Capturing Bisimulation-Invariant Complexity Classes with Higher-Order Modal Fixpoint Logic

Martin Lange, Etienne Lozes

Capturing Bisimulation-Invariant Complexity Classes with Higher-Order Modal Fixpoint Logic

Martin Lange and Etienne Lozes

School of Electrical Engineering and Computer Science
University of Kassel, Germany*

Abstract. Polyadic Higher-Order Fixpoint Logic (PHFL) is a modal fixpoint logic obtained as the merger of Higher-Order Fixpoint Logic (HFL) and the Polyadic μ-Calculus. Polyadicity enables formulas to make assertions about tuples of states rather than states only. Like HFL, PHFL has the ability to formalise properties using higher-order functions. We consider PHFL in the setting of descriptive complexity theory: its fragment using no functions of higher-order is exactly the Polyadic μ-Calculus, and it is known from Otto's Theorem that it captures the bisimulation-invariant fragment of PTIME. We extend this and give capturing results for the bisimulation-invariant fragments of EXPTIME, PSPACE, and NLOGSPACE.

1 Introduction

Higher-Order Fixpoint Logic. Higher-Order Fixpoint Logic (HFL) [1] is a modal logic obtained by combining the modal μ-calculus [2] and the simply typed λ-calculus. The modal μ-calculus is found in HFL as formulas only using the base type, and consequently they denote predicates over the states of a transition system. HFL formulas of higher types which are formed using λ-abstraction for example denote predicate transformers, predicate transformer transformers and so on which can be defined recursively by means of least and greatest fixpoints.

It is known that model-checking formulas with recursive predicate transformers of order at most k is k-EXPTIME complete [3]. On the other hand, its expressiveness is poorly understood and natural questions like a capturing automaton model, a capturing game semantics, the existence of an alternation hierarchy, or the role of fixpoints in a guarded fragment have not been addressed sufficiently yet. This work provides a first step towards the understanding of the expressiveness of higher-order recursive definitions in terms of descriptive complexity.

Descriptive (Bisimulation-Invariant) Complexity. Descriptive complexity studies characterisations of classes of decision problems through means of formal

* The European Research Council has provided financial support under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no 259267.

descriptions of such problems, for instance through logical formulas. A logic defines a class of decision problems, namely the membership problem for the class of models of each of the logic's formulas.

One of the main aims of descriptive complexity theory is to provide characterisations of complexity classes in terms of logics, for instance Fagin's Theorem [4] stating that NP consists of *exactly* those problems which can be described by a formula of existential Second-Order Logic ($\exists\mathsf{SO}$). Thus, $\exists\mathsf{SO}$ *captures* NP.

The benefit of such capturing results are the characterisations of complexity classes without reference to a particular machine model or typical ressource bounds in terms of time and space consumption.

Many characterisations of known complexity classes in terms of logics have been found since: Second-Order Logic (SO) captures the polynomial time hierarchy PH [5], PSPACE is captured by SO enriched with a transitive closure operator [6] or, equivalently, First-Order Logic with an operator to define partial fixpoints [7], and so on. P has yet to be captured by a logic; it is known, though, that First-Order Logic with a least fixpoint operator captures P over the class of totally ordered structures [8, 9]. For a more detailed picture of results known in the area of descriptive complexity theory we refer to the respective literature [10, 11].

Another interesting result in a similar style is Otto's Theorem [12] about the polyadic μ-calculus [12, 13]. The polyadic μ-calculus is a variant of the (monadic) μ-calculus where formulas denote predicates of any arity as opposed to just monadic ones. The polyadic μ-calculus, like the monadic one, cannot distinguish between bisimilar structures; thus, it can only define bisimulation-invariant graph problems [14, 15]. Moreover, model-checking algorithms for the polyadic μ-calculus are slightly similar to the ones for the monadic μ-calculus, and in particular all problems expressed in the modal μ-calculus can be decided in P. Otto's Theorem states the converse of these: if a problem p is in the class $\mathsf{P}/_{\sim}$ of problems that are both bisimulation-invariant and decidable in P, then p can be expressed by a formula of the polyadic modal μ-calculus. In other words, the polyadic modal μ-calculus captures $\mathsf{P}/_{\sim}$.

Contributions. Here we address the question of the expressiveness of higher-order fixpoints in HFL by extending Otto's Theorem to higher orders. We define PHFL, the polyadic version of HFL and we turn our attention to the first-order fragment $\mathsf{PHFL}(1)$ of PHFL. Here, the term *order* refers to the typing order of functions used in the formulas. Thus, the fragment $\mathsf{PHFL}(0)$ of order 0 contains no proper functions, and it is equal to the polyadic μ-calculus. Note that there is a difference with the term *order* used in predicate logics: the polyadic μ-calculus is in fact a fragment of second-*order* predicate logic. The two interpretations of the term *order* are closely related: the fragment of formulas with functions of typing order at most k can be seen as a fragment of order $(k+2)$ predicate logic. We simply prefer to use the typing order for the indexing of fragments because then the lowest fragment is $\mathsf{PHFL}(0)$ instead of $\mathsf{PHFL}(2)$.

Our first contribution is to show that $\mathsf{PHFL}(1)$ captures the complexity class $\mathsf{EXPTIME}/_\sim$. We then turn our attention to tail-recursive functions. It is well-known that such functions are usually more space efficient than arbitrary recursive functions. Our second contribution is to give a formal account of this fact: we show that the fragment $\mathsf{PHFL}(1, \mathsf{tail})$ of order-1 tail-recursive functions captures $\mathsf{PSPACE}/_\sim$. We also develop the idea of tail-recursiveness for the polyadic μ-calculus, i.e. the fragment without proper functions, and obtain a fragment $\mathsf{PHFL}(0, \mathsf{tail})$ that captures $\mathsf{NLOGSPACE}/_\sim$ on structures equipped with a pre-order which induces a total order on the equivalence classes w.r.t. bisimilarity. This pre-order is to $\mathsf{NLOGSPACE}/_\sim$ and $\mathsf{PHFL}(0, \mathsf{tail})$ what a total order is to P and $\mathsf{FO[LFP]}$: it enables the definition of iterations via fixpoint operators. Interestingly, the cut-off point marking the apparent need for such an order in the bisimulation-invariant complexity hierarchy lies below that in the non-bismulation-invariant world, namely between $\mathsf{NLOGSPACE}/_\sim$ and $\mathsf{P}/_\sim$ rather than between P and NP.

Related Work. While this paper has taken the approach of characterising complexity classes classes by typing and syntactic restrictions, descriptive complexity theory predominantly has characterised complexity classes in terms of fixpoint combinators (like TC, LFP, or PFP), with the notable exception of characterisations based on Horn and Krom clauses [16].

In a different setting, higher-order grammars have been a topic intensively studied in the 80s that has recently revived in the context of verification of higher-order programs. A problem still open is whether languages defined by such grammars are context sensitive, or in other words if they belong to the complexity class $\mathsf{NLINSPACE}$ (non-deterministic linear space). Recent progresses on this problem have been achieved by Kobayashi *et al* [17], who showed that this is at least the case up to order 2 for tree languages and order 3 for word languages. Beside the fact that this line of research does not target a capturing result, the most significant difference with our work is that we consider a polyadic μ-calculus, or in different words, an automaton model that uses multiple tapes, whereas collapsible pushdown automata (the automaton model for higher-order grammars) only work with one tape.

Implicit complexity is another line of research that aims at ensuring the complexity of the execution of higher-order programs through typing. Our work here is not concerned with the time complexity of performing β-reductions or the space needed to represent the reduced terms but in the complexity of the queries defined by the formulas we consider (which are invariant under β-reduction).

Outline. In section 2 we recall Otto's Theorem that states that the polyadic μ-calculus captures $\mathsf{P}/_\sim$. In section 3, we introduce the higher-order polyadic μ-calculus. Section 4 establishes that order 1 captures $\mathsf{EXPTIME}/_\sim$. Section 5 studies the tail-recursive fragment and establishes that there, order 1 captures $\mathsf{PSPACE}/_\sim$ and order 0 $\mathsf{NLOGSPACE}/_\sim$. Due to space constraints, some details are missing and can be found in a longer version [18].

2 Background

Labeled transition systems, bisimulation, and queries. A *labeled transition system* (LTS) is a tuple $\mathfrak{M} = (Q, \Sigma, P, \Delta, v)$, where $Q = \{q, r, \dots\}$ is a set of states, $\Sigma = \{a, b, \dots\}$ is a finite set of actions, $P = \{p, \dots\}$ is a finite set of propositions, $\Delta \subseteq Q \times \Sigma \times Q$ is the set of labeled transitions, and $v : P \to 2^Q$ is a valuation that associates to every proposition a set of states. We write $q_1 \xrightarrow{a} q_2$ for $(q_1, a, q_2) \in \Delta$ and $q \models p$ for $q \in v(p)$.

A binary relation $R \subseteq Q^2$ is a *bisimulation* if it is a symmetric relation, and for every pair of states $(q_1, q_2) \in R$, it holds that (1) for all $a \in \Sigma$, for all $q_1' \in Q$, if $q_1 \xrightarrow{a} q_1'$, then there is $q_2' \in Q$ such that $q_2 \xrightarrow{a} q_2'$ and $q_1' \, R \, q_2'$, and (2) for all $p \in P$, if $q_1 \models p$, then $q_2 \models p$. Two states q_1, q_2 are *bisimilar*, written $q_1 \sim q_2$, if there is a bisimulation that contains the pair (q_1, q_2).

We assume a fixed encoding of a finite LTS $\mathfrak{M} = (Q, \Sigma, P, \Delta, v)$ as a word $w_{\mathfrak{M}}$ such that $|w_{\mathfrak{M}}|$ is linear in $|Q| \cdot |P| + |\Delta|$ (for instance, using a sparse matrix representation). An r-adic *query* $\mathcal{Q}$ is a set of tuples $(\mathfrak{M}, q_1, \dots, q_r)$ where $\mathfrak{M}$ is an LTS and $q_1, \dots, q_r$ are states of $\mathfrak{M}$. A query $\mathcal{Q}$ is said to belong to a complexity class $\mathcal{C}$ if the language of encodings of $\mathcal{Q}$ is in $\mathcal{C}$. A query $\mathcal{Q}$ is said to be *bisimulation-invariant* if for every two tuples $(\mathfrak{M}, \boldsymbol{q})$ and $(\mathfrak{M}', \boldsymbol{q}')$ such that $q_i \sim q_i'$ for all i, $(\mathfrak{M}, \boldsymbol{q}) \in \mathcal{Q}$ if and only if $(\mathfrak{M}', \boldsymbol{q}') \in \mathcal{Q}$.

Example 1. Let $\mathcal{Q}$ be the binary query consisting of tuples $(\mathfrak{M}, q, q')$ such that $q \sim q'$. Since bisimilarity can be decided in P, this query is in P. Moreover, since bisimilarity is a transitive relation, this query is bisimulation-invariant.

The polyadic μ-calculus. A formula of the (monadic) modal μ-calculus is often interpreted as a game played by two players (sometimes called Prover and Refuter) that alternatively move a single pebble along the transitions of an LTS. The polyadic μ-calculus is basically a multi-pebble version of this game: $d \geqslant 1$ pebbles are disposed on the states of a LTS, and for every modality $\langle a \rangle_i$ (respectively $[a]_i$), Prover (respectively Refuter) has to move the i-th pebble along an a-transition. Moreover, formulas can use the modality $\{i \leftarrow j\}$, that corresponds, in the game interpretation, to moving the i-th pebble to the same place as the j-th pebble.

Let $\mathsf{Var} = \{X, Y, Z, \dots\}$ be some fixed set of variables. Formulas of the polyadic μ-calculus $\mathcal{L}_\mu^\omega$ are given by the following grammar

$$\Phi, \Psi \ ::= \top \mid p_i \mid \Phi \vee \Psi \mid \neg\Phi \mid \langle a \rangle_i \Phi \mid \{\boldsymbol{i} \leftarrow \boldsymbol{j}\}\Phi \mid X \mid \mu X.\Phi$$

where $\boldsymbol{i} = (i_1, \dots, i_n)$ and $\boldsymbol{j} = (j_1, \dots, j_n)$ are equal-length tuples of natural numbers. As usual, we only consider formulas in which every bound variable occurs underneath an even number of negations counting from its μ-binder. We also use standard notations for derived logical connectives, namely $\wedge$, $\Rightarrow$, $\Leftrightarrow$, and $[a]_i$ for conjunction, implication, equivalence, and necessity respectively. A formula is d-adic if in each subformula p_i, $\langle a \rangle_j \Phi$, and $\{\boldsymbol{i} \leftarrow \boldsymbol{j}\}\Phi$ the indices $i, j, i_1, \dots, i_n, j_1, \dots, j_n$ are in $\{1, \dots, d\}$.

The semantics of a d-adic formula Φ is a set $[\![\Phi]\!]^d_{\mathfrak{M}}$ of d-tuples of states (see [12], and also Section 3). The r-adic query $\mathcal{Q}^r_\Phi$ associated to a closed d-adic formula Φ is the set of tuples $(\mathfrak{M}, q_1, \ldots, q_r)$ such that there is $\boldsymbol{s} \in [\![\Phi]\!]^d_{\mathfrak{M}}$ with $q_i = s_i$ for all $i = 1, \ldots, \min(r, d)$.

Example 2. A standard example of a 2-adic formula is $\Phi_\sim :=$

$$\nu X. \bigwedge_{a\in\Sigma} [a]_1\langle a\rangle_2 X \wedge [a]_2\langle a\rangle_1 X \wedge \bigwedge_{p\in P} p_1 \Leftrightarrow p_2$$

which denotes the set of pairs (q_1, q_2) such that $q_1 \sim q_2$. Thus, $\Phi_\sim$ defines bisimilarity [12, 13], and $\mathcal{Q}^2_\Phi$ is the same query as in Example 1.

Theorem 1 (Otto [12]). *Let $\mathcal{Q}$ be an r-adic query. The following two are equivalent. (1) $\mathcal{Q}$ is bisimulation-invariant and in* P*; (2) $\mathcal{Q} = \mathcal{Q}^r_\Phi$ for some $\Phi \in \mathcal{L}^\omega_\mu$.*

As a consequence of Otto's Theorem, we get for example that trace equivalence is not expressible in the polyadic modal μ-calculus (unless $\mathsf{P} = \mathsf{PSPACE}$), because of the PSPACE-completeness of trace equivalence [19].

3 A Polyadic Higher-Order Fixpoint Logic

In this section, we introduce the polyadic higher-order fixpoint logic, a logic that extends the polyadic modal μ-calculus with higher-order fixpoints *à la* Viswanathan and Viswanathan [1]. In Viswanathans' logic, order-0 formulas denote predicates, order-1 formulas denote *predicate transformers*, i.e. functions mapping predicates to predicates, and so on for higher orders. For instance, $(\lambda F.\ \lambda X.\ F\ (F\ X))\quad (\lambda Y.\ \langle a\rangle Y)\quad \top$ is equivalent to the formula $\langle a\rangle\langle a\rangle\top$. Moreover, the least fixpoint combinator can be applied to monotone predicate transformers of any order. For instance, the formula

$$\Big(\mu G.\ \lambda F.\ \lambda X.\ (F\ X) \vee (G\ (\lambda Z.\ F\ (F\ Z))\ X)\Big)\quad (\lambda Y.\ \langle a\rangle Y)\quad \top$$

is equivalent to the infinitary disjunction $\bigvee_{n\geqslant 0}\langle a\rangle^{2^n}\top$.

Formally, *formulas* $\Phi, \Psi, \ldots$, *types* $\tau, \sigma, \ldots$ and *variances* v of the polyadic higher-order fixpoint logic ($\mathsf{PHFL}(\omega)$) are defined by the grammar

$$v ::= + \mid - \mid 0 \qquad \sigma, \tau ::= \bullet \mid \sigma^v \to \tau$$
$$\Phi, \Psi ::= \top \mid p_i \mid \Phi \vee \Psi \mid \neg\Phi \mid \langle a\rangle_i\Phi \mid \{\boldsymbol{i}\leftarrow\boldsymbol{j}\}\Phi \mid X \mid \lambda X^{v,\tau}.\Phi \mid \Phi\ \Psi \mid \mu X^\tau.\Phi$$

where $X, Y, \ldots$ range over a finite set of variables, and i, j range over the set $\mathbb{N}$ of natural numbers. We use standard notations like $\Phi \wedge \Psi$, $[a]_i\Phi$, $\nu X^\tau.\Phi$, or $\Phi \Leftrightarrow \Psi$ for dual and derived connectives. The maximal arity $\mathsf{ma}(\tau)$ of a type τ is defined by induction on τ: $\mathsf{ma}(\bullet) = 1$, and $\mathsf{ma}(\tau_1 \to \tau_2 \to \cdots \to \tau_n \to \bullet) = \max(\{n\} \cup \{\mathsf{ma}(\tau_i) \mid i = 1, \ldots, n\})$. The order $\mathsf{ord}(\tau)$ of a type τ is defined by induction on τ: $\mathsf{ord}(\bullet) = 0$, and $\mathsf{ord}(\sigma \to \tau) = \max(1 + \mathsf{ord}(\sigma), \mathsf{ord}(\tau))$. The order

$$\Gamma \vdash \top : \bullet \qquad \Gamma \vdash p_i : \bullet \qquad \frac{\Gamma \vdash \Phi : \bullet}{\Gamma \vdash \langle a \rangle_i \Phi : \bullet} \qquad \frac{\Gamma \vdash \Phi : \bullet}{\Gamma \vdash \{i \leftarrow j\}\Phi : \bullet} \qquad \frac{\neg\Gamma \vdash \Phi : \tau}{\Gamma \vdash \neg\Phi : \tau}$$

$$\frac{\Gamma \vdash \Phi : \tau \qquad \Gamma \vdash \Psi : \tau}{\Gamma \vdash \Phi \vee \Psi : \tau} \qquad \frac{v \in \{+, 0\}}{\Gamma\ ,\ X^v : \tau \vdash X : \tau} \qquad \frac{\Gamma, X^v : \sigma \vdash \Phi : \tau}{\Gamma \vdash \lambda X^{v,\sigma}.\ \Phi : \sigma^v \to \tau}$$

$$\frac{\Gamma, X^+ : \tau \vdash \Phi : \tau}{\Gamma \vdash \mu X^\tau.\ \Phi : \tau} \qquad \frac{\Gamma \vdash \Phi : \sigma^+ \to \tau \qquad \Gamma \vdash \Psi : \sigma}{\Gamma \vdash \Phi\ \Psi : \tau}$$

$$\frac{\Gamma \vdash \Phi : \sigma^- \to \tau \qquad \neg\Gamma \vdash \Psi : \sigma}{\Gamma \vdash \Phi\ \Psi : \tau} \qquad \frac{\Gamma \vdash \Phi : \sigma^0 \to \tau \qquad \Gamma \vdash \Psi : \sigma \qquad \neg\Gamma \vdash \Psi : \sigma}{\Gamma \vdash \Phi\ \Psi : \tau}$$

Fig. 1. The type system of $\mathsf{PHFL}(\omega)$. The type environment $\neg\Gamma$ is the one in which every assumption $X^v : \tau$ is replaced with $X^{-v} : \tau$

of a formula Φ is $\max\{\mathsf{ord}(\tau) \mid \mu X^\tau.\Psi \text{ is a subformula of } \Phi\}$. We write $\mathsf{PHFL}(k)$ for the set of formulas where recursive predicates are annotated with types of order at most k. In particular, $\mathsf{PHFL}(0)$ is the polyadic modal μ-calculus $\mathcal{L}_\mu^\omega$. On the other hand, the 1-adic fragment of $\mathsf{PHFL}(\omega)$ is exactly Viswanathans' Higher-Order Modal Fixpoint Logic.

Given a set A and a bolean algebra B, the set of functions $f : A \to B$ is again a boolean algebra (for instance, $(\neg_{A\to B} f)(x) = \neg_B(f(x))$). A function $f : A \to B$ has variance $+$ if it is monotone, $-$ if $\neg_{A\to B} f$ is monotone, and 0 in any case. We associate to every type τ the boolean algebra D_τ as follows: (1) $D_\bullet$ is the set $\mathcal{P}(Q^d)$ of all d-adic predicates, and (2) if $\tau = \sigma^v \to \sigma'$, then D_τ is the set of functions $f : D_\sigma \to D_{\sigma'}$ that have variance v.

A term $\lambda X^{v,\tau}.\Phi$ denotes a function that expects an argument of type τ and has variance v in this argument. A type judgement is a judgement of the form $X_1^{v_1,\tau_1}, \ldots, X_n^{v_n,\tau_n} \vdash \Phi : \tau$. We say that a type judgement is derivable if it admits a derivation tree according to the rules of Figure 1.

A formula Φ is well-typed if $\vdash \Phi : \tau$ is derivable for some τ. In the remainder, we always implicitly assume that we are working with well-typed formulas and sometimes omit the type annotations.

The semantics of a well-typed formula Φ of type τ is a predicate or function living in D_τ that we are now about to define. The interpretation $[\![\Gamma]\!]$ of a type environment is the set of maps ρ that send each variable $X^v : \tau \in \Gamma$ to $\rho(X) \in D_\tau$. We write $\rho[X \mapsto \mathcal{X}]$ for the map ρ' that is equal to ρ except for $\rho'(X) = \mathcal{X}$. The interpretation $[\![\Gamma \vdash \Phi : \tau]\!]$ is a map from $[\![\Gamma]\!]$ to D_τ defined by induction on Φ as explained of Figure 2 (remember that $D_\bullet = \mathcal{P}(Q^d)$ is the set of all d-adic predicates).

Proposition 1. *For every formula Φ of ground type, for all $d \geqslant 1$, the query $\mathcal{Q}_\Phi^d$ is bisimulation-invariant.*

$$
\begin{aligned}
[\![\Gamma \vdash \top : \bullet]\!](\rho) &= Q^d \\
[\![\Gamma \vdash \langle a \rangle_i \Phi : \bullet]\!](\rho) &= \{\boldsymbol{q} \in Q^d \mid \exists \boldsymbol{q}' \in [\![\Gamma \vdash \Phi : \bullet]\!].\ \boldsymbol{q} \xrightarrow{a,i} \boldsymbol{q}'\} \\
[\![\Gamma \vdash \Phi \vee \Psi : \tau]\!](\rho) &= [\![\Gamma \vdash \Phi : \tau]\!](\rho) \sqcup_\tau [\![\Gamma \vdash \Psi : \tau]\!](\rho) \\
[\![\Gamma \vdash \neg \Phi : \tau]\!](\rho) &= \neg_\tau [\![\neg(\Gamma) \vdash \Phi : \tau]\!](\rho) \\
[\![\Gamma \vdash \{\boldsymbol{i} \leftarrow \boldsymbol{j}\} \Phi : \bullet]\!](\rho) &= \{\{\boldsymbol{i} \leftarrow \boldsymbol{j}\}(\boldsymbol{q}) \mid \boldsymbol{q} \in [\![\Gamma \vdash \Phi : \bullet]\!](\rho)\} \\
[\![\Gamma, X : \tau \vdash X : \tau]\!](\rho) &= \rho(X) \\
[\![\Gamma \vdash \mu X^\tau . \Phi]\!](\rho) &= \mathsf{LFP}\, [\![\Gamma \vdash \lambda X^{+,\tau}.\ \Phi]\!](\rho) \\
[\![\Gamma \vdash \lambda X^{v,\sigma}.\ \Phi : \sigma^v \to \tau]\!](\rho) &= \mathcal{X} \mapsto [\![\Gamma, X^v : \sigma \vdash \Phi : \tau]\!](\rho[X \mapsto \mathcal{X}]) \\
[\![\Gamma \vdash \Phi\ \Psi : \tau]\!](\rho) &= [\![\Gamma \vdash \Phi : \sigma^v \to \tau]\!](\rho)([\![\Gamma \vdash \Psi : \sigma]\!](\rho))
\end{aligned}
$$

where $\boldsymbol{q} \xrightarrow{a,i} \boldsymbol{q}'$ stands for $q_i \xrightarrow{a} q_i'$ and $q_j = q_j'$ for all $j \neq i$.

Fig. 2. Semantics of $\mathsf{PHFL}(\omega)$.

Proposition 1 can be proved for instance by extending Viswanathans' proof of the bisimulation-invariance of Higher-Order Fixpoint Logic [1] to polyadic formulas. Furthermore, using fixpoint unfolding and β-reduction it is also possible to see that over every *set* of structures, $\mathsf{PHFL}(\omega)$ is equivalent to infinitary polyadic modal logic, i.e. one with arbitrary disjuncts and conjuncts and no fixpoint quantifiers, no λ-abstraction and no formula application.

Example 3. Let $\Phi := \left(\nu F.\lambda X, Y.X \Leftrightarrow Y \wedge \bigwedge_{a \in \Sigma} F\ \langle a \rangle_1 X\ \langle a \rangle_2 Y\right) \top\ \top$. Then Φ can be unfolded to

$$
\begin{aligned}
F\ \top\ \top &= \top \Leftrightarrow \top \ \wedge\ \textstyle\bigwedge_{a \in \Sigma} F\ \langle a \rangle_1 \top\ \langle a \rangle_2 \top \\
&= \textstyle\bigwedge_{a \in \Sigma} \langle a \rangle_1 \top \Leftrightarrow \langle a \rangle_2 \top \ \wedge\ \bigwedge_{b \in \Sigma} F\ \langle ba \rangle_1 \top\ \langle ba \rangle_2 \top \\
&= \textstyle\bigwedge_{w \in \Sigma^*} \langle w \rangle_1 \top \Leftrightarrow \langle w \rangle_2 \top,
\end{aligned}
$$

where $\langle w \rangle_i$ stands for $\langle w_1 \rangle_i \langle w_2 \rangle_i \ldots \langle w_n \rangle_i$. Thus, Φ denotes those pairs (q_1, q_2) for which q_1 and q_2 have exactly the same traces, i.e Φ defines trace equivalence.

4 Capturing EXPTIME/$\sim$

The aim of this section is to show that the first-order fragment of PHFL, i.e. $\mathsf{PHFL}(1)$ captures the class of bisimulation-invariant queries which can be evaluated in deterministic exponential time. For one part of this result we show the stronger statement that queries of order k can be evaluated in k-fold exponential time.

Theorem 2. *Let $r, k \geqslant 1$, and $\Phi \in \mathsf{PHFL}(k)$. Then the query $\mathcal{Q}_\Phi^r$ is in k-*EXPTIME.

The proof essentially follows the same ideas as in the case of the (1-adic) Higher-Order Fixpoint Logic [3] (see [18] for details).

Theorem 2 shows in particular that all queries expressible in $\mathsf{PHFL}(1)$ are in EXPTIME/$\sim$. We now consider the converse implication and aim at a proof of

the following: if a query is in $\mathsf{EXPTIME}/_\sim$, then it can be expressed by a $\mathsf{PHFL}(1)$ formula. A direct, but tedious proof would encode the run of an $\mathsf{EXPTIME}$ Turing machine by a query. A more elegant proof can be obtained by making use of Immerman's characterisation of $\mathsf{EXPTIME}$ queries over structures [6] as those that are expressible in second-order logic with least fixed points ($\mathsf{SO[LFP]}$). The proof then proceeds in two steps: first, we transfer Immerman's result to the logic $\mathsf{SO[LFP]}/_\sim$ defined over labeled transition systems, and second we show how to encode $\mathsf{SO[LFP]}/_\sim$ into $\mathsf{PHFL}(1)$.

The first step is relatively easy with the main problem simply being a correct definition of the semantics of $\mathsf{SO[LFP]}/_\sim$ over LTS. We define the formulas of $\mathsf{SO[LFP]}$ (resp. $\mathsf{SO[LFP]}/_\sim$) as the ones derivable from the grammar

$$\begin{aligned}\Phi, \Psi \ ::= \ & p(x) \mid a(x,y) \mid \Phi \vee \Psi \mid \neg\Phi \mid \exists x.\Psi \mid \\ & \exists X.\Psi \mid \mathsf{LFP}(F, \boldsymbol{X}, \Phi)(\boldsymbol{Y})(\boldsymbol{x}) \mid X(\boldsymbol{x}) \mid F(\boldsymbol{X})(\boldsymbol{x})\end{aligned}$$

The semantics of $\mathsf{SO[LFP]}$ on labeled graphs is as expected (see for instance [6]). Now, in order to define the semantics of $\mathsf{SO[LFP]}/_\sim$ over LTS, we just need to map every LTS to a labeled graph and interpret the formula over this graph with the $\mathsf{SO[LFP]}$ semantics.

We call a tuple $(\mathfrak{M}, q_1, \ldots, q_r)$ *reduced* if all states of $\mathfrak{M}$ are reachable from at least one q_i, and if $\sim$ coincides with equality. To every tuple $(\mathfrak{M}, \boldsymbol{q})$, we associate the reduced tuple $\mathsf{RED}(\mathfrak{M}, \boldsymbol{q})$ obtained by quotienting with respect to $\sim$ and pruning all states that cannot be reached from at least one q_i. We say that a tuple $(\mathfrak{M}, \boldsymbol{q})$ satisfies a formula Φ of $\mathsf{SO[LFP]}/_\sim$ if the graph $\mathsf{RED}(\mathfrak{M}, \boldsymbol{q})$ satisfies Φ in the graph semantics.

Lemma 1. *A query $\mathcal{Q}$ is in $\mathsf{EXPTIME}/_\sim$ iff if it is definable in $\mathsf{SO[LFP]}/_\sim$.*

We refer to [18] for a detailed proof of this result, and now move to the more challenging part, namely the encoding of $\mathsf{SO[LFP]}/_\sim$ into $\mathsf{PHFL}(1)$. For every $\mathsf{SO[LFP]}/_\sim$ formula Φ with free second-order variables $X_1, \ldots, X_n$, we define a $\mathsf{PHFL}(1)$ formula Ψ with the same free (order 0) variables, so that the least fixpoint in $\mathsf{SO[LFP]}/_\sim$ is naturally represented by a least fixpoint in $\mathsf{PHFL}(1)$. First-order variables of $\mathsf{SO[LFP]}/_\sim$ are encoded differently. Without loss of generality, we may assume an enumeration $x_1, \ldots, x_r, x_{r+1}, \ldots x_d$ of all variables of the $\mathsf{SO[LFP]}$ formula, such that $x_1, \ldots, x_r$ are the free variables and $x_{r+1}, \ldots, x_d$ are the quantified ones. We thus code $p(x_i)$ and $a(x_i, x_j)$ as p_i and $\langle a \rangle_i \{1, 2 \leftarrow i, j\} \Phi_\sim$ respectively, where $\Phi_\sim$ is the formula that defines $\sim$. For $i > r$, we define the macro $\exists_i \Phi \ := \ \bigvee_{j=1}^{r} \{i \leftarrow j\} \mu X.\Phi \vee \bigvee_{a \in \Sigma} \langle a \rangle_i X$, where Φ is an arbitrary $\mathsf{PHFL}(1)$ formula in which X does not occur. Then $\exists_i \Phi$ defines the set of all tuples for which Φ holds once the i-th component has been replaced by some state reachable from one of the states denoted by $x_1, \ldots, x_r$. Due to the bisimulation-invariant semantics of $\mathsf{SO[LFP]}/_\sim$, this is enough to encode a first-order quantification.

We are thus left with the encoding of second-order quantifiers. There is no obvious way of adapting the same idea we used for first-order quantifiers, and our encoding of second-order quantifiers is significantly trickier.

Let us first recall that it is possible to define a 2-adic formula $\Phi_{\prec}$ that defines a transitive relation $\prec$ such that $\prec \cap \succ = \varnothing$ and $\prec \cup \succ = \not\sim$; we refer to Otto's work [12] where this formula is the crux of the proof that the polyadic μ-calculus captures $\mathsf{P}/_{\sim}$. Let $\mathfrak{M}$ be a reduced LTS, so that $\prec$ defines a total order on states, and let $\prec_{\mathsf{lex}}$ denote the lexicographic extension of $\prec$ over Q^d.

Lemma 2. *There is a predicate transformer $\langle \mathsf{dec} \rangle$ such that for every formula Φ, $\langle \mathsf{dec} \rangle\, \Phi$ denotes the upward closure of Φ with respect to $\prec_{\mathsf{lex}}$.*

The construction of $\langle \mathsf{dec} \rangle$ is rather straightforward (see [18]).

Lemma 3. *Consider the predicate transformer*

$$\mathsf{next} \;:=\; \lambda X.\, (\neg X \wedge \neg\langle \mathsf{dec} \rangle \neg X) \vee (X \wedge \langle \mathsf{dec} \rangle \neg X).$$

Then, for any predicate $\mathcal{X} \in \mathcal{P}(Q^d)$, there is $i \geqslant 0$ such that $\mathsf{next}^i(\varnothing) = \mathcal{X}$.

Proof. Consider the bijection $f : \mathcal{P}(Q^d) \to \{0, \ldots, 2^{|Q|^d} - 1\}$ defined by associating to every predicate $\mathcal{X} \in \mathcal{P}(Q^d)$ the integer $f(\mathcal{X})$ whose binary representation $b_1 b_2 \ldots b_{|Q|^d}$ is such that the ith bit b_i is equal to 1 if and only if the i-th element $\boldsymbol{q}$ of Q^d with respect to $\prec_{\mathsf{lex}}$ is in $\mathcal{X}$.

Our claim is that next maps every predicate $\mathcal{X}$ to the predicate $\mathcal{Y}$ such that $f(\mathcal{Y}) = 1 + f(\mathcal{X})$ modulo $2^{|Q|^d}$. Indeed, the ith bit in $\mathcal{Y}$ is 1 if either it is also 1 in $\mathcal{X}$ and a lower bit is 0 in $\mathcal{X}$, or it is 0 in $\mathcal{X}$ but all lower bits are 1 in $\mathcal{X}$. □

Let Φ be a $\mathsf{PHFL}(1)$ formula not containing the variable H, and let formula $\exists X.\Phi$ be defined as $\big(\mu H.\lambda X.\Phi \vee H\ (\mathsf{next}\ X)\big)\ \bot$. Then, thanks to Lemma 3, $\exists X.\Phi$ encodes a second-order existential quantification.

Lemma 4. *Let $r \geqslant 1$. For every formula Φ of $\mathsf{SO}[\mathsf{LFP}]/_{\sim}$, there is a formula Ψ of $\mathsf{PHFL}(1)$ such that $\mathcal{Q}^r_{\Phi} = \mathcal{Q}^r_{\Psi}$.*

Theorem 3. $\mathsf{PHFL}(1)$ *captures* $\mathsf{EXPTIME}/_{\sim}$ *over labeled transition systems.*

Proof. Lemmas 1 and 4 prove that every $\mathsf{EXPTIME}/_{\sim}$ query is expressible in $\mathsf{PHFL}(1)$. By Theorem 2 and Proposition 1, we know that $\mathsf{PHFL}(1)$ cannot express more than that. □

5 Tail Recursion and PSPACE$/_{\sim}$

Tail-recursive functions are functions that are never called recursively in intermediate steps of their body, either for evaluating a condition on branching, or for evaluating an argument of a function call. By analogy, we define tail-recursive formulas as the ones that can be seen as non-deterministic tail-recursive functions.

We assume from now on that the logical connective $\wedge$ is primitive in the syntax (and not just the dual of $\vee$). Without loss of generality, we restrict our attention to formulas in which every variable is bound at most once. We say

$$\bar{Y} \vdash \mathsf{tail}(p_i, \bar{X}) \qquad \frac{X \in \bar{X} \cup \bar{Y}}{\bar{Y} \vdash \mathsf{tail}(X, \bar{X})} \qquad \frac{\bar{Y} \vdash \mathsf{tail}(\Phi, \varnothing)}{\bar{Y} \vdash \mathsf{tail}(\neg\Phi, \bar{X})}$$

$$\frac{\bar{Y} \vdash \mathsf{tail}(\Phi, \bar{X})}{\bar{Y} \vdash \mathsf{tail}(\{\boldsymbol{i}\leftarrow\boldsymbol{j}\}\Phi, \bar{X})} \qquad \frac{\bar{Y} \vdash \mathsf{tail}(\Phi, \bar{X}) \qquad \bar{Y} \vdash \mathsf{tail}(\Psi, \bar{X})}{\bar{Y} \vdash \mathsf{tail}(\Phi \vee \Psi, \bar{X})} \qquad \frac{\bar{Y} \vdash \mathsf{tail}(\Phi, \bar{X})}{\bar{Y} \vdash \mathsf{tail}(\langle a \rangle_i \Phi, \bar{X})}$$

$$\frac{\bar{Y} \vdash \mathsf{tail}(\Phi, \varnothing) \qquad \bar{Y} \vdash \mathsf{tail}(\Psi, \bar{X})}{\bar{Y} \vdash \mathsf{tail}(\Phi \wedge \Psi, \bar{X})} \qquad \frac{\bar{Y} \vdash \mathsf{tail}(\Phi, \varnothing)}{\bar{Y} \vdash \mathsf{tail}([a]_i \Phi, \bar{X})}$$

$$\frac{\bar{Y} \vdash \mathsf{tail}(\Phi, \bar{X}) \qquad \bar{Y} \vdash \mathsf{tail}(\Psi, \varnothing)}{\bar{Y} \vdash \mathsf{tail}(\Phi\ \Psi, \bar{X})} \qquad \frac{\bar{Y} \cup \{Z\} \vdash \mathsf{tail}(\Phi, \bar{X})}{\bar{Y} \vdash \mathsf{tail}(\lambda Z^{\upsilon,\tau}.\Phi, \bar{X})} \qquad \frac{\bar{Y} \vdash \mathsf{tail}(\Phi, \bar{X} \cup \{Z\})}{\bar{Y} \vdash \mathsf{tail}(\mu Z^{\tau}.\Phi, \bar{X})}$$

Fig. 3. A closed formula Φ is tail-recursive if $\varnothing \vdash \mathsf{tail}(\Phi, \varnothing)$ is derivable.

that a formula $\mu X.\Phi$ is a tail-recursive definition if no subformula of Φ is of the form either $\Phi'\ \Psi$, or $\neg\Psi$, or $\Psi \wedge \Phi'$, with X occuring free in Ψ (see Figure 3 for an inductive definition). Observe that we therefore do not treat both sides of a conjunction symmetrically. A formula is tail recursive if every recursive definition is tail-recursive. We write $\mathsf{PHFL}(i, \mathsf{tail})$ for the set of tail-recursive formulas of $\mathsf{PHFL}(i)$.

Example 4. The formula $(\mu F.\lambda X.(F\ \langle a \rangle_1 X) \vee (X \wedge \langle a \rangle_2 (F\ X))\ (\mu Y.Y)$ is tail-recursive. On the other hand, the formula $\mu X.[a]_1 X$ is not tail-recursive because X occurs underneath $[a]_1$ (see Figure **??**). The formula $\mu F.\lambda X.(F\ X) \wedge (F\ (F\ X))$ is not tail-recursive either, for two different reasons: on the one hand F occurs on the left side of $\wedge$, and on the other hand F occurs in the argument $F\ X$ of F.

Theorem 4. *Let $r > 0$ and $\Phi \in \mathsf{PHFL}(1, \mathsf{tail})$. Then $\mathcal{Q}^r_\Phi$ is in* PSPACE.

Proof. The proof of this result is slightly more complicated than for Theorem 2. To achieve EXPTIME, a global model-checking algorithm that closely follows the semantics of $\mathsf{PHFL}(1)$ is enough. However, such an algorithm needs to represent functions denoted by predicate transformer in extension, which requires exponential space. If we want a model-checking algorithm running in PSPACE, we need to avoid representing functions in extension.

For a $\mathsf{PHFL}(1, \mathsf{tail})$ formula Φ, let the *recursion depth* $\mathsf{rd}(\Phi)$ of Φ be inductively defined as follows: $\mathsf{rd}(p_i) = \mathsf{rd}(X) = 0$, $\mathsf{rd}(\Phi_1 \vee \Phi_2) = \max(\mathsf{rd}(\Phi_1), \mathsf{rd}(\Phi_2))$, $\mathsf{rd}(\Phi_1 \wedge \Phi_2) = \max(\mathsf{rd}(\Phi_2), 1 + \mathsf{rd}(\Phi_1))$, $\mathsf{rd}(\Phi_1\ \Phi_2) = \max(\mathsf{rd}(\Phi_1, 1 + \mathsf{rd}(\Phi_2))$, $\mathsf{rd}(\langle a \rangle_i \Phi) = \mathsf{rd}(\lambda X.\Phi) = \mathsf{rd}(\mu X.\Phi) = \mathsf{rd}(\Phi)$, and $\mathsf{rd}(\neg\Phi) = 1 + \mathsf{rd}(\Phi)$.

If π is a list $\mathcal{X}_1, \ldots, \mathcal{X}_n$ of elements of $\mathcal{P}(Q^d)$, we write $\mathsf{hd}(\pi)$ for $\mathcal{X}_1$ and $\mathsf{tl}(\pi)$ for $\mathcal{X}_2, \ldots, \mathcal{X}_n$, and $\mathcal{X} :: \pi$ for the list π' with $\mathsf{hd}(\pi') = \mathcal{X}$ and $\mathsf{tl}(\pi') = \pi$. For simplicity, we assume without loss of generality that we work with formulas such that every variable is bound at most once. We call a variable *recursive* if it is

bound by a μ. For a function $c : \mathsf{Var} \to \mathbb{N}$ we write $c[X{+}{+}]$ for the function defined by $c[X{+}{+}](X) = 1 + c(X)$ and $c[X{+}{+}](Y) = c(Y)$ for all $Y \neq X$.

For a tuple $\boldsymbol{q} \in Q^d$, a $\mathsf{PHFL}(1, \mathsf{tail})$ formula Φ, a list $\pi \in \mathcal{P}(Q^d)^*$, a function $\rho : \mathsf{Var} \to \mathcal{P}(Q^d)$ and a function $c : \mathsf{Var} \to \mathbb{N}$, let $\mathbf{check}(\boldsymbol{q}, \Phi, \pi, \rho, c)$ be the non-deterministic recursive procedure defined as follows:

- if Φ is an atomic formula, return true if $\boldsymbol{q} \in [\![\Phi]\!](\rho)$, false otherwise;
- if $\Phi = \{\boldsymbol{i} \leftarrow \boldsymbol{j}\}\Psi$, return $\mathbf{check}(\{\boldsymbol{i} \leftarrow \boldsymbol{j}\}(\boldsymbol{q}), \Psi, \pi, \rho, c)$;
- if $\Phi = X$ and X is not a recursive variable, return true if $\boldsymbol{q} \in \rho(X)$, false otherwise;
- if $\Phi = X$ for a recursive variable X with $\mathsf{fp}_X = \mu X^\tau.\Psi$, let $N := |Q|^d$ if X is order 0, otherwise X is a l-ary predicate transformer for some $l > 0$, and we set $N := |Q|^{ld} \cdot 2^{l|Q|^d}$; if $c(X) = N$, return false, otherwise return $\mathbf{check}(\boldsymbol{q}, \Psi, \pi, \rho, c[X{+}{+}])$;
- if $\Phi = \neg\Psi$, return $\neg\mathbf{check}(\boldsymbol{q}, \Psi, \pi, \rho, c)$;
- if $\Phi = \Phi_1 \vee \Phi_2$, guess $i \in \{1, 2\}$ and return $\mathbf{check}(\boldsymbol{q}, \Phi_i, \mathcal{X}, \pi, \rho, c)$;
- if $\Phi = \langle a \rangle_i \Psi$, guess $\boldsymbol{s}$ such that $\boldsymbol{q} \xrightarrow{a,i} \boldsymbol{s}$ and return $\mathbf{check}(\boldsymbol{s}, \Psi, \pi, \rho, c)$;
- if $\Phi = \Psi_1\ \Psi_2$, compute first the set of tuples $\mathcal{X} := \{\boldsymbol{r} \mid \mathbf{check}(\boldsymbol{r}, \Psi_2, \pi, \rho, c) = \mathsf{true}\}$, then return $\mathbf{check}(\Psi_1, \mathcal{X} :: \pi, \rho, c)$;
- if $\Phi = \Phi_1 \wedge \Phi_2$, return false if $\mathbf{check}(\boldsymbol{q}, \Phi_1, \pi, \rho, c) = \mathsf{false}$, otherwise return $\mathbf{check}(\boldsymbol{q}, \Phi_2, \pi, \rho, c)$;
- if $\Phi = \lambda X.\Psi$, return $\mathbf{check}(\Psi, \mathsf{tl}(\pi), \rho[X \mapsto \mathsf{hd}(\pi)], c)$;
- if $\Phi = \mu X^\tau.\Phi'$, return $\mathbf{check}(\boldsymbol{q}, \Phi', \pi, \rho, c[X \mapsto 0])$.

Consider a fixed d-adic formula Φ with l variables and recursion depth $\mathsf{rd}(\Phi) = k$, and an LTS $\mathfrak{M}$ with n states. Encoding sets as bit vectors and integers in binary, ρ, π and c require $\mathcal{O}(n^{dl})$ space, whereas encoding $\boldsymbol{q}$ requires $\mathcal{O}(d \cdot \log n)$. Moreover, if we avoid to stack the calling context at every tail recursive call (recursive calls of the form $\mathsf{return}\ \mathbf{check}(\dots)$), then the height of the stack of calling contexts is bounded by the recursion depth k of the formula. So $\mathbf{check}(\boldsymbol{q}, \Phi, \pi, \rho, c)$ requires overall space $\mathcal{O}(k \cdot n^{dl})$. As a consequence, for a fixed parameter formula Φ, the procedure works in $\mathsf{NPSPACE}$, and by Savitch's theorem we get that $\mathcal{Q}^r_\Phi$ is in PSPACE. □

We are now interested in the proof of the converse of Theorem 4. In order to establish that $\mathsf{PHFL}(1, \mathsf{tail})$ captures $\mathsf{PSPACE}/_\sim$, we again pick a logic that captures PSPACE over graphs, transfer this result to transition systems, and encode this logic in $\mathsf{PHFL}(1, \mathsf{tail})$. The logic we consider is first-order logic with partial fixpoints ($\mathsf{FO}[\mathsf{PFP}]$) introduced by Abiteboul and Vianu [7]. The partial fixpoint $\mathsf{PFP}(f)$ of a predicate transformer $f : \mathcal{P}(Q^d) \to \mathcal{P}(Q^d)$ is the predicate defined as follows: if there is some $i \geqslant 0$ for which $f^i(\varnothing) = f^{i+1}(\varnothing)$, then $\mathsf{PFP}(f) := f^i(\varnothing)$; otherwise $\mathsf{PFP}(f) := \varnothing$. We only detail the encoding of this partial fixpoint combinator in $\mathsf{PHFL}(1, \mathsf{tail})$, since the rest of the proof does not significantly differ from the proofs of Lemmas 1 and 4 (see [18] for details).

Let $\forall_i \Phi$ denote the formula $\neg\exists_i\neg\Phi$, where $\exists_i\Phi$ is the formula we introduced for encoding first-order quantifiers. For a $\mathsf{PHFL}(1, \mathsf{tail})$ formula Φ with a free variable X, let $\Psi := \big(\mu F.\lambda X.(X \wedge \forall_1 \dots \forall_d(X \Leftrightarrow \Phi)) \vee F\ \Phi\big)\ \bot$. It can be checked

that Ψ is tail-recursive and order 1, and that it defines the partial fixpoint of $\lambda X.\Phi$.

Theorem 5. $\mathsf{PHFL}(1, \mathsf{tail})$ *captures* $\mathsf{PSPACE}/_\sim$ *over labeled transition systems.*

Next we consider tail-recursive formulas of order 0. The same algorithm that we used in the proof of Theorem 4 has a better space complexity for formulas of $\mathsf{PHFL}(0)$.

Theorem 6. *For all* $r \geqslant 0$*, for every formula* Φ *of* $\mathsf{PHFL}(0, \mathsf{tail})$*,* $\mathcal{Q}^r_\Phi$ *is in* $\mathsf{NLOGSPACE}$*.*

Proof. Consider again the procedure $\mathbf{check}(\boldsymbol{q}, \Phi, \pi, \rho, c)$ for local model-checking introduced in the proof of Theorem 4. When Φ is in $\mathsf{PHFL}(0, \mathsf{tail})$, all variables are recursive, so the parameters ρ and π are useless, and for every recursive variable X, the counter $c(X)$ remains smaller than $|Q|^d$, so c can be represented in logarithmic space. Since $\boldsymbol{q}$ can also be represented in logarithmic space, and the height of the stack of recursive calls is bounded by the constant $\mathsf{rd}(\Phi)$, **check** is a non-deterministic logarithmic space procedure. □

In order to capture $\mathsf{NLOGSPACE}/_\sim$, we consider the logic $\mathsf{FO}[\mathsf{TC}]$ whose syntax is defined as follows

$$\Phi ::= p(x_i) \mid a(x_i, x_j) \mid x_i \prec x_j \mid \Phi \vee \Psi \mid \neg\Phi \mid \exists x_i.\Phi \mid X(\bar{x}) \mid [\mathsf{TC}\ \Phi](\boldsymbol{x}, \boldsymbol{y}).$$

A formula Φ of $\mathsf{FO}[\mathsf{TC}]$ is d-adic if for every first-order variable x_i occurring in Φ the index i is smaller than $2d$. The semantics of a d-adic formula is then a binary relation $R \subseteq Q^{2d}$, with $[\![\mathsf{TC}\ \Phi]\!]$ being the reflexive transitive closure of $[\![\Phi]\!]$. As before, let $\mathsf{FO}[\mathsf{TC}]/_\sim$ be the syntactically same logic which is interpreted over reduced LTS only. Bisimulation is still definable in $\mathsf{FO}[\mathsf{TC}]$ because of the preorder $\prec$. Since bisimulation $\sim$ is P-complete [21], $\mathsf{FO}[\mathsf{TC}]/_\sim$ is very unlikely to capture $\mathsf{NLOGSPACE}/_\sim$ over all transition systems. The way to go around this problem is to assume that the preorder $\prec$ is given as part of the model.

We call *totally ordered LTS* a tuple $\mathfrak{M} = (Q, \prec, \Sigma, P, \Delta, v)$ where $\prec$ is a preoder over Q such that $\prec \cup \succ = \not\sim$. Observe that for a totally ordered LTS $\mathfrak{M}$, $(\mathfrak{M}, \boldsymbol{q})$ is reduced if and only if $\prec$ is a total order, and all states are reachable from one of the $\boldsymbol{q}$ root states, so the query containing all reduced $(\mathfrak{M}, \boldsymbol{q})$ is in $\mathsf{NLOGSPACE}$ over totally ordered LTS. Now, using the same arguments as in Lemma 1, we get that $\mathsf{FO}[\mathsf{TC}]/_\sim$ captures $\mathsf{NLOGSPACE}/_\sim$ over totally ordered LTS.

We call $\mathsf{PHFL}(0, \prec, \mathsf{tail})$ the set of formulas $\Phi[\Phi_\prec/X]$ such that Φ is in $\mathsf{PHFL}(0, \mathsf{tail})$. It follows from Theorem 6 that all queries over totally ordered LTS that are definable in $\mathsf{PHFL}(0, \prec, \mathsf{tail})$ are in $\mathsf{NLOGSPACE}/_\sim$.

Theorem 7. $\mathsf{PHFL}(0, \prec, \mathsf{tail})$ *captures* $\mathsf{NLOGSPACE}/_\sim$ *over totally ordered LTS.*

Proof. We need to show that for every $r > 0$ and every formula Φ of $\mathsf{FO}[\mathsf{TC}]$, there is a $\mathsf{PHFL}(0, \prec, \mathsf{tail})$ formula Ψ such that $\mathcal{Q}^r_\Phi = \mathcal{Q}^r_\Psi$ over totally ordered

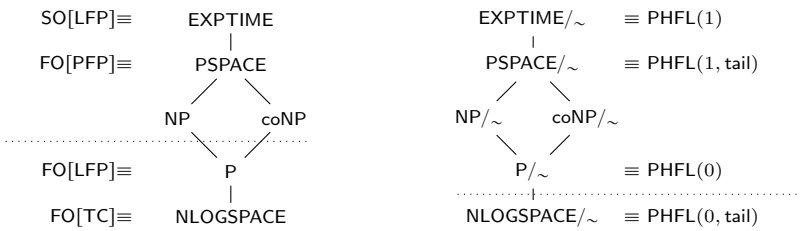

Fig. 4. Capturing bisimulation-invariant complexity classes, compared to their non-bisimulation-invariant counterparts. Below dotted lines, structures are assumed to be equipped with a total order (resp. a bisimulation-invariant preorder).

LTS. For any FO formula Φ and its already defined translation $\mathsf{tr}(\Phi)$ into a $\mathsf{PHFL}(0, \prec, \mathsf{tail})$ formula, consider the formulas

$$\Psi_0 := \bigwedge_{i=1}^{d} \{(1,2) \leftarrow (i, i+d)\} \neg \Phi_{\prec} \wedge \{(2,1) \leftarrow (i, i+d)\} \Phi_{\prec}$$
$$\Psi := \{s\}\big(\mu X.\Psi_0 \vee \exists_{2d+1} \ldots \exists_{3d}\big((\{s_1\}\mathsf{tr}(\Phi)) \wedge \{s_2\}X\big)\big)$$

where $s := \{1, \ldots, 2d \leftarrow i_1, \ldots, i_d, j_1, \ldots, j_d\}$, $\{s_1\} := \{1, \ldots, d \leftarrow 2d+1, 3d\}$ and $\{s_2\} := \{d+1, \ldots, 2d \leftarrow 2d+1, 3d\}$. Then Ψ is equivalent to $[\mathsf{TC}\ \Phi](\boldsymbol{x}, \boldsymbol{y})$ with $\boldsymbol{x} = x_{i_1}, \ldots, x_{i_d}$ and $\boldsymbol{y} = x_{j_1}, \ldots, x_{j_d}$. Moreover, as $\mathsf{tr}(\Phi)$ is closed and tail-recursive, so is Ψ. So any formula of $\mathsf{FO}[\mathsf{TC}]$ with a single application of the transitive closure has an equivalent in $\mathsf{PHFL}(0, \prec, \mathsf{tail})$; since $\mathsf{FO}[\mathsf{TC}]$ formulas have a normal form with a single application of the transitive closure [10], this proves the result. □

6 Conclusion and Further Work

The results obtained here are presented in Fig. 4 and compared to those results in the descriptive complexity of the non-bisimulation-invariant world which are being used to obtain the results of this paper.

Besides the obvious question of characterisations for $\mathsf{NP}/_{\sim}$ and $\mathsf{coNP}/_{\sim}$, bisimulation-invariant complexity classes beyond $\mathsf{EXPTIME}$ also remain to be captured. We believe that $k{-}\mathsf{EXPTIME}/_{\sim}$ is captured by $\mathsf{PHFL}(k)$ for $k > 1$ and $k{-}\mathsf{EXPSPACE}/_{\sim}$ is captured by $\mathsf{PHFL}(k-1, \mathsf{tail})$ for $k > 2$. In order to establish such results, we may want to look for other logical characterisations of these complexity classes and encode them in $\mathsf{PHFL}(\omega)$ as we did here. Although we can think about natural candidates for such logical characterisations we could not find suitable references in the literature and the generalisation of our results to higher orders and higher complexity classes is therefore deferred to future work.

References

1. Viswanathan, M., Viswanathan, R.: A higher order modal fixed point logic. In: CONCUR. Volume 3170 of LNCS., Springer (2004) 512–528
2. Kozen, D.: Results on the propositional μ-calculus. TCS **27** (1983) 333–354
3. Axelsson, R., Lange, M., Somla, R.: The complexity of model checking higher-order fixpoint logic. Logical Methods in Computer Science **3** (2007) 1–33
4. Fagin, R.: Generalized first-order spectra and polynomial-time recognizable sets. Complexity and Computation **7** (1974) 43–73
5. Stockmeyer, L.J.: The polynomial-time hierarchy. TCS **3**(1) (1976) 1–22
6. Immerman, N.: Languages that capture complexity classes. SIAM Journal of Computing **16**(4) (1987) 760–778
7. Abiteboul, S., Vianu, V.: Computing with first-order logic. Journal of Computer and System Sciences **50**(2) (1995) 309–335
8. Vardi, M.Y.: The complexity of relational query languages (extended abstract). In: STOC, ACM (1982) 137–146
9. Immerman, N.: Relational queries computable in polynomial time. Information and Control **68**(1–3) (1986) 86–104
10. Immerman, N.: Descriptive Complexity. Springer, New York (1999)
11. Grädel, E., Kolaitis, P.G., Libkin, L., Marx, M., Spencer, J., Vardi, M.Y., Venema, Y., Weinstein, S.: Finite Model Theory and its Applications. Springer (2007)
12. Otto, M.: Bisimulation-invariant PTIME and higher-dimensional μ-calculus. Theor. Comput. Sci. **224**(1-2) (1999) 237–265
13. Andersen, H.R.: A polyadic modal μ-calculus. Technical Report 145, IT Universitetet i København (1994)
14. Milner, R.: Communication and Concurrency. International Series in Computer Science. Prentice Hall (1989)
15. Stirling, C.: Modal and Temporal Properties of Processes. Texts in Computer Science. Springer (2001)
16. Grädel, E.: Capturing complexity classes by fragments of second-order logic. Theor. Comput. Sci. **101**(1) (1992) 35–57
17. Kobayashi, N., Inaba, K., Tsukada, T.: Unsafe order-2 tree languages are context-sensitive. In: FoSSaCS. Volume 8412 of Lecture Notes in Computer Science., Springer (2014) 149–163
18. Lange, M., Lozes, E.: Capturing bisimulation invariant complexity classes with higher-order modal fixpoint logic. available at http://carrick.fmv.informatik.uni-kassel.de/ lozes/tcs14-long.pdf
19. Meyer, A.R., Stockmeyer, L.J.: Word problems requiring exponential time. In: STOC, ACM (1973) 1–9
20. Tarski, A.: A lattice-theoretical fixpoint theorem and its application. Pacific Journal of Mathematics **5** (1955) 285–309
21. Àlvarez, C., Balcázar, J.L., Gabarró, J., Santha, M.: Parallel complexity in the design and analysis on concurrent systems. In: Parallel Architectures and Languages Europe, PARLE'91. Volume 505 of LNCS., Springer (1991) 288–303