Universität Stuttgart

Fakultät Informatik, Elektrotechnik und Informationstechnik

Institut für Formale
Methoden der Informatik

Universitätsstraße 38
D–70569 Stuttgart

Model-Checking Hierarchical Structures

Markus Lohrey

Report Nr. 2005/01

January 5, 2005

CR: F.1.3, F.4.1

Abstract

Hierarchical graph definitions allow a modular description of graphs using modules for the specification of repeated substructures. Beside this modularity, hierarchical graph definitions also allow to specify graphs of exponential size using polynomial size descriptions. In many cases, this succinctness increases the computational complexity of decision problems when input graphs are defined hierarchical. In this paper, the model-checking problem for first-order logic (FO), monadic second-order logic (MSO), and second-order logic (SO) on hierarchically defined input graphs is investigated. It is shown that in general these model-checking problems are exponentially harder than their non-hierarchical counterparts, where the input graphs are given explicitly. As a consequence, several new complete problems for the levels of the polynomial time hierarchy and the exponential time hierarchy are obtained. Based on classical results of Gaifman and Courcelle, two restrictions on the structure of hierarchical graph definitions that lead to more efficient model-checking algorithms are presented.

1 Introduction

Hierarchical graph definitions specify a graph via modules, where every module is a graph that may refer to modules of a smaller hierarchical level. In this way, large structures can be represented in a modular and succinct way. Hierarchical graph definitions were introduced in [29] in the context of VLSI design. Formally, hierarchical graph definitions can be seen as hyperedge replacement graph grammars [12, 22] that generate precisely one graph.

In this paper we consider the complexity of the model-checking problem for first-order logic (FO), monadic second-order logic (MSO), and second-order logic (SO) on hierarchically defined input graphs. FO allows only quantification over elements of the universe, MSO allows quantification over subsets (unary predicates) of the universe, and SO allows quantification over relations of arbitrary arity over the universe. The model-checking problem for some fixed logic (e.g. FO or MSO) asks whether a given sentence from that logic is true in a given finite structure (e.g. a graph). Usually, the structure is given explicitly, for instance by listing all tuples in each of the relations of the structure. In this paper, the input structure will be given in a compressed form via a hierarchical graph definition.

Each of the logics FO, MSO, and SO has many fascinating connections to other parts of computer science, e.g., automata theory, complexity theory, database theory, verification, etc. The interested reader is referred to the text books [11, 25, 30, 42] and the handbook article [44] for more details. It is therefore not surprising that the model-checking problem for these logics on explicitly given input structures is a very well-studied problem with many deep results. Let us just mention a few references: [13, 16, 17, 20, 21, 31, 33, 45, 46]. But whereas several papers study the complexity of specific algorithmic problems on hierarchically defined input graphs, like for instance reachability, planarity, circuit-value, and 3-colorability [27, 28, 29, 34, 35, 36], there is no systematic investigation of model-checking problems for hierarchically defined graphs so far (one should notice that all the algorithmic problems mentioned above can be formulated in SO). The only exception is the work from [1, 5] on hierarchical state machines, where the complexity of temporal logics (LTL, CTL, CTL*) over hierarchical state machines is investigated. Hierarchical state machines can be seen as a restricted form of hierarchical graph definitions that are tailored towards the modular specification of large reactive systems. We think that the investigation of model-checking problems for "general purpose logics" like FO and MSO over hierarchically defined graphs leads to a better understanding of hierarchical structures in a broad sense.

Our investigation of model-checking problems for hierarchically defined graphs will follow a methodology introduced by Vardi [45]. For a given logic $\mathcal{L}$ and a class of structures $\mathcal{C}$, Vardi introduced three different ways of measuring the complexity of the model-checking problem for $\mathcal{L}$ and $\mathcal{C}$: (i) One may consider a fixed sentence φ from the logic $\mathcal{L}$ and consider the complexity of verifying for a given structure $A \in \mathcal{C}$ whether $A \models \varphi$; thus, only the structure belongs to the input (data complexity or structure complexity). (ii) One may fix a structure A from the class $\mathcal{C}$ and consider the complexity of verifying for a given sentence φ from $\mathcal{L}$, whether $A \models \varphi$; thus, only the formula belongs to the input (expression complexity). (iii) Finally, both the structure and the formula may belong to the input (combined complexity). In the context of

hierarchically defined graphs, expression complexity will not lead to new results. Having a fixed hierarchically defined graph makes no difference to having a fixed explicitly given graph. Thus, we will only consider data and combined complexity for hierarchically defined graphs.

After introducing the necessary concepts in Section 3–7, we study model-checking problems for FO over hierarchically defined graphs in Section 8. Section 8.1 deals with data complexity whereas in Section 8.2, combined complexity is briefly considered. Section 9 carries out the same program for MSO and SO. In all cases, we measure the complexity of the model-checking problem in dependence on the structure of the quantifier prefix of the input formula. In some cases we observe an exponential jump in computational complexity when moving from explicitly to hierarchically defined input graphs. In other cases there is no complexity jump at all. We also consider structural restrictions of hierarchical graph definitions that lead to more efficient model-checking algorithms. Our results are collected in Table 1 and Table 2 at the end of the paper, see Section 4 and Section 5 for the relevant definitions.

2 Related work

Specific algorithmic problems (e.g. reachability, planarity, circuit-value, 3-colorability) on hierarchically defined graphs are studied in [27, 28, 29, 34, 35, 36]. A concept related to hierarchically defined graphs are hierarchical state machines [1, 5], which are a widely used concept for the modular and compact system specification in model-checking. Hierarchical state machines can be seen as a restricted form of hierarchically defined graphs. The papers [1, 5] study the complexity of model-checking temporal logics (LTL, CTL, CTL*) over hierarchical state machines. Other formalisms for the succinct description of structures, which were studied under a complexity theoretical perspective, are boolean circuits [6, 19, 38, 49], boolean formulas [21, 47], and binary decision diagrams [15, 48]. For these formalisms, general upgrading theorems can be shown, which roughly state that if a problem is complete for a complexity class C, then the compressed variant of this problem is complete for the exponentially harder version of C. For hierarchical graph definitions such an upgrading theorem fails [28].

3 General notations

The reflexive and transitive closure of a binary relation $\rightarrow$ is $\xrightarrow{*}$. Let $\equiv$ be an equivalence relation on a set A. Then, for $a \in A$, $[a]_\equiv = \{b \in A \mid a \equiv b\}$ denotes the equivalence class containing a. With $[A]_\equiv$ we denote the set of all equivalence classes. With $\pi_\equiv : A \rightarrow [A]_\equiv$ we denote the function with $\pi_\equiv(a) = [a]_\equiv$ for all $a \in A$. For sets A, A_1, and A_2 we write $A = A_1 \uplus A_2$ if $A = A_1 \cup A_2$ and $A_1 \cap A_2 = \emptyset$. For a function $f : A \rightarrow B$ let $\mathrm{dom}(f) = A$ and $\mathrm{ran}(f) = \{b \in B \mid \exists a \in A : f(a) = b\}$. For $C \subseteq A$ we define the restriction $f\restriction_C : C \rightarrow B$ by $f\restriction_C(c) = f(c)$ for all $c \in C$. For functions $f : A \rightarrow B$ and $g : B \rightarrow C$ we define the composition $g \circ f : A \rightarrow C$ by $(g \circ f)(a) = g(f(a))$ for all $a \in A$. For functions $f : A \rightarrow C$ and $g : B \rightarrow D$ with $A \cap B = \emptyset$ we define the function $f \cup g : A \cup B \rightarrow C \cup D$ by $(f \cup g)(a) = f(a)$ for $a \in A$

and $(f \cup g)(b) = g(b)$ for $b \in B$.

A ranked alphabet is a pair (Γ, rank), where Γ is a finite alphabet and $\mathrm{rank} : \Gamma \to \mathbb{N} = \{0, 1, 2, \ldots\}$ assigns to every $a \in \Gamma$ its rank. If the rank-function is clear from the context, we will omit it. An *ordered dag* (directed acyclic graph) is a triple $G = (V_G, \gamma_G, \mathrm{root}_G)$ where (i) V_G is a finite set of nodes, (ii) $\gamma_G : V_G \to V_G^*$ is the child-function, (iii) the relation $E_G := \{(u, v) \mid u, v \in V_G, v \text{ occurs in } \gamma_G(u)\}$ is acyclic, and (iv) root_G has indegree 0 in the graph (V_G, E_G). The size of G is $|G| = |V_G|$. The notion of a *root-path* $p \in \mathbb{N}^*$ in G together with its target-node $\tau_G(p) \in V_G$ are inductively defined as follows: (i) ε is a root-path in G and $\tau_G(\varepsilon) = \mathrm{root}_G$ and (ii) if p is a root-path in G, $v = \tau_G(p)$, and $n = |\gamma_G(v)|$, then pi is a root-path for all $1 \leq i \leq n$ and $\tau_G(pi)$ is the i-th node in the list $\gamma_G(v)$.

4 Complexity theory

We assume that the reader has some basic background in complexity theory [37]. In particular, we assume that the reader is familiar with the classes **L** (deterministic logarithmic space), **NL** (nondeterministic logarithmic space), and **P** (deterministic polynomial time). It is well known that each of these classes is closed under (deterministic) logspace reductions. A function f is *computable in nondeterministic logspace* [2] if

- for all x, $|f(x)| \leq |x|^c$ for some constant c and
- there exists a Turing machine M for which the working space is bounded by $\mathcal{O}(\log(n))$ and such that for every input x: on every computation path, either M rejects on that path or produces $f(x)$ on the output tape.

Of course the space on the output tape does not belong to the working space. We say that a language A is **NL**-reducible to a language B, if there exists a function f such that (i) f is computable in nondeterministic logspace and (ii) for all x, $x \in A$ if and only if $f(x) \in B$. It is not hard to see that if A is **NL**-reducible to $B \in \mathbf{NL}$, then also $A \in \mathbf{NL}$. One can use the same proof that shows that **L** is closed under (deterministic) logspace reductions: For an input x, one simulates an **NL**-machine for B on the input $f(x)$, but without actually producing $f(x)$. Each time, the machine for B needs the i-th bit of $f(x)$, then one starts a simulation of the machine that calculates f in nondeterministic logspace until the i-th bit of $f(x)$ is produced; if the machine for f rejects, then the overall simulation rejects.

Several times we will use alternating Turing-machines, see [7] for more details. Roughly speaking, an *alternating Turing-machine* M is a nondeterministic Turing-machine, where the set of states Q is partitioned into three sets: $Q_\exists$ (existential states), $Q_\forall$ (universal states), and F (accepting states). A configuration C with current state q is accepting, if

- $q \in F$, or
- $q \in Q_\exists$ and there exists a successor configuration of C that is accepting, or

- $q \in Q_\forall$ and every successor configuration of C is accepting.

An input word w is accepted by M if the corresponding initial configuration is accepting. An alternation on a computation path of M is a transition from a universal state to an existential state or vice versa.

By [23, 43], the class of all problems, that can be solved on an alternating Turing-machine in logarithmic space, where furthermore the number of alternations is bounded by some fixed constant, is still equal to $\mathbf{NL}$.

The levels of the *polynomial time hierarchy* are defined as follows: Let $k \geq 1$. Then $\mathbf{\Sigma_k^p}$ (resp. $\mathbf{\Pi_k^p}$) is the set of all problems that can be recognized on an alternating Turing-machine within $k-1$ alternations and polynomial time, where furthermore the initial state is assumed to be in $Q_\exists$ (resp. $Q_\forall$). The polynomial time hierarchy is $\mathbf{PH} = \bigcup_{k\geq 1} \mathbf{\Sigma_k^p}$. If we replace in these definitions the polynomial time bound by an exponential time bound (i.e., $2^{n^{\mathcal{O}(1)}}$), then we obtain the levels $\mathbf{\Sigma_k^e}$ (resp. $\mathbf{\Pi_k^e}$) of the (weak) *EXP time hierarchy* $\mathbf{EH} = \bigcup_{k\geq 1} \mathbf{\Sigma_k^e}$. If we replace the polynomial time bound by a logarithmic time bound $\mathcal{O}(\log(n))$, then we obtain the levels $\mathbf{\Sigma_k^{log}}$ (resp. $\mathbf{\Pi_k^{log}}$) of the *logtime hierarchy* $\mathbf{LH} = \bigcup_{k\geq 1} \mathbf{\Sigma_k^{log}}$, which is contained in $\mathbf{L}$. Here one assumes that the basic Turing-machine model is enhanced with a random access mechanism in form of a query tape that contains a binary coded position of the input tape. If the machine enters a distinguished query state, then the machine has random access to the input position that is addressed by the query tape. The logtime hierarchy is a uniform version of the circuit complexity class $\mathbf{AC}^0$.

5 Hierarchical graph definitions

Let Γ be a ranked alphabet. A Γ-labeled *hypergraph* is a tuple $H = (V, E, \lambda)$, where V is a finite set of nodes, E is a finite set of *hyperedges*, and $\lambda : E \to \{(A, \tau) \mid A \in \Gamma, \tau : \{1, \ldots, \mathrm{rank}(A)\} \to V\}$ is the labeling function. We also write $V^H = V$, $E^H = E$, and $\lambda^H = \lambda$. If $\lambda(e) = (A, \tau)$, then we say that e is an A-labeled hyperedge. Assume that $\equiv$ is an equivalence relation on the set of nodes V. Then we define the quotient hypergraph $H/_\equiv = ([V]_\equiv, E, \mu)$, where for all $e \in E$, $\mu(e) = (A, \pi_\equiv \circ \tau)$ if $\lambda(e) = (A, \tau)$. For a hyperedge $e \in E$ we define the hypergraph $H \setminus e = (V, E \setminus \{e\}, \lambda\restriction_{E\setminus\{e\}})$. We say that two hypergraphs $H_1 = (V_1, E_1, \lambda_1)$ and $H_2 = (V_2, E_2, \lambda_2)$ are disjoint if $V_1 \cap V_2 = E_1 \cap E_2 = \emptyset$. In this case, we define the hypergraph $H_1 \oplus H_2 = (V_1 \cup V_2, E_1 \cup E_2, \lambda_1 \cup \lambda_2)$. For $n \geq 0$, an *n-pointed hypergraph* is a pair $G = (H, \sigma)$, where H is a hypergraph and $\sigma : \{1, \ldots, n\} \to V^H$ is an *injective* mapping. The nodes $\sigma(i)$ $(1 \leq i \leq n)$ are also called the *pin nodes* of G. Nodes in $V^H \setminus \mathrm{ran}(\sigma)$ are called *internal nodes* of G. For $A \in \Gamma$ with $\mathrm{rank}(A) = n$ we define the n-pointed hypergraph $G_A = ((\{1, \ldots, n\}, \{e\}, \lambda), \mathrm{id})$ with $\lambda(e) = (A, \mathrm{id})$, where id is the identity function on $\{1, \ldots, n\}$.

Definition 5.1. *A* hierarchical graph definition *is a tuple $D = (\Gamma, N, S, P)$ such that the following holds:*

(1) $\Gamma \cup N$ *is a ranked alphabet,* N *is the set of nonterminals,* Γ *is the set of terminals,*

(2) $S \in N$ *is the start nonterminal, where* $\mathrm{rank}(S) = 0$*, and*

(3) P *is a set of productions. For every* $A \in N$*,* P *contains exactly one production* $A \to G$*, where* $G = (H, \sigma)$ *is a* $\mathrm{rank}(A)$*-pointed* $(\Gamma \cup N)$*-labeled hypergraph. We require that if* $\lambda^H(e) = (A, \tau)$ *with* $A \in N$*, then* τ *is injective.*

(4) Define the relation E_D *on* N *as follows:* $(A, B) \in E_D$ *if and only if for the unique production of the form* $A \to G$*,* G *contains a* B*-labeled hyperedge. Then we require that* E_D *is acyclic.*

By (4), the transitive closure $\succ_D$ *of the relation* E_D *is a partial order, we call it the* hierarchical order*. The size* $|D|$ *of* D *is defined by* $\sum_{(A \to (G,\tau)) \in P} |V^G| + |E^G|$*.*

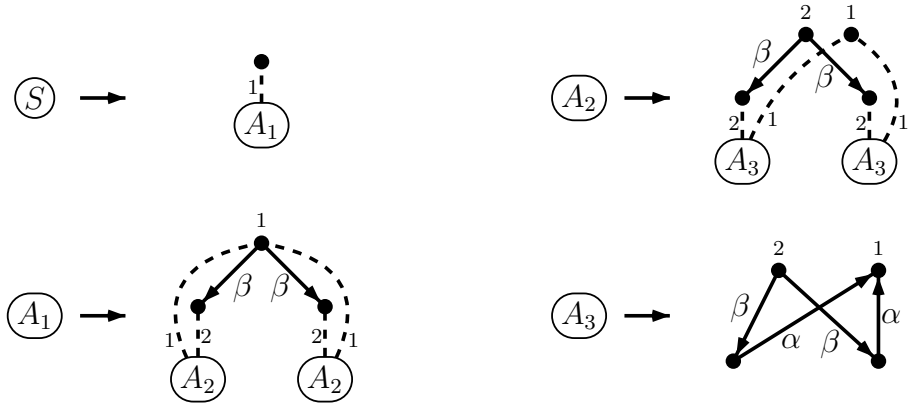
Let us fix a *hierarchical graph definition* $D = (\Gamma, N, S, P)$. For $i \in \{1, 2\}$ let $G_i = (H_i, \sigma_i)$ be an n-pointed $(\Gamma \cup N)$-labeled hypergraph for some $n \geq 0$ (thus σ_i is injective). Then we write $G_1 \Rightarrow_D G_2$ if and only if there exists a hyperedge $e \in E^{H_1}$ such that:

- $\lambda^{H_1}(e) = (A, \tau)$ with $A \in N$,
- $A \to (H, \sigma)$ is the unique production with left-hand side A (thus, also σ is injective),
- w.l.o.g. H and H_1 are disjoint,
- $H_2 = (H_1 \setminus e \oplus H)/_{\equiv}$ with $\equiv$ the smallest equivalence relation on $V^{H_1} \uplus V^H$ that contains all pairs of the form $(\tau(i), \sigma(i))$ for $1 \leq i \leq \mathrm{rank}(A)$, and
- $\sigma_2 = \pi_{\equiv} \circ \sigma_1$.

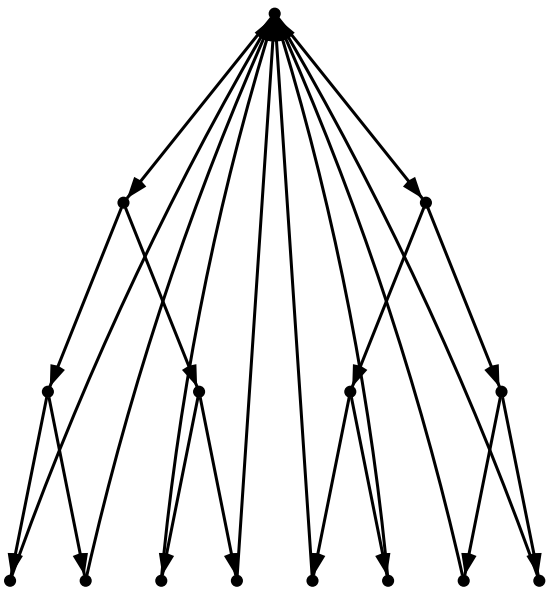
Note that the injectivity of σ_2 follows from the injectivity of σ and σ_1. It is easy to see that for every $A \in N$, there exists a unique Γ-labeled $\mathrm{rank}(A)$-pointed hypergraph $\mathrm{eval}_D(A)$ such that $G_A \Rightarrow^*_D \mathrm{eval}_D(A)$. Finally, we define $\mathrm{eval}(D) = \mathrm{eval}_D(S)$.

We assume the following conventions for the graphical representation of hypergraphs and productions of hierarchical graph definitions: A hyperedge e with $\lambda(e) = (A, \tau)$ for a nonterminal A is drawn as a big circle with inner label A. This circle is connected via dashed lines with the nodes $\tau(i)$ for $1 \leq i \leq \mathrm{rank}(A)$, where the connection to $\tau(i)$ is labeled with i. These dashed lines are also called *tentacles*. Only terminals of rank 1 or rank 2 will occur in diagrams and lower bound proofs. A terminal hyperedge e with $\lambda(e) = (f, \tau)$ and $\mathrm{rank}(f) = 2$ is drawn as a solid directed edge from $\tau(1)$ to $\tau(2)$ with label f. A terminal hyperedge e with $\lambda(e) = (a, \tau)$ and $\mathrm{rank}(a) = 1$ is represented by just labeling the node $\tau(1)$ with a. Our definition allows multiple edges with the same label as well as several node labels for a single node. If $G = (H, \sigma)$ is an n-pointed hypergraph, i.e., $\sigma : \{1, \ldots, n\} \to V^H$ is an injective mapping, then we label the pin node $\sigma(i)$ with i. In order to distinguish this label i better from node labels that correspond to terminals of rank 1, we will use a smaller font for the label i.

Example 5.2. *Let us consider the hierarchical graph definition* $D = (\Gamma, N, S, P)$*, where* Γ *contains two terminals* α *and* β *of rank* 2*,* $N = \{S, A_1, A_2, A_3\}$*, where* $\mathrm{rank}(S) = 0$*,* $\mathrm{rank}(A_1) = 1$*, and* $\mathrm{rank}(A_2) = \mathrm{rank}(A_3) = 2$*. The set* P *of productions contains the following rules:*



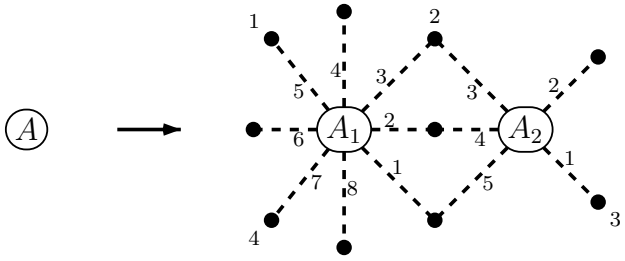
Then $\mathrm{eval}(D)$ *is the following graph. Edge labels are omitted; edges going down in the tree have to be labeled with* β*, and the other edges going from the leafs to the root have to be labeled with* α*.*



Definition 5.3. *A hierarchical graph definition* $D = (\Gamma, N, S, P)$ *is in* Chomsky normal form *if for every production* $A \to (H, \tau)$ *in* P*, either*

- $E^H = \{e_1, e_2\}$*, where* $\lambda^H(e_i) = (A_i, \tau_i)$*,* $A_i \in N$ *(*$i \in \{1, 2\}$*), and* $V^H = \mathrm{ran}(\tau_1) \cup \mathrm{ran}(\tau_2)$*, or*
- H *does not contain any hyperedge that is labeled with a nonterminal.*

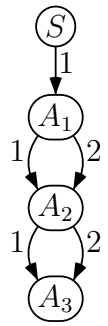
A typical production of the first type looks as follows, where $\mathrm{rank}(A) = 4$:



It is straight-forward to construct for a given hierarchical graph definition D another hierarchical graph definition D' in Chomsky normal form such that $\mathrm{eval}(D) = \mathrm{eval}(D')$. Moreover, this construction can be carried out by a logarithmic space bounded machine.

Definition 5.4. *With a hierarchical graph definition $D = (\Gamma, N, S, P)$ we associate an ordered dag $\mathrm{dag}(D) = (N, \gamma, S)$, where the child-function γ is defined as follows: Let $A \to G$ be the unique production with left-hand side $A \in N$ and let $e_1, \ldots, e_n$ be an enumeration of all hyperedges in G that are labeled with a nonterminal (this enumeration is somehow given by the input encoding of D). Then $\gamma(A) = A_1 \cdots A_n$, where A_i is the label of the hyperedge e_i.*

For instance, $\mathrm{dag}(D)$ for the hierarchical graph definition from Example 5.2 looks as follows:



Remark 5.5. *We list some simple algorithmic properties of hierarchical graph definitions that are useful for the further considerations.*

1. *A node of* $\mathrm{eval}(D)$ *can be uniquely represented by a pair (p, v) such that (i) p is a root-path in $\mathrm{dag}(D)$ with target node $A = \tau_{\mathrm{dag}(D)}(p)$ and (ii) $A \to (H, \tau)$ is the unique production with left-hand side A, where $v \in V^H \setminus \mathrm{ran}(\tau)$ is an internal node.*[1] *This representation is of size $\mathcal{O}(|D|)$ and given a pair (p, v) we can check in time $\mathcal{O}(|D|)$ (or alternatively in space $\mathcal{O}(\log(|D|))$, whether (p, v) represents a node of* $\mathrm{eval}(D)$.

2. *Given nodes $u_i = (p_i, v_i)$ for $1 \le i \le \mathrm{rank}(a)$, where a is a terminal label, we can verify in time $\mathcal{O}(|D|)$ (or alternatively in space $\mathcal{O}(\log(|D|))$), whether $H = \mathrm{eval}(D)$ contains a hyperedge e with $\lambda^H(e) = (a, \tau)$ and $\tau(i) = u_i$ for $1 \le i \le \mathrm{rank}(a)$.*

[1] The nodes in $\mathrm{ran}(\tau)$, i.e., the pin nodes of the right-hand side of A, are excluded here, because they were already generated by some larger (with respect to the hierarchical order $\succ_D$) nonterminal.

The following simple statement will be useful later:

Lemma 5.6. *For a given hierarchical graph definition D and a node $u = (p, v)$ of* $\mathrm{eval}(D)$*, we can construct in deterministic logarithmic space (and hence in polynomial time) a new hierarchical graph definition D' such that* $\mathrm{eval}(D)$ *and* $\mathrm{eval}(D')$ *are identical, except that in* $\mathrm{eval}(D')$ *the node u has the additional label α, where α is a node label that does not occur anywhere else in D.*

Proof. Assume that $p = i_1 i_2 \cdots i_n$ ($i_k \in \mathbb{N}$) and let $A_k = \tau_{\mathrm{dag}(D)}(i_1 i_2 \cdots i_k) \in N$ be the target node of the path $i_1 i_2 \cdots i_k$ for $k \in \{0, \ldots, n\}$. Thus, $A_0 = S$ (the start nonterminal). For every nonterminal A_i introduce a copy A'_i. Let $A_k \to G_k$ be the unique production for A_k in D. If $0 \leq k < n$, then we introduce for A'_k the production $A'_k \to G'_k$, where G'_k results from G_k by labeling the i_{k+1}-th nonterminal-hyperedge with A'_{k+1} (instead of A_{k+1} as in G_k). Finally, we add the rule $A'_n \to G'_n$, where G'_n results from G_n by adding the new label α to the node v of G_n. The resulting hierarchical graph definition D' has the property from the lemma. Clearly, the construction can be done using logarithmic working space. $\square$

6 Graph operations

Hierarchically defined graphs can be also described via graph operations on n-pointed hypergraphs in the sense of Courcelle [10]. Since this approach will be useful in Section 8.1 and 9.1, we briefly introduce this formalism in this section. Given an n_i-pointed hypergraph $G_i = (H_i, \sigma_i)$ ($i \in \{1, 2\}$) with H_1 and H_2 disjoint we define the disjoint union $G_1 \oplus G_2 = (H_1 \oplus H_2, \sigma)$, where $\sigma : \{1, \ldots, n_1 + n_2\} \to V^{H_1 \oplus H_2}$ is defined by $\sigma(i) = \sigma_1(i)$ for $1 \leq i \leq n_1$ and $\sigma(i) = \sigma_2(i - n_1)$ for $n_1 + 1 \leq i \leq n_1 + n_2$. For $1 \leq a < b \leq n$ and an n-pointed hypergraph $G = (H, \sigma)$ we define the $(n-1)$-pointed hypergraph $\mathrm{glue}_{a,b}(G) = (H/_{\equiv}, \sigma')$, where $\equiv$ is the smallest equivalence relation on V^H that contains the pair $(\sigma(a), \sigma(b))$ and $\sigma'(i) = \sigma(i)$ for $1 \leq i \leq b - 1$ and $\sigma'(i) = \sigma(i+1)$ for $b \leq i \leq n-1$. Finally, for an injective mapping $f : \{1, \ldots, m\} \to \{1, \ldots, n\}$ we define the m-pointed hypergraph $\mathrm{rename}_f(G) = (H, \sigma \circ f)$. A *graph straight-line program* $\mathcal{S} = (X_i := t_i)_{1 \leq i \leq n}$ over the ranked alphabet Γ is a sequence of definitions of the form $X_i := t_i$, where t_i is either an n-pointed Γ-labeled hypergraph (for some arbitrary n), or an expression of the form $X_j \oplus X_k$, $\mathrm{glue}_{a,b}(X_j)$, or $\mathrm{rename}_f(X_j)$, where $j, k < i$ and a, b, f are as above. Here, the X_i are formal variables. For every variable X_i we define inductively its type $\mathrm{type}(X_i)$ as follows: (i) if t_i is an n-pointed hypergraph, then $\mathrm{type}(X_i) = n$, (ii) if $t_i = X_j \oplus X_k$, then $\mathrm{type}(X_i) = \mathrm{type}(X_j) + \mathrm{type}(X_k)$, (iii) if $t_i = \mathrm{glue}_{a,b}(X_j)$, then $\mathrm{type}(X_i) = \mathrm{type}(X_j) - 1$, and (iv) if $t_i = \mathrm{rename}_f(X_j)$, then $\mathrm{type}(X_i) = |\mathrm{dom}(f)|$. Now we can define inductively the $\mathrm{type}(X_i)$-pointed Γ-labeled hypergraph $\mathrm{eval}(X_i)$ as follows: (i) if t_i is an n-pointed hypergraph G, then $\mathrm{eval}(X_i) = G$, (ii) if $t_i = X_j \oplus X_k$, then $\mathrm{eval}(X_i) = \mathrm{eval}(X_j) \oplus \mathrm{eval}(X_k)$, (iii) if $t_i = \mathrm{glue}_{a,b}(X_j)$, then $\mathrm{eval}(X_i) = \mathrm{glue}_{a,b}(\mathrm{eval}(X_i))$, and (iv) if $t_i = \mathrm{rename}_f(X_j)$, then $\mathrm{eval}(X_i) = \mathrm{rename}_f(\mathrm{eval}(X_j))$. Finally, we define $\mathrm{eval}(\mathcal{S}) = \mathrm{eval}(X_n)$. It is straight-forward to construct (in logarithmic space) from a given hierarchical graph definition D a graph straight-line program $\mathcal{S}$ such that $\mathrm{eval}(D) = \mathrm{eval}(\mathcal{S})$, see for instance [9].

7 Logic

A *relational signature* S is a finite set consisting of relational symbols r_i ($i \in I$) and constant symbols c_j ($j \in J$). Each relation symbol r_i has an associated arity α_i. A *structure over the signature* S is a tuple $\mathcal{A} = (A, (R_i)_{i \in I}, (a_j)_{j \in J})$, where $R_i \subseteq A^{\alpha_i}$ is the relation associated with the relation symbol r_i and $a_j \in A$ is the constant associated with the constant symbol c_j. As usual, a constant a may be replaced by the unary relation $\{a\}$, thus it suffices to consider relational signatures without constant symbols. But for convenience, we will consider signatures with constant symbols several times.

We identify a ranked alphabet Γ with the relational signature, where every $a \in \Gamma$ is a relation symbol of arity $\mathrm{rank}(a)$. Thus, a Γ-labeled hypergraph $H = (V, E, \lambda)$ is identified with the relational structure $(V, (R_a)_{a \in \Gamma})$, where $R_a = \{(v_1, \ldots, v_{\mathrm{rank}(a)}) \in V^{\mathrm{rank}(a)} \mid \exists e \in E : \lambda(e) = (a, \tau), v_i = \tau(i) \text{ for } 1 \leq i \leq \mathrm{rank}(a)\}$. An n-pointed Γ-labeled hypergraph $G = (H, \sigma)$ is identified with the same structure, where every pin node $\sigma(i)$ ($1 \leq i \leq n$) is added as an additional constant. The i-th pin node is denoted by the new constant symbol $\mathrm{pin}(i)$. Let us fix a ranked alphabet (i.e., a signature) Γ for the further discussion.

In this paper, we consider the logics FO (first-order logic), MSO (monadic second-order logic), and SO (second-order logic) over hierarchically defined hypergraphs. A detailed introduction into mathematical logic can be found in [11]. Atomic FO formulas over the signature Γ are of the form $x = y$, $x = \mathrm{pin}(i)$, and $a(x_1, \ldots, x_n)$, where $a \in \Gamma$ with $\mathrm{rank}(a) = n$, and $x, y, x_1, \ldots, x_n$ are first-order variables ranging over nodes. The interpretation of $a(x_1, \ldots, x_n)$ is $(x_1, \ldots, x_n) \in R_a$. In case $\mathrm{rank}(a) = 2$ we also write $x_1 \xrightarrow{a} x_2$, in case $\mathrm{rank}(a) = 1$ we also write $x_1 \in a$, i.e, we identify the node label a with the set of all a-labeled nodes. From these atomic subformulas we construct arbitrary FO formulas over the signature Γ using boolean connectives and (first-order) quantifications over nodes. A Σ_k-FO formula (resp. Π_k-FO formula) is a first-order formula of the form $B_1 B_2 \cdots B_k : \varphi$, where: (i) φ is a quantifier-free FO formula, (ii) for i odd, B_i is a block of existential (resp. universal) quantifiers, whereas (iii) for i even, B_i is a block of universal (resp. existential) quantifiers. An FO^k-formula ($k \geq 2$) is a first-order formula that only uses at most k different (bounded or free) variables.

SO extends FO by allowing the quantification over relations of arbitrary arity. For this, there exists for every $m \geq 1$ a set of second-order variables of arity m that range over m-ary relations over the universe. In addition to the atomic formulas of FO, SO allows atomic formulas of the form $(x_1, \ldots, x_m) \in X$, where X is an m-ary second-order variable and $x_1, \ldots, x_m$ are first-order variables. MSO is the fragment of SO (and the extension of FO) that only allows to use second-order variables of arity 1, i.e., quantification over subsets of the universe is allowed. A Σ_k-SO formula (resp. Π_k-SO formula) is an SO formula of the form $B_1 B_2 \cdots B_k : \varphi$, where: (i) φ is an SO formula that contains only first-order quantifiers, (ii) for i odd, B_i is a block of existential (resp. universal) SO quantifiers, whereas (iii) for i even, B_i is a block of universal (resp. existential) SO quantifiers. For an SO sentence φ, i.e., an SO formula without free variables, and a relational structure $\mathcal{A}$, we write $\mathcal{A} \models \varphi$ if the sentence φ is true in the structure $\mathcal{A}$.

The quantifier rank $\mathrm{qr}(\varphi)$ of an MSO formula (we won't need this notion for general SO

formulas) is inductively defined as follows: $\mathrm{qr}(\varphi) = 0$ if φ is atomic, $\mathrm{qr}(\neg\varphi) = \mathrm{qr}(\varphi)$, $\mathrm{qr}(\varphi \wedge \psi) = \mathrm{qr}(\varphi \vee \psi) = \max\{\mathrm{qr}(\varphi), \mathrm{qr}(\psi)\}$, and $\mathrm{qr}(\forall x\varphi) = \mathrm{qr}(\exists x\varphi) = \mathrm{qr}(\varphi) + 1$, where x is an FO or an MSO variable. It is well-known that for every $k \geq 1$, there are only finitely many pairwise nonequivalent formulas of quantifier rank at most k. This value only depends on k, see [26] for an explicit estimation. The k-FO theory (resp. k-MSO theory) of a structure $\mathcal{A}$, briefly k-FOTh($\mathcal{A}$) (resp. k-MSOTh($\mathcal{A}$)), consists of all FO sentences (resp. MSO sentences) of quantifier rank at most k that are true in $\mathcal{A}$; by the previous remark it is a finite set up to logical equivalence.

In Section 8.1 we will briefly consider *modal logic*, see e.g. [40] for more details. Modal logic is interpreted over directed graphs, where both edges and nodes are labeled. Let $G = (V, (E_\alpha)_{\alpha\in\Sigma}, (P_\gamma)_{\gamma\in\Gamma})$ be such a graph, where V is the set of nodes, $E_\alpha \subseteq V \times V$ is the set of all α-labeled edges, and $P_\gamma \subseteq V$ is the set of all γ-labeled nodes. Such a structure can be seen as a hypergraph, where all terminals have rank 1 or 2. Atomic formulas of modal logic are γ, where $\gamma \in \Gamma$ is a node label, *tt* (for true), and *ff* (for false). If φ and ψ are already formulas of modal logic, then also $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $[\alpha]\varphi$, and $\langle\alpha\rangle\varphi$ are formulas of modal logic, where $\alpha \in \Sigma$ is an edge label. The satisfaction relation $G, v \models \varphi$ (the modal logic formula φ is satisfied in the node $v \in V$ of G) is inductively defined as follows ($\alpha \in \Sigma$, $\gamma \in \Gamma$):

$$
\begin{array}{lcl}
G, v \models \mathrm{tt} & & \\
G, v \not\models \mathrm{ff} & & \\
G, v \models \gamma & \Leftrightarrow & v \in P_\gamma \\
G, v \models \neg\varphi & \Leftrightarrow & G, v \not\models \varphi \\
G, v \models \varphi \wedge \psi & \Leftrightarrow & G, v \models \varphi \text{ and } G, v \models \psi \\
G, v \models \varphi \vee \psi & \Leftrightarrow & G, v \models \varphi \text{ or } G, v \models \psi \\
G, v \models [\alpha]\varphi & \Leftrightarrow & G, u \models \varphi \text{ for every } u \in V \text{ with } (v, u) \in E_\alpha \\
G, v \models \langle\alpha\rangle\varphi & \Leftrightarrow & G, u \models \varphi \text{ for some } u \in V \text{ with } (v, u) \in E_\alpha
\end{array}
$$

It is well-known and easy to see that for every formula φ of modal logic we can construct an FO^2 formula $\varphi'(x)$ with one free variable such that for every node $v \in V$: $G, v \models \varphi$ if and only if $G \models \varphi'(v)$, see e.g. [30, Prop. 14.8].

7.1 Model-checking explicitly given input graphs

Let us briefly recall the known results concerning the complexity of the model-checking problem for the logics from the previous section when input graphs are represented explicitly. For Σ_k-FO (resp. Π_k-FO) the data complexity is $\mathbf{\Sigma_k^{log}}$ (resp. $\mathbf{\Pi_k^{log}}$) [4, 24], whereas the combined complexity goes up to $\mathbf{\Sigma_k^p}$ (resp. $\mathbf{\Pi_k^p}$) [13, 41]. For Σ_k-MSO (resp. Π_k-MSO), both the data and combined complexity is $\mathbf{\Sigma_k^p}$ (resp. $\mathbf{\Pi_k^p}$) [13, 33, 41]. For full second-order logic, the data complexity of Σ_k-SO is still $\mathbf{\Sigma_k^p}$ [13, 41], whereas the combined complexity becomes $\mathbf{\Sigma_k^e}$. For modal logic, the combined complexity is $\mathbf{P}$, in fact, for every fixed $\ell \geq 2$, the combined complexity of FO^ℓ is $\mathbf{P}$ as well [46].

8 FO over hierarchically defined graphs

In this section we study the model-checking problem for FO over hierarchically defined input graphs. Section 8.1 deals with data complexity. First, we prove that the data complexity of Σ_1-FO for hierarchically defined input graphs is **NL** (Theorem 8.2). Using this result, we show that for Σ_k-FO (resp. Π_k-FO) with $k > 1$ the data complexity becomes $\mathbf{\Sigma^p_{k-1}}$ (resp. $\mathbf{\Pi^p_{k-1}}$) (Theorem 8.3). Next, we study structural restrictions on hierarchical graph definitions that lead to more efficient model-checking algorithms. We introduce the apex restriction, which means that tentacles in a right-hand side are not allowed to access the pin nodes. We prove that under the apex restriction the data complexity of FO goes down to **NL** (Theorem 8.5). Finally, we consider hierarchical graph definitions, for which the rank of every nonterminal as well as the number of nonterminal hyperedges in a right-hand side is bounded by some fixed constant c (c-boundedness). We show that under this restriction the data complexity reduces to **P** (Theorem 8.6), but we cannot provide a matching lower bound.

In Section 8.2 we briefly consider combined complexity. We argue that the combined complexity for Σ_k-FO (resp. Π_k-FO) does not change when moving from explicitly to hierarchically defined input graphs (namely $\mathbf{\Sigma^p_k}$ resp. $\mathbf{\Pi^p_k}$) (Theorem 8.7).

8.1 Data complexity

A trivial lower bound for model-checking a fixed FO sentence over hierarchically defined input graphs is given by the following statement:

Proposition 8.1. *It is* **NL***-hard to verify for a given hierarchical graph definition D whether* $\mathrm{eval}(D)$ *is the empty graph. Thus, given D, it is* **NL***-hard to verify whether* $\mathrm{eval}(D) \models \exists x : x = x$*. Moreover, for the hierarchical graph definition D we can assume that the rank of every nonterminal is* 0 *and that every right-hand side of a production contains at most two nonterminal hyperedges.*

Proof. We prove the proposition by a reduction from the **NL**-complete graph accessibility problem for directed acyclic graphs [39]. Thus, let $G = (V, E)$ be a directed acyclic graph and let $u, v \in V$, where w.l.o.g. v has outdegree 0 and every node $a \in V$ has at most 2 direct successor nodes. For every node $a \in V$ we introduce a nonterminal A_a of rank 0; the start nonterminal is A_u. If a has the direct successor nodes $a_1, \ldots, a_k$ with $k \in \{1, 2\}$, then the right-hand side of A_a consists of k hyperedges labeled with $A_{a_1}, \ldots, A_{a_k}$ (and no nodes). If $a \in V \setminus \{v\}$ is a node of outdegree 0, then the right-hand side of A_a is the empty hypergraph. Finally, the right-hand side of A_v consists of a single node. Then, $(u, v) \in V$ if and only if the resulting hierarchical graph definition generates a non-empty graph. □

For Σ_1-FO we can also prove a matching **NL** upper bound:

Theorem 8.2. *For every fixed Σ_1-FO or Π_1-FO formula $\varphi(y_1, \ldots, y_m)$, the following problem is in* **NL** *(and hence in* **P***):*

INPUT: A hierarchical graph definition D and nodes $u_1, \ldots, u_m$ from $\mathrm{eval}(D)$ *(encoded as described in Remark 5.5).*

QUESTION: $\mathrm{eval}(D) \models \varphi(u_1, \ldots, u_m)$*?*

Proof. Due to the closure of **NL** under complement (see e.g. [37]), it suffices to prove the theorem for a Σ_1-FO formula. Let $D = (\Gamma, N, S, P)$. In a first step, take new node labels $\alpha_1, \ldots, \alpha_m$ and use Lemma 5.6 in order to construct in logarithmic space a new hierarchical graph definition D' such that $\mathrm{eval}(D')$ is identical to $\mathrm{eval}(D)$ except that in $\mathrm{eval}(D')$ the node u_i has the additional node label α_i. Then $\mathrm{eval}(D) \models \varphi(u_1, \ldots, u_m)$ if and only if $\mathrm{eval}(D') \models \exists x_1 \cdots \exists x_m : \varphi(x_1, \ldots, x_m) \wedge \bigwedge_{i=1}^m x_i \in \alpha_i$. Note that the latter sentence is a fixed Σ_1-FO sentence. Thus, it suffices to consider a fixed Σ_1-FO sentence of the form $\exists x_1 \cdots \exists x_n : \varphi(x_1, \ldots, x_n)$, where moreover φ is a conjunction of possibly negated atomic formulas (disjunctions can be shifted in front of the existential quantifiers). We may also assume that the input hierarchical graph definition D is in Chomsky normal form, see Definition 5.3.

A subformula ψ of φ is a conjunction of a subset of the conjuncts that occur in φ. With $\mathrm{Var}(\psi)$ we denote the set of those variables from $\{x_1, \ldots, x_n\}$ that occur in ψ. Clearly, there is only a constant number of subformulas. For a nonterminal $A \in N$ of rank m, we denote with $\mathcal{F}(A)$ the set of all formulas that result by replacing in an arbitrary subformula ψ of φ some of the variables from $\mathrm{Var}(\psi)$ by constants from $\{\mathrm{pin}(1), \ldots, \mathrm{pin}(m)\}$. For $\theta \in \mathcal{F}(A)$ we denote with θ^+ (resp. θ^-) the set of all positive atoms (resp. negated atoms) that occur in θ. An *assertion* is a pair (A, θ), where $\theta \in \mathcal{F}(A)$. Note that an assertion can be stored in logarithmic space. Let $\mathrm{eval}_D(A) = (H, \sigma)$. We write $\mathrm{valid}(A, \theta)$ for the assertion (A, θ) if there exists a mapping $\beta : \mathrm{Var}(\theta) \to V^H \setminus \mathrm{ran}(\sigma)$ such that θ becomes true in $\mathrm{eval}_D(A)$ when every variable $x \in \mathrm{Var}(\theta)$ is replaced by $\beta(x)$.

Let us consider an example. Assume that

$$\psi \equiv r_1(x_1, x_2, x_4) \wedge \neg r_2(x_2, x_3) \wedge r_3(x_4, x_3, x_2) \wedge \neg r_1(x_2, x_3, x_4).$$

If $\mathrm{rank}(A) = 3$, then for instance the following formula θ belongs to $\mathcal{F}(A)$:

$$r_1(x_1, \mathrm{pin}(3), \mathrm{pin}(1)) \wedge \neg r_2(\mathrm{pin}(3), x_3) \wedge r_3(\mathrm{pin}(1), x_3, \mathrm{pin}(3)) \wedge \neg r_1(\mathrm{pin}(3), x_3, \mathrm{pin}(1)).$$

We have

$$\begin{aligned} \theta^+ &= \{r_1(x_1, \mathrm{pin}(3), \mathrm{pin}(1)), r_3(\mathrm{pin}(1), x_3, \mathrm{pin}(3))\} \\ \theta^- &= \{\neg r_2(\mathrm{pin}(3), x_3), \neg r_1(\mathrm{pin}(3), x_3, \mathrm{pin}(1))\}. \end{aligned}$$

Claim: We can verify in **NL** whether for a given assertion (A, θ) with $\mathrm{Var}(\theta) = \emptyset$ we have $\mathrm{valid}(A, \theta)$.

Proof of the claim. The formula θ is a conjunction of a constant number of (negated) atoms of the form $(\neg) r(\mathrm{pin}(i_1), \ldots, \mathrm{pin}(i_k))$. Since **NL** is closed under complement, it suffices to verify a single atom $a = r(\mathrm{pin}(i_1), \ldots, \mathrm{pin}(i_k))$ in $\mathrm{eval}_D(A)$. Let $A \to (H, \tau)$ be the unique production

for A. If H only contains terminal hyperedges, then it is trivial to check $\mathrm{valid}(A, a)$ in **NL**. Otherwise, assume that $E^H = \{e_1, e_2\}$, where $\lambda^H(e_i) = (A_i, \tau_i)$ and $A_i \in N$ for $i \in \{1, 2\}$. In this case we nondeterministically choose an $i \in \{1, 2\}$ such that $\{\tau(i_1), \ldots, \tau(i_k)\} \subseteq \mathrm{ran}(\tau_i)$. If such an i does not exist then we reject immediately. Otherwise we proceed with the assertion (A_i, b), where the atom b results from the atom $r(\mathrm{pin}(i_1), \ldots, \mathrm{pin}(i_k))$ by replacing the constant $\mathrm{pin}(i_\ell)$ by $\mathrm{pin}(j)$ if $\tau(i_\ell) = \tau_i(j)$; since τ_i is injective (see (3) in Definition 5.1), j is determined uniquely. The atom b can be calculated in logspace from $r(\mathrm{pin}(i_1), \ldots, \mathrm{pin}(i_k))$. This proves the claim.

Now we present an **NL**-algorithm for verifying general assertions (with variables). The algorithm stores a list $\alpha_1\alpha_2 \cdots \alpha_k$ of assertions where $k \leq |\mathrm{Var}(\varphi)| + 2$. Since $|\mathrm{Var}(\varphi)|$ is a constant and every assertion α_i can be stored in logarithmic space, the algorithm works in logarithmic space as well. If the algorithm transforms the list $\alpha_1\alpha_2 \cdots \alpha_k$ into the list $\alpha'_1\alpha'_2 \cdots \alpha'_\ell$, then

$$\bigwedge_{i=1}^{k} \mathrm{valid}(\alpha_i) \quad \Leftrightarrow \quad \bigwedge_{i=1}^{\ell} \mathrm{valid}(\alpha'_i). \tag{1}$$

Initially, the list only contains the assertion (S, φ). The algorithm accepts, if the list of assertions is empty. Together with (1) this proves the correctness of the algorithm. It remains to describe a single step of the algorithm such that (1) is true.

Case 1. There exists an i such that $\alpha_i = (A, \theta)$ and $\mathrm{Var}(\theta) = \emptyset$. Then by the above claim, we can verify in **NL** whether $\mathrm{valid}(A, \theta)$ is true. If $\mathrm{valid}(A, \theta)$ turns out to be wrong, then we reject immediately, otherwise we continue with the list $\alpha_1 \cdots \alpha_{i-1}\alpha_{i+1} \cdots \alpha_k$. The correctness property (1) is clearly true.

Case 2. There does not exist an i such that $\alpha_i = (A, \theta)$ and $\mathrm{Var}(\theta) = \emptyset$. Then the algorithm removes an arbitrary assertion, say $\alpha_1 = (A, \theta)$, from the list and continues as follows:

Case 2.1. $A \to (H, \tau)$ is the unique production for A, and H does not contain nonterminal hyperedges. Then it is again trivial to check in **NL** whether $\mathrm{valid}(A, a)$ and we can proceed as in Case 1.

Case 2.2. $A \to (H, \tau)$ is the unique production for A, and $E^H = \{e_1, e_2\}$, where $\lambda^H(e_i) = (A_i, \tau_i)$ and $A_i \in N$ for $i \in \{1, 2\}$. Thus, $V^H = \mathrm{ran}(\tau_1) \cup \mathrm{ran}(\tau_2)$. We now guess:

- a partition $\mathrm{Var}(\theta) = Y \uplus X_1 \uplus X_2$ (each of the three sets X_1, X_2, and Y may be empty),
- a mapping $\beta : Y \to V^H \setminus \mathrm{ran}(\tau)$, and
- a partition $\theta^+ = \psi_1^+ \uplus \psi_2^+$ such that for every $i \in \{1, 2\}$, every atom $a \in \psi_i^+$, every constant $\mathrm{pin}(j)$, and every variable $x \in \mathrm{Var}(\theta)$ we have:

$$\begin{aligned} \mathrm{pin}(j) \text{ occurs in } a &\Rightarrow \tau(j) \in \mathrm{ran}(\tau_i) \\ x \text{ occurs in } a &\Rightarrow (x \in X_i \vee (x \in Y \wedge \beta(x) \in \mathrm{ran}(\tau_i))) \end{aligned} \tag{2}$$

Intuitively, Y is the set of all variables from $\mathrm{Var}(\theta)$ that will be assigned (via the mapping β) to a node in $V^H \setminus \mathrm{ran}(\tau) = (\mathrm{ran}(\tau_1) \cup \mathrm{ran}(\tau_2)) \setminus \mathrm{ran}(\tau)$ (which is the set of nodes that are directly generated by A), whereas X_i is the set of all variables that will be assigned to a node that is generated by the nonterminal A_i. The set ψ_i^+ contains all positive atoms a from θ such that the terminal hyperedge that will finally make the atom a true is generated by the nonterminal A_i. If the above data do not exist, then we reject immediately. Otherwise we construct for $i \in \{1, 2\}$ the conjunction $\theta_i \in \mathcal{F}(A_i)$ as follows:

- First define ψ_i as the conjunction of all atoms in

$$\psi_i^+ \cup \{(\neg a) \in \theta^- \mid a \text{ satisfies (2) for all constants } \mathrm{pin}(j) \text{ and all variables } x \in \mathrm{Var}(\theta)\}.$$

- Next, we replace in ψ_i every constant $\mathrm{pin}(j)$ by $\mathrm{pin}(\ell)$, where $\tau(j) = \tau_i(\ell)$, and we replace every variable $x \in Y$ by $\mathrm{pin}(\ell)$, where $\beta(x) = \tau_i(\ell)$. Let θ_i be the resulting conjunction.

We continue with the list $(A_1, \theta_1)(A_2, \theta_2)\alpha_2 \cdots \alpha_k$. Note that $\mathrm{Var}(\theta_i) \subseteq X_i$. To see that (1) is true, we have to argue that $\mathrm{valid}(A, \theta)$ if and only if $\mathrm{valid}(A_1, \theta_1)$ and $\mathrm{valid}(A_2, \theta_2)$. We leave the proof to the reader.

This concludes the description and the correctness proof of the algorithm. The number of assertions in the stored list is bounded by $|\mathrm{Var}(\varphi)| + 2$, because (i) there are at most two assertions (A, θ) with $\mathrm{Var}(\theta) = \emptyset$ in the list, and (ii) if (A_1, θ_1) and (A_2, θ_2) belong to the list, then $\mathrm{Var}(\theta_1) \cap \mathrm{Var}(\theta_2) = \emptyset$. This proves the theorem. □

Theorem 8.3. *For every fixed Σ_{k+1}-FO sentence (resp. Π_{k+1}-FO sentence) ψ, the question, whether* $\mathrm{eval}(D) \models \psi$ *for a given hierarchical graph definition D is in $\mathbf{\Sigma_k^p}$ (resp. $\mathbf{\Pi_k^p}$).*

Moreover, for every $k \geq 1$, there exists a fixed Σ_{k+1}-FO sentence (resp. Π_{k+1}-FO sentence) ψ such that the question, whether $\mathrm{eval}(D) \models \psi$ *for a given hierarchical graph definition D, is $\mathbf{\Sigma_k^p}$-complete (resp. $\mathbf{\Pi_k^p}$-complete). Finally, the sentence ψ is equivalent to an FO^2-sentence.*

Proof. For the upper bound assume that

$$\psi \equiv \exists \overline{x}_1 \cdots \forall \overline{x}_k \exists \overline{x}_{k+1}\ \theta(\overline{x}_1, \ldots, \overline{x}_k, \overline{x}_{k+1})$$

is a fixed Σ_{k+1}-FO formula, where k is assumed to be even (other cases can be dealt analogously) and $\overline{x}_i$ is a tuple of FO variables. Our alternating polynomial time algorithm guesses for every $1 \leq i \leq k$ a tuple $\overline{u}_i$ (of the same length as $\overline{x}_i$) of nodes from $\mathrm{eval}(D)$, using the representation for nodes from Remark 5.5 in Section 5. Since the size of this representation for a node is of polynomial size, this guessing needs polynomial time. Moreover, if i is odd (resp. even) we guess the tuple $\overline{u}_i$ in an existential (resp. universal) state. It remains to verify, whether

$$\mathrm{eval}(D) \models \exists \overline{x}_{k+1} \theta(\overline{u}_1, \ldots, \overline{u}_k, \overline{x}_{k+1}),$$

which is possible in polynomial time by Theorem 8.2.

For the proof of the second statement, note that for every $k \geq 1$ it suffices to prove the statement either for the class $\mathbf{\Sigma_k^p}$ or $\mathbf{\Pi_k^p}$, because these two classes are complementary to each

other, and the negation of a Σ_{k+1}-FO sentence is a Π_{k+1}-FO sentence and vice versa. For k even, we prove the statement for $\mathbf{\Sigma_k^p}$, for k odd, we prove the statement for $\mathbf{\Pi_k^p}$. For k odd, the following problem QSAT_k is $\mathbf{\Pi_k^p}$-complete [41, 50]:

INPUT: A quantified boolean formula Θ of the form

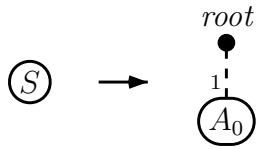
$$\forall x_1 \cdots \forall x_{\ell_1-1} \exists x_{\ell_1} \cdots \exists x_{\ell_2-1} \cdots \forall x_{\ell_{k-1}} \cdots \forall x_n : \varphi(x_1, \ldots, x_n),$$

where $1 < \ell_1 < \ell_2 < \cdots < \ell_{k-1} \leq n$ and φ is a boolean formula in 3-DNF over the variables $x_1, \ldots, x_n$.

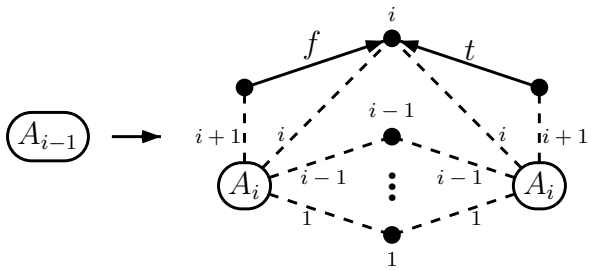
QUESTION: Is Θ true?

For k even, the corresponding problem that starts with a block of existential quantifiers is $\mathbf{\Sigma_k^p}$-complete. In the following, we will only consider the case that k is odd, the case k even can be dealt analogously. Thus, let us take an instance Θ of QSAT_k of the above form. Assume that $\varphi \equiv C_1 \vee C_2 \vee \cdots \vee C_m$ where every C_i is a conjunction of exactly three literals.

We define a hierarchical graph definition $D = (\Gamma, N, S, P)$ as follows: Let $N = \{S\} \cup \{A_i \mid 0 \leq i \leq n\}$, where $\mathrm{rank}(S) = 0$ and $\mathrm{rank}(A_i) = i + 1$. The terminal alphabet Γ contains the symbols $g, c, t, f, n_1, n_2, n_3, p_1, p_2, p_3$, and root where $\mathrm{rank}(x) = 2$ for $x \in \{g, c, t, f, n_1, n_2, n_3, p_1, p_2, p_3\}$ and $\mathrm{rank}(\mathrm{root}) = 1$. Exactly one node is labeled with root; it is generated in the first step starting from the start nonterminal S:



The *root*-labeled node will become the root of a binary tree which is generated with the following productions, where $1 \leq i \leq n$:



Note that for a non-leaf of the generated binary tree, the edge from the left (resp. right) child is labeled with f for false (resp. t for true). Thus, a path in the tree defines a truth assignment for the boolean variables x_i ($1 \leq i \leq n$). Via the j-labeled tentacles ($1 \leq j \leq i+1$), every A_i-labeled hyperedge e gets access to all nodes of the binary tree that were produced by ancestor-hyperedges of e. These nodes form a path starting at the root.

Finally, for A_n we introduce the production $A_n \to G$, where G is the following $(n+1)$-pointed hypergraph:

- The node set contains the $n+1$ pin nodes (which correspond to the $n+1$ nodes along a path from the root to a leaf in the generated tree) plus m additional internal nodes $c_1, \ldots, c_m$, where node c_i corresponds to the conjunction C_i.
- There is a g-labeled (g for guess) edge from pin 1 (which accesses the root) to pin ℓ_1, there is a g-labeled edge from pin ℓ_{i-1} to pin ℓ_i for $1 < i < k$, and there is a g-labeled edge from pin ℓ_{k-1} to pin $n+1$. These g-labeled edges allow to go from the root to a leaf of the tree in only k steps; thus, they provide shortcuts in the tree and will enable us to produce a truth assignment for the boolean variables $x_1, \ldots, x_n$ with only k edge traversals.
- There is a c-labeled (c for conjunction) edge from pin $n+1$ (which accesses a leaf in the tree) to each of the internal nodes $c_1, \ldots, c_m$, i.e., to each of the m conjunctions.
- There is a p_k-labeled edge (resp. n_k-labeled edge), where $k \in \{1, 2, 3\}$, from node c_i to pin $j+1$ ($1 \leq j \leq n$) if and only if x_j (resp. $\neg x_j$) is the k-th literal in the conjunction C_i.

This concludes the description of the hierarchical graph definition D. Let us consider an example for the last rule. Assume that

$$\theta \equiv \forall x_1 \forall x_2 \exists x_3 \exists x_4 \forall x_5 \forall x_6 \left\{ \begin{array}{l} (\neg x_1 \wedge \neg x_3 \wedge x_4) \ \vee \ (x_1 \wedge x_2 \wedge x_3) \vee \\ (x_3 \wedge x_4 \wedge x_5) \ \vee \ (\neg x_4 \wedge \neg x_5 \wedge \neg x_6) \end{array} \right\}.$$

Thus, $k = 3$, $n = 6$, $m = 4$, and the right-hand side for A_6 is shown in Figure 1. We have labeled the nodes $c_1, \ldots, c_m$ with the corresponding conjunction, but note that these conjunctions do not appear as node labels in the actual right-hand side. For the above formula, Figure 2 shows the path in eval(D) that corresponds to the truth assignment $x_1 = f, x_2 = x_3 = t, x_4 = x_5 = x_6 = f$. By construction of D, a leaf z of the binary tree, which corresponds to a boolean assignment for the variables $x_1, \ldots, x_n$, satisfies the disjunction $C_1 \vee C_2 \vee \cdots \vee C_m$ of the m conjunctions if and only if

$$\exists y, y_1, y_2, y_3, y_1', y_2', y_3' : z \xrightarrow{c} y \ \wedge \ \bigwedge_{i=1}^{3} (y \xrightarrow{p_i} y_i \xrightarrow{t} y_i' \vee y \xrightarrow{n_i} y_i \xrightarrow{f} y_i'). \tag{3}$$

Using the edge $z \xrightarrow{c} y$ we guess a conjunction that will evaluate to true under the assignment represented by the leaf z. Then $y \xrightarrow{p_i} y_i \xrightarrow{t} y_i' \vee y \xrightarrow{n_i} y_i \xrightarrow{f} y_i'$ checks whether the i-th literal of the guessed conjunction evaluates to true. For instance, for the path in Figure 2, the formula in (3) is indeed true; we have to choose the conjunction $\neg x_4 \wedge \neg x_5 \wedge \neg x_6$ for the FO variable y. From this observation, it follows that for the FO sentence

$$\psi \equiv \forall z_0 \in \text{root} \, \forall z_1 : z_0 \xrightarrow{g} z_1 \ \Rightarrow \ \exists z_2 : z_1 \xrightarrow{g} z_2 \ \wedge \ \cdots \forall z_k : z_{k-1} \xrightarrow{g} z_k \ \Rightarrow$$
$$\exists y, y_1, y_2, y_3, y_1', y_2', y_3' : z_k \xrightarrow{c} y \ \wedge \ \bigwedge_{i=1}^{3} (y \xrightarrow{p_i} y_i \xrightarrow{t} y_i' \vee y \xrightarrow{n_i} y_i \xrightarrow{f} y_i')$$

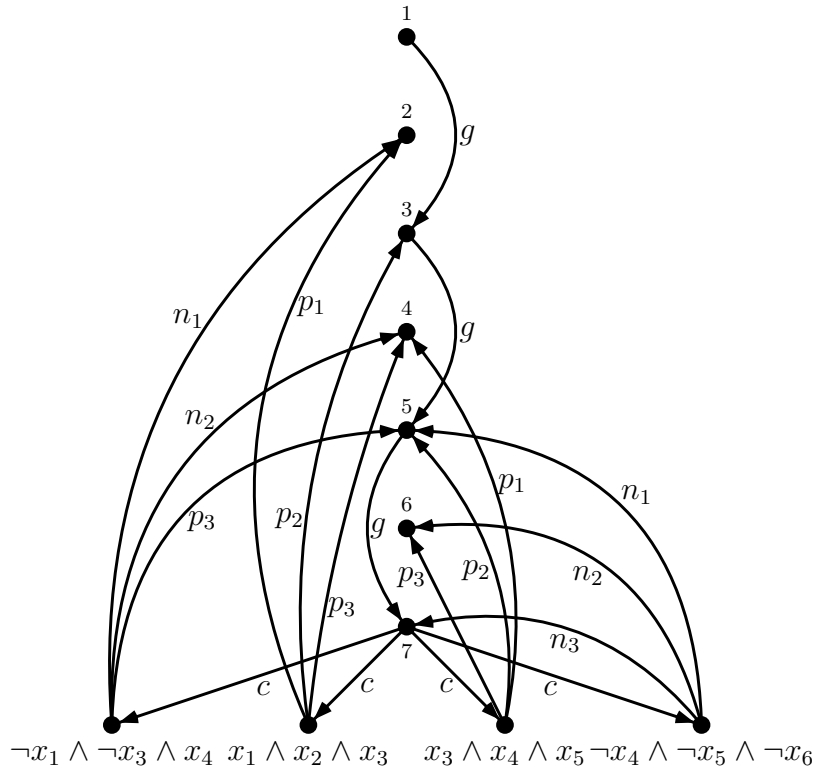


Figure 1:

we have $\mathrm{eval}(D) \models \psi$ if and only if θ is a true instance of QSAT_k. If we bring ψ into prenex normal form, we obtain a Π_{k+1}-FO sentence. Finally, note that $\mathrm{eval}(D) \models \psi$ if and only if $\mathrm{eval}(D), \mathit{root} \models \psi'$, where ψ' is the following sentence of modal logic:

$$\underbrace{[g]\langle g\rangle \cdots [g]}_{k \text{ many}}\langle c\rangle \bigwedge_{i=1}^{3} (\langle p_i\rangle\langle t\rangle tt \ \vee \ \langle n_i\rangle\langle f\rangle tt)$$

By the remark from the end of Section 7, this modal sentence is equivalent to an FO^2-sentence. This proves the theorem. □

In the rest of this section we consider two structural restrictions of hierarchical graph definitions that lead to more efficient model-checking algorithms for FO.

A hierarchical graph definition $D = (\Gamma, N, S, P)$ is *apex* if for every production $A \to (H, \sigma)$ from P the following holds: For every $e \in E^H$ such that $\lambda^H(e) = (B, \tau)$ for some $B \in N$ we have $\mathrm{ran}(\sigma) \cap \mathrm{ran}(\tau) = \emptyset$. Thus, pin nodes of a right-hand side cannot be accessed by nonterminal hyperedges. Apex hierarchical graph definitions are called 1-level restricted in [34]. We will prove that under the apex restriction the data complexity for FO over hierarchically defined

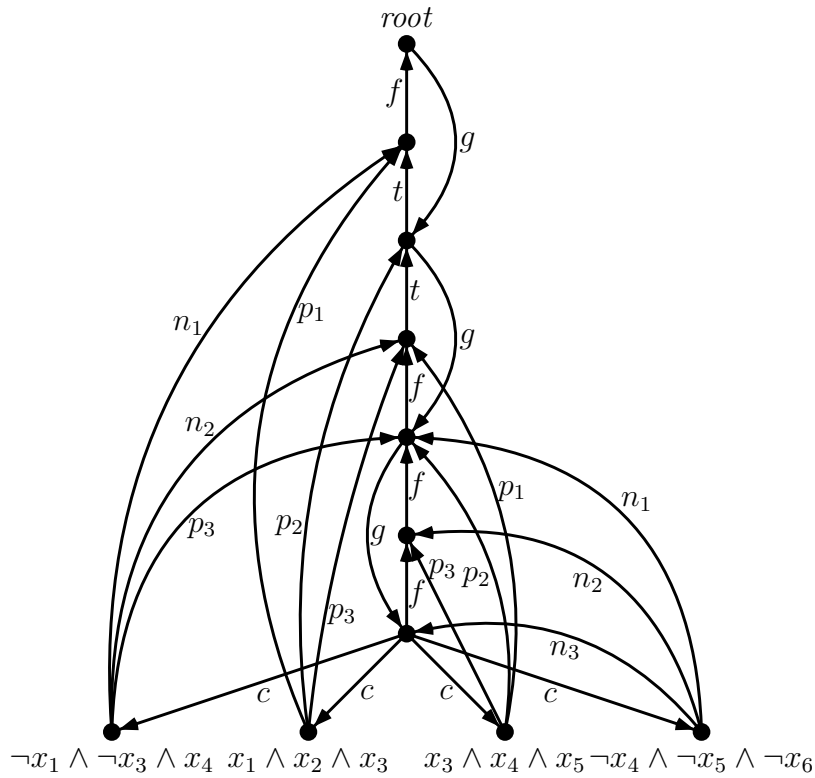


Figure 2:

input graphs becomes **NL**. The proof of this result is based on Gaifman's locality theorem [18]. First we have to introduce a few notations.

For a given relational structure $\mathcal{A} = (A, R_1, \ldots, R_k)$, where R_i is a relation of arbitrary arity n_i over A, we define the *Gaifman-graph* $G_{\mathcal{A}}$ of the structure $\mathcal{A}$ as the following undirected graph:

$$G_{\mathcal{A}} = (A, \{(a,b) \in A \times A \mid \bigvee_{1 \le i \le k} \exists (c_1, \ldots, c_{n_i}) \in R_i \; \exists j,k : c_j = a \neq b = c_k\}).$$

Thus, two nodes are adjacent in the Gaifman-graph if the nodes are related by some of the relations of the structure $\mathcal{A}$. For $a, b \in A$ we denote with $d_{\mathcal{A}}(a,b)$ the distance between a and b in the Gaifman-graph $G_{\mathcal{A}}$. Note that $d_{\mathcal{A}}(x,y) \le r$ can be expressed by an FO formula over the signature of $\mathcal{A}$. We just write $d(x,y) \le r$ for this FO formula. For $r \ge 0$ and $a \in A$, the r-sphere $S_{\mathcal{A}}(r,a)$ is the set of all $b \in A$ such that $d_{\mathcal{A}}(a,b) \le r$. With $N_{\mathcal{A}}(r,a) = (S_{\mathcal{A}}(r,a), (R_i \cap S_{\mathcal{A}}(r,a)^{n_i})_{1 \le i \le k})$ we denote the restriction of the structure $\mathcal{A}$ to the r-sphere $S_{\mathcal{A}}(r,a)$.

Now let φ be an FO formula over the signature of $\mathcal{A}$ and let x be a variable. Then the FO formula $\varphi^{(r,x)}$ results from φ by relativizing all quantifiers to $S_{\mathcal{A}}(r,x)$. It can be defined inductively, for instance $(\varphi_1 \wedge \varphi_2)^{(r,x)} \equiv \varphi_1^{(r,x)} \wedge \varphi_2^{(r,x)}$, $(\exists y\, \varphi)^{(r,x)} \equiv \exists y \{d(x,y) \le r \;\wedge\; \varphi^{(r,x)}\}$ (where

y has to be renamed into a fresh variable if $y = x$), and $R_i(x_1, \ldots, x_n)^{(r,x)} \equiv R_i(x_1, \ldots, x_n)$ for atomic formulas. It is allowed that the formula φ contains the variable x free. Moreover, the formula $\varphi^{(r,x)}$ certainly contains x free if φ contains at least one quantifier. If φ contains at most x free, then we write $(N_{\mathcal{A}}(r,a), a) \models \varphi(x)^{(r,x)}$ if the formula $\varphi(x)^{(r,x)}$ is true in the sphere $N_{\mathcal{A}}(r,a)$ when the variable x is set to a. Gaifman's Theorem states the following [18].

Theorem 8.4. *Every FO sentence is logically equivalent to a boolean combination of sentences of the form*

$$\exists x_1 \cdots \exists x_m \left\{ \bigwedge_{1 \le i < j \le m} d(x_i, x_j) > 2r \ \wedge \bigwedge_{1 \le i \le m} \psi(x_i)^{(r,x_i)} \right\}$$

where $\psi(x)$ is an FO formula that contains at most x free and $\psi(x_i)$ results from $\psi(x)$ by replacing every free occurrence of x by x_i.

Theorem 8.5. *For every fixed FO sentence φ, the question, whether* $\mathrm{eval}(D) \models \varphi$ *for a given apex hierarchical graph definition D, is in* **NL**.

Proof. By Gaifmans's Theorem it suffices to consider a fixed local sentence of the form

$$\exists x_1 \cdots \exists x_m \left\{ \bigwedge_{1 \le i < j \le m} d(x_i, x_j) > 2r \ \wedge \bigwedge_{1 \le i \le m} \psi(x_i)^{(r,x_i)} \right\}. \tag{4}$$

Thus, for a given hierarchical graph definition $D = (\Gamma, N, S, P)$ we have to check, whether there are at least m disjoint r-spheres in $\mathrm{eval}(D)$ that satisfy $\psi(x)^{(r,x)}$. Let $d = 2r$ be the diameter of r-spheres. We say that a sphere $S_{\mathrm{eval}(D)}(r,a)$ is a *ψ-sphere*, if $(N_{\mathrm{eval}(D)}(r,a), a) \models \psi(x)^{(r,x)}$.

Let $A \in N$ and let $A \to (H, \sigma)$ be the production for A. Let $\mathrm{eval}_D(A) = (H', \sigma')$. We identify V^H with a subset of $V^{H'}$ in the natural way. Then we say that $\mathrm{eval}_D(A)$ contains a *top-level occurrence of a ψ-sphere*, if there exists $v \in V^{H'}$ such that (i) $S_{H'}(v,r) \cap V^H \neq \emptyset$, (ii) $S_{H'}(v,r) \cap \mathrm{ran}(\sigma') = \emptyset$, and (iii) $(N_{H'}(v,r), v) \models \psi(x)^{(r,x)}$. This means that if we consider a subgraph of $\mathrm{eval}(D)$ that is generated from the nonterminal A, then this subgraph completely contains a ψ-sphere (by (ii) and (iii)). Moreover, this sphere is *not* completely generated by a smaller (w.r.t. the hierarchical order $\succ_D$) nonterminal (by (i)). Note that the pin nodes of $\mathrm{eval}_D(A)$ are generated by a nonterminal that is larger than A w.r.t. the hierarchical order $\succ_D$; thus, we exclude them from a potential top-level occurrence of a ψ-sphere in (ii).

Claim 1: We can verify in **L**, whether $\mathrm{eval}_D(A)$ contains a top-level occurrence of a ψ-sphere.

Proof of Claim 1. Due to the apex restriction, if $\mathrm{eval}_D(A)$ contains a top-level occurrence of a ψ-sphere, then every node of that occurrence is generated by a nonterminal B that is at most d steps below A in $\mathrm{dag}(D)$. Thus, in order to search for a top-level occurrence of a ψ-sphere in $\mathrm{eval}_D(A)$ we only have to unfold the nonterminal A up to depth d. Since d is a fixed constant, this partial unfolding results in a graph of polynomial size. Every node of that graph can be represented in logarithmic space. A more formal exposition follows.

We define a hierarchical graph definition $D(d, A)$ as follows:

- The set of terminals is Γ.
- The set of nonterminals contains for every $B \in N$ and every $0 \leq i \leq d+1$ a copy B_i of the same rank as B.
- The start nonterminal is A_{d+1}.
- The set of productions contains the following productions:
  - For every $1 \leq i \leq d+1$ and every $B \in N$ with $(B \to (H, \sigma)) \in P$ we take the production $B_i \to (H_{i-1}, \sigma)$, where H_{i-1} results from G by replacing every nonterminal label C in H by C_{i-1}. Moreover, if $(A \to (H, \sigma)) \in P$, then in H_d we additionally label every internal node $v \in V^H \setminus \mathrm{ran}(\sigma)$ with the new node label α and we label every pin node $v \in \mathrm{ran}(\sigma)$ with the new node label β.
  - For every $B \in N$ with $(B \to (H, \sigma)) \in P$ we take the production $B_0 \to (H', \sigma)$, where H' results from H by (i) removing all nonterminal hyperedges and (ii) labeling every node $v \in V^H$ that is accessed by some tentacle of a nonterminal hyperedge of H (i.e., $v \in \mathrm{ran}(\tau)$ for some $e \in E^H$ with $\lambda^H(e) = (C, \tau)$, $C \in N$) with the node label β.

Clearly, $D(d, A)$ can be constructed in logspace. Due to the apex restriction, $\mathrm{eval}_D(A)$ contains a top-level occurrence of a ψ-sphere if and only if

$$\mathrm{eval}(D(d, A)) \models \exists x \left\{ \psi^{(r,x)} \wedge \forall y \in \beta : d(x, y) > r \wedge \exists y \in \alpha : d(x, y) \leq r \right\}. \tag{5}$$

Note that this is a fixed FO sentence. Moreover, the representation of a node from the graph $\mathrm{eval}(D(d, A))$ according to Remark 5.5 can be stored in logarithmic space: it is a pair (p, v), where v is an internal node in a right-hand side and p is a root-path in $\mathrm{dag}(D(d, A))$, and this root-path has length at most d (a constant). Every number in the path p needs logarithmic space (it denotes a hyperedge in a right-hand side). Since by Remark 5.5 we can also check in $\mathbf{L}$, whether a tuple of nodes in $\mathrm{eval}(D(d, A))$ is connected via a hyperedge, any $\mathbf{L}$-algorithm for verifying a fixed FO sentence over an explicitly given input graph can be also applied to check whether (5) holds. This proves Claim 1.

Let N_{top} be the set of those $A \in N$ such that $\mathrm{eval}_D(A)$ contains a top-level occurrence of a ψ-sphere. Thus, we can check in $\mathbf{L}$ whether a given nonterminal belongs to N_{top}. In the following we assume that $S \in N_{\mathrm{top}}$. The other case can be dealt analogously. Let $\mathcal{P}(D)$ be the set of all root-paths in $\mathrm{dag}(D)$ that end at some nonterminal from N_{top} and that are not a proper prefix of some other root-path that is also ending in some nonterminal from N_{top}.

Claim 2: $\mathrm{eval}(D)$ contains at least $|\mathcal{P}(D)|$ many disjoint ψ-spheres.

Proof of Claim 2. Each of the root-paths in $\mathcal{P}(D)$ ends at some nonterminal from N_{top} and hence it gives rise to an occurrence of a ψ-sphere in $\mathrm{eval}(D)$. Since none of the root-paths in $\mathcal{P}(D)$ is a prefix of another root-path in $\mathcal{P}(D)$, all these ψ-spheres are pairwise disjoint. Thus, there are at least $|\mathcal{P}(D)|$ many disjoint ψ-spheres. This proves Claim 2.

For a number $n \in \mathbb{N}$ define $n\rceil^m$ by

$$n\rceil^m = \begin{cases} n & \text{if } n \leq m \\ m & \text{otherwise} \end{cases}$$

Recall that m is a fixed constant in our consideration (it appears in the sentence (4)).

Claim 3: We can verify in **NL**, whether $|\mathcal{P}(D)|\rceil^m$ equals a given number $k \in \{0, \ldots, m\}$.

Proof of Claim 3. For a given number k we first guess a number $0 \leq j \leq k$ and we guess j many nonterminals $A_1, \ldots, A_j \in N_{\text{top}}$. Next we guess for every $1 \leq i \leq j$ a number $k_i \in \{1, \ldots, k\}$ such that $k = k_1 + k_2 + \cdots k_j$. Note that these data can be stored in logarithmic space, because k is bounded by the fixed constant m. We now verify the following:

1. For every $1 \leq i \leq j$, in $\text{dag}(D)$ there are at least k_i many different root-paths ending in A_i.

2. For every $1 \leq i \leq j$, and for all $B \in N_{\text{top}} \setminus \{A_i\}$, there is no path from A_i to B in $\text{dag}(D)$.

First note that these conditions ensure that $|\mathcal{P}(D)|\rceil^m \geq k$. To verify condition (1) in **NL**, we use k_i (which is bounded by the constant m) many pointers for tracing the k_i many different paths. Condition (2) is a **coNL** condition; thus, the whole algorithm is an alternating logspace algorithm with at most one alternation; hence, it can be transformed into an **NL**-algorithm [23, 43]. Thus, we can check in **NL**, whether $|\mathcal{P}(D)|\rceil^m \geq k$. Using the complement closure of **NL** we can also check in **NL**, whether $|\mathcal{P}(D)|\rceil^m < k + 1$ (if $k < m$). This proves Claim 3.

Our overall **NL**-algorithm first verifies in **NL** whether $|\mathcal{P}(D)|\rceil^m = m$, i.e., whether $|\mathcal{P}(D)| \geq m$. If this is true, then by Claim 2 $\text{eval}(D)$ contains at least m disjoint ψ-spheres and we can accept. Thus, let us assume in the following that $|\mathcal{P}(D)| < m$.

For $A \in N \setminus N_{\text{top}}$ and $B \in N_{\text{top}}$ denote with $p(A, B)$ the number of all paths p in $\text{dag}(D)$ such that (i) p is path from A to B and (ii) except the last node B, p does not visit any other nodes from N_{top}.

Claim 5: We can verify in **NL**, whether $p(A, B)\rceil^m$ equals a given number from $\{0, \ldots, m\}$.

Proof of Claim 5. Similarly to the proof of Claim 3.

We next define a new hierarchical graph definition $D(d, m)$ as follows:

- The set of terminals is the same as for D.

- The set of nonterminals of $D(d, m)$ contains:

  - all $A \in N_{\text{top}}$,
  - for all $A \in N_{\text{top}}$ a copy A' of rank 0, and
  - for all $A \in N \setminus N_{\text{top}}$ and all $0 \leq i \leq d + 1$ a copy A_i (of the same rank as A).

- The start nonterminal of $D(d, m)$ is S (recall that we assume $S \in N_{\text{top}}$).

- The set of productions of $D(d,m)$ contains the following productions:
  - For every $A \in N_{\text{top}}$ such that $(A \to (H,\sigma)) \in P$ we take the productions $A \to (H_{d+1},\sigma)$ and $A' \to (H'_{d+1},\emptyset)$. Here H_{d+1} results from H by replacing every label $B \in N \setminus N_{\text{top}}$ by B_{d+1} and H'_{d+1} is the same hypergraph but with all pin nodes labeled with a new node label β (note that $(H'_{d+1},\emptyset)$ is a 0-pointed hypergraph).
  - For every $A \in N \setminus N_{\text{top}}$ such that $(A \to (H,\sigma)) \in P$ and every $1 \le i \le d+1$ we take the production $A_i \to (H_{i-1},\sigma)$, where H_{i-1} is defined as above (with $i-1$ instead of $d+1$).
  - Finally, for every $A \in N \setminus N_{\text{top}}$ we take the production $A_0 \to G$. Here G is the rank(A)-pointed hypergraph that consists of rank(A) many isolated pin nodes that are labeled with β and $p(A,B)\rceil^m$ many isolated B'-labeled hyperedges (for every $B \in N_{\text{top}}$).

The idea underlying this definition is the following: We unfold nonterminals from $N \setminus N_{\text{top}}$ in the same way as in D but only up to depth d, because by the apex restriction this is sufficient in order to generate the part of the graph that belongs to any ψ-sphere that is generated by a nonterminal from N_{top} on a higher hierarchical level. From a nonterminal A_0 (with $A \in N \setminus N_{\text{top}}$) we make a shortcut and directly produce those nonterminals from N_{top} (more precisely, the corresponding copies from N'_{top}) that are below A in dag(D). The following claim follows directly.

Claim 6: eval(D) contains m disjoint ψ-spheres if and only if eval$(D(d,m))$ contains m disjoint ψ'-spheres, where $\psi' = \psi \wedge \forall y : y \notin \beta$.

Claim 7: The function that maps a hierarchical graph definition D to $D(d,m)$ can be calculated in nondeterministic logspace.

Proof of Claim 7. In fact, the construction of $D(d,m)$ from D can be done in deterministic logspace, except for the calculation of the values $p(A,B)\rceil^m$. Here, we simply guess the value $p(A,B)\rceil^m$ and verify the correctness of the guess in **NL** using Claim 5.

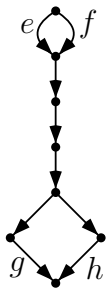
By Claim 6 and 7 as well as the closure of **NL** under **NL**-reductions, it suffices to verify in **NL**, whether eval$(D(d,m))$ (represented by $D(d,m)$ on the input tape) contains at least m disjoint ψ'-spheres. This will be shown in the rest of the proof.

Let $\widehat{N}_{\text{top}} = N_{\text{top}} \cup N'_{\text{top}}$. Similarly to $\mathcal{P}(D)$, define $\mathcal{P}(D(d,m))$ as the set of all root-paths in dag$(D(d,m))$ that end at a nonterminal from $\widehat{N}_{\text{top}}$ and that are not a proper prefix of some other root-path also ending in a nonterminal from $\widehat{N}_{\text{top}}$. From the definition of $D(d,m)$ it follows that $|\mathcal{P}(D)|\rceil^m = |\mathcal{P}(D(d,m))|\rceil^m$; thus, also $|\mathcal{P}(D(d,m))| < m$. Each of the paths from $\mathcal{P}(D(d,m))$ can be represented by a sequence $(e_1,\ldots,e_k)$ of edges of dag$(D(d,m))$, where

- e_1 starts in the root of $D(d,m)$ (i.e., in S), e_k is the last edge of the path (i.e., it ends in a nonterminal from $\widehat{N}_{\text{top}}$),
- $e_2,\ldots,e_{k-1}$ are intermediate edges of the path,

- $k \leq |\mathcal{P}(D(d,m))| < m$, and
- there is exactly one path from the target of e_i to the source of e_{i+1} for $1 \leq i < k$.

Let us consider an example. Assume that $\mathrm{dag}(D(d,m))$ looks as follows, where e, f, g, and h are edge names. of the edges received names:



Then each of the four different paths (of length 6) from the root to the leaf can be specified with only 2 edges: (e,g), (e,h), (f,g), and (f,h).

Claim 8: Every node of $\mathrm{eval}(D(d,m))$ can be represented in space $\mathcal{O}(\log(|D(d,m)|))$ (in particular the size of $\mathrm{eval}(D(d,m))$ is bounded polynomially in the size of $D(d,m)$).

Proof of Claim 8. According to Remark 5.5, a node of $\mathrm{eval}(D(d,m))$ is represented by a pair (p,v), where p is a root-path in $\mathrm{dag}(D(d,m))$ (ending in a nonterminal A) and v is an internal node in the right-hand side of A. Thus, it suffices to show that an arbitrary root-path in $\mathrm{dag}(D(d,m))$ can be stored in logspace. Note that in $\mathrm{dag}(D(d,m))$, every nonterminal of $D(d,m)$ has distance at most $d+1$ from a nonterminal of $\widehat{N}_{\mathrm{top}}$. Since $d+1$ is a fixed constant, it suffices to store an arbitrary root-path in $\mathrm{dag}(D(d,m))$ ending at a nonterminal from $\widehat{N}_{\mathrm{top}}$ in logspace. Now, every root-path ending in a nonterminal from $\widehat{N}_{\mathrm{top}}$ is a prefix of some path from $\mathcal{P}(D(d,m))$. By the remark preceding Claim 8, such a path can be represented by a sequence $(e_1,\ldots,e_k)$ of $k < m$ edges of $D(d,m)$. Since m is a fixed constant, logarithmic space suffices. To sum up, a node of $\mathrm{eval}(D(d,m))$ can be represented by a tuple $((e_1,\ldots,e_k),q,v)$, where $(e_1,\ldots,e_k)$ is as above, q specifies a path of length at most $d+1$ in $\mathrm{dag}(D(d,m))$ that starts at the target node of the edge e_k and ends in some nonterminal A, and v is an internal node in the right-hand side of A.

Claim 9: Let $(u_1,\ldots,u_\ell)$ be a tuple of nodes of $\mathrm{eval}(D(d,m))$ represented as in the proof of Claim 8. Then, given $D(d,m)$ and $(u_1,\ldots,u_\ell)$ as input, it can be checked in **NL**, whether $u_1,\ldots,u_\ell$ are connected by a hyperedge in $\mathrm{eval}(D,m)$.

Proof of Claim 9. Let $((e_1,\ldots,e_k),q,v)$ be the logspace representation of u_i from the proof of Claim 8. Thus, $((e_1,\ldots,e_k),q)$ represents a root-path p in $\mathrm{dag}(D(d,m))$. Then, (p,v) is the ordinary (polynomial size) representation of u_i according to Remark 5.5. Note that the function that maps $((e_1,\ldots,e_k),q)$ to p can be calculated in nondeterministic logspace by simply guessing the path p in $\mathrm{dag}(D(d,m))$ and thereby checking whether each of the edges

e_i is visited and that the path q is a suffix of the path p. Now, by the second statement of Remark 5.5, given the ordinary (polynomial size) representation of $u_1, \ldots, u_\ell$, it can be checked in logspace, whether these nodes are connected by a hyperedge in $\mathrm{eval}(D, m)$. Claim 9 follows from the closure of **NL** under **NL**-reductions.

Now we can finish the proof of the theorem. Recall that it suffices to check, whether $\mathrm{eval}(D(d, m))$ (represented by $D(d, m)$) contains at least m disjoint ψ'-spheres. Thus, we have to verify a fixed first-order sentence φ in $\mathrm{eval}(D(d, m))$. We will do this using an alternating logspace machine, where the number of alternations is bounded by the number of quantifier alternations of φ (a fixed constant). For each existential (universal) quantifier of φ we guess existentially (universally) a node u of $\mathrm{eval}(D(d, m))$ using the logspace representation from Claim 8. After guessing such a representation, we have to verify that the guessed data indeed represents a node of $\mathrm{eval}(D(d, m))$, but this is clearly possible in **NL**. Finally, we have to verify atomic statements on the logspace representations of the guessed nodes, which is possible in **NL** by Claim 9. This finishes the proof of the theorem. □

For $c \in \mathbb{N}$, a hierarchical graph definition $D = (\Gamma, N, S, P)$ is *c-bounded* if $\mathrm{rank}(A) \leq c$ for every $A \in N$ and every right-hand side of a production from P contains at most c hyperedges that are labeled with a nonterminal. The crucial fact for the further consideration is that for a given *c-bounded* hierarchical graph definition D we can construct an equivalent graph straight-line program $\mathcal{S} = (X_i := t_i)_{1 \leq i \leq n}$ in the sense of Section 6 such that for every formal variable X_i, $\mathrm{type}(X_i) \leq d(c)$. Here $d(c)$ is a constant that only depends on c.

Theorem 8.6. *For every fixed FO sentence φ, every fixed $c \in \mathbb{N}$, and every fixed ranked alphabet Γ, the question, whether* $\mathrm{eval}(D) \models \varphi$ *for a given c-bounded hierarchical graph definition D with terminal alphabet Γ, is in* **P**.

Proof. The basic idea for the proof of the theorem is based on Courcelle's technique for evaluating fixed MSO formulas in linear time over graph classes of bounded tree width [9]. Let φ be a fixed FO sentence of quantifier rank k. Let $D = (\Gamma, N, S, P)$ be a c-bounded hierarchical graph definition over the fixed terminal alphabet Γ. By the remark above, we can construct from D an equivalent graph straight-line program $\mathcal{S} = (X_i := t_i)_{1 \leq i \leq n}$ over the ranked alphabet Γ such that for every formal variable X_i, $\mathrm{type}(X_i) \leq d(c)$. Thus, for every $1 \leq i \leq n$, the graph $\mathrm{eval}(X_i)$ can be viewed as a relational structure over some subsignature Θ_i of the *fixed signature* $\Theta = \Gamma \cup \{\mathrm{pin}(1), \ldots, \mathrm{pin}(d(c))\}$. Since this signature Θ is fixed (i.e., does not vary with the input) and since moreover also the quantifier rank k is fixed in the theorem, the number of pairwise nonequivalent FO sentences of quantifier rank at most k over the signature Θ is bounded by some constant $g(k)$. Thus, also the number of possible k-FO theories (in the sense of Section 7) over the signature Θ is bounded by some constant.

By [10] (see also [14, 32]), there exist functions $F_\oplus$, $F_{a,b}$, and F_f (where $1 \leq a, b \leq d(c)$ and $f : \{1, \ldots, a\} \to \{1, \ldots, b\}$) over the set of all k-FO theories over the signature Θ such that

- $\text{k-FOTh}(G_1 \oplus G_2) = F_\oplus(\text{k-FOTh}(G_1), \text{k-FOTh}(G_2))$,
- $\text{k-FOTh}(\mathrm{glue}_{a,b}(G)) = F_{a,b}(\text{k-FOTh}(G))$, and

- k-FOTh(rename$_f(G)$) $= F_f$(k-FOTh(G)).

Again, these functions do not depend from the input; they can be assumed to be given hard-wired.

Now we replace the graph straight-line program $\mathcal{S}$ by a straight-line program for calculating k-FOTh(eval($\mathcal{S}$)) as follows:

1. If $X_i := t_i$ is a definition from $\mathcal{S}$ such that t_i is an n-pointed ($n \leq d(c)$) hypergraph G, then we calculate k-FOTh(G), which is possible in polynomial time (in fact in $\mathbf{AC}^0$ [4, 24]) and replace the definition $X_i := t_i$ by $X_i :=$ k-FOTh(G).

2. A definition of the form $X_i := X_j \oplus X_k$ is replaced by $X_i := F_\oplus(X_j, X_k)$ and similarly for definitions of the form $X_i :=$ glue$_{a,b}(X_j)$ and $X_i :=$ rename$_f(X_j)$.

Note that this is a straight-line program over a fixed set, namely the set of all k-FO theories. Hence, we can evaluate this straight-line program in polynomial time and thereby calculate k-FOTh(eval($\mathcal{S}$)). □

One may generalize Theorem 8.6 by considering graph straight-line programs that in addition to the operators $\oplus$, glue$_{a,b}$, and rename$_f$ contain further operators that are compatible with the calculation of the k-FO theory, see [32] for such operations.

Theorem 8.3, 8.5, and 8.6 give us a clear picture on the conditions that make the model-checking problem for FO over hierarchically defined input graphs difficult: nonterminals have to access pin nodes (i.e., references can be passed along nonterminals) and nonterminals have to access an unbounded number of nodes.

8.2 Combined complexity

In the previous section, we have seen that for Σ_k-FO, data complexity increases considerably when moving from explicitly given input graphs to hierarchically defined input graphs (from $\mathbf{\Sigma_k^{log}}$ to $\mathbf{\Sigma_{k-1}^p}$). For the combined complexity of Σ_k-FO, such a complexity jump does not occur (recall that the combined complexity of Σ_k-FO for explicitly given input graphs is $\mathbf{\Sigma_k^p}$):

Theorem 8.7. *The following problem is $\mathbf{\Sigma_k^p}$-complete (resp. $\mathbf{\Pi_k^p}$-complete):*

INPUT: A hierarchical graph definition D and a Σ_k-FO sentence (resp. Π_k-FO sentence) φ
QUESTION: eval$(D) \models \varphi$ *?*

Proof. The lower bound follows from the corresponding result for explicitly given input graphs. For the upper bound we can follow the arguments for the upper bound from Theorem 8.3. □

For explicitly given input graphs, the combined complexity reduces from **PSPACE** to **P** when moving from FO to FO^m for some fixed m [46]. A slight modification of the proof of Theorem 8.3 shows that for hierarchically defined graphs, **PSPACE**-hardness already holds for the data complexity of FO^2 (without any restriction on the quantifier prefix). We just have to start from an instance of QBF (quantified boolean satisfiability) and carry out the construction in the proof of Theorem 8.3.

9 MSO and SO over hierarchically defined graphs

In this section we study the model-checking problem for MSO and SO over hierarchically defined input graphs. We prove that the data complexity of Σ_k-SO (resp. Π_k-SO) for hierarchically defined input graphs is $\mathbf{\Sigma_k^e}$ (resp. $\mathbf{\Pi_k^e}$) (Theorem 9.2). In fact, the lower bound already holds for Σ_k-MSO. For c-bounded hierarchical graph definitions we can show that the data complexity of Σ_k-MSO (resp. Π_k-MSO) goes down to $\mathbf{\Sigma_k^p}$ (resp. $\mathbf{\Pi_k^p}$) (Theorem 9.6). Finally, in Section 8.2 we show that also the combined complexity for Σ_k-SO (resp. Π_k-SO) and hierarchically defined input graphs is $\mathbf{\Sigma_k^e}$ (resp. $\mathbf{\Pi_k^e}$) (Theorem 9.7). In fact, the lower bound already holds for Σ_k-MSO and 2-bounded hierarchical graph definitions (Theorem 9.8).

We should remark that the apex restriction from Section 8.1 does not lead to more efficient model-checking algorithms in the context of MSO. For an arbitrary hierarchical graph definition D we can enforce the apex restriction by inserting additional edges (labeled with some new terminal α) whenever a tentacle of a nonterminal hyperedge accesses a pin node. If D' denotes this new hierarchical graph definition, then $\mathrm{eval}(D)$ results from $\mathrm{eval}(D')$ by contracting all α-labeled edges. But this contraction is MSO-definable.

9.1 Data complexity

In order to obtain a sharp lower bound on the data complexity of Σ_k-MSO over hierarchically defined graphs, we will use the following computational problem QOΣ_k-SAT (resp. QOΠ_k-SAT) for $k \geq 1$ (where QO stands for "quantified oracle"). For $m \geq 1$ let $\mathcal{F}_m$ be the set of all m-ary boolean functions. If k is even, then the input for QOΣ_k-SAT is a formula Θ of the form

$$\exists f_1 \in \mathcal{F}_m \cdots \forall f_k \in \mathcal{F}_m \exists \overline{x}_1, \ldots, \overline{x}_k \in \{0,1\}^m \, \exists \overline{y} \in \{0,1\}^\ell : \varphi((\overline{x}_i)_{1 \leq i \leq k}, \overline{y}, (f_i(\overline{x}_j))_{1 \leq i,j \leq k}),$$

where φ is a boolean formula in $mk + \ell + k^2$ boolean variables. For k odd, the input Θ for QOΣ_k-SAT has the form

$$\exists f_1 \in \mathcal{F}_m \forall f_2 \in \mathcal{F}_m \cdots \exists f_k \in \mathcal{F}_m \forall \overline{x}_1, \ldots, \overline{x}_k \in \{0,1\}^m \, \forall \overline{y} \in \{0,1\}^\ell : \\ \varphi((\overline{x}_i)_{1 \leq i \leq k}, \overline{y}, (f_i(\overline{x}_j))_{1 \leq i,j \leq k}).$$

In both cases, we ask whether Θ is a true formula. The problem QOΠ_k-SAT is defined analogously, we only start with a universal quantifier over $\mathcal{F}_m$.

Proposition 9.1. *For all $k \geq 1$, the problem QOΣ_k-SAT (resp. QOΠ_k-SAT) is $\mathbf{\Sigma_k^e}$-complete (resp. $\mathbf{\Pi_k^e}$-complete).*

Proof. We demonstrate the general idea for the class $\mathbf{\Sigma_3^e}$, the same ideas also work for the other classes. Let M be an alternating Turing-machine that accepts a $\mathbf{\Sigma_3^e}$-complete language $L(M)$ and let w be an input for M of length n. We may assume that M makes on every computation path exactly 2 alternations and that M makes exactly $2^{p(n)}$ (for a polynomial $p(n)$) transitions between two alternations. Thus, the total running time is $3 \cdot 2^{p(n)}$. Let $q = p(n)$.

There exists a polynomial time predicate ϕ over $\{0,1\}^*$ such that $w \in L(M)$ if and only if

$$\exists \overline{x}_1 \in \{0,1\}^{2^q} \forall \overline{x}_2 \in \{0,1\}^{2^q} \exists \overline{x}_3 \in \{0,1\}^{2^q} : \phi(w\,\overline{x}_1\,\overline{x}_2\,\overline{x}_3).$$

Since ϕ is a polynomial time predicate, we can apply the construction from the proof of the Cook-Levin Theorem and obtain a 3-CNF formula ψ_w of size exponential in $n = |w|$ such that $w \in L(M)$ if and only if

$$\exists \overline{x}_1 \in \{0,1\}^{2^q} \forall \overline{x}_2 \in \{0,1\}^{2^q} \exists \overline{x}_3 \in \{0,1\}^{2^q} \exists \overline{y} \in \{0,1\}^{2^{cq}} : \psi_w(\overline{x}_1\,\overline{x}_2\,\overline{x}_3\,\overline{y}),$$

where c is some constant. By padding the sequences $\overline{x}_1, \overline{x}_2$, and $\overline{x}_3\overline{y}$ to some length 2^m, where $m \in \mathcal{O}(q)$ and $2^m \geq 2^q + 2^{cq}$, we can bring the above formula into the form

$$\exists \overline{x}_1 \in \{0,1\}^{2^m} \forall \overline{x}_2 \in \{0,1\}^{2^m} \exists \overline{x}_3 \in \{0,1\}^{2^m} : \psi_w(\overline{x}_1\,\overline{x}_2\,\overline{x}_3). \tag{6}$$

We encode each of the $3 \cdot 2^m$ many variables in the sequence $\overline{x}_1\overline{x}_2\overline{x}_3$ by a pair $(i,b) \in \{0,1\}^2 \times \{0,1\}^m$. The pair (i,b) encodes the b-th variable of $\overline{x}_i$. Here b and i are interpreted as binary numbers. Let us denote this variable by $x(i,b)$. Then a typical clause from ψ_w has the form

$$(t_1 \oplus x(i_1,b_1)) \;\vee\; (t_2 \oplus x(i_2,b_2)) \;\vee\; (t_3 \oplus x(i_3,b_3)), \tag{7}$$

where $t_j \in \{0,1\}$, $i_j \in \{0,1\}^2$, $b_j \in \{0,1\}^m$, and $\oplus$ denotes the boolean exclusive or (note that $0 \oplus x = x$ and $1 \oplus x = \neg x$). Now the crucial point is that the clauses that are constructed in the proof of the Cook-Levin Theorem follow a very regular pattern. More precisely, from the input w it can be checked in polynomial time, whether a clause of the form (7) belongs to ψ_w. Thus, there exists a boolean predicate p_w, which can be computed in polynomial time from w such that (7) belongs to the 3-CNF formula ψ_w if and only if $p_w(b_1,b_2,b_3,i_1,i_2,i_3,t_1,t_2,t_3)$ is true, see also [3, proof of Prop. 4.2].

Let $\mathcal{F}_m$ be the set of all m-ary boolean functions. Then, (6) is equivalent to

$$\begin{aligned}
&\exists f_1 \in \mathcal{F}_m \forall f_2 \in \mathcal{F}_m \exists f_3 \in \mathcal{F}_m \\
&\qquad \forall b_1,b_2,b_3 \in \{0,1\}^m \forall i_1,i_2,i_3 \in \{0,1\}^2 \forall t_1,t_2,t_3 \in \{0,1\} : \\
&\qquad\qquad t_1 \oplus f_{i_1}(b_1) \;\vee\; t_2 \oplus f_{i_2}(b_2) \;\vee\; t_3 \oplus f_{i_3}(b_3) \;\vee \\
&\qquad\qquad\qquad \neg p_w(b_1,b_2,b_3,i_1,i_2,i_3,t_1,t_2,t_3).
\end{aligned}$$

Finally, we replace in this formula every $f_i(b)$ by

$$(i = 01 \;\wedge\; f_1(b)) \;\vee\; (i = 10 \;\wedge\; f_2(b)) \;\vee\; (i = 11 \;\wedge\; f_3(b)).$$

The resulting formula is of the desired form. □

Theorem 9.2. *For every fixed Σ_k-SO sentence (resp. Π_k-SO sentence) φ, the question, whether* $\mathrm{eval}(D) \models \varphi$ *for a given hierarchical graph definition D, is in $\mathbf{\Sigma_k^e}$ (resp. $\mathbf{\Pi_k^e}$).*

Moreover, for every level $\mathbf{\Sigma_k^e}$ (resp. $\mathbf{\Pi_k^e}$) of **EH**, *there exists a fixed Σ_k-MSO sentence (resp. Π_k-MSO sentence) φ such that the question, whether* $\mathrm{eval}(D) \models \varphi$ *for a given hierarchical graph definition D, is $\mathbf{\Sigma_k^e}$-hard (resp. $\mathbf{\Pi_k^e}$-hard).*

Proof. For the first statement, assume that

$$\varphi \equiv \exists \overline{X}_1 \forall \overline{X}_2 \cdots Q \overline{X}_k \, \psi(\overline{X}_1, \ldots, \overline{X}_k)$$

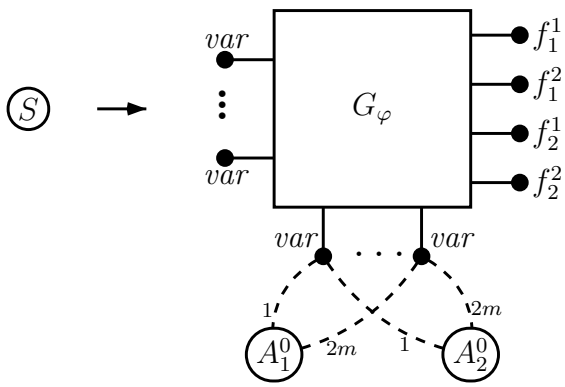
is a fixed Σ_k-SO sentence, where $\overline{X}_i$ is a tuple of SO variables, ψ is an FO formula, $Q = \exists$ if k is odd, and $Q = \forall$ if k is even. Our alternating exponential time algorithm guesses for every $1 \leq i \leq k$ a tuple $\overline{U}_i$ of relations over the set V of nodes of eval(D). For every quantified SO variable of arity m we have to guess (existentially if i is odd, universally if i is even) an m-ary relation over V. Since $|V|$ is bounded by $2^{\mathcal{O}(|D|)}$, the number of m-tuples in an m-ary relation is bounded by $2^{\mathcal{O}(m \cdot |D|)}$, which is exponential in the input size. Thus, an m-ary relation over V can be guessed in exponential time. At the end, we have to verify, whether eval$(D) \models \psi(\overline{U}_1, \ldots, \overline{U}_k)$, where ψ only contains FO quantifiers. This is possible in deterministic exponential time: Assume that ψ is in prenex normal form and contains ℓ distinct FO variables. Then there are only $2^{\mathcal{O}(\ell \cdot |D|)}$ many assignments from the set of FO variables to the nodes of eval(D). Next, unfold the graph eval(D) and check for each of the $2^{\mathcal{O}(\ell \cdot |D|)}$ possible assignments, whether the quantifier-free kernel of ψ evaluates to true under that assignment. This takes time $2^{\mathcal{O}(\ell \cdot |D|)} \cdot 2^{\mathcal{O}(|D|)}$, i.e., exponential time. From the resulting data we can easily determine in exponential time, whether eval$(D) \models \psi(\overline{U}_1, \ldots, \overline{U}_k)$. Thus, we obtain an exponential time algorithm with precisely $k-1$ alternations.

The second statement from the theorem will be shown by a reduction from QOΣ_k-SAT (resp. QOΠ_k-SAT). We will present the construction only for QOΣ_2-SAT, the general case can be dealt analogously. Thus, let Θ be a formula of the form

$$\exists f_1 \in \mathcal{F}_m \forall f_2 \in \mathcal{F}_m \exists \overline{x}_1, \overline{x}_2 \in \{0,1\}^m \, \exists \overline{y} \in \{0,1\}^\ell : \varphi(\overline{x}_1, \overline{x}_2, \overline{y}, (f_i(\overline{x}_j))_{1 \leq i,j \leq 2}). \tag{8}$$

We will construct a hierarchical graph definition D and a fixed Σ_2-MSO sentence ψ such that Θ is a positive QOΣ_2-SAT-instance if and only if eval$(D) \models \psi$. In a first step we will construct a fixed Σ_3-MSO sentence with this property, then this sentence will be further reduced to an equivalent Σ_2-MSO sentence.

We will use the node labels tt, ff, AND, OR, NOT, *root*, *var*, f_1^1, f_2^1, f_1^2, and f_2^2 and the edge labels 1 and 2 (and an additional label for unlabeled edges). The nonterminals are S, $A_1^0, A_2^0, \ldots, A_1^m, A_2^m$, where rank$(S) = 0$ and rank$(A_i^j) = 2m + j$. The initial rule of D has the following form:

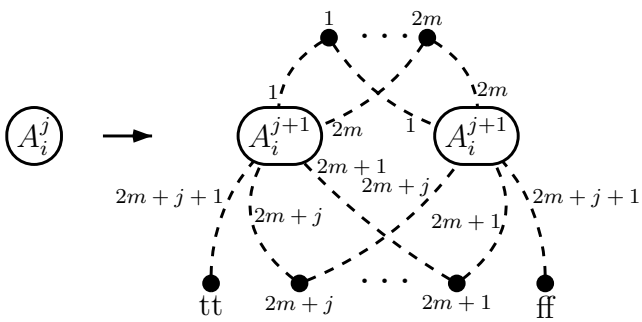


In the right-hand side, there are $2m + \ell$ many *var*-labeled nodes, which represent the variables in the sequences $\overline{x}_1, \overline{x}_2$, and $\overline{y}$. The *var*-labeled node that is connected via the i-th (resp. $(m+i)$-th) tentacle of the A_j^0-labeled hyperedge ($1 \leq i \leq m$, $j \in \{1,2\}$) represents the i-th variable of the sequence $\overline{x}_1$ (resp. $\overline{x}_2$). The ℓ remaining *var*-labeled nodes on the left side of the rectangular box represent the variables in $\overline{y}$. The unique f_i^j-labeled node represents the input $f_i(\overline{x}_j)$ of the formula φ. The box labeled with G_φ represents the boolean formula φ, encoded as a directed acyclic graph (dag) in the usual way. The nodes of this dag correspond to the subexpressions of φ, and every node is labeled with the topmost boolean operator (AND, OR, or NOT) of the corresponding subexpression of φ. The root of the dag is in addition also labeled with *root*. In the following let Λ denote those nodes labeled with a symbol from $\{\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}, \mathit{root}, \mathit{var}, f_1^1, f_2^1, f_1^2, f_2^2\}$. Assume that $X \subseteq \Lambda$. Then, the fixed formula $\mathrm{valid}(X)$, which is defined as

$$\mathrm{valid}(X) \equiv \forall x, y, z \in \Lambda \left\{ \begin{array}{l} (y \to x \leftarrow z \wedge y \neq z \wedge x \in \mathrm{AND}) \Rightarrow \\ \qquad (x \in X \Leftrightarrow y \in X \wedge z \in X) \wedge \\ (y \to x \leftarrow z \wedge y \neq z \wedge x \in \mathrm{OR}) \Rightarrow \\ \qquad (x \in X \Leftrightarrow y \in X \vee z \in X) \wedge \\ (y \to x \wedge x \in \mathrm{NOT}) \Rightarrow \\ \qquad (x \in X \Leftrightarrow y \notin X) \end{array} \right\} \wedge X \cap \mathit{root} \neq \emptyset$$

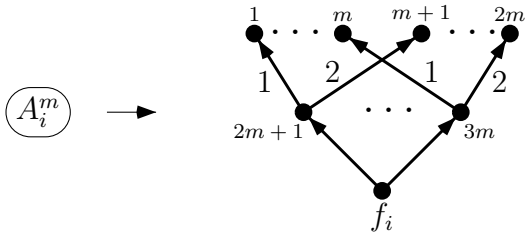
expresses that $X \subseteq \Lambda$ defines a consistent truth assignment to the subformulas of φ such that moreover φ evaluates to true (i.e., the root node belongs to X).

The nonterminals A_1^0 and A_2^0 generate a graph structure that enables us to quantify over two m-ary boolean functions. For this, we introduce the following rules, where $i \in \{1,2\}$ and $0 \leq j < m$:

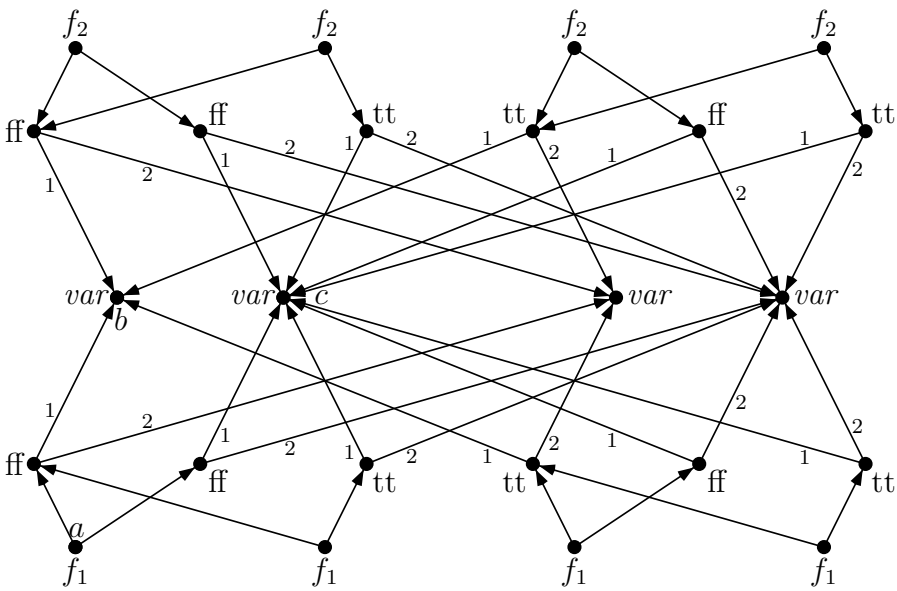


The tentacles with labels $1, \ldots, 2m$ of each A_i^j-labeled hyperedge access the $2m$ many *var*-labeled nodes that represent the variables in the sequences $\overline{x}_1$ and $\overline{x}_2$; thus, the access to these nodes is just passed from A_i^j to A_i^{j+1}. The other tentacles (with labels $2m+1, \ldots, 2m+j$) access nodes that are either labeled with tt or ff. These labels represent the truth values true and false, respectively. Note that in the production above, two new nodes are generated, one is labeled with tt and the other one is labeled with ff.

Finally, for $i \in \{1,2\}$ we introduce the following rule:



In general, pin $2m + i$ $(1 \leq i \leq m)$ is connected with each of the pins $(j-1)m + i$ (which represents the i-th variable of x_j) via a j-labeled edge $(1 \leq j \leq k = 2)$. For $i \in \{1, 2\}$ these productions generate 2^m many f_i-labeled nodes (because 2^m many A_i^m-labeled hyperedges are generated), one for each possible argument to the function f_i. Thus, a quantification over $f_i \in \mathcal{F}_m$ corresponds to a quantification over a subset F_i of the f_i-labeled nodes. The following picture shows for $m = k = 2$ the graph that is generated from the nonterminals A_1^0 and A_2^0. We did not draw multiple edges with the same label between two nodes. The three labels a, b, and c are introduced in order to denote these three nodes, they do not represent actual node labels.



The first two *var*-labeled nodes b and c represent the pair of variables $\overline{x}_1$ and the second two *var*-labeled nodes represent $\overline{x}_2$. The function $f_1 \in \mathcal{F}_2$ with $f_1(0,0) = 1$ and $f_1(0,1) = f_1(1,0) = f_1(1,1) = 0$ for instance is represented by the subset F_1 of the f_1-labeled nodes that contains only the left-most f_1-labeled node a. An assignment to all the $2m + \ell = 4 + \ell$ variables in the sequences $\overline{x}_1, \overline{x}_2$, and $\overline{y}$ will be encoded by a subset X of the *var*-labeled nodes that were generated with the rule for the start nonterminal S. For instance, if $b, c \notin X$, then this means that $(0,0)$ is assigned to $\overline{x}_1$. Then the fact that $f_1(\overline{x}_1) = f_1(0,0) = 1$ for this f_1 and $\overline{x}_1$ can be expressed by the fact that

$$\forall y \forall z : a \to y \xrightarrow{1} z \;\Rightarrow\; (y \in \mathrm{tt} \Leftrightarrow z \in X).$$

In general, if F_i is a subset of the f_i-labeled nodes that represents the function $f_i \in \mathcal{F}_m$ and X is a subset of the *var*-labeled nodes that represents an assignment to the variables in $\overline{x}_1, \overline{x}_2$, and $\overline{y}$, then the fact that $f_i(\overline{x}_j) = 1$ can be expressed by the following formula $\psi_{i,j}(F_i, X)$:

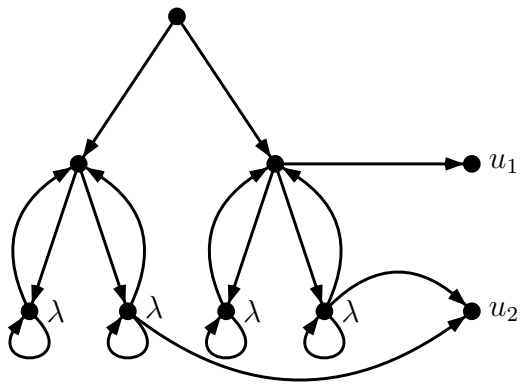
$$\psi_{i,j}(F_i, X) \;\equiv\; \exists x \in F_i \, \forall y \, \forall z : x \to y \xrightarrow{j} z \;\Rightarrow\; (y \in \mathrm{tt} \Leftrightarrow z \in X)$$

Now let ψ be the following Σ_3-MSO sentence (recall that Λ is the set of those nodes that are labeled with a symbol from $\{\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}, \mathit{root}, \mathit{var}, f_1^1, f_2^1, f_1^2, f_2^2\}$):

$$\psi \;\equiv\; \exists F_1 \subseteq f_1 \, \forall F_2 \subseteq f_2 \, \exists X \subseteq \Lambda : \left\{ \bigwedge_{1 \le i,j \le 2} X \cap f_i^j \neq \emptyset \Leftrightarrow \psi_{i,j}(F_i, X) \right\} \wedge \mathrm{valid}(X), \qquad (9)$$

where $X \cap f_i^j \neq \emptyset$ is an abbreviation for $\exists y : y \in X \wedge y \in f_i^j$. Then $\mathrm{eval}(D) \models \psi$ if and only if Θ in (8) is a positive $\mathrm{QO}\Sigma_2$-SAT-instance.

Note that the above sentence is a Σ_3-MSO sentence instead of a Σ_2-MSO sentence. On the other hand, the innermost existential MSO quantifier $\exists X \subseteq \Lambda$ ranges over a set of nodes of polynomial size in $\mathrm{eval}(D)$ (Λ has polynomial size). We will use this fact and replace $\exists X \subseteq \Lambda$ by an additional first-order quantifier. For this we have to introduce some additional graph structure of exponential size. Note that all nodes from Λ are generated directly from the start nonterminal S. Assume that $\delta = |\Lambda|$. We now add to the right-hand side of S a new nonterminal B of rank δ, whose tentacles access precisely the nodes from Λ. From B we generate a graph structure that is shown for $\delta = 2$ in the following picture, where $\Lambda = \{u_1, u_2\}$ (u_1 and u_2 are not node labels) and λ is a new node label.



In general, we generate a binary tree of height δ, where every leaf is labeled with λ. From every leaf there is an edge back to every node on the unique path from that leaf to the root (including the leaf itself), except to the root. Moreover, from every node on the i-th level of the tree ($1 \le i \le \delta$), which is a *right* child of its parent node, there exists an edge to the node $u_i \in \Lambda$. This graph structure is easy to generate with a small hierarchical graph definition, the construction is similar to the one used in the proof of Theorem 8.3. Using this additional graph structure we can

- replace the MSO quantification $\exists X \subseteq \Lambda$ in the formula ψ in (9) by $\exists x \in \lambda$, where x is a new FO variable, and
- replace every atomic formula $y \in X$ in the formula ψ in (9) by $\exists z : x \to z \to y$.

The resulting formula is a Σ_2-MSO sentence that is true in eval(D) if and only if ψ from (9) is true in eval(D). $\square$

We will next show that for c-bounded hierarchical graph definitions the data complexity of Σ_k-MSO (resp. Π_k-MSO) goes down to the level $\mathbf{\Sigma_k^p}$ (resp. $\mathbf{\Pi_k^p}$) of the polynomial time hierarchy. Thus, the same complexity as for explicitly given input graphs is obtained. For this, we have to introduce a few definitions.

A *quantifier prefix* π is a sequence $\pi = Q_1\alpha_1 Q_2\alpha_2 \cdots Q_n\alpha_n$, where $Q_i \in \{\exists, \forall\}$ and α_i is an FO or MSO variable. A π-formula is a formula of the form $\pi : \psi$, where ψ is a quantifier-free MSO formula. We define *generalized* π-formulas inductively as follows: If $\pi = \varepsilon$, then a generalized π-formula is just a quantifier-free MSO formula. If $\pi = Q\alpha\,\pi'$, then a generalized π-formula is a positive boolean combination of formulas of the form $Q\alpha\,\psi$, where ψ is a generalized π'-formula. If π is of the form $\pi_1 \cdots \pi_k\pi'$, where π' only contains FO quantifiers and π_i is a block of existential (if i is odd) or universal (if i is even) MSO quantifiers, then a generalized π-formula is logically equivalent to a Σ_k-MSO formula. Moreover, if the quantifier prefix π has length k, then a generalized π-formula has quantifier rank k. Thus, up to logical equivalence, there are only finitely many generalized π-sentences over some fixed signature. The *generalized π-theory* of a structure $\mathcal{A}$ (over some signature S), briefly gen-π-Th($\mathcal{A}$), consists of all generalized π-sentences over the signature S that are true in $\mathcal{A}$.

The following proposition is a refinement of the well-known Feferman-Vaught decomposition theorem [14] for MSO, see [10, 32]. In fact, an analysis of the inductive proof of [10, Lemma 4.5] yields the statement of the proposition. For two structures $\mathcal{A}_1$ and $\mathcal{A}_2$ over signatures S_1 and S_2 (we may have $S_1 \cap S_2 \neq \emptyset$), respectively, we consider the disjoint union $\mathcal{A}_1 \oplus \mathcal{A}_2$ as a structure over the signature $S_1 \cup S_2$ in the natural way. We only have to require that the set of constant symbols from S_1 and S_2, respectively, are disjoint.

Proposition 9.3. *Let S_1 and S_2 be relational signatures and let*

$$\theta(X_1, \ldots, X_\ell, y_1, \ldots, y_m, z_1, \ldots, z_n)$$

be a generalized π-formula over the signature $S_1 \cup S_2$. Then there exist a boolean formula $\bigvee_{i\in I}(a_{i,1} \wedge a_{i,2})$ and generalized π-formulas $\theta_{i,1}$ (over the signature S_1) and $\theta_{i,2}$ (over the signature S_2) such that for all structures $\mathcal{A}_1$ and $\mathcal{A}_2$ over the signatures S_1 and S_2, respectively, and all $B_1, \ldots, B_\ell \subseteq \mathcal{A}_1 \oplus \mathcal{A}_2$, $b_1, \ldots, b_m \in \mathcal{A}_1$, and $c_1, \ldots, c_n \in \mathcal{A}_2$ we have:

$$\begin{aligned}(\mathcal{A}_1 \oplus \mathcal{A}_2) \models \theta(B_1, \ldots, B_\ell, b_1, \ldots, b_m, c_1, \ldots, c_n) \quad &\Leftrightarrow \\ \bigvee_{i\in I}[\mathcal{A}_1 \models \theta_{i,1}(B_1 \cap \mathcal{A}_1, \ldots, B_\ell \cap \mathcal{A}_1, b_1, \ldots, b_m) \wedge \\ \mathcal{A}_2 \models \theta_{i,2}(B_1 \cap \mathcal{A}_2, \ldots, B_\ell \cap \mathcal{A}_2, c_1, \ldots, c_n)]\end{aligned}$$

Corollary 9.4. *Let S_1 and S_2 be relational signatures and let θ be a generalized π-sentence over the signature $S_1 \cup S_2$. Then there exist a boolean formula $\bigvee_{i \in I}(a_{i,1} \wedge a_{i,2})$ and generalized π-sentences $\theta_{i,1}$ (over the signature S_1) and $\theta_{i,2}$ (over the signature S_2) such that for all structures $\mathcal{A}_1$ and $\mathcal{A}_2$ over the signatures S_1 and S_2, respectively, we have:*

$$(\mathcal{A}_1 \oplus \mathcal{A}_2) \models \theta \quad \Leftrightarrow \quad \bigvee_{i \in I} \mathcal{A}_1 \models \theta_{i,1} \wedge \mathcal{A}_2 \models \theta_{i,2}$$

The statements of the following lemma correspond to [10, Lemma 4.6, Lemma 4.7].

Lemma 9.5. *Let S be a relational signature and let θ be a generalized π-sentence over the signature S. Then there exist generalized π-sentences θ' and θ'' over the signature S such that for all structures $\mathcal{A}$ over the signature S we have:*

$$\begin{aligned} \mathit{glue}_{a,b}(\mathcal{A}) \models \theta \quad &\Leftrightarrow \quad \mathcal{A} \models \theta' \\ \mathit{rename}_f(\mathcal{A}) \models \theta \quad &\Leftrightarrow \quad \mathcal{A} \models \theta'' \end{aligned}$$

Theorem 9.6. *For every fixed Σ_k-MSO sentence (resp. Π_k-MSO sentence) φ, every fixed $c \in \mathbb{N}$, and every fixed ranked alphabet Γ, the question, whether* $\mathrm{eval}(D) \models \varphi$ *for a given c-bounded hierarchical graph definition D with terminal alphabet Γ, is in $\mathbf{\Sigma_k^p}$ (resp. $\mathbf{\Pi_k^p}$).*

Proof. It suffices to prove the statement for Σ_k-MSO sentences. As in the proof of Theorem 8.6, the basic idea is again based on Courcelle's technique for evaluating fixed MSO formulas in linear time over graphs of bounded tree width [9]. Let φ be a fixed Σ_k-MSO sentence of quantifier rank k. Let $D = (\Gamma, N, S, P)$ be a c-bounded hierarchical graph definition over the terminal alphabet Γ. As in the proof of Theorem 8.6 we first transform D into an equivalent graph straight-line program $\mathcal{S} = (X_i = t_i)_{1 \leq i \leq n}$, where $\mathrm{type}(X_i) \leq d(c)$ for every formal variable X_i. Again, $\mathrm{eval}(X_i)$ is a relational structure over some subsignature Θ_i of the fixed signature Θ, and the number of pairwise nonequivalent MSO sentences of quantifier rank at most k over the signature Θ is bounded by some constant $g(k)$.

Recall that in the proof of Theorem 8.6 we first have calculated in polynomial time the theories k-FOTh(G_i) for every definition $X_i := G_i$, where G_i is a graph. In the present situation, the direct calculation of k-MSOTh(G_i) would lead to a $\mathbf{P}^{\mathbf{\Sigma_k^p}}$-algorithm, i.e., a polynomial time algorithm with access to an oracle for $\mathbf{\Sigma_k^p}$. It is believed that $\mathbf{\Sigma_k^p}$ is a proper subset of $\mathbf{P}^{\mathbf{\Sigma_k^p}}$. The notion of generalized π-theories was introduced in order to get a $\mathbf{\Sigma_k^p}$-algorithm.

Assume that our input formula φ is a π-sentence for some quantifier prefix π. Thus, since φ is a Σ_k-MSO sentence, π is of the form $\pi_1 \cdots \pi_k \pi'$, where π' only contains FO quantifiers and π_i is a block of existential (if i is odd) or universal (if i is even) MSO quantifiers. From Corollary 9.4 and Lemma 9.5 we obtain the following statement:

There exist *monotonic* (w.r.t. set inclusion) functions $F_\oplus$, $F_{a,b}$, and F_f (where $1 \leq a, b \leq d(c)$ and $f : \{1, \ldots, a\} \to \{1, \ldots, b\}$ injective) over the set of all generalized π-theories (over the signature Θ) such that

- gen-π-Th$(G_1 \oplus G_2) = F_\oplus(\text{gen-}\pi\text{-Th}(G_1), \text{gen-}\pi\text{-Th}(G_2))$,

- gen-π-Th(glue$_{a,b}(G)$) $= F_{a,b}$(gen-π-Th(G)), and
- gen-π-Th(rename$_f(G)$) $= F_f$(gen-π-Th(G)).

Now we verify eval$(\mathcal{S}) \models \varphi$ in $\mathbf{\Sigma_k^p}$ as follows:

(1) Guess in an existential state for every formal variable X_i a set T_i of generalized π-sentences over the signature Θ_i such that

  (a) $\varphi \in T_n$,
  (b) if $t_i = X_j \oplus X_k$, then $T_i \subseteq F_\oplus(T_j, T_k)$,
  (c) if $t_i = \text{glue}_{a,b}(X_j)$, then $T_i \subseteq F_{a,b}(T_j)$, and
  (d) if $t_i = \text{rename}_f(X_j)$, then $T_i \subseteq F_f(T_j)$.

(2) For every i such that t_i is a hypergraph G_i we verify in $\mathbf{\Sigma_k^p}$ whether $G_i \models \bigwedge_{\varphi \in T_i} \varphi$.

We have to show that (i) this is indeed a $\mathbf{\Sigma_k^p}$-algorithm and (ii) it is correct. For (i), first notice that step (2) is indeed in $\mathbf{\Sigma_k^p}$: There are at most n many i such that t_i is an explicitly given hypergraph G_i; let I be the set of all these i. For every $i \in I$ we have to check whether $G_i \models \bigwedge_{\varphi \in T_i} \varphi$. Note that also $\bigwedge_{\varphi \in T_i} \varphi$ is a generalized π-sentence and hence equivalent to a Σ_k-MSO sentence ϕ_i. Now, we verify for all $i \in I$ the property $G_i \models \phi_i$ in parallel. We first guess existentially for each variable in one of the leading existential quantifier blocks of the ϕ_i a value from G_i, then we proceed with the following blocks of universal quantifiers and so on. Finally, the initial existential guessing in step (1) can be merged with the initial existential guessing in step (2). Thus, the overall algorithm is a $\mathbf{\Sigma_k^p}$ algorithm.

It remains to verify the correctness of the algorithm. If eval$(\mathcal{S}) \models \varphi$, then we obtain a successful run of the algorithm by guessing $T_i =$ gen-π-Th(eval(X_i)) for every formal variable X_i in step (1). On the other hand, if the algorithm accepts the straight-line program $\mathcal{S}$, then there exists for every formal variable X_i a set T_i of generalized π-sentences such that the inclusions in (1b)–(1d) hold, and moreover $G_i \models \bigwedge_{\varphi \in T_i} \varphi$ for every i such that $t_i = G_i$ is a hypergraph. We prove inductively, that $T_i \subseteq$ gen-π-Th(eval(X_i)) for all $1 \leq i \leq n$. If t_i is an explicitly given hypergraph, this is clear. If $t_i = X_j \oplus X_k$ for $j, k < i$, then, by induction, $T_j \subseteq$ gen-π-Th(eval(X_j)) and $T_k \subseteq$ gen-π-Th(eval(X_k)). Since $F_\oplus$ is monotonic, we obtain

$$\begin{aligned} T_i \subseteq F_\oplus(T_j, T_k) &\subseteq F_\oplus(\text{gen-}\pi\text{-Th}(\text{eval}(X_j)), \text{gen-}\pi\text{-Th}(\text{eval}(X_k))) \\ &= \text{gen-}\pi\text{-Th}(\text{eval}(X_i)). \end{aligned}$$

For the operators glue$_{a,b}$ and rename$_f$ we can argue analogously. Thus, we obtain $\varphi \in T_n \subseteq$ gen-π-Th(eval(X_n)) $=$ gen-π-Th(eval$(\mathcal{S})$), i.e., $\mathcal{S} \models \varphi$. $\square$

9.2 Combined complexity

By the next theorem, the $\mathbf{\Sigma_k^e}$ (resp. $\mathbf{\Pi_k^e}$) upper bound for Σ_k-SO (resp. Π_k-SO) generalizes from data to combined complexity.

Theorem 9.7. *For every $k \geq 1$, the following problem is $\mathbf{\Sigma_k^e}$-complete (resp. $\mathbf{\Pi_k^e}$-complete):*

INPUT: A hierarchical graph definition D and a Σ_k-SO sentence (resp. Π_k-SO sentence) φ
QUESTION: $\mathrm{eval}(D) \models \varphi$ *?*

Proof. The lower bound follows from Theorem 9.2. For the upper bound, note that in the upper bound proof of Theorem 9.2, it is not relevant that the Σ_k-SO sentence is fixed; it is only important that the number of quantifier blocks k is fixed. Thus, we can reuse the arguments from the proof of Theorem 9.2. □

Due to the following theorem, hardness for $\mathbf{\Sigma_k^e}$ (resp. $\mathbf{\Pi_k^e}$) even holds for 2-bounded hierarchical graph definitions and MSO:

Theorem 9.8. *For every $k \geq 1$ and every $c \geq 2$, the following problem is $\mathbf{\Sigma_k^e}$-complete (resp. $\mathbf{\Pi_k^e}$-complete):*

INPUT: A c-bounded hierarchical graph definition D and a Σ_k-MSO sentence (resp. Π_k-MSO sentence) φ
QUESTION: $\mathrm{eval}(D) \models \varphi$ *?*

Proof. We use a construction from [8, 31]. For k odd, we prove the theorem for $\mathbf{\Sigma_k^e}$, for k even, we prove the theorem for $\mathbf{\Pi_k^e}$. We only consider the case that k is odd. Let M be an alternating Turing-machine with a $\mathbf{\Sigma_k^e}$-complete membership problem. Let $\Gamma = \{a_1, \ldots, a_m\}$ be the tape alphabet with $a_m = \Box$, $Q = Q_\exists \uplus Q_\forall \uplus F$ be the state set, and $q_0 \in Q_\exists$ be the initial state. Let $p(n)$ be a polynomial such that when M is started on an input word of length n, the running time is bounded by $2^{p(n)}$. W.l.o.g. we may assume that on every computation path, M makes precisely $k-1$ alternations, We may also assume that a final state from F can be only reached from a state in $Q_\exists$, i.e., there does not exist a transition from a state in $Q_\forall$ to a state in F.

We will consider structures of the form $([0, N], S)$ where $N \in \mathbb{N}$, $[0, N] = \{0, \ldots, N\}$, and S is the successor function on the interval $[0, N]$. When viewed as a graph, it is easy to generate the structure $([0, 2^n - 1], S)$ with a 2-bounded hierarchical graph definition of size $\mathcal{O}(n)$: take a context-free grammar in Chomsky normal form of size $\mathcal{O}(n)$ that generates only the word a^{2^n} and view it as a hierarchical graph definition. For an input word w for M we will construct a formula ψ_w such that $([0, 2^{p(|w|)} - 1], S) \models \psi_w$ if and only if w is accepted by M. In a first step, we will consider the richer structure $([0, 2^{p(|w|)} - 1], +)$, where $+$ denotes the addition of natural numbers on the interval $[0, 2^{p(|w|)} - 1]$. In a second step, we will show how to eliminate $+$ using the successor function S.

Let $[0, N]$ be an initial segment of the natural numbers, where $N \geq |Q| - 1$. We may identify the state set Q with the numbers $\{0, \ldots, |Q| - 1\}$. An instantaneous description of M of length N will be encoded by a tuple $\overline{A} = (A_1, \ldots, A_{m+2})$ with $A_i \subseteq [0, N]$, where A_i $(1 \leq i \leq m)$ is

the set of all those $k \in [0, N]$ such that tape cell k contains the tape symbol a_i, $A_{m+1} = \{k\}$ with k the current position of the tape head, and $A_{m+2} = \{q\}$ with q the current state. For subsets $P_1, P_2 \subseteq Q$ and two tuples $\overline{A}, \overline{B} \in (2^{[0,N]})^{m+2}$ we write $\overline{A} \Rightarrow^N_{P_1,P_2} \overline{B}$ if and only if

- $\overline{A}$ and $\overline{B}$ describe instantaneous descriptions of M,
- $\overline{B}$ can be obtained from $\overline{A}$ within at most N moves of M, where no tape position greater than N is reached and only transitions out of states from P_1 are allowed, and
- $B_{m+2} = \{q\}$ with $q \in P_2$, i.e., we end in a state from P_2.

Using the construction from [31], it is possible to construct a *fixed* Σ_1-MSO formula $\psi_{P_1,P_2}(\overline{X}, \overline{Y})$ such that for every $N \geq |Q| - 1$:

$$([0, N], +) \models \psi_{P_1,P_2}(\overline{A}, \overline{B}) \quad \Leftrightarrow \quad \overline{A} \Rightarrow^N_{P_1,P_2} \overline{B}.$$

Now construct formulas η_i $(1 \leq i < k)$ as follows:

$$\begin{aligned}
\eta_1(\overline{X}) &\equiv \exists \overline{Y} : \psi_{Q_\exists,F}(\overline{X}, \overline{Y}) \\
\eta_{i+1}(\overline{X}) &\equiv \forall \overline{Y} : \psi_{Q_\forall,Q_\exists}(\overline{X}, \overline{Y}) \Rightarrow \eta_i(\overline{Y}) \quad \text{if } i \text{ is odd} \\
\eta_{i+1}(\overline{X}) &\equiv \exists \overline{Y} : \psi_{Q_\exists,Q_\forall}(\overline{X}, \overline{Y}) \wedge \eta_i(\overline{Y}) \quad \text{if } i \text{ is even}
\end{aligned}$$

Then an input word $w = b_0 b_1 \cdots b_{n-1}$ with $b_i \in \Gamma \setminus \{\Box\}$ is accepted by the machine M if and only if the sentence

$$\exists \overline{X} \left\{ \begin{array}{l} \bigwedge_{i=1}^{m-1} X_i = \{k \mid b_k = a_i\} \ \wedge \ X_m = [n, 2^{p(n)} - 1] \ \wedge \\ X_{m+1} = \{0\} \ \wedge \ X_{m+2} = \{q_0\} \ \wedge \ \eta_k(\overline{X}) \end{array} \right\}$$

is true in $([0, 2^{p(n)} - 1], +)$ (recall that $a_m = \Box$, thus, $X_m = [n, 2^{p(n)} - 1]$ expresses that the tape positions $n, \ldots, 2^{p(n)} - 1$ contain the blank symbol). It is easy to write down an equivalent sentence of size $\mathcal{O}(n)$ in the language of addition. Moreover, if we shift MSO quantifiers to the front, the resulting sentence becomes a Σ_k-MSO sentence.

It remains to eliminate $+$ using the successor function S on the interval $[0, 2^{p(n)} - 1]$. For this, we will show that addition on numbers in the range $[0, 2^{p(n)} - 1]$ can be expressed using an FO formula of size $\mathcal{O}(p(n)^2)$ over the successor function S. First of all, using a standard trick we can construct formulas $d_i(x, y)$ $(0 \leq i < p(n))$ of size $\mathcal{O}(i)$ such that $([0, 2^{p(n)} - 1], S) \models d_i(a, b)$ if and only if $b - a = 2^i$:

$$\begin{aligned}
d_0(x, y) &\equiv y = S(x) \\
d_{i+1}(x, y) &\equiv \exists z \, \forall u \, \forall v : ((u = x \wedge v = z) \vee (u = z \wedge v = y)) \Rightarrow d_i(u, v)
\end{aligned}$$

Σ_k-FO	explicit [4, 13, 24, 41]	apex	c-bounded	general
data	$\mathbf{\Sigma_k^{log}}$	$\mathbf{NL}$	$\mathbf{NL} \cdots \mathbf{P}$	$\mathbf{NL}(k=1)$ $\mathbf{\Sigma_{k-1}^{p}}$ $(k>1)$
combined	$\mathbf{\Sigma_k^{p}}$			

Table 1: FO over hierarchical graphs

Σ_k-MSO	explicit [13, 33, 41]	c-bounded	general
data	$\mathbf{\Sigma_k^{p}}$		$\mathbf{\Sigma_k^{e}}$
combined			

Table 2: MSO over hierarchical graphs

Next, for bits $x_i \in \{0,1\}$ $(0 \leq i < p(n))$ let $n(x_0, \ldots, x_{p(n)-1}) = \sum_{i=0}^{p(n)-1} x_i \cdot 2^i$. Using the carry look ahead algorithm for addition of natural numbers, it is straightforward to write down a formula $\mathrm{plus}((x_i)_{0\leq i<p(n)}, (y_i)_{0\leq i<p(n)}, (z_i)_{0\leq i<p(n)})$ in $3p(n)$ variables such that

$$([0, 2^{p(n)} - 1], S) \models \mathrm{plus}((x_i)_{0\leq i<p(n)}, (y_i)_{0\leq i<p(n)}, (z_i)_{0\leq i<p(n)})$$

if and only if $x_i, y_i, z_i \in \{0,1\}$ and $n(x_0, \ldots, x_{p(n)-1}) + n(y_0, \ldots, y_{p(n)-1}) = n(z_0, \ldots, z_{p(n)-1})$. The size of this formula is $\mathcal{O}(p(n)^2)$. Let $\mathrm{bin}((x_i)_{0\leq i<p(n)}, x)$ be the formula

$$\exists (u_i)_{0\leq i\leq p(n)} \left\{ \begin{array}{l} u_0 = 0 \ \wedge \ u_{p(n)} = x \ \wedge \\ \bigwedge_{i=0}^{p(n)-1} ((x_i = 0 \Rightarrow u_i = u_{i+1}) \ \wedge \ (x_i = 1 \Rightarrow d_i(u_i, u_{i+1}))) \end{array} \right\}.$$

Then $x + y = z$ for $x, y, z \in [0, 2^{p(n)} - 1]$ if and only if

$$\begin{array}{l} \exists (x_i)_{0\leq i<p(n)} \exists (y_i)_{0\leq i<p(n)} \exists (z_i)_{0\leq i<p(n)} : \\ \quad \mathrm{plus}((x_i)_{0\leq i<p(n)}, (y_i)_{0\leq i<p(n)}, (z_i)_{0\leq i<p(n)}) \wedge \\ \quad \mathrm{bin}((x_i)_{0\leq i<p(n)}, x) \ \wedge \ \mathrm{bin}((y_i)_{0\leq i<p(n)}, y) \ \wedge \ \mathrm{bin}((z_i)_{0\leq i<p(n)}, z), \end{array}$$

which is a formula of size $\mathcal{O}(p(n)^2)$. $\square$

10 Conclusion and open problems

In Table 1 and 2 our complexity results for hierarchically defined graphs together with the known results for explicitly given input graphs are collected. The only open problem that remains from these tables is the precise complexity of the model-checking problem for FO and c-bounded hierarchical graph definitions. There is a gap between **NL** and **P** for this problem. Currently, we are investigating the complexity of parity games and various fixed point logics over hierarchically defined graphs.

References

[1] R. Alur and M. Yannakakis. Model checking of hierarchical state machines. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 23(3):273–303, 2001.

[2] C. Àlvarez and B. Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107(1):3–30, 1993.

[3] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[4] D. A. M. Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41:274–306, 1990.

[5] M. Benedikt, P. Godefroid, and T. W. Reps. Model checking of unrestricted hierarchical state machines. In F. Orejas, P. G. Spirakis, and J. van Leeuwen, editors, *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP 2001), Crete (Greece)*, number 2076 in Lecture Notes in Computer Science, pages 652–666. Springer, 2001.

[6] B. Borchert and A. Lozano. Succinct circuit representations and leaf language classes are basically the same concept. *Information Processing Letters*, 59(4):211–215, 1996.

[7] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, 1981.

[8] K. J. Compton and C. W. Henson. A uniform method for proving lower bounds on the computational complexity of logical theories. *Annals of Pure and Applied Logic*, 48:1–79, 1990.

[9] B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 193–242. Elsevier Science Publishers, 1990.

[10] B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[11] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer, 1991.

[12] J. Engelfriet. Context-free graph grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, pages 125–213. Springer, 1997.

[13] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R. M. Karp, editor, *Complexity and Computation*, volume 7 of *SIAM-AMS Proceedings*, pages 43–73, 1974.

[14] S. Feferman and R. L. Vaught. The first order properties of products of algebraic systems. *Fundamenta Mathematicae*, 47:57–103, 1959.

[15] J. Feigenbaum, S. Kannan, M. Y. Vardi, and M. Viswanathan. The complexity of problems on graphs represented as OBDDs. *Chicago Journal of Theoretical Computer Science*, 1999.

[16] M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *Journal of the Association for Computing Machinery*, 48(6):1184–1206, 2001.

[17] M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science (LICS'2002)*, pages 215–224. IEEE Computer Society Press, 2002.

[18] H. Gaifman. On local and nonlocal properties. In J. Stern, editor, *Logic Colloquium '81*, pages 105–135. North Holland, 1982.

[19] H. Galperin and A. Wigderson. Succinct representations of graphs. *Information and Control*, 56(3):183–198, 1983.

[20] G. Gottlob, P. G. Kolaitis, and T. Schwentick. Existential second-order logic over graphs: Charting the tractability frontier. *Journal of the Association for Computing Machinery*, 51(2):312–362, 2004.

[21] G. Gottlob, N. Leone, and H. Veith. Succinctness as a source of complexity in logical formalisms. *Annals of Pure and Applied Logic*, 97(1–3):231–260, 1999.

[22] A. Habel. *Hyperedge Replacement: Grammars and Languages.* Number 643 in Lecture Notes in Computer Science. Springer, 1992.

[23] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.

[24] N. Immerman. Expressibility and parallel complexity. *SIAM Journal on Computing*, 18(3):625–638, 1989.

[25] N. Immerman. *Descriptive Complexity.* Springer, 1999.

[26] R. E. Ladner. Application of model theoretic games to discrete linear orders and finite automata. *Information and Computation*, 33(4):281–303, 1977.

[27] T. Lengauer. Hierarchical planarity testing algorithms. *Journal of the Association for Computing Machinery*, 36(3):474–509, 1989.

[28] T. Lengauer and K. W. Wagner. The correlation between the complexities of the non-hierarchical and hierarchical versions of graph problems. *Journal of Computer and System Sciences*, 44:63–93, 1992.

[29] T. Lengauer and E. Wanke. Efficient solution of connectivity problems on hierarchically defined graph. *SIAM Journal on Computing*, 17(6):1063–1080, 1988.

[30] L. Libkin. *Elements of Finite Model Theory.* Springer, 2004.

[31] J. F. Lynch. Complexity classes and theories of finite models. *Mathematical Systems Theory*, 15:127–144, 1982.

[32] J. A. Makowsky. Algorithmic aspects of the Feferman-Vaught theorem. *Annals of Pure and Applied Logic*, 126(1–3):159–213, 2004.

[33] J. A. Makowsky and Y. B. Pnueli. Arity and alternation in second-order logic. *Annals of Pure and Applied Logic*, 78(1–3):189–202, 1996.

[34] M. V. Marathe, H. B. Hunt III, and S. S. Ravi. The complexity of approximation PSPACE-complete problems for hierarchical specifications. *Nordic Journal of Computing*, 1(3):275–316, 1994.

[35] M. V. Marathe, H. B. Hunt III, R. E. Stearns, and V. Radhakrishnan. Approximation algorithms for PSPACE-hard hierarchically and periodically specified problems. *SIAM Journal on Computing*, 27(5):1237–1261, 1998.

[36] M. V. Marathe, V. Radhakrishnan, H. B. Hunt III, and S. S. Ravi. Hierarchically specified unit disk graphs. *Theoretical Computer Science*, 174(1–2):23–65, 1997.

[37] C. H. Papadimitriou. *Computational Complexity.* Addison Wesley, 1994.

[38] C. H. Papadimitriou and M. Yannakakis. A note on succinct representations of graphs. *Information and Control*, 71(3):181–185, 1986.

[39] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[40] C. Stirling. *Modal and Temporal Properties of Processes.* Springer, 2001.

[41] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[42] H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity.* Birkhäuser, Boston, Basel, Berlin, 1994.

[43] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

[44] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, volume III*, pages 389–455. Springer, 1997.

[45] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing (STOC 1982)*, pages 137–146. ACM Press, 1982.

[46] M. Y. Vardi. On the complexity of bounded-variable queries. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1995)*, pages 266–276. ACM Press, 1995.

[47] H. Veith. Languages represented by Boolean formulas. *Information Processing Letters*, 63(5):251–256, 1997.

[48] H. Veith. How to encode a logical structure by an OBDD. In *Proceedings of the 13th Annual IEEE Conference on Computational Complexity*, pages 122–131. IEEE Computer Society, 1998.

[49] H. Veith. Succinct representation, leaf languages, and projection reductions. *Information and Computation*, 142(2):207–236, 1998.

[50] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):23–33, 1976.