Model-Checking Hierarchical Structures

Markus Lohrey

FMI, University of Stuttgart, Germany
lohrey@fmi.uni-stuttgart.de

Abstract

Hierarchical graph definitions allow a modular description of structures using modules for the specification of repeated substructures. Beside this modularity, hierarchical graph definitions allow to specify structures of exponential size using polynomial size descriptions. In many cases, this succinctness increases the computational complexity of decision problems when input structures are defined hierarchically. 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 structures is investigated. It is shown that in general these model-checking problems are exponentially harder than their non-hierarchical counterparts, where the input structures 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.

Key words: model-checking, hierarchical structures, logic in computer science, complexity

1 Introduction

Hierarchical graph definitions specify a structure via modules, where every module is a graph that may refer to modules on a smaller hierarchical level. In this way, large structures can be represented in a modular and succinct way. Hierarchical graph definitions were introduced in [30] in the context of VLSI design. Formally, hierarchical graph definitions can be seen as hyperedge replacement graph grammars [12,23] that generate precisely one graph. In computer science, hierarchical graph definition can be used as a suitable abstract formalism whenever systems with repeated (or shared) substructures appear. A typical example are large software systems with shared modules/objects.

In this paper we consider the model-checking problem for hierarchically defined input structures. Model-checking is a computational problem of central importance in many fields of computer science, like for instance verification or database theory. It is asked whether a given logical formula from some prespecified 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. The logics we consider are first-order logic (FO), monadic second-order logic (MSO), and second-order logic (SO). 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.

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 [11,26,31,45] and the handbook article [47] 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 give a few references: [13,16,17,21,22,33,35,48,49]. 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 [28–30,36–38], there is no systematic investigation of model-checking problems for hierarchically defined structures so far (one should notice that all the algorithmic problems mentioned above can be formulated in SO). The only exception is the work from [1,2,39], where the complexity of temporal logics (LTL, CTL, CTL*) over hierarchically defined strings [39] and hierarchical state machines [1,2] 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 structures leads to a better understanding of hierarchical structures in a broad sense. Our investigation of model-checking problems for hierarchically defined structures will follow a methodology introduced by Vardi [48]. 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 $\mathcal{U} \in \mathcal{C}$ whether $\mathcal{U} \models \varphi$; thus, only the structure belongs to the input (data complexity or structure complexity). (ii) One may fix a structure $\mathcal{U}$ from the class $\mathcal{C}$ and consider the complexity of verifying for a given sentence φ from $\mathcal{L}$, whether $\mathcal{U} \models \varphi$; 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 structures, expression complexity will not lead to new results. Having a fixed hierarchically defined structure makes no difference to having a fixed explicitly given structure. Thus, we will only consider data and combined complexity for hierarchically defined structures.

After introducing the necessary concepts in Section 3–6, we study model-checking problems for FO over hierarchically defined structures in Section 7. Section 7.1 deals with data complexity whereas in Section 7.2, combined complexity is briefly considered. Section 8 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 structures. 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 Section 6 together with the known results for model-checking explicitly given input structures (see Section 4 and 5.1 for the relevant definitions). As can be seen from these tables, there is a tight correspondence between the bounded quantifier-alternation fragments of FO/MSO and the polynomial/exponential time hierarchy. Due to the common game theoretical foundation of these concepts, this is not really surprising.

A short version of this paper appeared in [32]. In a subsequent conference paper [20], the research program from [32] was extended to parity games and various fixpoint logics.

2 Related work

Specific algorithmic problems (e.g. reachability, planarity, circuit-value, 3-colorability) on hierarchically defined structures are studied in [28–30,36–38]. A concept related to hierarchical graph definitions are hierarchical state machines [2,1], 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 hierarchical graph definitions. The work of Alur et al [1,2] studies 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 [6,19,41,52], boolean formulas [22,50], and binary decision diagrams [15,51]. 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 [29].

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 with $A_1 \cap A_2 = \emptyset$ and $A = A_1 \cup A_2$ we sometimes write $A = A_1 \uplus A_2$ in order to emphasize the fact that A is the disjoint union of A_1 and A_2. 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 $f \cup g : A \uplus B \rightarrow C \cup D$ by $(f \cup g)(a) = f(a)$ for $a \in A$ and $(f \cup g)(b) = g(b)$ for $b \in B$.

A *signature* $\mathcal{R}$ is a finite set consisting of relational symbols r_i $(i \in I)$ and constant symbols c_j $(j \in J)$. Each relational symbol r_i has an associated arity α_i. A (finite) *structure over the signature* $\mathcal{R}$ is a tuple $\mathcal{U} = (U, (R_i)_{i \in I}, (u_j)_{j \in J})$, where U is a finite set (the universe of $\mathcal{U}$), $R_i \subseteq U^{\alpha_i}$ is the relation associated with the relational symbol r_i, and $u_j \in U$ is the constant associated with the constant symbol c_j. If the structure $\mathcal{U}$ is clear from the context, we will identify R_i (respectively u_j) with the relational symbol r_i (respectively the constant symbol c_j). Sometimes, when we want to refer to the universe U, we will refer to $\mathcal{U}$ itself. For instance, we will write $u \in \mathcal{U}$ instead of $u \in U$, or $f : \{1, \ldots, n\} \rightarrow \mathcal{U}$ if f is a function from $\{1, \ldots, n\}$ to U. The size $|\mathcal{U}|$ of $\mathcal{U}$ is $|U| + \sum_{i \in I} \alpha_i \cdot |R|$. As usual, a constant u may be replaced by the unary relation $\{u\}$. Thus, in the following, we will only consider signatures without constant symbols, except when we explicitly introduce constants. Let $\mathcal{R} = \{r_i \mid i \in I\}$ be such a signature and let $\mathcal{U} = (U, (R_i)_{i \in I})$ be a structure over $\mathcal{R}$. For an equivalence relation $\equiv$ on U we define the quotient $\mathcal{U}/_\equiv = ([U]_\equiv, (R_i/_\equiv)_{i \in I})$, where $R_i/_\equiv = \{(\pi_\equiv(v_1), \ldots, \pi_\equiv(v_{\alpha_i})) \mid (v_1, \ldots, v_{\alpha_i}) \in R_i\}$. For two structures $\mathcal{U}_1 = (U_1, (R_{i,1})_{i \in I})$ and $\mathcal{U}_1 = (U_2, (R_{i,2})_{i \in I})$ over the same signature $\mathcal{R}$ and with disjoint universes U_1 and U_2, respectively, we define the disjoint union $\mathcal{U}_1 \oplus \mathcal{U}_2 = (U_1 \uplus U_1, (R_{i,1} \uplus R_{i,2})_{i \in I})$. For $n \geq 0$, an n-pointed structure is a pair $(\mathcal{U}, \tau)$, where $\mathcal{U}$ is a structure and $\tau : \{1, \ldots, n\} \rightarrow \mathcal{U}$ is injective. The elements in $\mathrm{ran}(\tau)$ (respectively $\mathcal{U} \setminus \mathrm{ran}(\tau)$) are called *contact nodes* (respectively *internal nodes*). The node $\tau(i)$ is called the *i-th contact node*.

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, where V_G^* is the set of finite strings over V_G, (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 $1 \le i \le n$ 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 background in complexity theory [40]. 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 $f : \{0,1\}^* \to \{0,1\}^*$ is *computable in nondeterministic logspace* [3] if there exists a nondeterministic Turing machine M for which the working space is bounded by $\mathcal{O}(\log(n))$ and such that for every input $x \in \{0,1\}^*$: on every computation path, either M rejects on that path or writes $f(x)$ on the output tape and then terminates. As usual, the space on the output tape does not belong to the working space. Note that since the running time of M must be bounded polynomially, there must exist a constant c such that $|f(x)| \le |x|^c$ for all $x \in \{0,1\}^*$. 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 $x \in \{0,1\}^*$, $x \in A$ if and only if $f(x) \in B$. It is not hard to see that if A is NL-reducible to $B \in \mathsf{NL}$, then also $A \in \mathsf{NL}$. 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. In fact, all complexity classes occurring in this article are closed under L/NL-reductions.

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 [24,46], 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 NL.

The levels of the *polynomial time hierarchy* are defined as follows: Let $k \geq 1$. Then $\mathbf{\Sigma_k^p}$ (respectively $\mathbf{\Pi_k^p}$) 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 $Q_\exists$ (respectively $Q_\forall$). The polynomial time hierarchy is $\mathsf{PH} = \bigcup_{k\geq 1} \mathbf{\Sigma_k^p}$. 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 $\mathbf{\Sigma_k^e}$ (respectively $\mathbf{\Pi_k^e}$) of the (weak) EXP *time hierarchy* $\mathsf{EH} = \bigcup_{k\geq 1} \mathbf{\Sigma_k^e}$. If we replace the polynomial time bound by a logarithmic time bound $\mathcal{O}(\log(n))$, then we obtain the levels $\mathbf{\Sigma_k^{\log}}$ (respectively $\mathbf{\Pi_k^{\log}}$) of the *logtime hierarchy* $\mathsf{LH} = \bigcup_{k\geq 1} \mathbf{\Sigma_k^{\log}}$, which is contained in 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 AC^0.

5 Hierarchical formalisms

In this section, we will consider two hierarchical formalisms for the succinct specification of large relational structures: *hierarchical graph definitions* and *straight-line programs.*

5.1 Hierarchical graph definitions

A *hierarchical graph definition* is a tuple $D = (\mathcal{R}, N, S, P)$ such that:

(1) $\mathcal{R}$ is a signature.
(2) N is a finite set of *nonterminals* (or reference names). Every $A \in N$ has a *rank* $\text{rank}(A) \in \mathbb{N}$.
(3) $S \in N$ is the initial nonterminal, where $\text{rank}(S) = 0$.

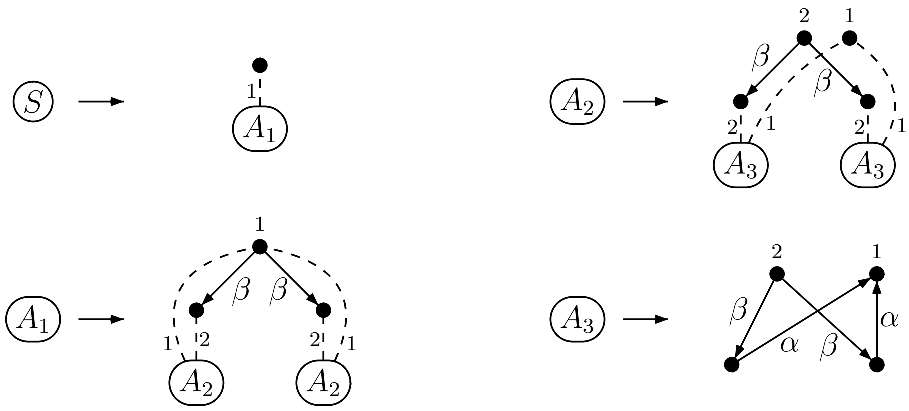


Fig. 1. The productions of the hierarchical graph definition from Example 1

(4) P is a set of productions. For every $A \in N$, P contains exactly one production $A \to (\mathcal{U}, \tau, E)$, where $(\mathcal{U}, \tau)$ is a $\mathrm{rank}(A)$-pointed structure over the signature $\mathcal{R}$ and $E \subseteq \{(B, \sigma) \mid B \in N, \sigma : \{1, \ldots, \mathrm{rank}(B)\} \to \mathcal{U}$ is injective$\}$ (the set of *references*).
(5) 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 $A \to (\mathcal{U}, \tau, E)$, E contains a reference of the form (B, σ). Then we require that E_D is acyclic.

By (5), the transitive closure $\succ_D$ of the relation E_D is a partial order, we call it the *hierarchical order.* In (4), a pair (B, σ) with $B \in N$ and $\sigma : \{1, \ldots, \mathrm{rank}(B)\} \to \mathcal{U}$ injective is also called a *B-labeled reference.* The size $|D|$ of D is defined by $\sum_{(A \to (\mathcal{U}, \tau, E)) \in P} |\mathcal{U}| + |E|$.

In the lower bound proofs in the rest of the paper, we will only use relational structures where all relations have arity one or two. We will view and visualize such a structure as a directed graph, where nodes are labeled with unary relational symbols and edges are labeled with binary relational symbols. Note that our definition allows several node labels for a single node. In pictures, a reference (A, σ) will be drawn as a big circle with inner label A. This circle is connected via dashed lines with the nodes $\sigma(i)$ for $1 \leq i \leq \mathrm{rank}(A)$, where the connection to $\sigma(i)$ is labeled with i. These dashed lines are also called *tentacles.* If $G = (\mathcal{U}, \tau)$ is an n-pointed relational structure, then we label the contact node $\tau(i)$ with i. In order to distinguish this label i better from node labels that correspond to unary relational symbols, we will use a smaller font for the label i.

Example 1 *Let $D = (\mathcal{R}, N, S, P)$ be the hierarchical graph definition, where the signature $\mathcal{R}$ contains two binary relational symbols α and β, and $N = \{S, A_1, A_2, A_3\}$ with* $\mathrm{rank}(S) = 0$, $\mathrm{rank}(A_1) = 1$, *and* $\mathrm{rank}(A_2) = \mathrm{rank}(A_3) = 2$. *The set P of productions is shown in Figure 1.*

Let us now define the structure $\mathrm{eval}(D)$, which results from unfolding a hierarchical graph definition $D = (\mathcal{R}, N, S, P)$. For every $A \in N$ we define a

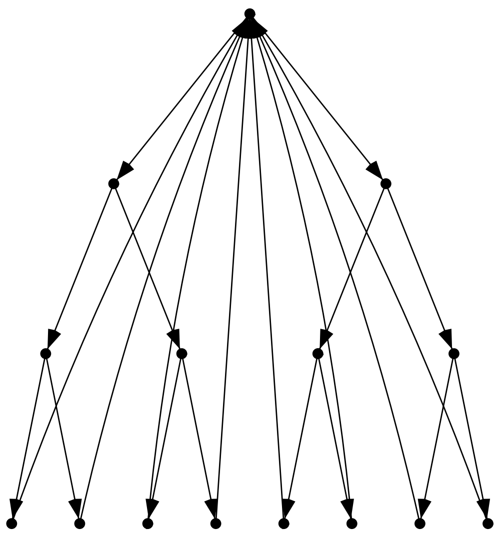


Fig. 2. The graph eval(D) for the hierarchical graph definition from Example 1

rank(A)-pointed structure eval(A) over the signature $\mathcal{R}$. The idea is to take the structure $\mathcal{U}$ from the unique production $(A \to (\mathcal{U}, \tau, E)) \in P$ and to replace every reference $(B, \sigma) \in E$ by the rank(B)-pointed structure eval$(B) = (\mathcal{U}', \tau')$. Finally, we identify the node $\sigma(i)$ with the contact node $\tau'(i)$ for every $1 \leq i \leq \text{rank}(B)$. Formally, assume that $A \to (\mathcal{U}, \tau, E)$ is the unique production for A in P. Let $E = \{(A_i, \sigma_i) \mid 1 \leq i \leq n\}$. Of course we may have $A_i = A_j$ for $i \neq j$. Assume that $\text{eval}(A_i) = (\mathcal{U}_i, \tau_i)$ is already defined. Then

$$\text{eval}(A) = ((\mathcal{U} \oplus \mathcal{U}_1 \oplus \cdots \oplus \mathcal{U}_n)/_{\equiv}, \pi_{\equiv} \circ \tau),$$

where $\equiv$ is the smallest equivalence relation on the universe of $\mathcal{U} \oplus \mathcal{U}_1 \oplus \cdots \oplus \mathcal{U}_n$, which contains $\{(\sigma_i(j), \tau_i(j)) \mid 1 \leq i \leq n, 1 \leq j \leq \text{rank}(A_i)\}$. Finally, we define eval(D) = eval(S); since rank$(S) = 0$ it can be viewed as an ordinary (0-pointed) structure. It is not hard to see that $|\text{eval}(D)| \in 2^{\mathcal{O}(|D|)}$. Thus, D can be seen as a compressed representation of the structure eval(D). As a consequence, computational problems may become more difficult, if input structures are represented by a hierarchical graph definition.

Example 1 (continued). *Then graph* eval(D) *for the hierarchical graph definition D from Example 1 is shown in Figure 2. 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. Figure 6 shows the 2-pointed structure* eval(A_2)*. Two intermediate structures that occur during the unfolding of D are shown in Figure 3.*

Definition 2 *We say that the hierarchical graph definition $D = (\mathcal{R}, N, S, P)$ is c-*bounded *if* rank$(A) \leq c$ *for every $A \in N$ and moreover for every production $(A \to (\mathcal{U}, \tau, E)) \in P$ we have $|E| \leq c$. We say that D is* apex*, if for every production $(A \to (\mathcal{U}, \tau, E)) \in P$ and every reference $(B, \sigma) \in E$ we*

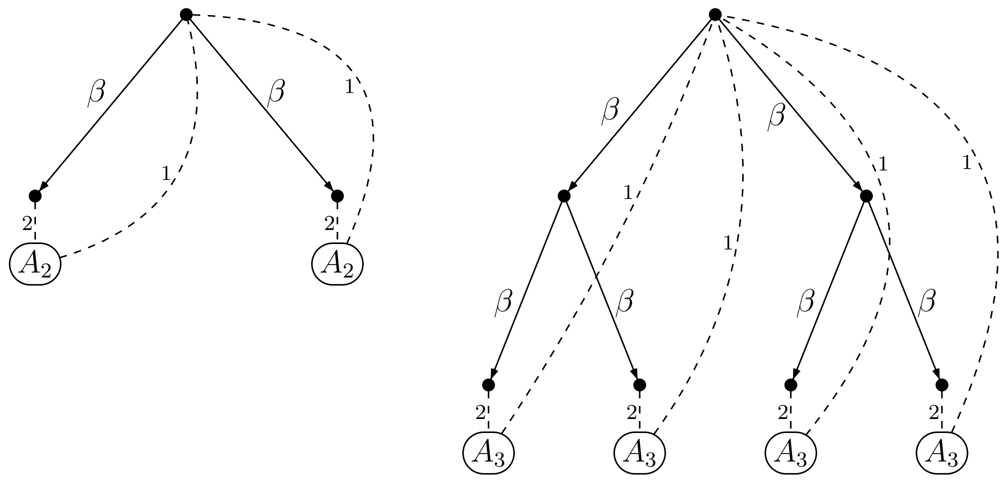


Fig. 3. Two intermediate structures that arise when unfolding D from Example 1

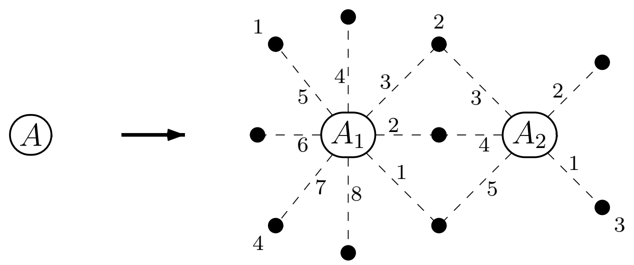


Fig. 4. A typical production for a hierarchical graph definition in Chomsky normal form

have $\mathrm{ran}(\sigma) \cap \mathrm{ran}(\tau) = \emptyset$*. Thus, contact nodes of a right-hand side cannot be accessed by references.*

Apex hierarchical graph definitions are called 1-level restricted in [36]. The hierarchical graph definition D from Example 1 is 2-bounded (but not 1-bounded) and not apex.

Definition 3 *A hierarchical graph definition* $D = (\mathcal{R}, N, S, P)$ *is in* Chomsky normal form *if for every production* $(A \to (\mathcal{U}, \tau, E)) \in P$*, either*

- $E = \emptyset$*, or*
- *all relations of* $\mathcal{U}$ *are empty (i.e.,* $\mathcal{U}$ *is a naked set),* $|E| = 2$*, and* $\mathcal{U} = \bigcup_{(B,\sigma)\in E} \mathrm{ran}(\sigma)$.

A typical production of the second type is shown in Figure 4, where $\mathrm{rank}(A) = 4$.

Remark 4 *For a given hierarchical graph definition* $D = (\mathcal{R}, N, S, P)$ *one can construct a hierarchical graph definition* D' *in Chomsky normal form such that* $\mathrm{eval}(D) = \mathrm{eval}(D')$*. Moreover, this construction can be carried out by*

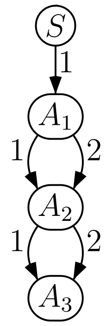


Fig. 5. The dag dag(G) for the hierarchical graph definition D from Example 1

a logspace bounded machine and is similar to the corresponding construction for context-free string grammars: By introducing fresh nonterminals for node-tuples in right-hand sides that belong to a relation of $\mathcal{R}$, one can enforce that for every production $(A \to (\mathcal{U}, \tau, E)) \in P$, either $E = \emptyset$ or all relations of $\mathcal{U}$ are empty and $|E| \geq 1$. In the latter case, if $\mathcal{U}$ contains nodes which are not accessed by a tentacle, then we access these nodes by a fresh dummy nonterminal. This ensures that $\mathcal{U} = \bigcup_{(B,\sigma)\in E} \mathrm{ran}(\sigma)$. It remains to enforce $|E| = 2$. Productions with $|E| = 1$ can be eliminated by unfolding the right-hand side until the number of nonterminals is either zero or at least two. Finally, productions with $|E| > 2$ have to be split into several productions in the same way as for context-free string grammars.

Definition 5 *With a hierarchical graph definition $D = (\mathcal{R}, N, S, P)$ we associate an ordered dag $dag(D) = (N, \gamma, S)$, where the child-function γ is defined as follows: Let $A \to (\mathcal{U}, \tau, E)$ be the unique production with left-hand side $A \in N$ and let $(A_1, \sigma_1), \ldots, (A_m \sigma_n)$ be an enumeration of the references in E (this enumeration is somehow given by the input encoding of D). Then $\gamma(A) = A_1 \cdots A_n$.*

For instance, dag(D) for the hierarchical graph definition D from Example 1 is shown in Figure 5, where an edge from nonterminal B to C with label i means that C is the i-th symbol in $\gamma_{\mathrm{dag}(G)}(B)$.

Remark 6 *We list some simple algorithmic properties of hierarchical graph definitions that are useful for the further considerations.*

(1) A node of eval(D) *can be uniquely represented by a pair (p, v) such that (i) p is a root-path in $dag(D)$ with target node $A = \tau_{dag(D)}(p)$ and (ii) $A \to (\mathcal{U}, \tau, E)$ is the unique production with left-hand side A, where $v \in \mathcal{U} \setminus \mathrm{ran}(\tau)$ 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*

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

$\mathcal{O}(\log(|D|))$, *whether* (p, v) *represents indeed a node of* $\mathrm{eval}(D)$.

(2) Given nodes $u_i = (p_i, v_i)$ *for* $1 \leq i \leq n$ *and a relational symbol* $r \in \mathcal{R}$ *of arity* n, *we can verify in time* $\mathcal{O}(|D|)$ *(or alternatively in space* $\mathcal{O}(\log(|D|))$*), whether* $(u_1, \ldots, u_n) \in r$ *in the structure* $\mathrm{eval}(D)$.

Also the following simple statement will be useful later:

Lemma 7 *For a given hierarchical graph definition* $D = (\mathcal{R}, N, S, P)$ *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* $\alpha \notin \mathcal{R}$ *is a new unary relational symbol.*

PROOF. Assume that $p = i_1 i_2 \cdots i_n$ ($i_k \in \mathbb{N}$ for $1 \leq k \leq n$) 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 $A_k \to (\mathcal{U}_k, \tau_k, E_k)$ be the unique production for A_k in D. If $0 \leq k < n$, then we introduce for A'_k the production $A'_k \to (\mathcal{U}_k, \tau_k, E'_k)$, where E'_k results from E_k by replacing the i_{k+1}-th reference (A_{k+1}, σ) (in the order on the references, given by the input encoding of D) of E_k by (A'_{k+1}, σ). Finally, we add the rule $A'_n \to (\mathcal{U}'_n, \tau_n, E_n)$, where $\mathcal{U}'_n$ results from $\mathcal{U}_n$ by adding the new label α to the internal node $v \in \mathcal{U}_n \setminus \mathrm{ran}(\tau_n)$. The resulting hierarchical graph definition D' has the property from the lemma. Clearly, the construction can be done using logarithmic working space. $\square$

5.2 Straight-line programs

Hierarchical graph definitions are our favorite formalism for the succinct specification of large structures. For some upper bound proofs however, *straight-line programs* are more convenient than hierarchical graph definitions. A (graph) straight-line program is a sequence of operations on n-pointed structures. These operations allow the disjoint union, the rearrangement, and the gluing of its contact nodes, see also [9,10]. For the formal definition, let us fix a signature $\mathcal{R}$.

Let $G_i = (\mathcal{U}_i, \tau_i)$ be an n_i-pointed structure ($i \in \{1, 2\}$) over the signature $\mathcal{R}$, where U_i is the universe of $\mathcal{U}_i$ and $U_1 \cap U_2 = \emptyset$. We define the disjoint union $G_1 \oplus G_2$ as the $(n_1 + n_2)$-pointed structure $(\mathcal{U}_1 \oplus \mathcal{U}_2, \tau)$, where $\tau : \{1, \ldots, n_1 + n_2\} \to U_1 \uplus U_2$ with $\tau(i) = \tau_1(i)$ for all $1 \leq i \leq n_1$ and $\tau(i + n_1) = \tau_2(i)$ for all $1 \leq i \leq n_2$. For an n-pointed structure $G = (\mathcal{U}, \tau)$ and an injective mapping $f : \{1, \ldots, m\} \to \{1, \ldots, n\}$ ($m \leq n$), we define $\mathrm{rename}_f(G) = (\mathcal{U}, \tau \circ f)$. Finally, if $n \geq 2$, then $\mathrm{glue}(G) = (\mathcal{U}/_{\equiv}, (\pi_{\equiv} \circ \tau) \restriction \{1, \ldots, n-1\})$, where $\equiv$ is the smallest equivalence relation on $\mathcal{U}$ which contains the pair $(\tau(n), \tau(n-1))$.

Thus, the glue-operation simply merges the last two contact nodes. Note that the combination of rename_f and glue allows to merge arbitrary contact nodes.

A *straight-line program (SLP)* $\mathcal{S} = (X_i := t_i)_{1 \le i \le \ell}$ (over the signature $\mathcal{R}$) is a sequence of definitions, where the right hand side t_i of the assignment is either an n-pointed *finite* structure (over the signature $\mathcal{R}$) for some n or an expression of the form $X_j \oplus X_k$, $\mathrm{rename}_f(X_j)$, or $\mathrm{glue}(X_j)$ with $j, k < i$, where $1 \le i \le \ell$ and $f : \{1, \dots, m\} \to \{1, \dots, n\}$ is injective. Here, $X_1, \dots, X_\ell$ are formal variables. For every variable X_i its *rank* $\mathrm{rank}(X_i)$ is inductively defined as follows: (i) if t_i is an n-pointed structure, then $\mathrm{rank}(X_i) = n$, (ii) if $t_i = X_j \oplus X_k$, then $\mathrm{rank}(X_i) = \mathrm{rank}(X_j) + \mathrm{rank}(X_k)$, (iii) if $t_i = \mathrm{rename}_f(X_j)$ and $f : \{1, \dots, m\} \to \{1, \dots, n\}$, then $\mathrm{rank}(X_i) = m$, and (iv) if $t_i = \mathrm{glue}(X_j)$, then $\mathrm{rank}(X_i) = \mathrm{rank}(X_j) - 1$. The $\mathrm{rank}(X_i)$-pointed finite structure $\mathrm{eval}(X_i)$ is inductively defined by: (i) if t_i is an n-pointed structure G, then $\mathrm{eval}(X_i) = G$, (ii) if $t_i = X_j \oplus X_k$, then $\mathrm{eval}(X_i) = \mathrm{eval}(X_j) \oplus \mathrm{eval}(X_k)$, and (iii) if $t_i = \mathrm{op}(X_j)$ for $\mathrm{op} \in \{\mathrm{rename}_f, \mathrm{glue}\}$, then $\mathrm{eval}(X_i) = \mathrm{op}(\mathrm{eval}(X_j))$. We define $\mathrm{eval}(\mathcal{S}) = \mathrm{eval}(X_\ell)$. The SLP $\mathcal{S}$ is called *c-bounded* ($c \in \mathbb{N}$) if $\mathrm{rank}(X_i) \le c$ for all $1 \le i \le \ell$. Finally, the *size* $|\mathcal{S}|$ is defined as ℓ plus the size of all explicit n-pointed structures that appear in a right-hand side t_i. It easy to see that $|\mathrm{eval}(\mathcal{S})| \in 2^{\mathcal{O}(|\mathcal{S}|)}$.

Example 8 *In Figure 6, the 2-pointed structure* $\mathrm{eval}(A_2)$*, where* A_2 *is a nonterminal from the hierarchical graph definition* D *from Example 1, is shown. The following SLP generates this graph:*

$A_3 := G$*, where* G *is the right-hand side of* A_3 *from Figure 1*
$B_0 := \overset{2}{\bullet} \xleftarrow{\beta} \overset{1}{\bullet} \xrightarrow{\beta} \overset{3}{\bullet}$
$B_1 := B_0 \oplus A_3$
$B_2 := B_1 \oplus A_3$ *(this is a 7-pointed graph)*
$B_3 := \mathrm{rename}_{f_1}(B_2)$*, with* $f_1 : 3 \mapsto 6, 6 \mapsto 3, 2 \mapsto 4, 4 \mapsto 2, i \mapsto i$ *for* $i \in \{1, 5, 7\}$
$B_4 := \mathrm{glue}(B_3)$ *(this is a 6-pointed graph)*
$B_5 := \mathrm{rename}_{f_2}(B_4)$*, with* $f_2 : i \mapsto i$ *for* $1 \le i \le 5$*, i.e.,* $\mathrm{dom}(f_2) = \{1, \dots, 5\}$
$B_6 := \mathrm{glue}(B_5)$ *(this is a 4-pointed graph)*
$B_7 := \mathrm{rename}_{f_3}(B_6)$*, with* $f_3 : i \mapsto i$ *for* $1 \le i \le 3$*, i.e.,* $\mathrm{dom}(f_3) = \{1, 2, 3\}$
$B_8 := \mathrm{glue}(B_7)$ *(this is a 2-pointed graph)*
$A_2 := \mathrm{rename}_{f_4}(B_8)$*, with* $f_4 : 1 \mapsto 2, 2 \mapsto 1$

Note that the operation rename_{f_2} *just makes the 6-th contact node internal in* $\mathrm{eval}(B_4)$*.*

Remark 9 *It is not hard to see that from a given hierarchical graph definition* D *one can construct in polynomial time a straight-line program* $\mathcal{S}$ *with* $\mathrm{eval}(\mathcal{S}) = \mathrm{eval}(D)$*, see also [9]. Moreover, if* D *is c-bounded, then* $\mathcal{S}$ *is* $c(c+1)$*-bounded.*

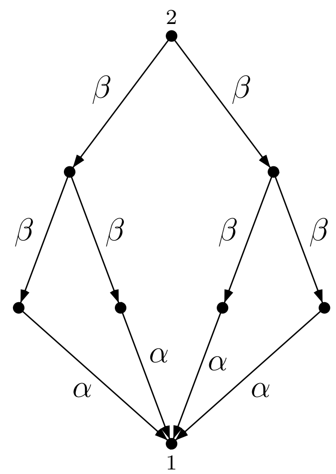


Fig. 6. The graph eval(A_2) for the hierarchical graph definition from Example 1

6 Logic

In this paper, we consider the logics FO (first-order logic), MSO (monadic second-order logic), and SO (second-order logic). A detailed introduction into mathematical logic can be found in [11]. Let us fix a signature $\mathcal{R}$ of relational symbols. Atomic FO formulas over the signature $\mathcal{R}$ are of the form $x = y$ and $r(x_1, \ldots, x_n)$, where $r \in \mathcal{R}$ has arity n and $x, y, x_1, \ldots, x_n$ are first-order variables ranging over elements of the universe. In case r is binary, we also write $x_1 \stackrel{r}{\to} x_2$ instead of $r(x_1, x_2)$. From these atomic subformulas we construct arbitrary FO formulas over the signature $\mathcal{R}$ using boolean connectives and (first-order) quantifications over elements of the universe. A Σ_k-FO formula (respectively 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 (respectively universal) quantifiers, whereas (iii) for i even, B_i is a block of universal (respectively existential) quantifiers. An FO^k-formula ($k \geq 2$) is a first-order formula that 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. Second-order variables (respectively first-order variables) will be always denoted by upper case (respectively lower case) letters. 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 (respectively 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 (respectively universal) SO quantifiers, whereas (iii) for i even, B_i is a block of universal (respectively existential) SO quantifiers. An SO sentence is an SO formula without free variables. For an SO formula $\varphi(X_1, \ldots, X_m, x_1, \ldots, x_n)$, a relational structure $\mathcal{U}$ with universe U, relations $R_i \subseteq U^{\alpha_i}$ (where α_i is the arity of the second-order variable X_i), and $u_1, \ldots, u_n \in U$ we write $\mathcal{U} \models \varphi(R_1, \ldots, R_m, u_1, \ldots, u_n)$ if the sentence φ is true in the structure $\mathcal{U}$ when the variable X_i (respectively x_j) is instantiated by R_i (respectively u_j).

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 $\mathrm{qr}(\forall\alpha\varphi) = \mathrm{qr}(\exists\alpha\varphi) = \mathrm{qr}(\varphi) + 1$, where α 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 over the signature $\mathcal{R}$. This value only depends on k and the signature $\mathcal{R}$, see [27] for an explicit estimation. The k-FO theory (respectively k-MSO theory) of a structure $\mathcal{U}$, briefly k-FOTh($\mathcal{U}$) (respectively k-MSOTh($\mathcal{U}$)), consists of all FO sentences (respectively MSO sentences) of quantifier rank at most k over the signature of $\mathcal{U}$ that are true in $\mathcal{U}$; by the previous remark it is a finite set up to logical equivalence.

In Section 7.1 we will briefly consider *modal logic*, see e.g. [43] 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. 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. [31, Prop. 14.8].

Let us briefly recall the known results concerning the complexity of the model-

<table>
<tr><th>Σ_k-FO</th><th>explicit [5,13,25,44]</th><th>apex</th><th>c-bounded</th><th>unrestricted</th></tr>
<tr><td>data</td><td>$\mathbf{\Sigma_k^{log}}$-compl.</td><td>NL-compl.</td><td>NL-hard
in P</td><td>NL-compl. ($k = 1$)
$\mathbf{\Sigma^p_{k-1}}$-compl. ($k > 1$)</td></tr>
<tr><td>combined</td><td colspan="4">$\mathbf{\Sigma_k^p}$-compl.</td></tr>
</table>

Table 1
FO over hierarchically defined structures

<table>
<tr><th>Σ_k-MSO</th><th>explicit [13,35,44]</th><th>c-bounded</th><th>unrestricted</th></tr>
<tr><td>data</td><td rowspan="2">$\mathbf{\Sigma_k^p}$-compl.</td><td>$\mathbf{\Sigma_k^p}$-compl.</td><td rowspan="2">$\mathbf{\Sigma_k^e}$-compl.</td></tr>
<tr><td>combined</td><td>$\mathbf{\Sigma_k^e}$-compl.</td></tr>
</table>

Table 2
MSO over hierarchically defined structures

checking problem for the fragments of FO and MSO introduced above, when input structures are represented explicitly, e.g., by listing all tuples in all relations of the structure. For Σ_k-FO (respectively Π_k-FO) the data complexity is $\mathbf{\Sigma_k^{log}}$-complete (respectively $\mathbf{\Pi_k^{log}}$-complete) [2] [5,25], whereas the combined complexity goes up to $\mathbf{\Sigma_k^p}$-completeness (respectively $\mathbf{\Pi_k^p}$-completeness) [13,44]. For Σ_k-MSO (respectively Π_k-MSO), both the data and combined complexity is $\mathbf{\Sigma_k^p}$-complete (respectively $\mathbf{\Pi_k^p}$-complete) [13,35,44]. For full second-order logic, the data complexity of Σ_k-SO is still $\mathbf{\Sigma_k^p}$-complete [13,44], whereas the combined complexity becomes $\mathbf{\Sigma_k^e}$-complete [22]. For modal logic, the combined complexity is P-complete, in fact, for every fixed $\ell \geq 2$, the combined complexity of FO^ℓ is P-complete as well [49].

Table 1 and 2 collects the known results for model-checking FO and MSO on explicitly given input structures together with our results for various classes of hierarchically defined input structures. We distinguish on structures which are given by apex, c-bounded (for some fixed c), and unrestricted hierarchical graph definitions.

[2] This means that for every fixed Σ_k-FO sentence, the data complexity is $\mathbf{\Sigma_k^{log}}$ and that there exists a fixed Σ_k-FO sentence, for which the data complexity is $\mathbf{\Sigma_k^{log}}$-hard.

7 FO over hierarchically defined structures

In this section we study the model-checking problem for FO on hierarchically defined input structures. Section 7.1 deals with data complexity. First, we prove that the data complexity of Σ_1-FO for hierarchically defined input structures is NL (Theorem 11). Using this result, we show that for Σ_k-FO (respectively Π_k-FO) with $k > 1$ the data complexity becomes $\mathbf{\Sigma^P_{k-1}}$ (respectively $\mathbf{\Pi^P_{k-1}}$) (Theorem 15 and 16). Next, we study structural restrictions on hierarchical graph definitions that lead to more efficient model-checking algorithms. We prove that under the apex restriction the data complexity of FO goes down to NL (Theorem 19). Finally, we restrict the input to c-bounded hierarchical graph definitions for some fixed integer c. We show that under this restriction, the data complexity of FO reduces to P (Theorem 31), but we cannot provide a matching lower bound.

In Section 7.2 we briefly consider combined complexity. We argue that the combined complexity for Σ_k-FO (respectively Π_k-FO) does not change when moving from explicitly to hierarchically defined input structures (namely $\mathbf{\Sigma^P_k}$ respectively $\mathbf{\Pi^P_k}$) (Theorem 32).

7.1 Data complexity

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

Proposition 10 *It is hard for* NL *to verify for a given hierarchical graph definition* D *whether* $\mathrm{eval}(D)$ *is the empty structure. Thus, given* D*, it is hard for* NL *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 references.*

PROOF. We prove the proposition by a reduction from the NL-complete graph accessibility problem for directed acyclic graphs [42]. 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. For $a \in V \setminus \{v\}$ we introduce the production $A_a \to (\emptyset, \emptyset, \{(A_b, \emptyset) \mid (a, b) \in E\})$. For A_v we introduce the production $A_v \to (\{1\}, \emptyset, \emptyset)$ (1 is just an arbitrary element). Then, $(u, v) \in V$ if and only if the resulting hierarchical graph definition generates a non-empty structure. $\square$

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

Theorem 11 *For every fixed Σ_1-FO or Π_1-FO formula $\varphi(y_1, \dots, y_m)$, the following problem is in NL (and hence in P):*

INPUT: A hierarchical graph definition D and nodes $u_1, \dots, u_m$ from $\mathrm{eval}(D)$ (encoded as described in Remark 6).

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

PROOF. Due to the closure of NL under complement (see e.g. [40]), it suffices to prove the theorem for a Σ_1-FO formula. Let $D = (\mathcal{R}, N, S, P)$. In a first step, take new unary relational symbols $\alpha_1, \dots, \alpha_m$ and use Lemma 7 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 $\mathrm{eval}(D) \models \varphi(u_1, \dots, u_m)$ if and only if $\mathrm{eval}(D') \models \exists x_1 \cdots \exists x_m : \varphi(x_1, \dots, x_m) \wedge \bigwedge_{i=1}^m \alpha_i(x_i)$. 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 $\exists x_1 \cdots \exists x_n : \varphi(x_1, \dots, x_n)$, 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 3 and Remark 4.

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 $\{x_1, \dots, x_n\}$ that occur in ψ. Clearly, there is only a constant number of subformulas. Let $A \in N$ be a nonterminal of rank m and let $\mathrm{eval}_D(A) = (\mathcal{V}, \tau)$. Take new constant symbols $\mathrm{pin}(1), \dots, \mathrm{pin}(m)$, where $\mathrm{pin}(i)$ refers to the i-th contact node $\tau(i)$ of $(\mathcal{V}, \tau)$. Thus, $(\mathcal{V}, \tau)$ can be considered as a structure over the signature $\mathcal{R} \cup \{\mathrm{pin}(1), \dots, \mathrm{pin}(m)\}$. 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 $\{\mathrm{pin}(1), \dots, \mathrm{pin}(m)\}$. For $\theta \in \mathcal{F}(A)$ we denote with θ^+ (respectively θ^-) the set of all positive atoms (respectively negated atoms) that occur in θ. An *assertion* is a pair (A, θ), where $\theta \in \mathcal{F}(A)$. Note that an assertion (A, θ) can be stored in logarithmic space: For A, we just need to store a pointer to the input. Moreover, in each subformula ψ of φ the number of occurrences of variables is bounded by a constant. Hence, when replacing in ψ some of the variables by constants from $\{\mathrm{pin}(1), \dots, \mathrm{pin}(m)\}$ (which can be written down in logarithmic space), we obtain a string of logarithmic length.

We write $\mathrm{valid}(A, \theta)$ for the assertion (A, θ) if there exists a *witness mapping* $\beta : \mathrm{Var}(\theta) \to \mathcal{V} \setminus \mathrm{ran}(\tau)$ such that θ becomes true in $(\mathcal{V}, \tau)$ when every variable $x \in \mathrm{Var}(\theta)$ is replaced by $\beta(x)$.

Example 12 *Let*

$$\psi \equiv r_1(x_1, x_2, x_4) \wedge \neg r_2(x_2, x_3) \wedge r_3(x_4, x_3, x_5) \wedge \neg r_1(x_2, x_3, x_4).$$

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

$$r_1(x_1, \mathrm{pin}(3), \mathrm{pin}(1)) \ \wedge \ \neg r_2(\mathrm{pin}(3), x_3) \ \wedge \\ r_3(\mathrm{pin}(1), x_3, x_5) \ \wedge \ \neg r_1(\mathrm{pin}(3), x_3, \mathrm{pin}(1)). \quad (1)$$

We have

$$\begin{aligned} \theta^+ &= \{r_1(x_1, \mathrm{pin}(3), \mathrm{pin}(1)), r_3(\mathrm{pin}(1), x_3, x_5)\}, \\ \theta^- &= \{\neg r_2(\mathrm{pin}(3), x_3), \neg r_1(\mathrm{pin}(3), x_3, \mathrm{pin}(1))\}, \ \textit{and} \\ \mathrm{Var}(\theta) &= \{x_1, x_3, x_5\}. \end{aligned}$$

Assume that $\mathcal{V} = (\{1, \ldots, 10\}, r_1, r_2, r_3, r_4)$ *where* $r_1 = \{(1, 8, 3), (6, 3, 1)\}$*,* $r_2 = \emptyset$*, and* $r_3 = \{(3, 5, 9), (4, 7, 10)\}$*, and that* $\tau(1) = 3$*,* $\tau(2) = 4$*,* $\tau(3) = 8$*, and* $\tau(4) = 10$*. Then* $\mathrm{valid}(A, \theta)$ *holds: We have to choose for* β *the witness with* $\beta(x_1) = 1$*,* $\beta(x_3) = 5$*, and* $\beta(x_5) = 9$*.*

Claim 13 *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 Claim 13. The formula θ is a conjunction of a constant number of (negated) atoms of the form $(\neg)r(\mathrm{pin}(i_1), \ldots, \mathrm{pin}(i_k))$. It suffices to verify a single atom

$$a = r(\mathrm{pin}(i_1), \ldots, \mathrm{pin}(i_k))$$

in $\mathrm{eval}_D(A)$. Let $A \to (\mathcal{U}, \tau, E)$ be the unique production for A. If $E = \emptyset$, then it is trivial to check $\mathrm{valid}(A, a)$ in NL. Otherwise, assume that $E = \{(A_1, \sigma_1), (A_2, \sigma_2)\}$, where $\mathrm{ran}(\sigma_1) \cup \mathrm{ran}(\sigma_2) = \mathcal{U}$ and all relations in $\mathcal{U}$ are empty (recall that D is in Chomsky normal form). In this case we nondeterministically choose an $i \in \{1, 2\}$ such that $\{\tau(i_1), \ldots, \tau(i_k)\} \subseteq \mathrm{ran}(\sigma_i)$. If such an i does not exist then we can reject immediately. Otherwise we proceed with the assertion (A_i, b), where the atom b results from the atom a by replacing the constant $\mathrm{pin}(i_\ell)$ by $\mathrm{pin}(j)$ if $\tau(i_\ell) = \sigma_i(j)$; since σ_i is injective (see (3) in the definition of hierarchical graph definitions), j is determined uniquely. The atom b can be calculated in logspace from the atom a. This proves Claim 13.

Now we present a nondeterministic logspace algorithm for verifying general assertions (with variables). The algorithm stores a list $\alpha_1 \alpha_2 \cdots \alpha_k$ of assertions where $\mathrm{Var}(\theta_i) \cap \mathrm{Var}(\theta_j) = \emptyset$ if $\alpha_i = (A_i, \theta_i)$, $\alpha_j = (A_j, \theta_j)$, $i \neq j$, and moreover

$$k \leq |\mathrm{Var}(\varphi)| + 2. \quad (2)$$

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. In a single step, the algorithm either rejects or transforms a list of assertions $\alpha_1\alpha_2\cdots\alpha_k$ into a list of assertions $\alpha'_1\alpha'_2\cdots\alpha'_\ell$ such that the following invariant is preserved:

$$\bigwedge_{i=1}^{k} \mathrm{valid}(\alpha_i) \quad\Leftrightarrow\quad \bigwedge_{i=1}^{\ell} \mathrm{valid}(\alpha'_i) \tag{3}$$

Initially, the list only contains the assertion (S, φ). The algorithm accepts, if the list of assertions is empty. Together with (3) this proves the correctness of the algorithm. It remains to describe a single step of the algorithm such that (3) and the space requirement (2) is fulfilled.

Case 1. There exists an i such that $\alpha_i = (A, \theta)$ and $\mathrm{Var}(\theta) = \emptyset$. Then by Claim 13, we can verify in NL whether $\mathrm{valid}(A, \theta)$ is true. If $\mathrm{valid}(A, \theta)$ is rejected, then also the overall algorithm rejects, otherwise it continues with the shorter list $\alpha_1 \cdots \alpha_{i-1}\alpha_{i+1}\cdots\alpha_k$. The correctness property (3) 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. $A \to (\mathcal{U}, \tau, \emptyset)$ is the unique production for A. Then it is again trivial to check in NL whether $\mathrm{valid}(A, \alpha_1)$ and we can proceed as in Case 1.

Case 2.2. $A \to (\mathcal{U}, \tau, \{(A_1, \sigma_1), (A_2, \sigma_2)\})$ is the unique production, where $\mathcal{U} = \mathrm{ran}(\sigma_1) \cup \mathrm{ran}(\sigma_2)$ and all relations of $\mathcal{U}$ are empty. We now guess

(a) 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),
(b) a mapping $\gamma : Y \to \mathcal{U} \setminus \mathrm{ran}(\tau)$, and
(c) 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:

$$\begin{aligned} \mathrm{pin}(j) \text{ occurs in } a \;&\Rightarrow\; \tau(j) \in \mathrm{ran}(\sigma_i) \\ x \text{ occurs in } a \;&\Rightarrow\; (x \in X_i \vee (x \in Y \wedge \gamma(x) \in \mathrm{ran}(\sigma_i))) \end{aligned} \tag{4}$$

These data can be stored in logarithmic space. Intuitively, Y is the set of all variables from $\mathrm{Var}(\theta)$ that will be assigned (via a witness mapping β) to a node in $\mathcal{U} \setminus \mathrm{ran}(\tau) = (\mathrm{ran}(\sigma_1) \cup \mathrm{ran}(\sigma_2)) \setminus \mathrm{ran}(\tau)$ (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 only positive atoms a from θ such that the relational tuple that will finally make the atom a true belongs to the substructure $\mathrm{eval}_D(A_i)$ of $\mathrm{eval}_D(A)$ (the partition $\theta^+ = \psi_1^+ \uplus \psi_2^+$ is not unique, since we may have $\mathrm{ran}(\sigma_1) \cap$

$\mathrm{ran}(\sigma_2) \neq \emptyset$). 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

$$\psi_i^+ \cup \{(\neg a) \in \theta^- \mid a \text{ satisfies (4) for all constants } \mathrm{pin}(j) \\ \text{and all variables } x \in \mathrm{Var}(\theta)\}$$

  (note that a negated atom $\neg a$ may belong to $\psi_1 \cap \psi_2$).
- Next, we replace in ψ_i every constant $\mathrm{pin}(j)$ by $\mathrm{pin}(\ell)$, where $\tau(j) = \sigma_i(\ell)$, and we replace every variable $x \in Y$ by $\mathrm{pin}(\ell)$, where $\gamma(x) = \sigma_i(\ell)$. Let θ_i be the resulting conjunction. Note that $\mathrm{Var}(\theta_i) = X_i$.

We continue with the list $(A_1, \theta_1)(A_2, \theta_2)\alpha_2 \cdots \alpha_k$. Note that $\mathrm{Var}(\theta_i) \subseteq X_i$. Preservation of the invariant (3) follows from the following claim:

Claim 14 $\mathrm{valid}(A, \theta)$ *if and only if there exist* $Y, X_1, X_2, \gamma, \psi_1^+$, *and* ψ_2^+ *such that (a)–(c) hold and* $\mathrm{valid}(A_1, \theta_1)$ *and* $\mathrm{valid}(A_2, \theta_2)$ *for the resulting conjunctions* θ_1 *and* θ_2.

Proof of Claim 14. Recall that $A \to (\mathcal{U}, \tau, \{(A_1, \sigma_1), (A_2, \sigma_2)\})$ is the unique production for A, where $\mathcal{U} = \mathrm{ran}(\sigma_1) \cup \mathrm{ran}(\sigma_2)$ and all relations of $\mathcal{U}$ are empty. Let $\mathrm{eval}(A) = (\mathcal{V}, \tau)$ and $\mathrm{eval}(A_i) = (\mathcal{V}_i, \sigma_i)$ for $i \in \{1, 2\}$. Thus, $\mathcal{U}, \mathcal{V}_1, \mathcal{V}_2 \subseteq \mathcal{V}$.

Let us first assume that $\mathrm{valid}(A, \theta)$ holds. Let $\beta : \mathrm{Var}(\theta) \to \mathcal{V}$ be a witness for this (according to the paragraph before Example 12). Let

$$\begin{aligned} Y &= \{x \in \mathrm{Var}(\theta) \mid \beta(x) \in \mathcal{U} \setminus \mathrm{ran}(\tau)\} \\ X_i &= \{x \in \mathrm{Var}(\theta) \setminus Y \mid \beta(x) \in \mathcal{V}_i\} \\ \beta_i &= \beta \restriction X_i, \\ \gamma &= \beta \restriction Y \end{aligned}$$

where $i \in \{1, 2\}$. Moreover choose a partition $\theta^+ = \theta_1^+ \uplus \theta_2^+$ such that every $a \in \theta_i^+$ becomes true in $\mathcal{V}_i$ under the assignment β. To see that such a partition exists, note that all relations of $\mathcal{U}$ are empty. Thus, every atom in θ^+ has to become true in $\mathcal{V}_1$ or in $\mathcal{V}_2$ under the assignment β (a can be true in both $\mathcal{V}_1$ and $\mathcal{V}_2$ if $\mathrm{ran}(\sigma_1) \cap \mathrm{ran}(\sigma_2) \neq \emptyset$). It is now easy to check that (a)–(c) as well as $\mathrm{valid}(A_1, \theta_1)$ and $\mathrm{valid}(A_2, \theta_2)$ hold.

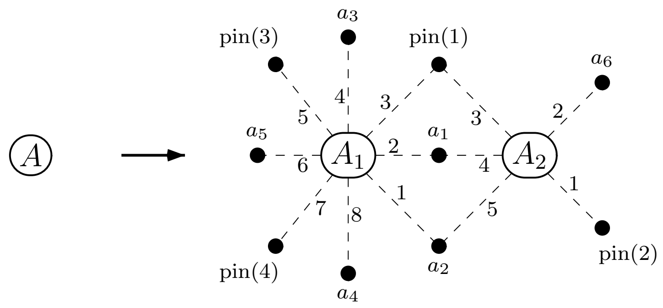
For the other direction, assume that $Y, X_1, X_2, \gamma, \psi_1^+$, and ψ_2^+ are such that (a)–(c) as well as $\mathrm{valid}(A_1, \theta_1)$ and $\mathrm{valid}(A_2, \theta_2)$ hold. Let β_i be a witness for $\mathrm{valid}(A_i, \theta_i)$. Note that $X_i = \mathrm{Var}(\theta_i) = \mathrm{dom}(\beta_i)$. Hence, $\mathrm{dom}(\gamma), \mathrm{dom}(\beta_1)$, and $\mathrm{dom}(\beta_2)$ are pairwise disjoint and we can define $\beta = \beta_1 \cup \beta_2 \cup \gamma$. It follows that β is a witness for $\mathrm{valid}(A, \theta)$. For this, one should notice that a negated atom $\neg a \in \theta^-$ is true under the assignment β if there does *not* exist $i \in \{1, 2\}$

such that a satisfies (4) for all constants $\mathrm{pin}(j)$ and all variables $x \in \mathrm{Var}(\theta)$. The reason is again that all relations of $\mathcal{U}$ are empty.

Example 12 (continued). *Recall that our current assertion is (A, θ), where θ contains the following (negated) atoms:*

$$r_1(x_1, \mathrm{pin}(3), \mathrm{pin}(1)),\ \neg r_2(\mathrm{pin}(3), x_3),\ r_3(\mathrm{pin}(1), x_3, x_5),\ \neg r_1(\mathrm{pin}(3), x_3, \mathrm{pin}(1))$$

Assume that the rule for the nonterminal A is:



Then we may guess for instance $Y = \{x_5\}$, $X_1 = \{x_1\}$, and $X_2 = \{x_3\}$, $\gamma(x_5) = a_1$, $\psi_1^+ = \{r_1(x_1, \mathrm{pin}(3), \mathrm{pin}(1))\}$, and $\psi_2^+ = \{r_3(\mathrm{pin}(1), x_3, x_5)\}$. We finally get: $\theta_1 = r_1(x_1, \mathrm{pin}(5), \mathrm{pin}(3))$ and $\theta_2 = r_3(\mathrm{pin}(3), x_3, \mathrm{pin}(4))$. The two negated atoms $\neg r_2(\mathrm{pin}(3), x_3)$ and $\neg r_1(\mathrm{pin}(3), x_3, \mathrm{pin}(1))$ are automatically satisfied by the above guess, because x_3 is generated by A_2 (since $x_3 \in X_2$) and hence cannot be in any relation of $\mathrm{eval}(A)$ with $\mathrm{pin}(3)$. If the additional negated atom $\neg r_2(\mathrm{pin}(1), x_5)$ would belong to θ, then it would belong to $\psi_1 \cap \psi_2$ and we would have $\theta_1 = r_1(x_1, \mathrm{pin}(5), \mathrm{pin}(3)) \wedge \neg r_2(\mathrm{pin}(3), \mathrm{pin}(2))$ and $\theta_2 = r_3(\mathrm{pin}(3), x_3, \mathrm{pin}(4)) \wedge \neg r_2(\mathrm{pin}(3), \mathrm{pin}(4))$.

For the space requirements of our algorithm, note that 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. $\Box$

Using Theorem 11, we can easily prove an upper bound of $\mathbf{\Sigma_k^P}$ for the data complexity of a fixed Σ_{k+1}-FO sentence on hierarchically defined input structures:

Theorem 15 *For every fixed Σ_{k+1}-FO (respectively Π_{k+1}-FO) sentence ψ, the question, whether $\mathrm{eval}(D) \models \psi$ for a given hierarchical graph definition D is in $\mathbf{\Sigma_k^P}$ (respectively $\mathbf{\Pi_k^P}$).*

PROOF. 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 6 in Section 5.1. Since the size of this representation for a node is of polynomial size, this guessing needs polynomial time. Moreover, if i is odd (respectively even) we guess the tuple $\overline{u}_i$ in an existential (respectively universal) state of our alternating machine. 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 11. □

Next, we prove a matching lower bound:

Theorem 16 *For every $k \geq 1$, there exists a fixed Σ_{k+1}-FO (respectively Π_{k+1}-FO) sentence ψ such that the question, whether* $\mathrm{eval}(D) \models \psi$ *for a given hierarchical graph definition D, is hard for $\mathbf{\Sigma^p_k}$ (respectively $\mathbf{\Pi^p_k}$). Finally, the sentence ψ is equivalent to an FO^2-sentence.*

PROOF. Note that for every $k \geq 1$ it suffices to prove the statement either for the class $\mathbf{\Sigma^p_k}$ or $\mathbf{\Pi^p_k}$, because these two classes are complementary to each other, and the negation of a Σ_{k+1}-FO sentence is equivalent to a Π_{k+1}-FO sentence and vice versa. For k even, we prove the statement for $\mathbf{\Sigma^p_k}$, for k odd, we prove the statement for $\mathbf{\Pi^p_k}$. For k odd, the following problem QSAT_k is $\mathbf{\Pi^p_k}$-complete [44,53]:

INPUT: A quantified boolean formula Θ of the form

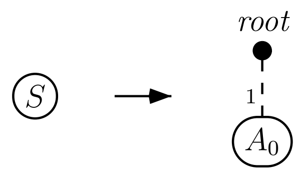
$$\forall x_1 \cdots \forall x_{\ell_1 - 1} \exists x_{\ell_1} \cdots \exists x_{\ell_2 - 1} \cdots \forall x_{\ell_{k-1}} \cdots \forall x_n : \varphi(x_1, \ldots, x_n),$$

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

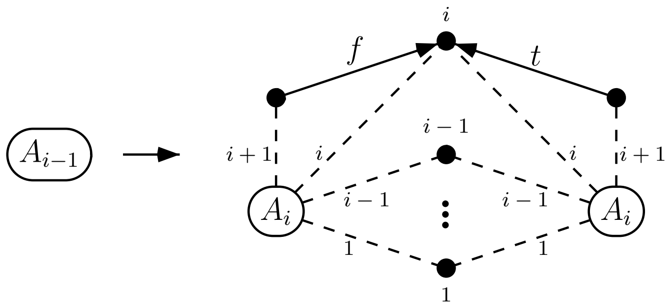
QUESTION: Is Θ true?

For k even, the corresponding problem that starts with a block of existential quantifiers and where φ is in 3-CNF is $\mathbf{\Sigma^p_k}$-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 $D = (\mathcal{R}, N, S, P)$ 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 signature $\mathcal{R}$ contains the binary symbols $g, c, t, f, n_1, n_2, n_3, p_1, p_2, p_3$ and the unary symbol *root*. 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 (respectively right) child is labeled with f for false (respectively 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 reference e gets access to all nodes of the binary tree that were produced by ancestor-references of e. These nodes form a path starting at the root.

Finally, for A_n we introduce the production $A_n \to (\mathcal{U}, \tau, E)$ such that:

- The universe of $\mathcal{U}$ consists of the $n+1$ contact nodes $\tau(1), \ldots, \tau(n+1)$ (which correspond to the $n+1$ nodes along a path from the root to a leaf in the generated tree) and additional 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 contact node $\tau(1)$ (which accesses the root) to contact node $\tau(\ell_1)$, there is a g-labeled edge from $\tau(\ell_{i-1})$ to $\tau(\ell_i)$ for $1 < i < k$, and there is a g-labeled edge from $\tau(\ell_{k-1})$ to $\tau(n+1)$. 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 (recall that k is a constant).
- There is a c-labeled (c for conjunction) edge from $\tau(n+1)$ (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 (respectively n_k-labeled edge), where $k \in \{1, 2, 3\}$, from node c_i to $\tau(j+1)$ ($1 \leq j \leq n$) if and only if x_j (respectively $\neg x_j$) is the k-th literal in the conjunction C_i.

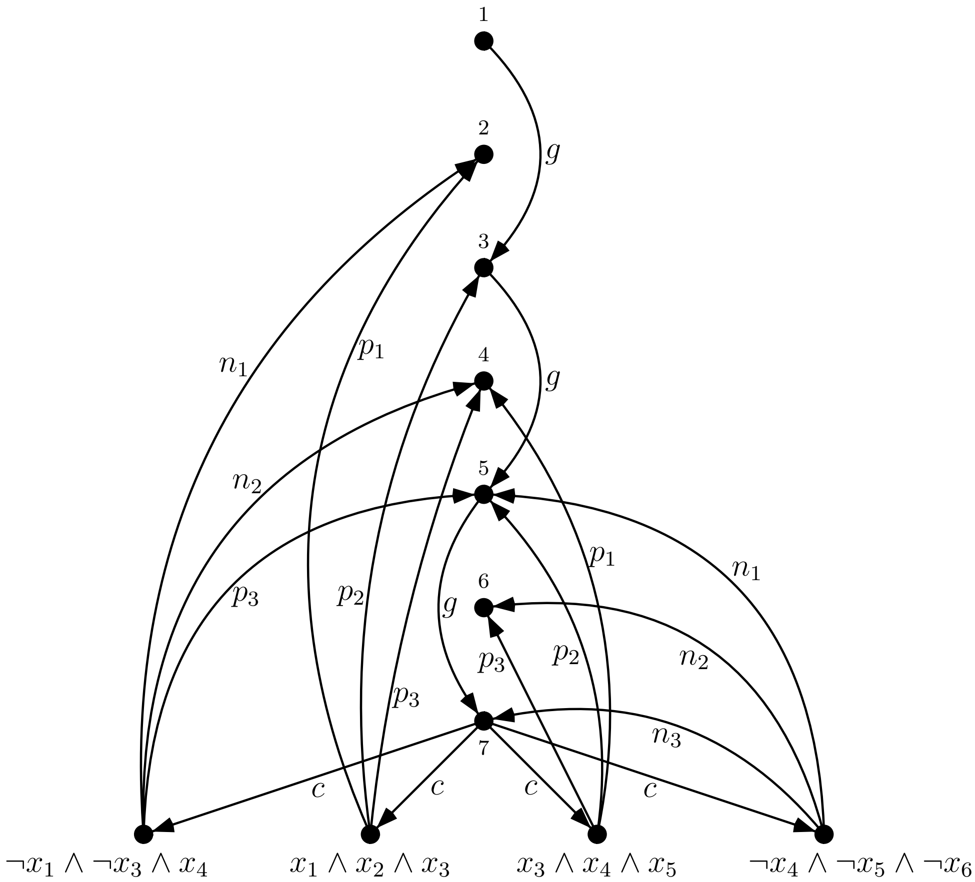


Fig. 7.

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

Example 17 *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, $k = 3$, $n = 6$, $m = 4$. The right-hand side for A_6 is shown in Figure 7. 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 8 shows the path in $\mathrm{eval}(D)$ *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

$$\exists y, y_1, y_2, y_3, y'_1, y'_2, y'_3 : z \xrightarrow{c} y \ \wedge \ \bigwedge_{i=1}^{3} (y \xrightarrow{p_i} y_i \xrightarrow{t} y'_i \vee y \xrightarrow{n_i} y_i \xrightarrow{f} y'_i). \qquad (5)$$

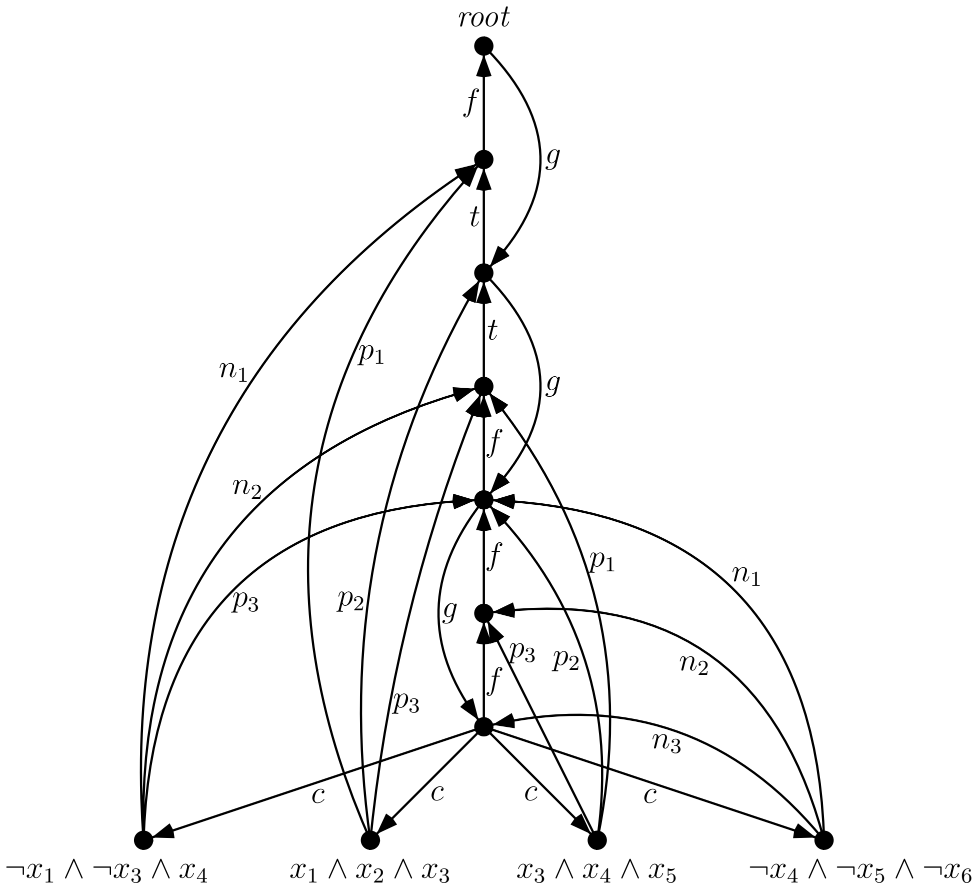


Fig. 8.

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 8, the formula in (5) 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 fixed FO sentence

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

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 fixed 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 6, this modal sentence is equivalent to an FO^2-sentence. This proves the theorem. $\square$

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

Recall from Definition 2 that a hierarchical graph definition $D = (\mathcal{R}, N, S, P)$ is apex, if for every production $(A \to (\mathcal{U}, \tau, E)) \in P$ and every reference $(B, \sigma) \in E$ we have $\mathrm{ran}(\sigma) \cap \mathrm{ran}(\tau) = \emptyset$. Thus, contact nodes of a right-hand side cannot be accessed by references. We will prove that under the apex restriction the data complexity for FO over hierarchically defined input structures becomes NL. The proof of this result is based on Gaifman's locality theorem [18,11]. First we have to introduce a few notations.

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

$$G_{\mathcal{U}} = (U, \{(u, v) \in U \times U \mid \bigvee_{1 \leq i \leq k} \exists (u_1, \dots, u_{\alpha_i}) \in R_i \; \exists j, k : u_j = u \neq v = u_k\}).$$

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

Now let φ be an FO formula over the signature of $\mathcal{U}$ and let x be a variable. Then the FO formula $\varphi^{(r,x)}$ results from φ by relativizing all quantifiers to $S_{\mathcal{U}}(r, x)$. It can be defined inductively, for instance $(\varphi_1 \wedge \varphi_2)^{(r,x)} \equiv \varphi_1^{(r,x)} \wedge \varphi_2^{(r,x)}$, $(\exists y \, \psi)^{(r,x)} \equiv \exists y \{d(x, y) \leq r \ \wedge \ \psi^{(r,x)}\}$ (where y has to be renamed into a fresh variable if $y = x$), and $R_i(x_1, \dots, x_n)^{(r,x)} \equiv R_i(x_1, \dots, x_n)$ 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 (x occurs freely in $\exists y : \{d(x, y) \leq r \ \wedge \ \psi^{(r,x)}\}$ if $y \neq x$). If φ contains at most x free, then we write $(N_{\mathcal{U}}(r, u), u) \models \varphi(x)^{(r,x)}$ if the formula $\varphi(x)^{(r,x)}$ is true in the sphere $N_{\mathcal{U}}(r, u)$ when the variable x is instantiated by u. Gaifman's Theorem states the following [18].

Theorem 18 *Every FO sentence is logically equivalent to a boolean combina-*

tion 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 19 *For every fixed FO sentence* φ*, the question, whether* $\text{eval}(D) \models \varphi$ *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

$$\exists x_1 \cdots \exists x_m \left\{ \bigwedge_{1 \le i < j \le m} d(x_i, x_j) > 2r \;\wedge\; \bigwedge_{1 \le i \le m} \psi(x_i)^{(r,x_i)} \right\}. \tag{6}$$

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

Let $A \in N$ and let $A \to (\mathcal{U}, \tau, E)$ be the production for A. Let $\text{eval}_D(A) = (\mathcal{V}, \tau)$. We identify $\mathcal{U}$ with the substructure of $\mathcal{V}$ induced by those nodes of $\mathcal{V}$ that belong to $\mathcal{U}$; therefore τ denotes both the pin-mapping in $\mathcal{U}$ and $\mathcal{V}$. Then we say that $\text{eval}_D(A)$ contains a *top-level occurrence of a ψ-sphere*, if there exists $v \in \mathcal{V}$ such that

(i) $S_{\mathcal{V}}(v, r) \cap \mathcal{U} \neq \emptyset$,
(ii) $S_{\mathcal{V}}(v, r) \cap \text{ran}(\tau) = \emptyset$, and
(iii) $(N_{\mathcal{V}}(v, r), v) \models \psi(x)^{(r,x)}$.

This means that if we consider a substructure of $\text{eval}(D)$ that is generated from the nonterminal A, then this substructure 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 contact nodes of $\text{eval}_D(A)$ are generated by nonterminals that are 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 20 *We can verify in* L*, whether* $\text{eval}_D(A)$ *contains a top-level occurrence of a ψ-sphere.*

Proof of Claim 20. Due to the apex restriction, if $\text{eval}_D(A)$ 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 structure of polynomial size. Every node of this structure can be represented in logarithmic space. In order to give a more formal exposition, we define a hierarchical graph definition $D(d, A)$ that unfolds A up to depth d:

- The signature of $D(d, A)$ is $\mathcal{R} \uplus \{\alpha, \beta\}$, where α and β are fresh unary symbols.
- 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 $(B \to (\mathcal{U}, \tau, E)) \in P$ we take the production $B_i \to (\mathcal{U}', \tau, E_{i-1})$, where $E_{i-1} = \{(C_{i-1}, \sigma) \mid (C, \sigma) \in E\}$ and $\mathcal{U}' = \mathcal{U}$ if ($B \neq A$ or $i \neq d+1$). For $B = A$ and $i = d+1$ we take for $\mathcal{U}'$ the structure $\mathcal{U}$, where additionally, every internal node $v \in \mathcal{U} \setminus \mathrm{ran}(\tau)$ is labeled with the new unary symbol α and every contact node $v \in \mathrm{ran}(\tau)$ is labeled with the new unary symbol β.
  - For every $(B \to (\mathcal{U}, \tau, E)) \in P$ we take the production $B_0 \to (\mathcal{U}', \tau, \emptyset)$, where $\mathcal{U}'$ results from $\mathcal{U}$ by labeling every node $\sigma(i) \in \mathcal{U}$ such that $(C, \sigma) \in E$ and $1 \leq i \leq \mathrm{rank}(C)$ for some C (i.e., this node is accessed by some reference in E) with the unary symbol 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

$$\mathrm{eval}(D(d, A)) \models \exists x \left\{ \begin{array}{ll} \psi^{(r,x)} \wedge & \text{(i')} \\ \exists y : (\alpha(y) \wedge d(x, y) \leq r) \wedge & \text{(ii')} \\ \forall y : (\beta(y) \to d(x, y) > r) & \text{(iii')} \end{array} \right\}. \tag{7}$$

Note that this is a fixed FO sentence. The subformula (j') (j $\in \{\mathrm{i}, \mathrm{ii}, \mathrm{iii}\}$) ensures property (j) from above. The representation of a node from the structure $\mathrm{eval}(D(d, A))$ (see Remark 6) 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 reference in a right-hand side). Since by Remark 6 we can also check in L, whether a tuple of nodes in $\mathrm{eval}(D(d, A))$ belongs to a given relation from the signature $\mathcal{R}$, any logspace-algorithm for verifying a fixed FO sentence over an explicitly given input structure can be also applied to check whether (7) holds. This proves Claim 20.

Let N_{top} be the set of those $A \in N$ such that $\text{eval}_D(A)$ contains a top-level occurrence of a ψ-sphere. Thus, by Claim 20, we can check in L whether a given nonterminal belongs to N_{top}. Let $\mathcal{P}(D)$ be the set of all root-paths in $\text{dag}(D)$ 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 21 $\text{eval}(D)$ *contains at least* $|\mathcal{P}(D)|$ *many disjoint* ψ*-spheres.*

Proof of Claim 21. 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 $\text{eval}(D)$. 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 21.

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

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

Recall that m is a fixed constant in our consideration (it appears in the fixed sentence (6)).

Claim 22 *The question, whether* $|\mathcal{P}(D)|\rceil^m = k$ *for a given* $k \in \{0, \dots, m\}$ *belongs to* NL*, .*

Proof of Claim 22. For a given number $k \in \{0, \dots, m\}$ we first guess a number $0 \le j \le k$ and we guess j many nonterminals $A_1, \dots, A_j \in N_{\text{top}}$; recall that by Claim 20 we can check membership in N_{top} in logspace. Next we guess for every $1 \le i \le j$ a number $k_i \in \{1, \dots, k\}$ 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 $1 \le i \le j$, in $\text{dag}(D)$ there are at least k_i many different root-paths ending in A_i.
(2) For every $1 \le i \le j$, 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 $|\mathcal{P}(D)|\rceil^m \ge k$. To verify condition (1) in NL, we use k_i (which is bounded by the constant m) many pointers for tracing nondeterministically k_i many different paths in $\text{dag}(D)$. 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 [24,46]. Thus, we can check in NL, whether $|\mathcal{P}(D)|\rceil^m \ge k$. Using the complement closure of NL we can also check in NL, whether $|\mathcal{P}(D)|\rceil^m <$

$k+1$ (which is only necessary if $k < m$). This proves Claim 22.

Our overall NL-algorithm for checking formula (6) first checks in NL whether $|\mathcal{P}(D)|\rceil^m = m$, i.e., whether $|\mathcal{P}(D)| \geq m$. If this is true, then by Claim 21 eval(D) contains at least m disjoint ψ-spheres and we can accept. Thus, let us assume in the following that

$$|\mathcal{P}(D)| < m. \tag{8}$$

Property (8) will enable us to construct (in nondeterministic logspace) a new hierarchical graph definition $D(d)$ such that (i) eval($D(d)$) has only polynomial size and (ii) eval(D) contains at least m disjoint ψ-spheres if and only if eval($D(d)$) contains at least m disjoint ψ'-spheres, where ψ' is a slight modification of ψ, see Claim 26 below. The latter property can be checked in logspace using a logspace algorithm for model-checking a fixed FO sentence in an explicitly given input structure.

For the definition of $D(d)$, we need the following concept: 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 dag(D) such that (i) p is a path from A to B and (ii) except the last node B, p does not visit any other nodes from N_{top}.

Claim 23 *$p(A, B) < m$ for every $A \in N \setminus N_{\text{top}}$ and $B \in N_{\text{top}}$.*

Proof of Claim 23. Assume that $p(A, B) \geq m$. Thus, there are at least m different paths from A to $B \in N_{\text{top}}$. Choose a nonterminal $C \in N_{\text{top}}$ such that C can be reached from B but there does not exist a nonterminal in $N_{\text{top}} \setminus \{C\}$, which can be reached from C. We may have $B = C$. Then there exist at least m many paths from the start nonterminal S to C.[3] Thus, $|\mathcal{P}(D)| \geq m$, which contradicts (8).

Claim 24 *The question, whether $p(A, B) = k$ for given $k \in \{0, \dots, m-1\}$, $A \in N \setminus N_{\text{top}}$, and $B \in N_{\text{top}}$ belongs to NL.*

Proof of Claim 24. The proof is similar to the proof of Claim 22. We use k (which is bounded by the constant m) many pointers for tracing nondeterministically k many different paths in dag(D) from A to B. For each visited node it has to be checked, whether it belongs to $N \setminus N_{\text{top}} \cup \{B\}$. By Claim 20 this is possible in logspace. In this way, we can check in NL, whether $p(A, B) \geq k$. The rest of the argument is the same as in the proof of Claim 22.

[3] For this we have to assume that B can be reached from S. In fact, we can eliminate at the beginning all nonterminals which are not reachable from S. Nondeterministic logspace suffices for this preprocessing.

Now, we can define the hierarchical graph definition $D(d)$. The idea is to unfold nonterminals from $N \setminus N_{\text{top}}$ only up to depth d. As in the definition of $D(d, A)$ (see the proof of Claim 20), we introduce copies $A_0, \ldots, A_{d+1}$ for every $A \in N \setminus N_{\text{top}}$ for this purpose. When arriving at A_0 we do not stop unfolding completely (as in $D(d, A)$) but make a jump in the unfolding process and directly produce $p(A, B)$ many copies of every nonterminal $B \in N_{\text{top}}$ (in fact, we have to introduce a copy B' of B with a slightly modified right-hand side and produce $p(A, B)$ many copies of B').

- The signature of $D(d)$ is $\mathcal{R} \uplus \{\beta\}$, where β is a fresh unary symbol.
- The set of nonterminals of $D(d)$ 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 $D(d)$ is S in case $S \in N_{\text{top}}$, otherwise it is S_0.
- The set of productions of $D(d)$ contains the following productions:

(a) For every $(A \to (\mathcal{U}, \tau, E)) \in P$ with $A \in N_{\text{top}}$ we take the productions $A \to (\mathcal{U}, \tau, E_{d+1})$ and $A' \to (\mathcal{U}', \emptyset, E_{d+1})$. Here E_{d+1} results from E by replacing every reference (B, σ) with $B \in N \setminus N_{\text{top}}$ by (B_{d+1}, σ), and $\mathcal{U}'$ is the structure $\mathcal{U}$, where moreover every old contact node $\tau(i)$ has the additional label β.

(b) For every $(A \to (\mathcal{U}, \tau, E)) \in P$ with $A \in N \setminus N_{\text{top}}$ and every $1 \leq i \leq d+1$ we take the production $A_i \to (\mathcal{U}, \tau, E_{i-1})$, where E_{i-1} is defined as above (with $i-1$ instead of $d+1$).

(c) For every $A \in N \setminus N_{\text{top}}$ we take the production $A_0 \to (\mathcal{U}, \tau, E)$, where $\mathcal{U}$ only consists of the $\text{rank}(A)$ many contact nodes $\tau(1), \ldots \tau(\text{rank}(A))$, which are all labeled with the new unary symbol β. The set of references E contains for every $B \in N_{\text{top}}$, $p(A, B)$ $(< m)$ many references $(B', \emptyset)$. [4]

By (b), we unfold nonterminals from $N \setminus N_{\text{top}}$ in the same way as in D but only up to depth d; by the apex restriction this is sufficient in order to generate the part of the structure that belongs to any ψ-sphere that is generated by a nonterminal from N_{top} on a higher hierarchical level. By (c), from a nonterminal A_0 (with $A \in N \setminus N_{\text{top}}$) we make a shortcut and directly produce $p(A, B)$ many copies of $B' \in N'_{\text{top}}$ for every $B \in N_{\text{top}}$. Note that there is a one-to-one correspondence between paths from A to B in $\text{dag}(D)$ and copies of B that can be derived from A during the unfolding process. We put $p(A, B)$ many copies of B' into the right-hand side of A_0, because $p(A, B)$ is exactly the number of copies of B that can be derived from A when restricting the

[4] Note that E is in fact a multiset. One might easily change the definition of hierarchical graph definitions by allowing the set of references to be a multiset. Alternatively, one can introduce additional nonterminals in order to make a set of references out of a multiset of references.

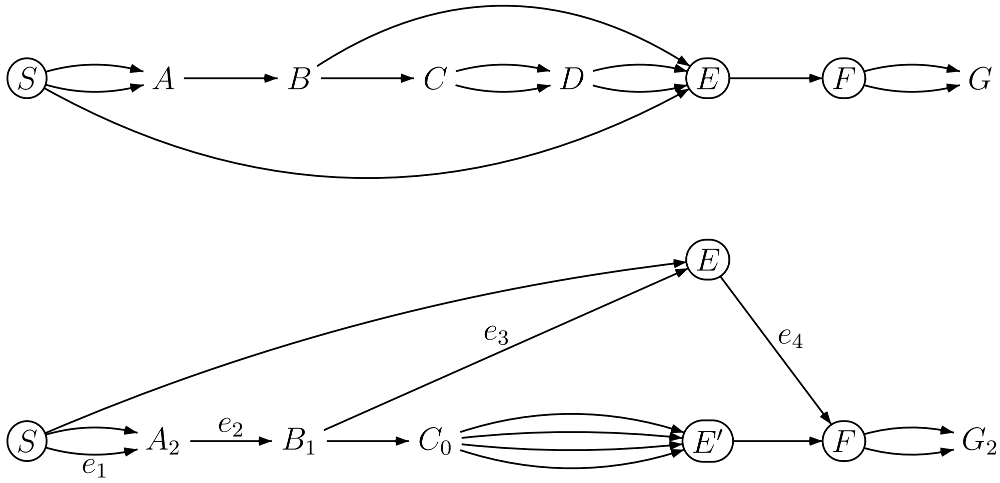


Fig. 9. The dags $\mathrm{dag}(D)$ and $\mathrm{dag}(D(d))$ (the latter restricted to those nodes reachable from S) for $d = 1$ and $N_{\mathrm{top}} = \{S, E, F\}$

unfolding process to nonterminals from $N \setminus N_{\mathrm{top}} \cup \{B\}$.

Example 25 *In this example we only consider the dags associated to hierarchical graph definitions. Assume that the top dag in Figure 9 is* $\mathrm{dag}(D)$ *from some hierarchical graph definition* D*. Assume that* $N_{\mathrm{top}} = \{S, E, F\}$*; these nonterminals are enclosed by circles in Figure 9. Moreover, in Figure 9 we omit the edge labels from* $\mathbb{N}$*; these labels are not relevant in this context. We have* $|\mathcal{P}(D)| = 11$*. The lower part of Figure 9 shows* $\mathrm{dag}(D(d))$ *restricted to those nodes that are reachable from the start nonterminal* S*. The labels* $e_1, \ldots, e_4$ *just denote some of the edges; they will be useful in a later example.*

The following claim follows directly from the definition of $D(d)$.

Claim 26 *Let* $\psi' = \psi \wedge \forall y : \neg\beta(y)$*. Then* $\mathrm{eval}(D)$ *contains at least* m *disjoint* ψ*-spheres if and only if* $\mathrm{eval}(D(d))$ *contains at least* m *disjoint* ψ'*-spheres.*

Claim 27 *The function that maps a hierarchical graph definition* D *to* $D(d)$ *can be calculated in nondeterministic logspace.*

Proof of Claim 27. In fact, the construction of $D(d)$ from D can be done in deterministic logspace, except for the calculation of the values $p(A, B)$. Here, we simply guess the value $p(A, B)$ and verify the correctness of the guess in NL using Claim 24.

By Claim 26 and 27 as well as the closure of NL under NL-reductions, it suffices to verify in NL, whether $\mathrm{eval}(D(d))$ (represented by $D(d)$ on the input tape) contains at least m disjoint ψ'-spheres. This will be shown in the rest of the proof. For this, we will first show that the size of the structure $\mathrm{eval}(D(d))$ is polynomially bounded.

Let $\widehat{N}_{\text{top}} = N_{\text{top}} \cup N'_{\text{top}}$. Similarly to $\mathcal{P}(D)$, define $\mathcal{P}(D(d))$ as the set of all root-paths in $\text{dag}(D(d))$ 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 $\widehat{N}_{\text{top}}$.

Claim 28 $|\mathcal{P}(D(d))| < m$

Proof of Claim 28. By (8) it suffices to show that $|\mathcal{P}(D)| = |\mathcal{P}(D(d))|$. This follows directly from the construction of $D(d)$: Every root path in $\text{dag}(D)$ ending in a nonterminal $A \in N_{\text{top}}$ corresponds in a natural way to a root path in $\text{dag}(D(d))$ ending either in A or A' and vice versa. See Example 25 for an illustration of this fact.

We know show that the structure $\text{eval}(D(d))$ has only polynomial size. For this, we will show how to compress paths from $\mathcal{P}(D(d))$. Take a path $p \in \mathcal{P}(D(d))$, given by a sequence of consecutive edges in $\text{dag}(D(d))$, which ends in $A \in \widehat{N}_{\text{top}}$. If e is an edge of p such that in $\text{dag}(D(d))$ there is a unique path from the source node of e to A, then we can safely omit the edge e (and all successive edges on p) from the description of the path p. By repeating this argument, it follows, that p can be specified by a sequence $(e_1, \ldots, e_k, A)$ of edges of $\text{dag}(D(d))$, where

- $A \in \widehat{N}_{\text{top}}$ is the target node of p,
- $e_1, \ldots, e_k$ are edges from the path p,
- there is exactly one path from the target of e_i to the source of e_{i+1} for $1 \leq i < k$, but there are at least two paths from the source of e_i to the source of e_{i+1}, and
- there is exactly one path from the target of e_k to the node A, but there are at least two paths from the source of e_k to A.

By the last two points, there are at least $k + 1$ paths from the root S to $A \in \widehat{N}_{\text{top}}$; thus,

$$k + 1 \leq |\mathcal{P}(D(d))| < m$$

(in fact, $2^k \leq |\mathcal{P}(D(d))| < m$).

Example 25 (continued). *Consider the lower dag $\text{dag}(D(d))$ in Figure 9. The path (e_1, e_2, e_3, e_4) belongs to $P(D(d))$. It will be encoded by the sequence (e_1, e_3, F).*

Claim 29 *Every node of the structure* $\text{eval}(D(d))$ *can be represented in space* $\mathcal{O}(\log(|D(d)|))$ *(in particular the size of* $\text{eval}(D(d))$ *is bounded polynomially in the size of $D(d)$).*

Proof of Claim 29. According to Remark 6, a node of $\mathrm{eval}(D(d))$ is represented by a pair (p, v), where p is a root-path in $\mathrm{dag}(D(d))$ (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 $\mathrm{dag}(D(d))$ can be stored in logspace. Note that in $\mathrm{dag}(D(d))$, every nonterminal of $D(d)$ 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 $\mathrm{dag}(D(d))$ 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 $\mathcal{P}(D(d))$. By the remark preceding Claim 29, such a path can be represented by a sequence $(e_1, \ldots, e_k, A)$ of $k < m$ edges of $\mathrm{dag}(D(d))$ and one nonterminal $A \in \widehat{N}_{\mathrm{top}}$. Since m is a fixed constant, logarithmic space suffices. To sum up, a node of $\mathrm{eval}(D(d))$ can be represented by a tuple $((e_1, \ldots, e_k), q, v)$, where $(e_1, \ldots, e_k)$ are edges of $\mathrm{dag}(D(d))$, $k < m$, q is a sequence of edges that specifies a path of length at most $d+1$ in $\mathrm{dag}(D(d))$ that starts in a node from $\widehat{N}_{\mathrm{top}}$ and ends in some nonterminal A, and v is an internal node from the right-hand side of A.

Claim 30 *Let $(u_1, \ldots, u_\ell)$ be a tuple of nodes of* $\mathrm{eval}(D(d))$ *represented as in the proof of Claim 29. Then, given $D(d)$ and $(u_1, \ldots, u_\ell)$ as input, it can be checked in* NL*, whether $(u_1, \ldots, u_\ell)$ belongs in* $\mathrm{eval}(D(d))$ *to some given relation $R \in \mathcal{R}$.*

Proof of Claim 30. Let $((e_1, \ldots, e_k), q, v)$ be the logspace representation of u_i from the proof of Claim 29. Thus, $((e_1, \ldots, e_k), q)$ represents a root-path p in $\mathrm{dag}(D(d))$. Then, (p, v) is the ordinary (polynomial size) representation of u_i according to Remark 6. 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 $\mathrm{dag}(D(d))$ 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 6, given the ordinary (polynomial size) representation of $u_1, \ldots, u_\ell$, it can be checked in logspace, whether $(u_1, \ldots, u_\ell) \in R$. Claim 30 follows from the closure of NL under NL-reductions.

Now we can finish the proof of Theorem 19. Recall that it suffices to check in NL, whether the structure $\mathrm{eval}(D(d))$ (represented by $D(d)$) contains at least m disjoint ψ'-spheres. Thus, we have to verify a fixed first-order sentence φ in $\mathrm{eval}(D(d))$. 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 $\mathrm{eval}(D(d))$ using the logspace representation from Claim 29. After guessing such a representation, we have to verify that the guessed data indeed represent a node of $\mathrm{eval}(D(d))$. This is easily possible in NL, since it can be checked in NL, whether there is a unique path between two nodes of a dag. Finally, we have to verify atomic statements on the logspace representations of the guessed nodes, which is possible in NL

by Claim 30. This finishes the proof of the theorem. $\square$

Recall from Definition 2 that a hierarchical graph definition $D = (\mathcal{R}, N, S, P)$ is c-bounded ($c \in \mathbb{N}$), 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 references.

Theorem 31 *For every fixed FO sentence φ and every fixed $c \in \mathbb{N}$, the question, whether* $\mathrm{eval}(D) \models \varphi$ *for a given c-bounded hierarchical graph definition D 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 $\mathcal{R}$ be the fixed signature, over which φ is defined. W.l.o.g. we may assume that our input hierarchical graph definition is also defined over the signature $\mathcal{R}$. Thus, let $D = (\mathcal{R}, N, S, P)$ be a c-bounded hierarchical graph definition. By Remark 9, we can construct from D an equivalent straight-line program $\mathcal{S} = (X_i := t_i)_{1 \leq i \leq \ell}$ over the fixed signature $\mathcal{R}$ such that for every formal variable X_i, $\mathrm{rank}(X_i) \leq d(c)$, where $d(c)$ is a constant that only depends on c. Thus, for every $1 \leq i \leq \ell$, the structure $\mathrm{eval}(X_i)$ can be viewed as a relational structure over some subsignature Θ_i of the *fixed signature* $\Theta = \mathcal{R} \cup \{\mathrm{pin}(1), \ldots, \mathrm{pin}(d(c))\}$. Here, as in the proof of Theorem 11, $\mathrm{pin}(i)$ is a constant symbol that denotes the i-th contact node of $\mathrm{eval}(X_i)$. 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 6) over the signature Θ is bounded by some constant.

By [10] (see also [14,34]), there exist functions $F_{\oplus}$, F_{glue}, and F_f (where $f : \{1, \ldots, m\} \to \{1, \ldots, n\}$ is injective, $n, m \leq d(c)$) over the set of all k-FO theories over the signature Θ such that

- $\text{k-FOTh}(G_1 \oplus G_2) = F_{\oplus}(\text{k-FOTh}(G_1), \text{k-FOTh}(G_2))$,
- $\text{k-FOTh}(\mathrm{glue}(G)) = F_{\mathrm{glue}}(\text{k-FOTh}(G))$, and
- $\text{k-FOTh}(\mathrm{rename}_f(G)) = F_f(\text{k-FOTh}(G))$.

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

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

(1) If $X_i := t_i$ is a definition from $\mathcal{S}$ such that t_i is an n-pointed ($n \leq d(c)$) structure G, then we calculate k-FOTh(G), which is possible in polynomial time (in fact in AC^0 [5,25]) and replace the definition $X_i := t_i$ by $X_i :=$ k-FOTh(G).
(2) A definition of the form $X_i := X_p \oplus X_q$ is replaced by $X_i := F_\oplus(X_p, X_q)$ and similarly for definitions of the form $X_i := \mathrm{glue}(X_j)$ and $X_i := \mathrm{rename}_f(X_j)$.

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($\mathcal{S}$)). We finally check, whether $\varphi \in$ k-FOTh(eval($\mathcal{S}$)). $\square$

One may generalize Theorem 31 by considering straight-line programs that in addition to the operators $\oplus$, glue, and rename_f contain further operators that are compatible with the calculation of the k-FO theory, see [34] for such operations.

Theorem 15, 16, 19, and 31 give us a clear picture on the conditions that make the model-checking problem for FO on hierarchically defined input structures difficult: references have to access contact nodes and references have to access an unbounded number of nodes.

7.2 Combined complexity

In the previous section, we have seen that for Σ_k-FO, data complexity increases considerably when moving from explicitly given input structures to hierarchically defined input structures (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 structures is $\mathbf{\Sigma_k^p}$):

Theorem 32 *The following problem is complete for $\mathbf{\Sigma_k^p}$ (respectively $\mathbf{\Pi_k^p}$):*

INPUT: A hierarchical graph definition D and a Σ_k-FO (respectively Π_k-FO) sentence φ

QUESTION: $\mathrm{eval}(D) \models \varphi$ *?*

PROOF. The lower bound follows from the corresponding result for explicitly given input structures. For the upper bound we can follow the arguments for the proof of Theorem 15. $\square$

For explicitly given input structures, the combined complexity reduces from PSPACE to P when moving from FO to FO^m for some fixed m [49]. A slight modification of the proof of Theorem 16 shows that for hierarchically defined structures, PSPACE-hardness already holds for the data complexity of FO^2 (without any restriction on the quantifier prefix). We just have to start with an instance of QBF (quantified boolean satisfiability) and carry out the construction in the proof of Theorem 16.

8 MSO and SO over hierarchically defined structures

In this section we study the model-checking problem for MSO and SO over hierarchically defined input structures. We prove that the data complexity of Σ_k-SO (respectively Π_k-SO) for hierarchically defined input structures is $\mathbf{\Sigma_k^e}$ (respectively $\mathbf{\Pi_k^e}$) (Theorem 34). 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 (respectively Π_k-MSO) goes down to $\mathbf{\Sigma_k^p}$ (respectively $\mathbf{\Pi_k^p}$) (Theorem 40). Finally, in Section 8.2 we show that also the combined complexity for Σ_k-SO (respectively Π_k-SO) and hierarchically defined input structures is $\mathbf{\Sigma_k^e}$ (respectively $\mathbf{\Pi_k^e}$) (Theorem 41). In fact, the lower bound already holds for Σ_k-MSO and 2-bounded hierarchical graph definitions (Theorem 42).

We should remark that the apex restriction from Section 7.1 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 binary symbol α) whenever a tentacle of a reference accesses a contact 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.

8.1 Data complexity

In order to obtain a sharp lower bound on the data complexity of Σ_k-MSO over hierarchically defined structures, we will use the following computational problem $\mathrm{QO}\Sigma_k$-SAT (respectively $\mathrm{QO}\Pi_k$-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 an input for $\mathrm{QO}\Sigma_k$-SAT is a formula Θ of the form

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

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

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

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 $\mathcal{F}_m$. Thus, in these problems we allow to quantify over boolean functions of arbitrary arity, which are objects of exponential size. It is therefore clear that QOΣ_k-SAT (respectively QOΠ_k-SAT) belongs to the level $\mathbf{\Sigma_k^e}$ (respectively $\mathbf{\Pi_k^e}$) of the EXP time hierarchy. The following proposition is shown for $k = 1$ in [4].

Proposition 33 *For all $k \ge 1$, the problem QOΣ_k-SAT (respectively QOΠ_k-SAT) is complete for $\mathbf{\Sigma_k^e}$ (respectively $\mathbf{\Pi_k^e}$).*

PROOF. We demonstrate the general idea for the class $\mathbf{\Sigma_3^e}$, the same ideas also work for the other levels of the EXP time hierarchy. Let M be a fixed alternating Turing-machine such that:

(a) M accepts a $\mathbf{\Sigma_3^e}$-complete language $L(M)$,
(b) the initial state is an existential state,
(c) M makes on every computation path exactly 2 alternations, and
(d) for an input of length n, M makes exactly $2^{p(n)}$ (for a polynomial $p(n)$) transitions between two alternations as well as after (respectively before) the last (respectively first) alternation.

Thus, the total running time is $3 \cdot 2^{p(n)}$ on every computation path. Properties (c) and (d) can be easily enforced without losing property (a). Let w be an input for M of length n and 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 $2^m \ge 2^q + 2^{cq}$, we can bring the above formula into the form

$$\exists \overline{x}_1 \in \{0,1\}^{2^m} \forall \overline{x}_2 \in \{0,1\}^{2^m} \exists \overline{x}_3 \in \{0,1\}^{2^m} : \psi_w(\overline{x}_1\, \overline{x}_2\, \overline{x}_3). \tag{9}$$

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 every clause of ψ_w has the form

$$(t_1 \oplus x(i_1,b_1)) \ \vee\ (t_2 \oplus x(i_2,b_2)) \ \vee\ (t_3 \oplus x(i_3,b_3)), \tag{10}$$

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 (10) belongs to ψ_w. Thus, there exists a boolean predicate p_w, which can be computed in polynomial time from w such that (10) 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 [4, proof of Prop. 4.2].

Let $\mathcal{F}_m$ be the set of all m-ary boolean functions. Then, (9) is equivalent to

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

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. $\square$

Theorem 34 *For every fixed Σ_k-SO sentence (respectively Π_k-SO sentence) φ, the question, whether* $\mathrm{eval}(D) \models \varphi$ *for a given hierarchical graph definition D, is in $\mathbf{\Sigma^e_k}$ (respectively $\mathbf{\Pi^e_k}$).*

Moreover, for every level $\mathbf{\Sigma^e_k}$ (respectively $\mathbf{\Pi^e_k}$) of the EXP *time hierarchy* EH*, there exists a fixed Σ_k-MSO sentence (respectively Π_k-MSO sentence) φ such that the question, whether* $\mathrm{eval}(D) \models \varphi$ *for a given hierarchical graph definition D, is hard for $\mathbf{\Sigma^e_k}$ (respectively $\mathbf{\Pi^e_k}$).*

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 universe U of $\mathrm{eval}(D)$. 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 U. Since $|U|$ 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 U can be guessed in exponential time. At the end, we have to verify, whether $\mathrm{eval}(D) \models \psi(\overline{U}_1, \ldots, \overline{U}_k)$, 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 U. We unfold the structure $\mathrm{eval}(D)$ 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 $\mathrm{eval}(D) \models \psi(\overline{U}_1, \ldots, \overline{U}_k)$. 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 $\mathrm{QO}\Sigma_k$-SAT (respectively $\mathrm{QO}\Pi_k$-SAT). We will present the construction only for $\mathrm{QO}\Sigma_2$-SAT, the general case can be dealt analogously. Thus, let Θ be a formula of the form

$$\exists f_1 \in \mathcal{F}_m \forall f_2 \in \mathcal{F}_m \exists \overline{x}_1, \overline{x}_2 \in \{0,1\}^m \, \exists \overline{y} \in \{0,1\}^\ell : \\ \varphi(\overline{x}_1, \overline{x}_2, \overline{y}, (f_i(\overline{x}_j))_{1\leq i,j\leq 2}). \quad (11)$$

We will construct a hierarchical graph definition D and a fixed Σ_2-MSO sentence ψ such that Θ is a positive $\mathrm{QO}\Sigma_2$-SAT-instance if and only if $\mathrm{eval}(D) \models \psi$. 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 unary relational symbols tt, ff, AND, OR, NOT, *root*, *var*, f_1^1, f_2^1, f_1^2, and f_2^2 and the binary symbols 1 and 2 (and an additional symbol for unlabeled edges). The nonterminals are S, $A_1^0, A_2^0, \ldots, A_1^m, A_2^m$, where $\mathrm{rank}(S) = 0$ and $\mathrm{rank}(A_i^j) = 2m + j$. The initial rule of D is shown in Figure 10. 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 accessed via the i-th (respectively $(m+i)$-th) tentacle of the A_j^0-labeled reference ($1 \leq i \leq m$, $j \in \{1,2\}$) represents the i-th variable of the sequence $\overline{x}_1$ (respectively $\overline{x}_2$). 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 in the usual way as a directed acyclic graph (dag) with edge relation $\to$. The nodes of this dag correspond to the subexpressions of φ, and every node is labeled with the topmost boolean operator

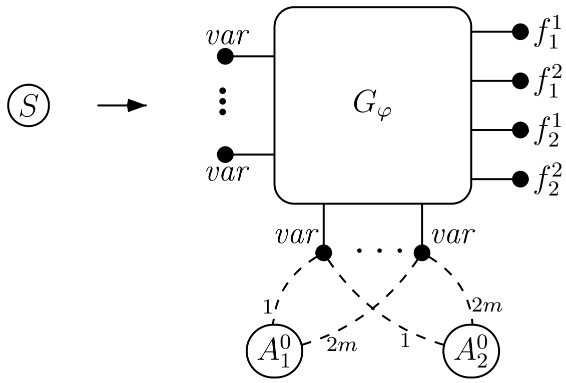


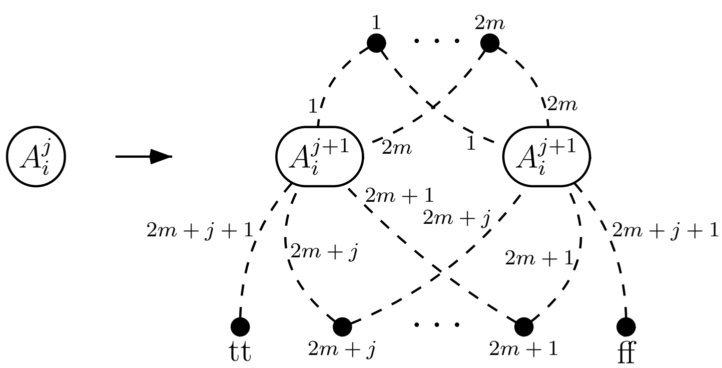
Fig. 10. The initial rule for the hierarchical graph definiton D

(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 $\{\text{AND}, \text{OR}, \text{NOT}, \mathit{root}, \mathit{var}, f_1^1, f_2^1, f_1^2, f_2^2\}$. Assume that $X \subseteq \Lambda$. Then, the fixed formula valid(X), which is defined as

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

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 $0 \leq j < m$:



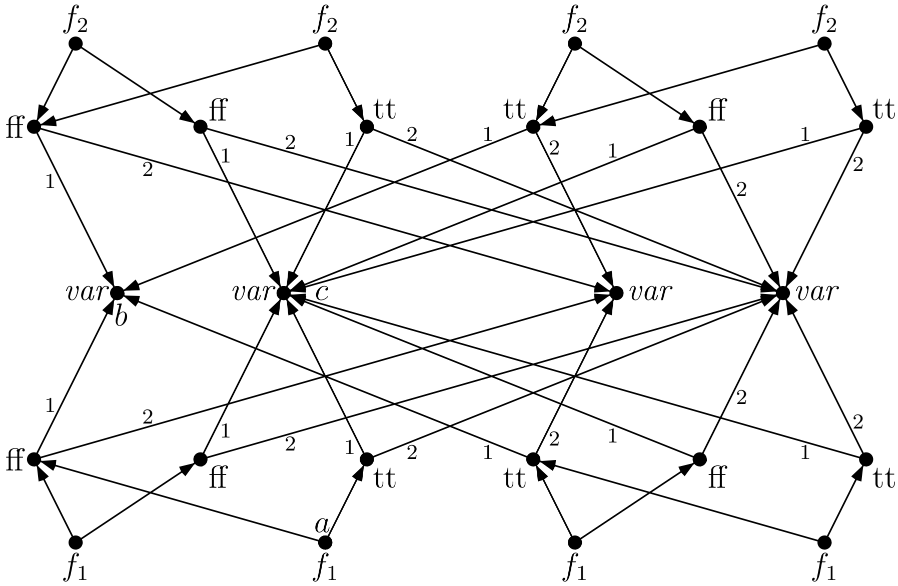
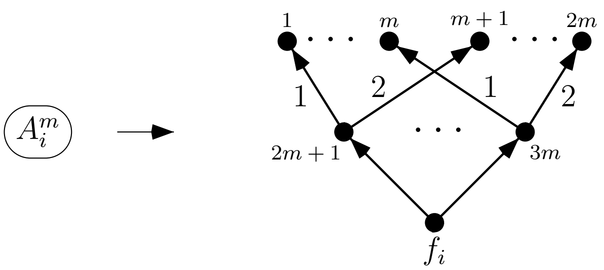


Fig. 11. The graph generated from A_1^0 and A_2^0 for $m = k = 2$

The tentacles with labels $1, \ldots, 2m$ of each A_i^j-labeled reference 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; recall that 1 and 2 are binary relational symbols:



In general, for every $1 \leq j \leq k = 2$ and every $1 \leq i \leq m$, the $(2m+i)$-th contact node is connected with the $((j-1)m+i)$-th contact node (which represents the i-th variable of x_j) via a j-labeled edge. For $i \in \{1, 2\}$ these productions generate 2^m many f_i-labeled nodes (because 2^m many A_i^m-labeled references are generated from A_i^0), one for each possible argument tuple 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.

Example 35 *Figure 11 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 $f_1(0,0) = f_1(1,0) = f_1(1,1) = 0$ and $f_1(0,1) = 1$ is represented by the subset F_1 of the f_1-labeled nodes that contains only the 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 start nonterminal S. For instance, if $b \notin X$ but $c \in X$, then this means that 0 is assigned to the first variable of $\overline{x}_1$ and 1 is assigned to the second (= last) variable of $\overline{x}_1$. Then the fact that $f_1(\overline{x}_1) = f_1(0,1) = 1$ for this f_1 and $\overline{x}_1$ can be expressed by the fact that*

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

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 boolean 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 fixed formula $\psi_{i,j}(F_i, X)$:

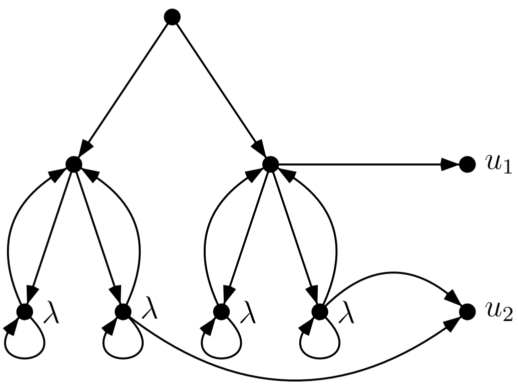
$$\psi_{i,j}(F_i, X) \ \equiv\ \exists x \in F_i\, \forall y\, \forall z : x \to y \xrightarrow{j} z \ \Rightarrow\ (y \in \mathrm{tt} \Leftrightarrow z \in X)$$

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

$$\psi \equiv \exists F_1 \subseteq f_1\, \forall F_2 \subseteq f_2\, \exists X \subseteq \Lambda \left\{ \begin{array}{l} \bigwedge\limits_{1 \le i,j \le 2} \psi_{i,j}(F_i, X) \Leftrightarrow \exists y \in X : f_i^j(y) \\ \wedge\ \mathrm{valid}(X) \end{array} \right\}$$

Then $\mathrm{eval}(D) \models \psi$ if and only if Θ in (11) is a positive $\mathrm{QO}\Sigma_2$-SAT-instance.

Note that the above sentence ψ 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 in order to 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 unary relational symbol.



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 can be easily generated with a small hierarchical graph definition, the construction is similar to the one used in the proof of Theorem 16. Using this additional graph structure we can

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

The resulting formula is a Σ_2-MSO sentence that is true in $\mathrm{eval}(D)$ if and only if $\mathrm{eval}(D) \models \psi$. $\square$

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

A *quantifier prefix* π is a sequence $\pi = Q_1\alpha_1Q_2\alpha_2 \cdots Q_n\alpha_n$, 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 does not contain any (FO or MSO) quantifiers. We define *generalized π-formulas* inductively as follows: If $\pi = \varepsilon$, then a generalized π-formula is just a formula without quantifiers. If $\pi = Q\alpha\,\pi'$ for a quantifier prefix π', 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 $\mathcal{U}$ (over some signature $\mathcal{R}$), briefly gen-π-Th$(\mathcal{U})$, consists of all generalized π-sentences over the signature $\mathcal{R}$ that are true in $\mathcal{U}$.

Example 36 *A typical generalized* $(\exists X \forall y \exists Z)$*-sentence may have the form*

$$\begin{aligned}&\exists X (\forall y\ (\exists Z : \varphi_1(X,y,Z)\ \wedge\ \exists Z : \varphi_2(X,y,Z)) \vee \\ &\quad\ \forall y\ (\exists Z : \varphi_3(X,y,Z))) \wedge \\ &\exists X\ \forall y\ \exists Z : \varphi_4(X,y,Z),\end{aligned}$$

where $\varphi_1, \ldots, \varphi_4$ *do not contain quantifiers.*

The following proposition is a refinement of the well-known Feferman-Vaught decomposition theorem [14] for MSO, see [10,34]. In fact, an analysis of the inductive proof of [10, Lemma 4.5] yields the statement of the proposition. For two structures $\mathcal{U}_1$ and $\mathcal{U}_2$ over signatures $\mathcal{R}_1$ and $\mathcal{R}_2$ (we may have $\mathcal{R}_1 \cap \mathcal{R}_2 \neq \emptyset$), respectively, we consider the disjoint union $\mathcal{U}_1 \oplus \mathcal{U}_2$ as a structure over the signature $\mathcal{R}_1 \cup \mathcal{R}_2$ in the natural way. We only have to require that the set of constant symbols from $\mathcal{R}_1$ and $\mathcal{R}_2$, respectively, are disjoint.

Proposition 37 *Let* $\mathcal{R}_1$ *and* $\mathcal{R}_2$ *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* $\mathcal{R}_1 \cup \mathcal{R}_2$*. Then there exist a finite index set* I *and generalized* π*-formulas* $\theta_{i,1}$ *(over the signature* $\mathcal{R}_1$*) and* $\theta_{i,2}$ *(over the signature* $\mathcal{R}_2$*),* $i \in I$*, such that for all structures* $\mathcal{U}_1$ *and* $\mathcal{U}_2$ *over the signatures* $\mathcal{R}_1$ *and* $\mathcal{R}_2$*, respectively, and all* $V_1, \ldots, V_\ell \subseteq \mathcal{U}_1 \oplus \mathcal{U}_2$*,* $b_1, \ldots, b_m \in \mathcal{U}_1$*, and* $c_1, \ldots, c_n \in \mathcal{U}_2$ *we have:*

$$\begin{aligned}(\mathcal{U}_1 \oplus \mathcal{U}_2) \models \theta(V_1, \ldots, V_\ell, b_1, \ldots, b_m, c_1, \ldots, c_n) \quad &\Leftrightarrow \\ \bigvee_{i \in I} [\mathcal{U}_1 \models \theta_{i,1}(V_1 \cap \mathcal{U}_1, \ldots, V_\ell \cap \mathcal{U}_1, b_1, \ldots, b_m) \wedge& \\ \mathcal{U}_2 \models \theta_{i,2}(V_1 \cap \mathcal{U}_2, \ldots, V_\ell \cap \mathcal{U}_2, c_1, \ldots, c_n)]&\end{aligned}$$

Corollary 38 *Let* $\mathcal{R}_1$ *and* $\mathcal{R}_2$ *be relational signatures and let* θ *be a generalized* π*-sentence over the signature* $\mathcal{R}_1 \cup \mathcal{R}_2$*. Then there exist a finite index set* I *and generalized* π*-sentences* $\theta_{i,1}$ *(over the signature* $\mathcal{R}_1$*) and* $\theta_{i,2}$ *(over the signature* $\mathcal{R}_2$*),* $i \in I$*, such that for all structures* $\mathcal{U}_1$ *and* $\mathcal{U}_2$ *over the signatures* $\mathcal{R}_1$ *and* $\mathcal{R}_2$*, respectively, we have:*

$$(\mathcal{U}_1 \oplus \mathcal{U}_2) \models \theta \quad \Leftrightarrow \quad \bigvee_{i \in I} \mathcal{U}_1 \models \theta_{i,1} \wedge \mathcal{U}_2 \models \theta_{i,2} \tag{12}$$

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

Lemma 39 *Let $\mathcal{R}$ be a relational signature and let θ be a generalized π-sentence over the signature $\mathcal{R}$. Then there exist generalized π-sentences θ' and θ'' over the signature $\mathcal{R}$ such that for all structures $\mathcal{U}$ over the signature $\mathcal{R}$ we have:*

$$\mathrm{glue}(\mathcal{U}) \models \theta \quad \Leftrightarrow \quad \mathcal{U} \models \theta' \tag{13}$$

$$\mathit{rename}_f(\mathcal{U}) \models \theta \quad \Leftrightarrow \quad \mathcal{U} \models \theta'' \tag{14}$$

Theorem 40 *For every fixed Σ_k-MSO sentence (respectively Π_k-MSO sentence) φ and every fixed $c \in \mathbb{N}$, the question, whether* $\mathrm{eval}(D) \models \varphi$ *for a given c-bounded hierarchical graph definition D is in $\mathbf{\Sigma^p_k}$ (respectively $\mathbf{\Pi^p_k}$).*

PROOF. It suffices to prove the statement for Σ_k-MSO sentences. As in the proof of Theorem 31, 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 and let $\mathcal{R}$ be the signature over which φ is defined. Let $D = (\mathcal{R}, N, S, P)$ be a c-bounded hierarchical graph definition over this fixed signature $\mathcal{R}$. As in the proof of Theorem 31 we first transform D into an equivalent straight-line program $\mathcal{S} = (X_i = t_i)_{1 \leq i \leq \ell}$, where $\mathrm{rank}(X_i) \leq d(c)$ for every formal variable X_i. Again, $\mathrm{eval}(X_i)$ is a relational structure over some subsignature Θ_i of the fixed signature $\Theta = \mathcal{R} \cup \{\mathrm{pin}(1), \ldots, \mathrm{pin}(d(c))\}$, 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 31 we first have calculated in polynomial time the theories k-FOTh(G_i) for every definition $X_i := G_i$, where G_i is an explicitly given structure. In the present situation, the direct calculation of k-MSOTh(G_i) would lead to a $\mathsf{P}^{\mathbf{\Sigma^p_k}}$-algorithm, i.e., a polynomial time algorithm with access to an oracle for $\mathbf{\Sigma^p_k}$. It is believed that $\mathbf{\Sigma^p_k}$ is a proper subset of $\mathsf{P}^{\mathbf{\Sigma^p_k}}$. The notion of generalized π-theories was introduced in order to get a $\mathbf{\Sigma^p_k}$-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 38 and Lemma 39 we obtain the following statement:

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

- gen-π-Th$(G_1 \oplus G_2) = F_\oplus($gen-$\pi$-Th$(G_1)$, gen-$\pi$-Th$(G_2))$,
- gen-π-Th$(\mathrm{glue}(G)) = F_{\mathrm{glue}}($gen-$\pi$-Th$(G))$, and

- gen-π-Th(rename$_f(G)$) $= F_f$(gen-π-Th(G)).

Monotonicity of $F_\oplus$ follows from the fact that the right part of the equivalence (12) does not contain negations (i.e., only $\models$ but not $\not\models$ occurs). Analogously, monotonicity of F_{glue} and F_f follows from (13) and (14), respectively.

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 of the straight-line program $\mathcal{S} = (X_i = t_i)_{1 \le i \le \ell}$ a set T_i of generalized π-sentences over the signature Θ_i such that
   (a) $\varphi \in T_\ell$,
   (b) if $t_i = X_p \oplus X_q$, then $T_i \subseteq F_\oplus(T_p, T_q)$,
   (c) if $t_i = \text{glue}(X_j)$, then $T_i \subseteq F_{\text{glue}}(T_j)$, 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 an explicitly given structure G_i, we verify in $\mathbf{\Sigma_k^p}$ whether $G_i \models \bigwedge_{\chi \in T_i} \chi$.

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 ℓ many i such that t_i is an explicitly given structure G_i; let I be the set of all these i. For every $i \in I$ we have to check whether $G_i \models \bigwedge_{\chi \in T_i} \chi$. Note that also $\bigwedge_{\chi \in T_i} \chi$ 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

$$T_i = \text{gen-}\pi\text{-Th}(\text{eval}(X_i))$$

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 $G_i \models \bigwedge_{\chi \in T_i} \chi$ for every i such that $t_i = G_i$ is an explicitly given structure. We prove inductively, that $T_i \subseteq$ gen-π-Th(eval(X_i)) for all $1 \le i \le \ell$. If t_i is an explicit structure, this is clear. If $t_i = X_p \oplus X_q$ for $p, q < i$, then, by induction, $T_p \subseteq$ gen-π-Th(eval(X_p)) and $T_q \subseteq$ gen-π-Th(eval(X_q)). Since $F_\oplus$ is monotonic, we obtain with (1b)

$$\begin{aligned} T_i \subseteq F_\oplus(T_p, T_q) &\subseteq F_\oplus(\text{gen-}\pi\text{-Th}(\text{eval}(X_p)), \text{gen-}\pi\text{-Th}(\text{eval}(X_q))) \\ &= \text{gen-}\pi\text{-Th}(\text{eval}(X_i)). \end{aligned}$$

For the operators glue and rename_f we can argue analogously. Thus, we get $\varphi \in T_\ell \subseteq \text{gen-}\pi\text{-Th}(\text{eval}(X_\ell)) = \text{gen-}\pi\text{-Th}(\text{eval}(\mathcal{S}))$, i.e., $\text{eval}(\mathcal{S}) \models \varphi$. $\square$

8.2 Combined complexity

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

Theorem 41 *For every $k \geq 1$, the following problem is complete for $\mathbf{\Sigma_k^e}$ (respectively $\mathbf{\Pi_k^e}$):*

INPUT: A hierarchical graph definition D and a Σ_k-SO (respectively Π_k-SO) sentence φ

QUESTION: $\text{eval}(D) \models \varphi$ *?*

PROOF. The lower bound follows from Theorem 34. For the upper bound, note that in the upper bound proof of Theorem 34, 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 34. $\square$

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

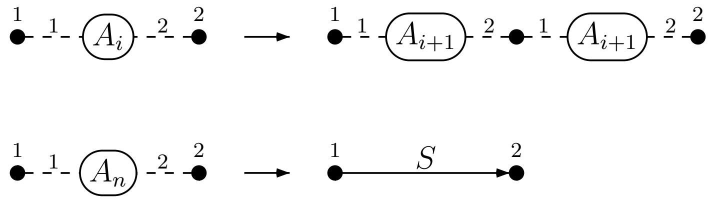
Theorem 42 *For every $k \geq 1$ and every $c \geq 2$, the following problem is complete for $\mathbf{\Sigma_k^e}$ (respectively $\mathbf{\Pi_k^e}$):*

INPUT: A c-bounded hierarchical graph definition D and a Σ_k-MSO sentence (respectively Π_k-MSO sentence) φ

QUESTION: $\text{eval}(D) \models \varphi$ *?*

PROOF. We use a construction from [8,33]. 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 a fixed alternating Turing-machine with a $\mathbf{\Sigma_k^e}$-complete membership problem. Let $\Gamma = \{a_1, \dots, a_m\}$ be the tape alphabet with $a_m = \square$, let $Q = Q_\exists \uplus Q_\forall \uplus F$ be the state set, and let $q_0 \in Q_\exists$ 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 $[0,N]$. The structure $([0,2^n-1],S)$ can be generated by the following 2-bounded hierarchical graph definition of size $\mathcal{O}(n)$ (A_0 is the start nonterminal):



for $0 \leq i \leq n-1$

For an input word w for M we will construct a formula ψ_w such that

$$([0,2^{p(|w|)}-1],S) \models \psi_w \;\Leftrightarrow\; w \text{ is accepted by } M.$$

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 ($1 \leq i \leq m = |\Gamma|$) 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 [33], 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

$$\exists X_1 \cdots \exists X_{m+2} \left\{ \begin{array}{l} \bigwedge_{i=1}^{m-1} X_i = \{k \mid b_k = a_i\} \ \wedge \ X_m = [n, 2^{p(n)} - 1] \ \wedge \\ X_{m+1} = \{0\} \ \wedge \ X_{m+2} = \{q_0\} \ \wedge \ \eta_k(\overline{X}) \end{array} \right\}$$

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 $b - a = 2^i$:

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

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, one can easily 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

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

Thus, $\mathrm{bin}((x_i)_{0\le i<p(n)}, x)$ expresses that $x_0 \cdots x_{p(n)-1}$ is the binary expansion of the number x. Then $x + y = z$ for $x, y, z \in [0, 2^{p(n)} - 1]$ if and only if

$$\begin{aligned}&\exists (x_i)_{0\le i<p(n)} \exists (y_i)_{0\le i<p(n)} \exists (z_i)_{0\le i<p(n)} :\\ &\qquad \mathrm{plus}((x_i)_{0\le i<p(n)}, (y_i)_{0\le i<p(n)}, (z_i)_{0\le i<p(n)}) \wedge \\ &\qquad \mathrm{bin}((x_i)_{0\le i<p(n)}, x) \ \wedge \ \mathrm{bin}((y_i)_{0\le i<p(n)}, y) \ \wedge \ \mathrm{bin}((z_i)_{0\le i<p(n)}, z),\end{aligned}$$

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

9 Conclusion and open problems

In Table 1 and 2 our complexity results for hierarchically defined structures together with the known results for explicitly given input structures 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.

References

[1] R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. W. Reps, and M. Yannakakis. Analysis of Recursive State Machines. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 27(4):786–818, 2005.

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

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

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

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

[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. *Annals of Pure and Applied Logic*, 130(1–3):3–31, 2004.

[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] S. Göller and M. Lohrey. Fixpoint logics on hierarchical structures. In R. Ramanujam and Sandeep Sen, editors, *Proceedings of the 25th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2005), Hyderabad (Indien)*, number 3821 in Lecture Notes in Computer Science, pages 483–494. Springer, 2005.

[21] 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.

[22] 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.

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

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

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

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

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

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

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

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

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

[32] M. Lohrey. Model-checking hierarchical structures. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS 2005), Chicago (USA)*. IEEE Computer Society Press, 2005. 168–177.

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

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

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] N. Markey and P. Schnoebelen. Model checking a path. In *Proceedings of the 14th International Conference on Concurrency Theory (CONCUR 2003), Marseille (France)*, number 2761 in Lecture Notes in Computer Science, pages 248–262, 2003.

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

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

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

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

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

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

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

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

[48] 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.

[49] 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.

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

[51] 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.

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

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