Fixpoint logics over hierarchical structures

Stefan Göller and Markus Lohrey[*]
Universität Leipzig, Institut für Informatik,
{lohrey,goeller}@informatik.uni-leipzig.de

April 2, 2009

Abstract

Hierarchical graph definitions allow a modular description of graphs using modules for the specification of repeated substructures. Beside this modularity, hierarchical graph definitions also allow to specify graphs of exponential size using polynomial size descriptions. In many cases, this succinctness increases the computational complexity of decision problems. In this paper, the model-checking problem for the modal μ-calculus and (monadic) least fixpoint logic on hierarchically defined input graphs is investigated. In order to analyze the modal μ-calculus, parity games on hierarchically defined input graphs are investigated. Precise upper and lower complexity bounds are derived. A restriction on hierarchical graph definitions that leads to more efficient model-checking algorithms is presented.

1 Introduction

A hierarchical graph definition specifies 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 [21] in the context of VLSI design. Formally, hierarchical graph definitions can be seen as hyperedge replacement graph grammars [11, 13] that generate precisely one graph. Specific algorithmic problems (e.g. reachability, planarity, circuit-value, 3-colorability) on hierarchically defined graphs are studied in [19, 20, 21, 27, 28, 29].

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 over hierarchically defined graphs. LFP is the extension of classical first-order logic that allows the definition of least fixpoints of arbitrary arity [22]. MLFP is the fragment of LFP where only monadic fixpoints can be defined. Finally, the modal μ-calculus is the fragment of MLFP that is obtained from classical modal logic extended by a monadic

[*]The authors are supported by the DFG research project GELO

fixpoint operator. The model-checking problem for some logic (e.g. LFP or MLFP) 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*. For the purpose of proving upper complexity bounds we will use the related formalism of *straight-line programs*, see also [23]. The term "straight-line program" is used, because a straight-line program is just a sequence of instructions. The left-hand side of each instruction is a variable and the right-hand side is either an explicitly given graph or consists of an elementary operation (a graph operation in our context) applied to previously defined variables. The term "straight-line program" is also used in other contexts, e.g. for hierarchically defined strings [32] or trees [24]. Every hierarchical graph definition can be transformed in polynomial time into a straight-line program that generates the same structure, see [5, 23]. A graph that is represented by a hierarchical graph definition or a straight-line program is called a *hierarchically defined graph* in the following.

LFP and its fragments MLFP and the modal μ-calculus found many applications in database theory, finite model theory, and verification. The interested reader is referred to the text books [7, 22]. 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: [9, 10, 14, 15, 16, 33, 34, 35]. Concerning hierarchically defined graphs, in [1] the complexity of the temporal logics LTL and CTL on 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 both straight-line programs generalize hierarchical state machines and CTL is efficiently translatable into the modal μ-calculus, our work is a natural extension of [1]. Moreover, our work extends the previous paper [23] of the second author, where the model-checking problem of first-order logic, monadic second-order logic, and full second-order logic on hierarchically defined graphs was studied.

Our investigation of model-checking problems for hierarchically defined graphs follows a methodology introduced by Vardi [34]. 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 only consider data and combined complexity for hierarchically defined graphs.

After introducing the necessary concepts in Section 2 we study parity games on hierarchically defined graphs in Section 3. Parity games are the main tool for most model-checking algorithms for the modal μ-calculus. The main result of this paper states that the winner of a parity game on a hierarchically defined graph can be determined in PSPACE. Our PSPACE-algorithm is inspired by Obdržálek's polynomial time algorithm for parity games on graphs of bounded tree width [30]. For the restricted class of c-bounded straight-line programs (where $c \in \mathbb{N}$ is some fixed constant; c-boundedness roughly means that a module may refer to at most c many other nodes) we obtain the better upper bound of $\mathsf{NP} \cap \mathsf{coNP}$ for parity games.

In Section 4 we show that the classical polynomial time reduction of the model-checking problem for the modal μ-calculus to parity games [8, 9] can be extended to hierarchically defined graphs. Together with a PSPACE lower bound from [1] for CTL over hierarchical state machines we obtain PSPACE-completeness of the model-checking problem for the modal μ-calculus on hierarchically defined graphs. Note that the PSPACE upper bound generalizes the corresponding result for CTL from [1].

A hierarchical graph definition can be viewed as a pushdown automaton, where the stack height is bounded by some polynomially large number (which is the maximal nesting depth in the hierarchical graph definition). Under this viewpoint, a hierarchically defined graph corresponds to the transition graph of a stack bounded pushdown automaton. It is therefore interesting to compare our complexity results with those for (arbitrary) pushdown automata. Note that pushdown graphs (i.e. transition graphs of pushdown automata) are in general infinite. Computing the winner in a parity game on a pushdown graph is EXPTIME-complete [17, 37]. It follows that modal μ-calculus model-checking is EXPTIME-complete for pushdown graphs. The EXPTIME lower bound already holds for CTL [36].

In Section 5 we study least fixpoint logic (LFP) and its fragment monadic least fixpoint logic (MLFP) on hierarchically defined input graphs. MLFP is still more expressive than the modal μ-calculus. It turns out that in most cases the complexity of the model-checking problem on hierarchically defined input graphs becomes EXPTIME-complete. Only for the data complexity of MLFP on graphs given by c-bounded (for some fixed c) straight-line programs we obtain a polynomial time algorithm. Note that this is the same complexity as for explicitly given input graphs [14]. Our results for model-checking problems are collected in Table 1 at the end of Section 2.5 together with the known results for explicitly given input structures.

2 Preliminaries

2.1 General notations

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 a function $f : A \to B$ let $\mathrm{ran}(f) = \{b \in B \mid \exists a \in A : f(a) = b\}$, and for every $b \in B$, let $f^{-1}(b) = \{a \in A \mid f(a) = b\}$. 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 a set A, we denote by id_A the identity function over A.

2.2 Complexity theory

We assume that the reader has some basic background in complexity theory [31]. 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), PSPACE (polynomial space), and $\mathsf{EXPTIME}$ (deterministic exponential time). Several times we will use alternating Turing-machines, see [4] 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 $\{q_f\}$ (the accepting state). A configuration C with current state q is *accepting*, if

- $q = q_f$, 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.

It is well known that PSPACE (resp. $\mathsf{EXPTIME}$) equals the class of all problems that can be solved on an alternating Turing-machine in polynomial time (resp. polynomial space). The levels of the *polynomial time hierarchy* are defined as follows: Let $k \geq 1$. Then $\Sigma^{\mathrm{p}}_{\mathrm{k}}$ 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 $Q_\exists$. The polynomial time hierarchy is $\mathsf{PH} = \bigcup_{k \geq 1} \Sigma^{\mathrm{p}}_{\mathrm{k}}$. Note that $\Sigma^{\mathrm{p}}_1 = \mathsf{NP}$.

2.3 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 $n_r \in \mathbb{N}$. 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 $r^{\mathcal{A}}$ also with r. The size $|\mathcal{A}|$ of $\mathcal{A}$ is $|A| + \sum_{r \in \mathcal{R}} |r^{\mathcal{A}}| \cdot n_r$. For an equivalence relation relation $\equiv$ on A we define the quotient $\mathcal{A}/_{\equiv} = ([A]_{\equiv}, (r^{\mathcal{A}}/_{\equiv})_{r \in \mathcal{R}})$, where $r^{\mathcal{A}}/_{\equiv} = \{(\pi_{\equiv}(a_1), \ldots, \pi_{\equiv}(a_{n_r})) \mid (a_1, \ldots, a_{n_r}) \in r^{\mathcal{A}}\}$. For two relational structures $\mathcal{A}_1$ and $\mathcal{A}_2$ over the same signature $\mathcal{R}$ and with disjoint universes A_1 and A_2, respectively, we define the disjoint union $\mathcal{A}_1 \oplus \mathcal{A}_2 = (A_1 \cup A_1, (r^{\mathcal{A}_1} \cup r^{\mathcal{A}_2})_{r \in \mathcal{R}})$.

For $n \geq 0$, an *n-pointed structure* is a pair $(\mathcal{A}, \tau)$, where $\mathcal{A}$ is a structure with universe A and $\tau : \{1, \ldots, n\} \to A$ is injective. The elements in $\mathrm{ran}(\tau)$ (resp. $A \backslash \mathrm{ran}(\tau)$) are also called *contact nodes* (resp. *internal nodes*). In diagrams, the i-th contact node will be labeled with

i, set in boldface font. We now define several operations on n-pointed structures, see also [5]. 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 $(\mathcal{A}_1 \oplus \mathcal{A}_2, \tau)$, where $\tau : \{1,\ldots,n_1+n_2\} \to A_1 \cup A_2$ with $\tau(i) = \tau_1(i)$ for all $1 \le i \le n_1$ and $\tau(i+n_1) = \tau_2(i)$ for all $1 \le i \le n_2$. Now let $G = (\mathcal{A}, \tau)$ be an n-pointed structure, where A is the universe of $\mathcal{A}$. For a bijective mapping $f : \{1,\ldots,n\} \to \{1,\ldots,n\}$ define $\text{rename}_f(G) = (\mathcal{A}, \tau \circ f)$. If $n \ge 1$, then $\text{forget}(G) = (\mathcal{A}, \tau \restriction \{1,\ldots,n-1\})$. Finally, if $n \ge 2$, then $\text{glue}(G) = (\mathcal{A}/_{\equiv}, (\pi_{\equiv} \circ \tau) \restriction \{1,\ldots,n-1\})$, 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 is a sequence of operations on n-pointed structures. These operations allow the disjoint union, the rearrangement, the forgetting, and the gluing of its contact nodes. More formally, a *straight-line program (SLP)* $\mathcal{S} = (X_i := t_i)_{1 \le i \le l}$ (over the signature $\mathcal{R}$) is a sequence of definitions, where the right hand side t_i of the assignment $X_i := t_i$ is either an n-pointed *finite* structure (over the signature $\mathcal{R}$) for some n or an expression of the form $X_j \oplus X_k$, $\text{rename}_f(X_j)$, $\text{forget}(X_j)$, or $\text{glue}(X_j)$, where $1 \le j,k < i \le l$ and $f : \{1,\ldots,n\} \to \{1,\ldots,n\}$ is a permutation. Here, the X_i are formal variables. For every variable X_i the *rank* $\text{rank}(X_i)$ is inductively defined as follows:

- If t_i is an n-pointed structure, then $\text{rank}(X_i) = n$.
- If $t_i = X_j \oplus X_k$, then $\text{rank}(X_i) = \text{rank}(X_j) + \text{rank}(X_k)$.
- If $t_i = \text{rename}_f(X_j)$, then $\text{rank}(X_i) = \text{rank}(X_j)$ and we require that f is a permutation on $\{1,\ldots,\text{rank}(X_j)\}$.
- If $t_i = \text{op}(X_j)$ for $\text{op} \in \{\text{forget}, \text{glue}\}$, then $\text{rank}(X_i) = \text{rank}(X_j) - 1$ and we require that $\text{rank}(X_j) > 0$.

The $\text{rank}(X_i)$-pointed finite structure $\text{eval}(X_i)$ is inductively defined by:

- If t_i is an n-pointed structure G, then $\text{eval}(X_i) = G$.
- If $t_i = X_j \oplus X_k$, then $\text{eval}(X_i) = \text{eval}(X_j) \oplus \text{eval}(X_k)$.
- If $t_i = \text{op}(X_j)$ for $\text{op} \in \{\text{rename}_f, \text{forget}, \text{glue}\}$, then $\text{eval}(X_i) = \text{op}(\text{eval}(X_j))$.

We define $\text{eval}(\mathcal{S}) = \text{eval}(X_l)$. The SLP $\mathcal{S}$ is called *c-bounded* ($c \in \mathbb{N}$) if $\text{rank}(X_i) \le c$ for all $1 \le i \le l$. Finally, the *size* $|\mathcal{S}|$ of the SLP $\mathcal{S}$ is defined as l plus the size of all explicit n-pointed structures that appear in a right-hand side t_i.

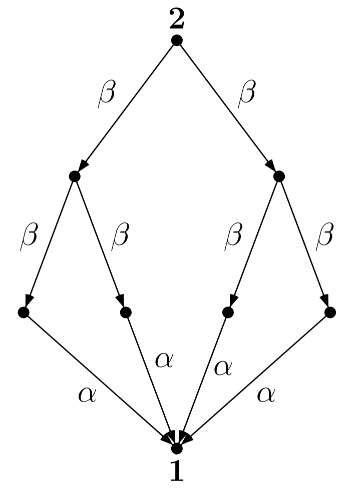
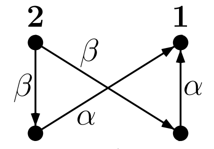
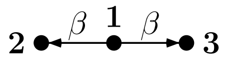


Figure 1: The graph eval($\mathcal{S}$) for the SLP from Example 2.1

Example 2.1. *In Figure 1, the* 2*-pointed structure* eval($\mathcal{S}$)*, where* $\mathcal{S}$ *is the SLP consisting of the following operations, is shown:*

$X_1 :=$ 

$X_2 :=$ 

$X_3 := X_2 \oplus X_1$

$X_4 := X_3 \oplus X_1$ *(this is a* 7*-pointed graph)*

$X_5 := \mathrm{rename}_{f_1}(X_4)$*, with* $f_1 : 3 \mapsto 6, 6 \mapsto 3, 2 \mapsto 4, 4 \mapsto 2, i \mapsto i$ *for* $i \in \{1, 5, 7\}$

$X_6 := \mathrm{glue}(X_5)$ *(this is a* 6*-pointed graph)*

$X_7 := \mathrm{forget}(X_6)$

$X_8 := \mathrm{glue}(X_7)$ *(this is a* 4*-pointed graph)*

$X_9 := \mathrm{forget}(X_8)$

$X_{10} := \mathrm{glue}(X_9)$ *(this is a* 2*-pointed graph)*

$X_{11} := \mathrm{rename}_{f_4}(X_{10})$*, with* $f_4 : 1 \mapsto 2, 2 \mapsto 1$

In [23] we used *hierarchical graph definitions* for the specification of large structures. Hierarchical graph definitions will be introduced in Section 5.2. Every hierarchical graph definition can be transformed in polynomial time into an SLP that generates the same structure [5, 23].

2.4 Transition systems

Formulas of the modal μ-calculus are interpreted on 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 (i) Q is a finite set of states, (ii) $R \subseteq Q \times Q$, and (iii) $\lambda : Q \to 2^{\mathcal{P}}$. Thus, a state may be labeled with several atomic propositions. An *initialized transition system over* $\mathcal{P}$ is a pair (T, q_{init}), where $T = (Q, R, \lambda)$ is a transition system over $\mathcal{P}$ and $q_{\text{init}} \in Q$ is the *initial state.* 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. Note that if two states q_1 and q_2 are glued via the glue-operation, where the set $P_i \subseteq \mathcal{P}$ is associated with state q_i, then $P_1 \cup P_2$ is associated with the resulting state.

2.5 Least fixpoint logic

More details concerning the material in this section can be found in [7, 22]. Let us fix a signature $\mathcal{R}$ for the further discussion. *First-order (FO) formulas* over the signature $\mathcal{R}$ are built from atomic formulas of the form $x = y$ and $r(x_1, \ldots, x_{n_r})$ (where $r \in \mathcal{R}$ and $x, y, x_1, \ldots, x_{n_r}$ 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. Hence, atomic formulas of the form $R(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are first-order variables, are allowed in LFP-formulas. 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 tuples of first-order variables with $\bar{x}$ repetition-free, $\bar{P}$ is a 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 $\text{lfp}_{\bar{x},R}\ \varphi(\bar{x}, R, \bar{P}, \bar{y})$ 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 Fixpointtheorem, the smallest fixpoint $\text{fix}(F_\varphi)$ exists. Now for $\bar{a} \in A^{|\bar{x}|}$ we have $\mathcal{A} \models [\text{lfp}_{\bar{x},R}\ \varphi(\bar{x}, R, \bar{S}, \bar{b})](\bar{a})$ if and only if $\bar{a} \in \text{fix}(F_\varphi)$. The greatest fixpoint operator can be defined as $\text{gfp}_{\bar{x},R}\ \varphi(\bar{x}, R, \bar{P}, \bar{y}) = \neg\text{lfp}_{\bar{x},R}\ \neg\varphi(\bar{x}, \neg R/R, \bar{P}, \bar{y})$, its semantics can be defined in the same way as the lfp-operator, except that we refer to the greatest fixpoint of the function F_φ.

Monadic least fixpoint logic (MLFP) is the fragment of LFP that only contains unary (i.e., monadic) fixpoint variables.

The *rank* $\text{rank}(\varphi)$ of an MLFP formula (we will not need this notion for general LFP-formulas) is inductively defined as follows:

$$\begin{aligned}
\mathrm{rank}(\varphi) &= 0 \text{ if } \varphi \text{ is atomic}\\
\mathrm{rank}(\neg\varphi) &= \mathrm{rank}(\varphi)\\
\mathrm{rank}(\varphi \,\mathrm{op}\, \psi) &= \max\{\mathrm{rank}(\varphi), \mathrm{rank}(\psi)\} \text{ for op} \in \{\wedge, \vee\}\\
\mathrm{rank}(Q\,x\,\varphi) = \mathrm{rank}(\mathrm{lfp}_{x,P}\ \varphi(x, P, \bar{P}, \bar{y})) &= \mathrm{rank}(\varphi) + 1 \text{ for } Q \in \{\exists, \forall\}
\end{aligned}$$

It is well-known that for every $k \geq 1$, there are only finitely many pairwise nonequivalent formulas of rank at most k over the signature $\mathcal{R}$. This value only depends on k and the signature $\mathcal{R}$, see [18] for an explicit estimation. The MLFP_k-theory of a structure $\mathcal{A}$, briefly $\mathrm{MLFP}_k(\mathcal{A})$, consists of all MLFP-sentences of rank at most k over the signature of $\mathcal{A}$ that are true in $\mathcal{A}$; by the previous remark it is a finite set up to logical equivalence.

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 on initialized transition systems as defined in Section 2.4. Let $\mathcal{P}$ be a finite set of atomic propositions. The set of *formulas* $\mathcal{F}_\mu = \mathcal{F}_\mu(\mathcal{P})$ over $\mathcal{P}$ of the modal μ-calculus is inductively defined as follows:

- $p, \neg p \in \mathcal{F}_\mu$ for all $p \in \mathcal{P}$
- $X \in \mathcal{F}_\mu$ for every unary fixpoint variable X
- if $\varphi, \psi \in \mathcal{F}_\mu$, then $\varphi \wedge \psi, \varphi \vee \psi \in \mathcal{F}_\mu$
- if $\varphi \in \mathcal{F}_\mu$, then $\Box\varphi, \Diamond\varphi \in \mathcal{F}_\mu$
- if X is a unary fixpoint variable and $\varphi \in \mathcal{F}_\mu$, then $\mu X.\varphi, \nu X.\varphi \in \mathcal{F}_\mu$.

We define the semantics of a formula $\varphi \in \mathcal{F}_\mu$ 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:

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

For an initialized transition system (T, q_{init}) over $\mathcal{P}$ with $T = (Q, R, \lambda)$ and a formula $\varphi \in \mathcal{F}_\mu$, we write $(T, q_{\mathrm{init}}) \models \varphi$ if $\mathcal{A}_T \models \|\varphi\|(q_{\mathrm{init}})$.

Example 2.2. *Let (T, q_{init}) be an initialized transition system. Then $(T, q_{\mathrm{init}}) \models \mu X.(\varphi \vee \Diamond X)$ if and only if there exists a state in which φ holds and which is reachable from q_{init}.*

		explicit [9, 14, 33, 34]	c**-bounded SLP**	**unrestricted SLP**
μ**-calc.**	**data**	P-complete	P-complete	PSPACE-complete
	combined	P-hard, in NP $\cap$ coNP	PSPACE-complete	PSPACE-complete
MLFP	**data**	P-complete	P-complete	EXPTIME-complete
	combined	PSPACE-complete	EXPTIME-complete	EXPTIME-complete
LFP	**data**	P-complete	EXPTIME-complete	EXPTIME-complete
	combined	EXPTIME-complete	EXPTIME-complete	EXPTIME-complete

Table 1: Data and combined complexity for fixpoint logics

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 $\mathcal{A} \models \varphi$. Following Vardi [34] we distinguish between the following three measures of complexity:

- *Data Complexity:* Input is the structure $\mathcal{A}$. The formula φ is fixed.
- *Expression Complexity:* The structure $\mathcal{A}$ is fixed and the input is the formula φ.
- *Combined Complexity:* Both the structure $\mathcal{A}$ and the formula φ are the input.

In this paper, we will only consider data and combined complexity for structures that are represented by SLPs. Considering expression complexity in this context does not lead to new insights: Having a fixed SLP is the same as having a fixed structure.

Table 1 collects the known results as well as our new results concerning the (data and combined) complexity of the model-checking problems for the logics LFP, MLFP, and the modal μ-calculus.

2.6 Parity games

In this section we introduce *parity games* and state the close relationship between parity games and the modal μ-calculus.

A parity game between two players, called Adam and Eve, is played on a particular kind of relational structure, called game graphs. Let $C = \{0, \dots, k\}$ ($k \in \mathbb{N}$) be a finite set of priorities. A *game graph* G over the set of priorities C is a tuple $G = (V, E, \rho)$ such that V is a finite set of nodes, $E \subseteq V \times C \times V$ is the set of labeled edges, and $\rho : V \to \{\text{Eve}, \text{Adam}\}$ assigns to every node v a player $\rho(v)$. The *size* of a game graph is defined by $|G| = |V| + |E|$. We define $\overline{\text{Eve}} = \text{Adam}$ and $\overline{\text{Adam}} = \text{Eve}$. Let $V_\sigma = \rho^{-1}(\sigma)$ denote the set of σ*-nodes* for a given player $\sigma \in \{\text{Eve}, \text{Adam}\}$. 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 [9, 12, 30] since priorities are assigned to edges

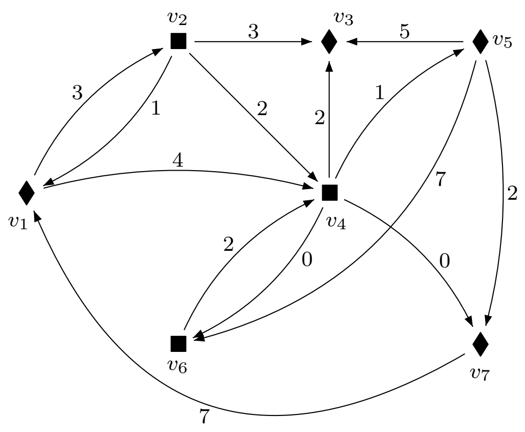


Figure 2: The game graph G from Example 2.3

instead to nodes. This is no restriction when considering parity games. We call a sequence $\pi = v_0, c_0, v_1, c_1, \ldots \in V(CV)^\omega$ an *infinite path* in G if for all $i \geq 0$ we have $(v_i, c_i, v_{i+1}) \in E$. A sequence $\pi = v_0, c_0, v_1, \ldots, c_{n-1}, v_n \in V(CV)^*$ is called a *finite path* in G if for all $0 \leq i \leq n-1$ 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 often in the path π. We call a path *maximal* if and only if it is infinite or it ends in a dead end, i.e., a node v with $vE = \emptyset$.

Example 2.3. *Figure 2 shows a game graph $G = (V, E, \rho)$ over the priorities $C = \{0, 1, \ldots, 7\}$. Here, ⧫ denotes an* Eve-*node and* ■ *denotes an* Adam-*node. An infinite path is for example $v_1, 3, v_2, 2, (v_4, 0, v_6, 2)^\omega \in V(CV)^\omega$. The finite path $v_7, 7, v_1, 3, v_2, 3, v_3 \in V(CV)^*$ ends in v_3 which is the only dead end of G.*

Clearly, the game graph $G = (V, E, \rho)$ can be identified with the relational structure $(V, (\{(u, v) \mid (u, c, v) \in E\})_{c \in C}, V_{\text{Eve}}, V_{\text{Adam}})$. 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 only defined (as an $(n-1)$-game graph) 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.

Example 2.4. *Figure 3 shows a* 3*-game graph G and the resulting* 2*-game-graph* $\mathrm{glue}(G)$*. Contact node $\tau(i)$ is labeled with i.*

In the following let $G = (V, E, \rho)$ be a game graph over the priorities $C = \{0, \ldots, k\}$ ($k \in \mathbb{N}$). A *play* is a maximal path in G. Let $\pi = v_0, c_0, v_1, \ldots$ be an infinite play in G and $\sigma \in \{\text{Eve}, \text{Adam}\}$ a player. We say that *player* Eve *(resp.* Adam*) wins the infinite play* π if and only if $\max(\mathrm{Inf}(\pi))$ is even (resp. odd). Let $\pi = v_0, c_0, v_1, \ldots, c_{n-1}, v_n$ be a finite

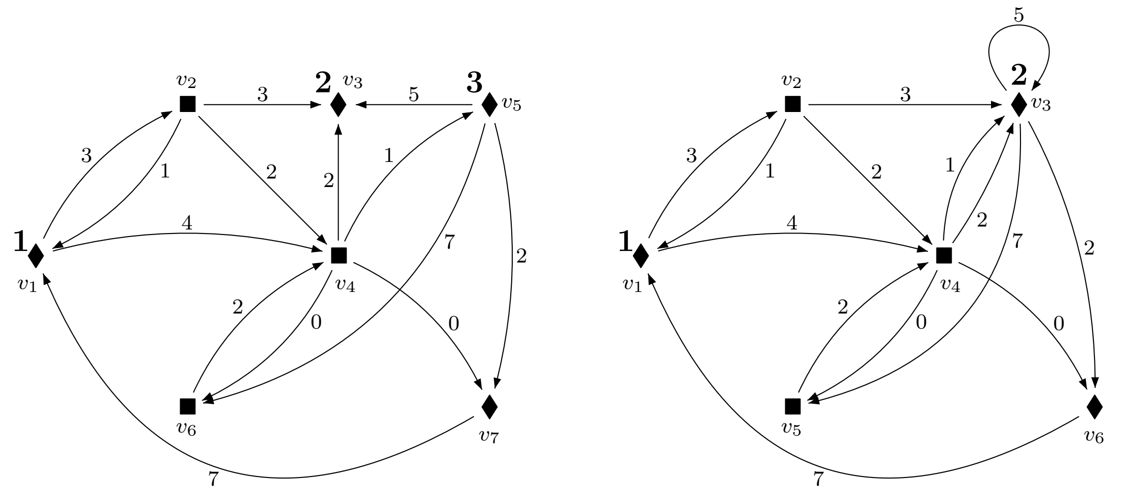


Figure 3: A 3-game graph G and the 2-game graph glue(G)

play. We say that player σ wins the finite play π if and only if $\rho(v_n) = \overline{\sigma}$, i.e., the play ends in a dead end that belongs to player $\overline{\sigma}$.

It is an important question whether a given player $\sigma \in \{\text{Eve}, \text{Adam}\}$ 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. Let $\sigma \in \{\text{Eve}, \text{Adam}\}$ be a player. Then a map $\mathscr{S}_\sigma : V_\sigma \setminus \{v \mid vE = \emptyset\} \to V$ such that $\mathscr{S}_\sigma(v) \in vE$ for all $v \in V_\sigma \setminus \{v \mid vE = \emptyset\}$ is called a *memoryless strategy* for player σ. We say that a finite play $\pi = v_0, c_0, v_1, \ldots c_{n-1}, v_n$ is *$\mathscr{S}_\sigma$-confirm* if and only if for all $0 \leq i \leq n-1$ we have $v_i \in V_\sigma \Rightarrow \mathscr{S}_\sigma(v_i) = v_{i+1}$. Similarly an infinite play $\pi = v_0, c_0, v_1, \ldots$ is called *$\mathscr{S}_\sigma$-confirm* if and only if for all $i \geq 0$ we have $v_i \in V_\sigma \Rightarrow \mathscr{S}_\sigma(v_i) = v_{i+1}$. For $v \in V$ we call the memoryless strategy $\mathscr{S}_\sigma$ a *memoryless winning strategy for player σ from the node v* if and only if player σ wins every $\mathscr{S}_\sigma$-confirm play which begins in v. Note that the question whether the memoryless strategy $\mathscr{S}_\sigma$ for player σ is a winning strategy can be answered in deterministic polynomial time by searching for a play which player $\overline{\sigma}$ wins in the subgraph of G which is restricted by $\mathscr{S}_\sigma$.

A triple (G, v, σ), where G is a game graph, v is a node of G, and $\sigma \in \{\text{Eve}, \text{Adam}\}$ is a player is called an *instance of the parity game problem.* We call an instance (G, v, σ) *positive* if there exists a memoryless winning strategy for player σ from v. The set of all positive instances of the parity game problem is denoted by PARITY. The determinacy theorem for parity games [9] states that $(G, v, \sigma) \in \text{PARITY}$ if and only if $(G, v, \overline{\sigma}) \notin \text{PARITY}$. It implies that PARITY belongs to $\mathsf{NP} \cap \mathsf{coNP}$.

Example 2.5. *Let the game graph $G = (V, E, \rho)$ over the priorities $\{0, 1, 2\}$ be given in Figure 4. Apparently, player* Eve *wins the node set $W_{\text{Eve}} = \{v_5, v_6, v_7\}$, whereas player* Adam *wins $W_{\text{Adam}} = \{v_1, v_2, v_3, v_4, v_8\}$. The fat drawn edges show a winning strategy for player* Eve *for the nodes in W_{Eve}. If player* Adam *always controls the game to node v_1,*

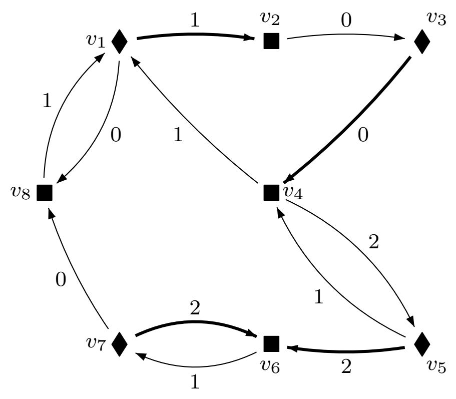


Figure 4: The game graph G from Example 2.5

player Eve *either has to go to* v_2 *or to* v_8. *The largest infinitely often occurring priority is* 1*, hence player* Adam *wins on the set* W_{Adam}. *Player* Eve *can force the game into the cycle* $(v_6, 1, v_7, 2)^\omega$ *for all nodes from* $\{v_5, v_6, v_7\}$. *Therefore she wins the set* W_{Eve}.

Theorem 2.6 ([8, 9])**.** *Let* $\mathcal{P}$ *be a set of atomic propositions,* (T, q_{init}) *an initialized transition system over* $\mathcal{P}$*, and* $\varphi \in \mathcal{F}_\mu(\mathcal{P})$. *Then there exists a game graph* $G_{T,\varphi}$ *and a node* v *of* $G_{T,\varphi}$ *s.t.* $(G_{T,\varphi}, v, \text{Eve}) \in \text{PARITY}$ *if and only if* $(T, q_{\text{init}}) \models \varphi$. *Furthermore, the reduction can be done in polynomial time.*

We will extend Theorem 2.6 in Section 4 to the case of hierarchically defined graphs.

3 Parity games on SLP-defined graphs

In this section we will prove a PSPACE upper bound for parity games on game graphs that are given via SLPs. Our construction is inspired by [30], where parity games and the modal μ-calculus on graphs of bounded tree width are examined. Thereby, first a strategy for player Eve is fixed. Then optimal reactions of player Adam are calculated efficiently on the tree decomposition in a bottom-up manner. For our PSPACE-algorithm we first have to introduce several concepts.

3.1 The strategy reduct of an n-game graph

Let $G = (H, \tau)$ be an n-game graph with $H = (V, E, \rho)$ and let $W \subseteq \rho^{-1}(\text{Eve}) \cap \text{ran}(\tau)$ be a set of contact nodes that belong to Eve. Then we call an n-game graph G' a *strategy reduct* of G w.r.t. W if and only 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 $w \in \rho^{-1}(\text{Eve}) \setminus (W \cup \{v \in V \mid vE = \emptyset\})$. Thus, a strategy reduct of G is the remainder of G by restricting G to a given strategy for Eve and making certain contact nodes that belong Eve to dead ends. Note that a strategy reduct is always defined w.r.t. a subset W of contact nodes that

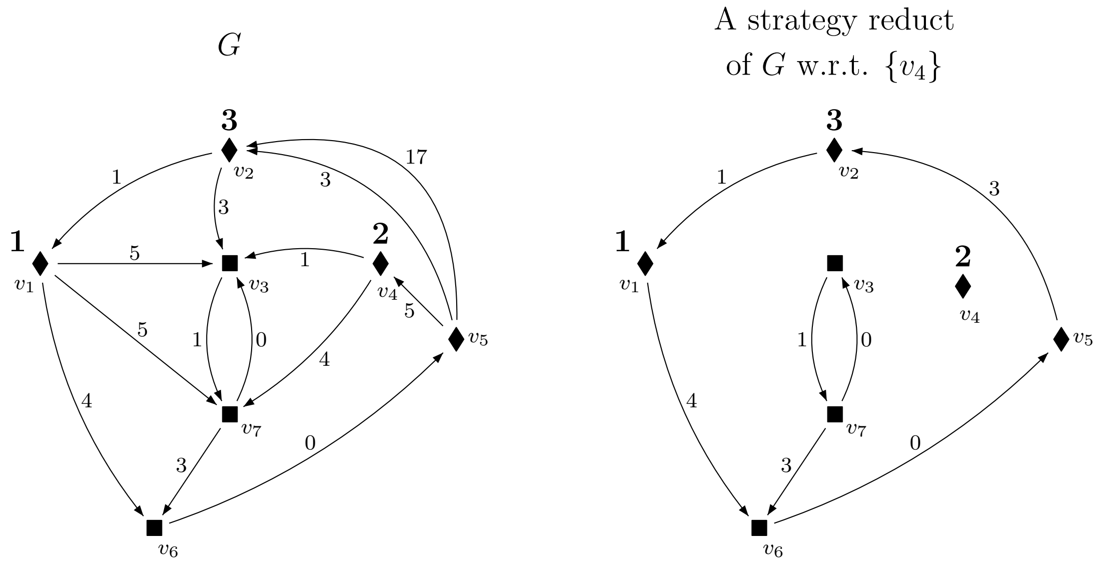


Figure 5: A strategy reduct of a 3-game graph G w.r.t. $\{v_4\}$

belong to Eve 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.

Example 3.1. *In Figure 5 a* 3*-game graph* G *together with a strategy reduct w.r.t.* $\{v_4\}$ *is shown.*

3.2 The evaluation function reward

For some guessed strategy reduct G' of a potentially exponentially large n-game graph $G = (H, \tau)$ we will only store a polynomial amount of relevant information in a so called n-interface. More precisely, for each pair of contact nodes $\tau(i)$ and $\tau(j)$ we will only store the maximal priority along an optimal path for player Adam from $\tau(i)$ to $\tau(j)$. In order to define this formally, we introduce the evaluation function reward, see also [30]. Let $C = \{0, \ldots, k\}$ ($k \in \mathbb{N}$) be a set of priorities. Then we define reward : $2^C \setminus \{\emptyset\} \to C$ as follows, where $B \subseteq C$, $B \neq \emptyset$:

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

Intuitively, reward(B) is the best priority in B for Adam: if there is an odd priority in B, then the largest odd priority is the best for Adam. But if there are only even priorities in B, then the smallest priority in B causes the smallest harm for Adam.

Let G be an (n-)game graph over the priorities $C = \{0, \ldots, k\}$ ($k \in \mathbb{N}$) and $\Pi \neq \emptyset$ a set of finite paths in G. Then we define

$$\text{reward}(\Pi) = \text{reward}(\{\ \max(\text{Occ}(\pi)) \mid \pi \in \Pi\}).$$

The intuition behind this definition is the following: If G' is a strategy reduct of an n-game graph G, then it is only player Adam who can freely choose the next outgoing edge in G'. Hence, if Π 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 $\pi \in \Pi$ with $\max(\text{Occ}(\pi)) = \text{reward}(\Pi)$ when going from $\tau(i)$ to $\tau(j)$. Note that $\max(\text{Occ}(\pi))$ is the relevant priority on the path π. We have to take the maximum of $\text{Occ}(\pi)$ since this priority is the relevant one to be considered. Hence, we can replace the set of paths Π by a single edge from $\tau(i)$ to $\tau(j)$ with priority $\text{reward}(\Pi)$. For technical reasons we will only put paths into Π that do not visit any contact nodes except its start and end node. We call such paths $\tau\tau$-internal paths and introduce them next.

3.3 $(\tau)\tau$-internal paths

Let $G = (H, \tau)$ be an n-game graph over the priorities $C = \{0, \ldots, k\}$ ($k \in \mathbb{N}$). For $v_0, v_n \in \text{ran}(\tau)$ 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 for all $1 \leq i \leq n-1$ we have $v_i \notin \text{ran}(\tau)$; note that $v_0 = v_n$ is allowed. We will be also interested in maximal paths that start in a contact node, but that never visit a contact node again. We call such paths τ-internal paths. More precisely, we call a finite *non-empty* maximal path $\pi = v_0, c_0, v_1 \ldots, c_{n-1}, v_n$ a *finite τ-internal path* from v_0 to v_n in G if $v_0 \in \text{ran}(\tau)$ and for all $1 \leq i \leq n$ we have $v_i \notin \text{ran}(\tau)$. Note that v_n must be a dead end, since π is assumed to be maximal. We call an infinite path $\pi = v_0, c_0, v_1, \ldots$ an *infinite τ-internal path* if $v_0 \in \text{ran}(\tau)$ and for all $i > 0$ we have $v_i \notin \text{ran}(\tau)$. Later, we will be only interested in τ-internal paths which can be won by player Adam.

Let $G = (H, \tau)$ be an n-game graph over the priorities $C = \{0, \ldots, k\}$ ($k \in \mathbb{N}$). Then $\Pi^{\tau}_{i,j}(G)$ denotes the set of all $\tau\tau$-internal paths from $\tau(i)$ to $\tau(j)$ for all $1 \leq i, j \leq n$ in G. 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 starts with a sequence of $\tau\tau$-internal paths possibly followed by a τ-internal path. Intuitively, this is the reason, why we do not lose any information by only considering $(\tau)\tau$-internal paths.

Example 3.2. *Figure 6 shows a fat drawn $\tau\tau$-internal path in a 3-game graph G from contact node $\tau(2)$ to contact node $\tau(3)$.*

3.4 The reduce operation

Assume that G' is a strategy reduct of an 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.

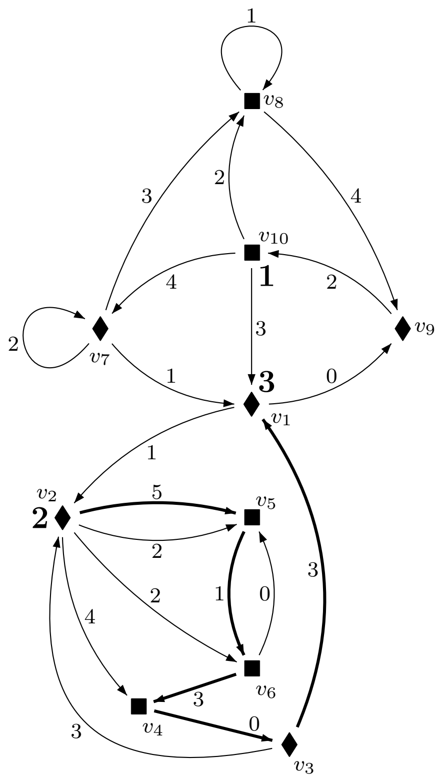


Figure 6: A $\tau\tau$-internal path in a 3-game graph G

Hence we can replace the set $\Pi^{\tau}_{i,j}(G)$ of all $\tau\tau$-internal paths from $\tau(i)$ to $\tau(j)$ by a single edge with priority reward$(\Pi^{\tau}_{i,j}(G))$. 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.

Let $G = (H, \tau)$ be an n-game graph over the priorities $C = \{0, \dots, k\}$ ($k \in \mathbb{N}$), where $H = (V, E, \rho)$. Then reduce(G) is the game graph $(\{1, \dots, n\}, F, \varrho)$, where $\varrho(i) = \rho(\tau(i))$ for all $1 \leq i \leq n$ and $(i, p, j) \in F$ if and only if $\Pi^{\tau}_{i,j}(G) \neq \emptyset$ and reward$(\Pi^{\tau}_{i,j}(G)) = p$. We identify reduce$(G)$ with the n-game graph $((\{1, \dots, n\}, F, \varrho), \mathrm{id}_{\{1,\dots,n\}})$. Note that if G is not a strategy reduct, then player Adam cannot, in general, force an optimal path with maximal priority reward$(\Pi^{\tau}_{i,j}(G))$ from $\tau(i)$ to $\tau(j)$. But if G is a strategy reduct, then he can do so, because Eve has no choice anymore.

Example 3.3. *In Figure 7 a* 3*-game graph G together with* reduce(G) *is shown.*

In Section 3.6 we will need the following two lemmas:

Lemma 3.4. *Let $n \in \mathbb{N}$ ($n \geq 1$),* op $\in$ {forget, glue}, *and $G = (H, \tau)$ an $(n+1)$-game graph s.t.* op(G) *exists (as an n-game graph). Then, we have* reduce(op(G)) = reduce(op(reduce(G))).

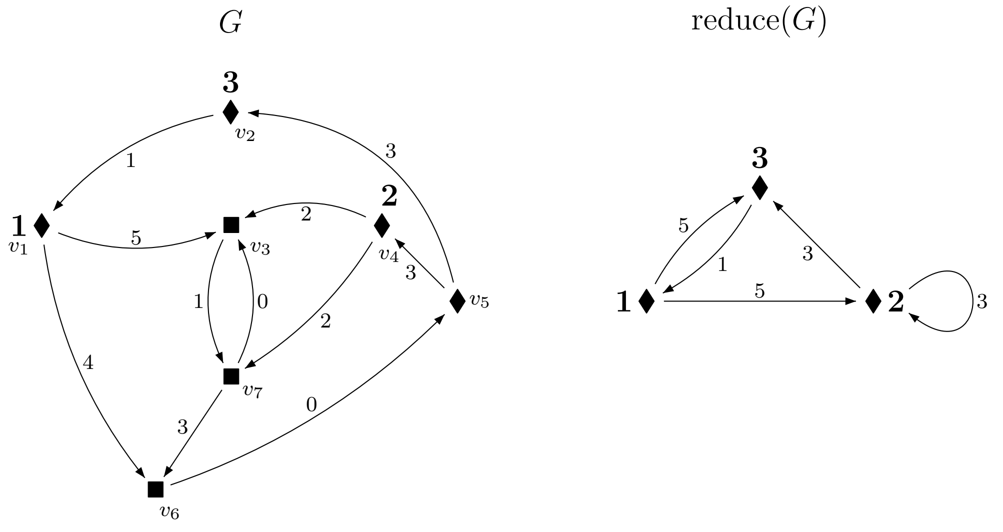


Figure 7: Comparison between G and reduce(G)

Proof. We have to show that

$$\text{reward}(\Pi^{\tau}_{i,j}(\text{op}(G))) = \text{reward}(\Pi^{\tau}_{i,j}(\text{op}(\text{reduce}(G)))) \tag{1}$$

for all $1 \leq i, j \leq n$. First note that $\Pi^{\tau}_{i,j}(\text{op}(G)) = \emptyset$ if and only if $\Pi^{\tau}_{i,j}(\text{op}(\text{reduce}(G))) = \emptyset$, in which case both sides of equation (1) are undefined. Moreover, for every path $\pi \in \Pi^{\tau}_{i,j}(\text{op}(\text{reduce}(G)))$ there exists a path $\pi' \in \Pi^{\tau}_{i,j}(\text{op}(G))$ such that $\max(\text{Occ}(\pi)) = \max(\text{Occ}(\pi'))$, because every edge (i', c, j') in reduce(G) corresponds to an optimal $\tau\tau$-internal path for player Adam from $\tau(i')$ to $\tau(j')$ in G with priority c. On the other hand, for every optimal path $\pi \in \Pi^{\tau}_{i,j}(\text{op}(G))$ for player Adam we find a path $\pi' \in \Pi^{\tau}_{i,j}(\text{op}(\text{reduce}(G)))$ with $\max(\text{Occ}(\pi)) = \max(\text{Occ}(\pi'))$. This implies (1). □

Lemma 3.5. *Let $G = (H, \tau)$ be an n-game graph over the priorities C. Then* reduce(G) *can be computed in deterministic polynomial time (w.r.t. $|G|$ and $|C|$).*

Proof. For a game graph G' and two nodes u and v of G' let $\Pi_{u,v}(G')$ denote the set of all paths in G' from u to v. Let $G_{i,j}$ be the game subgraph of G which is induced by $(V \setminus \text{ran}(\tau)) \cup \{\tau(i), \tau(j)\}$ for all $1 \leq i, j \leq n$ (where V is the node set of G). Then we have

$$\text{reward}(\Pi^{\tau}_{i,j}(G)) = \text{reward}(\Pi_{\tau(i),\tau(j)}(G_{i,j}))$$

for all $1 \leq i, j \leq n$. The algorithm in Table 2 computes $\text{reward}(\Pi_{\tau(i),\tau(j)}(G_{i,j}))$ by successively removing edges from $G_{i,j}$.

The first `if`-condition can be checked for instance by Dijkstra's algorithm deterministically in polynomial time. The number of loops is bounded by $|C|$. We execute the algorithm for all pairs $1 \leq i, j \leq n$ and get a polynomial time bound. □

```
procedure reward($G_{i,j}, \tau(i), \tau(j)$) return $p \in C$ is
   $p_{\min} := \max(C)$
   for $c = \max(C)$ downto 0 do
      if $\exists \pi \in \Pi_{\tau(i),\tau(j)}(G_{i,j}) : \max(\mathrm{Occ}(\pi)) = c$ then
         if $c$ is odd then return $c$
         else
            $p_{\min} := c$
            remove all edges $(u, c, v)$ from $G_{i,j}$
         endif
      endif
   endfor
   return $p_{\min}$
end reward
```

Table 2: Algorithm for computing reduce(G)

3.5 Interfaces and realizability

An n-interface stores all the relevant information for a given strategy reduct. For a given variable X_i of an SLP, the rank(X_i)-game graph eval(X_i) may have exponential size, and the same is true for some strategy reduct G' of eval(X_i). But any n-interface for G' can be stored in polynomial space, and this will be crucial in our overall PSPACE-algorithm. The notion of an interface is inspired by the notion of a border from [30].

An *n-interface* S ($n \in \mathbb{N}$) over the priorities $C = \{0, \dots, k\}$ ($k \in \mathbb{N}$) is a 5-tuple $S = (\{1, \dots, n\}, F, \varrho, I, U)$ s.t.

- $(\{1, \dots, n\}, F, \varrho)$ is a game graph over the priorities C, which we denote by graph(S),
- $I \subseteq \{1, \dots, n\}$ is a subset of the set of nodes $\{1, \dots, n\}$, and
- $U \subseteq \varrho^{-1}(\text{Eve})$ is a subset of the nodes which belong to player Eve.

We identify graph(S) with the n-game graph $((\{1, \dots, n\}, F, \varrho), \mathrm{id}_{\{1,\dots,n\}})$, which formally also contains the identity over $\{1, \dots, n\}$ as a component.

Formally an n-interface is nothing more than a game graph with node set $\{1, \dots, n\}$ and two subsets of $\{1, \dots, n\}$. We now define what it means that an n-interface is realized by an n-game graph.

Definition 3.6. *We say that an n-interface $S = (\{1, \dots, n\}, F, \varrho, I, U)$ is* realized *by an n-game graph $G = (H, \tau)$ if there exists a strategy reduct $G' = (H', \tau)$ of G w.r.t. $\tau(U)$ s.t.*

(1) graph(S) = reduce(G'), *and*

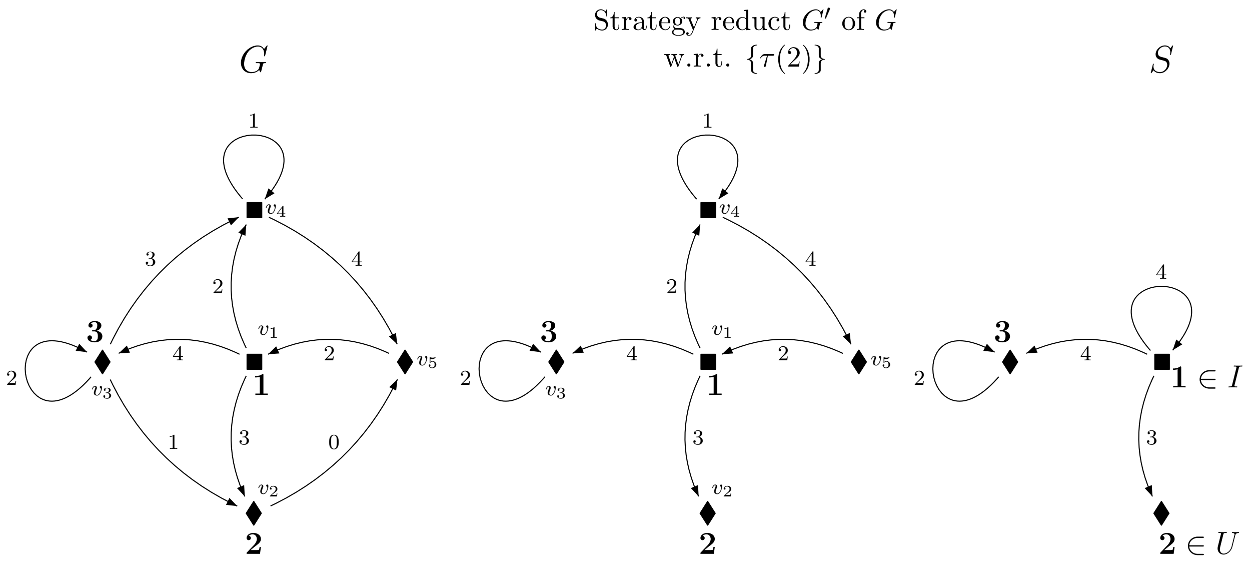


Figure 8: The 3-game graph G realizes the 3-interface S

(2) $i \in I$ if and only if there exists a τ-internal path π in G' which begins in $\tau(i)$ and which player Adam *wins (recall that π is necessarily non-empty by the definition of τ-internal paths).*

We also say that G' is a witness *that S is realized by G.*

So the notion of realization intuitively expresses the fact that an n-interface correctly summarizes reactions of player Adam in a remainder on an n-game graph w.r.t to a restricted strategy for Eve.

Remark 3.7. *Note that Condition (2) in Definition 3.6 can be checked in polynomial time for a given strategy reduct G' and $1 \leq i \leq n$.*

Example 3.8. *In Figure 8 a 3-game graph G together with a strategy reduct G' w.r.t. $\{\tau(2)\}$ shown. The interface $S = (\{1, 2, 3\}, F, \rho, I, U)$ with $I = \{1\}$ and $U = \{2\}$ on the right is realized by G, and G' is a witness for this. We have $1 \in I$, because the infinite τ-internal path $v_1, 2, (v_4, 1)^\omega$ starts at node $v_1 = \tau(1)$ 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 $v_1, 2, v_4, 4, v_5, 2, v_1$ in G'.*

Lemma 3.9. *Let $S = (\{1, \dots, n\}, E, \rho, I, U)$ be an n-interface, let G be an n-game graph, and let G' be a strategy reduct of G w.r.t. $\tau(U)$. Then it can be decided in polynomial time, whether G' is a witness that S is realized by G.*

Proof. We compute reduce(G') deterministically in polynomial time (Lemma 3.5) and check the two conditions of Definition 3.6 in polynomial time, see Remark 3.7. □

Lemma 3.10. *Let $S = (\{1, \ldots, n\}, F, \rho, I, U)$ be an n-interface and let G be an n-game graph. Then the question whether S is realized by G is in* NP.

Proof. We guess a strategy reduct G' of G w.r.t. $\tau(U)$ and apply Lemma 3.9. □

3.6 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 expressed by Definition 3.11, which is formulated 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 $\tau(i)$. The owner of node $\tau(i)$ will not choose this edge, because she/he will immediately loose at node v. Hence the new edge has no influence on the winner of a parity game.

Definition 3.11. *Let* op *be a partial operation, mapping a k-tuple $(G_1, \ldots, G_k)$, where G_i is an n_i-game graph, to an n-game graph* $\mathrm{op}(G_1, \ldots, G_k)$. *We say that* op *has a* faithful polynomial implementation (briefly FPI) *on interfaces, if there exists a partial operation* op^s, *mapping a a k-tuple $(S_1, \ldots, S_k)$, where S_i is an n_i-interface, to an n-interface* $\mathrm{op}(S_1, \ldots, S_k)$ *s.t. the following holds:*

- op^s *is computable in polynomial time.*
- *Assume that $G = \mathrm{op}(G_1, \ldots, G_k)$, where G_i is an n_i-game graph and G is an n-game graph, and let S be 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. $S = \mathrm{op}(S_1, \ldots, S_k)$ and G_i realizes S_i.*

Lemma 3.12. *The operations* $\oplus$, rename_f, forget, *and* glue *have FPIs on interfaces.*

Proof. Let $S_i = (\mathrm{graph}(S_i), I_i, U_i)$ $(i \in \{1, 2\})$ be an n_i-interface. We set $S_1 \oplus^s S_2 = (\mathrm{graph}(S_1) \oplus \mathrm{graph}(S_2), I_1 \cup (n_1 + I_2), U_1 \cup (n_1 + U_2))$, where $n_1 + I_2 = \{n_1 + i \mid i \in I_2\}$ and similarly for $n_1 + U_2$.

For an n-interface $S = (\mathrm{graph}(S), I, U)$ and a permutation $f : \{1, \ldots, n\} \to \{1, \ldots, n\}$ let $\mathrm{rename}^s_f(S) = (\mathrm{rename}_f(\mathrm{graph}(S)), f(I), f(U))$.

For an $(n + 1)$-interface $S = (\{1, \ldots, n + 1\}, E, \rho, I, U)$ we define $\mathrm{forget}^s(S)$ only if $n + 1 \notin U$. Then $\mathrm{forget}^s(S) = (\{1, \ldots, n\}, E', \rho', I', U')$, where:

(a) $\mathrm{graph}(\mathrm{forget}^s(S)) = \mathrm{reduce}(\mathrm{forget}(\mathrm{graph}(S)))$

(b) $I' = \begin{cases} I \setminus \{n+1\} \cup \{i \mid 1 \leq i \leq n \wedge n+1 \in iE\} & \text{if } n+1 \in I \\ I & \text{else} \end{cases}$

(c) $U' = U$

The intuition behind this definition is the following. Assume that the $(n+1)$-interface $S = (\{1,\dots,n+1\}, E, \rho, I, U)$ is realized by an $(n+1)$-game graph $G = (H,\tau)$ and let G' be a witness for this. We want to define $\text{forget}^s(S) = (\{1,\dots,n\}, E', \rho', I', U')$ in such a way that $\text{forget}^s(S)$ is realized by $\text{forget}(G)$ and moreover $\text{forget}(G')$ is a witness for this. Since $n+1$ is no longer a contact node in $\text{forget}(G)$, there may be more $\tau\tau$-internal paths in $\text{forget}(G')$ between two contact nodes $\tau(i)$ and $\tau(j)$. In order to determine the maximal priority of an optimal path (for player Adam) from $\tau(i)$ to $\tau(j)$ in $\text{forget}(G')$, it suffices to look at the n-game graph $\text{forget}(\text{graph}(S))$, i.e., to calculate $\text{reduce}(\text{forget}(\text{graph}(S)))$. This graph will be therefore $\text{graph}(\text{forget}^s(S))$. Second, if in the strategy reduct G' there exists a $\tau\tau$-internal path from the contact node i to the contact node $n+1$ (i.e., in the interface S there is an edge from i to $n+1$) and $n+1 \in I$ (i.e., there exists a τ-internal path starting from $\tau(n+1)$ in G' and which player Adam wins), then there exists a τ-internal path starting from $\tau(i)$ in $\text{forget}(G')$ and which player Adam wins. Therefore we put i into I'. Finally, we require $n+1 \notin U$, because after applying the forget-operation, the former contact node $\tau(n+1)$ is no longer accessible, in particular it cannot be glued with another node and will not get any further outgoing edges. But if $\tau(n+1)$ belongs to Eve, for a strategy of Eve we have to guess precisely one outgoing edge for $\tau(n+1)$; recall that we assume that every contact node, and hence also $\tau(n+1)$, has at least one outgoing edge in G. If we would have $n+1 \in U$, then we would remove all outgoing edges for $\tau(n+1)$, and this would not change anymore, since $\tau(n+1)$ remains inaccessible after the forget-operation.

Finally, for an $(n+1)$-interface ($n \geq 1$) $S = (\{1,\dots,n+1\}, E, \rho, I, U)$ we define $\text{glue}^s(S)$ only if

(1) $\rho(n+1) = \rho(n)$ (thus, node n and $n+1$ belong to the same player and can actually be glued) and

(2) if $\rho(n+1) = \rho(n) = \text{Eve}$ then $n \in U$ or $n+1 \in U$.

Then we define the n-interface $\text{glue}^s(S) = (\{1,\dots,n\}, E', \rho', I', U')$ as follows:

(a) $\text{graph}(\text{glue}^s(S)) = \text{reduce}(\text{glue}(\text{graph}(S)))$.

(b) $I' = \begin{cases} I \setminus \{n+1\} \cup \{n\} & \text{if } I \cap \{n, n+1\} \neq \emptyset \\ I & \text{else} \end{cases}$

(c) $U' = \begin{cases} U \setminus \{n+1\} & \text{if } n, n+1 \in U \\ U \setminus \{n, n+1\} & \text{else} \end{cases}$

The intuition behind this definition is the following. Assume that the $(n+1)$-interface $S = (\{1,\dots,n+1\}, E, \rho, I, U)$ is realized by an $(n+1)$-game graph $G = (H,\tau)$ and let G' be a witness for this. We want to define $\text{glue}^s(S) = (\{1,\dots,n\}, E', \rho', I', U')$ in such a way that $\text{glue}^s(S)$ is realized by $\text{glue}(G)$ and moreover $\text{glue}(G')$ is a witness for this. Note that by (1), (2), and (c), $\text{glue}(G')$ is in fact a strategy reduct of $\text{glue}(G)$ w.r.t. U'. In particular,

$$\begin{tikzcd}
G_2 \arrow[r, "\text{strategy reduct w.r.t. } U_2"] \arrow[dd, "\text{glue}"] & G_2' \arrow[r, "\text{reduce}"] \arrow[dd, "\text{glue}"] & \text{graph}(S_2) \arrow[d, "\text{glue}"] \\
 & & {} \arrow[d, "\text{reduce}"] \\
G_1 \arrow[r, "\text{strategy reduct w.r.t. } U_1"] & G_1' \arrow[r, "\text{reduce}"] & \text{graph}(S_1)
\end{tikzcd}$$

Figure 9: The situation in the proof of Lemma 3.12

(2) is necessary for this, since by our assumption both $\tau(n)$ and $\tau(n+1)$ have at least one outgoing edge in G and hence would have both precisely one outgoing edge in G' if we would have $n \notin U$ and $n+1 \notin U$. Thus, the n-th contact node of glue(G') would have two outgoing edges. The assignment $\text{graph}(\text{glue}^s(S)) = \text{reduce}(\text{glue}(\text{graph}(S)))$ in (a) can be explained as for the forget-operation. Note that in glue(graph(S)), there may be more than one edge between two contact nodes. By applying reduce to glue(graph(S)) we select the optimal edge for player Adam between two contact nodes. Finally, if $n \in I$ or $n+1 \in I$, i.e., there exists a τ-internal path in G' that starts in $\tau(n)$ or in $\tau(n+1)$ and which player Adam wins, then we can be sure that there exists a τ-internal path in glue(G') that starts in $\tau(n)$ and which player Adam wins. Here it is important that τ-internal paths are always non-empty. Hence, we put n into the set I'.

This concludes the definition of the operations on interfaces. Each of these operations can be computed in polynomial time; for forget^s and glue^s we need Lemma 3.4 in order to compute reduce(op(graph(S))) (op $\in$ {forget, glue}). We present the proof for the second condition of Definition 3.11 only for the glue-operation.

Let $G_1 = (H_1, \tau_1)$ be an n-game graph and $G_2 = (H_2, \tau_2)$ an $(n+1)$-game graph with glue$(G_2) = G_1$. Let S_1 be an n-interface. We have to show that the following two properties are equivalent:

(1) S_1 is realized by G_1.

(2) There exists an $(n+1)$-interface S_2, which is realized by G_2 and such that glue$^s(S_2) = S_1$.

Figure 9 makes the situation clearer.

(1) $\Rightarrow$ (2): Assume that G_1 realizes $S_1 = (\{1, \dots, n\}, E_1, \rho_1, I_1, U_1)$ and let G_1' be a witness for this. Since we have $G_1 = \text{glue}(G_2)$, there exists a strategy reduct G_2' of G_2 w.r.t. a set U_2 satisfying

$$U_1 = \begin{cases} U_2 \setminus \{n+1\} & \text{if } n, n+1 \in U_2 \\ U_2 \setminus \{n, n+1\} & \text{else.} \end{cases} \tag{2}$$

Furthermore, we have $\mathrm{glue}(G'_2) = G'_1$. We put i into I_2 if and only if there exists a τ-internal path starting from $\tau_2(i)$ in G'_2 which player Adam wins. We set $S_2 = (\mathrm{reduce}(G'_2), I_2, U_2)$. Then, G'_2 is a witness that G_2 realizes S_2. In order to prove $\mathrm{glue}^s(S_2) = S_1$, we show the three conditions (a),(b), and (c) from the definition of the glue^s-operation. Condition (a) follows from:

$$\begin{aligned}
\mathrm{reduce}(\mathrm{glue}(\mathrm{graph}(S_2))) &= \mathrm{reduce}(\mathrm{glue}(\mathrm{reduce}(G'_2))) \\
&\overset{\text{Lemma 3.4}}{=} \mathrm{reduce}(\mathrm{glue}(G'_2)) \\
&= \mathrm{reduce}(G'_1) \\
&= \mathrm{graph}(S_1)
\end{aligned}$$

In order to show condition (b), we distinguish the following two cases (note that $G'_1 = \mathrm{glue}(G'_2)$ and that G'_i is a witness that G_i realizes S_i):

- $I_2 \cap \{n, n+1\} = \emptyset$: Then we have $I_1 = I_2$ due to Definition 3.6. Hence, condition (b) is satisfied.

- $I_2 \cap \{n, n+1\} \neq \emptyset$: Then we have $n \in I_1$. Thus, $i \in I_1$ if and only if $i \in I_2 \setminus \{n + 1\} \cup \{n\}$. Hence, condition (b) is satisfied.

Condition (c) is satisfied by equation (2) above.

(2) $\Rightarrow$ (1): Assume that $S_2 = (\{1, \dots, n+1\}, E_2, \rho_2, I_2, U_2)$ is an $(n+1)$-interface, which is realized by G_2 and such that $\mathrm{glue}^s(S_2) = S_1$. Let G'_2 be a witness for this. We set $G'_1 = \mathrm{glue}(G'_2)$. We have to verify condition (1) and (2) of Definition 3.6 for S_1, G'_1 and G_1. Condition (1), i.e. $\mathrm{graph}(S_1) = \mathrm{reduce}(G'_1)$, follows from Lemma 3.4 analogously to the first part of the proof. For condition (2) of Definition 3.6, we again distinguish between the following two cases:

- $I_2 \cap \{n, n+1\} = \emptyset$: Then we have $I_1 = I_2$ according to the definition of the glue^s-operation. Moreover, there does not exist a τ-internal path in G'_2 starting in $\tau_2(n)$ or $\tau_2(n+1)$ and which Adam wins. Thus, for all $i \in \{1, \dots, n\}$ we have:

$$\begin{aligned}
i \in I_1 &\iff i \in I_2 \\
&\iff \exists\ \tau\text{-internal path in } G'_2 \text{ starting in } \tau_2(i) \text{ and which Adam wins} \\
&\iff \exists\ \tau\text{-internal path in } G'_1 = \mathrm{glue}(G'_2) \text{ starting in } \tau_1(i) \text{ and} \\
&\qquad \text{which Adam wins}
\end{aligned}$$

- $I_2 \cap \{n, n+1\} \neq \emptyset$: Then we have $I_1 = I_2 \setminus \{n+1\} \cup \{n\}$ according to the definition of the glue^s-operation. Moreover, there exists a τ-internal path in G'_2 starting in $\tau_2(n)$ or $\tau_2(n+1)$ and which Adam wins. Thus, for all $i \in \{1, \dots, n\}$ we have:

$$\begin{aligned}
i \in I_1 &\iff i \in I_2 \setminus \{n+1\} \cup \{n\} \\
&\iff i = n \text{ or} \\
&\qquad \exists\ \tau\text{-internal path in } G'_2 \text{ starting in } \tau_2(i) \text{ and which Adam wins} \\
&\iff \exists\ \tau\text{-internal path in } G'_1 = \mathrm{glue}(G'_2) \text{ starting in } \tau_1(i) \text{ and} \\
&\qquad \text{which Adam wins}
\end{aligned}$$

This concludes the proof of Lemma 3.12. □

3.7 Upper bounds for parity games on SLP-defined graphs

We are now ready to prove an upper bound of PSPACE for the parity game problem on general SLPs. For c-bounded SLPs we will obtain the better upper bound of $\mathsf{NP} \cap \mathsf{coNP}$. W.l.o.g. we will restrict to SLPs such that for every right hand side, which is an n-game graph G, every contact node of G has at least one outgoing edge, see the remark at the beginning of Section 3.6. Note that this property transfers to every game graph $\mathrm{eval}(X)$ for a variable X of the underlying SLP.

Theorem 3.13. *The following problem is in* PSPACE*:*
INPUT: An SLP $\mathcal{S} = (X_i := t_i)_{1 \le i \le l}$ *generating a* 1*-game graph* $\mathrm{eval}(\mathcal{S}) = (G, \tau)$.
QUESTION: $(G, \tau(1), \mathrm{Eve}) \in \mathrm{PARITY}$*?*

Proof. Without loss of generality we can assume that node $\tau(1)$ belongs to Eve and that $\tau(1)$ has no incoming edges. Otherwise we construct an SLP that generates G' by adding a new node v to G whose only edge is an outgoing one leading to $\tau(1)$ and give v to Eve. Then we have $(G', v, \mathrm{Eve}) \in \mathrm{PARITY}$ if and only if $(G, \tau(1), \mathrm{Eve}) \in \mathrm{PARITY}$. Due to this convention, the following holds: $(G, \tau(1), \mathrm{Eve}) \in \mathrm{PARITY}$ if and only if $\mathrm{eval}(G)$ realizes the interface $S_l = (\{1\}, \emptyset, [1 \mapsto \mathrm{Eve}], \emptyset, \emptyset)$.[1] 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 $\mathrm{rank}(X_i)$-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 $\mathrm{rank}(X_i)$-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 $1 \le j \le k$ guess $\mathrm{rank}(X_{i_j})$-interfaces $S_{i_j}$ s.t. $S_i = \mathrm{op}^s(S_{i_1},\dots,S_{i_k})$   $(**)$
    $(Q_\forall)$: return $\bigwedge_{1 \le j \le k} \mathcal{P}(i_j, S_{i_j})$
  endif
```

The correctness of the algorithm follows easily by induction on the index $i \in \{1,\dots,l\}$ using Definition 3.11. For the alternating polynomial time bound note that: (i) the test in line $(*)$ is in NP by Lemma 3.10, (ii) an interface can be stored in polynomial space, i.e., polynomial time suffices for guessing an interface in line $(**)$, and (iii) each of the operations op^s in line $(**)$ is computable in polynomial time by the definition of an FPI. □

One might present the above algorithm also in terms of top-down tree automata. The state set of the tree automaton is the set of all n-interfaces, where n is the maximal rank

[1]Since $\tau(1)$ has no incoming edges, we can assume that the interface S_l has no edges, i.e., consists of an isolated point. Since the I-component of S_l is empty, we assert that Adam cannot win from node $\tau(1)$, i.e., Eve wins.

of a variable X_i. The tree on which the automaton runs is the unfolding of the SLP viewed as a dag (directed acyclic graph), where the variables are the nodes.

By the following theorem, we can improve the PSPACE upper bound from Theorem 3.13 to $\mathsf{NP} \cap \mathsf{coNP}$, when we restrict to c-bounded SLPs for some fixed constant c.

Theorem 3.14. *Let $c \in \mathbb{N}$ be a fixed constant. Then the following problem is in $\mathsf{NP}\cap\mathsf{coNP}$:*
INPUT: A c-bounded SLP $\mathcal{S} = (X_i := t_i)_{1 \leq i \leq l}$ such that $\mathrm{eval}(\mathcal{S})$ is a 1-game graph (G, τ).
QUESTION: $(G, \tau(1), \mathrm{Eve}) \in \mathrm{PARITY}$?

Proof. In analogy to the proof of Theorem 3.13 we may assume that node $\tau(1)$ belongs to Eve and that $\tau(1)$ has no incoming edges. Now, we guess for all $1 \leq i \leq l$ a set of 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 $\mathcal{S}$. Moreover, for every $1 \leq i \leq l$ there maximally exist $|C|^{c^2} 2^{2c}$ possible interfaces. Hence, since c is a constant, polynomial space suffices in order to store all interfaces in $\bigcup_{1 \leq i \leq l} M_i$. Next, we check in polynomial time whether for all $1 \leq i \leq l$ the set M_i is a subset of the set of interfaces which are realized by $\mathrm{eval}(X_i)$. If the interface $S_l = (\{1\}, \emptyset, [1 \mapsto \mathrm{Eve}], \emptyset, \emptyset)$ additionally belongs to M_l, then we know that $(G, \tau(1), \mathrm{Eve}) \in \mathrm{PARITY}$. In Table 3 the algorithm is shown. For the correctness of the algorithm we prove the following two points:

(1) If $(G, \tau(1), \mathrm{Eve}) \in \mathrm{PARITY}$, then there exists a run in our non-deterministic algorithm of Table 3, where true is returned.

(2) If the algorithm of Table 3 returns true, then $(G, \tau(1), \mathrm{Eve}) \in \mathrm{PARITY}$.

To show (1), we simply guess in line $(*)$ for all $1 \leq i \leq l$ exactly the set of interfaces that are realized by $\mathrm{eval}(X_i)$. Moreover, in line $(**)$ we guess for every $S \in M_i$ such that $t_i = G$ is an n-game graph a witness $G(i, S)$ that G realizes S. Then the algorithm will return true. For (2) let M_i be the set of interfaces for $\mathrm{eval}(X_i)$ $(1 \leq i \leq l)$ that are guessed in a successful run of the algorithm. By induction over i we easily obtain that every interface in M_i is realized by $\mathrm{eval}(X_i)$. Hence, $(\{1\}, \emptyset, [1 \mapsto \mathrm{Eve}], \emptyset, \emptyset)$ is realized by $\mathrm{eval}(\mathcal{S}) = \mathrm{eval}(X_l)$, i.e., $(G, \tau(1), \mathrm{Eve}) \in \mathrm{PARITY}$.

By Lemma 3.9 the test in line $(***)$ can be done in polynomial time. The tests in the other cases can be also done in polynomial time, which implies the upper bound of NP. Due to the determinacy theorem for parity games [9], the problem is also in coNP. $\square$

4 The modal μ-calculus on SLP-defined graphs

In this section, we show that both the data and combined complexity of the modal μ-calculus on transition systems that are represented by SLPs is PSPACE-complete. The PSPACE upper bound generalizes a corresponding result for CTL from [1], and will be shown by a reduction to parity games, which is analogous to the corresponding reduction for explicitly given input graphs. For this, we need a few notions concerning the modal μ-calculus.

```
procedure $\mathcal{P}(\mathcal{S})$ return boolean is
  for $i = 1$ to $l$ do
    guess a set $M_i$ of rank($X_i$)-interfaces.                                  $(*)$
    if $t_i = G$ for a rank($X_i$)-game graph $G$ then
      guess a strategy reduct $G(i, S)$ of $G$ for every $S \in M_i$              $(**)$
   endif
  endfor
  for $i = 1$ to $l$ do
    if $t_i = G$ for a rank($X_i$)-game graph $G$ then
      for $S \in M_i$ do
        if $G(i, S)$ is not a witness that $G$ realizes $S$ then                 $(***)$
          return false
        endif
      endfor
    elseif $t_i = \mathrm{op}(X_{i_1}, \ldots, X_{i_k})$ then
      if $\exists S_i \in M_i \, \forall (S_{i_1}, \ldots, S_{i_k}) \in \prod_{j=1}^{k} M_{i_j} : S_i \neq \mathrm{op}^s(S_{i_1}, \ldots, S_{i_k})$ then
        return false
      endif
    endif
  endfor
  return $(\{1\}, \emptyset, [1 \to \mathrm{Eve}], \emptyset, \emptyset) \in M_l$
end $\mathcal{P}$
```

Table 3: NP-algorithm for the c-bounded case

Let $\mathcal{P}$ be a set of atomic propositions. If φ is a subformula of ψ we also write $\varphi \preceq \psi$. In the following we assume w.l.o.g. that all sentences $\varphi \in \mathcal{F}_\mu(\mathcal{P})$ have the property that for every fixpoint variable X that occurs in φ there is a unique subformula $\sigma X.\psi \preceq \varphi$ with $\sigma \in \{\mu, \nu\}$ and such that all occurrences of X in φ are inside of $\sigma X.\psi$.

Theorem 4.1. *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 node q_{init} of* $\mathrm{eval}(\mathcal{S}_t)$*, and a sentence φ of the modal μ-calculus s.t. φ has precisely 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* $\mathrm{eval}(\mathcal{S}_g)$ *such that* $(\mathrm{eval}(\mathcal{S}_t), q_{\mathrm{init}}) \models \varphi$ *if and only if* $(\mathrm{eval}(\mathcal{S}_g), v, \mathrm{Eve}) \in \mathrm{PARITY}$.

Proof. Let us first repeat the construction for explicitly given input graphs [8, 9, 12]. Thus, let $T = (Q, R, \lambda)$ be a transition system, let $\Theta := \{\psi \mid \psi \preceq \varphi\}$ denote the set of all subformulas of the formula φ, and let $\{\psi_1, \psi_2, \ldots, \psi_k\}$ be an enumeration of these subformulas (i.e. $|\Theta| = k$). The alternation depth $\alpha(\psi)$ of a formula can be defined as in

[12, p 176]; the concrete definition is not important for the further construction. We define the map $\chi : \Theta \to \{0, \dots, \alpha(\varphi)\}$ for all $\psi \in \Theta$ as follows:

$$\chi(\psi) = \begin{cases} \text{smallest even number greater or equal to } \alpha(\psi) - 1 \text{ if } \psi = \nu X.\psi' \\ \text{smallest odd number greater or equal to } \alpha(\psi) - 1 \text{ if } \psi = \mu X.\psi' \\ 0 \text{ else} \end{cases}$$

Let $G_{\varphi,T} = (V, E, \rho)$ be the game graph that is defined as follows: The set of nodes is $V = (Q \times \Theta) \cup \{\perp, \top\}$. We set

$$\rho(v, \psi) = \begin{cases} \text{Adam} & \text{if } \psi \text{ of the form } \neg p \ (p \in \mathcal{P}) \text{ or } \psi_1 \wedge \psi_2 \text{ or } \Box\psi' \\ \text{Eve} & \text{else} \end{cases}$$

and $\rho(\perp) = \rho(\top) = \text{Eve}$. Finally the set E contains precisely the following edges:

$$\begin{array}{rcll}
\top & \xrightarrow{0} & \top & \\
\perp & \xrightarrow{1} & \perp & \\
(q, p) & \xrightarrow{0} & \begin{cases} \top & \text{if } p \in \mathcal{P},\ p \in \lambda(q) \\ \perp & \text{if } p \in \mathcal{P},\ p \notin \lambda(q) \end{cases} & \\
(q, \neg p) & \xrightarrow{0} & \begin{cases} \top & \text{if } p \in \mathcal{P},\ p \notin \lambda(q) \\ \perp & \text{if } p \in \mathcal{P},\ p \in \lambda(q) \end{cases} & \\
(q, \sigma X.\psi) & \xrightarrow{\chi(\psi)} & (q, \psi) & \text{for } \sigma \in \{\mu, \nu\} \\
(q, X) & \xrightarrow{\chi(\sigma X.\psi)} & (q, \sigma X.\psi) & \text{if } \sigma X.\psi \text{ is the unique subformula of } \varphi \text{ binding } X \\
(q, \psi_1 \text{ op } \psi_2) & \xrightarrow{\chi(\psi_i)} & (q, \psi_i) & \text{for } i \in \{1, 2\} \text{ and op} \in \{\wedge, \vee\} \\
(q, \sigma\psi) & \xrightarrow{\chi(\psi)} & (q', \psi) & \text{if } (q, q') \in R \text{ and } \sigma \in \{\Box, \Diamond\}
\end{array}$$

Then for every $q \in Q$ we have $(T, q) \models \varphi$ if and only if $(G_{\varphi,T}, (q, \varphi), \text{Eve}) \in \text{PARITY}$, see [8, 9, 12].

Hence, for a given SLP $\mathcal{S}_t$ defining a transition system $\text{eval}(\mathcal{S}_t)$, we have to construct an SLP defining the game graph $G_{\varphi,\text{eval}(\mathcal{S}_t)}$. In fact, we will construct an SLP $\mathcal{S}_g$ for a slight variant of $G_{\varphi,\text{eval}(\mathcal{S}_t)}$. Let $\mathcal{S}_t = (X_i := t_i)_{1 \le i \le l}$. In the SLP $\mathcal{S}_g$ we will use generalized versions of the operations glue and forget. First of all, if $G = (H, \tau)$ is an n-game graph, then for every $m \le n$ we define the $(n-m)$-game graph $\text{forget}_m(G) = (H, \tau \restriction \{1, \dots, n-m\})$, i.e., we forget the last m contact nodes. Moreover, for every $m \le n$ with $2m \le n$ we define the $(n-m)$-game graph $\text{glue}_m(G) = (H/_\equiv, \tau')$, where $\equiv$ is the smallest equivalence relation on $\{1, \dots, n\}$ that contains every pair $(n-i, n-m-i)$ for $0 \le i \le m-1$ and $\tau' = (\pi_\equiv \circ \tau)\restriction\{1, \dots, n-m\}$.

Now, we define the SLP $\mathcal{S}_g = (Y_i := u_i)_{1 \le i \le l}$ as follows. First of all, the rank of Y_i will be $\text{rank}(X_i) \cdot k$; recall that k is the number of subformulas of φ. If t_i is an n-transition system

(T, τ), then u_i is the $(n \cdot k)$-game graph $(G_{\varphi,T}, \tau')$, where $\tau'(r + j \cdot k) = (\tau(j+1), \psi_r)$ for $1 \leq r \leq k$ and $0 \leq j \leq n-1$. Next, if $t_i = X_j \oplus X_k$, then $u_i = Y_j \oplus Y_k$. If $t_i = \mathrm{op}(X_j)$ for $\mathrm{op} \in \{\mathrm{forget}, \mathrm{glue}\}$, then $u_i = \mathrm{op}_k(Y_j)$. Finally, if $t_i = \mathrm{rename}_f(X_j)$, then $u_i = \mathrm{rename}_{f'}(Y_j)$, where $f'(r + s \cdot k) = r + (f(s+1) - 1) \cdot k$ for $1 \leq r \leq k$ and $0 \leq s \leq \mathrm{rank}(X_i) - 1$. The only difference between $\mathrm{eval}(\mathcal{S}_g)$ and $G_{\varphi,\mathrm{eval}(\mathcal{S}_t)}$ is that there are several copies of the nodes $\top$ and $\bot$, and moreover, from a node of the form (q, p) with $p \in \mathcal{P}$ we may have edges to both $\bot$ and $\top$: If q_1 and q_2 are glued by some instruction $X_i = \mathrm{glue}(X_j)$ of the straight-line program $\mathcal{S}_t$, where q_1 is labeled with the atomic proposition p in $\mathrm{eval}(X_j)$ but q_2 is not, then the node of $\mathrm{eval}(Y_i)$ that results from gluing (q_1, p) with (q_2, p) has edges to both $\bot$ and $\top$. But for every node q of $\mathrm{eval}(\mathcal{S}_t)$ we have: if q is labeled with p in $\mathrm{eval}(\mathcal{S}_t)$, then in $\mathrm{eval}(\mathcal{S}_g)$ there is at least one edge from (q, p) to a $\top$-node plus possibly additional edges to $\bot$-nodes, whereas if q is not labeled with p in $\mathrm{eval}(\mathcal{S}_t)$, then there are only edges from (q, p) to $\bot$-nodes. But note that in the first case (q is labeled with p), additional edges to $\bot$-nodes are not problematic. The node (q, p) belongs to Eve, and she wins a $\top$-node but loses a $\bot$-node. Hence, she will not choose an edge from (q, p) to $\bot$. A similar argument applies to nodes of the form $(q, \neg p)$; note that such a node belongs to Adam. Therefore we still have as desired $(\mathrm{eval}(\mathcal{S}_t), q) \models \varphi$ if and only if $(\mathrm{eval}(\mathcal{S}_g), (q, \varphi), \mathrm{Eve}) \in \mathrm{PARITY}$. □

Corollary 4.2. *The following problem is* PSPACE*-complete:*
INPUT: An SLP $\mathcal{S}_t$ defining a transition system $\mathrm{eval}(\mathcal{S}_t)$*, a node q_{init} of* $\mathrm{eval}(\mathcal{S}_t)$*, and a sentence φ of the modal μ-calculus.*
QUESTION: $(\mathrm{eval}(\mathcal{S}_t), q_{\mathrm{init}}) \models \varphi$ *?*
Moreover,

- *the above problem is already* PSPACE*-complete when restricted to c-bounded SLPs (for a suitable large c), and*
- *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 Theorem 3.13 and 4.1. For the lower bounds, we can use two results from [1]:

- The combined complexity of CTL for hierarchical state machines is PSPACE-complete [1, Theorem 9]; recall that a CTL-formula can be translated in polynomial time into an equivalent formula of the modal μ-calculus. Hierarchical state machines are a slightly restricted class of hierarchical graph definitions in the sense of [23]. Moreover, it is easy to see that the hierarchical state machines that are constructed in the proof of [1, Theorem 9] can be translated into 4-bounded SLPs.
- There exists already a fixed CTL-sentence, for which the model-checking problem for hierarchical state machines is PSPACE-complete [1, Theorem 11].

□

In the next section, we will see that the data complexity of the modal μ-calculus becomes polynomial time when the input graph is given by a c-bounded SLP. In fact, we will prove a polynomial upper bound for the more expressive monadic least fixpoint logic (MLFP).

5 LFP and MLFP on hierarchically defined graphs

In this section we study the complexity of the model-checking problems for the fixpoint logics LFP and MLFP on hierarchically defined graphs. We start with upper bounds in Section 5.1. In Section 5.2 we will introduce hierarchical graph definitions, which are closely related to straight-line programs, but which are more suitable for the purpose of proving lower bounds in Section 5.3.

5.1 Upper bounds for fixpoint logics on SLP-defined structures

An upper bound for the most general case (combined complexity of LFP) is given by the following theorem:

Theorem 5.1. *The following problem belongs to* EXPTIME*:*
INPUT: An SLP $\mathcal{S}$ and a sentence φ of LFP.
QUESTION: $\mathrm{eval}(\mathcal{S}) \models \varphi$*?*

Proof. We can use the standard EXPTIME-algorithm that evaluates a fixpoint formula on a finite structure by building for a subformula $\mathrm{lfp}_{\bar{x},R}\varphi(\bar{x}, R)$ a sequence of increasing approximations of the fixpoint until convergence is reached [34]. If n is the size of the structure $\mathcal{A}$ and φ is an LFP-formula, where ℓ is the nesting depth of alternating fixpoint operations and k is the maximal arity of fixpoint variables in φ, then $\mathcal{A} \models \varphi$ can be checked in time $|\varphi|^{O(1)} \cdot n^{k\cdot\ell}$ [34]. Now if the structure $\mathcal{A}$ is given by an SLP $\mathcal{S}$, then $n \in 2^{O(|\mathcal{S}|)}$. Thus, the running time is $|\varphi|^{O(1)} \cdot 2^{O(|\mathcal{S}|)\cdot k\cdot\ell}$, which is still exponential. $\square$

Only for the data complexity of MLFP on graphs given by c-bounded (for some fixed c) straight-line programs we obtain a polynomial time algorithm.

Theorem 5.2. *For every fixed MLFP sentence φ and every fixed constant $c \in \mathbb{N}$ the following problem belongs to* P*:*
INPUT: A c-bounded SLP $\mathcal{S}$
QUESTION: $\mathrm{eval}(\mathcal{S}) \models \varphi$*?*

Proof. Except for the cited results, the proof is in fact identical to a corresponding proof for first-order logic from [23], which is based on Courcelle's technique for evaluating fixed MSO formulas in linear time over graph classes of bounded tree width [5]. Let us repeat the arguments for completeness.

Let φ be a fixed MLFP-sentence of rank k. Let $\mathcal{R}$ be the fixed signature, over which φ is defined. W.l.o.g. we may assume that our c-bounded input SLP $\mathcal{S} = (X_i := t_i)_{1\leq i\leq\ell}$ is also defined over the signature $\mathcal{R}$. For every $1 \leq i \leq \ell$, the structure $\mathrm{eval}(X_i)$ can be viewed as a relational structure over some subsignature Θ_i of the *fixed signature* $\Theta = \mathcal{R} \cup \{\text{pin}(1), \ldots, \text{pin}(c)\}$. Here, $\text{pin}(i)$ is a constant symbol that denotes the i-th contact node of $\text{eval}(X_i)$. Since this signature Θ is fixed (i.e., does not vary with the input) and since moreover also the rank k is fixed in the theorem, the number of pairwise nonequivalent MLFP-sentences of rank at most k over the signature Θ is bounded by some constant $g(k)$. Thus, also the number of possible MLFP_k-theories (in the sense of Section 2.5) over the signature Θ is bounded by some constant.

The crucial fact for our polynomial time algorithm is the existence of functions $F_\oplus$, F_{forget}, F_{glue}, and F_f (where $f : \{1, \ldots, n\} \to \{1, \ldots, n\}$ is bijective, $n \leq c$) over the set of all MLFP_k-theories over the signature Θ such that:

$$\begin{aligned}
\text{MLFP}_k(G_1 \oplus G_2) &= F_\oplus(\text{MLFP}_k(G_1), \text{MLFP}_k(G_2)) && (3)\\
\text{MLFP}_k(\text{forget}(G)) &= F_{\text{forget}}(\text{MLFP}_k(G)) && (4)\\
\text{MLFP}_k(\text{glue}(G)) &= F_{\text{glue}}(\text{MLFP}_k(G)) && (5)\\
\text{MLFP}_k(\text{rename}_f(G)) &= F_f(\text{MLFP}_k(G)) && (6)
\end{aligned}$$

The existence of F_{forget}, F_{glue}, and F_f follows easily, since the graph-operations forget, glue, and rename_f can be defined by a quantifier free transductions [6, 25]. For the existence of $F_\oplus$ see [26, Theorem 10] and [3, Theorem 28]; it is based on an Ehrenfeucht-Fraissé game for MFLP [2, 3, 7]. Note that the functions in (3)–(6) do not depend on the input; they can be assumed to be given hard-wired.

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

1. If $X_i := t_i$ is a definition from $\mathcal{S}$ such that t_i is an n-pointed ($n \leq c$) structure G, then we calculate $\text{MLFP}_k(G)$, which is possible in polynomial time [14] and replace the definition $X_i := t_i$ by $X_i := \text{MLFP}_k(G)$.

2. A definition of the form $X_i := X_p \oplus X_q$ is replaced by $X_i := F_\oplus(X_p, X_q)$ and similarly for definitions of the form $X_i := \text{forget}(X_j)$, $X_i := \text{glue}(X_j)$, and $X_i := \text{rename}_f(X_j)$.

Note that this is a straight-line program over a fixed finite set, namely the set of all MLFP_k-theories. Hence, we can evaluate this straight-line program in polynomial time and thereby calculate $\text{MLFP}_k(\text{eval}(\mathcal{S}))$. We finally check, whether $\varphi \in \text{MLFP}_k(\text{eval}(\mathcal{S}))$. $\square$

5.2 Hierarchical graph definitions

Fix a signature $\mathcal{R}$. A *hierarchical graph definition* (over the signature $\mathcal{R}$) is a triple $D = (N, S, P)$ such that:

(1) N is a finite set of *reference names*. Every $B \in N$ has a rank $\text{rank}(B) \in \mathbb{N}$.

(2) $S \in N$ is the *initial reference name*, where $\text{rank}(S) = 0$.

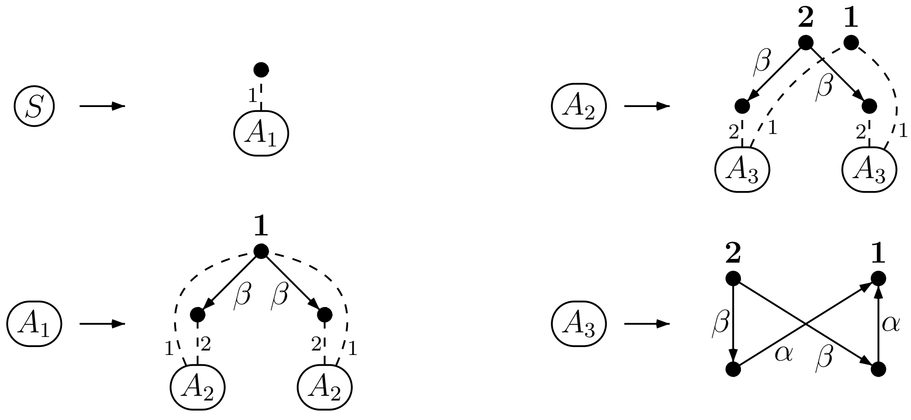


Figure 10: The productions of the hierarchical graph definition from Example 5.3

(3) P is a set of *productions*. For every $B \in N$, P contains exactly one production $B \to (\mathcal{A}, \tau, E)$, where $(\mathcal{A}, \tau)$ is a rank(B)-pointed relational structure (over the signature $\mathcal{R}$) with universe A and $E \subseteq \{(B', \sigma) \mid B' \in N, \sigma : \{1, \ldots, \text{rank}(B')\} \to A \text{ is injective}\}$ (the set of *references*).

(4) Define the relation E_D on N as follows: $(B, C) \in E_D$ if and only if for the unique production of the form $B \to (\mathcal{A}, \tau, E)$, E contains a reference of the form (C, σ). Then we require that E_D is acyclic.

The size $|D|$ of D is defined by $\sum_{(B \to (\mathcal{A}, \tau, E)) \in P} |\mathcal{A}| + |E|$. We say that D is c-bounded if rank$(B) \leq c$ for every $B \in N$ and moreover for every rule $B \to (\mathcal{A}, \tau, E)$ we have $|E| \leq c$.

In the lower bound proofs in the rest of the paper, we will only use relational structures where all relations have arity one or two. In diagrams, relations of arity two will be drawn as labeled edges, where the edge label is the name of the relation. The fact that a node v belongs to a unary relation r will be indicated by labeling v with r. Note that our definition allows several node labels for a single node. A reference (B, σ) will be drawn as a big circle with inner label B. This circle is connected via dashed lines with the nodes $\sigma(i)$ for $1 \leq i \leq \text{rank}(B)$, where the connection to $\sigma(i)$ is labeled with i. These dashed lines are also called *tentacles*. If $G = (\mathcal{A}, \tau)$ is an n-pointed relational structure, then we label the contact node $\tau(i)$ with i. In order to distinguish this label i better from node labels that correspond to unary relations, we will use, as for SLPs, boldface font for the label i.

Example 5.3. *Let $D = (N, S, P)$ be the hierarchical graph definition over the signature $\mathcal{R}$ containing two binary relational symbols α and β, where $N = \{S, A_1, A_2, A_3\}$ with* rank$(S) = 0$*,* rank$(A_1) = 1$*, and* rank$(A_2) = \text{rank}(A_3) = 2$*. The set P of productions is shown in Figure 10.*

Let us now define the structure eval(D), which results from unfolding a hierarchical graph definition $D = (N, S, P)$ (over the signature $\mathcal{R}$). For every $B \in N$ we define a rank(B)-pointed relational structure eval(B) (over the signature $\mathcal{R}$) as follows: Assume that $B \to$

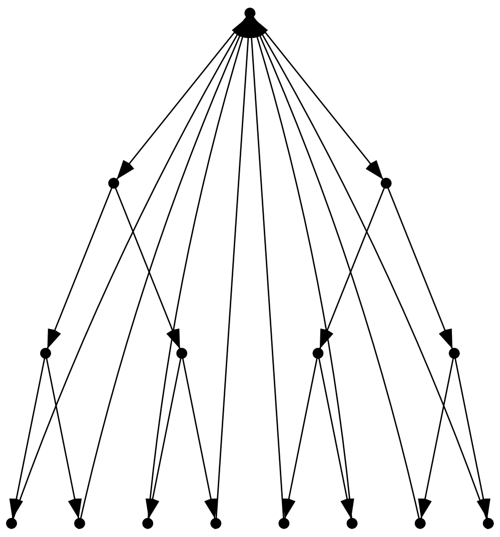


Figure 11: The graph eval(D) for the hierarchical graph definition from Example 5.3

$(\mathcal{A}, \tau, E)$ is the unique production for B in P. Let $E = \{(B_i, \sigma_i) \mid 1 \leq i \leq n\}$. Of course we may have $B_i = B_j$ for $i \neq j$. Assume that eval$(B_i) = (\mathcal{A}_i, \tau_i)$ is already defined. Then eval$(B) = ((\mathcal{A} \oplus \mathcal{A}_1 \oplus \cdots \oplus \mathcal{A}_n)/_{\equiv}, \pi_{\equiv} \circ \tau)$, where $\equiv$ is the smallest equivalence relation on the universe of $\mathcal{A} \oplus \mathcal{A}_1 \oplus \cdots \oplus \mathcal{A}_n$, which contains every pair $(\sigma_i(j), \tau_i(j))$ for $1 \leq i \leq n$ and $1 \leq j \leq \text{rank}(B_i)$. Finally, we define eval$(D) = $ eval(S); since rank$(S) = 0$ it can be viewed as an ordinary relational structure. From this definition, it is obvious that from a hierarchical graph definition D we can construct in polynomial time a straight-line program $\mathcal{S}$ with eval$(\mathcal{S}) = $ eval(D), see also [5, 23]. Moreover, if D is c-bounded, then $\mathcal{S}$ is $c(c+1)$-bounded.

Example 5.3 (continued). *The graph* eval(D) *for the hierarchical graph definition D from Example 5.3 is shown in Figure 11. Edge labels are omitted; edges going down in the tree have to be labeled with β, and the other edges going from the leafs to the root have to be labeled with α. Figure 1 in Section 2.3 shows the* 2*-pointed structure* eval(A_2)*. Two intermediate structures that occur during the unfolding of D are shown in Figure 12.*

As already mentioned in the introduction, hierarchical state machines [1] can be viewed as a particular form of hierarchical graph definitions, which is tailored towards the specification of modular reactive systems (or automata). For this purpose, modules (which correspond to right-hand sides in hierarchical graph definitions) in hierarchical state machines have two types of contact nodes: entry nodes (where control enters) and exit nodes (where control leaves the module).

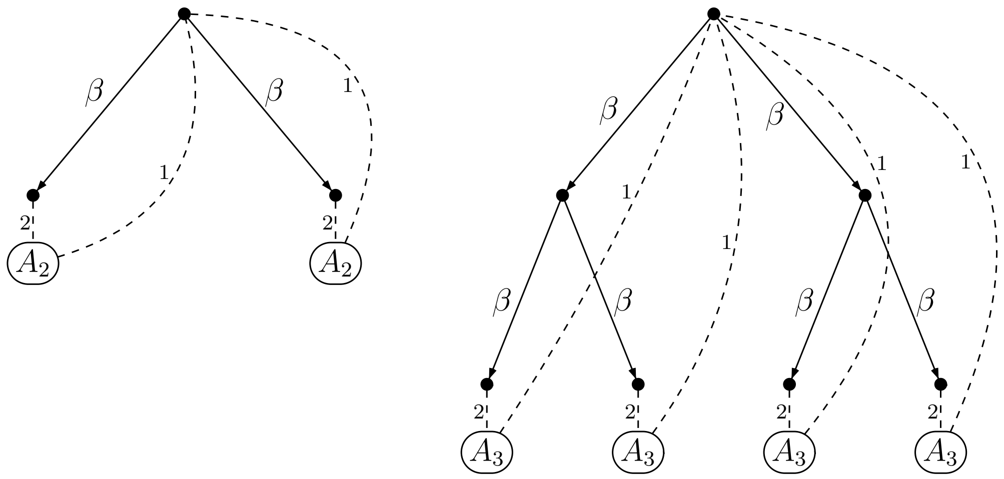


Figure 12: Two intermediate structures that arise when unfolding D from Example 5.3

5.3 Lower bounds for fixpoint logics on hierarchically defined structures

When looking at Table 1, we see that we have to prove three EXPTIME lower bounds in order to obtain together with Theorem 5.1 the EXPTIME completeness results in Table 1:

- The data complexity of LFP for c-bounded hierarchical graph definitions is EXPTIME-hard.
- The data complexity of MLFP for (unrestricted) hierarchical graph definitions is EXPTIME-hard.
- The combined complexity of MLFP for c-bounded hierarchical graph definitions is EXPTIME-hard.

We start with the data complexity of LFP for c-bounded hierarchical graph definitions.

Theorem 5.4. *There exists a fixed LFP-sentence φ such that the following problem is* EXPTIME*-hard:*
INPUT: A 2-bounded hierarchical graph definition D.
QUESTION: $\text{eval}(D) \models \varphi$*?*

Proof. Let us fix a deterministic exponential time Turing machine $T = (Q, \Sigma, q_0, q_f, \delta)$ with an EXPTIME-complete membership problem. Note that Q is the set of states, Σ is the tape alphabet, q_0 is the initial state, q_f is the unique accepting state, and δ is the transition function. W.l.o.g. assume that T operates in time 2^n on any input of length n. Let $\Box \in \Sigma$ be the blank symbol of T. We assume that the tape cell 1 (resp. 2^n) always contains the

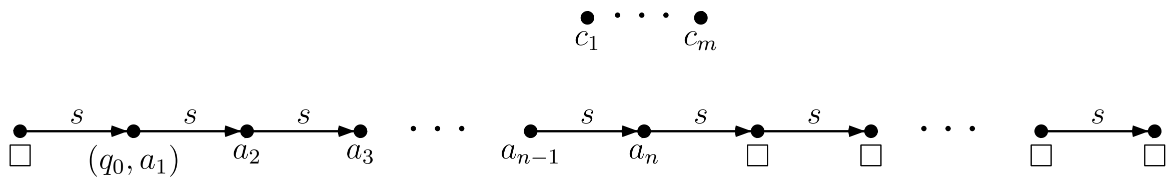


Figure 13: The structure eval(D) from the proof of Theorem 5.4

blank symbol $\square$. Let $\Gamma = \Sigma \cup (Q \times \Sigma)$ and let $c_1, \ldots, c_m$ be an arbitrary enumeration of Γ. A configuration of the machine can be encoded as a word over Γ of length 2^n, where exactly one position contains a symbol from $Q \times \Sigma \subseteq \Gamma$. We view every $c \in \Gamma$ as a relational symbol of arity one, i.e., as a node label. Let Δ be the set of all tuples $(c_0, c_1, c_2, c) \in \Delta$ such that the following is true: If at some point of time t three consecutive tape positions $i-1$, i, and $i+1$ contain the symbols c_0, c_1, and c_2, respectively, then at time $t+1$ the tape cell i contains the symbol c. Let $w = a_1 \cdots a_n$ be an input of length n for T. Then the initial tape content is $\square a_1 \cdots a_n \square \cdots \square$ and the read-write head scans the first symbol a_1 of w. It is straight-forward to construct a 2-bounded hierarchical graph definition D such that eval(D) is the structure in Figure 13, where the s-chain (s is a binary relation symbol) consists of 2^n many Γ-labeled nodes. Thus, eval(D) is a chain of length 2^n encoding the initial configuration together with $|\Gamma| = m$ many isolated nodes. For every $c \in \Gamma$ there is exactly one isolated node with label c.

Tape positions and time points will be both represented as nodes of the s-chain. A triple (x, y, z), where x and y belong to the s-chain and z is the isolated c_i-labeled node, encodes the fact that in the unique computation of T on input w at time y the tape cell x contains the symbol c_i. The set of all "correct" triples for which this is actually true will be generated as a fixpoint.

In order to construct the fixed LFP-sentence φ from the theorem, we first define a few auxiliary formulas:

$$\begin{aligned}
\omega(x) &\equiv \exists y : s(x,y) \ \vee \ s(y,x) \quad (x \text{ belongs to the } s\text{-chain}) \\
\text{zero}(x) &\equiv \omega(x) \ \wedge \ \neg\exists y : s(y,x) \quad (x \text{ is the first node of the } s\text{-chain}) \\
\text{last}(x) &\equiv \omega(x) \ \wedge \ \neg\exists y : s(x,y) \quad (x \text{ is the last node of the } s\text{-chain}) \\
\text{border}(x,y,z) &\equiv (\text{zero}(x) \ \vee \ \text{last}(x)) \ \wedge \ \square(z) \\
\text{init}(x,y,z) &\equiv \omega(x) \ \wedge \ \text{zero}(y) \ \wedge \ \neg\omega(z) \ \wedge \ \bigvee_{c \in \Gamma} (c(x) \ \wedge \ c(z)) \\
\text{consistent}(z_0,z_1,z_2,z) &\equiv \neg\omega(z) \ \wedge \ \bigwedge_{i=0}^{2} \neg\omega(z_i) \ \wedge \ \bigvee_{(c_0,c_1,c_2,c)\in\Delta} (c(z) \wedge \bigwedge_{i=0}^{2} c_i(z_i))
\end{aligned}$$

$$\psi(x,y,z,R) \equiv \text{init}(x,y,z)\ \vee \text{border}(x,y,z) \vee$$
$$\exists x_0,x_2,y',z_0,z_1,z_2 \left\{ \begin{array}{l} s(x_0,x) \wedge s(x,x_2) \wedge s(y',y) \wedge \\ \text{consistent}(z_0,z_1,z_2,z) \wedge \\ R(x_0,y',z_0) \wedge R(x,y',z_1) \wedge R(x_2,y',z_2) \end{array} \right\}$$

Note that init(x,y,z) is true for a triple (x,y,z) if and only if this triple is a correct triple (in the above sense) for the initial configuration, whereas border(x,y,z) is true for a triple (x,y,z) if and only if this triple is a correct triple for the left-most or right-most tape cell. Now, the input w is accepted by T if and only if the following sentence φ is true in eval(D), where $A = \{(q_f,a) \mid a \in \Sigma\} \subseteq \Gamma$ (recall that q_f is the unique accepting state):

$$\exists s,t,u : [\text{lfp}_{(x,y,z),R}\ \psi(x,y,z,R)](s,t,u)\ \wedge\ \bigvee_{c\in A} c(u)$$

This concludes the proof of the theorem. □

If we do not restrict to c-bounded hierarchical graph definitions then an EXPTIME lower bound can be also shown for MLFP:

Theorem 5.5. *There exists a fixed MLFP-sentence φ such that the following problem is* EXPTIME*-hard:*
INPUT: A hierarchical graph definition D.
QUESTION: eval$(D) \models \varphi$*?*

Proof. As in the previous proof we start with a fixed deterministic exponential time machine $T = (Q,\Sigma,q_0,q_f,\delta)$ with an EXPTIME-complete membership problem and which operates in time 2^n on an input of length n. We make the same assumptions on T as in the previous proof. Let $\Gamma = \Sigma \cup (Q \times \Sigma)$ and let $c_1,\ldots,c_m$ be an arbitrary enumeration of Γ. Let $w = a_1a_2\cdots a_n$ be an input of length n for T and define $a_0 = \Box$ and $a_i = \Box$ for $n < i < 2^n$.

We will construct a hierarchical graph definition D such that eval(D) is the following structure $\mathcal{A}$: The universe of $\mathcal{A}$ is

$$\{(i,w) \mid 0 \le i < 2^n, w \in \{0,1\}^{\le n}\} \cup \{(i,w,c) \mid 0 \le i < 2^n, w \in \{0,1\}^n, c \in \Gamma\} \cup \{0,\ldots,n\},$$

where $\{0,1\}^{\le n}$ denotes the set of all string over $\{0,1\}$ of length at most n and $\{0,1\}^n$ denotes the set of all string over $\{0,1\}$ of length exactly n.

The idea is that the nodes (i,ε) $(0 \le i < 2^n)$ form a chain of length 2^n using a binary relation s. Here i is a point of time in the run of the machine T. Every node (i,ε) is the root of a binary tree T_i of height n. The left (resp. right) child-relation is s_0 (resp. s_1). The node set of the tree T_i is $\{(i,w) \mid 0 \le i < 2^n, w \in \{0,1\}^{\le n}\}$. A leaf (i,w) (where $|w| = n$) of the tree T_i represents the tape cell w (where w is viewed as the binary coding of a number in $\{0,\ldots,2^n-1\}$) at time i. For every node (i,w) of T_i, there is an ℓ-labeled edge (ℓ for level) to the "level-node" $|w| \in \{0,\ldots,n\}$. Using these edges, we can express

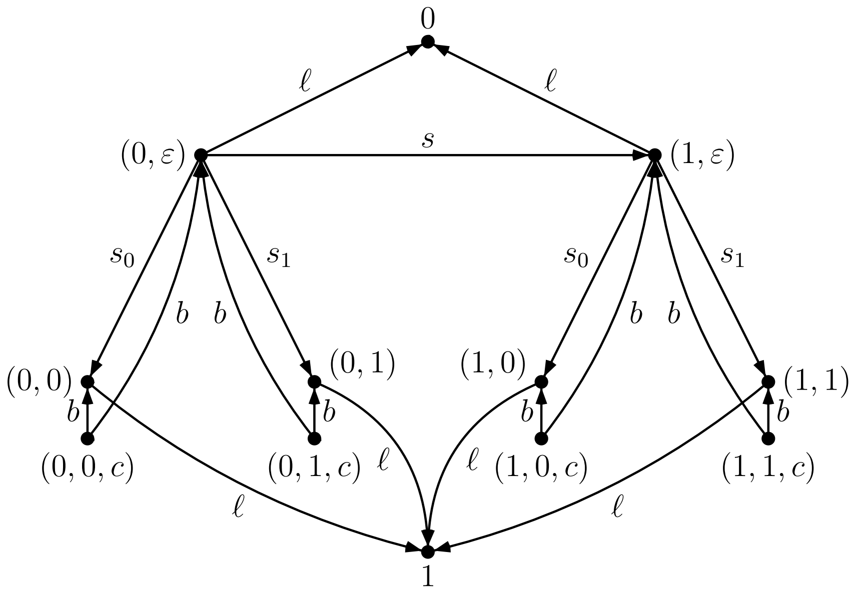


Figure 14:

that two nodes in possibly two different trees T_i and T_j are on the same level. This is needed in order to express that for two leafs (i, v) and (j, w) (of two different trees) we have $v = w$, i.e., the tape cell is the same. Finally, to every leaf (i, w) (with $|w| = n$) of T_i we attach for every $c \in \Gamma$ an additional c-labeled node (i, w, c), representing the fact that at time i tape cell w contains the symbol c. Thus, the meaning of such a node is the same as that of a triple in the previous proof. Again we will generate the set of all "correct" nodes (i, w, c) (i.e., in the unique computation on input w, at time i tape cell w actually contains the symbol c) as a (this time unary) fixpoint. From every node (i, w, c) there is a b-labeled (b for back) "back-edge" to every node along the path from the root (i, ε) of the tree T_i to the leaf (i, w). An additional unary relation init will represent the initial configuration of the machine T. It contains a triple $(0, w, a_i)$ if and only if w is the binary coding of i ($\text{bin}(i) = w$ for short). For the trivial case $n = 1$ the graph $\text{eval}(D)$ without the init-relation is shown in Figure 14, where we furthermore assume that $\Gamma = \{c\}$ has only one element. Formally, the relations of $\mathcal{A}$ are ($\preceq$ denotes the prefix relation on strings):

$$\begin{aligned}
s &= \{[(i,\varepsilon),(i+1,\varepsilon)] \mid 0 \le i < 2^n - 1\} \\
s_0 &= \{[(i,w),(i,w0)] \mid 0 \le i < 2^n, w \in \{0,1\}^{<n}\} \\
s_1 &= \{[(i,w),(i,w1)] \mid 0 \le i < 2^n, w \in \{0,1\}^{<n}\} \\
\ell &= \{[(i,w),|w|] \mid 0 \le i < 2^n, w \in \{0,1\}^{\le n}\} \\
b &= \{[(i,w,c),(i,v)] \mid 0 \le i < 2^n, w \in \{0,1\}^n, v \preceq w, c \in \Gamma\} \\
c &= \{(i,w,c) \mid 0 \le i < 2^n, w \in \{0,1\}^n, c \in \Gamma\} \text{ for every } c \in \Gamma
\end{aligned}$$

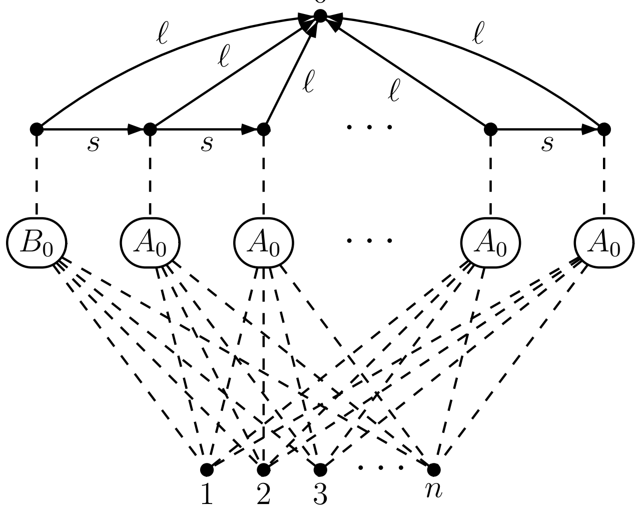


Figure 15: The structure generated from the initial variable S

$$\text{init} = \{(0, w, a_i) \mid w \in \{0,1\}^n, \text{bin}(i) = w\}$$

Let us now sketch a hierarchical graph definition D that generates this structure. It is straight-forward to generate from the initial reference name S the structure shown in Figure 15, where the s-chain consists of 2^n many nodes. Here, A_0 and B_0 are reference names. Using additional reference names $A_1, \dots, A_{n-1}$, we generate from the A_0-labeled reference the binary trees T_j $(1 \leq j < 2^n)$ as well as the ℓ-labeled edges to the level-nodes from $\{1, \dots, n\}$. The rule for A_{i-1} $(1 \leq i \leq n)$ is shown in Figure 16. The rule for A_n is shown if Figure 17; it generates the c-labeled $(c \in \Gamma)$ nodes and the b-labeled back-edges. Every reference name A_i $(0 \leq i \leq n)$ has rank $n+1$. The first $i+1$ tentacles (labeled with $0, \dots, i$ in Figure 16) of an A_i-labeled reference e access those nodes of the binary tree that were produced by ancestor-references of e. These nodes form a path starting at the root of the tree. The last $n-i$ tentacles (labeled with $i+1, \dots, n$ in Figure 16) access the level-nodes $i+1, \dots, n$ of the structure $\mathcal{A}$. From the reference B_0 we generate the tree T_0. Recall that T_0 is the same tree as T_i for $i > 0$ except that every node of the form $(0, w, a_i)$ with $\text{bin}(i) = w$ belongs to the unary init-relation. The rules for generating T_0 are similar to the rules for the reference name A_i $(0 \leq i \leq n)$, we leave the details to the reader.

Let us now describe a fixed MLFP-sentence φ such that $\text{eval}(D) \models \varphi$ if and only if the machine T accepts the input word w. We first define a few auxiliary formulas:

$$\gamma(x) \equiv \bigvee_{a \in \Gamma} c(x) \quad (x \text{ is a node from } \{(i, w, c) \mid 0 \leq i < 2^n, w \in \{0,1\}^n, c \in \Gamma\})$$

$$\text{leaf}(x) \equiv \neg\gamma(x) \ \wedge \ \neg\exists y : s_0(x, y) \vee \ell(y, x) \quad (x \text{ is a node from } \{(i, w) \mid 0 \leq i < 2^n, w \in \{0,1\}^n\}, \text{ i.e. } x \text{ is a leaf of the tree } T_i)$$

$$\text{succ-time}(x, y) \equiv \gamma(x) \ \wedge \ \gamma(y) \ \wedge \ \exists x', y' : b(x, x') \ \wedge \ b(y, y') \ \wedge \ s(x', y')$$

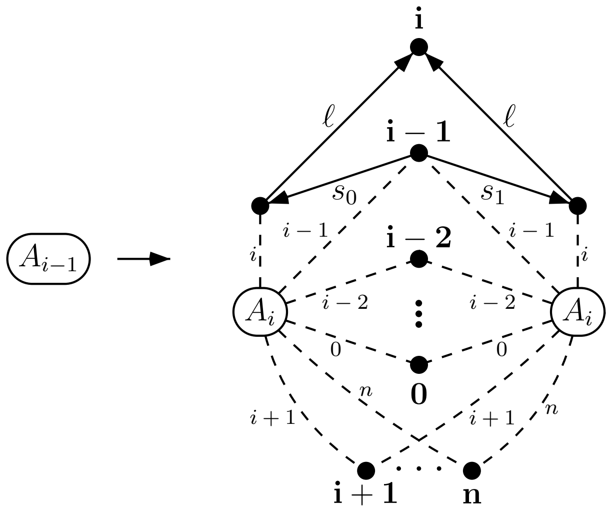


Figure 16: Production rule for A_{i-1}

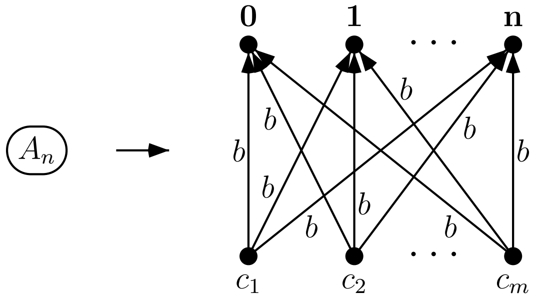


Figure 17: Production rule for A_n

$$\text{succ-pos}(x,y) \equiv \gamma(x) \;\wedge\; \gamma(y) \wedge \\ \exists x', x'', y', y'', z \left\{ \begin{array}{l} b(x,x') \;\wedge\; b(y,y') \;\wedge\; \text{leaf}(x') \;\wedge\; \text{leaf}(y') \wedge \\ s_0(z,x'') \;\wedge\; s_1(z,y'') \wedge \\ [\text{lfp}_{u,U}(u = x'' \;\vee\; \exists v \in U : s_1(v,u))](x') \wedge \\ [\text{lfp}_{u,U}(u = y'' \;\vee\; \exists v \in U : s_0(v,u))](y') \end{array} \right\}$$

$$\text{same-pos}(x,y) \equiv \gamma(x) \;\wedge\; \gamma(y) \wedge \\ \forall x', y' \left\{ \begin{array}{l} b(x,x') \wedge b(y,y') \wedge \exists z : (\ell(x',z) \wedge \ell(y',z)) \Rightarrow \\ \bigwedge_{i=0,1} \exists x'' : s_i(x'',x') \Leftrightarrow \exists y'' : s_i(y'',y') \end{array} \right\}$$

$$\text{border}(x) \equiv \square(x) \;\wedge\; \exists x', y \left\{ \begin{array}{l} b(x,x') \;\wedge\; b(x,y) \;\wedge\; \text{leaf}(x') \wedge \\ \exists z : (s(y,z) \;\vee\; s(z,y)) \wedge \\ \bigvee_{i=0,1} [\text{lfp}_{u,U}(u = y \;\vee\; \exists v \in U : s_i(v,u))](x') \end{array} \right\}$$

The formula succ-time(x,y) expresses that the time point associated with the node y is one plus the time point associated with the node x. The formula succ-pos(x,y) expresses that the nodes x and y belong to the same binary tree (i.e., the point of time is the same) and moreover the tape position associated with y is one plus the tape position associated with x. This is expressed by saying that the leafs x' and y', to which x and y, respectively, are associated, have a common predecessor z in the tree such that the unique path from z to x' (resp. y') belongs to the relation $s_0 \circ s_1^*$ (resp. $s_1 \circ s_0^*$). The formula same-pos(x, y) expresses that the tape positions associated to x and y are the same, but x and y may belong to different trees. For this, we have to say that whenever x' and y' can be reached via a b-labeled back-edge from x and y, respectively, and x' and y' are one the same level (i.e., $\exists z : \ell(x', z) \wedge \ell(y', z)$), then x' is an s_i-successor of its parent node in the tree if and only if y' is an s_i-successor of its parent node ($i \in \{0, 1\}$). Finally, border(x) expresses that the blank symbol is associated to x and that the tape position associated to x is either the left-most or the right-most one. Note that $b(x, y) \wedge \exists z : (s(y, z) \vee s(z, y))$ says that y is the root of the tree to which x belongs. Now let $\psi(x, X)$ be the following formula, where Δ has the same meaning as in the previous proof:

$$\mathrm{init}(x) \;\vee\; \mathrm{border}(x) \;\vee\; \exists x_1, x_2, x_3 \in X \left\{ \begin{array}{l} \displaystyle\bigvee_{(c_1,c_2,c_3,c)\in\Delta} c(x) \wedge \bigwedge_{i=1}^{3} c_i(x_i) \wedge \\ \displaystyle\bigwedge_{i=1}^{2} \text{succ-pos}(x_i, x_{i+1}) \wedge \\ \displaystyle\bigwedge_{i=1}^{3} \text{succ-time}(x_i, x) \;\wedge\; \text{same-pos}(x_2, x) \end{array} \right\}$$

Let $A = \{(q_f, a) \mid a \in \Sigma\} \subseteq \Gamma$. Then w is accepted by the Turing-machine T if and only if $\mathrm{eval}(D) \models \exists z : \bigvee_{c \in A} c(z) \;\wedge\; [\mathrm{lfp}_{x,X}\psi(x, X)](z)$. □

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

Theorem 5.6. *The following problem is* EXPTIME*-hard:*
INPUT: A 2-bounded hierarchical graph definition D and an MLFP-sentence φ.
QUESTION: $\mathrm{eval}(D) \models \varphi$*?*

Proof. Since EXPTIME equals alternating polynomial space, we can start with a fixed alternating PSPACE-machine $T = (Q, \Sigma, q_0, q_f, \delta)$ with an EXPTIME-complete membership problem. Here $\delta \subseteq Q \times \Sigma \times Q \times \Sigma \times \{L, R\}$ is the transition relation. A tuple (q, a, p, b, L) for instance means that if the machine T is in state q and reads an a, then it may enter state p, write b, and move left. W.l.o.g. assume that T operates in space n on an input of length n. Let $\Gamma = \Sigma \cup (Q \times \Sigma)$. Let $w = a_0 a_1 \cdots a_{n-1} \in \Sigma^n$ be an input for the the machine T. A configuration of T is a word from the language $\mathcal{C} = \bigcup_{i=0}^{n-1} \Sigma^i (Q \times \Sigma) \Sigma^{n-1-i} \subseteq \Gamma^n$. From n, it is easy to construct a 2-bounded hierarchical graph definition D such that $\mathrm{eval}(D) = (\mathrm{pref}(\mathcal{C}), (s_a)_{a \in \Gamma})$, where $\mathrm{pref}(\mathcal{C})$ is the set of all prefixes of words in the language $\mathcal{C}$ and $s_a = \{(c, ca) \mid c, ca \in \mathrm{pref}(\mathcal{C})\}$. The leafs of $\mathrm{eval}(D)$ precisely correspond to the configurations of T of length n. First of all, let us define a formula $\varphi(x_1, x_2)$ such that

$\mathrm{eval}(D) \models \varphi(c_1, c_2)$ if and only if c_1 and c_2 are leafs of $\mathrm{eval}(D)$ and the configuration represented by c_1 can evolve in one step into the configuration represented by c_2. For this we need a formula same-label(z_1, x_1, z_2, x_2), saying that the path from z_1 to x_1 in the tree $(\mathrm{pref}(\mathcal{C}), (s_a)_{a\in\Gamma})$ is labeled with the same word as the path from z_2 to x_2. This can be expressed as follows:

$$\bigvee_{i=0}^{n-2} \exists u_1, \ldots, u_{i+1}, v_1, \ldots, v_{i+1} \left\{ \begin{array}{l} z_1 = u_1 \ \wedge\ z_2 = v_1 \ \wedge\ x_1 = u_{i+1} \ \wedge\ x_2 = v_{i+1} \ \wedge \\ \displaystyle\bigwedge_{j=1}^{i} \bigvee_{a\in\Sigma} s_a(u_j, u_{j+1}) \ \wedge\ s_a(v_j, v_{j+1}) \end{array} \right\}$$

Now we can define $\varphi(x_1, x_2)$ as follows:

$$\begin{aligned} \varphi(x_1, x_2) = & \bigwedge_{a\in\Gamma, i=1,2} \neg\exists y : s_a(x_i, y) \wedge \\ & \exists y, y_1, y_2, z_1, z_2 : \text{same-label}(z_1, x_1, z_2, x_2) \wedge \\ & \Bigg(\bigvee_{\substack{(q,a,p,b,L)\in\delta,\\ c\in\Sigma}} (s_c(y, y_1) \wedge s_{(q,a)}(y_1, z_1) \wedge s_{(p,c)}(y, y_2) \wedge s_b(y_2, z_2)) \vee \\ & \bigvee_{\substack{(q,a,p,b,R)\in\delta,\\ c\in\Sigma}} (s_{(q,a)}(y, y_1) \wedge s_c(y_1, z_1) \wedge s_b(y, y_2) \wedge s_{(p,c)}(y_2, z_2)) \Bigg). \end{aligned}$$

The following formula univ(x) expresses that the leaf x represents a configuration, where the current state is a universal state:

$$\mathrm{univ}(x) = \bigvee_{i=0}^{n} \exists x_0, \ldots, x_i : \bigvee_{q\in Q_\forall, a\in\Sigma} s_{(q,a)}(x_0, x_1) \ \wedge\ \bigwedge_{j=1}^{i-1} \bigvee_{a\in\Sigma} s_a(x_j, x_{j+1}) \ \wedge\ x = x_i$$

Similarly we can construct a formula exist(x) (resp. accept(x)) expressing that x represents a configuration, where the current state is an existential (resp. the accepting) state. Now let us define the formula $\psi(x, X)$ by:

$$\mathrm{accept}(x) \ \vee\ (\mathrm{univ}(x) \ \wedge\ \forall y : \varphi(x, y) \Rightarrow y \in X) \ \vee\ (\mathrm{exist}(x) \ \wedge\ \exists y : \varphi(x, y) \wedge y \in X).$$

Then w is accepted by the Turing-machine T if and only if

$$\mathrm{eval}(D) \models \exists x_0 \cdots \exists x_n : s_{(q_0,a_0)}(x_0, x_1) \ \wedge\ \bigwedge_{i=1}^{n-1} s_{a_i}(x_i, x_{i+1}) \ \wedge\ [\mathrm{lfp}_{x,X}\psi(x, X)](x_n).$$

This concludes the proof of the theorem. □

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] U. Bosse. An "Ehrenfeucht-Fraïssé Game" for fixpoint logic and stratified fixpoint logic. In *Proceedings of the 6th Workshop on Computer Science Logic (CSL '92)*, number 707 in Lecture Notes in Computer Science, pages 100–114. Springer, 1993.

[3] U. Bosse. *Zur Modelltheorie der Fixpunktlogik.* PhD thesis, Universität Freiburg, 1994.

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

[5] 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.

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

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

[8] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (FOCS'91)*, pages 132–142. IEEE Computer Society Press, 1991.

[9] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model checking for the μ-calculus and its fragments. *Theoretical Computer Science*, 258(1-2):491–522, 2001.

[10] E. A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science (LICS'86)*, pages 267–278. IEEE Computer Society Press, 1986.

[11] 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.

[12] E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games.* Number 2500 in Lecture Notes in Computer Science. Springer, 2002.

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

[14] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68(1–3):86–104, 1986.

[15] M. Jurdziński. Deciding the winner in parity games is in UP and co-UP. *Information Processing Letters*, 68(3):119–124, 1998.

[16] M. Jurdziński. Small progress measures for solving parity games. In *Proceedings of the17th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2000)*, number 1770 in Lecture Notes in Computer Science, pages 290–301. Springer, 2000.

[17] O. Kupferman and M. Y. Vardi. An automata-theoretic approach to reasoning about infinite-state systems. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000)*, number 1855 in Lecture Notes in Computer Science, pages 36–52. Springer, 2000.

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

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

[20] 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.

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

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

[23] M. Lohrey. Model-checking hierarchical structures. to appear in *Journal of Computer and System Sciences*, 2007.

[24] M. Lohrey and S. Maneth. The complexity of tree automata and XPath on grammar-compressed trees. *Theoretical Computer Science*, 363(2):196–210, 2006.

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

[26] J. A. Makowsky and E. V. Ravve. Incremental model checking for fixed point properties on decomposable structures. `http://www.cs.technion.ac.il/~admlogic/TR/readme.html`, 1995.

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

[28] 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.

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

[30] J. Obdržálek. Fast mu-calculus model checking when tree-width is bounded. In *Proceedings of the 15th International Conference on Computer Aided Verification (CAV 2003)*, number 2725 in Lecture Notes in Computer Science, pages 80–92. Springer, 2003.

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

[32] W. Plandowski and W. Rytter. Complexity of language recognition problems for compressed words. In *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 262–272. Springer, 1999.

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

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

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

[36] I. Walukiewicz. Model checking CTL properties of pushdown systems. In *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2000)*, number 1974 in Lecture Notes in Computer Science, pages 127–138. Springer, 2000.

[37] I. Walukiewicz. Pushdown processes: games and model-checking. *Information and Computation*, 164(2):234–263, 2001.