Fixpoint logics on hierarchical structures

Stefan Göller and Markus Lohrey

FMI, Universität Stuttgart, Germany
goeller,lohrey@informatik.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. In this paper, the model-checking problem for the modal μ-calculus and (monadic) least fixpoint logic on hierarchically defined graphs is investigated. In order to analyze the modal μ-calculus, parity games on hierarchically defined graphs are studied.

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 [14] in the context of VLSI design. Formally, hierarchical graph definitions can be seen as hyperedge replacement graph grammars [6] that generate precisely one graph. Specific algorithmic problems (e.g. reachability, planarity, circuit-value, 3-colorability) on hierarchically defined graphs are studied in [12–14, 18].

In this paper we consider the complexity of the model-checking problem for *least fixpoint logic* (LFP) and its fragments *monadic least fixpoint logic* (MLFP) and the *modal μ-calculus*. LFP is the extension of classical first-order logic that allows the definition of least fixpoints of arbitrary arity [15]. MLFP is the fragment of LFP where only monadic fixpoints can be defined. The modal μ-calculus is the fragment of MLFP that is obtained from classical modal logic extended by a monadic fixpoint operator. The model-checking problem for a certain logic 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, input structures will be given in a hierarchical form via *straight-line programs*. Straight-line programs are equivalent to hierarchical graph definitions [16] w.r.t. succinctness, but are more convenient for our purpose.

LFP and its fragments MLFP and the modal μ-calculus found many applications in data base theory, finite model theory, and verification, see e.g. [15]. It is therefore not surprising that the model-checking problem for these logics on explicitly given input structures is a very well-studied problem. Let us just mention a few references: [2, 4, 5, 9–11, 21, 22]. Concerning hierarchically defined graphs, in [1] the complexity of the temporal logics LTL and CTL over hierarchical state machines was 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. Since LTL and CTL can be efficiently translated into the modal μ-calculus, our work is a natural extension of [1]. Moreover, our work continues the previous paper [17] of the second author, where the model-checking problem of first-order logic, monadic second-order logic, and full second-order logic over hierarchically defined graphs was studied.

Our investigation of model-checking problems follows Vardi's methodology from [21]. For a 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 fix a formula $\varphi \in \mathcal{L}$ 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 $A \in \mathcal{C}$ and consider the complexity of verifying for a given formula $\varphi \in \mathcal{L}$, 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 structures, expression complexity does not lead to new results. Having a fixed hierarchically defined structure is the same as having a fixed explicitly given structure. Thus, we only consider data and combined complexity for hierarchically defined structures.

Section 2 introduces the necessary concepts. In Section 3 we show that the winner of a *parity game* on a hierarchically defined graph can be determined in PSPACE. Since the classical reduction of the model-checking problem for the modal μ-calculus to parity games [3, 4] can be extended to hierarchically defined graphs (Thm. 3), we obtain PSPACE-completeness of the model-checking problem for the modal μ-calculus on hierarchically defined graphs. The upper bound generalizes a corresponding result for CTL/LTL from [1]. For a restricted class of hierarchically defined graphs we obtain the better upper bound of NP $\cap$ coNP for parity games, which leads to the same upper bound for the data complexity of the modal μ-calculus. In Section 5 we study least fixpoint logic (LFP) and its fragment monadic least fixpoint logic (MLFP) over hierarchically defined input structures. MLFP is still more expressive than the modal μ-calculus. It turns out that in most cases the complexity of the model-checking problem over hierarchically defined input structures becomes EXP. Our results are collected in Table 1 in Section 2. Proofs that are omitted due to space restrictions can be found in the full version [7].

2 Preliminaries

General notations Let $\equiv$ be an equivalence relation on a set A. For $a \in A$, $[a]_\equiv = \{b \in A \mid a \equiv b\}$ is 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 a function $f : A \to B$ let $\mathrm{ran}(f) = \{b \in B \mid \exists a \in A : f(a) = b\}$. For $C \subseteq A$ we define the restriction $f\restriction_C : C \to B$ by $f\restriction_C(c) = f(c)$ for all $c \in C$. 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 $n \in \mathbb{N}$ we denote by $\mathrm{id}_{\{1,\dots,n\}}$ the identity function over $\{1, \dots, n\}$.

Complexity theory We assume some basic background in complexity theory [20]. In particular, we assume that the reader is familiar with the classes P (deterministic polynomial time), NP (nondeterministic polynomial time), coNP (complements of problems in NP), PH (the polynomial time hierarchy), PSPACE (polynomial space), and EXP (deterministic exponential time). We will use alternating Turing-machines, see [20] for more details. 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 (i) $q \in F$, or (ii) $q \in Q_\exists$ and there exists a successor configuration of C that is accepting, or (iii) $q \in Q_\forall$ and every successor configuration of C is accepting. An input word w is accepted by M if the corresponding initial configuration is accepting. It is well known that PSPACE equals the class of all problems that can be solved on an alternating Turing-machine in polynomial time.

Relational structures and straight-line programs A *signature* is a finite set $\mathcal{R}$ of relational symbols, where each relational symbol $r \in \mathcal{R}$ has an associated arity n_r. A *(relational) structure* over the signature $\mathcal{R}$ is a tuple $\mathcal{A} = (A, (r^{\mathcal{A}})_{r\in\mathcal{R}})$, where A is a set (the universe of $\mathcal{A}$) and $r^{\mathcal{A}}$ is a relation of arity n_r over the set A, which interprets the relational symbol r. Usually, we denote the relation $r^{\mathcal{A}}$ with r as well. The size $|\mathcal{A}|$ of $\mathcal{A}$ is $|A| + \sum_{r\in\mathcal{R}} n_r \cdot |r^{\mathcal{A}}|$. For an equivalence relation relation $\equiv$ on A we define the quotient $\mathcal{A}/_\equiv = ([A]_\equiv, (\{(\pi_\equiv(a_1), \dots, \pi_\equiv(a_{n_r})) \mid (a_1, \dots, a_{n_r}) \in r^{\mathcal{A}}\})_{r\in\mathcal{R}})$. For $n \geq 0$, an n-pointed structure is a pair $(\mathcal{A}, \tau)$, where $\mathcal{A}$ is a structure with universe A and $\tau : \{1, \dots, n\} \to A$ is injective. The elements in $\mathrm{ran}(\tau)$ (resp. $A \setminus \mathrm{ran}(\tau)$) are also called *contact nodes* (resp. *internal nodes*). Let $G_i = (\mathcal{A}_i, \tau_i)$ be an n_i-pointed structure ($i \in \{1, 2\}$) over the signature $\mathcal{R}$, where A_i is the universe of $\mathcal{A}_i$ and $A_1 \cap A_2 = \emptyset$. We define the disjoint union $G_1 \oplus G_2$ as the $(n_1 + n_2)$-pointed structure $((A_1 \cup A_2, (r^{\mathcal{A}_1} \cup r^{\mathcal{A}_2})_{r\in\mathcal{R}}), \tau)$, where $\tau : \{1, \dots, n_1 + n_2\} \to A_1 \cup A_2$ with $\tau(i) = \tau_1(i)$ for all $1 \leq i \leq n_1$ and $\tau(i + n_1) = \tau_2(i)$ for all $1 \leq i \leq n_2$. Now let $G = (\mathcal{A}, \tau)$ be an n-pointed structure, where A is the universe of $\mathcal{A}$. For a permutation $f : \{1, \dots, n\} \to \{1, \dots, n\}$ define $\mathrm{rename}_f(G) = (\mathcal{A}, \tau \circ f)$. If $n \geq 1$, then $\mathrm{forget}(G) = (\mathcal{A}, \tau \restriction \{1, \dots, n-1\})$. Finally, if $n \geq 2$, then $\mathrm{glue}(G) = (\mathcal{A}/_\equiv, (\pi_\equiv \circ \tau) \restriction \{1, \dots, n-1\})$, where $\equiv$ is the smallest equivalence relation on A which contains the pair $(\tau(n), \tau(n-1))$. Note that the combination of rename_f and glue (resp. forget) allows to glue (resp. forget) arbitrary contact nodes.

Straight-line programs offer a succinct representation of large structures. A *straight-line program (SLP)* $\mathcal{S} = (X_i := t_i)_{1\leq i\leq l}$ is a sequence of definitions, where every right hand side t_i is either an n-pointed *finite* structure (for some n) or an expression of the form $X_j \oplus X_k$, $\mathrm{rename}_f(X_j)$, $\mathrm{forget}(X_j)$, or $\mathrm{glue}(X_j)$ with $j, k < i$ and $f : \{1, \dots, n\} \to \{1, \dots, n\}$ a permutation. Here, X_i is a formal variable. For every variable X_i, $\mathrm{type}(X_i)$ is inductively defined as follows: (i) if t_i is an n-pointed structure, then $\mathrm{type}(X_i) = n$, (ii) if $t_i = X_j \oplus X_k$, then $\mathrm{type}(X_i) = \mathrm{type}(X_j) + \mathrm{type}(X_k)$, (iii) if $t_i = \mathrm{rename}_f(X_j)$, then $\mathrm{type}(X_i) = \mathrm{type}(X_j)$, and (iv) if $t_i = \mathrm{op}(X_j)$ for $\mathrm{op} \in \{\mathrm{forget}, \mathrm{glue}\}$, then $\mathrm{type}(X_i) = \mathrm{type}(X_j) - 1$. The $\mathrm{type}(X_i)$-pointed finite structure $\mathrm{eval}(X_i)$ is inductively defined by: (i) if t_i is an n-pointed structure, then $\mathrm{eval}(X_i) = t_i$, (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 $\mathrm{op} \in \{\mathrm{rename}_f, \mathrm{forget}, \mathrm{glue}\}$, then $\mathrm{eval}(X_i) = \mathrm{op}(\mathrm{eval}(X_j))$. We define $\mathrm{eval}(\mathcal{S}) = \mathrm{eval}(X_l)$. The SLP $\mathcal{S}$ is *c-bounded* ($c \in \mathbb{N}$) if $\mathrm{type}(X_i) \leq c$ for all $1 \leq i \leq l$. Finally, the *size* $|\mathcal{S}|$ of $\mathcal{S}$ is l plus the size of all explicit n-pointed structures that appear as a right-hand side t_i. In [17] we used *hierarchical graph definitions* for the specification of large structures. A hierarchical graph definition can be translated in polynomial time into an SLP defining the same structure [7, 16].

Fixpoint logics Fix a signature $\mathcal{R}$. *First-order (FO) formulas* over the signature $\mathcal{R}$ are built from atomic formulas of the form $x = y$ and $r(x_1, \dots, x_{n_r})$ (where $r \in \mathcal{R}$ and $x, y, x_1, \dots, x_{n_r}$ are first-order variables ranging over elements of the universe) using boolean connectives and (first-order) quantifications over elements of the universe. *Least fixpoint logic* (LFP) extends FO by the definition of least fixpoints. For this, let us take a countably infinite set of *fixpoint variables*. Each fixpoint variable R has an associated arity n and ranges over n-ary relations over the universe. Fixpoint variables will be denoted by capital letters. Syntactically, LFP extends FO by the following formula building rule: Let $\varphi(\bar{x}, R, \bar{P}, \bar{y})$ be a formula of LFP. Here, $\bar{x}$ and $\bar{y}$ are repetition-free tuples of first-order variables, $\bar{P}$ is a repetition-free tuple of fixpoint variables, the arity of the fixpoint variable R is $|\bar{x}|$ (the length of the tuple $\bar{x}$), and R only occurs positively in φ (i.e., within an even number of negations). Then also $\mathrm{lfp}_{\bar{x},R}\ \varphi(\bar{x}, R, \bar{P}, \bar{y})$ is a formula of LFP. The semantics of the lfp-operator is the following: Let $\bar{b} \in A^{|\bar{y}|}$ and let $\bar{S}$ be a tuple of relations that is interpreting the tuple $\bar{P}$ of fixpoint variables. Since R only occurs positively in $\varphi(\bar{x}, R, \bar{P}, \bar{y})$, the function F_φ that maps $T \subseteq A^{|\bar{x}|}$ to $\{\bar{a} \in A^{|\bar{x}|} \mid \mathcal{A} \models \varphi(\bar{a}, T, \bar{S}, \bar{b})\}$ is monotonic. Hence, by the Knaster-Tarski fixpoint theorem, the smallest fixpoint $\mathrm{fix}(F_\varphi)$ exists. Now for $\bar{a} \in A^{|\bar{x}|}$ we have $\mathcal{A} \models [\mathrm{lfp}_{\bar{x},R}\ \varphi(\bar{x}, R, \bar{S}, \bar{b})](\bar{a})$ if and only if $\bar{a} \in \mathrm{fix}(F_\varphi)$. The greatest fixpoint operator can be defined as $\mathrm{gfp}_{\bar{x},R}\ \varphi(\bar{x}, R, \bar{P}, \bar{y}) = \neg \mathrm{lfp}_{\bar{x},R}\ \neg\varphi(\bar{x}, \neg R/R, \bar{P}, \bar{y})$, it defines the greatest fixpoint of F_φ.

Monadic least fixpoint logic (MLFP) is the fragment of LFP that only contains unary (i.e., monadic) fixpoint variables. The modal μ-calculus can be defined as a fragment of MLFP that is defined as follows. Formulas of the *modal μ-calculus* are interpreted over special relational structures that are called transition systems. Let $\mathcal{P}$ be a finite set of *atomic propositions*. A *transition system* (over $\mathcal{P}$) is a tuple $T = (Q, R, \lambda)$, where Q is a finite set of states, $R \subseteq Q \times Q$, and $\lambda : Q \to 2^{\mathcal{P}}$. Note that a state may be labeled with several atomic propositions. Clearly, T can be identified with the relational structure $\mathcal{A}_T = (Q, R, (\{q \in Q \mid p \in \lambda(q)\})_{p \in \mathcal{P}})$. This allows us to use SLPs in order to construct large transition systems. The set of *formulas* $\mathcal{F}_\mu(\mathcal{P})$ over $\mathcal{P}$ of the modal μ-calculus is defined by the following EBNF, where $p \in \mathcal{P}$ and X is a unary fixpoint variable: $\varphi ::= p \mid \neg p \mid X \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Diamond\varphi \mid \Box\varphi \mid \mu X.\varphi \mid \nu X.\varphi$ We define the semantics of a formula $\varphi \in \mathcal{F}_\mu(\mathcal{P})$ by translating it to an MLFP-formula $\|\varphi\|(x)$ over the signature $\{R\} \cup \mathcal{P}$, where R has rank 2, every $p \in \mathcal{P}$ has rank 1, and x is a first-order variable. The translation is done inductively:

$$\begin{aligned}
\|(\neg)p\|(x) &= (\neg)p(x) & \|X\|(x) &= X(x)\\
\|\varphi \wedge \psi\|(x) &= \|\varphi\|(x) \ \wedge\ \|\psi\|(x) & \|\varphi \vee \psi\|(x) &= \|\varphi\|(x) \ \vee\ \|\psi\|(x)\\
\|\Box\varphi\|(x) &= \forall y : R(x, y) \Rightarrow \|\varphi\|(y) & \|\Diamond\varphi\|(x) &= \exists y : R(x, y) \wedge \|\varphi\|(y)\\
\|\mu X.\varphi\|(x) &= [\mathrm{lfp}_{x,X} \|\varphi\|(x)](x) & \|\nu X.\varphi\|(x) &= [\mathrm{gfp}_{x,X} \|\varphi\|(x)](x)
\end{aligned}$$

For a transition system $T = (Q, R, \lambda)$, a state $q \in Q$, and a formula $\varphi \in \mathcal{F}_\mu(\mathcal{P})$, we write $(T, q) \models \varphi$ if $\mathcal{A}_T \models \|\varphi\|(q)$.

Table 1. Data and combined complexity for fixpoint logics

		explicit [4, 9, 21]	c**-bounded SLP**	**unrestricted SLP**
μ**-calc.**	**data**	P	P $\cdots$ NP $\cap$ coNP	PSPACE
	combined	P $\cdots$ NP $\cap$ coNP	PSPACE	PSPACE
MLFP	**data**	P	P $\cdots$ PH	EXP
	combined	EXP	EXP	EXP
LFP	**data**	P	EXP	EXP
	combined	EXP	EXP	EXP

The model checking problem for a logic $\mathcal{L}$ asks whether for a structure $\mathcal{A}$ and a sentence $\varphi \in \mathcal{L}$ we have $\mathcal{A} \models \varphi$. As already explained in the introduction, we will be interested in combined complexity (both, the formula and the structure belong to the input) and data complexity (the formula is fixed and only the structure belongs to the input), where the structure is represented via an SLP. Table 1 collects the known results as well as our new results concerning the (data and combined) complexity of the model-checking problem for the logics LFP, MLFP, and the modal μ-calculus. Only for the data complexity of MLFP and the modal μ-calculus on structures defined by c-bounded SLPs (for some fixed $c \in \mathbb{N}$) we do not obtain matching lower and upper bounds.

Parity games The modal μ-calculus has a close relationship to *parity games*, which are played between two players, called Adam and Eve, on a particular kind of relational structure, called a game graph. Let $C = \{0, \ldots, k\}$ ($k \in \mathbb{N}$) be a finite set of *priorities*. A *game graph* G over C is a tuple $G = (V, E, \rho)$ s.t. V is a finite set of nodes, $E \subseteq V \times C \times V$ is the set of labeled edges, and $\rho : V \to \{\mathrm{Eve}, \mathrm{Adam}\}$ assigns to every node v a player $\rho(v)$. Its *size* is defined by $|G| = |V| + |E|$. We define $\overline{\mathrm{Eve}} = \mathrm{Adam}$ and $\overline{\mathrm{Adam}} = \mathrm{Eve}$. The set of successor nodes of a given node $v \in V$ is $vE = \{u \in V \mid \exists c \in C : (v, c, u) \in E\}$. Note that we diverge from common conventions as in [4, 8, 19] since priorities are assigned to edges instead to nodes. This is no restriction when considering parity games. A sequence $\pi = v_0, c_0, v_1, c_1, \ldots \in V(CV)^\omega$ (resp. $\pi = v_0, c_0, v_1, \ldots, c_{n-1}, v_n \in V(CV)^*$) is an *infinite path* (resp. *finite path*) in G if for all $i \geq 0$ (resp. $0 \leq i \leq n-1$) we have $(v_i, c_i, v_{i+1}) \in E$. A finite path π is called *empty* if $\pi = v$ for some $v \in V$. The set of priorities occurring in π is denoted by $\mathrm{Occ}(\pi)$. For an infinite path π we denote with $\mathrm{Inf}(\pi) \subseteq \mathrm{Occ}(\pi)$ the set of those priorities that occur infinitely many times in the path π. Clearly, the game graph $G = (V, E, \rho)$ can be identified with the relational structure $(V, (\{(u, v) \mid (u, c, v) \in E\})_{c \in C}, \rho^{-1}(\mathrm{Eve}), \rho^{-1}(\mathrm{Adam}))$. This allows us to generate large game graphs using SLPs. Here we have to be careful with the glue-operation. If (G, τ) is an n-pointed relational structure, where G is the game graph $G = (V, E, \rho)$ — we call such a structure an n*-game graph* — then $\mathrm{glue}(G, \tau)$ is an $(n-1)$-game graph only if $n \geq 2$ and $\rho(\tau(n-1)) = \rho(\tau(n))$, i.e., the two nodes that are glued belong to the same player. Thus, glue is only a partial operation on n-game graphs.

Let $G = (V, E, \rho)$ be a game graph over the priorities $C = \{0, \ldots, k\}$ $(k \in \mathbb{N})$. A *play* is either an infinite path in G or a finite path in G that ends in a node v with $vE = \emptyset$ (i.e., a dead end). Eve (resp. Adam) *wins* an infinite play π if and only if $\max(\mathrm{Inf}(\pi)) \equiv 0 \mod 2$ (resp. $\max(\mathrm{Inf}(\pi)) \equiv 1 \mod 2$). Player $\sigma \in \{\mathrm{Eve}, \mathrm{Adam}\}$ *wins* a finite play π if and only if $\rho(v) = \overline{\sigma}$ for the last node v of π, i.e., the dead end, where π ends, belongs to player $\overline{\sigma}$. It is an important question whether Eve/Adam has the possibility to force the game to a play which she/he can win, i.e., if she/he has a winning-strategy. For parity games, so called memoryless strategies suffice. A *memoryless strategy* for player $\sigma \in \{\mathrm{Eve}, \mathrm{Adam}\}$ is a map $\mathscr{S}_\sigma : \{v \mid \rho(v) = \sigma, vE \neq \emptyset\} \to V$ s.t. $\mathscr{S}_\sigma(v) \in vE$ for all $v \in \{v \mid \rho(v) = \sigma, vE \neq \emptyset\}$. We say that a finite play $\pi = v_0, c_0, v_1, \ldots c_{n-1}, v_n$ (resp. an infinite play $\pi = v_0, c_0, v_1, \ldots$) is $\mathscr{S}_\sigma$-*confirm* w.r.t. a memoryless strategy $\mathscr{S}_\sigma$ if for all $0 \leq i \leq n-1$ (resp. for all $i \geq 0$) we have $\mathscr{S}_\sigma(v_i) = v_{i+1}$ whenever $\rho(v_i) = \sigma$. For $v \in V$ we call the memoryless strategy $\mathscr{S}_\sigma$ a *memoryless winning strategy for player σ from the node v* if player σ wins every $\mathscr{S}_\sigma$-confirm play which starts in v. The question whether a memoryless strategy $\mathscr{S}_\sigma$ for player σ is a winning strategy can be answered in deterministic polynomial time. With PARITY we denote the set of all triples (G, v, σ), where G is a game graph, v is a node of G, $\sigma \in \{\mathrm{Eve}, \mathrm{Adam}\}$, and there exists a memoryless winning strategy for player σ from v. The determinacy theorem for parity games [4] states that $(G, v, \sigma) \in \mathrm{PARITY}$ if and only if $(G, v, \overline{\sigma}) \notin \mathrm{PARITY}$. It implies that PARITY belongs to $\mathsf{NP} \cap \mathsf{coNP}$. By a result of [3, 4], the model-checking problem for the modal μ-calculus can be reduced to PARITY.

3 Parity games over SLP-defined graphs

In this section we will prove a PSPACE upper bound for parity games over game graphs that are given via SLPs. Our construction is inspired by [19], where parity games on graphs of bounded tree width are examined. For the following considerations, let us fix a set $C = \{0, \ldots, k\}$ $(k \in \mathbb{N})$ of priorities and let $G = (H, \tau)$ be an n-game graph over C with $H = (V, E, \rho)$.

Strategy reducts Let $W \subseteq \rho^{-1}(\mathrm{Eve}) \cap \mathrm{ran}(\tau)$ be a set of contact nodes that belong to Eve. An n-game graph G' is a *strategy reduct* of G w.r.t. W if G' can be obtained from G by (i) removing all outgoing edges for all $w \in W$, and (ii) keeping exactly one outgoing edge for all $w \in \rho^{-1}(\mathrm{Eve}) \setminus (W \cup \{v \mid vE = \emptyset\})$. Thus, a strategy reduct of G is the remainder of G by restricting G to a given strategy and making certain Eve-nodes to dead ends. Note that a strategy reduct is always defined w.r.t. a subset W of Eve-nodes and is not unique in general. The reason for making an Eve-node u to a dead end in G is the fact that u is a contact node which will be glued with another contact node u' from another n'-game graph G' in an SLP, and for u' an outgoing edge (as a part of the strategy for Eve on G') has already been guessed.

The reward function For a guessed strategy reduct G' of the potentially exponentially large n-game graph G we only store a polynomial amount of relevant information. More precisely for each pair of contact nodes $\tau(i)$ and $\tau(j)$ we only store the maximal priority along an optimal path for Adam from $\tau(i)$ to $\tau(j)$. In order to define this formally, we introduce the function $\mathrm{reward} : 2^C \setminus \{\emptyset\} \to C$, see also [19]. Let $B \subseteq C$, $B \neq \emptyset$:

$$\mathrm{reward}(B) = \begin{cases} \max(B \cap \{2n+1 \mid n \in \mathbb{N}\}) & \text{if } B \cap \{2n+1 \mid n \in \mathbb{N}\} \neq \emptyset \\ \min(B) & \text{else} \end{cases}$$

Intuitively, $\mathrm{reward}(B)$ is the best priority among B for Adam: if there is an odd priority in B, then the largest odd priority is the best for Adam. If there are only even priorities in B, then the smallest priority in B causes the smallest harm for Adam. For a set $\varPi \neq \emptyset$ of finite paths in G let $\mathrm{reward}(\varPi) = \mathrm{reward}(\{\max(\mathrm{Occ}(\pi)) \mid \pi \in \varPi\})$. The intuition behind this definition is the following: If G is a strategy reduct of G, then it is only Adam who can freely choose the next outgoing edge in G'. Hence, if $\varPi$ is the set of all paths in G' between two contact nodes $\tau(i)$ and $\tau(j)$, then, if Adam is smart, he will choose a path $\pi \in \varPi$ with $\max(\mathrm{Occ}(\pi)) = \mathrm{reward}(\varPi)$ when going from $\tau(i)$ to $\tau(j)$. Hence, we can replace the set of paths $\varPi$ by a single edge from $\tau(i)$ to $\tau(j)$ with priority $\mathrm{reward}(\varPi)$. For technical reasons we only put paths into $\varPi$ that do not visit any contact nodes except its start and end node. We call such paths $\tau\tau$-internal paths.

$(\tau)\tau$-internal paths For two contact nodes $v_0, v_n \in \mathrm{ran}(\tau)$ we call a *non-empty* finite path $\pi = v_0, c_0, v_1, \ldots, c_{n-1}, v_n$ a *$\tau\tau$-internal path* from v_0 to v_n if $v_1, \ldots, v_{n-1} \notin \mathrm{ran}(\tau)$. Note that $v_0 = v_n$ is allowed. We call a *non-empty play* π (i.e., either π is an infinite path or it ends in a dead end but is non-empty) a *τ-internal path* if its first node is a contact node but it never visits a contact node again. We will be only interested in τ-internal paths, which player Adam wins. For $1 \leq i, j \leq n$ we denote with $\varPi^{\tau}_{i,j}(G)$ the set of all $\tau\tau$-internal paths in G from $\tau(i)$ to $\tau(j)$. Note that an arbitrary path between two contact nodes can be split up into consecutive $\tau\tau$-internal paths. Similarly an arbitrary maximal path that begins in a contact node consists of a sequence of $\tau\tau$-internal paths possibly followed by a τ-internal path. Hence, we do not lose any information by only considering $(\tau)\tau$-internal paths.

The reduce operation Assume that G' is a strategy reduct of the n-game graph G. Then it is only player Adam who can choose any path in G'. Of course, there is no reason for player Adam to move from contact node $\tau(i)$ to contact node $\tau(j)$ along a path which is not optimal for him. Hence we can replace the set $\varPi^{\tau}_{i,j}(G)$ of all $\tau\tau$-internal paths from $\tau(i)$ to $\tau(j)$ by a single edge with priority $\mathrm{reward}(\varPi^{\tau}_{i,j}(G))$. The operation reduce is doing this for every pair of contact nodes. We define the reduce-operation on arbitrary n-game graphs, but later we will only apply it to strategy reducts: $\mathrm{reduce}(G)$ is the game graph $(\{1, \ldots, n\}, F, \varrho)$, where $\varrho(i) = \rho(\tau(i))$ for all $1 \leq i \leq n$ and $(i, p, j) \in F$ if and only if $\varPi^{\tau}_{i,j}(G) \neq \emptyset$ and $\mathrm{reward}(\varPi^{\tau}_{i,j}(G)) = p$. We identify $\mathrm{reduce}(G)$ with the n-game graph $((\{1, \ldots, n\}, F, \varrho), \mathrm{id}_{\{1,\ldots,n\}})$.

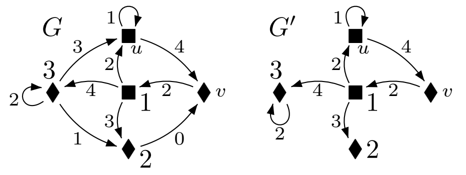
Lemma 1. *The operation* reduce *is polynomial time computable.*

Interfaces and realizability For a given variable X_i of an SLP $\mathcal{S}$, the $\mathrm{type}(X_i)$-game graph $\mathrm{eval}(X_i)$ may have exponential size in $|\mathcal{S}|$, and the same is true for some strategy reduct G' of $\mathrm{eval}(X_i)$. We will store all the relevant information about G' in a so called

n-interface of G', which can be stored in polynomial space. An *n-interface S* over the priorities C is a tuple $S = (\{1, \dots, n\}, F, \varrho, I, U)$ s.t. (i) $(\{1, \dots, n\}, F, \varrho)$ is a game graph over the priorities C, (ii) $I \subseteq \{1, \dots, n\}$, and (iii) $U \subseteq \varrho^{-1}(\mathrm{Eve})$ is a subset of the nodes which belong to Eve. The notion of an interface is inspired by the notion of a border from [19]. The n-interface S is *realized* by the n-game graph $G = (H, \tau)$ if there is a strategy reduct $G' = (H', \tau)$ of G w.r.t. $\tau(U)$ s.t. (i) $(\{1, \dots, n\}, F, \varrho) = \mathrm{reduce}(G')$, and (ii) $i \in I$ if and only if there is a τ-internal path π in G' which begins at $\tau(i)$ and which player Adam wins (recall that π has to be non-empty). We also say that G' is a *witness* that S is realized by G. By fi rst guessing a strategy reduct and then applying Lemma 1, we obtain:

Lemma 2. *The problem of checking, whether a given n-interface is realized by a given n-game graph, belongs to* NP.

Example 1. The following fi gure shows a 3-game graph G together with a strategy reduct G' w.r.t. $\{\tau(2)\}$ (node $\tau(i)$ is labeled with i and ♦-labeled (resp. ■-labeled) nodes belong to Eve (resp. Adam). The interface $S = (\{1, 2, 3\}, F, \varrho, I, U)$ with $I = \{1\}$ and $U = \{2\}$ is realized by G, and G' is a witness for this.



We have $1 \in I$, because the infi nite τ-internal path $\tau(1), 2, (u, 1)^\omega$ starts in node $\tau(1)$ in G' and Adam wins this path. The loop with priority 4 at node 1 in S exists due to the $\tau\tau$-internal path $\tau(1), 2, u, 4, v, 2, \tau(1)$ in G'.

Operations on interfaces Our PSPACE-algorithm will only manipulate n-interfaces instead of whole n-game graphs. In order to do this, we have to extend the operations $\oplus$, rename_f, forget, and glue on interfaces. The crucial correctness property is formalized in this section for arbitrary operations. In the following we restrict to n-game graphs $G = (H, \tau)$ such that every contact node $\tau(i)$ has at least one outgoing edge. This can be ensured by adding for a contact node $\tau(i)$ without outgoing edges an outgoing edge to a new internal node v, which is a dead end and which belongs to the same player as $\tau(i)$. This new edge has no influence on the winner of a parity game.

Let op be a partial operation, mapping a k-tuple $(G_1, \dots, G_k)$, where G_i is an n_i-game graph, to an n-game graph $\mathrm{op}(G_1, \dots, G_k)$. We say that op has a *faithful polynomial implementation (briefly FPI) on interfaces*, if there exists a partial *polynomial time computable* operation op^s, mapping a k-tuple $(S_1, \dots, S_k)$, where S_i is an n_i-interface, to an n-interface $\mathrm{op}(S_1, \dots, S_k)$ s.t. the following is true: Whenever $G = \mathrm{op}(G_1, \dots, G_k)$, where G_i is an n_i-game graph and G is an n-game graph, and S is an n-interface, then G realizes S if and only if there exist n_i-interfaces S_i $(1 \leq i \leq k)$ s.t. $S = \mathrm{op}(S_1, \dots, S_k)$ and G_i realizes S_i.

Lemma 3. *The operations* $\oplus$, rename_f, forget*, and* glue *have FPIs on interfaces.*

Proof. The operations $\oplus^s$ and rename^s_f are straight-forward: $\oplus^s$ builds the disjoint union of two interfaces, and rename^s_f renames the contact nodes according to the permutation f. Let us now describe the operation glue^s, the operation forget^s can be defined similarly. Let $S = (\{1,\dots,n\}, F, \varrho, I, U)$ be an n-interface. Then $\mathrm{glue}^s(S)$ is only defined if (i) $n \geq 2$, (ii) $\varrho(n) = \varrho(n-1)$ (thus, node $n-1$ and n belong to the same player and can actually be glued), and (iii) if $\varrho(n) = \varrho(n-1) = \mathrm{Eve}$ then $n-1 \in U$ or $n \in U$. Then $\mathrm{glue}^s(S) = (\{1,\dots,n-1\}, F', \varrho', I', U')$, where:

- $(\{1,\dots,n-1\}, F', \varrho') = \mathrm{reduce}(\mathrm{glue}(\{1,\dots,n\}, F, \varrho))$.
- $I' = I \setminus \{n\} \cup \{n-1\}$ if ($n-1 \in I$ or $n \in I$), otherwise $I' = I$.
- $U' = U \setminus \{n\}$ if $n-1, n \in U$, otherwise $U' = U \setminus \{n-1, n\}$.

The intuition behind this definition is the following. Assume that the n-interface S is realized by an n-game graph $G = (H, \tau)$ and let G' be a witness for this. We want to define $\mathrm{glue}^s(S) = (\{1,\dots,n-1\}, F', \varrho', I', U')$ in such a way that $\mathrm{glue}^s(S)$ is realized by $\mathrm{glue}(G)$ and moreover $\mathrm{glue}(G')$ is a witness for this. Note that by assumption (i)–(iii), $\mathrm{glue}(G')$ is in fact a strategy reduct of $\mathrm{glue}(G)$. In order to determine the maximal priority of an optimal path for Adam from $\tau(i)$ to $\tau(j)$ in $\mathrm{glue}(G')$, it suffices to look at the $(n-1)$-game graph $K = \mathrm{glue}(\{1,\dots,n\}, F, \varrho)$, i.e., to calculate $\mathrm{reduce}(K)$. This graph will be therefore $(\{1,\dots,n-1\}, F', \varrho')$. Note that in K, there may be more than one edge between two contact nodes. By applying reduce to K we select the optimal edge for player Adam between two contact nodes. Finally, if $n-1 \in I$ or $n \in I$, i.e., there exists a τ-internal path in G' that starts in $\tau(n-1)$ or in $\tau(n)$ and which player Adam wins, then we can be sure that there exists a τ-internal path in $\mathrm{glue}(G')$ that starts in $\tau(n-1)$ and which player Adam wins. Here it is important that τ-internal paths are non-empty. Hence, we put $n-1$ into I'. ☐

Parity games over SLP-defined graphs We are now ready to prove an upper bound of PSPACE for the parity game problem on graphs that are represented by SLPs:

Theorem 1. *For a given SLP $\mathcal{S} = (X_i := t_i)_{1 \leq i \leq l}$, where $\mathrm{eval}(\mathcal{S}) = (G, \tau)$ is a 1-game graph, we can decide in* PSPACE*, whether $(G, \tau(1), \mathrm{Eve}) \in \mathrm{PARITY}$.*

Proof. W.l.o.g. we can assume that node $\tau(1)$ belongs to Eve and that $\tau(1)$ has no incoming edge; this property can be easily enforced by adding a new node. Due to this convention, we have $(G, \tau(1), \mathrm{Eve}) \in \mathrm{PARITY}$ if and only if $\mathrm{eval}(G)$ realizes the interface $S_l = (\{1\}, \emptyset, [1 \mapsto \mathrm{Eve}], \emptyset, \emptyset)$. We present the algorithm in form of the following procedure $\mathcal{P}$, which works on a polynomial time bounded alternating Turing machine; $(Q_\forall)$ (resp. $(Q_\exists)$) indicates that the machine branches universally (resp. existentially). Procedure $\mathcal{P}$ has two parameters, the current line i of the SLP and a $\mathrm{type}(X_i)$-interface S_i, and it returns `true` if and only if S_i is realized by $\mathrm{eval}(X_i)$. At the beginning we call $\mathcal{P}$ with the parameter (l, S_l).

```
procedure  $\mathcal{P}(i \in \{1,\dots,l\}, S_i)$ return boolean is
  if $t_i$ is a $\mathrm{type}(X_i)$-game graph then return ($t_i$ realizes $S_i$)          $(*)$
  elseif $t_i = \mathrm{op}(X_{i_1},\dots,X_{i_k})$ then
    $(Q_\exists)$: for $1 \leq j \leq k$ guess $\mathrm{type}(X_{i_j})$-interfaces $S_{i_j}$ s.t. $S_i = \mathrm{op}^s(S_{i_1},\dots,S_{i_k})$
    $(Q_\forall)$: return $\bigwedge_{1 \leq j \leq k} \mathcal{P}(i_j, S_{i_j})$
  endif
```

The correctness of the algorithm follows easily by induction on the index $i \in \{1, \dots, l\}$. For the alternating polynomial time bound note that: (i) the test whether t_i realizes S_i in line $(*)$ is in NP by Lemma 2 and (ii) each of the operations op^s is computable in polynomial time by Lemma 3 and the definition of an FPI. □

By the following theorem, we can improve the PSPACE upper bound from Thm. 1 to NP ∩ coNP, when we restrict to c-bounded SLPs for some fixed constant c.

Theorem 2. *Let $c \in \mathbb{N}$ be a fixed constant. The problem of checking $(G, \tau(1), \mathrm{Eve}) \in \mathrm{PARITY}$ for a given c-bounded SLP $\mathcal{S} = (X_i := t_i)_{1 \le i \le l}$, where $\mathrm{eval}(\mathcal{S}) = (G, \tau)$ is a 1-game graph, belongs to* NP ∩ coNP.

Proof. By the determinacy theorem it suffices to prove membership in NP. The main idea is to guess for all $1 \le i \le l$ a set of $\mathrm{type}(X_i)$-interfaces M_i. Note that for the representation of a single interface $c^2 \log |C| + 2c$ bits suffice, where C is the set of priorities used in the SLP $\mathcal{S}$. Thus, every M_i contains at most $|C|^{c^2} 2^{2c}$ many interfaces. Hence, since c is a constant, we can guess in polynomial time the set $\bigcup_{1 \le i \le l} M_i$ of interfaces. Then we check whether for all $1 \le i \le l$ the set M_i is a subset of the set of interfaces which are realized by $\mathrm{eval}(X_i)$. In case t_i is an n-game graph we can do this in NP by Lemma 2. If $t_i = \mathrm{op}(X_{i_1}, \dots, X_{i_k})$, then one has to check, whether for every $S_i \in M_i$ there are $S_{i_j} \in M_{i_j}$ $(1 \le j \le k)$ s.t. $S_i = \mathrm{op}^s(S_{i_1}, \dots, S_{i_k})$. □

4 The modal μ-calculus over SLP-defined graphs

In this section, we show that both the data and combined complexity of the modal μ-calculus over transition systems that are represented by SLPs is PSPACE-complete. The upper bound extends [1, Thm. 9] concerning CTL. Note that a translation of the modal μ-calculus into MSO and an application of the MSO-model-checking algorithm from [17] leads to a higher upper bound, namely within the exponential time hierarchy (already for data complexity). For c-bounded SLPs we obtain an upper bound of NP ∩ coNP for the data complexity, whereas the combined complexity remains PSPACE. For the upper bounds we use a reduction to parity games, which is analogous to the corresponding reduction for explicitly given input graphs [3, 4]:

Theorem 3. *The following problem can be calculated in polynomial time:*
INPUT: A c-bounded SLP $\mathcal{S}_t$ defining a transition system $\mathrm{eval}(\mathcal{S}_t)$, a state q of $\mathrm{eval}(\mathcal{S}_t)$, and a sentence φ of the modal μ-calculus having exactly k subformulas.
OUTPUT: A $(c \cdot k)$-bounded SLP $\mathcal{S}_g$ defining a game graph $\mathrm{eval}(\mathcal{S}_g)$ and a node v of $\mathrm{eval}(\mathcal{S}_g)$ s.t. $(\mathrm{eval}(\mathcal{S}_t), q) \models \varphi$ if and only if $(\mathrm{eval}(\mathcal{S}_g), v, \mathrm{Eve}) \in \mathrm{PARITY}$.

Corollary 1. *The following problem is* PSPACE*-complete:*
INPUT: An SLP $\mathcal{S}$ defining a transition system $\mathrm{eval}(\mathcal{S})$, a state q of $\mathrm{eval}(\mathcal{S})$, and a sentence φ of the modal μ-calculus.
QUESTION: $(\mathrm{eval}(\mathcal{S}), q) \models \varphi$?
Moreover: (i) the above problem is already PSPACE*-complete when restricted to c-bounded SLPs (for a suitably large c), and (ii) there exists already a fixed sentence of the modal μ-calculus for which the above problem is* PSPACE*-complete.*

Proof. The upper bound follows from Thm. 1 and 3. For the lower bounds, we use two results from [1]: The combined complexity of CTL for hierarchical state machines is PSPACE-complete [1, Thm. 9]; recall that CTL is a fragment of the modal μ-calculus. It is easy to see that the hierarchical state machines from the proof of [1, Thm. 9] can be generated by a 5-bounded SLP, which gives us (i). Moreover, there is already a fixed CTL-sentence s.t. the model-checking problem for hierarchical state machines (and thus SLPs) is PSPACE-complete [1, Thm. 11]. This implies (ii). $\square$

When we restrict both to c-bounded SLPs and to a fixed sentence φ, then we obtain a better upper bound:

Corollary 2. *Let $c \in \mathbb{N}$ be a fixed constant and φ be a fixed sentence of the modal μ-calculus. The problem of checking $(\mathrm{eval}(\mathcal{S}_t), q) \models \varphi$ for a given c-bounded SLP $\mathcal{S}_t$ (s.t. $\mathrm{eval}(\mathcal{S}_t)$ is a transition system) and a state q of $\mathrm{eval}(\mathcal{S}_t)$ belongs to* NP $\cap$ coNP.

Proof. If φ has k many subformulas, then the SLP $\mathcal{S}_g$ from Thm. 3 is $(c \cdot k)$-bounded. Since φ and φ are fixed, $c \cdot k$ is a fixed constant. The corollary follows from Thm. 2. $\square$

5 LFP and MLFP over SLP-defined graphs

In this section we state our results concerning MLFP and LFP. An upper bound for the most general case (combined complexity of LFP) is given by the next theorem; recall that EXP is also the combined complexity of LFP for explicitly given structures.

Theorem 4. *It can be checked in* EXP*, whether $\mathrm{eval}(\mathcal{S}) \models \varphi$ for a given SLP $\mathcal{S}$ and a given LFP-sentence φ.*

Only for the data complexity of MLFP we obtain a better upper bound. MLFP is a fragment of MSO (monadic second order logic). Since for every fixed MSO-sentence φ and every fixed constant c the model-checking problem for φ on structures represented by c-bounded SLPs belongs to the polynomial time hierarchy PH [17, Thm. 6.3], we obtain:

Theorem 5. *For every fixed MLFP sentence φ and every fixed constant $c \in \mathbb{N}$, the problem of checking $\mathrm{eval}(\mathcal{S}) \models \varphi$ for a given c-bounded SLP $\mathcal{S}$ belongs to* PH.

Finally, we state several EXP lower bounds. Together with Thm. 4 we get the EXP completeness results in Table 1. We start with the data complexity of LFP:

Theorem 6. *There is a fixed LFP-sentence φ s.t. it is* EXP*-hard to check whether $\mathrm{eval}(\mathcal{S}) \models \varphi$ for a given 4-bounded SLP $\mathcal{S}$.*

If we do not restrict to c-bounded hierarchical graph definitions, then an EXP lower bound can be also shown for the data complexity of MLFP:

Theorem 7. *There exists a fixed MLFP-sentence φ s.t. it is* EXP*-hard to check whether $\mathrm{eval}(\mathcal{S}) \models \varphi$ for a given SLP $\mathcal{S}$.*

For the combined complexity of MLFP, we can derive an EXP lower bound also for the c-bounded case:

Theorem 8. *It is* EXP*-hard to check $\mathrm{eval}(\mathcal{S}) \models \varphi$ for a given 3-bounded SLP $\mathcal{S}$ and a given MLFP-sentence φ.*

References

1. R. Alur and M. Yannakakis. Model checking of hierarchical state machines. *ACM Trans. Program. Lang. Syst.*, 23(3):273–303, 2001.
2. S. Dziembowski. Bounded-variable fixpoint queries are PSPACE-complete. In *Proc. CSL'96*, LNCS 1258, pages 89–105. Springer, 1996.
3. E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *Proc. FOCS'91*, pages 132–142. IEEE Computer Society Press, 1991.
4. E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model checking for the μ-calculus and its fragments. *Theor. Comput. Sci.*, 258(1-2):491–522, 2001.
5. E. A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *Proc. LICS'86*, pages 267–278. IEEE Computer Society Press, 1986.
6. 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.
7. S. Göller and M. Lohrey. Fixpoint logics on hierarchical structures. Tech. Rep. 2005/3, University of Stuttgart, Germany, 2005. ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/TR-2005-04/.
8. E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infi nite Games*. LNCS 2500. Springer, 2002.
9. N. Immerman. Relational queries computable in polynomial time. *Inf. Control*, 68(1–3):86–104, 1986.
10. M. Jurdziński. Deciding the winner in parity games is in UP and co-UP. *Inf. Process. Lett.*, 68(3):119–124, 1998.
11. M. Jurdziński. Small progress measures for solving parity games. In *Proc. STACS 2000*, LNCS 1770, pages 290–301. Springer, 2000.
12. T. Lengauer. Hierarchical planarity testing algorithms. *J. Assoc. Comput. Mach.*, 36(3):474–509, 1989.
13. T. Lengauer and K. W. Wagner. The correlation between the complexities of the nonhierarchical and hierarchical versions of graph problems. *J. Comput. Syst. Sci.*, 44:63–93, 1992.
14. T. Lengauer and E. Wanke. Efficient solution of connectivity problems on hierarchically defined graphs. *SIAM J. Comput.*, 17(6):1063–1080, 1988.
15. L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.
16. M. Lohrey. Model-checking hierarchical graphs. Tech. Rep. 2005/1, University of Stuttgart, Germany, 2005. ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/TR-2005-1/.
17. M. Lohrey. Model-checking hierarchical structures. In *Proc. LICS 2005*, pages 168–177. IEEE Computer Society Press, 2005.
18. M. V. Marathe, H. B. Hunt III, R. E. Stearns, and V. Radhakrishnan. Approximation algorithms for PSPACE-hard hierarchically and periodically specified problems. *SIAM J. Comput.*, 27(5):1237–1261, 1998.
19. J. Obdržálek. Fast mu-calculus model checking when tree-width is bounded. In *CAV'03*, LNCS 2725, pages 80–92. Springer, 2003.
20. C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.
21. M. Y. Vardi. The complexity of relational query languages (extended abstract). In *Proc. STOC 1982*, pages 137–146. ACM Press, 1982.
22. M. Y. Vardi. On the complexity of bounded-variable queries. In *Proc. PODS 1995*, pages 266–276. ACM Press, 1995.