots w_n$ if $\mathcal{A}$ goes from state p to state q while reading the word $w_j \cdots w_k$, so, if $\delta^*(p, w_j \cdots w_k) = q$. This formula existentially quantifies for all states $q \in Q$ the set of positions from w such that $\mathcal{A}$ assumes the state q after reading this position. The formula then checks whether for all pairs of neighbouring positions these choices are consistent with the transition function δ.

Under $\sqrt[i]{n}$-MSO semantics, an almost identical formula $\psi^{p,q}_{\sqrt[i]{n}}(j,k)$ can check whether $\delta^*(p, w_{j+1} \cdots w_k) = q$ holds, as long as $w_{j+1} \cdots w_k$ is a substring of length at most $\sqrt[i]{n}$.

Similar to part (a) of this proof, we extend this formula to substrings of larger size. The formula $\psi^{p,q}_{\sqrt[i]{n^2}}(j,k)$ that extends $\psi^{p,q}_{\sqrt[i]{n}}(j,k)$ to substrings of length $\sqrt[i]{n^2}$ existentially quantifies a set P of positions with the intention that these positions divide the substring of length $\sqrt[i]{n^2}$ into $\sqrt[i]{n}$ substrings of length $\sqrt[i]{n}$. Additionally, it uses existential quantifiers

[2]For the sake of notational clarity we use abbreviations "$\subseteq$" and "$\in$" that are easily expressible in $\sqrt[i]{n}$-MSO.

to assign a state from Q to every position from P. For all pairs $j' < k'$ of positions from P such that no $\ell \in P$ with $j' < \ell < k'$ exists, the formula checks that $\psi^{p_1,p_2}_{\sqrt[i]{n}}(j', k')$ is satisfied for the states p_1, p_2 that are assigned to the positions j' and k'.

We obtain formulas $\psi^{p,q}_{n}(j, k)$ by repeating this construction. The formula φ_L that expresses MEMBER(L) only needs to iterate over all pairs $p \in Q$ and $f \in F$ of states and check that for one such pair $\delta(q_0, w_1) = p$ and $\psi^{p,f}_{n}(1, n)$ hold.

(c) We follow a similar approach as for part (b) of the proof and construct formulas for larger and larger parts of the input tree. To define subtrees of a tree easily, we use a concept similar to the concept of triangles as defined in Subsection 4.1.1. Let $T = (V, E, r)$ be a rooted tree, $t \in V$ and $B \subseteq V$. The tree $T(t, B)$ is the subtree of T rooted at t, but all inner nodes of this subtree that are in B become leaves. More formally, let $T(t) = (V', E')$ be the subtree of T rooted at t, and let $B' \overset{\text{def}}{=} B \cap V'$. The tree $T(t, B)$ results from $T(t)$ by removing all children of B' and their subtrees.

Let L be a regular tree language over a ranked alphabet Σ with maximal rank k and let $\mathcal{A} = (Q, \Sigma, \delta, F)$ be a corresponding deterministic finite tree automaton. Suppose we are given a set B of tree nodes and a partition $B_{q_1}, \dots, B_{q_\ell}$ of this set that associates a state $q_i \in Q$ to every node from B. Analogously to part (b), there is an $\sqrt[i]{n}$-MSO formula $\psi^{q}_{\sqrt[i]{n}}(t, B_{q_1}, \dots, B_{q_\ell})$ that expresses for trees $T(t, B)$ of size $\mathcal{O}(\sqrt[i]{n})$ that $\mathcal{A}$ assigns the state q to t in $T(t, B)$ if the initial states on the leaf nodes from B are as given by the partition. This formula existentially quantifies for each state from Q a set of nodes[3] and verifies that $\mathcal{A}$ actually assigns these states to the corresponding nodes.

We now present a formula $\psi^{q}_{\sqrt[i]{n^2}}(t, B_{q_1}, \dots, B_{q_\ell})$ for subtrees $T(t, B)$ of size $\mathcal{O}(\sqrt[i]{n^2})$. This formula existentially quantifies a set B' of nodes as well as a partition into ℓ sets $B'_{q_1}, \dots, B'_{q_\ell}$ with the intention that for each $t' \in B' \cup \{t\}$ the subtrees $T(t', B \cup B')$ have size at most $k\sqrt[i]{n}$. It then checks that for all $t' \in B' \cup \{t\}$ the formula $\psi^{q}_{\sqrt[i]{n}}(t', B_{q_1} \cup B'_{q_1}, \dots, B_{q_\ell} \cup B'_{q_\ell})$ is satisfied.

We iterate this step and obtain a formula $\psi^{q}_{n}(t)$ that expresses whether $\mathcal{A}$ assigns the state q to t in $T(t)$, and the final formula for the word problem of L only needs to check whether $\psi^{f}_{n}(r)$ holds for any accepting state $f \in F$. □

We will actually not directly apply parts (b) and (c) of Theorem 6.4 later. We state and prove them here because they show that $\sqrt[i]{n}$-MSO and MSO have the same expressive power on words and ranked trees, respectively, for each $i \in \mathbb{N}$, which might be of independent interest. This is not true any more if the quantifiers are further restricted.

Proposition 6.5. *Let $f(n)$ be a function from $n^{o(1)}$. There is no $f(n)$-MSO formula φ that expresses* MEMBER(L_{aa})*, the word problem for the regular language of strings with an even number of* a*'s.*

Proof. Let $f(n) \in n^{o(1)}$ be fixed. We assume that there is an $f(n)$-MSO formula φ for MEMBER(L_{aa}) and derive a contradiction to Håstad's lower bound on the size of constant-depth circuits for PARITY [Håstad 1986].

[3]We allow here and in the following to quantify sets of size $\mathcal{O}(\sqrt[i]{n})$, which is actually realised by quantifying $\mathcal{O}(1)$ sets of size $\sqrt[i]{n}$.

Let d be the depth of the syntax tree of φ. Given some n, from the syntax tree of φ we construct in the natural way a depth-d Boolean circuit C_n for input strings of length n as follows. Every existential quantifier is replaced by an "or" gate, with fan-in n for a first-order quantifier and with fan-in upper-bounded by $n^{f(n)}$ for an $f(n)$-MSO second-order quantifier. Analogously, universal quantifiers are replaced by "and" gates with fan-in n or $n^{f(n)}$, respectively. The obtained circuit is a tree of depth d and of degree at most $n^{f(n)}$, so the number of gates is upper-bounded by $(n^{f(n)})^{d+1} = 2^{\log n(d+1)f(n)}$ which for $f(n) \in n^{o(1)}$ is not in $2^{\Omega(n^{\frac{1}{d-1}})}$, the lower bound for depth-d circuits that compute a parity function on n input bits [Håstad 1986]. $\square$

The same way we defined the restriction $f(n)$-MSO of MSO we can restrict full second-order logic that allows to quantify over relations instead of plain sets. Let $r\mathsf{SO}$, for $r \in \mathbb{N}$, be the subset of second-order logic that allows quantification over r-ary relations. The logic $f(n)$-$r\mathsf{SO}$ is the logic that restricts $r\mathsf{SO}$ to quantification over r-ary relations with at most $f(n)$ many tuples, where n is the size of the underlying domain.

Also the restriction of SO to quantification over relations of size $\sqrt[i]{n}$ is very expressive, especially if we allow formulas to use a built-in linear order.

Theorem 6.6. *Let q be a query that is computable in NL. There is an $r \in \mathbb{N}$ such that q is expressible in $\sqrt[i]{n}$-$r\mathsf{SO}(\leq)$, for every $i \in \mathbb{N}$.*

Proof. Let q and i be fixed. As REACH is NL-complete under first-order reductions [Immerman 1987], there is an r-dimensional first-order reduction Ψ from q to REACH, for some $r \in \mathbb{N}$. Note that Ψ uses a built-in linear order on the domain. Thanks to Theorem 6.4 there is an $\sqrt[i]{n}$-MSO formula φ that expresses REACH. Combining φ with Ψ yields an $\sqrt[i]{n}$-$r\mathsf{SO}(\leq)$ formula that expresses q. $\square$

6.2.2 Reachability, regular languages and changes of polylogarithmic size

With the help of the results of the previous subsection we now construct dynamic programs that maintain UREACH and regular (tree) languages under changes of polylogarithmic size. In light of Proposition 6.3, these results are optimal with respect to the size of the change: no $\mathsf{DynFO}(\leq, +, \times)$ program can maintain these queries under changes of larger size.

Our approach is as follows. We show that $\mathsf{FO}(\leq, +, \times)$ formulas can simulate $\sqrt[i]{n}$-SO formulas on substructures of polylogarithmic size, for some $i \in \mathbb{N}$. For the different dynamic queries we then prove that the query result and all other necessary auxiliary relations can be updated using $\sqrt[i]{n}$-SO expressible relations on the set of elements that are affected by the change.

The following lemma is an adaptation of Proposition 4.11 in combination with the explanations in the proof of Theorem 4.3 on how to encode subsets of a set of size $\log n$.

Lemma 6.7. *Let d and r be arbitrary natural numbers, and let $i \overset{def}{=} 2d$. Let $\psi(\bar{x})$ be an $\sqrt[i]{n}$-$r\mathsf{SO}(\leq)$ formula over some relational schema σ. There is an $\mathsf{FO}(\leq, +, \times)$*

formula $\varphi(\bar{x})$ such that for every σ-structure $\mathcal{D}$ with domain D, every subset $D' \subseteq D$ of size $|D'| \leq \log^d |D|$ and every tuple $\bar{a}$ of elements from D' it holds that $\mathcal{D}[D'] \models \psi(\bar{a})$ if and only if $(\mathcal{D}, D') \models \varphi(\bar{a})$.

Proof. Let d, i, r and ψ be as in the statement of the lemma. The idea to construct φ is as follows. Given a structure $\mathcal{D}$ with universe D of size n, the formula ψ is only applied to substructures of $\mathcal{D}$ of size $\log^d n$. So, it only quantifies relations with at most $\sqrt[i]{\log^d n} = \sqrt{\log n}$ many tuples over a set of $\log^d n$ many elements. We will see that relations of this form and size can be encoded by $\mathcal{O}(\log n)$ many bits, which in turn can be represented by constantly many elements from D. An $\mathsf{FO}(\leq, +, \times)$ formula φ can quantify these elements and decode the represented relation.

Let $D' \subseteq D$ be a set of size at most $\log^d n$. We explain step by step how relations of small size over D' can be encoded. We will use the well-known fact that $\mathsf{FO}(\leq, +, \times)$ formulas can identify a set D' of polylogarithmic size with the first $|D'|$ elements of the domain D with respect to $\leq$, see for example [Durand et al. 2006, Corollary 1]. So, we can represent any element from D' by a number between 0 and $|D'| - 1$, and, using the $\mathsf{FO}(\leq, +, \times)$-expressible BIT predicate, by a bit string of length $\log |D'| = d \log\log n$, for a set D' of size $\log^d n$. An arbitrary r-tuple over elements from D' can consequently be represented by a bit string of length $dr \log\log n$, and a set of at most $\sqrt[i]{\log^d n} = \sqrt{\log n}$ of these tuples by a bit string of length $dr\sqrt{\log n} \log\log n$. The latter number is smaller than $dr \log n$, for all n larger than some number n_0, which in turn is the length of a bit string that can be represented by rd quantified elements in the presence of BIT. Small structures with less than n_0 elements can be dealt with separately.

We construct the $\mathsf{FO}(\leq, +, \times)$ formula φ from ψ by first replacing every first-order existential quantification $\exists x \psi'(x)$ by $\exists x \left(D'(x) \wedge \psi'(x)\right)$, and every universal quantification $\forall x\, \psi''(x)$ by $\forall x \left(D'(x) \to \psi''(x)\right)$. Then, we replace every existential second-order quantifier $\exists X$ by first-order quantifiers $\exists x_1 \cdots \exists x_{rd}$, and symmetrically universal quantifiers $\forall X$ by $\forall x_1 \cdots \forall x_{dr}$. Every atom $X(\bar{y})$ is replaced by an $\mathsf{FO}(\leq, +, \times)$ subformula $\varphi_X(\bar{y})$ that checks whether the bit string of length $dr \log\log n$ associated with the interpretation of $\bar{y}$ is contained in the bit string represented by $x_1, \ldots, x_{dr}$. We make this more precise. Let $(a_1, \ldots, a_r) \in {D'}^r$ be the interpretation of $\bar{y}$, and $s_1, \ldots, s_r$ the bit strings of length $d \log\log n$ associated with these elements as explained above. The formula φ_X checks whether there is a $j \leq \sqrt{\log n}$ such that for each $r' \leq r$, each $d' \leq d$ and each $p \leq \log\log n$ the bit at position $(d' - 1) \log\log n + p$ of $s_{r'}$ equals the bit at position $(j - 1) \log\log n + p$ in the bit string represented by $x_{(d'-1)r+r'}$. □

Lemma 6.7 in conjunction with Theorem 6.6 enables us to express in $\mathsf{FO}(\leq, +, \times)$ any NL-computable query on structures of polylogarithmic size.

Corollary 6.8. *Let k and d be arbitrary natural numbers, and let q be a k-ary, NL-computable query on σ-structures. There is an $\mathsf{FO}(\leq, +, \times)$ formula φ over schema $\sigma \cup \{D'\}$ such that for any σ-structure $\mathcal{D}$ with n elements, any subset D' of its domain of size at most $\log^d n$ and any k-tuple $\bar{a} \in {D'}^k$ it holds that $\bar{a} \in q(\mathcal{D}[D'])$ if and only if $(\mathcal{D}, D') \models \varphi(\bar{a})$.*

In the following, we simulate different NL computations on the part of the input structure that is directly affected by a change of polylogarithmic size. As a first example, we show that reachability in directed graphs can be maintained under insertions of polylogarithmically many edges.

Theorem 6.9. *The dynamic query* $(\textsc{Reach}, \{\textsc{ins}_E[\log^d n]\})$ *is in* $\mathsf{DynFO}(\leq, +, \times)$*, for every* $d \in \mathbb{N}$.

Proof. Let $d \in \mathbb{N}$ be fixed. We extend the dynamic program from Example 2.8 to insertions of $\log^d n$ many edges. So, the dynamic program we construct only maintains the query relation $\textsc{Ans}$, the transitive closure of the input graph. Whenever a set E^+ of edges is inserted into a graph $G = (V, E)$, the dynamic program updates $\textsc{Ans}$ with the help of the transitive closure relation of some graph H that is defined on the set V_{aff} of affected nodes, that is, the set of incident nodes of the edges in E^+. The edge set E_H of H contains the newly inserted edges E^+, and additionally edges (u, v) for all pairs (u, v) of nodes from V_{aff} that are connected by a path in G. The transitive closure of H equals the transitive closure relation of $G' \stackrel{\text{def}}{=} (V, E \cup E^+)$ restricted to V_{aff}: it accounts for all paths from a node $u \in V_{\text{aff}}$ to another node $v \in V_{\text{aff}}$ that may use both newly inserted edges as well as edges that are already present in G. The transitive closure relation of G' can then easily be expressed as follows. Every path ρ in G' consists of three consecutive subpaths $\rho_1\rho_2\rho_3 = \rho$, where ρ_1 and ρ_3 are defined as the maximal subpaths of ρ that do not rely on edges from E^+. These subpaths already exist in G and are represented in $\textsc{Ans}$. The subgraph ρ_2 by definition starts and ends at nodes from V_{aff}, so its existence is given by the transitive closure relation of H.

We now give a more formal construction. Let $G = (V, E)$ be a graph with $|V| = n$ and $\textsc{Ans}$ the transitive closure of E, and let E^+ be a set of at most $\log^d n$ edges. Let H be the graph with node set $V_H \stackrel{\text{def}}{=} \{u, v \mid (u, v) \in E^+\}$ and edge set $E_H \stackrel{\text{def}}{=} E^+ \cup \{(u, v) \in V_H^2 \mid (u, v) \in \textsc{Ans}\}$. The graph H is clearly FO-definable from G, E^+ and $\textsc{Ans}$, and by Corollary 6.8 its transitive closure relation TC_H can be expressed in $\mathsf{FO}(\leq, +, \times)$, as V_H is of polylogarithmic size with respect to n. The transitive closure of $G' \stackrel{\text{def}}{=} (V, E \cup E^+)$ can then be defined by the formula $\varphi(s, t) \stackrel{\text{def}}{=} \textsc{Ans}(s, t) \vee \exists x_1 \exists x_2\, (\textsc{Ans}(s, x_1) \wedge \text{TC}_H(x_1, x_2) \wedge \textsc{Ans}(x_2, t))$. □

So far we do not know how to maintain directed reachability under polylogarithmically many deletions in $\mathsf{DynFO}(\leq, +, \times)$. There are preliminary results in this direction, showing that $\textsc{Reach}$ can be maintained under fewer, but non-constantly many deletions in $\mathsf{DynFO}(\leq, +, \times)$, or under polylogarithmically many deletions in extensions of $\mathsf{DynFO}(\leq, +, \times)$, which we will discuss in more detail in the following Section 6.3.

When it comes to undirected reachability, we can actually extend the dynamic program from Dong and Su [1998] for $\textsc{UReach}$ to edge insertions and deletions of polylogarithmic size.

Theorem 6.10. *The dynamic query* $(\textsc{UReach}, \Delta_E[\log^d n])$ *is in* $\mathsf{DynFO}(\leq, +, \times)$*, for every* $d \in \mathbb{N}$.

Proof. The dynamic program from [Dong and Su 1998] that maintains undirected reachability in DynFO under single-edge changes uses, in addition to the transitive closure relation of the input graph, two binary auxiliary relations that represent a directed spanning forest of the input graph and its transitive closure, respectively. We show that these relations can still be maintained in DynFO$(\leq, +, \times)$ under changes of $\log^d n$ many edges, for some fixed d.

For edge insertions, the construction idea is very similar to the proof of Theorem 6.9. We define a graph H, where nodes correspond to connected components of the input graph that include an affected node, and edges indicate that some inserted edge connects the respective connected components. As this graph is of polylogarithmic size, thanks to Corollary 6.8 we can express a spanning forest for H and its transitive closure in FO$(\leq, +, \times)$, which is sufficient to update the respective relations for the whole input graph.

In the case of edge deletions, the update formulas need to replace deleted spanning tree edges, whenever this is possible. Our approach is very similar to the case of edge insertions. The spanning tree decomposes into polylogarithmically many connected components when edges are deleted. These components can be merged again if non-tree edges exist that connect them, and these edges become tree edges of the spanning forest. For a correspondingly defined graph of polylogarithmic size we can again define a spanning forest and its transitive closure, and from this information select the new tree edges.

We explain both cases in more detail. Let $G = (V, E)$ be the undirected input graph of size n with transitive closure $\textsc{Ans}$, and let S and TC_S be a directed spanning forest for G and its transitive closure, respectively. Suppose that a set E^+ of size at most $\log^d n$ is inserted. We define a graph H as follows. It contains a node $v \in V$ if (1) v is affected, that is, if E^+ contains an edge of v, and (2) v is the smallest affected node in its connected component of G with respect to $\leq$. It contains an edge (u, v) if $(u', v') \in E^+$ for some nodes u', v' with $(u, u') \in \textsc{Ans}$ and $(v, v') \in \textsc{Ans}$, so, if the connected components of u and v are connected by an inserted edge. The graph H is easily seen to be FO-definable using $\textsc{Ans}$. Because H is of polylogarithmic size with respect to n and a spanning forest of a graph can be computed in NL (for example the breadth-first spanning forest with the minimal nodes of each component, with respect to $\leq$, as roots), we can define a spanning tree S_H and its transitive closure TC_{S_H} in FO$(\leq, +, \times)$, thanks to Corollary 6.8.

The update formulas define updated auxiliary relations for the graph $G' = (V, E \cup E^+)$ as follows. Intuitively, an edge $(u, v) \in S_H$ means that the connected components of u and v in G shall be connected in G' directly by a new tree edge. There might be several edges in E^+ that may serve this purpose, and we need to choose one of them. So, an edge $(u', v') \in E^+$ becomes part the updated spanning forest if there is an edge $(u, v) \in S_H$ such that u' and u as well as v' and v are in the same connected component of G, respectively, and (u', v') is the lexicographically minimal edge with these properties. This is clearly FO$(\leq, +, \times)$-expressible using the old auxiliary relations. The old tree edges from S are taken over to the updated version, although some directions need to be inverted, see [Dong and Su 1998] and Theorem 5.4 for how this can be expressed in first-order logic. The update formulas for TC_S from [Dong and Su 1998] can easily be extended to our more general case, using TC_{S_H}. We note that $\textsc{Ans}$ is first-order expressible from TC_S.

In conclusion, all auxiliary relations can be updated in $\mathsf{FO}(\leq, +, \times)$.

Now suppose that a set E^- of at most $\log^d n$ edges is deleted. Let S' be the spanning forest that results from S after all tree edges from E^- are removed, and let $\mathrm{TC}_{S'}$ be its transitive closure, which is easily FO-expressible from TC_S. Similarly as above we define a graph H, with nodes being the minimal affected nodes in a weakly connected component of S', which are connected by an edge if the respective weakly connected components of S' are connected by some edge from $E \setminus E^-$. The same way as above, $\mathsf{FO}(\leq, +, \times)$ formulas can define a spanning forest and its transitive closure for H and then use this information to define a spanning forest and its transitive closure for the changed graph $G' = (V, E \setminus E^-)$. □

We have shown in Theorem 6.4 that regular languages on words and trees can be expressed in $\sqrt[i]{n}$-MSO, for any $i \in \mathbb{N}$. Now we use this result to show that these languages can be maintained under changes of the labels of polylogarithmically many positions, very similarly to the proof of Theorem 6.9. We refer to Section 5.4 and to [Gelade et al. 2012] for technical details of the setting.

Theorem 6.11. *The following dynamic queries are in* $\mathsf{DynFO}(\leq, +, \times)$*, for every* $d \in \mathbb{N}$*:*

(a) $(\textsc{Member}(\mathrm{L}), \Delta_\Sigma[\log^d n])$*, for any regular language* L *over some alphabet* Σ*,*
(b) $(\textsc{Member}(\mathrm{L}), \Delta_\Sigma[\log^d n])$*, for any regular tree language* L *over some ranked alphabet* Σ*.*

Proof.

(a) Let L be a regular language over an alphabet Σ, and let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a corresponding DFA. To maintain $\textsc{Member}(\mathrm{L})$ under changes of single positions, Gelade et al. [2012] use auxiliary relations of the form $S_{p,q}(i,j)$, where $p, q \in Q$ are states of $\mathcal{A}$ and $(i,j) \in S_{p,q}$ holds if and only if $\delta^*(p, w_{i+1} \cdots w_{j-1}) = q$. Here, $w = w_1 \cdots w_n$ is the input word and δ^* is the extension of the transition function δ from symbols $\sigma \in \Sigma$ to words from Σ^*.

We show that these auxiliary relations can be maintained in $\mathsf{DynFO}(\leq, +, \times)$ under changes of polylogarithmic size. In a nutshell, we show that using the auxiliary relations, the behaviour of $\mathcal{A}$ on a string with polylogarithmically many changed positions can be characterised by another DFA that only reads a string of polylogarithmic length. Whether this automaton accepts can be determined in $\mathsf{FO}(\leq, +, \times)$ by Corollary 6.8; we show that is enough to update all auxiliary relations.

We elaborate further on this approach. Suppose the labels of the set $P = \{i_1, \ldots, i_m\}$ of at most $\log^d n$ positions are changed, resulting in an input word $w' = w'_1 \cdots w'_n$. We show how the auxiliary relations can be updated for the substring $w'_{i_1} \cdots w'_{i_m}$ that contains all changed positions. More precisely, we construct for every pair p, q of states of $\mathcal{A}$ an $\mathsf{FO}(\leq, +, \times)$ formula $\varphi_{p,q}$ that checks whether $\delta^*(p, w'_{i_1} \cdots w'_{i_m}) = q$. From these formulas we easily obtain another $\mathsf{FO}(\leq, +, \times)$ formula that updates the query relation: as the prefix of w' up to position $i_1 - 1$ and the suffix from position $i_m + 1$ onwards are unchanged and therefore the old auxiliary relations are still valid, it suffices to check that for some pair p, q of states $\delta^*(q_0, w'_1 \cdots w'_{i_1 - 1}) = p$ and $\delta^*(q, w'_{i_m+1} \cdots w'_n) \in F$ hold, which is easy given the auxiliary relations, and that $\varphi_{p,q}$ is satisfied. The other auxiliary relations can be updated analogously, and we do not spell out further details.

It remains to explain how the formulas $\varphi_{p,q}$ are constructed. Using the existing auxiliary relations, we can determine the behaviour of $\mathcal{A}$ on the substring $w'_{i_j} \cdots w'_{i_{j+1}-1}$ of $w'_{i_1} \cdots w'_{i_m}$ that for each $j < m$ starts with a changed position i_j and ends just before the next changed position, and that only consists of the symbol at position i_m for $j = m$. More formally, an FO formula can associate with every $j < m$ a function $\gamma_j \colon Q \to Q$ such that $\gamma_j(p') = q'$ if and only if $\delta^*(p', w'_{i_j} \cdots w'_{i_{j+1}-1}) = q'$ holds, for any $p', q' \in Q$, so, if $\mathcal{A}$ ends in state q' if it starts in state p', reads the new symbol at position i_j, and then reads the unchanged subword $w'_{i_j+1} \cdots w'_{i_{j+1}-1} = w_{i_j+1} \cdots w_{i_{j+1}-1}$. The formula can associate with the last position i_m the function $\gamma_m(p') \overset{\text{def}}{=} \delta(p', w'_{i_m})$.

It now holds that $\delta^*(p, w'_{i_1} \cdots w'_{i_m}) = q$ if and only if $(\gamma_m \circ \cdots \circ \gamma_1)(p) = q$. The set $L_{p,q}$ of sequences $\gamma'_1 \cdots \gamma'_\ell$ such that $(\gamma'_\ell \circ \cdots \circ \gamma'_1)(p) = q$ is clearly a regular language over the alphabet Γ of all functions from $Q \to Q$. The formula $\varphi_{p,q}$ thus only needs to FO-define the word $\gamma_1 \cdots \gamma_m$ and check that it is included in $L_{p,q}$, which is possible in FO$(\leq, +, \times)$ by Corollary 6.8.

(b) The proof is very similar to part (a). Let L be a regular tree language over a ranked alphabet Σ. We assume that the symbols in Σ have rank 0 or 2, so, that L is a regular language of binary trees. Let $\mathcal{A} = (Q, \Sigma, \delta, F)$ be a finite deterministic bottom-up tree automaton that decides L. We denote the labelled rooted input tree by T.

To maintain MEMBER(L) when the label of a single node changes, Gelade et al. [2012] mainly use binary auxiliary relations that are again of the form $S_{p,q}(u, v)$, for states $p, q \in Q$. Intuitively, $(u, v) \in S_{p,q}$ means that $\mathcal{A}$ assigns the state p to the node u, if the state for node v is forced to be q — no matter which state $\mathcal{A}$ actually assigns to v based on its subtree and the transition function δ.

We show that these auxiliary relations can be updated after changes of the labels of polylogarithmically many nodes. For this we assume the existence of the descendant relation on the tree nodes, which can be made available as an auxiliary relation. Our strategy is analogous to part (a): we FO-define a labelled binary tree with polylogarithmically many nodes such that from the information whether this tree is member of some regular tree language we can infer whether the changed input tree is in L, and how the other auxiliary relations need to be updated. Thanks to Corollary 6.8, this approach can be implemented by FO$(\leq, +, \times)$ update formulas.

We give some details. Let P be the set of at most $\log^d n$ nodes whose labels are modified by a change, resulting in an input tree T'. Without loss of generality, we assume that for two nodes $v, v' \in P$ also their lowest common ancestor is in P, and that if for a node $v \in P$ a descendant from its left or right subtree is contained in P, also some descendant from its other subtree is in P. So, with the help of the descendant relation on T, we can define in FO a binary tree T_P with node set P and edges from a node to its "nearest descendants" in T.

We assign labels to the nodes of T_P according to the behaviour of $\mathcal{A}$ on the subtree of T' that is rooted in the respective node. If $v \in P$ is a leaf of T_P, then its label is just the state $q \in Q$ that $\mathcal{A}$ assigns to this node in T'. As the only difference in the subtree rooted at v between T and T' is the label of v itself, this state can easily be expressed using the old auxiliary relations and the transition function of $\mathcal{A}$. If $v \in P$ is an inner node of T_P, its label is a function $\gamma\colon Q \times Q \to Q$. Intuitively, this labels says that if $\mathcal{A}$ assigns in T' some states p, q to the nodes u_1, u_2 that are the children of v in T_P, then it assigns the state $\gamma(p, q)$ to v in T'. This label can be expressed in first-order logic from the old auxiliary relations as follows. Let v_1, v_2 be the children of v in T such that u_1 is a descendant of v_1 and u_2 is a descendant of v_2. If the state p is assigned to u_1 in T', then the state p' is assigned to v_1 that satisfies $(u_1, v_1) \in S_{p,p'}$. Symmetrically one can determine the state q' that is assigned to v_2 in T'. The state that is assigned to v can then be read from the transition function of $\mathcal{A}$.

Another bottom-up tree automaton can "evaluate" T_P in the natural way. By Corollary 6.8, an $\mathsf{FO}(\leq, +, \times)$ update formula can determine the state that this automaton assigns to every inner node, and in particular the state $q' \in Q$ that it assigns to the root of T_p. From this state, the update formula can determine the state q that $\mathcal{A}$ assigns to the root r of T', and therefore can check whether $T' \in L$. The auxiliary relations of the form $S_{p,q}$ can be updated similarly. $\square$

6.3 The linear algebra approach for bulk changes

The results of the previous section are obtained by lifting the known dynamic programs from [Dong and Su 1998] and [Gelade et al. 2012] towards changes of polylogarithmic size, using basically the same auxiliary relations. The important idea that makes these results possible is that restricted second-order logics can express the respective queries, and even all NL-computable queries, and that these logics can be simulated in $\mathsf{FO}(\leq, +, \times)$ on substructures of polylogarithmic size.

This way we obtained maintainability results for undirected reachability and the word problem for regular languages and regular tree languages that are optimal for $\mathsf{DynFO}(\leq, +, \times)$ with respect to the size of the change. For the directed reachability query Reach, we could only give a dynamic program for insertions of polylogarithmic size. This dynamic program, as constructed in the proof of Theorem 6.9, does not even support single-edge deletions. Naturally, we wonder whether Reach can also be maintained in $\mathsf{DynFO}(\leq, +, \times)$ under insertions *and* deletions of polylogarithmically many edges.

The approach of Section 6.2 does not seem to be applicable to the dynamic program from [Datta et al. 2018a] that maintains Reach under single-edge changes. In [Datta et al. 2018a], the problem to determine whether there is a directed path from some node s to some node t in a graph G is reduced to the question whether a certain matrix has full rank. In [Datta et al. 2018b], a different approach is used to show that Reach can be maintained in $\mathsf{DynFO}(\leq, +, \times)$ under insertions and deletions of non-constant size, namely of size $\frac{\log n}{\log \log n}$, where n is the number of nodes. The authors show that using the (appropriately extended) Muddling technique, the approach for maintaining Reach under single-edge changes from [Datta et al. 2018a] can be reformulated as a reduction to maintaining the inverse of a matrix. They then show that the inverse of a matrix of dimension $\frac{\log n}{\log \log n} \times \frac{\log n}{\log \log n}$ over an appropriate field can be determined in $\mathsf{FO}(\leq,+,\times)$. The Sherman-Morrison-Woodbury formula (cf. [Henderson and Searle 1981], see Equation 6.1) states that the inverse of an $n \times n$ matrix can be updated after changes of k entries on the basis of the inverse of a $k \times k$ matrix. It then can be concluded that $(\textsc{Reach}, \Delta_E[\frac{\log n}{\log\log n}])$ is $(\mathsf{AC}^2, \log^2 n)$-maintainable.

We propose the following variant of the Muddling Theorem 3.3 for bulk changes.

Theorem 6.12 (Muddling Theorem for bulk changes)**.** *Let $f, g \in \log^{\mathcal{O}(1)} n$ be first-order constructible functions. Let q be a NL-computable, almost domain independent query, and Δ a set of single-tuple change operations. If $(q, \Delta[g(n)])$ is $(\mathsf{AC}[f(n)], f(n))$-maintainable, then $(q, \Delta[g(n)]) \in \mathsf{DynFO}(\leq,+,\times)$.*

Proof sketch. The proof follows the proof of Theorem 3.3. The only difference is how the query result is given during the first $f(n)$ change steps. Up to step $f(n)$, the active domain of the input structure contains $\mathcal{O}(f(n)g(n))$ elements, which is bounded by $\log^d n$ for some $d \in \mathbb{N}$ that only depends on f, g and Δ. So, by Corollary 6.8 an $\mathsf{FO}(\leq,+,\times)$ formula can express the query result. ☐

Theorem 6.12 together with the insight that $(\textsc{Reach}, \Delta_E[\frac{\log n}{\log\log n}])$ is $(\mathsf{AC}^2, \log^2 n)$-maintainable proves the main result of [Datta et al. 2018b].

Theorem 6.13 ([Datta et al. 2018b])**.** *The dynamic query $(\textsc{Reach}, \Delta_E[\frac{\log n}{\log\log n}])$ is in $\mathsf{DynFO}(\leq,+,\times)$.*

Actually, the result of [Datta et al. 2018b] is stronger, as it allows not only insertions and deletions of $\frac{\log n}{\log\log n}$ edges, but insertions and deletions that affect $\frac{\log n}{\log\log n}$ nodes.

This result can in fact be extended to bulk changes operations that can modify more than $\frac{\log n}{\log\log n}$ edges. Let us for example consider a change operation $\textsc{clear-node}(p)$ that deletes all incident edges of a given node.

Theorem 6.14. *The dynamic query $(\textsc{Reach}, \{\textsc{ins}_E, \textsc{del}_E, \textsc{clear-node}\}[\frac{\log n}{\log\log n}])$ is in $\mathsf{DynFO}(\leq,+,\times)$.*

Proof. In [Datta et al. 2018b] it is shown that $\textsc{Reach}$ is $(\mathsf{AC}^2, \log^2 n)$-maintainable under edge changes that affect $\mathcal{O}(\frac{\log n}{\log\log n})$ nodes. So, let $\mathcal{A}$ be an AC^2-algorithm with depth $\ell \log^2 n$ for some constant ℓ and $\mathcal{P}$ a dynamic program that maintains $\textsc{Reach}$ for $\log^2 n$ of these changes starting from auxiliary relations initialised by $\mathcal{A}$.

We show that from $\mathcal{A}$ and $\mathcal{P}$ we can construct an AC^2-algorithm $\mathcal{A}'$ and a dynamic program $\mathcal{P}'$ that witness that $\textsc{Reach}$ is still $(\mathsf{AC}^2, \log^2 n)$-maintainable when additionally $\textsc{clear-node}$ changes for $\frac{\log n}{\log\log n}$ nodes are allowed. It then follows from Theorem 6.12 that $\textsc{Reach}$ can be maintained in $\mathsf{DynFO}(\leq,+,\times)$ under these changes.

Let $G = (V, E)$ be the input graph, with $|V| = n$. The main idea is to maintain the reachability information in a slightly modified graph G' where each $\textsc{clear-node}$ change can be simulated by a constant number of edge insertions and deletions. The graph G' is defined as follows: for every node v of G, G' contains the nodes $v_1^{\text{in}}, \dots, v_{\log^2 n}^{\text{in}}$ that will receive the incoming edges of v, and the nodes $v_1^{\text{out}}, \dots, v_{\log^2 n}^{\text{out}}$ that will hold the outgoing edges from v. At each time, for every node v only one pair v_i^{in} and v_i^{out} is actually used, in the sense that they are connected by an edge to v. We call these nodes the *active gateways* for v. Initially, the nodes v_1^{in} and v_1^{out} are the active gateways of v, and the edges (v_1^{in}, v) and (v, v_1^{out}) are present in G', for every $v \in V$. Every edge (u, w) in G is represented by the edge $(u_1^{\text{out}}, w_1^{\text{in}})$ in G'.

The initialisation $\mathcal{A}'$ just computes G' and applies $\mathcal{A}$ to this graph. The graph G' has $n' \stackrel{\text{def}}{=} n \log^2 n$ many nodes, so the depth of $\mathcal{A}'$ is $\ell \log^2 n' = \ell \log^2(n \log^2 n)$ which is bounded by $2\ell \log^2 n$ for sufficiently large n. So, $\mathcal{A}'$ is indeed an AC^2-algorithm.

It remains to explain $\mathcal{P}'$. For edge insertions and deletions, $\mathcal{P}'$ just simulates $\mathcal{P}$ in the obvious way for the corresponding changes on G', using the unique active gateways of the nodes that are affected by the change. For CLEAR-NODE changes, we observe that every change CLEAR-NODE(v) for a node v in G can be simulated by a constant number of edge deletions and insertions in G': if v_i^{in} and v_i^{out} are the active gateways of v, it suffices to delete the edges between v, v_i^{in} and v_i^{out} and to insert edges (v_{i+1}^{in}, v) and $(v, v_{i+1}^{\text{out}})$. Because at most $\log^2 n$ changes CLEAR-NODE(v) can occur for each node v during a change sequence of length $\log^2 n$, enough "fresh" gateways are available. So, whenever CLEAR-NODE changes for $\frac{\log n}{\log \log n}$ nodes occurs, $\mathcal{P}'$ only needs to simulate $\mathcal{P}$ for the corresponding $\mathcal{O}(\frac{\log n}{\log \log n})$ edge insertions and deletions. In all cases, $\mathcal{P}'$ also applies the changes to G'.

It is clear from the construction that at all times there is a path from some node u to some node v in G if and only if such a path also exists in G'. Because $\mathcal{P}$ maintains REACH in G' for change sequences of length $\log^2 n$, so does $\mathcal{P}'$ in G. □

In the same way, also change operations that activate and deactivate nodes, as used by Kähler and Wilke [2003], can be supported.

In the remainder of this section we apply the ideas from the proof of Theorem 6.13 to the DISTANCE query, which asks, given a directed graph G, for the length of the shortest path between two nodes s and t. More formally, the query result DISTANCE(G) is a ternary relation with $(s, t, \ell) \in$ DISTANCE(G) if and only if $(s, t) \in$ REACH(G) and the shortest path from s to t consists of ℓ edges.[4]

It is not even known whether DISTANCE can be maintained in DynFO under single-edge changes. Hesse [2003a] shows that (REACH, Δ_E) is in $\mathsf{DynFO{+}Maj}(\leq, +, \times)$, the extension of $\mathsf{DynFO}(\leq, +, \times)$ that is defined via $\mathsf{FO{+}Maj}(\leq, +, \times)$ update formulas. These formulas may use majority quantifiers $\text{Maj}(x)\, \varphi(x)$, and such a formula is satisfied in a structure $\mathcal{D}$ with domain D, if $\mathcal{D} \models \varphi(a)$ holds for more than half of the elements $a \in D$. It is well-known that $\mathsf{FO{+}Maj}(\leq, +, \times)$ corresponds to the circuit complexity class uniform TC^0 [Barrington et al. 1990], which is defined analogously to AC^0, but additionally allows for majority gates.

The dynamic program from [Hesse 2003a] actually witnesses that (DISTANCE, $\Delta_E) \in \mathsf{DynFO{+}Maj}(\leq, +, \times)$. In the following subsections we generalise this result in two directions. First we show that DISTANCE can still be maintained in $\mathsf{DynFO{+}Maj}(\leq, +, \times)$ under edge changes of polylogarithmic size. Then we show, towards a DynFO maintenance

[4]We suppose that the node set V of G is linearly ordered, so we can identify nodes with numbers from $\{0, \ldots, |V| - 1\}$.

result for the distance query, that $\mathsf{DynFO}(\leq,+,\times)$ can maintain auxiliary relations under edge changes of size $\frac{\log n}{\log\log n}$ such that a $\mathsf{FO}{+}\mathsf{Maj}(\leq,+,\times)$ formula can extract the query result for the distance query. We start be introducing the necessary notions from linear algebra.

6.3.1 Basic concepts from linear algebra

In the following subsections we use matrix operations and standard results from linear algebra. An $n \times n$ matrix M of natural numbers up to n is formally represented by a ternary relation M with $(i, j, m) \in M$ if the entry in the i-th row and the j-column of M is m. We denote this entry by $[M]_{ij}$. Analogously, matrices of dimension $n^k \times n^k$ with polynomial entries up to n^k, for some fixed $k \in \mathbb{N}$, are represented by $3k$-ary relations. Numbers m up to size $2^{n^k} - 1$ can be represented by a k-ary relation that includes a tuple $\bar{a}$ if the $\bar{a}$-th bit in the binary representation of m is 1; here we view $\bar{a}$ as a number between 1 and n^k. So, we can represent matrices with entries of exponential size in n. Matrices with entries from sets other than $\mathbb{N}$ can be represented in a similar fashion.

Let $\mathbb{Z}$ denote the integers. By $\mathbb{Z}[[x]]$ we denote the ring of formal power series over $\mathbb{Z}$, that is, the ring with elements $\sum_i^\infty c_i x^i$, with $c_i \in \mathbb{Z}$ for each i, and the natural addition and multiplication. An element $\sum_i^\infty c_i x^i \in \mathbb{Z}[[x]]$ is *normalised* if $c_0 = 1$. All normalised elements of $\mathbb{Z}[[x]]$ have an inverse, see [Godsil 1993, Lemma 3.3.1]. The ring $\mathbb{Z}[[X]]$ can be embedded into the field of fractions $\mathbb{Z}((x)) \stackrel{\text{def}}{=} \{\frac{g(x)}{h(x)} \mid g(x), h(x) \in Z[[x]] \text{ and } h(x) \neq 0\}$. We denote the subring of $\mathbb{Z}[[x]]$ consisting of all finite polynomials by $\mathbb{Z}[x]$, and the field of fractions of elements from $\mathbb{Z}[x]$ by $\mathbb{Z}(x)$.

A matrix $A \in \mathbb{Z}[[x]]^{n\times n}$ is invertible over $\mathbb{Z}[[x]]$ if there is a matrix $B \in \mathbb{Z}[[x]]^{n\times n}$ with $AB = I$, where I is the identity matrix with 1 on its diagonal and 0 anywhere else. Given an invertible matrix A, the inverse A^{-1} of A can be computed by $[A^{-1}]_{ij} = (-1)^{i+j}\frac{\det A_{ji}}{\det A}$, where the matrix C_{ji} results from a matrix C by removing the j-th row and the i-th column, and $\det C$ denotes the determinant of C. Therefore, a matrix $A \in \mathbb{Z}[[x]]^{n\times n}$ is invertible over $\mathbb{Z}((x))$ if its determinant is non-zero, and it is invertible over $\mathbb{Z}[[x]]$ if the constant term of $\det(A)$ is 1. If A is of the form $I + xC$ for some matrix $C \in \mathbb{Z}[[x]]^{n\times n}$ then A is invertible over $\mathbb{Z}[[x]]$ as $\det(I + xC)$ is normalised.

For a polynomial $g(x) \in \mathbb{Z}[x]$ we abbreviate its degree by $\deg g(x)$ and write $\|g(x)\|$ for the value of its largest coefficient. The degree of a representation $\frac{g(x)}{h(x}$ of an element of $\mathbb{Z}(x)$ is the maximum of the degrees of $g(x)$ and $h(x)$, and similarly for the largest coefficient. Degree and maximal coefficient are defined similarly for matrices over $\mathbb{Z}[x]$ and $\mathbb{Z}(x)$.

6.3.2 Distances and update formulas that use majority

The goal of this subsection is to extend a result from [Hesse 2003a] to changes of polylogarithmically many edges, leading to the following theorem.

Theorem 6.15. *The dynamic query* $(\textsc{Distance}, \Delta_E[\log^d n])$ *is in* $\mathsf{DynFO}{+}\mathsf{Maj}(\leq,+,\times)$*, for every* $d \in \mathbb{N}$*.*

We follow the approach of Hesse [2003a] and use formal power series for counting the number of paths of each length. Fix a graph G with adjacency matrix $\text{Ad}(G) \in \mathbb{Z}^{n\times n}$ and a formal variable x. Then $D \overset{\text{def}}{=} \sum_{i=0}^{\infty}(x\text{Ad}(G))^i$ is a matrix of formal power series from $\mathbb{Z}[[x]]$ such that if $[D]_{st} = \sum_{i=0}^{\infty} c_i x^i$ then c_i is the number of paths from s to t of length i. In particular, the distance between s and t is the smallest i such that c_i is non-zero. Note that if such an i exists, then $i < n$.

The matrix D is invertible over $\mathbb{Z}[[x]]$ and can be written as $(I - x\text{Ad}(G))^{-1}$ (cf. [Godsil 1993, Example 3.6.1]). The maintenance of distances thus reduces to maintaining for a matrix $A \in \mathbb{Z}[[x]]$ the smallest $i < n$ such that the i-th coefficient of $[A^{-1}]_{st}$ is non-zero, for each pair $s, t \leq n$. In particular, Theorem 6.15 is a consequence of the following theorem.

Theorem 6.16. *Suppose $A \in \mathbb{Z}[[x]]^{n\times n}$ stays invertible over $\mathbb{Z}[[x]]$. For all $s, t \in [n]$ one can maintain the smallest $i < n$ such that the i-th coefficient of the st-entry of A^{-1} is non-zero in* DynFO+Maj$(\leq, +, \times)$ *under changes of* $\log^d n$ *entries, for each fixed $d \in \mathbb{N}$.*

We gradually explain the idea. When the matrix A is changed to $A + \Delta A$, where ΔA has $\log^d n$ non-zero entries, then one can decompose the change matrix ΔA into a matrix product UBV where the matrices U, B, V have dimension $n \times \log^d n$, $\log^d n \times \log^d n$ and $\log^d n \times n$, respectively, by the following lemma.

Lemma 6.17 ([Datta et al. 2018b, Lemma 7])**.** *Fix a ring R. Suppose $M \in R^{n\times n}$ with non-zero rows $r_{i_1}, \dots, r_{i_k}$ and columns $c_{j_1}, \dots, c_{j_k}$. Then $M = UBV$ with $U \in R^{n\times k}$, $B \in R^{k\times k}$, and $V \in R^{k\times n}$ where*

1. *B is obtained from M by removing all-zero rows and columns.*
2. $U = \begin{pmatrix} \bar{u}_1 \\ \vdots \\ \bar{u}_n \end{pmatrix}$ *where* $\bar{u}_i = \begin{cases} \bar{0}^T & \text{if } i \notin \{i_1, \dots, i_k\} \\ \bar{e}_m^T & \text{if } i = i_m \end{cases}$
3. $V = \begin{pmatrix} \bar{v}_1, \dots, v_n \end{pmatrix}$ *where* $\bar{v}_j = \begin{cases} \bar{0} & \text{if } j \notin \{j_1, \dots, j_k\} \\ \bar{e}_m & \text{if } j = j_m \end{cases}$

Here, $\bar{e}_m$ denotes the m-th unit vector.

The following equation, called here the Sherman-Morrison-Woodbury identity, describes how the inverse of a matrix can be updated after a change that is characterised by a matrix product UBV [Henderson and Searle 1981, Equation 23]:

$$(A + UBV)^{-1} = A^{-1} - A^{-1}U(I + BVA^{-1}U)^{-1}BVA^{-1} \tag{6.1}$$

Note that if the involved matrices are from $\mathbb{Z}[[x]]$ then the inverse in general is from the field $\mathbb{Z}((x))$ of fractions of formal power series. For the proof of Theorem 6.16 the resulting inverses are in $\mathbb{Z}[[x]]$ by assumption.

In the remainder of this subsection we explain how the update described by Equation 6.1 can be "implemented" in FO+Maj$(\leq, +, \times)$.

Of course, computing with inherently infinite formal power series is not possible in DynFO+Maj$(\leq, +, \times)$. However, as stated in Theorem 6.16, in the end we are only interested in the first $i < n$ coefficients of a power series. We therefore show that it suffices to truncate all occurring power series at the n-th term and use FO+Maj$(\leq, +, \times)$'s ability to define iterated sums and products of polynomials [Hesse et al. 2002].

Formally, we have to show that no precision for the first $i < n$ coefficients is lost when computing with truncated power series. This motivates the following definition. A formal power series $g(x) = \sum_i c_i x^i \in \mathbb{Z}[[x]]$ is an *m-approximation* of a formal power series $h(x) = \sum_i d_i x^i \in \mathbb{Z}[[x]]$, denoted by $g(x) \approx_m h(x)$, if $c_i = d_i$ for all $i \leq m$. This notion naturally extends to matrices over $\mathbb{Z}[[x]]$: a matrix $A \in \mathbb{Z}[[x]]^{\ell \times k}$ is an m-approximation of a matrix $B \in \mathbb{Z}[[x]]^{\ell \times k}$ if each entry of A is an m-approximation of the corresponding entry of B. The notion of m-approximation is preserved under all arithmetic operations that will be relevant.

Lemma 6.18. *Fix an $m \in \mathbb{N}$.*

(1) Suppose $g(x), g'(x), h(x), h'(x) \in \mathbb{Z}[[x]]$ with $g(x) \approx_m g'(x)$ and $h(x) \approx_m h'(x)$. Then

(i) $g(x) + h(x) \approx_m g'(x) + h'(x)$,

(ii) $g(x)h(x) \approx_m g'(x)h'(x)$, and

(iii) $\frac{1}{g(x)} \approx_m \frac{1}{g'(x)}$ whenever $g(x)$ and $g'(x)$ are normalised.

(2) Suppose $A, A', B, B' \in \mathbb{Z}[[x]]^{n \times n}$ with $A \approx_m A'$ and $B \approx_m B'$. Then

(i) $A + B \approx_m A' + B'$,

(ii) $AB \approx_m A'B'$,

(iii) If A is invertible over $\mathbb{Z}[[x]]$ then so is A', and $A^{-1} \approx_m A'^{-1}$.

Remember that a formal power series is normalised if its constant term is 1.

Proof. The first two parts of (1) are straightforward. For the last part, first observe that $\frac{1}{g(x)}$ and $\frac{1}{g'(x)}$ actually exist, as $g(x)$ and $h(x)$ are normalised. Suppose that $\frac{1}{g(x)} = \sum_i d_i x^i$ and $\frac{1}{g'(x)} = \sum_i d'_i x^i$. We write $g(x)$ and $g'(x)$ as $g(x) = \sum_{i=0}^{m} c_i x^i + r(x)$ and $g'(x) = \sum_{i=0}^{m} c_i x^i + r'(x)$ where the formal power series $r(x)$ and $r'(x)$ are divisible by x^{m+1} (and therefore the coefficient of x^{m+1} is the first that is allowed to be non-zero). Because $g(x)\frac{1}{g(x)} = 1$, we have that $c_0 d_0 = 1$ and $\sum_{i=0}^{j} c_i d_{j-i} = 0$ for all $j \in [m]$. Similarly, $c_0 d'_0 = 1$ and $\sum_{i=0}^{j} c_i d'_{j-i} = 0$ hold. Solving both systems of equations yields $d_i = d'_i$ for $i \in [m]_0$.

The first two parts of (2) follow immediately from (1). For the third part, recall that A is invertible over $\mathbb{Z}[[x]]$ if and only if $\det(A)$ is non-zero and normalised. This translates to the matrix A', because $A \approx_m A'$. Furthermore

$$[A^{-1}]_{st} = (-1)^{s+t}\frac{\det(A_{ts})}{\det(A)} \approx_m (-1)^{s+t}\frac{\det(A'_{ts})}{\det(A')} = [A'^{-1}]_{st} \qquad \square$$

An approximation of the inverse of a matrix $A \in \mathbb{Z}[[x]]^{n \times n}$ can be updated using Equation 6.1.

Proposition 6.19. *Suppose $A \in \mathbb{Z}[[x]]^{n\times n}$ is invertible over $\mathbb{Z}[[x]]$, and $C \in \mathbb{Z}[[x]]^{n\times n}$ is an m-approximation of A^{-1}. If $A + \Delta A$ is invertible over $\mathbb{Z}[[x]]$ and $\Delta A = UBV$ with $U \in \mathbb{Z}[[x]]^{n\times k}, B \in \mathbb{Z}[[x]]^{k\times k}$ and $V \in \mathbb{Z}[[x]]^{k\times n}$ for some $k \leq n$, then*

$$(A + \Delta A)^{-1} \approx_m C - CU(I + BVCU)^{-1}BVC$$

Proof. This follows immediately from Equation 6.1 and Lemma 6.18. □

The application of Equation 6.1 for Theorem 6.16 involves inverting $\log^d n \times \log^d n$ matrices, which reduces to computing the determinant of such matrices. We show that this is possible in $\mathsf{FO{+}Maj}(\leq, +, \times)$ for matrices of polynomials.

Lemma 6.20. *Fix a domain of size n and let $k = \log^d n$, for some fixed $d \in \mathbb{N}$. The determinant of a matrix $A \in \mathbb{Z}[x]^{k\times k}$, with entries of polynomial degree in n, can be defined in $\mathsf{FO{+}Maj}(\leq, +, \times)$.*

Proof. We show that the value can be computed in uniform TC^0, which is as powerful as $\mathsf{FO{+}Maj}(\leq, +, \times)$ [Barrington et al. 1990].

Computing the determinant of an $k \times k$ matrix is equivalent to computing the iterated matrix product of k matrices [Cook 1985]. The reduction produces k matrices of dimension at most $(k + 1) \times (k + 1)$ and can be computed in uniform TC^0, as can be seen implicitly in [Mahajan and Vinay 1999, p. 482]. Thus, the lemma statement follows from the fact that iterated products of matrices $A_1, \ldots, A_k \in \mathbb{Z}[x]^{k\times k}$ with $k = \log^d n$ can be computed in uniform TC^0, which can be proven in the spirit of [Agrawal and Vinay 2008, p. 69].

For the sake of completeness we outline the proof. We first explain how $\sqrt{\log n}$ such matrices can be multiplied. Each entry in such a product is the sum of $(\log^d n)^{\sqrt{\log n}}$ many products of $\sqrt{\log n}$ polynomials. Such products can be computed in uniform TC^0 due to [Hesse et al. 2002, Corollary 6.5]. The sum can be computed in TC^0 as it is over at most polynomially many terms:

$$(\log^d n)^{\sqrt{\log n}} = 2^{\log((\log^d n)^{\sqrt{\log n}})} = 2^{d\sqrt{\log n}\log\log n} \in 2^{\mathcal{O}(\sqrt{\log n}\sqrt{\log n})} = n^{\mathcal{O}(1)}$$

The idea for computing the product of $A_1, \ldots, A_k$ with $k = \log^d n$ is to partition the sequence into fragments of length $\sqrt{\log n}$ each. The product B_i of the matrices of the i-th fragment can be computed by the procedure from above, and there are $\log^{d-\frac{1}{2}} n$ such fragments. This procedure can now be repeated recursively, that is, the sequence B_i is partitioned into fragments of length $\sqrt{\log n}$, and so on. After $2d$ repetitions, the final product is obtained. □

Finally we turn to the proof of Theorem 6.16.

Proof (of Theorem 6.16). The dynamic program maintains an n-approximation $C \in \mathbb{Z}[x]^{n\times n}$ of A^{-1} that truncates A^{-1} at degree n. When A is updated to $A + \Delta A$ then:

1. ΔA is decomposed into suitable $U \in \mathbb{Z}[x]^{n\times \log^d n}, B \in \mathbb{Z}[x]^{\log^d n\times\log^d n}$, and $V \in \mathbb{Z}[x]^{\log^d n\times n}$;
2. C is updated via $C' \overset{\text{def}}{=} C - CU(I + BVCU)^{-1}BVC$;

3. All entries of C' are truncated at degree n.

The steps can be defined in FO+Maj$(\leq, +, \times)$ due to Lemma 6.17, Lemma 6.20, and the fact that iterated addition and multiplication of polynomials can be defined in FO+Maj$(\leq, +, \times)$, see [Hesse et al. 2002]. The maintained matrix C is indeed an n-approximation of A^{-1} due to Proposition 6.19. □

6.3.3 Towards distances in **DynFO**

It is still open whether Distance can be maintained in DynFO under single-edge changes, let alone under changes of non-constant size. In this subsection we combine ideas from the proofs of Theorem 6.13 and Theorem 6.15 to show that, even under changes of non-constant size, DynFO$(\leq, +, \times)$ can maintain auxiliary relations from which a FO+Maj$(\leq, +, \times)$ formula can extract the result for the Distance query.

Theorem 6.21. *The query result for* Distance *can be defined by a* FO+Maj$(\leq, +, \times)$ *query from auxiliary relations that can be maintained in* DynFO$(\leq, +, \times)$ *under changes of* $\frac{\log n}{\log \log n}$ *edges, where* n *is the number of nodes of the graph.*

The general proof approach is again via maintenance of matrix inverses, just as for the proof of Theorem 6.15. We establish the following theorem, which is a variation of Theorem 6.16. In particular, the described auxiliary relations can be maintained for $\log^2 n$ many change steps after an AC^2-computable initialisation, so Theorem 6.21 follows by the reduction explained in the previous subsection and Theorem 6.12.

Theorem 6.22. *Suppose* $A \in \mathbb{Z}[[x]]^{n \times n}$ *stays of the form* $I - xA'$ *for some* $A' \in \{0,1\}^{n \times n}$ *and is thus invertible over* $\mathbb{Z}[[x]]$*. After an* AC^2*-computable initialisation to an arbitrary initial matrix of the stated form, one can maintain in* DynFO$(\leq, +, \times)$ *auxiliary relations under changes of* $\frac{\log n}{\log \log n}$ *entries, from which for all* $s, t \in [n]$ *the smallest* $i < n$ *such that the* i*-th coefficient of the* st*-entry of* A^{-1} *is non-zero can be defined in* FO+Maj$(\leq, +, \times)$.

The proof strategy is the same as for Theorem 6.16. However, here it does not suffice to just truncate formal power series and maintain approximated polynomials, as the involved polynomials have polynomial degree and coefficients with exponential values in n. Consequently, the update formulas in the proof of Theorem 6.16 need to multiply $\mathcal{O}(n)$-bit numbers and express the sum of polynomially many of these numbers, which both are possible in FO+Maj$(\leq, +, \times)$ but not in FO$(\leq, +, \times)$.

Therefore our goal is to maintain *implicit* representations of n-approximations $C(x)$ of the entries from A^{-1} from which the smallest non-zero term of each of the entries can be extracted. The idea is to store and update the evaluation $C(a)$ for several numbers $a \in \mathbb{N}$. If the entries of $C(x)$ have small degree, then the smallest non-zero term can be extracted from this via Cauchy interpolation, which is possible in FO+Maj$(\leq, +, \times)$ [Hesse et al. 2002]. However, when only storing $C(x)$ implicitly via $C(a)$ it is not possible to truncate the polynomials after each step, as in the proof of Theorem 6.16. Furthermore, the update formula for $C(x)$ provided by Proposition 6.19 does not ensure that polynomials keep a small degree if they are not truncated.

For this reason we proceed as follows. We first introduce a representation where each entry of $C(x)$ is represented by a fraction $\frac{g(x)}{h(x)}$ such that $g(x)$ and $h(x)$ are polynomials of degree $\mathcal{O}(n^c)$ for some c. The program then stores $g(a)$ and $h(a)$ for several numbers $a \in \mathbb{N}$. Actually the numbers $g(a)$ and $h(a)$ might be very large, indeed exponential in n, and therefore we will store all numbers in Chinese remainder representation.

Next we prepare by proving several intermediate propositions and lemmas that will ensure the correctness of this course of action. Afterwards we prove Theorem 6.22 by presenting the dynamic program in detail.

We start by introducing a representation of the matrix $C(x)$ in terms of fractions. A *quotient m-approximation* of a formal power series $f(x)$ is a fraction $\frac{g(x)}{h(x)} \in \mathbb{Z}(x)$ with normalised $h(x)$ such that $\frac{g(x)}{h(x)} \approx_m f(x)$ when $\frac{g(x)}{h(x)}$ is treated as a formal power series. Quotient m-approximations for matrices are defined analogously. Using this representation and a specific way to apply Equation 6.1, see Algorithm 1, we bound the degree and the size of the coefficients in an updated m-approximation.

Proposition 6.23. *Suppose $A \in \mathbb{Z}[[x]]^{n\times n}$ is invertible over $\mathbb{Z}[[x]]$, and $C \in \mathbb{Z}(x)^{n\times n}$ is a quotient m-approximation of A^{-1}. If $A + \Delta A$ is invertible over $\mathbb{Z}[[x]]$ and ΔA can be written as UBV with $U \in \mathbb{Z}[x]^{n\times k}, B \in \mathbb{Z}[x]^{k\times k}$, and $V \in \mathbb{Z}[x]^{k\times n}$ for some $k \leq n$, then*

$$C' \overset{def}{=} C - CU(I + BVCU)^{-1}BVC$$

is an m-approximation of $(A + \Delta A)^{-1}$. Furthermore, if C' is computed with Algorithm 1, B is of the form xB^* for some $B^* \in \mathbb{Z}[x]^{k\times k}$ that takes only values from $\{-1, 0, 1\}$, and U, V take only values from $\{0, 1\}$, then

(1) $\deg C' \in \mathcal{O}(k^3 \deg C)$ *and* $\|C'\| \in (\|C\|k \deg C)^{\mathcal{O}(k^3)}$, *and*

(2) if the denominators in C are normalised then the denominators of C' are normalised as well.

Proof. That C' is an m-approximation of $(A + \Delta A)^{-1}$ follows from Proposition 6.19 and the assumption that C is a quotient m-approximation. If all denominators of C are normalised and B is of the form xB^*, an inspection of Algorithm 1 yields that also all denominators in C' are normalised.

It remains to show the bounds on $\deg C'$ and $\|C'\|$. Note that for polynomials $h_1, \ldots, h_k \in \mathbb{Z}[x]$ one has $\deg(\sum_i h_i) = \max_i \deg(h_i)$ and $\deg(\prod_i h_i) \in \mathcal{O}(\sum_i \deg(h_i))$ when $\sum_i h_i$ and $\prod_i h_i$ are computed in the naive way. Hence in Algorithm 1, E has degree $\mathcal{O}(\deg C)$ and $f(x)$ has degree $\mathcal{O}(k^2 \deg C)$, and therefore $\det(f(x)E)$ and $(I+BVCU)^{-1}$ have degree $\mathcal{O}(k^3 \deg C)$. It follows that $\deg C' \in \mathcal{O}(k^3 \deg C)$.

The estimation of $\|C'\|$ is similar, using the facts that $\|\sum_i h_i\| \in \mathcal{O}(\sum_i \|h_i\|)$ and $\|\prod_i h_i\| \in \mathcal{O}(\prod_i \deg(h_i) \prod_i \|h_i\|)$. We observe that $\|E\| \in \mathcal{O}(\|C\|)$ and $\|f(x)\| \in \mathcal{O}(\|C\|^{k^2} k^2 \deg C) \subseteq (\|C\| \deg C)^{\mathcal{O}(k^2)}$, which is still true for $\|f(x)E\|$, and therefore $\|\det(f(x)E)\| \in (\|C\|k \deg C)^{\mathcal{O}(k^3)}$. It follows that the same bound holds for $\|C'\|$. □

Our dynamic program will maintain an implicit representation of the matrix C from the previous theorem. For extracting the smallest non-zero terms it suffices to look at the

Algorithm 1 Updating a quotient m-approximation

```
Input: Matrices $C \in \mathbb{Z}(x)^{n\times n}, U \in \mathbb{Z}[x]^{n\times k}, B \in \mathbb{Z}[x]^{k\times k}$, and $V \in \mathbb{Z}[x]^{k\times n}$
Output: Matrix $C' \overset{\text{def}}{=} C - CU(I + BVCU)^{-1}BVC$
1: Compute $E = I + BVCU \in \mathbb{Z}(x)^{k\times k}$
2: Compute the product $f(x)$ of all denominators of $E$.
3: for all $i, j \leq k$ do
4:     Compute the $ij$-th entry of the inverse $E^*$ of $f(x)E$ as $(-1)^{i+j}\frac{\det(f(x)E_{ji})}{\det(f(x)E)}$.
5: Compute the inverse of $E$ as $f(x)E^*$.
6: Compute $C'$.
```

numerators of C, as long as the denominators are normalised, as given by the following lemma.

Lemma 6.24. *Suppose $\frac{g(x)}{h(x)} \in \mathbb{Z}(x)$ for some $g(x) = \sum_i c_i x^i$ and normalised $h(x)$, and that $\frac{g(x)}{h(x)} = \sum_i d_i x^i$. Then i is the smallest number such that $c_i \neq 0$ if and only if it is the smallest number such that $d_i \neq 0$.*

Proof. Suppose $h(x) = \sum_i e_i x^i$ with $e_0 = 1$. Then $c_0 = e_0 d_0$ and hence $c_0 = 0$ if and only if $d_0 = 0$. Further, $c_i = \sum_{j=0}^{i} e_j d_{i-j}$. Hence, if $c_\ell = d_\ell = 0$ for all $\ell < i$ then $c_i = 0$ if and only if $d_i = 0$. ☐

Instead of working with quotient approximations directly, the dynamic program will maintain in $\mathsf{DynFO}(\leq, +, \times)$ evaluations of numerators and denominators under several numbers from $a \in \mathbb{N}$. By interpolating the polynomials we can extract the smallest non-zero term from this representation in $\mathsf{FO{+}Maj}(\leq, +, \times)$. That is even true if the evaluated values are stored in a Chinese remainder representation, that is, the evaluation is done modulo several primes.

Lemma 6.25. *For all numbers $d, d', e \in \mathbb{N}$ with $d' > d$ there are numbers $e', b \in \mathbb{N}$ such that the following is true. Fix a domain of size n. Suppose $g(x) = \sum_i c_i x^i \in \mathbb{Z}[x]$ with $\deg g(x) \leq n^d$ and $\|g(x)\| \leq 2^{n^e}$. Let $S \subseteq [n^{d'}]_0 \subseteq \mathbb{N}$ with $|S| \geq n^d + 1$, and P a set of $n^{e'}$ primes (among the first n^b numbers). The smallest i such that $c_i \neq 0$ can be defined in $\mathsf{FO{+}Maj}(\leq, +, \times)$ from a relation that stores the value $g(a) \pmod p$ for each $a \in S$ and each $p \in P$.*

Proof. The value $g(a)$ with $a \leq n^{d'}$ is bounded by $2^{n^{e'}}$ for some e' that only depends on d, d' and e. As the product of the primes in P exceeds this number, $g(a)$ is uniquely determined by the Chinese remainder representation given by the values $g(a) \pmod p$, and can be decoded in $\mathsf{FO{+}Maj}(\leq, +, \times)$ [Hesse et al. 2002, Theorem 4.1]. The statement follows from the fact that $g(x)$ of degree at most n^d is uniquely determined by the values $g(a)$ for $n^d + 1$ pairwise distinct data points a, and Cauchy interpolation is in $\mathsf{FO{+}Maj}(\leq, +, \times)$ [Hesse et al. 2002, Corollary 6.5]. ☐

We can finally proceed to the proof of Theorem 6.22.

Proof sketch (of Theorem 6.22). Suppose that the matrix $C \in \mathbb{Z}(x)$ is a normalised quotient n-approximation of A^{-1}. Then by Lemma 6.24, the smallest $i < n$ such that the i-th coefficient of the st-entry of A^{-1} is non-zero is equal to the smallest such i for the numerator of the st-entry of C. This i can be extracted from the relations stated in Lemma 6.25 by an $\mathsf{FO{+}Maj}(\leq, +, \times)$ formula.

Our goal is therefore to maintain relations that store the values from Lemma 6.25 for each entry of C. We will apply a variant of Theorem 6.12 and exhibit a dynamic program that maintains such relations for $k \overset{\text{def}}{=} \frac{\log n}{\log \log n}$ changes of size k each, starting from initial auxiliary relations that represent a normalised quotient n-approximation $C \in \mathbb{Z}(x)$ with numerators of degree at most n and denominator 1. More details on this part, and in particular why such a dynamic program implies that the relations can be maintained under change sequences of arbitrary length, are given towards the end of this sketch.

We show that the auxiliary relations can be maintained for $k = \frac{\log n}{\log \log n}$ change steps, when initial auxiliary data as described above is given. For a domain of size n, let $S \overset{\text{def}}{=} [n^\lambda]$ and let P be the set of the first n^μ primes, for numbers λ and μ to be determined later. The dynamic program implicitly maintains a quotient n-approximation $C \in \mathbb{Z}(x)$ of A^{-1} as follows. For each $a \in S$, each $p \in P$ and each entry (s,t) of C, it maintains $g(a) \pmod p$ and $h(a) \pmod p$ if $\frac{g(x)}{h(x)}$ is the st-th entry of C. Whenever $h(a) = 0 \pmod p$ then p is declared invalid for a; whenever $h(a) = 0$ for some denominator $h(x)$ and $a \in S$ then a is declared invalid. The set of primes valid for a value a is denoted by P_a.

The initial amount of values in S and in P is chosen such that after k changes, sufficiently many valid values remain in S and in each P_a in order to apply Lemma 6.25 for extracting the smallest non-zero coefficients of denominators from this implicit representation.

Suppose C is a quotient n-approximation of A^{-1} implicitly stored by the dynamic program. Then we denote by $\mathcal{C}_a \pmod p$ the evaluation of C at position a modulo prime p for valid a and p. By $\mathcal{C}_a$ we denote the tuple $(\mathcal{C}_a \pmod{p_1}, \dots, \mathcal{C}_a \pmod{p_\eta})$ where $p_1, \dots, p_\eta$ are the primes still valid for a. By $\mathcal{C}$ we denote the tuple $\mathcal{C}_{a_1}, \dots, \mathcal{C}_{a_\kappa}$ where $a_1, \dots, a_\kappa$ are valid numbers.

Formally the program uses a relation D that stores a tuple $(\bar{a}, \bar{p}, s, t, \bar{v})$, where $\bar{m}$ denotes the base-n encoding of a number m, if and only if (i) a is valid, (ii) p is valid for a, and (iii) v is the value of the denominator of the st-th entry modulo p. Similarly, a relation N stores the numerators. These relations encode $\mathcal{C}_a \pmod p$ as well as the sets S and P_a. In the following we abstain from using this formal perspective for the sake of clarity. The descriptions to follow can be easily translated to this formal framework.

When a change ΔA occurs, by the restrictions of Theorem 6.22 regarding the form of A, this change matrix has only entries from $\{-x, 0, x\}$. The dynamic program updates the sets S and P_a as well as the tuple $\mathcal{C}$ according to Algorithm 2.

The steps mainly consist of multiplying numbers that are polynomial in n (and thus can be encoded by $\mathcal{O}(\log n)$ bits) and addition of $k \in \mathcal{O}(\log n)$ of these numbers, which can be done in $\mathsf{FO}(\leq, +, \times)$, see for example [Hesse et al. 2002, Theorem 5.1]. The loops from Lines 2–4 translate to parallel evaluation in $\mathsf{FO}(\leq, +, \times)$.

Let us analyse the necessary amount of values in S and P. By Proposition 6.23(1), the degrees of denominators $h(x)$ of C grow by a factor k^3 after each change. Thus, after k

Algorithm 2 Updating the auxiliary data for Theorem 6.22

```
Input: A change $\Delta A$ with entries from $\{-x, 0, x\}$
1: Decompose $\Delta A$ into $UBV$ with $U \in \mathbb{Z}^{n\times k}, B \in \mathbb{Z}[x]^{k\times k}$, and $V \in \mathbb{Z}^{k\times n}$
         according to Lemma 6.17.
2: for each $a \in S$ do
3:     for each prime $p \in P_a$ do
4:         for all $(s,t) \in [k]^2$ do
5:             Compute $g_{st}(a) \pmod{p}$ and $h_{st}(a) \pmod{p}$
         where $\frac{g_{st}(x)}{h_{st}(x)}$ is the $st$-th entry of $(I + BVCU)^{-1}$, following Algorithm 1.
6:             If $h_{st}(a) = 0 \pmod{p}$ then remove $p$ from $P_a$
7:             If $P_a = \emptyset$ then remove $a$ from $S$
8:             If $P_a \neq \emptyset$ then compute $\mathcal{C}'_a \pmod{p}$
         according to $C - CU(I + BVCU)^{-1}BVC$ and following Algorithm 1.
```

change steps starting from fractions of polynomials with maximal degree n, the degree of the denominators is bounded by $\mathcal{O}(nk^{3k}) \subseteq \mathcal{O}(n^r)$ for some $r \in \mathbb{N}$. Therefore each denominator h evaluates to 0 for at most $\mathcal{O}(n^r)$ many $a \in S$. All in all there are at most $\mathcal{O}(n^{r+2})$ many $a \in S$ such that $h(a) = 0$ for some denominator h of C after one update step, and at most $\mathcal{O}(n^{r+3})$ such a in the course of k change steps.

By Proposition 6.23(2), the values of coefficients of C' are bounded by $(\|C\| k \deg C)^{\mathcal{O}(k^3)}$ after one change step. Thus, after k changes, the values $h(a)$ for $a \in S$ are bounded by $2^{\mathcal{O}(n^{s(r)})}$ for some $s \in \mathbb{N} \to \mathbb{N}$. Thus, if $h(a) \neq 0$ then $h(a) \equiv 0 \pmod{p}$ for at most $\mathcal{O}(n^{s(r)})$ many primes p and for a denominator $h(x)$ of C. All in all, at most $\mathcal{O}(n^{s(r)+2})$ many primes p are removed from P_a in Line 6 in one step. If $h(a) \neq 0$ after each of k many changes, then at most $\mathcal{O}(n^{s(r)+3})$ many primes p have been removed from P_a. If $h(a) = 0$ after some change, then $h(a) \equiv 0 \pmod{p}$ for all primes p and therefore a is removed from S in Line 7.

Thus, there exists numbers λ, μ such that if one starts with $S = [n^\lambda]_0$ and n^μ primes P, then after $k = \frac{\log n}{\log\log n}$ many changes there are still at least n^r numbers a in S, each of them with a set P_a of size at least $n^{e'}$, for the number e' that is guaranteed to exist by Lemma 6.25 applied to $d \stackrel{\text{def}}{=} r, d' \stackrel{\text{def}}{=} \lambda, e \stackrel{\text{def}}{=} s(r)$. In particular one can define distances with an FO+Maj$(\leq, +, \times)$ formula according to Lemma 6.25.

We remark on how the described dynamic program can be used to maintain the auxiliary relations for an arbitrary number of changes, starting from relations that are initialised in AC^2. Observe that for an arbitrary input matrix A of the stated form the computation of a normalised quotient n-approximation C of A^{-1} and its evaluation on multiple points modulo many primes can be done in AC^2. The dynamic program from above thus can maintain the auxiliary relations for the first k change steps.

A variant of Theorem 6.12 yields that this can also be done arbitrarily long. In the proof of Theorem 6.12 (or Theorem 3.3, respectively), the initialisation algorithm is only applied to the input structure. Inspecting the proof one can see that the initialisation also has access to the auxiliary relations that are maintained for its input. So, given the auxiliary relations that are maintained up to a time point, an $\mathsf{FO}+\mathsf{Maj}(\leq,+,\times)$ initialisation can interpolate the polynomials $g(x), h(x)$ for each entry $\frac{g(x)}{h(x)}$ of C, compute the first $n+1$ coefficients $d_0, \dots, d_n$ of $\frac{g(x)}{h(x)}$, and therefore obtain a new entry for the quotient n-approximation with numerator $\sum_{i=0}^{n} d_i x^i$, which has degree n, and denominator 1. Additionally, the initialisation can compute $C(a) \pmod{p}$ for all $a \in [n^\lambda]_0$ and all n^μ primes $p \in P$. As $\mathsf{FO}+\mathsf{Maj}(\leq,+,\times) =$ uniform $\mathsf{TC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{AC}[\frac{\log n}{\log\log n}]$ [Chandra et al. 1984, Theorem 4.3] and the necessary relations can be maintained for $k = \frac{\log n}{\log\log n}$ change steps, a generalisation of the proof of Theorem 6.12, that applies the initialisation also to the auxiliary database, shows the claim. □

6.4 Outlook and bibliographic remarks

In this chapter we introduced changes that insert and delete sets of tuples with bounded size, and studied which queries can still be maintained under these changes. We observed that UREACH as well as all regular languages of strings and trees can be maintained in $\mathsf{DynFO}(\leq,+,\times)$ under changes of polylogarithmic size, as well as REACH under insertions. These results are optimal with respect to the size of the change. Also, we extended the result from [Datta et al. 2018b] that REACH can be maintained under insertions and deletions of size $\frac{\log n}{\log\log n}$ to more general change operations, and showed that DISTANCE can be maintained in $\mathsf{DynFO}+\mathsf{Maj}(\leq,+,\times)$ under changes of polylogarithmic size. For insertions and deletions of $\frac{\log n}{\log\log n}$ edges, we saw that DISTANCE can be maintained in $\mathsf{DynFO}(\leq,+,\times)$ with an additional $\mathsf{FO}+\mathsf{Maj}(\leq,+,\times)$ query to extract the query result.

Several open questions remain. Of course our result for DISTANCE implies that REACH can be maintained under insertions *and* deletions of polylogarithmically many edges in $\mathsf{DynFO}+\mathsf{Maj}(\leq,+,\times)$. Is this also true for $\mathsf{DynFO}(\leq,+,\times)$? Possible steps towards such a result is to study REACH under $\log^d n$ insertions and single-edge deletions, or to consider restricted classes of input graphs. For example, can REACH be maintained under $\log^d n$ insertions and deletions on graphs of bounded treewidth or on planar graphs, for every $d \in \mathbb{N}$?

Regarding the DISTANCE query, membership in DynFO is open even for single-edge changes. See also [Mukherjee 2019] for multiple preliminary results in that direction.

It is open whether the result of Theorem 4.3 on maintenance of MSO-definable queries for graphs of bounded treewidth can be extended to changes of non-constant size. Probably such a result cannot be obtained by combining the techniques from Section 4.1 and Section 6.2 directly, because this would (1) require auxiliary data of size $n^{\log^d n}$, and (2) involve to translate arbitrary MSO sentences to $\sqrt[i]{n}$-MSO sentences.

While we showed by Theorem 6.4(b) and (c) that $\sqrt[i]{n}$-MSO is equally expressive as MSO on strings and tree for any $i \in \mathbb{N}$, we strongly believe that on general structures MSO is more expressive than $\sqrt[i]{n}$-MSO if $i > 1$, and even $\sqrt[i]{n}$-$r\mathsf{SO}$ for any $r \in \mathbb{N}$, as otherwise well-known hypotheses fail. We recall the NP-complete problem 3-SAT, which asks whether a propositional formula, given in conjunctive normal form with at most three literals per clause, is satisfiable. It is easy to show that 3-SAT can be expressed by an MSO formula. We show that if there is an $\sqrt[i]{n}$-$r\mathsf{SO}$ formula for 3-SAT, for any r and any $i > 1$, then the exponential time hypothesis is false. This hypothesis, formulated by Impagliazzo and Paturi [2001], states that 3-SAT cannot be solved in time $2^{o(n)}$.

Proposition 6.26. *If* 3-SAT $\in \sqrt[i]{n}$-$r\mathsf{SO}$ *for some* $i, r \in \mathbb{N}$ *with* $i > 1$*, then the exponential time hypothesis fails.*

Proof sketch. We argue similarly as for Proposition 6.5. Suppose there is an $\sqrt[i]{n}$-$r\mathsf{SO}$ formula φ that expresses 3-SAT, where $i > 1$ and r is arbitrary. For each n we can construct from φ a Boolean circuit C_n for inputs of 3-SAT encoded by strings of length n. The construction is analogous to the proof of Proposition 6.5, with the extension that $\sqrt[i]{n}$-$r\mathsf{SO}$ quantifiers are replaced by gates whose fan-in is upper-bounded by $n^{r\sqrt[i]{n}}$. The size of the resulting circuit C_n is bounded by $2^{\mathcal{O}(\log n \sqrt[i]{n})}$ which is in $2^{o(n)}$ for $i > 1$. It follows that 3-SAT can be solved in time $2^{o(n)}$, contradicting the exponential time hypothesis. □

It is also open whether formal languages beyond regular languages can be maintained under bulk changes. As MSO expresses exactly the regular languages, the techniques of Section 6.2 do not seem to be applicable.

When we use $\mathsf{FO}(\leq, +, \times)$ update formulas, we cannot hope for meaningful maintainability results for changes of size beyond polylogarithmic, see Proposition 6.3. This is not true any more if we allow update formulas from stronger logics. Of course, PARITY can trivially be maintained under any changes in $\mathsf{DynFO}{+}\mathsf{Maj}(\leq, +, \times)$, but for REACH and DISTANCE update mechanisms that cannot express all of NL are interesting. In [Datta et al. 2018b] it is shown that via NC^1-computable updates, both queries can be maintained under changes that affect $\mathcal{O}(2^{\sqrt{\log n / \log^* n}})$ nodes. Here, $\log^* n$ denotes the smallest number i such that i-fold application of log yields a number smaller than 1.

We already mentioned in Section 6.1 that FO update formulas, without built-in arithmetic, cannot be used in general to maintain queries under changes of non-constant size. This is basically the case because, unlike for single-tuple changes, first-order update formulas cannot extend a linear order on the active domain when a non-constant number of elements enter the active domain at once. If the change would contain more information than just the changed tuples, as for example a linear order and the BIT predicate on the domain of the change (that is, all elements that are affected by the change), then all results for $\mathsf{DynFO}(\leq, +, \times)$ of this chapter can be translated to DynFO. However, the functions $f(n)$ that give the maximal size of a change need to be functions in the size of the *active* domain, not of the whole domain. In this way, the results also transfer to the appropriately extended FOIES framework.

Bibliographic remarks

The framework for changes of non-constant size is introduced in [Datta et al. 2018b], which is joint work with Samir Datta, Anish Mukherjee and Thomas Zeume. The results from Section 6.3, apart from Theorem 6.14, are published there, together with the discussions on stronger update formalisms and on first-order updates without built-in arithmetic from the end of Section 6.4. The use of [Datta et al. 2018b, Theorem 4] in that paper is flawed, here we give a slightly restricted variant as Theorem 6.12 that nevertheless is sufficient to replace [Datta et al. 2018b, Theorem 4]. Theorem 6.14 was originally developed for [Schmidt et al. 2020] jointly with Jonas Schmidt, Thomas Schwentick, Thomas Zeume and Ioannis Kokkinis, but appears there only in a weaker version.

The results from Section 6.2 are obtained jointly with Thomas Zeume, they are not published so far.

CHAPTER 7

Experimental evaluation of the dynamic approach

In Chapter 1 we motivated the study of the dynamic approach, amongst others, by its relative efficiency with respect to static algorithms. We expressed the hope that applications would run faster if they would update a previous result instead of recomputing it from scratch. The choice of first-order logic as update language is justified by its relationship to SQL and the widespread use of relational database management systems for which dynamic programs can straightforwardly be implemented. The bold message here is: DynFO makes database systems run faster!

Of course, dynamic complexity is a theoretical framework and its main purpose is to *understand* the fundamentals of dynamic query maintenance, not to directly yield efficient algorithms for real-life applications. Still, as dynamic programs can be transferred comparatively easily into runnable code, we should test how well they actually work in practice.

Goal 7.1. Evaluate the performance of dynamic programs in practical scenarios.

Actual implementations and evaluations of the dynamic approach of query evaluation are rare. Dong et al. [1999] present SQL code for maintaining reachability for directed acyclic, undirected, and general directed graphs under single-edge changes[1] and analyse the number of joins needed to process a single-edge change. Empirical results for shortest distances in weighted undirected graphs under deletions, reported by Pang et al. [2005], indicate that the dynamic complexity approach can outperform recomputation from scratch significantly. The compilation of a class of non-recursive queries into incremental programs has been studied and implemented in [Koch 2010; Koch et al. 2014] as well as

[1]The code for reachability in general directed graphs cannot be translated to a DynFO-program, as it uses exponentially many constants not present in the input graph. As a result the update formulas may need to be evaluated for exponentially many tuples per change step.

[Nikolic et al. 2016]. Evaluations of different strategies to maintain joins of base relations are given by Vista [1998].

Contributions In this chapter we empirically compare the evaluation of the undirected reachability query UReach under first-order defined insertions of edges using

(1) dynamic programs for complex changes, following the approach of Theorem 5.4;
(2) dynamic programs that process complex changes by treating them as a sequence of single-edge changes;
(3) evaluation from scratch using SQL's recursive queries; and
(4) evaluation from scratch using standard imperative algorithms, implemented in Python.

We do *not* include graph database systems into our comparison. In a preliminary test using `Neo4j`[2] and its standard Cypher interface, query evaluation did not terminate in reasonable time even on very small instances. For general recursively defined queries a similar behaviour was observed in a recent study [Bagan et al. 2017]. We note that plug-ins[3] are available that enable `Neo4j` to evaluate UReach efficiently. The corresponding approach of query evaluation is, except of the chosen programming language, covered by method (4) in our comparison.

We perform three experiments to evaluate the four methods. For the first two experiments we use families of random graphs that consist of graphs with a varying number of disjoint subgraphs of varying edge density. One of these experiments involves a change that connects many formerly disjoint connected components, the second experiment in turn uses a change that inserts several edges between a small number of connected components. The change operations have small bridge bounds, following the guiding principle of Subsection 5.2.2 that such changes lead to small and thus efficient dynamic programs. For the third experiment, both changes are evaluated on a very large graph with nearly two million nodes, obtained from the DBLP database.

Our results give strong arguments for the use of the dynamic approach for query evaluation. For complex changes, our implementations of dynamic programs are considerably faster than recomputation from scratch within SQL in almost all of the considered scenarios, and in some scenarios even faster than the Python-based recomputation from scratch. Furthermore, the dynamic program for complex changes performs in general better than the dynamic program for single-edge changes which is invoked once for every changed edge. Especially the former program still performs well in the third experiment on the large DBLP-based graph.

All implemented variants can also deal with single-edge deletions, yet we only evaluate them for first-order definable insertions, since Pang et al. [2005] already evaluated the (single-tuple) deletion of edges.

Outline We give more details on our implementation in Section 7.1. In Section 7.2 we present our experiments and their results. We close with a short outlook in Section 7.3.

[2]see `https://neo4j.com/`

[3]e.g. `https://neo4j-contrib.github.io/neo4j-graph-algorithms/`

7.1 Implementation

In this section we describe our implementation in more detail. We implemented change operations and the evaluation methods (1)-(3) we presented in the introduction of this chapter as PL/pgSQL functions for a PostgreSQL database system. PL/pgSQL[4] is a procedural language that enables the definition of functions and allows the use of control structures in addition to SQL queries. From the latter, we only use if-then-else blocks.

In our implementation, a change function writes the set of all inserted edges to a table `delta`. The PL/pgSQL function `DYN-complex`, which implements the dynamic program that processes complex changes, accesses this table and updates the auxiliary relations accordingly. It is invoked before the changes are actually applied to the database. For processing a complex change as a sequence of edge insertions, the dynamic program for single-edge insertions from [Dong and Su 1998] is used. The PL/pgSQL function `DYN-single` implementing this program is invoked by a trigger when the changes of `delta` are actually applied to the database, once for every edge. The PL/pgSQL function `STATIC-SQL` that recomputes UREACH from scratch is invoked after the changes are applied and uses Common Table Expressions, more specifically the `WITH RECURSIVE` construct[5], an SQL construct that allows evaluating recursively defined queries like reachability, although it is not expressible in relational algebra. This function also has access to `delta`, which is used, e.g., to avoid recomputation when no edge between two distinct connected components was inserted. The algorithm `STATIC-Python` computes the connected components of an undirected input graph and writes the result to the database.

In order to store the query result for UREACH in a compact way, the functions use a binary table `connected_components` that contains for each node a unique representative of its connected component. In the implementation, nodes are represented by integers and the representative of a connected component is its minimal node with respect to the natural linear order. The actual transitive closure relation of the graph is easily definable in SQL or first-order logic using this table. The reason to use `connected_components` is its succinctness: it is of linear size with respect to the number of nodes, whereas the full transitive closure might be of quadratic size.

For [Dong and Su 1998, Theorem 4.3] and Theorem 5.4, some effort is made to maintain a linear order on the nodes, which is then used by the corresponding dynamic programs. As each database comes with a linear order—in our case, the natural order on the integers—it is not necessary to maintain such an order explicitly. Apart from this difference and the rather straightforward maintenance of `connected_components`, the implementations of the dynamic programs follow the proofs of the corresponding results closely.

Our implementations can actually maintain UREACH under first-order definable insertions on node coloured graphs, that is, graphs with additional unary relations $C_1, \dots, C_\ell$ representing ℓ colours. For our experiments we hence use coloured graphs.

[4] `https://www.postgresql.org/docs/current/static/plpgsql-overview.html`
[5] `https://www.postgresql.org/docs/current/static/queries-with.html`

7.2 Experiments and discussions

We conducted three experiments comparing the performance of the four approaches for evaluating UReach under complex insertions. The first two experiments explore how the approaches differ depending on the number of connected components that are joined by bridges. The third experiment examines how they scale to large graphs. The experiments use insertions with small bridge bounds, following the insights from Proposition 5.5 and Proposition 5.6.

All experiments were conducted on a machine with 28 CPU cores (56 threads) with 2.6 GHz base frequency, of which we use only one, and 52 GB main memory, running Ubuntu 16.04, with a local default installation of PostgreSQL 11devel. `STATIC-Python` uses Python 3.5.2 and version 1.11 of the `NetworkX` package[6]. If not stated otherwise, running times are obtained from six runs per graph instance. The first run is discarded, as well as the fastest and the slowest run. The reported time is the average of the three remaining runs (cf. [Bagan et al. 2017]). Individual timings include the time to update `connected_components` and, for the dynamic programs, a spanning forest and its transitive closure. Not included is the time needed to compute `delta`, to apply the changes to the database, and, for `STATIC-Python`, to build the input graph object.

First experiment In the first experiment, we tested the evaluation methods for a change that connects many different connected components of the input graph. The considered change operation, expressed by the formula $\rho_1(v) = \mu_E(v; x, y) = E(x, y) \vee ((x = v) \wedge C_1(y))$, inserts edges from some specified node v to all C_1 coloured nodes in a coloured graph. This insertion query is CQ-definable, and its bridge bound is 2. We tested this change on graphs of different sizes and for a varying number of edges. For each $n \in \{10000, 20000, 30000, 40000, 50000, 75000, 100000\}$ and $p \in \{0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$ we constructed five graphs with n nodes, whose colours were chosen uniformly at random from ten colours. Each graph was the disjoint union of an appropriate number of graphs with 50 nodes each. For each of these graphs, the probability of an (undirected) edge to be present was p. For each n we randomly chose a node v_n, applied the change $\rho_1(v_n)$ to each graph of size n, and measured the time it took each evaluation method to (re-)compute the table `connected_components`. Table 7.1 and Figure 7.1 give details on the graphs and the results of the experiment, where the stated values for every pair n, p average over the five instances. Because of the large running time, `STATIC-SQL` was, for $p \neq 0$, only tested for instances with at most 30000 nodes. Values marked * were obtained with a reduced testing schedule: due to large running times, only two runs on two instances were performed for every pair n, p, and the average over the four runs is reported.

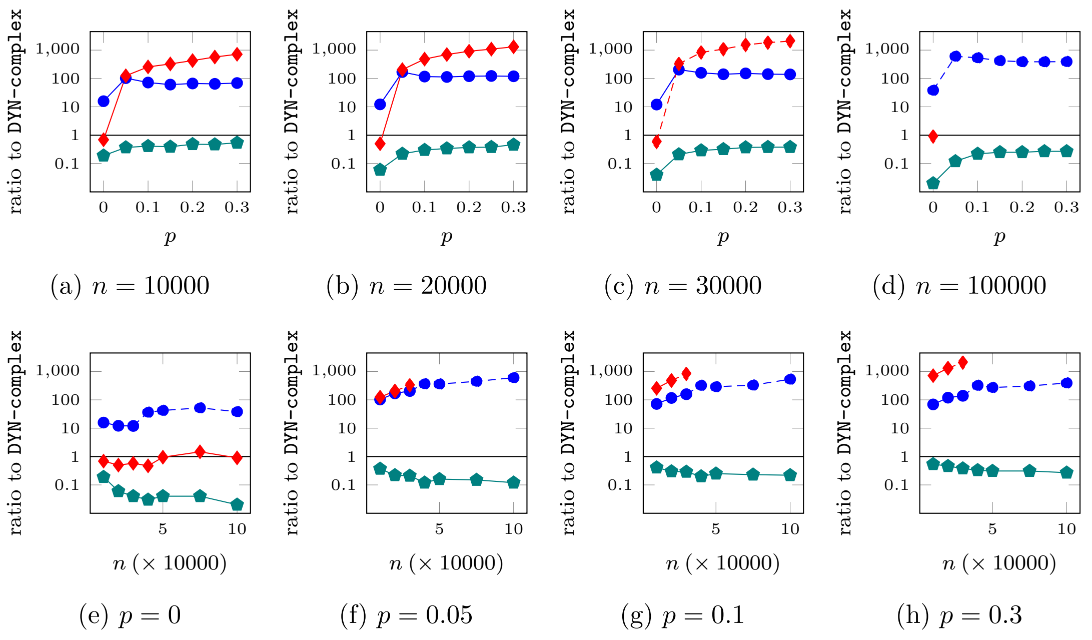
We observe that in this setting, where many connected components can be joined by one change, `DYN-complex` runs around two orders of magnitude faster than `DYN-single`. This is as expected, since the latter program has to update the auxiliary information for many nodes multiple times. Apart from the case $p = 0$ of initially empty graphs, direct

[6]see `https://networkx.github.io/`

Table 7.1: Results of the first experiment (insertion of a star into graphs with n nodes, consisting of random subgraphs of size 50 and edge probability p).

p	n	$\|E\|$	cc's	$\Delta\|E\|$	Δcc's	`DYN-c`	`DYN-s`	`STAT-SQL`	`STAT-Py`	$\frac{\texttt{DYN-s}}{\texttt{DYN-c}}$	$\frac{\texttt{STAT-SQL}}{\texttt{DYN-c}}$	$\frac{\texttt{STAT-Py}}{\texttt{DYN-c}}$
0	10000	0	10000	943	-943	1.74	27.43	1.2	0.33	15.79	0.69	0.19
0	20000	0	20000	2048	-2048	9.75	118.9	4.86	0.55	12.19	0.5	0.06
0	30000	0	30000	3118	-3118	19.51	234.48	11.3	0.78	12.02	0.58	0.04
0	40000	0	40000	3970	-3970	39.06	1426.98*	18.36	1.05	36.53*	0.47	0.03
0	50000	0	50000	5038	-5038	33.19	1404.24*	31.55	1.22	42.31*	0.95	0.04
0	75000	0	75000	7490	-7490	47.47	2473.57*	69.59	1.91	52.11*	1.47	0.04
0	100000	0	100000	10120	-10120	145.93	5588.25*	130.78	2.42	38.29*	0.9	0.02
0.05	10000	12259.6	1124	998.8	-308.2	0.91	93.42	113.06	0.34	102.78	124.39	0.37
0.05	20000	24577.6	2229.8	1991.8	-600.8	2.38	402.25	487.48	0.53	168.86	204.64	0.22
0.05	30000	36835.8	3377.2	2988	-922.6	3.79	777.28	1237.56*	0.79	205.15	326.64*	0.21
0.05	40000	48991.6	4530.2	3969.6	-1230.2	7.85	2887.69*	-	0.97	367.98*	-	0.12
0.05	50000	61259	5649.8	4943	-1523.4	7.93	2867.01*	-	1.25	361.65*	-	0.16
0.05	75000	91862	8457.6	7406.2	-2289.4	12.55	5643.84*	-	1.88	449.56*	-	0.15
0.05	100000	122644	11291	9975.2	-3063.6	20.16	12227.03*	-	2.35	606.38*	-	0.12
0.1	10000	24546.4	261.4	983.2	-203	0.75	54.03	192.53	0.31	71.8	255.85	0.41
0.1	20000	48911	519.4	1988.2	-407.4	1.8	209.54	865.77	0.53	116.63	481.87	0.3
0.1	30000	73412.4	789.8	2977	-613.2	2.59	410.03	2145.46*	0.76	158.07	827.08*	0.29
0.1	40000	97919.4	1034.6	4002.4	-817.8	4.43	1432.73*	-	0.91	323.42*	-	0.2
0.1	50000	122317	1297.6	5003.8	-1023.8	5.08	1473.1*	-	1.26	289.72*	-	0.25
0.1	75000	183908.8	1923.6	7505.8	-1538	7.84	2595.37*	-	1.84	330.89*	-	0.23
0.1	100000	245126.4	2575.8	9966.8	-2043	11.24	6004.65*	-	2.50	534.15*	-	0.22
0.15	10000	36659.4	204.2	985	-198.8	0.76	46.35	248.44	0.3	60.76	325.67	0.39
0.15	20000	73491.2	406	1982	-397	1.58	179.11	1106.47	0.54	113.02	698.19	0.34
0.15	30000	110142.8	609.8	2975.4	-598	2.52	355.1	2732.98*	0.81	140.84	1083.99*	0.32
0.15	40000	147038	815.4	3996.2	-797	4.01	1253.63*	-	1.05	312.48*	-	0.26
0.15	50000	183608.2	1017.6	5027.2	-996.6	4.74	1252.73*	-	1.26	264.24*	-	0.27
0.15	75000	275649.8	1527.8	7377.8	-1491.2	7.02	2148.04*	-	1.84	306.06*	-	0.26
0.15	100000	367722.8	2037.6	10038.2	-1992.6	10.36	4390.39*	-	2.56	423.73*	-	0.25
0.2	10000	48912	200.2	1001.6	-198	0.64	42.45	301.25	0.3	66.55	472.25	0.48
0.2	20000	97934.6	400.6	2004.8	-396.4	1.48	177.88	1350.96	0.55	120.14	912.48	0.37
0.2	30000	146978.6	600.6	2991.8	-597.6	2.17	327.65	3321.56*	0.8	151.03	1531.12*	0.37
0.2	40000	195696.4	801	3984.6	-795	3.86	1189.67*	-	1.04	307.87*	-	0.27
0.2	50000	245184.2	1000.8	4975.8	-992.6	4.29	1190.05*	-	1.29	277.59*	-	0.3
0.2	75000	367597.4	1501.2	7560.6	-1493.8	6.63	2064.55*	-	1.85	311.61*	-	0.28
0.2	100000	490189.4	2002	9961.6	-1988.8	10.07	3912.13*	-	2.56	388.51*	-	0.25
0.25	10000	61323.2	200	975.4	-198.6	0.65	41.86	369.78	0.3	64.59	570.55	0.47
0.25	20000	122483.6	400	2022	-397	1.43	175.22	1549.37	0.55	122.58	1083.88	0.38
0.25	30000	183640.2	600	2993.2	-596.8	2.17	308.71	3992.07*	0.83	142.37	1841.06*	0.38
0.25	40000	244909.2	800	3977	-794	3.44	1089.95*	-	1.04	316.77*	-	0.3
0.25	50000	306185	1000	4979	-993.4	4.07	1141.95*	-	1.31	280.37*	-	0.32
0.25	75000	459241.8	1500.4	7537.2	-1493	6.77	1954.61	-	1.88	288.89*	-	0.28
0.25	100000	612608	2000	9956.4	-1987.4	9.86	3817.87*	-	2.61	387.37*	-	0.27
0.3	10000	73516.6	200	987.6	-199	0.58	40.34	417.09	0.32	69	713.43	0.54
0.3	20000	146868.2	400	1998.2	-395.8	1.36	162.49	1780.68	0.62	119.43	1308.86	0.46
0.3	30000	220461.2	600	2999	-596.6	2.13	295.67	4408.44*	0.8	139.1	2074.04*	0.38
0.3	40000	294246.4	800	3992	-793.2	3.33	1078.65*	-	1.11	323.76*	-	0.33
0.3	50000	367958.6	1000	4939.6	-994.4	4.16	1134.23*	-	1.29	272.53*	-	0.31
0.3	75000	551294.2	1500	7481.2	-1489.6	6.19	1908.72*	-	1.92	308.38*	-	0.31
0.3	100000	735921.8	2000	10018.6	-1987	9.64	3789.45*	-	2.6	393.14*	-	0.27

The columns $|E|$ and cc's provide the number of edges and connected components before the change. $\Delta|E|$ provides the number of inserted edges by the change, and Δcc's gives the difference of the number of connected components. The columns `DYN-c`, `DYN-s`, `STAT-SQL`, `STAT-Py` give the runtime in seconds of the algorithms `DYN-complex`, `DYN-single`, `STATIC-SQL`, `STATIC-Python`, respectively. The values are avaraged over five instances for each pair (p, n).



(a) $n = 10000$ (b) $n = 20000$ (c) $n = 30000$ (d) $n = 100000$

(e) $p = 0$ (f) $p = 0.05$ (g) $p = 0.1$ (h) $p = 0.3$

Figure 7.1: Ratios of the running times in Experiment 1 of the different evaluation methods compared to `DYN-complex`, for different values of n and p, as listed in Table 7.1. Dashed lines indicate that the values are obtained with a reduced testing schedule. ●: $\frac{\texttt{DYN-single}}{\texttt{DYN-complex}}$ ♦: $\frac{\texttt{STATIC-SQL}}{\texttt{DYN-complex}}$ ⬟: $\frac{\texttt{STATIC-Python}}{\texttt{DYN-complex}}$

processing of complex changes is also faster than computation from scratch using SQL. In our tests, `DYN-complex` ran three to four orders of magnitude faster than `STATIC-SQL`, which did not even terminate in reasonable time on larger graphs. For $p = 0$, `STATIC-SQL` is faster than `DYN-complex`. This is as expected: because no edge was present before the change, no prior auxiliary information is available to the dynamic programs. Basically, they also have to recompute the query result from scratch.

In all cases, the dynamic approaches are outperformed by the Python-based evaluation from scratch. For $p \neq 0$, the speed-up is between 2 and 8.

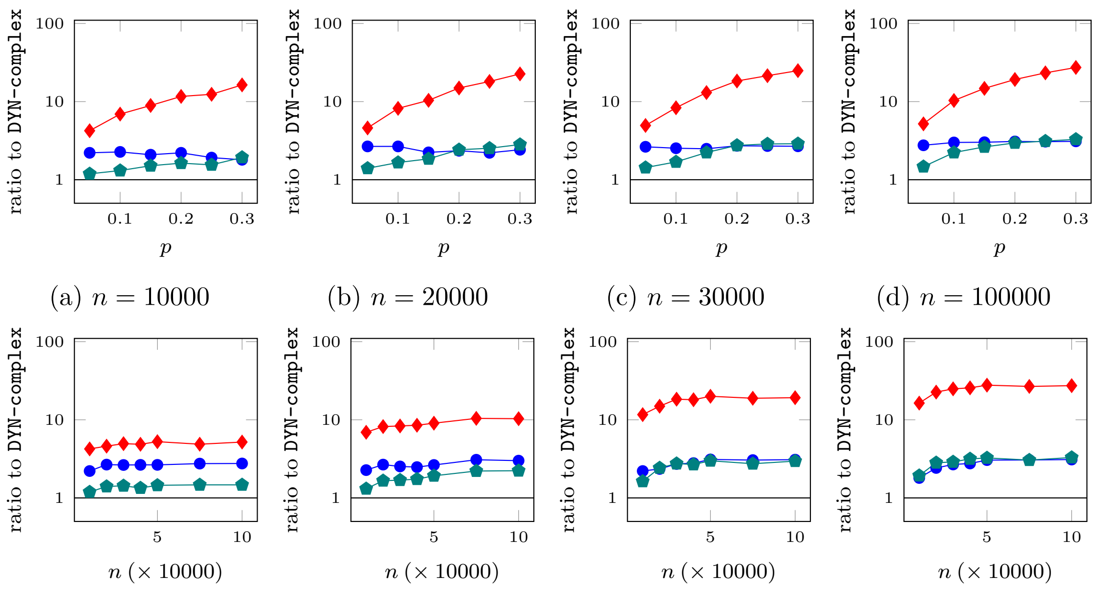
Second experiment In the second experiment, we compared the evaluation methods for the same graphs but for a change operation that connects only few connected components. More precisely, the change operation ρ_2 with parameters $v_1, \ldots, v_7$ connects all neighbours of v_1 and v_2 with the nodes $v_3, \ldots, v_7$ through the rule $\rho_2(\bar{v}) = \mu_E(v_1, \ldots, v_7; x, y) = E(x, y) \vee \big((E(x, v_1) \vee E(x, v_2)) \wedge (y = v_3 \vee y = v_4 \vee y = v_5 \vee y = v_6 \vee y = v_7)\big)$. This change can be expressed by a union of ten conjunctive queries. Its bridge bound is easily seen to be 2, which is much better than the bound of 20 given by Proposition 5.5.

The set-up of this experiment is analogous to the first experiment. The case $p = 0$ was omitted, since ρ_2 does not change empty graphs at all. The results are provided in Table 7.2 and Figure 7.2.

Table 7.2: Results of the second experiment (insertion of edges between up to seven connected components of graphs with n nodes, consisting of random subgraphs of size 50 and edge probability p).

p	n	$\|E\|$	cc's	$\Delta\|E\|$	Δcc's	`DYN-c`	`DYN-s`	`STAT-SQL`	`STAT-Py`	$\frac{\texttt{DYN-s}}{\texttt{DYN-c}}$	$\frac{\texttt{STAT-SQL}}{\texttt{DYN-c}}$	$\frac{\texttt{STAT-Py}}{\texttt{DYN-c}}$
0.05	10000	12259.6	1124	28	-5.8	0.25	0.56	1.07	0.3	2.21	4.24	1.19
0.05	20000	24577.6	2229.8	32	-6	0.4	1.06	1.83	0.56	2.67	4.6	1.4
0.05	30000	36835.8	3377.2	21	-5.6	0.54	1.42	2.66	0.77	2.64	4.95	1.43
0.05	40000	48991.6	4530.2	28	-6	0.73	1.94	3.55	0.98	2.65	4.86	1.34
0.05	50000	61259	5649.8	27	-5.6	0.84	2.21	4.39	1.22	2.65	5.25	1.45
0.05	75000	91862	8457.6	31	-6	1.26	3.45	6.12	1.84	2.75	4.87	1.47
0.05	100000	122644	11291	20	-5.8	1.59	4.39	8.25	2.34	2.76	5.19	1.47
0.1	10000	24546.4	261.4	34	-6	0.22	0.5	1.53	0.29	2.27	6.92	1.31
0.1	20000	48911	519.4	49	-6	0.33	0.88	2.71	0.55	2.67	8.19	1.66
0.1	30000	73412.4	789.8	43	-6	0.46	1.16	3.84	0.78	2.53	8.34	1.69
0.1	40000	97919.4	1034.6	56	-6	0.57	1.41	4.86	0.99	2.48	8.52	1.74
0.1	50000	122317	1297.6	47	-6	0.64	1.69	5.81	1.23	2.64	9.05	1.92
0.1	75000	183908.8	1923.6	47	-6	0.83	2.56	8.68	1.84	3.08	10.43	2.21
0.1	100000	245126.4	2575.8	48	-6	1.13	3.39	11.66	2.52	3	10.33	2.23
0.15	10000	36659.4	204.2	72	-6	0.21	0.44	1.9	0.32	2.09	8.92	1.51
0.15	20000	73491.2	406	71	-6	0.30	0.68	3.14	0.56	2.24	10.38	1.85
0.15	30000	110142.8	609.8	70	-6	0.35	0.87	4.57	0.78	2.49	13.05	2.23
0.15	40000	147038	815.4	73	-6	0.42	1.14	5.83	1.01	2.72	13.88	2.4
0.15	50000	183608.2	1017.6	59	-6	0.48	1.45	7.04	1.24	3.01	14.61	2.58
0.15	75000	275649.8	1527.8	62	-6	0.73	2.25	10.74	1.86	3.07	14.67	2.55
0.15	100000	367722.8	2037.6	65	-6	0.98	2.95	14.4	2.57	3.02	14.76	2.63
0.2	10000	48912	200.2	99	-6	0.19	0.42	2.19	0.31	2.21	11.65	1.63
0.2	20000	97934.6	400.6	94	-6	0.23	0.54	3.44	0.56	2.35	14.93	2.42
0.2	30000	146978.6	600.6	108	-6	0.29	0.79	5.34	0.8	2.72	18.36	2.75
0.2	40000	195696.4	801	98	-6	0.38	1.05	6.79	1.01	2.79	18.07	2.68
0.2	50000	245184.2	1000.8	89	-6	0.43	1.33	8.56	1.28	3.11	20	2.99
0.2	75000	367597.4	1501.2	91	-6	0.68	2.08	12.84	1.87	3.05	18.87	2.74
0.2	100000	490189.4	2002	107	-6	0.9	2.77	17.19	2.65	3.09	19.19	2.96
0.25	10000	61323.2	200	124	-6	0.20	0.39	2.51	0.31	1.92	12.4	1.55
0.25	20000	122483.6	400	107	-6	0.22	0.5	4.06	0.57	2.21	18.11	2.53
0.25	30000	183640.2	600	127	-6	0.28	0.76	6.06	0.8	2.7	21.46	2.87
0.25	40000	244909.2	800	119	-6	0.35	1	8.23	1.03	2.82	23.22	2.9
0.25	50000	306185	1000	116	-6	0.41	1.26	10.08	1.29	3.08	24.6	3.15
0.25	75000	459241.8	1500.4	134	-6	0.64	1.98	14.94	1.92	3.08	23.18	2.98
0.25	100000	612608	2000	104	-6	0.86	2.65	20.17	2.69	3.07	23.33	3.12
0.3	10000	73516.6	200	137	-6	0.16	0.28	2.56	0.3	1.8	16.36	1.93
0.3	20000	146868.2	400	138	-6	0.2	0.49	4.6	0.57	2.42	22.62	2.82
0.3	30000	220461.2	600	135	-6	0.28	0.75	6.96	0.81	2.68	24.9	2.89
0.3	40000	294246.4	800	159	-6	0.35	0.97	9.02	1.11	2.76	25.64	3.16
0.3	50000	367958.6	1000	160	-6	0.4	1.22	11.11	1.3	3.05	27.76	3.25
0.3	75000	551294.2	1500	115	-6	0.63	1.93	16.84	1.92	3.06	26.74	3.05
0.3	100000	735921.8	2000	142	-6	0.83	2.56	22.59	2.72	3.1	27.35	3.29

See Table 7.1 for an explanation of the columns.



(a) $n = 10000$ (b) $n = 20000$ (c) $n = 30000$ (d) $n = 100000$

(e) $p = 0.05$ (f) $p = 0.1$ (g) $p = 0.2$ (h) $p = 0.3$

Figure 7.2: Ratios of the running times in Experiment 2 of the different evaluation methods compared to `DYN-complex`, for different values of n and p, as listed in Table 7.2. ●: $\frac{\texttt{DYN-single}}{\texttt{DYN-complex}}$ ♦: $\frac{\texttt{STATIC-SQL}}{\texttt{DYN-complex}}$ ⬟: $\frac{\texttt{STATIC-Python}}{\texttt{DYN-complex}}$

In this experiment, `DYN-complex` was again considerably faster than `STATIC-SQL` and even slightly faster than `STATIC-Python`. Also it outperformed `DYN-single`. This program in turn performed very well on large graphs even in comparison to `STATIC-Python`. This seems to be because it can tell rapidly whether an inserted edge lies inside a connected component, and thus can be ignored. `STATIC-SQL` performed better than in the first experiment, since the connected components after the change are smaller here.

Third experiment For the last experiment, we tested the performance of the dynamic programs on a large real-world graph. The graph G_{dblp} is obtained from a snapshot of the DBLP dataset from June 2017[7]. The nodes of G_{dblp} correspond to authors, and edges are based on co-authorship: an (undirected) edge (u, v) implies that the authors corresponding to u and v are co-authors of some publication.

We tested the performance for both insertions ρ_1 and ρ_2. As ρ_1 is an insertion query for coloured graphs, we coloured one out of thousand nodes with colour C_1. Details on the DBLP graph and the results of the experiment are provided in Table 7.3.

While `DYN-complex` updated the query result in a reasonable amount of time in both tests, `DYN-single` was only able to do so for the "easier" change ρ_2. It needed more than 100 minutes to process the change ρ_1. However, for ρ_2, `DYN-single` was actually the fastest method. Again, `DYN-complex` was slightly faster than `STATIC-Python`.

[7]see `http://dblp.dagstuhl.de/xml/release/`

Table 7.3: Results of the third experiment (application of the changes of the first two experiments to a large real-world graph).

Graph	$\lvert V\rvert$	$\lvert E\rvert$	cc's	change	$\Delta\lvert E\rvert$	Δcc's	DYN-c	DYN-s	STAT-SQL	STAT-Py	$\frac{\texttt{DYN-s}}{\texttt{DYN-c}}$	$\frac{\texttt{STAT-SQL}}{\texttt{DYN-c}}$	$\frac{\texttt{STAT-Py}}{\texttt{DYN-c}}$
G_{dblp}	1949121	8823792	151874	ρ_1	1950	-248	51.29	6270.86	–	59.65	122.26	–	1.16
G_{dblp}	1949121	8823792	151874	ρ_2	140	-3	42.1	38.11	–	52.31	0.91	–	1.24

See Table 7.1 for an explanation of the columns. Timings for `STATIC-SQL` are not listed, as the experiment did not finish within twelve hours.

`STATIC-SQL` reached its limits in this experiment: it did not finish within twelve hours.

Discussion We conclude from the results of the experiments that using dynamic programs and in particular `DYN-complex` to maintain undirected reachability can reduce the running time significantly, especially compared with `STATIC-SQL`. Not surprisingly, a special purpose program evaluating from scratch can achieve better results. That said, we actually expected the gap to `DYN-complex` to be larger, and we are surprised that for some test cases, and in particular for the large real-world graph G_{dblp}, `DYN-complex` even performs slightly better. In any case, if no such program is at hand, the dynamic solution can be by far the best solution.

The results indicate that, as one would expect, the running time of `DYN-single` and `DYN-complex` is dominated by the number of connected components that are connected by the change; the running time of `STATIC-SQL` and `STATIC-Python` on the other hand is dominated by the size of the graph, and for `STATIC-SQL` in particular by the size and the density of the connected components. So, the relative performances of the different evaluation methods very much depend on the test case at hand.

7.3 Outlook and bibliographic remarks

The encouraging results concerning the performance of the dynamic approach that we obtained in this chapter can be seen as a practical proof of concept of the results obtained in Section 5.2 and especially Subsection 5.2.2. They were not at all predictable, since the benefit of reusing previously computed information might have been smaller than the cost of updating all auxiliary relations.

It is however too early to attest superiority of the dynamic approach over static query evaluation, even restricted to UReach. We have only evaluated the different approaches in a very limited number of scenarios. For better-founded conclusions we would need to consider more divers graphs and changes, preferably extracted from real-world data.

Now that we have promising results for UReach, the canonical next step is to evaluate the dynamic program for directed reachability [Datta et al. 2018a]. Preliminary tests suggest that one cannot expect competitive performances here. We mention some challenges. The dynamic program from [Datta et al. 2018a] maintains matrices of quadratic size in the number of nodes, so it requires space $\Omega(n^2)$, which might be too much for input graphs of "interesting" size. Of course this argument already applies to the query relation, but that relation could be stored more succinctly, for example as a tree of strongly connected components, analogously to the table `connected_components` we used above. Additionally, such matrices are maintained for many primes, which also increases the work significantly. If the computations are not done modulo many primes, the involved numbers increase in each step, so the algorithm might run into numerical problems.

In general, dynamic programs that rely on the Muddling technique do not seem to be good choices for being implemented, as this technique involves to simulate many instances of a dynamic program in parallel. While this gives low parallel time, the overall work to be done is high. In practical parallel settings with a constant number of processors, the efficiency of update routines largely depends on the work, which roughly corresponds to the number of gates in a circuit that expresses the update. The question for update rules with low parallel complexity and simultaneously low overall work needs to be further investigated. The class DynQF, defined via quantifier-free update formulas that may use functions [Hesse 2003b], provides a good starting point for this investigation, in particular when only unary auxiliary relations are used. See [Hesse 2003b; Gelade et al. 2012; Zeume 2015] for some upper and lower bounds for this class.

Implementing dynamic programs is tedious work. A high-level programming language together with a compiler would greatly facilitate this process.

Bibliographic remarks

The contents of this chapter are joint work with Thomas Schwentick and Thomas Zeume, and are published as [Schwentick et al. 2018]. The framework for the empirical evaluation and parts of the implementation of the dynamic programs were programmed by Dennis Ciba. We thank the Database and Information Systems group at TU Dortmund led by Jens Teubner for providing access to a compute server for the experiments.

CHAPTER 8

Conclusion

Whether the reachability query REACH for arbitrary directed graphs can be maintained in DynFO under single-edge insertions and deletions was by far the most important open question in the dynamic complexity framework, until Datta, Kulkarni, Mukherjee, Schwentick, and Zeume [2015; 2018a] gave a positive answer.

Solving this question brought new research goals within reach, in particular the question whether REACH, as well as other queries that are maintainable in DynFO under single-tuple changes, can still be maintained under more general changes. This thesis is a first comprehensive investigation of questions of this kind. We have shown multiple strong maintainability results, both for first-order definable changes as well as size-restricted bulk changes. We summarise them briefly; afterwards we briefly discuss results concerning static analysis problems in dynamic complexity and give an outlook on further research.

Summary of the results Our strongest results concern the undirected reachability query UREACH, which we showed to be maintainable in DynFO and DynFO$(\leq, +, \times)$ under all first-order definable insertions as well as under insertions and deletions of polylogarithmically many edges (with respect to the number of nodes), respectively (Theorem 5.4 and Theorem 6.10). The latter result is optimal in the sense that UREACH cannot be maintained in DynFO$(\leq, +, \times)$ under larger changes, see the discussion of Proposition 6.3.

We presented partial results for the directed reachability query REACH: it can be maintained under quantifier-free first-order definable insertions for acyclic graphs (Theorem 5.8) and under insertions of polylogarithmic size (Theorem 6.9). The result that REACH can be maintained under edge insertions and deletions that affect $\frac{\log n}{\log \log n}$ many nodes [Datta et al. 2018b], see also Theorem 6.13 and Theorem 6.14, is a very instructive example on how the techniques of [Datta et al. 2018a] can be generalised to complex changes, but is not part of this thesis. We use its techniques here to show that the DISTANCE query can be maintained in $\mathsf{DynFO{+}Maj}(\leq, +, \times)$ under polylogarithmically many insertions and deletions of edges (Theorem 6.15), and discuss in Subsection 6.3.3 a result on DISTANCE that requires $\mathsf{FO{+}Maj}(\leq, +, \times)$ formulas only to extract the query result—with the downside that only changes of size $\frac{\log n}{\log\log n}$ are supported.

Despite the high expressive power of first-order update formulas we were able to provide non-maintainability results, at least in restricted dynamic settings. We have shown that REACH is not in DynProp under quantifier-free first-order insertions or deletions (Theorem 5.11), and that REACH cannot be maintained in DynFO under very weak first-order definable insertions or deletions if only unary additional auxiliary relations are allowed (Theorem 5.9).

In addition to these results on the graph reachability queries, we extended the results of Gelade et al. [2012] on maintenance of formal languages. We proved that all context-free languages are in DynFO under changes that are defined by first-order formulas without quantifier alternation, see Section 5.4. All regular languages of strings and ranked trees can be maintained in $\mathsf{DynFO}(\leq, +, \times)$ under changes of polylogarithmically many positions (Theorem 6.11), which is again an optimal result for $\mathsf{DynFO}(\leq, +, \times)$ with respect to the size of the change, see once more Proposition 6.3. We also showed that all AC^1 queries are in DynFO under parameter-free first-order definable changes (Corollary 5.20).

Apart from the reachability query and formal language maintenance, which are both well-studied in the dynamic complexity framework, we started to investigate the class of MSO-definable queries and proved a dynamic version of Courcelle's Theorem for single-tuple changes: all MSO-definable queries are in DynFO for graphs with bounded treewidth under changes of single tuples, see Theorem 4.3 and Theorem 4.4. The Muddling technique as presented in Chapter 3 was an essential tool in the proofs of these results, and was used beyond that. We clarified the structure of DynProp and proved an arity hierarchy for Boolean graph queries under single-edge changes (Theorem 4.13).

We were not only able to prove expressibility results that are surprisingly strong, at least in the author's opinion, but our experimental evaluation presented in Chapter 7 showed that a direct implementation of dynamic programs may run considerably faster than naive re-computation from scratch and can achieve performance results that are comparable with those of special-purpose static algorithms. It also confirmed the intuition that a direct treatment of complex changes is advantageous compared to handling them as a sequence of single-tuple changes, which further justifies the research agenda pursued in this thesis.

Static analysis of dynamic programs This work focussed on the expressive power of first-order update formulas under complex changes and single-tuple changes. We now briefly discuss questions regarding the static analysis of dynamic programs: given a dynamic program, what is the complexity of deciding whether this program has some specific property? This line of work originated in the author's master's thesis [Vortmeier 2013], and was considerably extended jointly with Thomas Schwentick and Thomas

Zeume, and published as [Schwentick et al. 2015].

One motivation for studying static analysis of dynamic programs as stated in [Schwentick et al. 2015] is the wish to understand what a dynamic program at hand "does", and, more generally, to understand what various classes of restricted dynamic programs "can do" in general. Insights regarding these questions might ultimately lead to methods for proving non-maintainability result in restricted dynamic settings.

In [Schwentick et al. 2015], the vague question "what does a given dynamic program do?" is translated into three static analysis problems for classes of dynamic programs. The first problem concerns the *emptiness* of dynamic programs: given a dynamic program, is there a sequence of changes such that the query relation maintained by the dynamic program is non-empty? Related static analysis problems are standard in database theory and other fields. More specific for the dynamic setting is the *consistency* problem: given a dynamic program, do all sequences of changes that lead to the same input structure also lead to the same maintained query relation? This question basically asks whether the given dynamic program actually maintains some query. For a consistent dynamic program, the query relation is functionally dependent from the input relations. Some dynamic programs satisfy even the stronger condition that all auxiliary relations are functionally dependent from the input structure. We call such dynamic programs *history independent*, other authors use the terms *memoryless* [Patnaik and Immerman 1997] and *deterministic* [Dong and Su 1997]. History independent dynamic programs are studied for example in [Dong and Su 1997; Grädel and Siebertz 2012]. The *history independence* problems asks whether the given dynamic program is history independent. For all three problems, [Schwentick et al. 2015] only considers change sequences that consist of insertions and deletions of single tuples.

Not surprisingly, these three problems are undecidable for general dynamic programs, due to the undecidability of the satisfiability problem for first-order logic. Therefore [Schwentick et al. 2015] studies these problems for restricted classes of dynamic programs and investigates the precise border of undecidability. The considered classes are defined according to the following parameters:

- whether update formulas may use quantifiers,
- the maximal arity of input relations,
- the maximal arity of auxiliary relations, and
- only for the emptiness problem: whether the input program is guaranteed to be consistent.

It is observed that the emptiness problem is equivalent to the consistency problem for all considered classes of not necessarily consistent dynamic programs. Both problems are decidable only for a severely restricted class of dynamic programs. Slightly stronger decidability results were obtained for the emptiness problem for consistent dynamic programs and for the history independence problem. The precise results are summarised in Table 8.1.

Outlook: open problems and further research Several questions regarding the expressive power of first-order update formulas remain open. Some of them have already

	Emptiness, Consistency	Emptiness for consistent programs	History Independence

Undecidable	DynFO(1-in, 0-aux)
	DynProp(2-in, 0-aux)
	DynProp(1-in, 2-aux)
		DynFO(1-in, 2-aux)
		DynFO(2-in, 0-aux)
			DynFO(2-in, 0-aux)

Decidable	DynProp(1-in, 1-aux)	DynFO(1-in, 1-aux)
		DynProp(1-in)
		DynProp(1-aux)
			DynFO(1-in)
			DynProp(1-aux)

Open	—	DynProp(2-in, 2-aux) and beyond	DynProp(2-in, 2-aux) and beyond

Table 8.1: Summary of the results of [Schwentick et al. 2015], see [Schwentick et al. 2015, Table 1]. DynFO(ℓ-in, m-aux) stands for dynamic first-order programs with (at most) ℓ-ary input relations and m-ary auxiliary relations. DynFO(m-aux) and DynFO(ℓ-in) represent programs with m-ary auxiliary relations (and input relations with arbitrary arity) and programs with ℓ-ary input relations (and auxiliary relations with arbitrary arity), respectively. DynProp(ℓ-in, m-aux), DynProp(ℓ-in) and DynProp(m-aux) are defined analogously for classes of dynamic programs with quantifier-free first-order update formulas.

been stated at the end of the preceding chapters, and we repeat the most important ones. We also present topics for further research.

The precise dynamic complexity of the reachability query While we know many DynFO maintainability results for Reach and UReach, there are comparatively few results with respect to complex deletions. It is for example open whether UReach is in DynFO under (some non-trivial class of) first-order definable deletions, even if besides these deletions only single-edge insertions are allowed.

We showed in this thesis that Reach is in DynFO($\leq, +, \times$) under polylogarithmically many edge insertions; the corresponding dynamic program cannot handle any edge deletions. It is unclear whether this result can be extended to allow additionally deletions of single edges. The so far best result for DynFO($\leq, +, \times$) that allows for insertions and deletions covers changes of size $\frac{\log n}{\log \log n}$ [Datta et al. 2018b], and we ask whether this result can be extended towards insertions and deletions of logarithmic or even polylogarithmic size. A first step towards answering this question might be to only consider restricted graph classes, like the class of planar graphs or graphs of bounded treewidth.

We still do not know whether Reach is in DynProp under single-edge changes, although some lower bounds for special cases are known [Zeume and Schwentick 2015]. We ask for a proof of the general DynProp non-maintainability result.

Stronger reductions for dynamic complexity Bounded first-order reductions, as introduced by Patnaik and Immerman [1997], have the property that whenever such a reduction Υ maps an instance $\mathcal{D}$ to an instance $\Upsilon(\mathcal{D})$, then for any single-tuple change α to $\mathcal{D}$ the structures $\Upsilon(\mathcal{D})$ and $\Upsilon(\alpha(\mathcal{D}))$ only differ by some constant number of tuples. See Example 2.4 for an application.

The bulk changes we studied in Chapter 6 give rise to the stronger form of *polylogarithmically bounded* first-order reductions, which allow any single-tuple change α of $\mathcal{D}$ to lead to $\log^d n$ many changes to $\Upsilon(\mathcal{D})$, where n is the size of the domain of $\mathcal{D}$ and $d \in \mathbb{N}$ is some global constant. All classes that include $\mathsf{DynFO}(\leq, +, \times)$ are closed under these reductions. As we have shown that UReach is in $\mathsf{DynFO}(\leq, +, \times)$ under changes of size $\log^d n$ for every fixed $d \in \mathbb{N}$, we can infer $(q, \Delta) \in \mathsf{DynFO}(\leq, +, \times)$ for every dynamic query (q, Δ) such that (1) q can be reduced to UReach by a polylogarithmically bounded first-order reduction, and (2) Δ is a set of bulk change operations of at most polylogarithmic size. This approach can be pursued also for any other query that can be maintained under changes of polylogarithmic size.

We hope for new maintainability results by means of polylogarithmically bounded first-order reductions, and in particular for results for whole classes of queries. Remember that we cannot deduce from the fact $(\textsc{Reach}, \Delta_E) \in \mathsf{DynFO}$ [Datta et al. 2018a] that all NL queries are in DynFO under single-tuple changes, because Reach is not NL-hard under bounded first-order reductions as defined by Patnaik and Immerman [Patnaik and Immerman 1997]. Can this result be extended to polylogarithmically bounded first-order reductions? Are there natural classes of queries such that Reach or UReach is hard for these classes under polylogarithmically bounded first-order reductions?

Dynamic complexity, dynamic algorithms and practice We have seen in Chapter 7 that direct implementations of dynamic programs can exhibit competitive running times. This insight may lead to further research in two directions: more evaluations of dynamic programs and more emphasis on "efficient" dynamic programs. We briefly discuss these directions.

The conducted experiments we presented in Section 7.2 naturally cover only a limited number of use cases. Further experiments using the different implemented routines answering UReach might be helpful to get a more comprehensive overview of their efficiency. Most notably, such experiments should involve definable changes with a bridge bound larger than 2. Because of the encouraging results we obtained for UReach we advocate the implementation and evaluation of dynamic problems for other queries, as for example Reach. Further research might facilitate the implementation process. As dynamic programs are usually only sketched in proofs, a high-level specification language would be helpful to obtain a first complete formal description of a dynamic program. In a second step, a compiler could translate such a description into a prototypical SQL program. In might however still be necessary that domain experts subsequently optimise such a program to actually obtain an efficient implementation.

More experience regarding efficiency of implemented dynamic programs might also help to identify classes of dynamic programs in the dynamic complexity framework that "inherently" lead to efficient implementations, for example because the overall work of the dynamic program is low. Here, the *work* of a dynamic program is the (sequential) time that is necessary to evaluate it.

One key factor that affects the work is the number of tuples for which the update formulas are evaluated. This number is linear for unary auxiliary relations, which makes constructing dynamic programs with unary auxiliary structures a prime goal. Our results for UReach suggest however that also binary auxiliary relations can be updated in reasonable time if the relations are sparse. An orthogonal approach might be to study dynamic programs that only update the auxiliary relations for tuples that are within a certain distance to a changed tuple.

The other key factor that affects the overall work is the work that is necessary to evaluate an update formula once. Quantifier-free update formulas can be evaluated in constant time even when they are allowed to use functions, at least if we assume that functions and atomic formulas can be evaluated in constant time. Apart from update formulas of this form one could also consider update formalisms that do not lead to constant parallel time updates but to low work, as for example updates specified by NC^1 circuits of linear size.

These options lead to multiple different models of parallel update routines whose expressive power might be worth to be investigated. Maybe they could also help to bridge the gap between the research fields of dynamic complexity and dynamic algorithms, which so far do not seem to profit much from each other.

Bibliography

[Abboud and Williams 2014] Amir Abboud and Virginia Vassilevska Williams. 2014. Popular Conjectures Imply Strong Lower Bounds for Dynamic Problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*. IEEE Computer Society, 434–443. `https://doi.org/10.1109/FOCS.2014.53` (Cited on page 2.)

[Abiteboul et al. 1995] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley, Boston, MA, USA. `http://webdam.inria.fr/Alice/` (Cited on pages 3, 9, 11, and 87.)

[Agrawal and Vinay 2008] Manindra Agrawal and V. Vinay. 2008. Arithmetic Circuits: A Chasm at Depth Four. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*. IEEE Computer Society, 67–75. `https://doi.org/10.1109/FOCS.2008.32` (Cited on page 132.)

[Ajtai 1983] Miklós Ajtai. 1983. Σ^1_1-Formulae on finite structures. *Annals of Pure and Applied Logic* 24, 1 (1983), 1–48. `https://doi.org/10.1016/0168-0072(83)90038-6` (Cited on pages 19 and 115.)

[Amarilli et al. 2019] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. 2019. Enumeration on Trees with Tractable Combined Complexity and Efficient Updates. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019.*, Dan Suciu, Sebastian Skritek, and Christoph Koch (Eds.). ACM, 89–103. `https://doi.org/10.1145/3294052.3319702` (Cited on page 9.)

[Ameloot et al. 2013] Tom J. Ameloot, Jan Van den Bussche, and Emmanuel Waller. 2013. On the Expressive Power of Update Primitives. In *Proceedings of the 32Nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '13).*

ACM, New York, NY, USA, 139–150. https://doi.org/10.1145/2463664.2465218 (Cited on page 80.)

[Arnborg et al. 1987] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. 1987. Complexity of Finding Embeddings in a K-tree. *SIAM J. Algebraic Discrete Methods* 8, 2 (April 1987), 277–284. https://doi.org/10.1137/0608024 (Cited on page 40.)

[Arnborg et al. 1991] Stefan Arnborg, Jens Lagergren, and Detlef Seese. 1991. Easy Problems for Tree-Decomposable Graphs. *J. Algorithms* 12, 2 (1991), 308–340. https://doi.org/10.1016/0196-6774(91)90006-K (Cited on page 39.)

[Bagan et al. 2017] Guillaume Bagan, Angela Bonifati, Radu Ciucanu, George H. L. Fletcher, Aurélien Lemay, and Nicky Advokaat. 2017. gMark: Schema-Driven Generation of Graphs and Queries. *IEEE Trans. Knowl. Data Eng.* 29, 4 (2017), 856–869. https://doi.org/10.1109/TKDE.2016.2633993 (Cited on pages 142 and 144.)

[Balmin et al. 2004] Andrey Balmin, Yannis Papakonstantinou, and Victor Vianu. 2004. Incremental validation of XML documents. *ACM Trans. Database Syst.* 29, 4 (2004), 710–751. https://doi.org/10.1145/1042046.1042050 (Cited on page 9.)

[Barbosa et al. 2004] Denilson Barbosa, Alberto O. Mendelzon, Leonid Libkin, Laurent Mignet, and Marcelo Arenas. 2004. Efficient Incremental Validation of XML Documents. In *Proceedings of the 20th International Conference on Data Engineering, ICDE 2004, 30 March - 2 April 2004, Boston, MA, USA*, Z. Meral Özsoyoglu and Stanley B. Zdonik (Eds.). IEEE Computer Society, 671–682. https://doi.org/10.1109/ICDE.2004.1320036 (Cited on page 9.)

[Barrington et al. 1990] David A. Mix Barrington, Neil Immerman, and Howard Straubing. 1990. On Uniformity within NC^1. *J. Comput. Syst. Sci.* 41, 3 (1990), 274–306. https://doi.org/10.1016/0022-0000(90)90022-D (Cited on pages 4, 15, 22, 128, and 132.)

[Berkholz et al. 2017] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. 2017. Answering Conjunctive Queries under Updates. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts (Eds.). ACM, 303–318. https://doi.org/10.1145/3034786.3034789 (Cited on page 9.)

[Berkholz et al. 2018a] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. 2018a. Answering FO+MOD Queries under Updates on Bounded Degree Databases. *ACM Trans. Database Syst.* 43, 2 (2018), 7:1–7:32. https://doi.org/10.1145/3232056 (Cited on page 9.)

[Berkholz et al. 2018b] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. 2018b. Answering UCQs under Updates and in the Presence of Integrity Constraints. In *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018,* *Vienna, Austria (LIPIcs)*, Benny Kimelfeld and Yael Amsterdamer (Eds.), Vol. 98. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 8:1–8:19. `https://doi.org/10.4230/LIPIcs.ICDT.2018.8` (Cited on page 9.)

[Björklund et al. 2010] Henrik Björklund, Wouter Gelade, and Wim Martens. 2010. Incremental XPath evaluation. *ACM Trans. Database Syst.* 35, 4 (2010), 29:1–29:43. `https://doi.org/10.1145/1862919.1862926` (Cited on page 9.)

[Blakeley et al. 1989] José A. Blakeley, Neil Coburn, and Per-Åke Larson. 1989. Updating Derived Relations: Detecting Irrelevant and Autonomously Computable Updates. *ACM Trans. Database Syst.* 14, 3 (Sept. 1989), 369–400. `https://doi.org/10.1145/68012.68015` (Cited on page 9.)

[Blumensath et al. 2008] Achim Blumensath, Thomas Colcombet, and Christof Löding. 2008. Logical theories and compatible operations. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas] (Texts in Logic and Games)*, Jörg Flum, Erich Grädel, and Thomas Wilke (Eds.), Vol. 2. Amsterdam University Press, 73–106. (Cited on pages 14 and 49.)

[Bodlaender 1996] Hans L. Bodlaender. 1996. A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. *SIAM J. Comput.* 25, 6 (1996), 1305–1317. `https://doi.org/10.1137/S0097539793251219` (Cited on page 40.)

[Bouyer and Jugé 2017] Patricia Bouyer and Vincent Jugé. 2017. Dynamic Complexity of the Dyck Reachability. In *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings (Lecture Notes in Computer Science)*, Javier Esparza and Andrzej S. Murawski (Eds.), Vol. 10203. 265–280. `https://doi.org/10.1007/978-3-662-54458-7_16` (Cited on page 101.)

[Bouyer-Decitre et al. 2017] Patricia Bouyer-Decitre, Vincent Jugé, and Nicolas Markey. 2017. Courcelle's Theorem Made Dynamic. *CoRR* abs/1702.05183 (2017). arXiv:1702.05183 `http://arxiv.org/abs/1702.05183` (Cited on page 37.)

[Chandra et al. 1984] Ashok K. Chandra, Larry J. Stockmeyer, and Uzi Vishkin. 1984. Constant Depth Reducibility. *SIAM J. Comput.* 13, 2 (1984), 423–439. `https://doi.org/10.1137/0213028` (Cited on page 138.)

[Chen et al. 2016] Xi Chen, Igor Carboni Oliveira, Rocco A. Servedio, and Li-Yang Tan. 2016. Near-optimal small-depth lower bounds for small distance connectivity. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, Daniel Wichs and Yishay Mansour (Eds.). ACM, 612–625. `https://doi.org/10.1145/2897518.2897534` (Cited on page 118.)

[Comon et al. 2007] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. 2007. Tree Automata Techniques and Applications. Available on: `http://www.grappa.univ-lille3.fr/tata`. release October, 12th 2007. (Cited on page 117.)

[Cook 1985] Stephen A. Cook. 1985. A Taxonomy of Problems with Fast Parallel Algorithms. *Information and Control* 64, 1-3 (1985), 2–21. `https://doi.org/10.1016/S0019-9958(85)80041-3` (Cited on page 132.)

[Courcelle 1990] Bruno Courcelle. 1990. The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *Inf. Comput.* 85, 1 (1990), 12–75. `https://doi.org/10.1016/0890-5401(90)90043-H` (Cited on pages 39 and 40.)

[Courcelle 1994] Bruno Courcelle. 1994. The Monadic Second order Logic of Graphs VI: on Several Representations of Graphs By Relational Structures. *Discrete Applied Mathematics* 54, 2-3 (1994), 117–149. `https://doi.org/10.1016/0166-218X(94)90019-1` (Cited on page 46.)

[Courcelle and Engelfriet 2012] Bruno Courcelle and Joost Engelfriet. 2012. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach.* Encyclopedia of mathematics and its applications, Vol. 138. Cambridge University Press, New York, NY, USA. (Cited on page 46.)

[Datta et al. 2014] 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, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I (Lecture Notes in Computer Science)*, Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias (Eds.), Vol. 8572. Springer, 356–367. `https://doi.org/10.1007/978-3-662-43948-7_30` (Cited on pages 5 and 36.)

[Datta et al. 2015] Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. 2015. Reachability is in DynFO. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II (Lecture Notes in Computer Science)*, Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann (Eds.), Vol. 9135. Springer, 159–170. `https://doi.org/10.1007/978-3-662-47666-6_13` (Cited on pages 27, 33, 36, and 151.)

[Datta et al. 2018a] Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. 2018a. Reachability Is in DynFO. *J. ACM* 65, 5, Article 33 (Aug. 2018), 24 pages. `https://doi.org/10.1145/3212685` (Cited on pages 3, 5, 19, 20, 21, 22, 23, 25, 26, 28, 29, 32, 33, 36, 101, 114, 126, 149, 151, and 155.)

[Datta et al. 2009] Samir Datta, Nutan Limaye, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. 2009. Planar Graph Isomorphism is in Log-Space. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris,* *France, 15-18 July 2009*. IEEE Computer Society, 203–214. `https://doi.org/10.1109/CCC.2009.16` (Cited on page 78.)

[Datta et al. 2017] Samir Datta, Anish Mukherjee, Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. 2017. A Strategy for Dynamic Programs: Start over and Muddle Through. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland (LIPIcs)*, Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl (Eds.), Vol. 80. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 98:1–98:14. `https://doi.org/10.4230/LIPIcs.ICALP.2017.98` (Cited on pages 28, 33, and 78.)

[Datta et al. 2019] Samir Datta, Anish Mukherjee, Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. 2019. A Strategy for Dynamic Programs: Start over and Muddle through. *Logical Methods in Computer Science* Volume 15, Issue 2 (May 2019). `https://doi.org/10.23638/LMCS-15(2:12)2019` (Cited on pages 28, 33, 52, and 78.)

[Datta et al. 2018b] Samir Datta, Anish Mukherjee, Nils Vortmeier, and Thomas Zeume. 2018b. Reachability and Distances under Multiple Changes. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic (LIPIcs)*, Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella (Eds.), Vol. 107. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 120:1–120:14. `https://doi.org/10.4230/LIPIcs.ICALP.2018.120` (Cited on pages 32, 33, 114, 126, 127, 130, 138, 139, 140, 151, and 154.)

[Demetrescu et al. 2010] Camil Demetrescu, David Eppstein, Zvi Galil, and Giuseppe F. Italiano. 2010. Algorithms and Theory of Computation Handbook. Chapman & Hall/CRC, Chapter Dynamic Graph Algorithms, 9–9. `http://dl.acm.org/citation.cfm?id=1882757.1882766` (Cited on page 2.)

[Denenberg et al. 1986] Larry Denenberg, Yuri Gurevich, and Saharon Shelah. 1986. Definability by Constant-Depth Polynomial-Size Circuits. *Information and Control* 70, 2/3 (1986), 216–240. `https://doi.org/10.1016/S0019-9958(86)80006-7` (Cited on page 115.)

[Dong et al. 1999] Guozhu Dong, Leonid Libkin, Jianwen Su, and Limsoon Wong. 1999. Maintaining Transitive Closure of Graphs in SQL. *Int. Journal of Information Technology* 51, 1 (1999), 46–78. (Cited on pages 10 and 141.)

[Dong et al. 2003] Guozhu Dong, Leonid Libkin, and Limsoon Wong. 2003. Incremental recomputation in local languages. *Inf. Comput.* 181, 2 (2003), 88–98. `https://doi.org/10.1016/S0890-5401(03)00017-8` (Cited on pages 5 and 37.)

[Dong and Pang 1997] Guozhu Dong and Chaoyi Pang. 1997. Maintaining Transitive Closure in First Order After Node-Set and Edge-Set Deletions. *Inf. Process. Lett.* 62, 4 (1997), 193–199. `https://doi.org/10.1016/S0020-0190(97)00066-5` (Cited on pages 4 and 6.)

[Dong and Su 1993] 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, Manhattan, New York City, USA, 30 August - 1 September 1993 (Workshops in Computing)*, Catriel Beeri, Atsushi Ohori, and Dennis E. Shasha (Eds.). Springer, 295–308. (Cited on page 2.)

[Dong and Su 1995] Guozhu Dong and Jianwen Su. 1995. Incremental and Decremental Evaluation of Transitive Closure by First-Order Queries. *Inf. Comput.* 120, 1 (1995), 101–106. `https://doi.org/10.1006/inco.1995.1102` (Cited on pages 5, 20, 22, 23, 36, and 83.)

[Dong and Su 1997] Guozhu Dong and Jianwen Su. 1997. Deterministic FOIES are Strictly Weaker. *Ann. Math. Artif. Intell.* 19, 1-2 (1997), 127–146. `https://doi.org/10.1023/A:1018951521198` (Cited on pages 5, 20, 100, and 153.)

[Dong and Su 1998] 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. `https://doi.org/10.1006/jcss.1998.1565` (Cited on pages 5, 10, 22, 36, 37, 61, 74, 83, 85, 86, 122, 123, 126, and 143.)

[Dong et al. 1995] Guozhu Dong, Jianwen Su, and Rodney W. Topor. 1995. Nonrecursive Incremental Evaluation of Datalog Queries. *Ann. Math. Artif. Intell.* 14, 2-4 (1995), 187–223. `https://doi.org/10.1007/BF01530820` (Cited on pages 2, 6, 20, 22, 23, 36, and 110.)

[Dong and Topor 1992] Guozhu Dong and Rodney W. Topor. 1992. Incremental Evaluation of Datalog Queries. In *Database Theory - ICDT'92, 4th International Conference, Berlin, Germany, October 14-16, 1992, Proceedings (Lecture Notes in Computer Science)*, Joachim Biskup and Richard Hull (Eds.), Vol. 646. Springer, 282–296. `https://doi.org/10.1007/3-540-56039-4_48` (Cited on page 2.)

[Dong and Zhang 2000] Guozhu Dong and Louxin Zhang. 2000. Separating Auxiliary Arity Hierarchy of First-Order Incremental Evaluation Systems Using (3k+1)-ary Input Relations. *Int. J. Found. Comput. Sci.* 11, 4 (2000), 573–578. `https://doi.org/10.1142/S0129054100000302` (Cited on pages 5 and 61.)

[Durand et al. 2006] Arnaud Durand, Clemens Lautemann, and Malika More. 2006. Counting Results in Weak Formalisms. In *Circuits, Logic, and Games, 08.11. - 10.11.2006 (Dagstuhl Seminar Proceedings)*, Thomas Schwentick, Denis Thérien, and Heribert Vollmer (Eds.), Vol. 06451. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany. `http://drops.dagstuhl.de/opus/volltexte/2007/976` (Cited on page 121.)

[Eisenberg and Melton 1999] Andrew Eisenberg and Jim Melton. 1999. SQL: 1999, Formerly Known As SQL3. *SIGMOD Rec.* 28, 1 (March 1999), 131–138. `https://doi.org/10.1145/309844.310075` (Cited on page 3.)

[Elberfeld et al. 2016] Michael Elberfeld, Martin Grohe, and Till Tantau. 2016. Where First-Order and Monadic Second-Order Logic Coincide. *ACM Trans. Comput. Log.* 17, 4 (2016), 25:1–25:18. `https://doi.org/10.1145/2946799` (Cited on page 47.)

[Elberfeld et al. 2010] Michael Elberfeld, Andreas Jakoby, and Till Tantau. 2010. Logspace Versions of the Theorems of Bodlaender and Courcelle. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*. IEEE Computer Society, 143–152. `https://doi.org/10.1109/FOCS.2010.21` (Cited on pages 39, 41, and 76.)

[Etessami 1998] 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, June 1-3, 1998, Seattle, Washington, USA*, Alberto O. Mendelzon and Jan Paredaens (Eds.). ACM Press, 235–243. `https://doi.org/10.1145/275487.275514` (Cited on pages 4, 5, 21, 22, 23, 36, 37, 74, 76, 78, 80, and 86.)

[Fagin et al. 1985] Ronald Fagin, Maria M. Klawe, Nicholas Pippenger, and Larry J. Stockmeyer. 1985. Bounded-Depth, Polynomial-Size Circuits for Symmetric Functions. *Theor. Comput. Sci.* 36 (1985), 239–250. `https://doi.org/10.1016/0304-3975(85)90045-3` (Cited on page 115.)

[Feferman 1957] Solomon Feferman. 1957. Some recent work of Ehrenfeucht and Fraïssé. In *Proc. Summer Institute of Symbolic Logic*. 201–209. (Cited on page 14.)

[Feferman and Vaught 1959] Solomon Feferman and Robert L. Vaught. 1959. The first order properties of algebraic systems. *Fund. Math.* 47 (1959), 57–103. (Cited on page 14.)

[Furst et al. 1984] Merrick L. Furst, James B. Saxe, and Michael Sipser. 1984. Parity, Circuits, and the Polynomial-Time Hierarchy. *Mathematical Systems Theory* 17, 1 (1984), 13–27. `https://doi.org/10.1007/BF01744431` (Cited on pages 19 and 115.)

[Gelade et al. 2012] Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. 2012. The dynamic complexity of formal languages. *ACM Trans. Comput. Log.* 13, 3 (2012), 19. `https://doi.org/10.1145/2287718.2287719` (Cited on pages 5, 21, 22, 25, 36, 37, 38, 39, 58, 60, 66, 67, 78, 100, 101, 102, 103, 104, 124, 125, 126, 150, and 152.)

[Godsil 1993] Chris D. Godsil. 1993. *Algebraic combinatorics*. Chapman and Hall. (Cited on pages 129 and 130.)

[Gottlob et al. 2001] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2001. The complexity of acyclic conjunctive queries. *J. ACM* 48, 3 (2001), 431–498. `https://doi.org/10.1145/382780.382783` (Cited on page 87.)

[Grädel and Siebertz 2012] Erich Grädel and Sebastian Siebertz. 2012. Dynamic definability. In *15th International Conference on Database Theory, ICDT '12, Berlin, Germany,* *March 26-29, 2012*, Alin Deutsch (Ed.). ACM, 236–248. `https://doi.org/10.1145/2274576.2274601` (Cited on pages 4, 5, 19, 21, 22, 23, 36, 37, 80, and 153.)

[Grohe 2014] Martin Grohe. 2014. Algorithmic Meta Theorems for Sparse Graph Classes. In *Computer Science - Theory and Applications - 9th International Computer Science Symposium in Russia, CSR 2014, Moscow, Russia, June 7-11, 2014. Proceedings (Lecture Notes in Computer Science)*, Edward A. Hirsch, Sergei O. Kuznetsov, Jean-Éric Pin, and Nikolay K. Vereshchagin (Eds.), Vol. 8476. Springer, 16–22. `https://doi.org/10.1007/978-3-319-06686-8_2` (Cited on pages 38 and 39.)

[Gupta and Mumick 1999] Ashish Gupta and Iderpal Singh Mumick (Eds.). 1999. *Materialized Views: Techniques, Implementations, and Applications.* MIT Press, Cambridge, MA, USA. (Cited on page 8.)

[Håstad 1986] Johan Håstad. 1986. Almost Optimal Lower Bounds for Small Depth Circuits. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, Juris Hartmanis (Ed.). ACM, 6–20. `https://doi.org/10.1145/12130.12132` (Cited on pages 119 and 120.)

[Henderson and Searle 1981] H. Henderson and S. Searle. 1981. On Deriving the Inverse of a Sum of Matrices. *SIAM Rev.* 23, 1 (1981), 53–60. `https://doi.org/10.1137/1023004` (Cited on pages 127 and 130.)

[Henzinger and Fredman 1998] Monika Rauch Henzinger and Michael L. Fredman. 1998. Lower Bounds for Fully Dynamic Connectivity Problems in Graphs. *Algorithmica* 22, 3 (1998), 351–362. `https://doi.org/10.1007/PL00009228` (Cited on page 2.)

[Hesse 2003a] William Hesse. 2003a. The dynamic complexity of transitive closure is in DynTC^0. *Theor. Comput. Sci.* 296, 3 (2003), 473–485. `https://doi.org/10.1016/S0304-3975(02)00740-5` (Cited on pages 5, 36, 128, 129, and 130.)

[Hesse 2003b] William Hesse. 2003b. *Dynamic Computational Complexity.* Ph.D. Dissertation. University of Massachusetts Amherst. (Cited on pages 5, 10, 19, 21, 22, 23, 32, 36, 37, 100, and 150.)

[Hesse et al. 2002] William Hesse, Eric Allender, and David A. Mix Barrington. 2002. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. Syst. Sci.* 65, 4 (2002), 695–716. `https://doi.org/10.1016/S0022-0000(02)00025-9` (Cited on pages 131, 132, 133, 135, and 136.)

[Hesse and Immerman 2002] William Hesse and Neil Immerman. 2002. Complete Problems for Dynamic Complexity Classes. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings.* IEEE Computer Society, 313. `https://doi.org/10.1109/LICS.2002.1029839` (Cited on pages 5 and 19.)

[Idris et al. 2017] Muhammad Idris, Martín Ugarte, and Stijn Vansummeren. 2017. The Dynamic Yannakakis Algorithm: Compact and Efficient Query Processing Under Updates. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu (Eds.). ACM, 1259–1274. `https://doi.org/10.1145/3035918.3064027` (Cited on page 9.)

[Idris et al. 2018] Muhammad Idris, Martín Ugarte, Stijn Vansummeren, Hannes Voigt, and Wolfgang Lehner. 2018. Conjunctive Queries with Inequalities Under Updates. *PVLDB* 11, 7 (2018), 733–745. `https://doi.org/10.14778/3192965.3192966` (Cited on page 9.)

[Immerman 1987] Neil Immerman. 1987. Languages that Capture Complexity Classes. *SIAM J. Comput.* 16, 4 (1987), 760–778. `https://doi.org/10.1137/0216051` (Cited on page 120.)

[Immerman 1999] Neil Immerman. 1999. *Descriptive complexity.* Springer. `https://doi.org/10.1007/978-1-4612-0539-5` (Cited on pages 2, 3, 13, 15, 30, and 76.)

[Impagliazzo and Paturi 2001] Russell Impagliazzo and Ramamohan Paturi. 2001. On the Complexity of k-SAT. *J. Comput. Syst. Sci.* 62, 2 (2001), 367–375. `https://doi.org/10.1006/jcss.2000.1727` (Cited on page 139.)

[Jenner et al. 1998] Birgit Jenner, Pierre McKenzie, and Jacobo Torán. 1998. A Note on the Hardness of Tree Isomorphism. In *Proceedings of the 13th Annual IEEE Conference on Computational Complexity, Buffalo, New York, USA, June 15-18, 1998.* IEEE Computer Society, 101–105. `https://doi.org/10.1109/CCC.1998.694595` (Cited on page 38.)

[Kähler and Wilke 2003] Detlef Kähler and Thomas Wilke. 2003. Program Complexity of Dynamic LTL Model Checking. In *Computer Science Logic, 17th International Workshop, CSL 2003, 12th Annual Conference of the EACSL, and 8th Kurt Gödel Colloquium, KGC 2003, Vienna, Austria, August 25-30, 2003, Proceedings (Lecture Notes in Computer Science)*, Matthias Baaz and Johann A. Makowsky (Eds.), Vol. 2803. Springer, 271–284. `https://doi.org/10.1007/978-3-540-45220-1_23` (Cited on page 128.)

[Kara et al. 2019] Ahmet Kara, Hung Q. Ngo, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. 2019. Counting Triangles under Updates in Worst-Case Optimal Time. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal (LIPIcs)*, Pablo Barceló and Marco Calautti (Eds.), Vol. 127. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 4:1–4:18. `https://doi.org/10.4230/LIPIcs.ICDT.2019.4` (Cited on page 9.)

[Koch 2010] Christoph Koch. 2010. Incremental query evaluation in a ring of databases. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on*

Principles of Database Systems (PODS). 87–98. https://doi.org/10.1145/1807085.1807100 (Cited on pages 9 and 141.)

[Koch et al. 2014] Christoph Koch, Yanif Ahmad, Oliver Kennedy, Milos Nikolic, Andres Nötzli, Daniel Lupei, and Amir Shaikhha. 2014. DBToaster: higher-order delta processing for dynamic, frequently fresh views. *VLDB J.* 23, 2 (2014), 253–278. https://doi.org/10.1007/s00778-013-0348-4 (Cited on pages 9 and 141.)

[Ladner and Fischer 1980] Richard E. Ladner and Michael J. Fischer. 1980. Parallel Prefix Computation. *J. ACM* 27, 4 (Oct. 1980), 831–838. https://doi.org/10.1145/322217.322232 (Cited on page 103.)

[Libkin 2003] Leonid Libkin. 2003. Expressive power of SQL. *Theor. Comput. Sci.* 296, 3 (2003), 379–404. https://doi.org/10.1016/S0304-3975(02)00736-3 (Cited on page 3.)

[Libkin 2004] Leonid Libkin. 2004. *Elements of Finite Model Theory.* Springer. (Cited on page 11.)

[Libkin and Wong 1997] Leonid Libkin and Limsoon Wong. 1997. Incremental Recomputation of Recursive Queries with Nested Sets and Aggregate Functions. In *Database Programming Languages, 6th International Workshop, DBPL-6, Estes Park, Colorado, USA, August 18-20, 1997, Proceedings (Lecture Notes in Computer Science)*, Sophie Cluet and Richard Hull (Eds.), Vol. 1369. Springer, 222–238. https://doi.org/10.1007/3-540-64823-2_13 (Cited on pages 9 and 10.)

[Mahajan and Vinay 1999] Meena Mahajan and V. Vinay. 1999. Determinant: Old Algorithms, New Insights. *SIAM J. Discrete Math.* 12, 4 (1999), 474–490. https://doi.org/10.1137/S0895480198338827 (Cited on page 132.)

[Makowsky 2004] Johann A. Makowsky. 2004. Algorithmic uses of the Feferman-Vaught Theorem. *Ann. Pure Appl. Logic* 126, 1-3 (2004), 159–213. https://doi.org/10.1016/j.apal.2003.11.002 (Cited on pages 14 and 55.)

[Miltersen 1999] Peter Bro Miltersen. 1999. Cell Probe Complexity - a Survey. In *19th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 1999. Advances in Data Structures Workshop.* (Cited on page 2.)

[Miltersen et al. 1994] 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. https://doi.org/10.1016/0304-3975(94)90159-7 (Cited on page 9.)

[Mukherjee 2019] Anish Mukherjee. 2019. *Static and Dynamic Complexity of Reachability, Matching and Related Problems.* Ph.D. Dissertation. Chennai Mathematical Institute. (Cited on page 138.)

[Muñoz et al. 2016] Pablo Muñoz, Nils Vortmeier, and Thomas Zeume. 2016. Dynamic Graph Queries. In *19th International Conference on Database Theory, ICDT 2016, Bordeaux, France, March 15-18, 2016 (LIPIcs)*, Wim Martens and Thomas Zeume (Eds.), Vol. 48. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 14:1–14:18. `https://doi.org/10.4230/LIPIcs.ICDT.2016.14` (Cited on pages 36 and 101.)

[Nikolic et al. 2016] Milos Nikolic, Mohammad Dashti, and Christoph Koch. 2016. How to Win a Hot Dog Eating Contest: Distributed Incremental View Maintenance with Batch Updates. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 511–526. `https://doi.org/10.1145/2882903.2915246` (Cited on pages 9 and 142.)

[Otto 2000] Martin Otto. 2000. Epsilon-Logic Is More Expressive Than First-Order Logic Over Finite Structures. *J. Symb. Log.* 65, 4 (2000), 1749–1757. `https://doi.org/10.2307/2695073` (Cited on page 28.)

[Pang et al. 2005] Chaoyi Pang, Guozhu Dong, and Kotagiri Ramamohanarao. 2005. Incremental maintenance of shortest distance and transitive closure in first-order logic and SQL. *ACM Trans. Database Syst.* 30, 3 (2005), 698–721. `https://doi.org/10.1145/1093382.1093384` (Cited on pages 22, 141, and 142.)

[Patnaik and Immerman 1994] 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, May 24-26, 1994, Minneapolis, Minnesota, USA*, Victor Vianu (Ed.). ACM Press, 210–221. `https://doi.org/10.1145/182591.182614` (Cited on pages 2, 22, and 23.)

[Patnaik and Immerman 1997] Sushant Patnaik and Neil Immerman. 1997. Dyn-FO: A Parallel, Dynamic Complexity Class. *J. Comput. Syst. Sci.* 55, 2 (1997), 199–209. `https://doi.org/10.1006/jcss.1997.1520` (Cited on pages 2, 4, 5, 9, 11, 17, 18, 19, 20, 21, 22, 23, 35, 36, 80, 83, 92, 93, 100, 153, 154, and 155.)

[Przymus et al. 2010] Piotr Przymus, Aleksandra Boniewicz, Marta Burzanska, and Krzysztof Stencel. 2010. Recursive Query Facilities in Relational Databases: A Survey. In *Database Theory and Application, Bio-Science and Bio-Technology - International Conferences, DTA and BSBT 2010. Proceedings (Communications in Computer and Information Science)*, Yanchun Zhang, Alfredo Cuzzocrea, Jianhua Ma, Kyo-Il Chung, Tughrul Arslan, and Xiaofeng Song (Eds.), Vol. 118. Springer, 89–99. `https://doi.org/10.1007/978-3-642-17622-7_10` (Cited on page 3.)

[Robertson and Seymour 1986] Neil Robertson and Paul D. Seymour. 1986. Graph Minors. II. Algorithmic Aspects of Tree-Width. *J. Algorithms* 7, 3 (1986), 309–322. `https://doi.org/10.1016/0196-6774(86)90023-4` (Cited on page 39.)

[Robertson and Seymour 1990] Neil Robertson and Paul D. Seymour. 1990. Graph minors. IV. Tree-width and well-quasi-ordering. *J. Comb. Theory, Ser. B* 48, 2 (1990), 227–254. `https://doi.org/10.1016/0095-8956(90)90120-O` (Cited on page 77.)

[Schmidt et al. 2020] Jonas Schmidt, Thomas Schwentick, Nils Vortmeier, Thomas Zeume, and Ioannis Kokkinis. 2020. Dynamic Complexity Meets Parameterised Algorithms. (2020). Accepted for publication at CSL 2020. (Cited on pages 32 and 140.)

[Schwentick et al. 2015] 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, September 7-10, 2015, Berlin, Germany (LIPIcs)*, Stephan Kreutzer (Ed.), Vol. 41. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 308–324. `https://doi.org/10.4230/LIPIcs.CSL.2015.308` (Cited on pages 153 and 154.)

[Schwentick et al. 2017a] Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. 2017a. Dynamic Complexity under Definable Changes. In *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy (LIPIcs)*, Michael Benedikt and Giorgio Orsi (Eds.), Vol. 68. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 19:1–19:18. `https://doi.org/10.4230/LIPIcs.ICDT.2017.19` (Cited on pages 33, 111, and 112.)

[Schwentick et al. 2017b] Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. 2017b. Dynamic Complexity under Definable Changes. *CoRR* abs/1701.02494 (2017). arXiv:1701.02494 `http://arxiv.org/abs/1701.02494` (Cited on page 94.)

[Schwentick et al. 2018] Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. 2018. Dynamic Complexity Under Definable Changes. *ACM Trans. Database Syst.* 43, 3, Article 12 (Oct. 2018), 38 pages. `https://doi.org/10.1145/3241040` (Cited on pages 17, 23, 29, 78, 111, 112, and 150.)

[Schwentick and Zeume 2016] Thomas Schwentick and Thomas Zeume. 2016. Dynamic complexity: recent updates. *SIGLOG News* 3, 2 (2016), 30–52. `http://doi.acm.org/10.1145/2948896.2948899` (Cited on pages 17, 20, and 23.)

[Selberg 1949] Atle Selberg. 1949. An elementary proof of the prime-number theorem. *Ann. Math* 50, 2 (1949), 305–313. (Cited on pages 88 and 90.)

[Shalygina and Novikov 2017] Galina Shalygina and Boris Novikov. 2017. Implementing Common Table Expressions for MariaDB. In *Second Conference on Software Engineering and Information Management (SEIM-2017)*. 12–17. (Cited on page 3.)

[Shelah 1975] Saharon Shelah. 1975. The monadic theory of order. *Annals of Mathematics* (1975), 379–419. `https://doi.org/10.2307/1971037` (Cited on page 49.)

[Siebertz 2011] Sebastian Siebertz. 2011. *Dynamic Definability.* Diploma Thesis. RWTH Aachen. (Cited on page 6.)

[Vista 1998] Dimitra Vista. 1998. Integration of Incremental View Maintenance into Query Optimizers. In *Advances in Database Technology - EDBT'98, 6th International Conference on Extending Database Technology, Valencia, Spain, March 23-27, 1998, Proceedings (Lecture Notes in Computer Science)*, Hans-Jörg Schek, Fèlix Saltor, Isidro Ramos, and Gustavo Alonso (Eds.), Vol. 1377. Springer, 374–388. `https://doi.org/10.1007/BFb0100997` (Cited on page 142.)

[Vollmer 1999] Heribert Vollmer. 1999. *Introduction to Circuit Complexity - A Uniform Approach*. Springer. (Cited on page 57.)

[Vortmeier 2013] Nils Vortmeier. 2013. *Komplexitätstheorie verlaufsunabhängiger dynamischer Programme*. Master's thesis. TU Dortmund University. (Cited on pages 37, 101, and 152.)

[Vortmeier and Zeume 2020] Nils Vortmeier and Thomas Zeume. 2020. Dynamic Complexity of Parity Exists Queries. (2020). Accepted for publication at CSL 2020. (Cited on page 78.)

[Weber and Schwentick 2007] Volker Weber and Thomas Schwentick. 2007. Dynamic Complexity Theory Revisited. *Theory Comput. Syst.* 40, 4 (2007), 355–377. `https://doi.org/10.1007/s00224-006-1312-0` (Cited on pages 5, 21, 23, 36, and 101.)

[Zeume 2015] Thomas Zeume. 2015. *Small dynamic complexity classes*. Ph.D. Dissertation. TU Dortmund University, Germany. `http://hdl.handle.net/2003/34163` (Cited on pages 4, 10, 20, 21, 36, 37, 61, 68, 74, 78, 80, 94, 101, and 150.)

[Zeume 2017] Thomas Zeume. 2017. The dynamic descriptive complexity of k-clique. *Inf. Comput.* 256 (2017), 9–22. `https://doi.org/10.1016/j.ic.2017.04.005` (Cited on pages 5, 21, 37, 38, 61, 66, 67, and 78.)

[Zeume and Schwentick 2015] Thomas Zeume and Thomas Schwentick. 2015. On the quantifier-free dynamic complexity of Reachability. *Inf. Comput.* 240 (2015), 108–129. `https://doi.org/10.1016/j.ic.2014.09.011` (Cited on pages 5, 21, 23, 25, 37, 58, 61, 66, 67, 68, 73, 78, 96, 98, and 154.)

[Zeume and Schwentick 2017] Thomas Zeume and Thomas Schwentick. 2017. Dynamic conjunctive queries. *J. Comput. Syst. Sci.* 88 (2017), 3–26. `https://doi.org/10.1016/j.jcss.2017.03.014` (Cited on pages 5, 19, 21, 22, 33, and 108.)