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 graphs using modules for the specification of repeated substructures. Beside this modularity, hierarchical graph definitions allow to specify graphs of exponential size using polynomial size descriptions. In many cases, this succinctness increases the computational complexity of decision problems when input graphs are defined 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 graphs is investigated. Several new complete problems for the levels of the polynomial time hierarchy and the exponential time hierarchy are obtained. Two restrictions on the structure of hierarchical graph definitions that lead to more efficient model-checking algorithms are presented.

1 Introduction

Hierarchical graph definitions specify a graph via modules, where every module is a graph that may refer to modules of a smaller hierarchical level. In this way, large structures can be represented in a modular and succinct way. Hierarchical graph definitions were introduced in [17] in the context of VLSI design. Formally, hierarchical graph definitions can be seen as hyperedge replacement graph grammars [5] that generate precisely one graph.

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

Each of the logics FO, MSO, and SO has many fascinating connections to other parts of computer science, e.g., automata theory, complexity theory, database theory, and verification, see for instance [18] for more details and references. It is therefore not surprising that the model-checking problem for these logics on explicitly given input structures is a very well-studied problem with many deep results. Let us just mention a few references: [6, 8, 9, 12, 13, 20, 22, 27, 28]. 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 [15, 16, 17, 23], 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, 24], where the complexity of temporal logics (LTL, CTL, CTL*) over hierarchically defined strings [24] and hierarchical state machines [1] is investigated. Hierarchical state machines can be seen as a restricted form of hierarchical graph definitions that are tailored towards the modular specification of large reactive systems. We think that the investigation of model-checking problems for "general purpose logics" like FO and MSO over hierarchically defined graphs leads to a better understanding of hierarchical structures in a broad sense.

Our investigation of model-checking problems for hierarchically defined graphs will follow a methodology introduced by Vardi [27]. For a given logic $\mathcal{L}$ and a class of structures $\mathcal{C}$, Vardi introduced three different ways of measuring the complexity of the model-checking problem for $\mathcal{L}$ and $\mathcal{C}$: (i) One may consider a fixed sentence φ from the logic $\mathcal{L}$ and consider the complexity of verifying for a given structure $A \in \mathcal{C}$ whether $A \models \varphi$; thus, only the structure belongs to the input (data complexity or structure complexity). (ii) One may fix a structure A from the class $\mathcal{C}$ and consider the complexity of verifying for a given sentence φ from $\mathcal{L}$, whether $A \models \varphi$; thus, only the formula belongs to the input (expression complexity). (iii) Finally, both the structure and the formula may belong to the input (combined complexity). In the context of hierarchically defined graphs, expression complexity will not lead to new results. Having a fixed hierarchically defined graph makes no difference to having a fixed explicitly given graph. Thus, we will only consider data complexity and combined complexity for hierarchically defined graphs.

Let us mention that also other formalisms for the succinct description of structures were studied under a complexity theoretical perspective: boolean circuits [11, 25, 31], boolean formulas [13, 29], and binary decision diagrams [7, 30]. 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 [16].

After introducing the necessary concepts in Section 2–4, we study model-checking problems for FO over hierarchically defined graphs in Section 5. Section 5.1 deals with data complexity whereas in Section 5.2, combined complexity is briefly considered. Section 6 carries out the same program for MSO and SO. In all cases, we measure the complexity of the model-checking problem in dependence on the structure of the quantifier prefix of the input formula. In some cases we observe an exponential jump in computational complexity when moving from explicitly to hierarchically defined input graphs. In other cases there is no complexity jump at all. We also consider structural restrictions of hierarchical graph definitions that lead to more efficient model-checking algorithms. Our results are collected in Table 1 and Table 2 at the end of the paper, see Section 2–4 for the relevant definitions.

Complete proofs can be found in the full version [19].

2 Preliminaries

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 $\pi_\equiv : A \to [A]_\equiv$ we denote the function with $\pi_\equiv(a) = [a]_\equiv$ for all $a \in A$. For sets A, A_1, and A_2 we write $A = A_1 \uplus A_2$ if $A = A_1 \cup A_2$ and $A_1 \cap A_2 = \emptyset$. For a function $f : A \to B$ let $\mathrm{ran}(f) = \{b \in B \mid \exists a \in A : f(a) = b\}$. For $C \subseteq A$ let $f\restriction_C$ be the restriction of f to C. For functions $f : A \to B$ and $g : B \to C$ we define the composition $g \circ f : A \to C$ by $(g \circ f)(a) = g(f(a))$ for all $a \in A$. For functions $f : A \to C$ and $g : B \to D$ with $A \cap B = \emptyset$ we define the function $f \uplus g : A \uplus B \to C \cup D$ by $(f \uplus g)(a) = f(a)$ for $a \in A$ and $(f \uplus g)(b) = g(b)$ for $b \in B$.

We assume that the reader has some basic background in complexity theory. In particular, we assume that the reader is familiar with the classes $\mathbf{NL}$ (nondeterministic logarithmic space) and $\mathbf{P}$ (deterministic polynomial time). Several times we will use alternating Turing-machines, see [3] 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 either $q \in F$, 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.

The levels of the *polynomial time hierarchy* are defined as follows: Let $k \geq 1$. Then $\mathbf{\Sigma_k^p}$ (resp. $\mathbf{\Pi_k^p}$) is the set of all problems that can be recognized on an alternating Turing-machine within $k-1$ alternations and polynomial time, where furthermore the initial state is assumed to be in $Q_\exists$ (resp. $Q_\forall$). If we replace in these definitions the polynomial time bound by an exponential time bound (i.e., $2^{n^{\mathcal{O}(1)}}$), then we obtain the levels $\mathbf{\Sigma_k^e}$ (resp. $\mathbf{\Pi_k^e}$) of the (weak) *EXP time hierarchy*. If we replace the polynomial time bound by a logarithmic time bound $\mathcal{O}(\log(n))$, then we obtain the levels $\mathbf{\Sigma_k^{\log}}$ (resp. $\mathbf{\Pi_k^{\log}}$) of the *logtime hierarchy*, which is contained in deterministic logspace. Here one assumes that the basic Turing-machine model is enhanced with a random access mechanism; details are not important for this paper. The logtime hierarchy is a uniform version of the circuit complexity class $\mathbf{AC}^0$.

3 Hierarchical graph definitions

A ranked alphabet is a pair (Γ, rank), where Γ is a finite alphabet and $\mathrm{rank} : \Gamma \to \mathbb{N} = \{0, 1, 2, \ldots\}$ assigns to every $a \in \Gamma$ its rank. If the rank-function is clear from the context, we will omit it. Let Γ be a ranked alphabet. A Γ-labeled *hypergraph* is a tuple $H = (V, E, \lambda)$, where V is a finite set of nodes, E is a finite set of *hyperedges*, and $\lambda : E \to \{(A, \tau) \mid A \in \Gamma, \tau : \{1, \ldots, \mathrm{rank}(A)\} \to V\}$ is the labeling function. We also write $V^H = V$, $E^H = E$, and $\lambda^H = \lambda$. If $\lambda(e) = (A, \tau)$, then we say that e is an A-labeled hyperedge. For an equivalence relation $\equiv$ on the set of nodes V, we define the quotient hypergraph $H/_\equiv = ([V]_\equiv, E, \mu)$, where for all $e \in E$, $\mu(e) = (A, \pi_\equiv \circ \tau)$ if $\lambda(e) = (A, \tau)$. For a hyperedge $e \in E$ we define the hypergraph $H \setminus e = (V, E \setminus \{e\}, \lambda\restriction_{E \setminus \{e\}})$. Two hypergraphs $H_1 = (V_1, E_1, \lambda_1)$ and $H_2 = (V_2, E_2, \lambda_2)$ are disjoint if $V_1 \cap V_2 = E_1 \cap E_2 = \emptyset$. In this case, we define the hypergraph $H_1 \oplus H_2 = (V_1 \uplus V_2, E_1 \uplus E_2, \lambda_1 \uplus \lambda_2)$. For $n \geq 0$, an *n-pointed (Γ-labeled) hypergraph* is a pair $G = (H, \sigma)$, where H is a (Γ-labeled) hypergraph and $\sigma : \{1, \ldots, n\} \to V^H$ is an *injective* mapping. The nodes

$\sigma(i)$ ($1 \leq i \leq n$) are also called the *pin nodes* of G. Nodes in $V^H \setminus \mathrm{ran}(\sigma)$ are called *internal nodes* of G. For $A \in \Gamma$ with $\mathrm{rank}(A) = n$ we define the n-pointed hypergraph $G_A = ((\{1, \dots, n\}, \{e\}, \lambda), \mathrm{id})$ with $\lambda(e) = (A, \mathrm{id})$, where id is the identity function on $\{1, \dots, n\}$.

A *hierarchical graph definition* [17] is a tuple $D = (\Gamma, N, S, P)$, where:

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

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

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

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

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

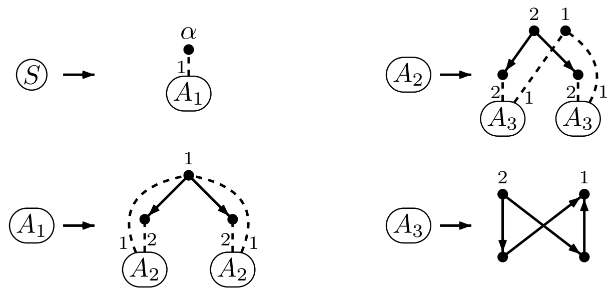
Let us fix a hierarchical graph definition $D = (\Gamma, N, S, P)$. For $i \in \{1, 2\}$ let $G_i = (H_i, \sigma_i)$ be an n-pointed $(\Gamma \uplus N)$-labeled hypergraph for some $n \geq 0$ (thus σ_i is injective). Then we write $G_1 \Rightarrow_D G_2$ if and only if there exists a hyperedge $e \in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

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

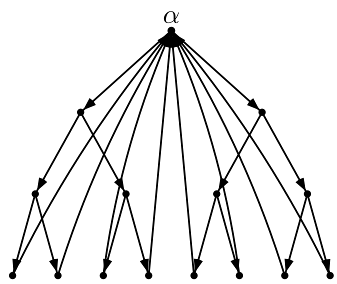
Note that the injectivity of σ_2 follows from the injectivity of σ and σ_1. It is easy to see that for every $A \in N$, there exists a unique Γ-labeled $\mathrm{rank}(A)$-pointed hypergraph $\mathrm{eval}_D(A)$ such that $G_A \overset{*}{\Rightarrow}_D \mathrm{eval}_D(A)$, where $\overset{*}{\Rightarrow}_D$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

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

Example 3.1 *Let us consider the hierarchical graph definition $D = (\Gamma, N, S, P)$, where $\Gamma = \{\alpha, \beta\}$ with $\mathrm{rank}(\alpha) = 1$ and $\mathrm{rank}(\beta) = 2$, $N = \{S, A_1, A_2, A_3\}$ 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 contains the following rules (all solid edges are labeled with β, hence we omit the β-labels):*



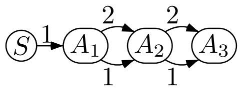
Here, the hierarchical order on the nonterminals N is $S \succ_D A_1 \succ_D A_2 \succ_D A_3$ and $\mathrm{eval}(D)$ is the following graph:



With a hierarchical graph definition $D = (\Gamma, N, S, P)$ we associate an ordered dag (directed acyclic graph) $\mathrm{dag}(D) = (N, E, S)$. The set of nodes is N, the root node is the start nonterminal S. The edge relation E is defined as follows: Let $A \to G$ be the unique production with left-hand side $A \in N$ and let $e_1, \dots, e_n$ be an enumeration of all hyperedges in G that are labeled with a nonterminal (this enumeration is somehow given by the input encoding of D). Assume that e_i is labeled with the nonterminal A_i.

Then $(A, i, A_i) \in E$, i.e., there is an i-labeled edge from A to A_i. A path in $\mathrm{dag}(D)$ that starts in the root S can be uniquely encoded by the sequence of numbers labeling the edges along that path. Such a sequence is called a *root-path* of $\mathrm{dag}(D)$.

For instance, $\mathrm{dag}(D)$ for the hierarchical graph definition from Example 3.1 looks as follows:



The following remark states some simple algorithmic properties of hierarchical graph definitions:

Remark 3.2 *A node of* $\mathrm{eval}(D)$ *can be uniquely represented by a pair* (p, v) *such that (i)* p *is a root-path in* $\mathrm{dag}(D)$ *that ends in the nonterminal* A *and (ii)* $A \to (H, \tau)$ *is the unique production with left-hand side* A*, where* $v \in V^H \setminus \mathrm{ran}(\tau)$ *is an internal node.*[1] *This representation is of size* $\mathcal{O}(|D|)$ *and given a pair* (p, v) *we can check in time* $\mathcal{O}(|D|)$ *(or alternatively in space* $\mathcal{O}(\log(|D|))$*), whether* (p, v) *represents a node of* $\mathrm{eval}(D)$.

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

4 Logic

We identify a ranked alphabet Γ with the relational signature, where every $a \in \Gamma$ is viewed as a relation symbol of arity $\mathrm{rank}(a)$. Thus, a Γ-labeled hypergraph $H = (V, E, \lambda)$ is identified with the relational structure $(V, (R_a)_{a \in \Gamma})$, where $R_a = \{(v_1, \ldots, v_{\mathrm{rank}(a)}) \in V^{\mathrm{rank}(a)} \mid \exists e \in E : \lambda(e) = (a, \tau), v_i = \tau(i) \text{ for } 1 \leq i \leq \mathrm{rank}(a)\}$. Let us fix a ranked alphabet (i.e., a relational signature) Γ for the further discussion.

In this paper, we consider the logics FO (first-order logic), MSO (monadic second-order logic), and SO (second-order logic) over relational structures. More details on these logics can be found for instance in [18]. Atomic FO formulas over the signature Γ are of the form $x = y$ and $a(x_1, \ldots, x_n)$, where $a \in \Gamma$ with $\mathrm{rank}(a) = n$, and $x, y, x_1, \ldots, x_n$ are first-order variables ranging over nodes. The interpretation of $a(x_1, \ldots, x_n)$ is $(x_1, \ldots, x_n) \in R_a$. In case $\mathrm{rank}(a) = 2$ we also write $x_1 \xrightarrow{a} x_2$, in case $\mathrm{rank}(a) = 1$ we also write $x_1 \in a$, i.e, we identify the node label a with the set of all a-labeled nodes. From these atomic formulas we construct arbitrary FO formulas over the signature Γ using boolean connectives and (first-order) quantifications over nodes. A Σ_k-FO formula (resp. Π_k-FO formula) is an FO formula of the form $B_1 B_2 \cdots B_k : \varphi$, where: (i) φ is a quantifier-free FO formula, (ii) for i odd, B_i is a block of existential (resp. universal) quantifiers, whereas (iii) for i even, B_i is a block of universal (resp. existential) quantifiers. An FO^m-formula ($m \geq 2$) is an FO formula that only uses at most m different (bounded or free) variables.

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

Note that the negation of a Σ_k-FO (resp. Σ_k-SO) formula is logically equivalent to a Π_k-FO (resp. Π_k-SO) formula and vice versa. Thus, it suffices to state complexity results for Σ_k-fragments. Then, corresponding results for Π_k-fragments with respect to the complementary complexity classes follow automatically.

Let us briefly recall the known results concerning the complexity of the model-checking problem for the logics introduced above on explicitly given input graphs. For Σ_k-FO the data complexity is $\mathbf{\Sigma_k^{log}}$ [14], whereas the combined complexity goes up to $\mathbf{\Sigma_k^P}$ [6, 26]. For Σ_k-MSO, both the data and combined complexity is $\mathbf{\Sigma_k^P}$ [6, 22, 26]. For full second-order logic, the data complexity of Σ_k-SO is still $\mathbf{\Sigma_k^P}$ [6, 26], whereas the combined complexity becomes $\mathbf{\Sigma_k^e}$ [13]. For every fixed $m \geq 2$, the combined complexity of FO^m is $\mathbf{P}$ [28].

5 FO over hierarchically defined graphs

In this section we study the model-checking problem for FO on hierarchically defined input graphs. Section 5.1 deals with data complexity. Our first result states that the data complexity of Σ_1-FO for hierarchically defined input graphs is $\mathbf{NL}$ (Proposition 5.1 and Theorem 5.2). Using this result, we show that for Σ_k-FO with $k > 1$ the data complexity becomes $\mathbf{\Sigma_{k-1}^P}$ (Theorem 5.3). Next, we study

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

structural restrictions on hierarchical graph definitions that lead to more efficient model-checking algorithms. We introduce the apex restriction, which means that tentacles in a right-hand side are not allowed to access the pin nodes. Under the apex restriction the data complexity of FO goes down to $\mathbf{NL}$ (Theorem 5.4). Finally, we consider hierarchical graph definitions, for which the rank of every nonterminal as well as the number of nonterminal hyperedges in a right-hand side is bounded by some fixed constant c (c-boundedness). Under this restriction the data complexity of FO reduces to $\mathbf{P}$ (Theorem 5.5), but we cannot provide a matching lower bound.

In Section 5.2 we briefly consider combined complexity. We argue that the combined complexity for Σ_k-FO does not change when moving from explicitly to hierarchically defined input graphs (namely $\mathbf{\Sigma_k^P}$, Theorem 5.6).

5.1 Data complexity

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

Proposition 5.1 *It is $\mathbf{NL}$-hard to verify for a given hierarchical graph definition D whether $\mathrm{eval}(D)$ is the empty graph. Thus, given D, it is $\mathbf{NL}$-hard to verify whether $\mathrm{eval}(D) \models \exists x : x = x$. Moreover, for the hierarchical graph definition D we can assume that the rank of every nonterminal is 0 and that every right-hand side of a production contains at most two nonterminal hyperedges.*

Proposition 5.1 can be shown by a straight-forward reduction from the $\mathbf{NL}$-complete graph accessibility problem. For Σ_1-FO, i.e., existential first-order logic, we can also prove a matching $\mathbf{NL}$ upper bound:

Theorem 5.2 *For every fixed Σ_1-FO or Π_1-FO formula $\varphi(y_1, \dots, y_m)$, the following problem is in $\mathbf{NL}$ (and hence in $\mathbf{P}$):*

INPUT: A hierarchical graph definition D and nodes $u_1, \dots, u_m$ from $\mathrm{eval}(D)$ (encoded as described in Remark 3.2).

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

The basic idea behind the proof of Theorem 5.2 is to simply guess for each quantified variable x a node of $\mathrm{eval}(D)$. Of course, this would lead to an $\mathbf{NP}$-algorithm. Roughly speaking, we guess the values for the variables incrementally, by recursively traversing the hierarchical graph definition in a top-down way. It can be shown that only a logarithmic amount of information has to be stored during this traversal.

Theorem 5.3 *For every fixed Σ_{k+1}-FO sentence ψ, the question, whether $\mathrm{eval}(D) \models \psi$ for a given hierarchical graph definition D is in $\mathbf{\Sigma_k^P}$.*

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

Proof. For the upper bound assume that

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

is a fixed Σ_{k+1}-FO formula, where k is assumed to be even (for the case that k is odd, we can argue 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 3.2. Since the size of this representation for a node is of polynomial size, this guessing needs polynomial time. Moreover, if i is odd (resp. even) we guess the tuple $\overline{u}_i$ in an existential (resp. universal) state. It remains to verify, whether $\mathrm{eval}(D) \models \exists \overline{x}_{k+1}\ \theta(\overline{u}_1, \dots, \overline{u}_k, \overline{x}_{k+1})$, which is possible in polynomial time by Theorem 5.2.

For the proof of the second statement, we will show that for k odd (resp. k even) there exists a fixed Π_{k+1}-FO sentence (resp. Σ_{k+1}-FO sentence) for which the model-checking problem is $\mathbf{\Pi_k^P}$-complete (resp. $\mathbf{\Sigma_k^P}$-complete). This suffices in order to prove the second statement, see the remarks in Section 4. For k odd, the following problem QSAT_k is $\mathbf{\Pi_k^P}$-complete [26, 32]:

INPUT: A quantified boolean formula Θ of the form

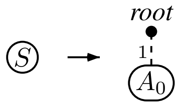
$$\forall x_1 \cdots \forall x_{\ell_1 - 1} \exists x_{\ell_1} \cdots \exists x_{\ell_2 - 1} \cdots \forall x_{\ell_{k-1}} \cdots \forall x_n : \varphi,$$

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

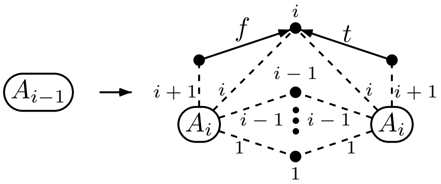
QUESTION: Is Θ true?

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

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



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



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

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

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

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

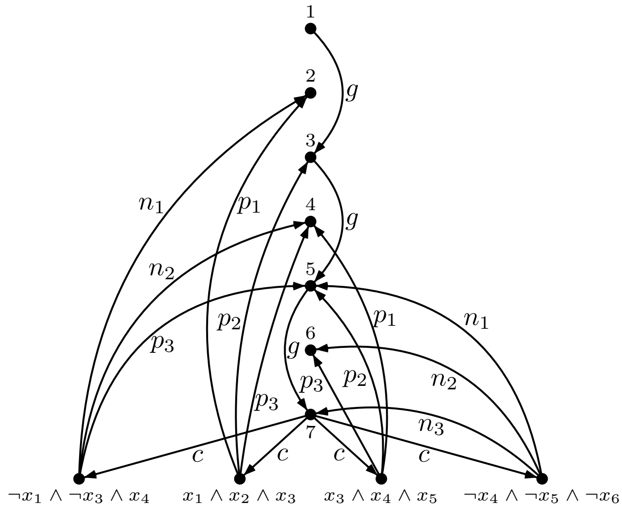


Figure 1.

$$\Theta \equiv \forall x_1 \forall x_2 \exists x_3 \exists x_4 \forall x_5 \forall x_6 : (\neg x_1 \wedge \neg x_3 \wedge x_4) \vee (x_1 \wedge x_2 \wedge x_3) \vee (x_3 \wedge x_4 \wedge x_5) \vee (\neg x_4 \wedge \neg x_5 \wedge \neg x_6).$$

Thus, $k = 3$, $n = 6$, $m = 4$. The right-hand side for A_6 is shown in Figure 1. We have labeled the nodes $c_1, \ldots, c_m$ with the corresponding conjunctions, but these conjunctions do not appear as actual node labels in the right-hand side. For the above formula, Figure 2 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

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

Using the edge $z \xrightarrow{c} y$ we guess a conjunction that will evaluate to true under the assignment represented by the leaf z. Then $y \xrightarrow{p_i} y_i \xrightarrow{t} y_i' \vee y \xrightarrow{n_i} y_i \xrightarrow{f} y_i'$ checks whether the i-th literal of the guessed conjunction evaluates to true. For instance, for the path in Figure 2, the formula in (1) is indeed true; we have to choose the conjunction $\neg x_4 \wedge \neg x_5 \wedge \neg x_6$ for the variable y. From this observation, it follows that

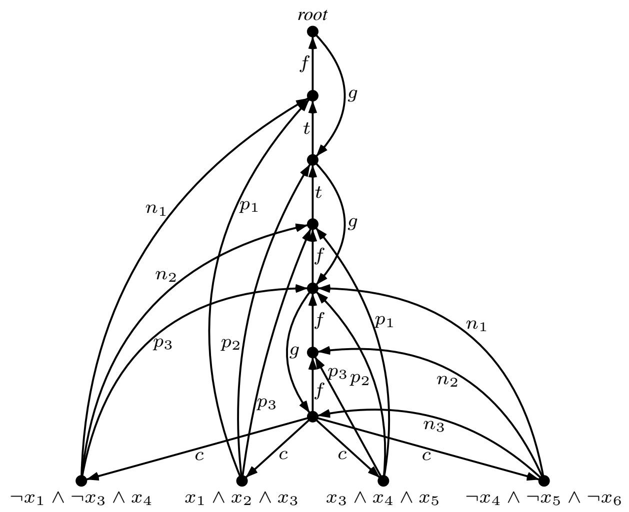


Figure 2.

for the FO sentence

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

we have $\text{eval}(D) \models \psi$ if and only if Θ is a true instance of QSAT_k. If we bring ψ into prenex normal form, we obtain a Π_{k+1}-FO sentence. Finally, it is easy to transform ψ into an equivalent FO^2-sentence. In fact, $\text{eval}(D) \models \psi$ if and only if the following sentence of modal logic is true in the unique *root*-labeled node ($[\alpha]\zeta$ (resp. $\langle\alpha\rangle\zeta$) means that for every (resp. for some) α-successor of the current node, ζ holds):

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

It is well-known that every sentence of modal logic can be transformed into an equivalent FO^2-sentence, see e.g. [18]. This proves the theorem. $\square$

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

A hierarchical graph definition $D = (\Gamma, N, S, P)$ is *apex* if for every production $A \to (H, \sigma)$ from P the following holds: For every $e \in E^H$ such that $\lambda^H(e) = (B, \tau)$ for some $B \in N$ we have $\text{ran}(\sigma) \cap \text{ran}(\tau) = \emptyset$. Thus, pin nodes of a right-hand side cannot be accessed by nonterminal hyperedges. Apex hierarchical graph definitions are called 1-level restricted in [23].

Theorem 5.4 *For every fixed FO sentence φ, the question, whether $\text{eval}(D) \models \varphi$ for a given apex hierarchical graph definition D, is in* $\mathbf{NL}$.

The proof of Theorem 5.4 is based on Gaifman's locality theorem [10], which states that every FO sentence is logically equivalent to a boolean combination of local FO sentences. Roughly speaking, a local sentence states that there are at least m disjoint spheres of radius r in $\text{eval}(D)$ that satisfy some first-order property.[2] Since we consider a fixed FO sentence, m and r are fixed constants. Now, the crucial point is that for an apex hierarchical graph definition D, the size of a sphere of constant radius r is bounded polynomially in the size of D, and nodes of that sphere can be represented in logarithmic space. For non-apex hierarchical graph definitions this is false. For instance, a generalization of Example 3.1 shows that we can generate with $n+1$ productions a binary tree of height n such that moreover every leaf is adjacent to the root of the tree. Thus, the sphere of radius 1 around the root contains $O(2^n)$ many nodes.

For $c \in \mathbb{N}$, a hierarchical graph definition $D = (\Gamma, N, S, P)$ is *c-bounded* if $\text{rank}(A) \le c$ for every $A \in N$ and every right-hand side of a production from P contains at most c hyperedges that are labeled with a nonterminal.

Theorem 5.5 *For every fixed FO sentence φ, every fixed $c \in \mathbb{N}$, and every fixed ranked alphabet Γ, the question, whether $\text{eval}(D) \models \varphi$ for a given c-bounded hierarchical graph definition D with terminal alphabet Γ, is in* $\mathbf{P}$.

The basic idea behind the proof of Theorem 5.5 is based on Courcelle's technique for evaluating fixed MSO formulas in linear time over graph classes of bounded tree width [4]. Starting from a fixed FO sentence φ and a given hierarchical graph definition D we can construct in polynomial time a (deterministic bottom-up) tree automaton A such that $\text{eval}(D) \models \varphi$ if and only if A accepts the derivation tree of the hierarchical graph definition D (which is the unfolding of $\text{dag}(G)$). For this step it is crucial that D is c-bounded for a constant c, and that the terminal alphabet Γ is fixed. The size of the derivation tree of D is exponential in the size of D. But we do not have to generate the derivation tree explicitly, it suffices to run the automaton A on $\text{dag}(G)$, which has polynomial size. Let us mention that if the quantifier nesting depth of φ is k, then the states of A are subsets of the set of all FO sentences of quantifier nesting depth at most k. The later set is finite up to logical equivalence, but its size is non-elementary in k. Thus, A is a fixed automaton that only depends on φ, but its size is non-elementary in the size of φ.

[2] The notion of a sphere is defined w.r.t. the Gaifman graph of $\text{eval}(D)$. The node set of the Gaifman graph is the node set of $\text{eval}(D)$ and two nodes are adjacent if they are related by some terminal hyperedge. Then the sphere of radius r around a node v contains all nodes that have distance at most r from v in the Gaifman graph.

Theorem 5.3–5.5 give us a clear picture on the conditions that make the model-checking problem for FO on hierarchically defined input graphs difficult: nonterminals have to access pin nodes (i.e., references can be passed along nonterminals) and nonterminals have to access an unbounded number of nodes.

5.2 Combined complexity

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

Theorem 5.6 *The following problem is $\mathbf{\Sigma_k^{p}}$-complete:*

INPUT: A hierarchical graph definition D and a Σ_k-FO sentence φ

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

Proof. The lower bound follows from the corresponding result for explicitly given input graphs. For the upper bound we can follow the arguments for the upper bound from Theorem 5.3. $\square$

For explicitly given input graphs, the combined complexity reduces from $\mathbf{PSPACE}$ to $\mathbf{P}$ when moving from FO to FO^m for some fixed m [28]. A slight modification of the proof of Theorem 5.3 shows that for hierarchically defined graphs, $\mathbf{PSPACE}$-hardness already holds for the combined complexity of FO^2. We just have to start with an instance of QBF (quantified boolean satisfiability) and carry out the construction in the proof of Theorem 5.3.

6 MSO and SO over hierarchically defined graphs

In this section we study the model-checking problem for MSO and SO on hierarchically defined input graphs. Both, the data and combined complexity of Σ_k-SO for hierarchically defined input graphs turn out to be $\mathbf{\Sigma_k^{e}}$ (Theorem 6.2 and Theorem 6.4). In fact, the lower bound for the data complexity already holds for Σ_k-MSO. For c-bounded hierarchical graph definitions we can show that the data complexity of Σ_k-MSO goes down to $\mathbf{\Sigma_k^{p}}$ (Theorem 6.3), whereas the combined complexity remains $\mathbf{\Sigma_k^{e}}$ (Theorem 6.5), even for $c = 2$.

We should remark that the apex restriction from Section 5.1 does not lead to more efficient model-checking algorithms in the context of MSO. For an arbitrary hierarchical graph definition D we can enforce the apex restriction by inserting additional edges (labeled with some new terminal α) whenever a tentacle of a nonterminal hyperedge accesses a pin node. If D' denotes this new hierarchical graph definition, then $\mathrm{eval}(D)$ results from $\mathrm{eval}(D')$ by contracting all α-labeled edges. But this contraction is MSO-definable.

6.1 Data complexity

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

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

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

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

In both cases, we ask whether Θ is a true formula. Using a generic reduction we can prove:

Proposition 6.1 *For all $k \geq 1$, the problem QOΣ_k-SAT is $\mathbf{\Sigma_k^{e}}$-complete.*

For $k = 1$ a proof of Proposition 6.1 can be found in [2].

Theorem 6.2 *For every fixed Σ_k-SO sentence φ, the question, whether* $\mathrm{eval}(D) \models \varphi$ *for a given hierarchical graph definition D, is in $\mathbf{\Sigma_k^{e}}$.*

Moreover, for every $k \geq 1$, there exists a fixed Σ_k-MSO sentence φ such that the question, whether $\mathrm{eval}(D) \models \varphi$ *for a given hierarchical graph definition D, is $\mathbf{\Sigma_k^{e}}$-hard.*

The upper bound in Theorem 6.2 follows from Theorem 6.4 in the next section. For the lower bound we use Proposition 6.1. In order to make quantification over m-ary boolean functions possible, we generate with a hierarchical graph definition 2^m many nodes that correspond to the arguments of an m-ary boolean function. Then, quantification over m-ary boolean functions can be simulated by quantification over an arbitrary subset of these 2^m many nodes. An additional graph structure is necessary for evaluating boolean functions that are encoded in this way.

By the next theorem, for c-bounded hierarchical graph definitions the data complexity of Σ_k-MSO goes down to the level $\mathbf{\Sigma_k^P}$ of the polynomial time hierarchy. Thus, the same complexity as for explicitly given input graphs is obtained.

Theorem 6.3 *For every fixed Σ_k-MSO sentence φ, every fixed $c \in \mathbb{N}$, and every fixed ranked alphabet Γ, the question, whether* $\mathrm{eval}(D) \models \varphi$ *for a given c-bounded hierarchical graph definition D with terminal alphabet Γ, is in* $\mathbf{\Sigma_k^P}$.

Similarly to Theorem 5.5 the basic idea behind the proof of Theorem 6.3 is based on Courcelle's technique for evaluating fixed MSO formulas in linear time over graphs of bounded tree width. In fact, we might again construct from the fixed MSO sentence φ and D a tree automaton A such that $\mathrm{eval}(D) \models \varphi$ if and only if A accepts the derivation tree of D. But in the context of MSO, the calculation of the tree automaton A would lead to a $\mathbf{P}^{\mathbf{\Sigma_k^P}}$-algorithm, i.e., a polynomial time algorithm with access to an oracle for $\mathbf{\Sigma_k^P}$. It is believed that $\mathbf{\Sigma_k^P}$ is a proper subset of $\mathbf{P}^{\mathbf{\Sigma_k^P}}$. In order to obtain a $\mathbf{\Sigma_k^P}$-algorithm we have to apply some additional ideas. In particular a refinement of the well-known Feferman-Vaught decomposition theorem for MSO, see e.g. [21], is necessary.

6.2 Combined complexity

Theorem 6.4 *For every $k \geq 1$, the following problem is* $\mathbf{\Sigma_k^e}$*-complete:*

INPUT: A hierarchical graph definition D and a Σ_k-SO sentence φ

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

Proof. The lower bound follows from Theorem 6.2. For the upper bound, one has to notice that an arbitrary m-ary relation over the node set of $\mathrm{eval}(D)$ can be guessed in time $2^{O(m|D|)}$, i.e., in exponential time. Once we have guessed all quantified relations of the input SO sentence, the remaining FO-kernel can be evaluated in deterministic exponential time by explicitly generating $\mathrm{eval}(D)$. $\square$

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

Theorem 6.5 *For every $k \geq 1$ and every $c \geq 2$, the following problem is* $\mathbf{\Sigma_k^e}$*-complete:*

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

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

In fact it suffices in Theorem 6.5 to take a hierarchical graph definition D that generates a linear chain of exponential size. The proof in the full version [19] uses ideas from [20].

<table>
<tr><th>Σ_k-FO</th><th>explicit [6, 14, 26]</th><th>apex</th><th>c-bounded</th><th>general</th></tr>
<tr><td>data</td><td>$\mathbf{\Sigma_k^{\log}}$</td><td>$\mathbf{NL}$</td><td>$\mathbf{NL} \cdots \mathbf{P}$</td><td>$\mathbf{NL} (k=1)$
$\mathbf{\Sigma_{k-1}^P}$ $(k>1)$</td></tr>
<tr><td>combined</td><td colspan="4">$\mathbf{\Sigma_k^P}$</td></tr>
</table>

Table 1. FO over hierarchical graphs

<table>
<tr><th>Σ_k-MSO</th><th>explicit [6, 22, 26]</th><th>c-bounded</th><th>general</th></tr>
<tr><td>data</td><td colspan="2">$\mathbf{\Sigma_k^P}$</td><td rowspan="2">$\mathbf{\Sigma_k^e}$</td></tr>
<tr><td>combined</td><td>$\mathbf{\Sigma_k^P}$</td><td>$\mathbf{\Sigma_k^e}$</td></tr>
</table>

Table 2. MSO over hierarchical graphs

7 Conclusion and open problems

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

References

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

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

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. In *Proc. LICS'2002*, pages 215–224. IEEE Computer Society Press, 2002.

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

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

[12] 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.

[13] 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.

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

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

[16] 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.

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

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

[19] M. Lohrey. Model-checking hierarchical structures. Technical Report 2005/1, University of Stuttgart, Germany, 2005. Available via ftp://ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/TR-2005-01/.

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

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

[22] 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.

[23] 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.

[24] N. Markey and Ph. Schnoebelen. Model checking a path. In *Proc. CONCUR 2003*, LNCS 2761, pages 248–262. Springer 2003.

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

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

[27] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *Proc. STOC 1982*, pages 137–146. ACM Press, 1982.

[28] M. Y. Vardi. On the complexity of bounded-variable queries. In *Proc. PODS 1995*, pages 266–276. ACM Press, 1995.

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

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

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

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