A Strategy for Dynamic Programs: Start over and Muddle Through*†

Samir Datta[1], Anish Mukherjee[2], Thomas Schwentick[3], Nils Vortmeier[4], and Thomas Zeume[5]

1 **Chennai Mathematical Institute, Chennai, India**
`sdatta@cmi.ac.in`
2 **Chennai Mathematical Institute, Chennai, India**
`anish@cmi.ac.in`
3 **TU Dortmund, Dortmund, Germany**
`thomas.schwentick@tu-dortmund.de`
4 **TU Dortmund, Dortmund, Germany**
`nils.vortmeier@tu-dortmund.de`
5 **TU Dortmund, Dortmund, Germany**
`thomas.zeume@tu-dortmund.de`

Abstract

A strategy for constructing dynamic programs is introduced that utilises periodic computation of auxiliary data from scratch and the ability to maintain a query for a limited number of change steps. It is established that if some program can maintain a query for $\log n$ change steps after an AC^1-computable initialisation, it can be maintained by a first-order dynamic program as well, i.e., in DynFO. As an application, it is shown that decision and optimisation problems defined by monadic second-order (MSO) and guarded second-order logic (GSO) formulas are in DynFO, if only change sequences that produce graphs of bounded treewidth are allowed. To establish this result, Feferman–Vaught-type composition theorems for MSO and GSO are established that might be useful in their own right.

1998 ACM Subject Classification F.1.2 Modes of Computation

Keywords and phrases dynamic complexity, treewidth, monadic second order logic

Digital Object Identifier 10.4230/LIPIcs.ICALP.2017.98

1 Introduction

Updating the result of a query after a small change to a relational database is an important problem. A theoretical framework for studying when a query can be updated in a declarative fashion was formalised by Patnaik and Immerman [11], and Dong, Su, and Topor [5]. In their formalisation, a dynamic program has a set of logical formulas that update a query after the insertion or deletion of a tuple. The formulas may use additional auxiliary relations that, of course, need to be updated as well. The queries maintainable in this way via first-order formulas constitute the dynamic complexity class DynFO.

* The full version of this paper is available as [4], `http://arxiv.org/abs/1704.07998`.

† The authors acknowledge the financial support by the DAAD-DST grant “Exploration of New Frontiers in Dynamic Complexity”. The first and the second authors were partially funded by a grant from Infosys foundation. The second author was partially supported by a TCS PhD fellowship. The last three authors acknowledge the financial support by DFG grant SCHW 678/6-2 on “Dynamic Expressiveness of Logics”.

44th International Colloquium on Automata, Languages, and Programming (ICALP 2017).
Editors: Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl;
Article No. 98; pp. 98:1–98:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Recent work has confirmed that DynFO is quite a powerful class, since it captures, e.g., the reachability query for directed graphs [2], and can even take care of pretty complex change operations [12].

In this paper, we introduce a general strategy for dynamic programs that further underscores the expressive power of DynFO. For a complexity class $\mathcal{C}$ and a function f, we call a query $\mathcal{Q}$ $(\mathcal{C}, f)$-maintainable, if there is a dynamic program (with first-order definable updates) that, starting from some input structure $\mathcal{A}$ and auxiliary relations computed in $\mathcal{C}$ from $\mathcal{A}$, can answer $\mathcal{Q}$ for $f(|\mathcal{A}|)$ many steps, where $|\mathcal{A}|$ denotes the size of the universe of $\mathcal{A}$.

We feel that this notion might be interesting in its own right. However, in this paper we concentrate on the case where $\mathcal{C}$ is (uniform) AC^1 and $f(n) = \log n$. We show that $(\mathrm{AC}^1, \log n)$-maintainable queries are actually in DynFO. We apply this insight to show that all queries and optimisation problems definable in monadic second-order logic (MSO) are in DynFO for (classes of) structures of bounded treewidth, by proving that they are $(\mathrm{AC}^1, \log n)$-maintainable. Likewise for guarded second-order logic (GSO). This implies that decision problems like 3-Colourability or HamiltonCycle as well as optimisation problems like VertexCover and DominatingSet are in DynFO, for such classes of structures.

The proof that MSO-definable queries are $(\mathrm{AC}^1, \log n)$-maintainable on structures of bounded treewidth makes use of a Feferman–Vaught-type composition theorem for MSO which might be useful in its own right.

The result that $(\mathrm{AC}^1, \log n)$-maintainable queries are in DynFO comes with a technical restriction: in a nutshell, it holds for queries that are invariant under insertion of (many) isolated elements. We call such queries *almost domain-independent* and refer to Section 3 for a precise definition.

We emphasise that the main technical challenge in maintaining MSO-queries on graphs of bounded treewidth is that tree decompositions might change drastically after an edge insertion, and can therefore not be maintained incrementally in any obvious way. In particular, the result does not simply follow from the DynFO-maintainability of regular tree languages shown in [8]. We circumvent this problem by periodically recomputing a new tree decomposition (in logarithmic space and thus in AC^1) and by showing that MSO-queries can be maintained for $\mathcal{O}(\log n)$ many change operations, even if they make the tree decomposition invalid.

Contributions

- We introduce the notion of $(\mathcal{C}, f)$-maintainability.
- We show that (almost domain-independent) $(\mathrm{AC}^1, \log n)$-maintainable queries are in DynFO.
- We show that MSO-definable (Boolean) queries are $(\mathrm{AC}^1, \log n)$-maintainable and therefore in DynFO, for structures of bounded treewidth. Likewise for MSO-definable optimisation problems and GSO-definable queries and optimisation problems.
- We state a Feferman–Vaught-type composition theorem for MSO-logic.

Related work. The simulation-based technique for proving that $(\mathrm{AC}^1, \log n)$-maintainable queries are in DynFO is inspired by proof techniques from [2] and [12]. As mentioned above, in [8] it has been shown that tree languages, i.e. MSO on trees, can be maintained in DynFO. In [1], the maintenance of parity games has been studied for graphs of bounded treewidth, though in the restricted setting where the tree decomposition stays the same for all changes.

Organisation. Basic terminology is recalled in Section 2, followed by a short introduction into dynamic complexity in Section 3. In Section 4 we introduce the notion of $(\mathcal{C}, f)$-maintainability and show that $(\mathrm{AC}^1, \log n)$-maintainable queries are in DynFO. A glimpse on the proof techniques for proving that MSO and GSO queries are in DynFO for graphs of bounded treewidth is given in Section 5 via the example 3-Colourability. The proof of the general results is presented in Section 6. An extension to optimisation problems can be found in Section 7. For many proofs, details are deferred to the full version of this paper [4].

2 Preliminaries

We assume familiarity with first-order logic FO and other notions from finite model theory [9, 10]. Some further notation regarding MSO logic and types will be introduced in Section 6.

In this paper we consider finite relational structures over relational signatures $\Sigma = \{R_1, \ldots, R_\ell, c_1, \ldots, c_m\}$, where each R_i is a relation symbol with a corresponding arity $\mathrm{Ar}(R_i)$, and each c_j is a constant symbol. A Σ-structure $\mathcal{A}$ consists of a finite *domain* A, a relation $R_i^{\mathcal{A}} \subseteq A^{\mathrm{Ar}(R_i)}$, and a constant $c_j^{\mathcal{A}} \in A$, for each $i \in \{1, \ldots, \ell\}, j \in \{1, \ldots, m\}$. Sometimes, especially in Section 3, we consider relational structures as relational databases This is basically a different terminology that is common in the context of dynamic complexity, since the original motivation for considering the class DynFO came from relational databases. In particular, the class DynFO will be defined as a class of queries of arbitrary arity.

However, we will mostly consider Boolean queries over structures with a single binary relation symbol E, which can equivalently be viewed as decision problems for graphs $G = (V, E)$. For a set $U \subseteq V$, $G[U]$ denotes the induced subgraph $(U, E \cap (U \times U))$.

We will often use structures that have a linear order $\leq$ and compatible ternary relations encoding arithmetical operations $+$ and $\times$ or a binary BIT relation on the universe. We write $\mathrm{FO}(+, \times)$ or FO(BIT) to emphasise that we allow first-order formulas to use such additional relations.[1] We also use that $\mathrm{FO}(+, \times) = \mathrm{FO}(\mathrm{BIT})$ [9].

A *tree decomposition* (T, B) of G consists of a (rooted) tree $T = (I, F, r)$ and a function $B\colon I \to 2^V$ such that (1) for all $v \in V$, the set $\{i \in I \mid v \in B(i)\}$ is non-empty, (2) for all $(u, v) \in E$, there is an $i \in I$ with $\{u, v\} \subseteq B(i)$, and (3) the subgraph $T[\{i \in I \mid v \in B(i)\}]$ is connected. We refer to the number of children of a node of T as its *degree*. We denote the parent node of a node i by $p(i)$. 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. A tree decomposition is *nice* if (1) T has depth at most $\mathcal{O}(\log n)$, (2) the degree of the nodes is at most 2, and (3) all bags are distinct. We use the following lemma which is an adaption of [7, Lemma 3.1].

▶ **Lemma 1.** *For every graph of treewidth k, a nice tree decomposition of width $4k + 5$ can be computed in logarithmic space.*

In this paper we only consider nice tree decompositions, and due to property (3) of these decompositions we can identify bags with nodes from I.

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, $i' \neq i$. A *triangle* δ of T is a triple (i_0, i_1, i_2) of nodes from I such that $i_1 \preceq i_0$, $i_2 \preceq i_0$, 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*.

[1] The question of $<$-invariance will not be relevant in the context of this paper since the order of insertion of elements to a structure will determine a linear order on the universe.

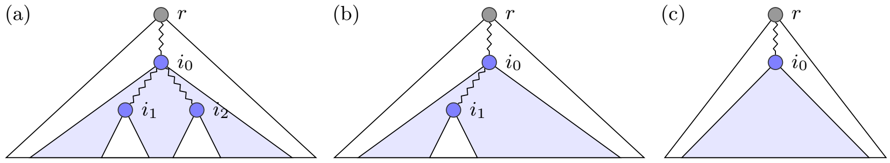

Figure 1 Illustration of (a) a proper triangle (i_0, i_1, i_2), (b) a unary triangle (i_0, i_1, i_1), and (c) an open triangle (i_1, i_1, i_1). The blue shaded area is the part of the tree contained in the triangle.

The subtree $T(\delta)$ *induced* by a triangle consists of all nodes j of T for which the following holds: (i) $j \preceq i_0$, (ii) if $i_1 \prec i_0$ then $j \not\prec i_1$, and (iii) if $i_2 \prec i_0$ then $j \not\prec i_2$. 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 are called *inner nodes*. The edge set of $G(\delta)$ consists of all edges of G that involve at least one inner node of $V(\delta)$.

Our main result refers to the complexity class (uniform) AC^1 whose definition can be found, e.g., in [15]. The precise definition of the class is not relevant for this paper. It suffices to know that it contains the classes LOGSPACE and NL and that it can be characterised as the class IND$[\log n]$ of problems that can be expressed by applying a first-order formula $\mathcal{O}(\log n)$ times [9, Theorem 5.22]. Here, n denotes the size of the universe and the formulas can use built-in relations $+$ and $\times$. Our proofs often assume that $\log n$ is a natural number, but they can be easily adapted to the general case.

3 Dynamic Complexity

We briefly repeat the essentials of dynamic complexity, closely following [13, 3].

The goal of a dynamic program is to answer a given query on an *input database* subjected to changes that insert or delete single tuples. The program may use an auxiliary data structure represented by an *auxiliary database* over the same domain. Initially, both input and auxiliary database are empty; and the domain is fixed during each run of the program.

A dynamic program has a set of update rules that specify how auxiliary relations are updated after a change of the input database. An *update rule* for updating an auxiliary relation T is basically a formula φ. As an example, if $\varphi(\vec{x}, \vec{y})$ is the update rule for auxiliary relation T under insertions into input relation R, then the new version of T after insertion of a tuple $\bar{a}$ to R is $T \stackrel{\text{def}}{=} \{\vec{b} \mid (\mathcal{I}, Aux) \models \varphi(\vec{a}, \vec{b})\}$ where $\mathcal{I}$ and Aux are the current input and auxiliary databases. For a state $\mathcal{S} = (\mathcal{I}, Aux)$ of the dynamic program with input database $\mathcal{I}$ and auxiliary database Aux we denote the state of the program after applying the change sequence α by $\mathcal{P}_\alpha(\mathcal{S})$. The dynamic program *maintains* a k-ary query $\mathcal{Q}$ if, for each non-empty sequence α of changes and each empty input structure $\mathcal{I}_\emptyset$, relation Q in $\mathcal{P}_\alpha(\mathcal{S}_\emptyset)$ and $\mathcal{Q}(\alpha(\mathcal{I}_\emptyset))$ coincide. Here, $\mathcal{S}_\emptyset = (\mathcal{I}_\emptyset, Aux_\emptyset)$, where $Aux_\emptyset$ denotes the empty auxiliary structure over the domain of $\mathcal{I}_\emptyset$, and $\alpha(\mathcal{I}_\emptyset)$ is the input database after applying α.

In this paper, we are particularly interested in maintaining queries for structures of bounded treewidth. There are several ways to adjust the dynamic setting to restricted classes $\mathcal{C}$ of structures. Here, we simply disallow change sequences that construct structures outside $\mathcal{C}$. That is, in the above definition, only change sequences α are considered, for which each prefix transforms an initially empty structure into a structure from $\mathcal{C}$. We say that a program maintains $\mathcal{Q}$ for a class $\mathcal{C}$ of structures, if Q contains its result after each change sequence α such that the application of each prefix of α to $\mathcal{I}_\emptyset$ yields a structure from $\mathcal{C}$.

The class of queries that can be maintained by a dynamic program is called DynFO. Programs for queries in $\mathsf{DynFO}(+,\times)$ have three particular auxiliary relations that are initialised as a linear order and the corresponding addition and multiplication relations.

We say that a query $\mathcal{Q}$ is in DynFO for a class $\mathcal{C}$ of structures, if there is a dynamic program that maintains $\mathcal{Q}$ for $\mathcal{C}$.

The *active domain* $\text{adom}(\mathcal{A})$ of a structure $\mathcal{A}$ contains all elements used in some tuple of $\mathcal{A}$. A query $\mathcal{Q}$ is *almost domain-independent* if there is a $c \in \mathbb{N}$ such that $\mathcal{Q}(\mathcal{A}) \upharpoonright (\text{adom}(\mathcal{A}) \cup B) = \mathcal{Q}(\mathcal{A} \upharpoonright (\text{adom}(\mathcal{A}) \cup B))$ for all structures $\mathcal{A}$ and sets $B \subseteq A \setminus \text{adom}(\mathcal{A})$ with $|B| \geq c$. The following proposition adapts Proposition 7 from [3].

▶ **Proposition 2.** *If a query $\mathcal{Q} \in \mathsf{DynFO}(+,\times)$ is almost domain-independent, then also $\mathcal{Q} \in$ DynFO.*

4 Algorithmic Technique

There are alternative definitions of DynFO, where the initial structure is non-empty and the initial auxiliary relations can be computed within some complexity [11, 16]. However, in a practical scenario of dynamic query answering it is conceivable that the quality of the auxiliary relations decreases over time and that they are therefore recomputed from scratch at times. We formalise this notion by a relaxed definition of maintainability in which the initial structure is non-empty, the dynamic program is allowed to apply some preprocessing, and query answers need only be given for a certain number of change steps.

We call a query $\mathcal{Q}$ *$(\mathcal{C}, f)$-maintainable*, for some complexity class[2] $\mathcal{C}$ and some function $f : \mathbb{N} \to \mathbb{R}$, if there is a dynamic program $\mathcal{P}$ and a $\mathcal{C}$-algorithm $\mathcal{A}$ such that for each input database $\mathcal{I}$ over a domain of size n, each linear order $\leq$ on the domain, and each change sequence α of length $|\alpha| \leq f(n)$, the relation Q in $\mathcal{P}_\alpha(\mathcal{S})$ and $\mathcal{Q}(\alpha(\mathcal{I}))$ coincide where $\mathcal{S} = (\mathcal{I}, \mathcal{A}(\mathcal{I}, \leq))$.

Although we feel that $(\mathcal{C}, f)$-maintainability deserves further investigation, in this paper we exclusively use it as a tool to prove that queries are actually maintainable in DynFO. To this end, we show next that every $(\text{AC}^1, \log n)$-maintainable query is actually in DynFO and prove later that the queries in which we are interested are $(\text{AC}^1, \log n)$-maintainable.

▶ **Theorem 3.** *Every $(\text{AC}^1, \log n)$-maintainable, almost domain-independent query is in* DynFO*.*

Proof Sketch (of Theorem 3. Assume that a dynamic program $\mathcal{P}$ witnesses that an almost domain-independent query $\mathcal{Q}$ is $(\text{AC}^1, \log n)$-maintainable. Thanks to Proposition 2 it suffices to construct a dynamic program $\mathcal{P}'$ that witnesses $\mathcal{Q} \in \mathsf{DynFO}(+,\times)$. We restrict ourselves to graphs, for simplicity.

The overall idea is to use a simulation technique similar to the ones used in [2] and [12]. We consider each application of one change as a *time step*. We 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 uses $\frac{1}{2}\log n$ steps to compute the auxiliary relations for G_t (using $\text{AC}^1 = \text{IND}[\log n]$) and then another $\frac{1}{2}\log n$ steps to apply the $\log n$ changes of time steps $t+1, \ldots, t+\log n$ (two at a time). After these

[2] Strictly speaking $\mathcal{C}$ should be a complexity class of functions. In this paper, the implied class of functions will always be clear from the stated class of decision problems.

$\log n$ steps the thread is ready to answer query $\mathcal{Q}$ about $G_{t+\log n}$ at time step $t + \log n$. Since one such thread starts at every time point, the program can answer query $\mathcal{Q}$, for each time point $\geq \log n$.

We next give details on the two phases and describe how to deal with earlier time points.

For the first phase, we make use of the equality $\mathrm{AC}^1 = \mathrm{IND}[\log n]$. Let ψ be an inductive formula that is applied $d \log n$ times, for some d, to get the auxiliary relations for a given graph G and the given order $\leq$. The program $\mathcal{P}'$ simply applies ψ to G_t for $2d$ times during each time step, and thus the fixpoint of ψ is reached after $\frac{1}{2}\log n$ steps. The change operations that occur during these steps are not applied to G_t directly but rather stored in some additional relation.

During the second phase the $\frac{1}{2}\log n$ stored change operations and the $\frac{1}{2}\log n$ change operations that happen during the next $\frac{1}{2}\log n$ steps are applied to the state after phase 1. 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 $\mathcal{Q}$ for $\log n$ changes, at the end of phase 2, at time point $t + \log n$, $\mathcal{P}'$ can give the correct query answer for $\mathcal{Q}$ about $G_{t+\log n}$.

To enable $\mathcal{P}'$ to answer $\mathcal{Q}$ also for time steps $t < \log n$, it proceeds as follows. It starts a new thread at time $\frac{t}{2}$ with a graph with at most $\frac{t}{2}$ edges and applies ψ relative to a domain D_t of size $2t + c$, where c is the constant from (almost) domain-independence. The first phase of this thread lasts from time points $\frac{t}{2} + 1$ to $\frac{3t}{4}$, and applies ψ for $4d$ times during each step. As a fixpoint is reached after $\frac{\log(2t+c)}{4} < \frac{t}{4}$ steps, the auxiliary relations are initialised properly (very small t can be handled separately). From time $\frac{3t}{4} + 1$ to time t the changes are applied, again two at a time and the thread is ready to answer $\mathcal{Q}$ at time point t. As at time t at most $2t$ elements are used by edges, the almost domain-independence of $\mathcal{Q}$ guarantees that the result computed by the thread relative to D_t coincides with the D_t-restriction of the query result for G_t. The full query result for G_t, possibly including tuples with elements from $V \setminus D_t$, is obtained as follows: 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 $D_t \setminus \mathrm{adom}(G_t)$ by elements from $V \setminus D_t$ (under consideration of equality constraints among these elements).

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. We can safely assume that $n \geq \log n$ and since at each time point at most $\log n$ threads are active, we can number them in a round robin fashion with numbers $1, \dots, n$. The arity of all auxiliary relations is incremented by one and the additional dimension is used to indicate the number of the thread to which a tuple belongs. ◀

5 Warm-up: 3-Colourability

In this section, we show that the 3-colourability problem 3Col for graphs of bounded treewidth can be maintained in DynFO. Given an undirected graph, 3Col asks whether its vertices can be coloured with three colours such that adjacent vertices have different colours.

▶ **Theorem 4.** *For every k, 3Col is in DynFO for graphs with treewidth at most k.*

The remainder of this section is dedicated to a proof sketch for this theorem. Thanks to Theorem 3 and the fact that 3Col is almost domain-independent, it suffices to show that 3Col is $(\mathrm{AC}^1, \log n)$-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 $T = (I, F, r)$ of width at most $4k + 5$ and maximum bag size $\ell \stackrel{\mathrm{def}}{=} 4k + 6$, as well as information about the 3-colourability of induced subgraphs of G. More precisely, it computes, for each triangle δ of T and each 3-colouring C of the nodes of $B(\delta)$, whether there exists a colouring C' of the inner vertices of $G(\delta)$, such that all edges involving at least one inner vertex are consistent with $C \cup C'$.

During the following $\log n$ change operations, the dynamic program does not need to do much. It only maintains a set S of *special* bags: for each *affected* graph node v that participates in any 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. It will be guaranteed that there are at most $4 \log n$ special bags. With the auxiliary information, a first-order formula φ can test whether G is 3-colourable as follows. By existentially quantifying 8ℓ variables, 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 and all that the formula φ needs to do is checking that this colouring of the special bags can be extended to a colouring of G. This can be done with the help of the auxiliary relations computed during the initialisation which provide all necessary information about subgraphs induced by triangles consisting of special bags.

6 MSO and GSO Queries

In this section, we show that for each k and each MSO-sentence φ the model checking problem for φ on structures of treewidth at most k is in DynFO.

After some definitions regarding MSO types, we will state a Feferman–Vaught-type composition theorem for the composition of at most $\mathcal{O}(\log n)$ many structures that meet in a set C of at most $\mathcal{O}(\log n)$ elements. We will show that if the structure is suitably extended by information about the types of the (disjoint) structures outside C, then MSO formulas can be replaced by first-order formulas. This part is formulated for arbitrary relational structures instead of graphs since we think it might be useful in other contexts as well.

Afterwards, we will use the Feferman–Vaught-type composition theorem to show the maintainability of MSO properties on structures of bounded treewidth. Finally, we explain how these results can be lifted to guarded second-order logic.

6.1 MSO-types

MSO-logic is the extension of first-order logic, which allows existential and universal quantification over set variables $X, X_1, \dots$. The depth of a MSO formula is the maximum nesting depth of (second-order and first-order) quantifiers in the syntax tree of the formula. For a signature Σ and a natural number $d \geq 0$, the *depth-d MSO-type* of a Σ-structure $\mathcal{A}$ is defined as the set of all MSO-sentences φ over Σ of quantifier depth at most d, for which $\mathcal{A} \models \varphi$ holds.

We also need to deal with situations, where we have to take a variable assignment and some additional elements of the structure into account, and therefore the general notion of types is slightly more involved. Let $\mathcal{A}$ be a Σ-structure and $\bar{v} = (v_1, \dots, v_m)$ a tuple of elements from $\mathcal{A}$. We write $(\mathcal{A}, \bar{v})$ for the structure over $\Sigma \cup \{c_1, \dots, c_m\}$ which interprets c_i as v_i, for every $i \in \{1, \dots, m\}$. For a set $\mathcal{Y}$ of first-order and second-order variables and an assignment α for the variables of $\mathcal{Y}$, the depth-d MSO-type of $(\mathcal{A}, \bar{v}, \alpha)$ is the set of MSO-formulas with free variables from $\mathcal{Y}$ of depth d that hold in $(\mathcal{A}, \bar{v}, \alpha)$.

We summarise some basic properties of types in the following. Unless not otherwise stated, *type* always refers to MSO-type. For any $d' < d$ the depth-d' type of a structure results from its depth-d type by simply removing all formulas of depth larger than d'.

For every depth-d type, there is a depth-d MSO formula α_τ that is true in exactly the structures and for those assignments of type τ.

Each depth-d type τ induces a set of depth-$(d-1)$ types over $\mathcal{Y} \cup \{x\}$ (assuming $x \notin \mathcal{Y}$) that can be realised in a structure of type τ, represented by the set of all formulas $\alpha_{\tau'}$ for depth-$(d-1)$ types τ' with free variables set $\mathcal{Y} \cup \{x\}$, for which $\exists x \alpha_{\tau'}$ is in τ. We call a type τ', for which $\exists x\, \alpha_{\tau'}$ is in τ, an x-realisation of τ. Likewise, a depth-$(d-1)$ type τ' is an X-realisation for a depth-d type τ, if $\exists X\, \alpha_{\tau'}$ is in τ. For more background on MSO-logic, types, and the above properties readers might consult, e.g., [10].

6.2 A Feferman–Vaught-type composition theorem

In the following, we give an adaptation of the Feferman–Vaught-type composition theorem from [6] that will be useful for maintaining MSO properties.

Intuitively, the idea is very easy, but the formal presentation will come with some technical complications. For simplicity, we explain the basic idea for graphs first.

In a nutshell, we consider graphs $G = (V, E)$ with a *center* $C \subseteq V$, such that the graph $G[V - C]$ is a disjoint union of components $D_1 - C, \dots, D_\ell - C$, such that, for some $w > 0$,

- $|D_i \cap C| \leq w$, for every i,
- all edges in E have both end nodes in C or in some D_i, and
- 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 $(C, D_1, \dots, D_\ell, v_1, \dots, v_\ell)$ is a *weak partition* of G with *center* C, and *connection width* w. We refer to the sets $D_1, \dots, D_\ell$ as *petals* and the nodes $v_1, \dots, v_\ell$ as *identifiers* of their respective petals. We emphasise that ℓ is not assumed to be bounded by any constant, only by $|C|$.

Readers who have read the proof sketch for Theorem 4 can think of C as the set of vertices from special bags (plus one inner vertex per clean triangle as identifier).

Our goal is to show that, if a graph G with a weak partition of logarithmic center size is extended by the information about the MSO types of its petals in a suitable way, resulting in a structure G', then MSO formulas over G have equivalent first-order formulas over G'.

In a first step, we show that, if (the center) of G is suitably extended by the information about the MSO types of its petals, then every MSO formula has an equivalent MSO formula whose quantification is restricted to C.[3] In a second step we show that, if in a MSO formula quantification is restricted to some node set C of logarithmic size then there is an equivalent (unrestricted) first-order formula. For the second step we assume that the graph has an additional relation that encodes subsets of C by bounded-size tuples over V.

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

The definition of weak partitions easily carries over to general Σ-structures. In particular, tuples need to be entirely in C or in some petal D_i. For every i, we call the set $I_i \stackrel{\text{def}}{=} D_i \cap C$ the *interface* of D_i and the nodes of a petal D_i that are *not* in C *inner elements* of D_i.

Let $\mathcal{A}$ be a Σ-structure, $P = (C, D_1, \dots, D_\ell, v_1, \dots, v_\ell)$ a weak partition of connection width w, and $d > 0$. For every i, let $\bar{u}^i = (u^i_1, \dots, u^i_w)$ be a tuple of elements from the interface of D_i such that $u^i_1 = v_i$ and every node from I_i occurs in $\bar{u}^i$. By $\mathcal{A}_i$ 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,

[3] As remarked by a reviewer, Proposition 5 below can probably be concluded from Shelah's Composition Theorem for generalised sums [14].

i.e., $\mathcal{A}_i$ only contains tuples with at least one inner element of D_i. The *depth d, width w MSO indicator structure* of $\mathcal{A}$ relative to P and tuples $\bar{u}^i$ is the unique structure $\mathcal{B}$ such that:

- $\mathcal{B}$ is an expansion of $\mathcal{A}$ (with the same universe and the same Σ-relations),
- $\mathcal{B}$ has an additional w-ary relation J that contains all tuples $\bar{u}^i$, and
- $\mathcal{B}$ has additional unary relation symbols R_τ, one for every depth-d MSO-type over $\Sigma \cup \{c_1, \dots, c_w\}$, and for each such τ, R_τ contains the identifier nodes v_i, for all i, for which the depth-d MSO-type of $(\mathcal{A}_i, \bar{u}^i)$ is τ.

The set of all indicator structures of $\mathcal{A}$ relative to P for varying tuples $\bar{u}^i$ is denoted by $\mathcal{S}(\mathcal{A}, P, w, d)$.

We call a 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 $\forall X\ (\forall x(X(x) \to C(x)) \to \varphi)$.

▶ **Proposition 5.** *For each $d > 0$, every MSO sentence φ with depth d, and each w, there is a C-restricted MSO sentence ψ such that for every Σ-structure $\mathcal{A}$ with a weak partition P of connection width w and every $\mathcal{B} \in \mathcal{S}(\mathcal{A}, P, w, d)$ it holds $\mathcal{A} \models \varphi$ if and only if $\mathcal{B} \models \psi$.*

Proof. The construction of ψ and the proof of its correctness is by induction on the structure of φ. It can be found in the full version of the paper [4]. ◀

To formalise the second step, we need some further notation. 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 6.** *For each C-restricted MSO-sentence ψ over a signature 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 (in $\mathcal{A}$), 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 $\text{Sub}(\bar{x}, y)$. And likewise for universal set quantification. ◀

By combining Propositions 5 and 6 we immediately get the following result.

▶ **Theorem 7.** *For each $d > 0$, every MSO sentence φ with depth d, and each w, there is a first-order sentence χ such that, for every Σ-structure $\mathcal{A}$ with a weak partition $P = (C, D_1, \dots, D_\ell, v_1, \dots, v_\ell)$ of connection width w, every $\mathcal{B} \in \mathcal{S}(\mathcal{A}, P, w, d)$ and a relation Sub that encodes subsets of C, it holds $\mathcal{A} \models \varphi$ if and only if $(\mathcal{B}, Sub) \models \chi$.*

6.3 MSO on structures of bounded treewidth

In this subsection we prove a dynamic version of Courcelle's Theorem: all MSO properties can be maintained in DynFO for graphs with bounded treewidth. More precisely, for a given MSO sentence φ we consider the model checking problem MC_φ that asks whether a given graph G satisfies φ, that is, whether $G \models \varphi$ holds.

▶ **Theorem 8.** *For every* MSO *sentence φ and every k,* MC_φ *is in* DynFO *for graphs with treewidth at most k.*

Thanks to Theorem 3 it suffices to show that MC_φ is $(\mathrm{AC}^1, \log n)$-maintainable for graphs G with treewidth at most k. We note that it is easy to see MSO-definable queries are almost domain-independent and the respective constant c depends only on the formula φ. The reason is that MSO-formulas can not make use of more than a constant number of isolated nodes.[4] The dynamic program that will be constructed in the proof works very similarly to the one of Theorem 4: during its initialisation it constructs a tree decomposition and appropriate MSO-types for all triangles. During the change sequence, a set C of special nodes is used that contains, for each affected graph node v, at least one bag containing v. The union of all bags represented by the set C induces a weak partition P and the dynamic program basically maintains an MSO indicator structure for G relative to P. Since there are only $\mathcal{O}(\log n)$ many change steps, $|C| = \mathcal{O}(\log n)$ and therefore Theorem 7 yields that from this auxiliary data it can be inferred in a first-order fashion whether $G \models \varphi$.

Proof (of Theorem 8). Thanks to Theorem 3 and the fact that MSO queries are almost domain-independent it suffices to show that MC_φ is $(\mathrm{AC}^1, \log n)$-maintainable in DynFO for graphs with treewidth at most k. Let d be the quantifier depth of φ.

Given a graph $G = (V, E)$, the AC^1 initialisation first computes a nice tree decomposition $T = (I, F, r)$ with bags of size at most $\ell \stackrel{\text{def}}{=} 4k + 6$, together with $\preceq$. With each node 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. The AC^1-initialisation also ensures that arithmetic relations $+$, $\times$ and BIT are available.

The dynamic program further uses additional auxiliary relations S, N, and D_τ, for each depth-d MSO-type τ over the signature that consists of the binary relation symbol E and $3\ell + 1$ constant symbols $c_1, \ldots, c_{3\ell+1}$. From these relations all ingredients needed to apply Theorem 7 can be first-order defined: a weak partition P with center C, an MSO indicator structure, and a relation Sub that encodes subsets of C.

The relation S stores tuples representing special bags, as in the proof of Theorem 4. The relations D_τ provide MSO type information for all triangles. More precisely, for 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 $(v(\delta), \bar{v}(i_0), \bar{v}(i_1), \bar{v}(i_2))$ if and only if the MSO depth-d type of $(G(\delta), v(\delta), \bar{v}(i_0), \bar{v}(i_1), \bar{v}(i_2))$ is τ, where $v(\delta)$ denotes the smallest inner node of $G(\delta)$ with respect to $\leq$.

The set C always contains all graph nodes that occur in special bags (and thus in S), plus one inner node $v(\delta)$, for each maximal[5] clean triangle with at least one inner node. Relation N maintains a bijection between C and an initial segment of $\leq$. From the auxiliary relations, the relations used for Theorem 7 can be defined as follows.

The relations S and C are used to define a weak partition as follows. Clean triangles with at least two inner nodes become petals of the weak partition. Thus the interface $I(\delta)$ of a petal corresponding to a 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 $v(\delta)$. Now, an indicator structure $\mathcal{B} \in \mathcal{S}(\mathcal{A}, P, w, d)$ can be first-order defined as follows. Clearly, clean triangles can be easily first-order defined from the relation S. For each clean triangle $\delta = (i_0, i_1, i_2)$ with at least two inner nodes, the relation J contains a tuple $(\delta(v), \bar{v}(i_0), \bar{v}(i_1), \bar{v}(i_2))$, and the relation R_τ contains $\delta(v)$ if and only if is $(\delta(v), \bar{v}(i_0), \bar{v}(i_1), \bar{v}(i_2)) \in D_\tau$. For defining Sub, we observe that C is of size $b \log n$ for some $b \in \mathbb{N}$. Thus a subset C' of C can be represented by a tuple $(a_1, \ldots, a_b)$ of nodes,

[4] It should be mentioned that structures with a linear order $\leq$ do not have any isolated nodes.
[5] Maximal basically means that all its corner nodes are special.

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. It is easy to see that the relations can be first-order defined from S, N and D_τ.

How the auxiliary relations are initialised and updated is detailed in the full version. ◀

6.4 Extension to GSO logic

We finally sketch how the results of this section can be extended to guarded second-order logic (GSO). In a nutshell, GSO extends MSO by *guarded second-order quantification*. Thus, it syntactically allows to quantify over non-unary relation variables. However, this quantification is semantically restricted: a tuple $\bar{t} = (a_1, \dots, a_m)$ can only occur in a quantified relation, if all elements from $\{a_1, \dots, a_m\}$ occur together in some tuple of the structure, in which the formula is evaluated.

To state the analogue of Proposition 5 for GSO, two definitions need to be modified: GSO indicator structures store information about the respective GSO types instead of MSO types. C-restricted formulas can use GSO-quantifiers only to quantify relations over C, e.g., formulas need to be restricted as in $\exists X\ (\forall \bar{x}(X(x_1, \dots, x_m) \rightarrow (C(x_1) \wedge \dots \wedge C(x_m))) \wedge \varphi)$. In the statement of Proposition 5 MSO can simply be replaced by GSO. The proof hardly changes. Of course, there is an additional case for GSO quantification but the types of petals can still be handled by MSO quantification. For Proposition 6, encoding of subsets has to be extended to encoding of subrelations. For the quantification of m-ary relations this encoding has to be done by a $(k+m)$-ary relation, for some k. Such an encoding only exists, if the number of tuples over C in $\mathcal{A}$ is only logarithmic. Analogously, Theorem 7 can be extended.

7 MSO Optimisation Problems

With the techniques presented in the previous section also 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 $\varphi(X)$ with a free set variable X. Given a graph G with vertex set V, it asks for a set $A \subseteq V$ of minimal[6] size such that $G \models \varphi(A)$.

From the point of view of dynamic programs, such an optimisation problem is just a unary query, that is, the result is defined by some formula $\psi(x)$ with a free first-order variable x.

▶ **Theorem 9.** *For every* MSO *formula* $\varphi(X)$ *and every* k*,* OPT_φ *is in* DynFO *for graphs with treewidth at most* k*.*

A dynamic program for an optimisation problem OPT_φ can be constructed by a modification of a program for the decision problem for the MSO sentence $\exists X\ \varphi$, as constructed in the proof of Theorem 8. Basically, we enrich the type information for each petal by information about the smallest set with which a given (X-realisation) type τ can be obtained.

Proof Sketch (of Theorem 9). We describe how a dynamic program for model checking $\psi \stackrel{\text{def}}{=} \exists X\ \varphi$, where $\varphi(X)$ is an MSO-formula of quantifier depth d, can be adapted to a program for OPT_φ. We reuse the notation from the proof of Theorem 8.

Most auxiliary relations remain as in the program of that proof. However, instead of relations D_τ for depth-$(d+1)$-types τ the program uses relations $\#D_{\tau'}$ and $S_{\tau'}$ for

[6] The adaptation to maximisation problems is straightforward.

X-realisations τ' of such types with the following intention. Let δ be a triangle. If τ' is a depth-d type over $\{E, X, c_1, \dots, c_{3\ell+1}\}$ that can be realised by some set A, then for the minimal size s of such a set A, $\#D_{\tau'}$ shall contain a tuple $(v(\delta), \bar{v}(i_0), \bar{v}(i_1), \bar{v}(i_2), v_s)$, similarly as in the proof of Theorem 8, but with v_s chosen as the $(s+1)$-th element[7] with respect to $\leq$. Furthermore, for the lexicographically minimal set A of this kind and size s, $S_{\tau'}$ shall contain all tuples $(v(\delta), \bar{v}(i_0), \bar{v}(i_1), \bar{v}(i_2), v)$, where $v \in A$.

The proof of Theorem 8 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 of a triangle δ might be achievable by a finite number of combinations of types of its sub-triangles. Here, the overall sizes of the underlying sets for X need to be computed and the minimal solution needs to be picked. This is possible by a FO(+,×)-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 8. Since D_τ needs no updates there, neither $\#D_\tau$ nor S_τ do, here.

It remains to explain how the actual query result can be defined. A close inspection of the proofs of Propositions 5 and 6 reveals that the first-order formula χ equivalent to $\exists X \varphi$, as guaranteed by Theorem 7, is of the form $\exists \bar{x} \exists (\bar{x}_\tau)_\tau \hat{\chi}$, where, in a nutshell, $\bar{x}$ represents all center nodes from C in X and $\bar{x}_\tau$ selects all petals $G(\delta_j)$ that have depth-d MSO type τ. The formula χ can first be adapted such that it defines the set of all nodes v selected in C or occurring in a tuple $(v(\delta), \bar{v}(i_0), \bar{v}(i_1), \bar{v}(i_2), v)$ of $S_{\tau'}$ for the type τ' that was chosen for the petal associated with δ. Next, using the ability of FO(+,×) to add up logarithmically many numbers [15, Theorem 1.21], the size of the thus represented set can be determined using $\#D_{\tau'}$. Then, a similar formula can check that there is no (lexicographically) smaller set that makes φ true. We emphasise that the resulting formula so far can become true only for one assignment α for $\bar{x}$ and $\bar{x}_\tau$ and thus a final formula with free variable x can be constructed which becomes true for an assignment α', if and only if $\alpha'(x)$ occurs in the set encoded by this assignment α. ◀

From the proof sketch it is easy to see that a dynamic program can also maintain the *size* s of an optimal solution, either implicitly as $|Q|$ for a distinguished relation Q or as $\{v_s\}$. Also, with the adjustments sketched in subsection 6.4, the result can be extended to GSO definable optimisation problems. We can conclude the following corollary from the results of this and the previous section.

▶ **Corollary 10.** *For every k, the Boolean queries* 3-Colourability *and* HamiltonCycle *and the optimisation problems* VertexCover*,* DominatingSet *and* ShortestPath *are in* DynFO *for graphs of treewidth at most k.*

8 Conclusion

In this paper, we introduced a strategy for maintaining queries by periodically restarting its computation from scratch and limiting the number of change steps that have to be taken into account. This has been captured in the notion of $(\mathcal{C}, f)$-maintainable queries, and we proved that all $(\mathrm{AC}^1, \log n)$-maintainable, almost domain-independent queries are actually in DynFO. As a consequence, decision and optimisation queries definable in MSO- and

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

GSO-logic are in DynFO for graphs of bounded treewidth. For this, we stated a Feferman-Vaught-type composition theorem for these logics, which might be interesting in its own right. Though we phrase our results for MSO and GSO for graphs only, their proofs translate swiftly to general relational structures.

We believe that our strategy will find further applications. For instance, it is conceivable that interesting queries on planar graphs, such as the shortest-path query, can be maintained for a bounded number of changes using auxiliary data computed by an AC^1 algorithm (in particular since many important data structures for planar graphs can be constructed in logarithmic space and therefore in AC^1).

References

1. Patricia Bouyer, Vincent Jugé, and Nicolas Markey. Dynamic complexity of parity games with bounded tree-width. *CoRR*, abs/1610.00571, 2016. URL: `http://arxiv.org/abs/1610.00571`.
2. Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in DynFO. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming – 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 2015. `doi:10.1007/978-3-662-47666-6_13`.
3. Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in DynFO. *CoRR*, abs/1502.07467, 2015. URL: `http://arxiv.org/abs/1502.07467`.
4. Samir Datta, Anish Mukherjee, Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. A strategy for dynamic programs: Start over and muddle through. *CoRR*, abs/1704.07998, 2017. URL: `http://arxiv.org/abs/1704.07998`.
5. Guozhu Dong, Jianwen Su, and Rodney W. Topor. Nonrecursive incremental evaluation of datalog queries. *Ann. Math. Artif. Intell.*, 14(2-4):187–223, 1995. `doi:10.1007/BF01530820`.
6. Michael Elberfeld, Martin Grohe, and Till Tantau. Where first-order and monadic second-order logic coincide. *ACM Trans. Comput. Log.*, 17(4):25:1–25:18, 2016. `doi:10.1145/2946799`.
7. Michael Elberfeld, Andreas Jakoby, and Till Tantau. 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*, pages 143–152. IEEE Computer Society, 2010. `doi:10.1109/FOCS.2010.21`.
8. Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. The dynamic complexity of formal languages. *ACM Trans. Comput. Log.*, 13(3):19, 2012. `doi:10.1145/2287718.2287719`.
9. Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999.
10. Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.
11. Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. *J. Comput. Syst. Sci.*, 55(2):199–209, 1997. `doi:10.1006/jcss.1997.1520`.
12. Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. Dynamic complexity under definable changes. In *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*, pages 19:1–19:18, 2017. `doi:10.4230/LIPIcs.ICDT.2017.19`.
13. Thomas Schwentick and Thomas Zeume. Dynamic complexity: recent updates. *SIGLOG News*, 3(2):30–52, 2016. `doi:10.1145/2948896.2948899`.

14 Saharon Shelah. The monadic theory of order. *Annals of Mathematics*, pages 379–419, 1975. doi:10.2307/1971037.

15 Heribert Vollmer. *Introduction to Circuit Complexity – A Uniform Approach.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999.

16 Volker Weber and Thomas Schwentick. Dynamic complexity theory revisited. *Theory Comput. Syst.*, 40(4):355–377, 2007. doi:10.1007/s00224-006-1312-0.