Space-Efficient Fragments of Higher-Order Fixpoint Logic

Florian Bruse[1,2,⋆] and Martin Lange[1] and Etienne Lozes[2]

[1] University of Kassel, Germany
[2] LSV, ENS Paris-Saclay, CNRS, France

Abstract. Higher-Order Fixpoint Logic (HFL) is a modal specification language whose expressive power reaches far beyond that of Monadic Second-Order Logic, achieved through an incorporation of a typed λ-calculus into the modal μ-calculus. Its model checking problem on finite transition systems is decidable, albeit of high complexity, namely k-EXPTIME-complete for formulas that use functions of type order at most $k > 0$. In this paper we present a fragment with a presumably easier model checking problem. We show that so-called tail-recursive formulas of type order k can be model checked in $(k-1)$-EXPSPACE, and also give matching lower bounds. This yields generic results for the complexity of bisimulation-invariant non-regular properties, as these can typically be defined in HFL.

1 Introduction

Higher-Order Modal Fixpoint Logic (HFL) [18] is an extension of the modal μ-calculus [9] by a simply typed λ-calculus. Formulas do not only denote sets of states in labelled transition systems but also functions from such sets to sets, functions from sets to functions on sets, etc. The syntax becomes a bit more complicated because the presence of fixpoint quantifiers requires formulas to be strongly typed in order to guarantee monotonicity of the function transformers (rather than just set transformers) whose fixpoints are quantified over.

HFL is an interesting specification language for reactive systems: the ability to construct functions at arbitrary type levels gives it an enormous expressive power compared to the μ-calculus, the standard yardstick for the expressive power of bisimulation-invariant specification languages [7]. HFL has the power to express non-MSO-definable properties [11, 18, 13] like certain assume-guarantee properties; all context-free and even some context-sensitive reachability properties; structural properties like being a balanced tree, being bisimilar to a word, etc. As a bisimulation-invariant fixpoint logic, HFL is essentially an extremely powerful logic for specifying complex reachability properties.

There is a natural hierarchy of fragments HFL^k formed by the maximal function order k of types used in a formula where HFL^0 equals the modal μ-

⋆ This work was supported by a fellowship within the FITweltweit programme of the German Academic Exchange Service (DAAD).

calculus. The aforementioned examples are all expressible in fragments of low order, namely in HFL^1 or in exceptional cases only HFL^2.

Type order is a major factor for model-theoretic and computational properties of HFL. It is known that HFL^{k+1} is strictly more expressive than HFL^k. The case of $k = 0$ is reasonably simple since the expressive power of the modal μ-calculus, i.e. HFL^0 is quite well understood, including examples of properties that are known not to be expressible in it. The aforementioned tree property of being balanced is such an example [4]. For $k > 0$ this follows from considerations in computational complexity: model checking HFL^k is k-EXPTIME-complete [3] and this already holds for the data complexity. I.e. each HFL^k, $k \geq 1$, contains formulas which express some decision problem that is hard for deterministic k-fold exponential time. Expressive strictness of the type order hierarchy is then a direct consequence of the time hierarchy theorem [6] which particularly shows that k-EXPTIME $\subsetneq$ $(k + 1)$-EXPTIME.

Here we study the complexity of HFL model checking w.r.t. space usage. We identify a syntactical criterion on formulas – *tail-recursion* – which causes space-efficiency in a relative sense. It has been developed for PHFL, a polyadic extension of HFL, in the context of descriptive complexity. Extending Otto's result showing that a polyadic version of the modal μ-calculus [1] captures the bisimulation-invariant fragment of polynomial time [14], $\mathrm{PHFL}^0 \equiv \mathrm{P}/{\sim}$ in short, it was shown that $\mathrm{PHFL}^1 \equiv \mathrm{EXPTIME}/{\sim}$ [12], i.e. polyadic HFL formulas of function order at most 1 express exactly the bisimulation-invariant graph properties that can be evaluated in deterministic exponential time. Tail-recursion restricts the allowed combinations of fixpoint types (least or greatest), modality types (existential or universal), Boolean operators (disjunctions and conjunctions) and nestings of function applications. Its name is derived from the fact that a standard top-down evaluation algorithm working on states of a transition system and formulas can be implemented tail-recursively and, hence, intuitively in a rather space-efficient way. In the context of descriptive complexity, it was shown that the tail-recursive fragment of PHFL^1 captures polynomial space modulo bisimilarity, $\mathrm{PHFL}^1_{\mathsf{tail}} \equiv \mathrm{PSPACE}/{\sim}$ [12].

These results can be seen as an indication that tail-recursion is indeed a synonym for space-efficiency. In this paper we show that this is not restricted to order 1. We prove that the model checking problem for the tail-recursive fragment of HFL^{k+1} is k-EXPSPACE-complete. This already holds for the data complexity which yields a strict hierarchy of expressive power within $\mathrm{HFL}_{\mathsf{tail}}$, as a consequence of the space hierarchy theorem [16].

In Sect. 2 we recall HFL and apply the concept of tail-recursion, originally developed for a polyadic extension, to this monadic logic. In Sect. 3 we present upper bounds; matching lower bounds are presented in Sect. 4.

2 Higher-Order Fixpoint Logic

Labeled Transition Systems. Fix a set $\mathcal{P} = \{p, q, \dots\}$ of atomic propositions and a set $\mathcal{A} = \{a, b, \dots\}$ of actions. A labeled transition system (LTS) is a tuple

$\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a}\}_{a\in\mathcal{A}}, \ell)$, where $\mathcal{S}$ is a set of states, $\xrightarrow{a}$ is a binary relation for each $a \in \mathcal{A}$ and $\ell\colon \mathcal{S} \to \mathfrak{P}(\mathcal{P})$ is a function assigning, to each state, the set of propositions that are satisfied in it. We write $s \xrightarrow{a} t$ to denote that $(s,t) \in \xrightarrow{a}$.

Types. The semantics of HFL is defined via complete function lattices over a transition system. In order to guarantee monotonicity (and other well-formedness conditions), formulas representing functions need to be strongly typed according to a simple type system. It defines types inductively from a ground type via function forming: the set of HFL-types is given by the grammar

$$\tau \ ::= \ \bullet \mid \tau^v \to \tau$$

where $v \in \{+, -, 0\}$ is called a variance. It indicates whether a function uses its argument in a monotone, antitone or arbitrary way.

The order $\mathsf{ord}(\tau)$ of a type τ is defined inductively as $\mathsf{ord}(\bullet) = 0$, and $\mathsf{ord}(\sigma \to \tau) = \max(1 + \mathsf{ord}(\sigma), \mathsf{ord}(\tau))$.

The function type constructor $\to$ is right-associative. Thus, every type is of the form $\tau_1^{v_1} \to \dots \tau_m^{v_m} \to \bullet$.

Formulas. Let $\mathcal{P}$ and $\mathcal{A}$ be as above. Additionally, let $\mathcal{V}_\lambda = \{x, y, \dots\}$ and $\mathcal{V}_{\mathsf{fp}} = \{X, Y, \dots\}$ be two sets of variables. We only distinguish them in order to increase readability of formulas, referring to $\mathcal{V}_\lambda$ as *λ-variables* and $\mathcal{V}_{\mathsf{fp}}$ as *fixpoint variables*. The set of (possibly non-well-formed) HFL formulas is then given by the grammar

$$\begin{aligned}\varphi ::= p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg\varphi \mid \langle a\rangle\varphi \mid [a]\varphi \mid x \mid \lambda(x^v\colon \tau).\ \varphi \mid \varphi\,\varphi \\ \mid X \mid \mu(X\colon \tau).\ \varphi \mid \nu(X\colon \tau).\ \varphi\end{aligned}$$

where $p \in \mathcal{P}, a \in \mathcal{A}, x \in \mathcal{V}_\lambda, X \in \mathcal{V}_{\mathsf{fp}}, \tau$ is an HFL-type and v is a variance. Derived connectives such as $\Rightarrow, \Leftrightarrow, \top, \bot$ can be added in the usual way, but we consider $\wedge$, $[a]$ and ν to be built-in operators instead of derived connectives. The set of subformulas $\mathsf{sub}(\varphi)$ of a formula φ is defined in the usual way. Note that fixpoint variables need no decoration by a variance since they can only occur in a monotonic fashion.

The intuition for the operators not present in the modal μ-calculus is as follows: $\lambda(x\colon \tau).\ \varphi$ defines a function that consumes an argument x of type τ and returns what φ evaluates to, x returns the value of λ-variable x, and $\varphi\ \psi$ applies ψ as an argument to the function φ. If a formula consists of several consecutive λ abstractions, we compress the argument display in favor of readability. For example, $\lambda(x\colon \tau).\ \lambda(y\colon \sigma).\ \psi$ becomes $\lambda(x\colon \tau, y\colon \sigma).\ \psi$ or even $\lambda(x, y\colon \tau).\ \psi$ if $\tau = \sigma$.

A sequence of the form $X_1^{v_1} : \tau_1, \dots, X_n^{v_n} : \tau_n, x_1^{v_1'} : \tau_1', \dots, x_j^{v_j'} : \tau_j'$ where the X_i are fixpoint variables, the x_j are λ-variables, the τ_i, τ_j' are types and the v_i, v_j' are variances, is called a *context*. We assume that each fixpoint variable and each λ-variable occurs only once per context. The context $\overline{\Gamma}$ is obtained from Γ by

$$\frac{}{\Gamma \vdash p : \bullet} \qquad \frac{\Gamma \vdash \varphi : \bullet}{\Gamma \vdash \langle a \rangle \varphi : \bullet} \qquad \frac{\Gamma \vdash \varphi : \bullet}{\Gamma \vdash [a] \varphi : \bullet} \qquad \frac{\overline{\Gamma} \vdash \varphi : \bullet}{\Gamma \vdash \neg \varphi : \bullet}$$

$$\frac{\Gamma \vdash \varphi : \bullet \qquad \Gamma \vdash \psi : \bullet}{\Gamma \vdash \varphi \vee \psi : \bullet} \qquad \frac{\Gamma \vdash \varphi : \bullet \qquad \Gamma \vdash \psi : \bullet}{\Gamma \vdash \varphi \wedge \psi : \bullet} \qquad \frac{v \in \{+, 0\}}{\Gamma\ ,\ x^v : \tau \vdash x : \tau}$$

$$\frac{}{\Gamma\ ,\ X^+ : \tau \vdash X : \tau} \qquad \frac{\Gamma, x^v : \sigma \vdash \varphi : \tau}{\Gamma \vdash \lambda(x^v : \sigma).\ \varphi : \sigma^v \to \tau} \qquad \frac{\Gamma, X^+ : \tau \vdash \varphi : \tau}{\Gamma \vdash \mu(X : \tau).\ \varphi : \tau}$$

$$\frac{\Gamma, X^+ : \tau \vdash \varphi : \tau}{\Gamma \vdash \nu(X : \tau).\ \varphi : \tau} \qquad \frac{\Gamma \vdash \varphi : \sigma^+ \to \tau \qquad \Gamma \vdash \psi : \sigma}{\Gamma \vdash \varphi\ \psi : \tau}$$

$$\frac{\Gamma \vdash \varphi : \sigma^- \to \tau \qquad \overline{\Gamma} \vdash \psi : \sigma}{\Gamma \vdash \varphi\ \psi : \tau} \qquad \frac{\Gamma \vdash \varphi : \sigma^0 \to \tau \qquad \Gamma \vdash \psi : \sigma \qquad \overline{\Gamma} \vdash \psi : \sigma}{\Gamma \vdash \varphi\ \psi : \tau}$$

Fig. 1. The HFL typing system

replacing all typing hypotheses of the form $X^+ \colon \tau$ by $X^- \colon \tau$ and vice versa, and doing the same for λ-variables. An HFL-formula φ has type τ in context Γ if $\Gamma \vdash \varphi \colon \tau$ can be derived via the typing rules in Fig. 1. A formula φ is *well-formed* if $\Gamma \vdash \varphi \colon \tau$ can be derived for some Γ and τ. Note that, while fixpoint variables may only be used in a monotonic fashion, contexts with fixpoint variables of negative variance are still necessary to type formulas of the form $\mu(X \colon \bullet).\ \neg\neg X$. In some examples, we may sometimes omit type and/or variance anotations.

Moreover, we also assume that in a well-formed formula φ, each fixpoint variable $X \in \mathcal{V}_{\mathsf{fp}}$ is bound at most once, i.e., there is at most one subformula of the form $\mu(X \colon \tau).\ \psi$ or $\nu(X \colon \tau).\ \psi$. Then there is a function $\mathsf{fp} \colon \mathcal{V}_{\mathsf{fp}} \to \mathsf{sub}(\varphi)$ such that $\mathsf{fp}(X)$ is the unique subformula $\sigma(X \colon \tau).\ \varphi'$ with $\sigma \in \{\mu, \nu\}$. Note that it is possible to order the fixpoints in such a formula as $X_1, \dots, X_n$ such that $\mathsf{fp}(X_i) \notin \mathsf{sub}(\mathsf{fp}(X_j))$ for $j > i$.

The *order* of a formula φ is the maximal type order k of any type used in a proof of $\emptyset \vdash \varphi \colon \bullet$. With HFL^k we denote the set of all well-formed HFL formulas of ground type whose order is at most k. In particular, HFL^0 is the modal μ-calculus $\mathcal{L}_\mu$. The notion of order of a formula can straightforwardly be applied to formulas which are not of ground type $\bullet$. We will therefore also speak of the order of some arbitrary subformula of an HFL formula.

Semantics. Given an LTS $\mathcal{T}$, each HFL type τ induces a complete lattice $[\![\tau]\!]^{\mathcal{T}}$ starting with the usual powerset lattice of its state space, and then lifting this to lattices of functions of higher order. When the underlying LTS is clear from the context we only write $[\![\tau]\!]$ rather than $[\![\tau]\!]^{\mathcal{T}}$. We also identify a lattice with its underlying set and write $f \in [\![\tau]\!]$ for instance. These lattices are then inductively defined as follows:

– $[\![\bullet]\!]^{\mathcal{T}}$ is the lattice $\mathfrak{P}(\mathcal{S})$ ordered by the inclusion relation $\subseteq$,
– $[\![\sigma^v \to \tau]\!]^{\mathcal{T}}$ is the lattice whose domain is the set of all (if $v = 0$), resp. monotone (if $v = +$), resp. antitone (if $v = -$) functions of type $[\![\sigma]\!]^{\mathcal{T}} \to [\![\tau]\!]^{\mathcal{T}}$ ordered pointwise, i.e. $f \sqsubseteq_{\sigma^v \to \tau} g$ iff $f(x) \sqsubseteq_\tau g(x)$ for all $x \in [\![\sigma]\!]^{\mathcal{T}}$.

$$\begin{aligned}
[\![\Gamma \vdash p\colon \bullet]\!]_\eta &= \{s \in \mathcal{S} \mid P \in \ell(s)\}\\
[\![\Gamma \vdash \varphi \vee \psi\colon \bullet]\!]_\eta &= [\![\Gamma \vdash \varphi\colon \bullet]\!]_\eta \cup [\![\Gamma \vdash \psi\colon \bullet]\!]_\eta\\
[\![\Gamma \vdash \varphi \wedge \psi\colon \bullet]\!]_\eta &= [\![\Gamma \vdash \varphi\colon \bullet]\!]_\eta \cap [\![\Gamma \vdash \psi\colon \bullet]\!]_\eta\\
[\![\Gamma \vdash \langle a\rangle\varphi\colon \bullet]\!]_\eta &= \{s \in \mathcal{S} \mid \text{ ex. } t \in [\![\Gamma \vdash \varphi\colon \bullet]\!]_\eta \text{ s.t. } s \xrightarrow{a} t\}\\
[\![\Gamma \vdash [a]\varphi\colon \bullet]\!]_\eta &= \{s \in \mathcal{S} \mid \text{ f.a. } t \in \mathcal{S} \text{ with } s \xrightarrow{a} t \text{ holds } t \in [\![\Gamma \vdash \varphi\colon \bullet]\!]_\eta\}\\
[\![\Gamma \vdash x\colon \tau]\!]_\eta &= \eta(x)\\
[\![\Gamma \vdash X\colon \tau]\!]_\eta &= \eta(X)\\
[\![\Gamma \vdash \lambda(x^v\colon \sigma)\colon \sigma^v \to \tau]\!]_\eta &= f \in [\![\sigma^v \to \tau]\!] \text{ s.t. f.a. } y \in [\![\sigma]\!].\ f(y)\\
&\qquad\qquad\qquad = [\![\Gamma, x^v\colon \sigma \vdash \varphi\colon \tau]\!]_{\eta[x\mapsto y]}\\
[\![\Gamma \vdash \varphi\,\psi\colon \tau]\!]_\eta &= [\![\Gamma \vdash \varphi\colon \sigma^v \to \sigma]\!]_\eta\,([\![\Gamma \vdash \psi\colon \sigma]\!]_\eta)\\
[\![\Gamma \vdash \mu(X\colon \tau).\varphi\colon \tau]\!]_\eta &= \bigsqcap\{d \in [\![\tau]\!] \mid [\![\Gamma, X\colon \tau^+ \vdash \varphi\colon \tau]\!]_{\eta[X\mapsto d]} \sqsubseteq_\tau d\}\\
[\![\Gamma \vdash \nu(X\colon \tau).\varphi\colon \tau]\!]_\eta &= \bigsqcup\{d \in [\![\tau]\!] \mid d \sqsubseteq_\tau [\![\Gamma, X\colon \tau^+ \vdash \varphi\colon \tau]\!]_{\eta[X\mapsto d]}\}
\end{aligned}$$

Fig. 2. Semantics of HFL

Given a context Γ, an *environment* η that respects Γ is a partial map from $\mathcal{V}_\lambda \cup \mathcal{V}_{\mathsf{fp}}$ such that $\eta(x) \in [\![\tau]\!]$ if $\Gamma \vdash x\colon \tau$ and $\eta(X) \in [\![\tau']\!]$ if $\Gamma \vdash x\colon \tau'$. From now on, all environments respect the context in question. The update $\eta[X \mapsto f]$ is defined in the usual way as $\eta[X \mapsto f](x) = \eta(x)$, $\eta[X \mapsto f](Y) = \eta(Y)$ if $Y \neq X$ and $\eta(Y) = f$ if $X = Y$. Updates for λ-variables are defined analogously.

The semantics of an HFL formula is defined inductively as per Fig. 2. We write $\mathcal{T}, s \models_\eta \varphi\colon \tau$ if $s \in [\![\Gamma \vdash \varphi\colon \tau]\!]_\eta$ for suitable Γ and abbreviate the special case with a closed formula of ground type writing $\mathcal{T}, s \models \varphi$ instead of $\mathcal{T}, s \models_\emptyset \varphi\colon \bullet$.

The Tail-Recursive Fragment. In general, a tail-recursive function is one that is never called recursively in an intermediate step of the evaluation of its body, either for evaluating a condition on branching, or for evaluating an argument of a function call. Tail-recursive functions are known to be more space-efficient in general as they do not require a call stack for their evaluation.

The notion of tail-recursion has been transposed to the framework of higher-order fixpoint logics, originally for a polyadic extension of HFL [12]. The adaptation to HFL is straight-forward, presented in the following. Intuitively, tail-recursion restricts the syntax of the formulas such that fixpoint variables do not occur freely under the operators $\wedge$ and $[a]$, nor in an operand position.

Definition 1. *An* HFL *formula* φ *is* tail-recursive *if the statement* $\mathsf{tail}(\varphi, \emptyset)$ *can be derived via the rules in Fig. 3.* $\mathrm{HFL}^k_{\mathsf{tail}}$ *consists of all tail-recursive formulas in* HFL^k*.*

Note that these rules do not treat conjunctions symmetrically. For instance, $\mu X.p \vee (q \wedge \langle - \rangle X)$ – the straight-forward translation of the CTL reachability

$$\frac{}{\mathsf{tail}(p,\bar{X})} \qquad \frac{}{\mathsf{tail}(x,\bar{X})} \qquad \frac{X \in \bar{X}}{\mathsf{tail}(X,\bar{X})} \qquad \frac{\mathsf{tail}(\varphi,\emptyset)}{\mathsf{tail}(\neg\varphi,\bar{X})} \qquad \frac{\mathsf{tail}(\varphi,\bar{X}) \quad \mathsf{tail}(\psi,\bar{X})}{\mathsf{tail}(\varphi \vee \psi,\bar{X})}$$

$$\frac{\mathsf{tail}(\varphi,\emptyset) \quad \mathsf{tail}(\psi,\bar{X})}{\mathsf{tail}(\varphi \wedge \psi,\bar{X})} \qquad \frac{\mathsf{tail}(\varphi,\bar{X})}{\mathsf{tail}(\langle a\rangle\varphi,\bar{X})} \qquad \frac{\mathsf{tail}(\varphi,\emptyset)}{\mathsf{tail}([a]\varphi,\bar{X})} \qquad \frac{\mathsf{tail}(\varphi,\bar{X}) \quad \mathsf{tail}(\psi,\emptyset)}{\mathsf{tail}(\varphi\ \psi,\bar{X})}$$

$$\frac{\mathsf{tail}(\varphi,\bar{X})}{\mathsf{tail}(\lambda(x\colon \tau^v).\varphi,\bar{X})} \qquad \frac{\mathsf{tail}(\varphi,\bar{X}\cup\{Z\})}{\mathsf{tail}(\mu(Z\colon \tau).\varphi,\bar{X})} \qquad \frac{\mathsf{tail}(\varphi,\bar{X}\cup\{Z\})}{\mathsf{tail}(\nu(Z\colon \tau).\varphi,\bar{X})}$$

Fig. 3. Derivation rules for establishing tail-recursiveness. The set $\bar{X}$ denotes the set of allowed free fixpoint variables of the formula in question.

property $\mathtt{E}(q\,\mathtt{U}\,p)$ – is tail-recursive, but $\mu X.p \vee (\langle - \rangle X \wedge q)$ is *not* tail-recursive because the rule for $\wedge$ in Fig. 3 only allows recursive calls to fixpoint variables via the right conjunct of a conjunction. Of course, adding one more rule to Fig. 3, one could make $\mathrm{HFL}_{\mathsf{tail}}$ closed under commutations of $\wedge$ operands, the only important point is that all of the free recursive variables occur on at most one side of each $\wedge$.

Example 2. The HFL^1 formula $\big(\nu F.\lambda x.\lambda y.(x \Rightarrow y) \wedge (F\ \langle a\rangle x\ \langle b\rangle y)\big)\ \top\ \langle b\rangle\top$ has been introduced for expressing a form of assume-guarantee property [18]. This formula is tail-recursive, as one can easily check.

The property of being a balanced tree can also be formalised by a tail-recursive HFL^1 formula: $\big(\mu F.\lambda x.[-]\bot \vee (F\ [-]x)\big)\ \bot$.

In the next section, we will see that these properties and any other expressible in $\mathrm{HFL}^1_{\mathsf{tail}}$ can be checked in polynomial space, thus improving a known exponential time upper bound [3, 2].

Example 3. Consider reachability properties of the form "there is a maximal path labelled with a word from L" where $L \subseteq \Sigma^*$ is some formal language. For context-free languages the logic formalising such properties is Propositional Dynamic Logic of Context-Free Programs [5]. It can be model checked in polynomial time [10]. However, formal-language constrained reachability is not restricted to context-free languages only. Consider the reachability problem above for $L = \{a^n b^n c^n \mid n \geq 1\}$. It can be formalised by the HFL^2 formula

$$\big(\mu F.\lambda f.\lambda g.\lambda h.\lambda x.f(g(h(x))) \vee (F\ (\lambda x.f\ \langle a\rangle x)\ (\lambda x.g\ \langle b\rangle x)\ (\lambda x.h\ \langle c\rangle x)\big)$$
$$id\ id\ id\ [-]\bot$$

with type $x : \bullet$; $f, g, h : \tau_1 := \bullet^+ \to \bullet$ and $F : \tau_1^+ \to \tau_1^+ \to \tau_1^+ \to \bullet^+ \to \bullet$. Again, one can check that it is tail-recursive. Since it is of order 2, Thm. 5 yields that the corresponding reachability problem can be checked using exponential space.

3 Upper Bounds in the Exponential Space Hierarchy

Consider an HFL fixpoint formula of the form $\psi = \sigma(X\colon \tau).\varphi$ and its finite approximants defined via

$$X^0 := \begin{cases} \bot & \text{, if } \sigma = \mu, \\ \top & \text{, otherwise} \end{cases} \qquad \text{and} \qquad X^{i+1} := \varphi[X^i/X] \ .$$

where $\varphi[X^i/X]$ denotes the simultaneous replacement of every free occurrence of X by X^i in φ.

It is known that over a finite LTS $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a}\}, \ell)$, ψ is equivalent to X^m, where m is the height $\mathsf{ht}(\tau)$ of the lattice of τ. Generally, $\mathsf{ht}(\tau)$ is k-fold exponential in the size of $|\mathcal{S}|$ for $k = \mathit{ord}(\tau)$ [3]. Note that a k-fold exponentially large number can be represented by $(k-1)$-fold exponentially many bits.

For an $\mathrm{HFL}^k_{\mathsf{tail}}$ formula φ, we define its *recursion depth* $\mathsf{rd}(\varphi)$:

$$\begin{aligned} \mathsf{rd}(p) = \mathsf{rd}(X) &:= 0 \\ \mathsf{rd}(\varphi_1 \vee \varphi_2) &:= \max(\mathsf{rd}(\varphi_1), \mathsf{rd}(\varphi_2)) \\ \mathsf{rd}(\varphi_1 \wedge \varphi_2) &:= \max(\mathsf{rd}(\varphi_2), 1 + \mathsf{rd}(\varphi_1)) \\ \mathsf{rd}(\varphi_1\ \varphi_2) &:= \max(\mathsf{rd}(\varphi_1, 1 + \mathsf{rd}(\varphi_2)) \\ \mathsf{rd}(\langle a\rangle\varphi) = \mathsf{rd}(\lambda X.\varphi) = \mathsf{rd}(\mu X.\varphi) = \mathsf{rd}(\nu X.\varphi) &:= \mathsf{rd}(\varphi) \\ \mathsf{rd}(\neg\varphi) = \mathsf{rd}([a]\varphi) &:= 1 + \mathsf{rd}(\varphi) \end{aligned}$$

The recursion depth of a formula measures the number of times that a top-down nondeterministic local model-checking procedure has to maintain calling contexts. For example, when verifying whether a state is a model of a disjunction, it is sufficient to nondeterministically guess a disjunct and continue with it; the other disjunct is irrelevant. For a conjunction, the procedure also descends into one of the conjuncts first, but has to remember, e.g., the environment at the conjunction itself in case the procedure has to backtrack. Note that the recursion depth of a formula is linear in its size.

We combine the bounded number of calling contexts and the above unfolding property into a model-checking algorithm that avoids the enumeration of full function tables for fixpoint definitions of the highest order by only evaluating it at arguments actually occurring in the formula. Unfolding a fixpoint expression is results in the evaluation of the same fixpoint at different arguments, and the unfolding property allows to give an upper bound on the number of unfoldings needed. Tail-recursiveness ensures that this procedure proceeds in a mostly linear fashion, since the number of calling contexts that need to maintained at any given moment during the evaluation is bounded by the recursion depth of the formula in question.

For the remainder we fix a formula $\psi \in \mathrm{HFL}^k_{\mathsf{tail}}$ and an LTS $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a}\}, \ell)$. We present two mutually recursive functions **check** and **buildFT**. The function **check**$(s, \varphi, (f_1, \dots, f_n), \eta, \mathsf{cnt})$ consumes a state $s \in \mathcal{S}$, a subformula φ of ψ, a list of function tables, an environment η and a partial function cnt from $\mathcal{V}_{\mathsf{fp}}$

to $\mathbb{N}$ and checks whether $s \models_\eta (\cdots(\varphi\, f_n)\cdots f_1)$ if all free fixpoint variables X of φ are replaced by $X^{\mathsf{cnt}(X)}$ in a suitable order. The function $\mathbf{buildFT}(\varphi,\eta)$ consumes a subformula φ of ψ and an environment η and builds the complete function table of φ with respect to η, i.e., computes $[\![\varphi]\!]_\eta$.

The definition of $\mathbf{check}(s,\varphi,(f_1,\ldots,f_n),\eta,\mathsf{cnt})$ depends on the form of φ:

– If φ is an atomic formula, return true if $s \models \varphi$ and false otherwise.
– If $\varphi = \varphi_1 \vee \varphi_2$, guess $i \in \{1,2\}$ and return $\mathbf{check}(s,\varphi_i,(f_1,\ldots,f_n),\eta,\mathsf{cnt})$.
– If $\varphi = \varphi_1 \wedge \varphi_2$, note that $\mathsf{rd}(\varphi_1) < \mathsf{rd}(\varphi)$ and that φ_1 has no free fixpoint variables. Return false if $\mathbf{check}(s,\varphi_1,(f_1,\ldots,f_n),\eta,\emptyset)$ returns false. Otherwise, return $\mathbf{check}(s,\varphi_2,(f_1,\ldots,f_n),\eta,\mathsf{cnt})$.
– If $\varphi = \langle a\rangle\varphi'$, guess t with $s \xrightarrow{a} t$ and return $\mathbf{check}(t,\varphi',(f_1,\ldots,f_n),\eta,\mathsf{cnt})$.
– If $\varphi = [a]\varphi'$, note that $\mathsf{rd}(\varphi') < \mathsf{rd}(\varphi)$ and that φ' has no free fixpoint variables. Iterate over all t with $s \xrightarrow{a} t$. If $\mathbf{check}(t,\varphi',(f_1,\ldots,f_n),\eta,\emptyset)$ returns false for at least one such t, return false. Otherwise, return true.
– If $\varphi = \neg\varphi'$, note that $\mathsf{rd}(\varphi') < \mathsf{rd}(\varphi)$ and that φ' has no free fixpoint variables. If $\mathbf{check}(s,\varphi',(f_1,\ldots,f_n),\eta,\emptyset)$ returns true, return false and vice versa.
– If $\varphi = \varphi'\,\varphi''$, note that $\mathsf{rd}(\varphi'') < \mathsf{rd}(\varphi)$ and that φ'' has no free fixpoint variables. Compute $f_{n+1} = \mathbf{buildFT}(\varphi'',\eta)$ and return $\mathbf{check}(s,\varphi',(f_1,\ldots,f_n, f_{n+1}),\eta,\mathsf{cnt})$.
– If $\varphi = x$, return true if $s \in (\cdots(\eta(x)\, f_n)\cdots f_1)$, return false otherwise.
– If $\varphi = \lambda x.\varphi'$, return $\mathbf{check}(s,\varphi',(f_1,\ldots,f_{n-1}),\eta[x \mapsto f_n],\mathsf{cnt})$.
– If $\varphi = \mu(X\colon\tau).\varphi'$, return $\mathbf{check}(s,\varphi',(f_1,\ldots,f_n),\eta,\mathsf{cnt}[X \mapsto \mathsf{ht}(\tau)])$.
– If $\varphi = \nu(X\colon\tau).\varphi'$, return $\mathbf{check}(s,\varphi',(f_1,\ldots,f_n),\eta,\mathsf{cnt}[X \mapsto \mathsf{ht}(\tau)])$.
– If $\varphi = X$, return false if $\mathsf{cnt}(X) = 0$ and X is a least fixpoint variable, return true if $\mathsf{cnt}(X) = 0$ and X is a greatest fixpoint variable, otherwise, return $\mathbf{check}(s,\mathsf{fp}(X),(f_1,\ldots,f_n),\eta,\mathsf{cnt}')$, where $\mathsf{cnt}'(Y) = \mathsf{cnt}(Y)$ if $Y \neq X$ and $\mathsf{cnt}'(X) = \mathsf{cnt}(X) - 1$.

The definition of $\mathbf{buildFT}(\varphi,\eta)$ is rather simple: If $\varphi\colon \tau_n \to \cdots \to \tau_1 \to \bullet$, iterate over all $s \in \mathcal{S}$ and all $(f_n,\ldots,f_1) \in [\![\tau_n]\!] \times \cdots \times [\![\tau_1]\!]$ and call $\mathbf{check}(s,\varphi,(f_1,\ldots,f_n),\eta,\emptyset)$ for each combination. This will yield the function table $[\![\varphi]\!]_\eta$ via $[\![\varphi]\!]_\eta = \{f \in [\![\tau_n \to \cdots \to \tau_1 \to \bullet]\!] \mid (\cdots(f\, f_n)\cdots f_1) = \{s \in \mathcal{S} \mid \mathbf{check}(s,\varphi,(f_1,\ldots,f_n),\eta,\emptyset) = \mathsf{true}\}\}$.

Theorem 4. *Let* $\psi \in \mathrm{HFL}_{\mathsf{tail}}$. *Then* $\mathbf{check}(s,\psi,\epsilon,\emptyset,\emptyset)$ *returns* true *iff* $\mathcal{T}, s \models \psi$.

Proof (Sketch). Fix an order of the fixpoint variables of ψ as $X_1,\ldots,X_m$ such that $\mathsf{fp}(X_i) \notin \mathsf{sub}(\mathsf{fp}(X_j))$ if $j > i$. Note that this also orders the possible values of cnt by ordering them lexicographically and assuming that undefined values are larger than $\mathsf{ht}(\tau)$ for any τ appearing in ψ.

Consider a subformula φ of ψ. Given a partial map cnt made total as in the previous paragraph, we write φ^{cnt} to denote $\varphi[X_1^{\mathsf{cnt}(X_1)}/X_1,\ldots,X_n^{\mathsf{cnt}(X_n)}/X_n]$, i.e, the result of simultaneously replacing free fixpoint variables of φ by their approximants as per cnt such that none of them occur free anymore.

In fact, $\mathbf{check}(s,\varphi,(f_1,\ldots,f_n),\eta,\mathsf{cnt})$ returns true iff $s \in [\![\varphi^{\mathsf{cnt}}]\!]_\eta\, f_n \cdots f_1$ assuming that $\mathbf{buildFT}(\varphi,\eta)$ computes $[\![\varphi]\!]_\eta$. It is easy to see that the statement in the theorem follows from this. The proof itself is a routine induction over the syntax of ψ to show that the above invariant is maintained. In each step, the procedure either passes to a proper subformula and maintains the value of cnt and recursion depth, or, in case of fixpoint unfoldings, properly decreases cnt but keeps recursion depth, or, in case of calls that are not tail-recursive, passes to a formula with properly reduced recursion depth. Moreover, in the case of function application, the call to **buildFT** will result in calls to **check** with properly reduced recursion depth, and **buildFT** just computes a tabular representation of the HFL semantics. Hence, the procedure eventually halts and works correctly. □

Theorem 5. *The model checking problem for* $\mathrm{HFL}^{k+1}_{\mathsf{tail}}$ *is in* k-EXPSPACE.

Proof. By Savitch's Theorem [15] and Thm. 4, it suffices to show that the nondeterministic procedure **check** can be implemented to use at most k-fold exponential space for formulas in $\mathrm{HFL}^{k+1}_{\mathsf{tail}}$.

The information required to evaluate $\textbf{check}(s, \varphi, (f_1, \dots, f_n), \eta, \mathsf{cnt})$ takes k-fold exponential space: references to a state and a subformula take linear space, each of the function tables $f_1, \dots, f_n$ appears in operand position and, hence, is a function of order at most k, which takes k-fold exponential space. An environment is just a partial map from $\mathcal{V}_\lambda$ to more function tables, also of order at most k. Finally, cnt stores $|\mathcal{V}_{\mathsf{fp}}|$ many numbers whose values are bounded by an $(k+1)$-fold exponential. Hence, they can be represented as k-fold exponentially long bit strings.

During evaluation, **check** operates in a tail-recursive fashion for most operators, which means that no stack has to be maintained and the space needed is restricted to what is described in the previous paragraph. A calling context (which is just an instance of **check** as described above, with an added logarithmically sized counter in case of $[a]\varphi$) has to be preserved only at the steps where the recursion depth decreases. In the case of negation, it is not necessary to maintain the complete calling context. Instead, the nondeterministic procedure for the negated subformula is called and the return value is inverted. By Savitch's Theorem, the procedure can actually be implemented to run deterministically with the same space requirements, and, hence, is safe to call in a nondeterministic procedure.

Since the recursion depth of an $\mathrm{HFL}^{k}_{\mathsf{tail}}$-formula is linear in the size of the formula, only linearly many such calling contexts have to be stored at any given point during the evaluation, which does not exceed nondeterministic k-fold exponential space. Moreover, Savitch's Theorem has to be applied only linearly often on any computation path. □

Note that occurrences of negation do not lead to proper backtracking to a calling context, but rather mark an invocation of Savitch's Theorem. Hence, the definition of recursion depth could be changed to not increase at negation. We chose to include applications of Savitch's Theorem into the definition of recursion depth for reasons of clarity.

4 Matching Lower Bounds

A typical k-EXPSPACE-complete problem (for $k \geq 0$) is the order-k corridor tiling problem [17]: A tiling system is of the form $\mathcal{K} = (T, H, V, t_I, t_\Box, t_F)$ where T is a finite set of tile types, $H, V \subseteq T \times T$ are the so-called horizontal and vertical matching relations, and $t_I, t_\Box, t_F \in T$ are three designated tiles called initial, blank and final.

Let $2^n_0 = n$ and $2^n_{k+1} = 2^{2^n_k}$. The *order-k corridor tiling problem* is the following: given a tiling system $\mathcal{K}$ as above and a natural number n encoded unarily, decide whether or not there is some m and a sequence $\rho_0, \dots, \rho_{m-1}$ of words over the alphabet T, with $|\rho_i| = 2^n_k$ for all $i \in \{0, \dots, m-1\}$, and such that the following four conditions hold. We write $\rho(j)$ for the j-th letter of the word ρ, beginning with $j = 0$.

- $\rho_0 = t_I t_\Box \dots t_\Box$
- For each $i = 0, \dots, m-1$ and $j = 0, \dots, 2^n_k - 2$ we have $(\rho_i(j), \rho_i(j+1)) \in H$.
- For each $i = 0, \dots, m-2$ and $j = 0, \dots, 2^n_k - 1$ we have $(\rho_i(j), \rho_{i+1}(j)) \in V$.
- $\rho_{m-1}(0) = t_F$

Such a sequence of words is also called a *solution* to the order-k corridor tiling problem on input $\mathcal{K}$ and n. The i-th word in this sequence is also called the *i-th row.*

Proposition 6 ([17]). *For each $k \geq 0$, the order-k corridor tiling problem is k-EXPSPACE-hard.*

In the following we construct a polynomial reduction from the order-k corridor tiling problem to the model checking problem for $\text{HFL}^{k+1}_{\mathsf{tail}}$. Fix a tiling system $\mathcal{K} = (T, H, V, t_I, t_F)$ and an $n \geq 1$. W.l.o.g. we assume $|T| \leq n$, and we fix an enumeration $T = \{t_0, \dots, t_{|T|-1}\}$ of the tiles such that $t_0 = t_I$, $t_{|T|-2} = t_\Box$, and $t_{|T|-1} = t_F$.

We define the transition system $\mathcal{T}_{\mathcal{K},n} = (\mathcal{S}, \{\xrightarrow{a}\}_{a \in \mathcal{A}}, \ell)$ as follows:

- $\mathcal{S} = \{0, \dots, n-1\}$,
- $\mathcal{A} = \{\mathsf{h}, \mathsf{v}, \mathsf{e}, \mathsf{u}, \mathsf{d}\}$ with $\xrightarrow{\mathsf{h}} = \{(i,j) \mid (t_i, t_j) \in H\}$ (for "horizontal"), $\xrightarrow{\mathsf{v}} = \{(i,j) \mid (t_i, t_j) \in V\}$ (for "vertical"), $\xrightarrow{\mathsf{e}} = \{0, \dots, n-1\} \times \{0, \dots, n-1\}$ (for "everywhere"), $\xrightarrow{\mathsf{u}} = \{(i,j) \mid 0 \leq i < j \leq n-1\}$ (for "up") and $\xrightarrow{\mathsf{d}} = \{(i,j) \mid 0 \leq j < i \leq n-1\}$ (for "down").
- $\ell(0) = \{p_I\}$, $\ell(|T|-2) = \{p_\Box\}$, and $\ell(|T|-1) = p_F$

The states of this transition system appear in two roles. On one hand, they encode the different tiles of the tiling problem $\mathcal{K}$, with the generic tiles $t_I, t_\Box, t_F$ identified by propositional labeling, while the rest remain anonymous. The horizontal and vertical matching relations are encoded by the accessibility relations h and v, respectively. On the other hand, the states double as the digits of the representation of large numbers. The relation u connects a digit to all digits of higher significance, d connects to all digits of lower significance, and e is the global accessibility relation.

Next we construct, for all $k \geq 1$, an $\mathrm{HFL}^{k+1}_{\mathsf{tail}}$ formula φ_k such that $\mathcal{T}_{\mathcal{K},n} \models \varphi_k$ holds iff $(\mathcal{K}, n)$ admits a solution to the order-k corridor tiling problem. We encode the rows of a tiling as functions of order k. Column numbers in $\{0, \ldots, 2^n_k - 1\}$ are encoded as functions of order $k-1$, following an approach similar to Jones [8].

Let $\tau_0 = \bullet$ and $\tau_{k+1} = \tau_k \to \bullet$ for all $k \geq 0$. For all $k \geq 0$ and $i \in \{0, \ldots, 2^n_{k+1} - 1\}$, let $\mathsf{jones}_k(i)$ be the function in the space $[\![\tau_k]\!]^{\mathcal{T}_{\mathcal{K},n}}$ defined as follows:

– $\mathsf{jones}_0(i)$ is the set of bits equal to 1 in the binary representation of i, i.e. $\mathsf{jones}_0(i) = S \subseteq \{0, \ldots, n-1\}$ where S is such that $i = \sum_{j \in S} 2^j$
– $\mathsf{jones}_{k+1}(i)$ maps $\mathsf{jones}_k(j)$ (for all $j \in \{0, \ldots, 2^n_{k+1} - 1\}$) to $\{0, \ldots, n-1\}$ if the j-th bit of i is 1, otherwise $\mathsf{jones}_{k+1}(i)$ maps $\mathsf{jones}_k(j)$ to $\emptyset$.

Consider the following formulas.

$$\begin{aligned}
\mathsf{ite} &= \lambda(b\colon \bullet), (x\colon \bullet), (y\colon \bullet).\ (b \wedge x) \vee (\neg b \wedge y)\\
\mathsf{zero}_0 &= \bot\\
\mathsf{zero}_{k+1} &= \lambda(m\colon \tau_k).\ \bot\\
\mathsf{gt}_0 &= \lambda(m_1, m_2\colon \tau_0).\ \langle \mathsf{e}\rangle\big(m_2 \wedge \neg m_1 \wedge [\mathsf{u}](m_1 \Rightarrow m_2)\big)\\
\mathsf{gt}_{k+1} &= \lambda(m_1, m_2\colon \tau_{k+1}).\ \mathsf{exists}_k\Big(\lambda(i\colon \tau_k).\ (m_2\ i) \wedge \neg(m_1\ i) \wedge\\
&\qquad\qquad \mathsf{forall}_k\big(\lambda(j\colon \tau_k).\ (\mathsf{gt_k}\ i\ j) \Rightarrow (m_1\ j) \Rightarrow (m_2\ j)\big)\Big)\\
\mathsf{next}_0 &= \lambda(m\colon \bullet).\ \mathsf{ite}\ m\ (\langle \mathsf{d}\rangle \neg m)\ ([\mathsf{d}]m)\\
\mathsf{next}_{k+1} &= \lambda(m\colon \tau_{k+1}, i\colon \tau_k).\ \mathsf{ite}\ (m\ i)\\
&\qquad\qquad \Big(\mathsf{exists}_k\big(\lambda(j_1\colon \tau_k).\ (\mathsf{gt}_k\ i\ j_1) \wedge \neg(m\ j_1)\big)\Big)\\
&\qquad\qquad \Big(\mathsf{forall}_k\big(\lambda(j_2\colon \tau_k).\ (\mathsf{gt}_k\ i\ j_2) \Rightarrow (m\ j_2)\big)\Big)\\
\mathsf{exists}_k &= \lambda(p\colon \tau_{k+1}).\ \Big(\big(\mu(F\colon \tau_k \to \bullet)\ .\ \lambda(m\colon \tau_k)\ .\ ([\mathsf{e}](p\ m)) \vee\\
&\qquad\qquad F\ (\mathsf{next}_k\ m)\big)\ \mathsf{zero}_k\Big)\\
\mathsf{forall}_k &= \lambda(p\colon \tau_{k+1}).\ \neg\mathsf{exists}_k\ (\neg p)
\end{aligned}$$

Let $\top_{\mathcal{S}} = [\![\top]\!]^{\mathcal{T}_{\mathcal{K},n}} = \{0, \ldots, n-1\}$ and $\bot_{\mathcal{S}} = [\![\bot]\!]^{\mathcal{T}_{\mathcal{K},n}} = \emptyset$. The functions above encode the if-then-else-operator, respectively arithmetic functions on Jones encodings of large natural numbers. The function gt_k allows to compare two integers : for all $m_1, m_2 \in \{0, \ldots, 2^n_{k+1} - 1\}$, $m_1 < m_2$ iff $\mathsf{gt}_k\ \mathsf{jones}_{m_1}(k)\ \mathsf{jones}_{m_2}(k)$ evaluates to $\top_{\mathcal{S}}$. Level 0 Jones encodings of numbers m_1 and m_2 are in relation gt_0 if, there is a bit that is set in $\mathsf{jones}_0(m_2)$ but not in $\mathsf{jones}_0(m_1)$, and all more significant bits that are set in $\mathsf{jones}_0(m_1)$ are also set in $\mathsf{jones}_0(m_2)$. The function gt_{k+1} operates on the same principle, except that bit positions are now level k Jones encodings of numbers, and the bit at position j is set in $\mathsf{jones}_{k+1}(m_i)$ iff $(m_i\ j)$ returns $\top_{\mathcal{S}}$. Moreover, quantification over all bit positions uses the functions forall_k and exists_k instead of the relation e.

The function next_k returns the level k Jones encoding of the number encoded by its input, incremented by one: If a bit is set in the encoding of the input, it stays set if and only if there is a bit of lesser significance that is not set. If it was not set in the input, it is set if and only if all lower bits were set in the input. For example, if m is the set $\{0,1,3\}$ that encodes the number 11, then next_0 returns the set $\{2,3\}$ which encodes 12. Encoding of bits and quantification over them works as in the case of gt_k.

Finally, the function exists_k checks for the existence of (the level k Jones encoding of) a number such that parameter p returns $\top_{\mathcal{S}}$ with this number as an argument. This is achieved by iterating over all level k Jones encodings of numbers between 0 and $2^n_{k+1}-1$. Consequently, exists_k expects an argument p of type τ_{k+1}, i.e., a function consuming an argument of type τ_k.

Lemma 7. *The following hold:*

1. *Assume $\eta(b) \in \{\top_{\mathcal{S}}, \bot_{\mathcal{S}}\}$. If $\eta(b) = \top_{\mathcal{S}}$, then $[\![\mathsf{ite}\ b\ x\ y]\!]_\eta$ is $\eta(x)$, else it is $\eta(y)$.*
2. *$[\![\mathsf{zero}_k]\!] = \mathsf{jones}_k(0)$ for all $k \geq 0$.*
3. *If $[\![\mathsf{next}_k\ m]\!]_\eta = \mathsf{jones}_k(i)$ and $\eta(m) = \mathsf{jones}_k(j)$, then $i = j+1$ modulo 2^m_{k+1}.*
4. *$[\![\mathsf{exists}_k\ p]\!]_\eta = \top_{\mathcal{S}}$ if there exists $\mathcal{X} \in [\![\tau_k]\!]^{\mathcal{T}_{\mathcal{K},n}}$ such that $[\![p\ x]\!]_{\eta[x\mapsto\mathcal{X}]} = \top_{\mathcal{S}}$, otherwise $[\![\mathsf{exists}_k\ p]\!]_\eta = \bot_{\mathcal{S}}$*

We are now ready to define the encoding of rows of width 2^n_k as functions in the space $[\![\tau_k]\!]^{\mathcal{T}_{\mathcal{K},n}}$. Let $\rho = \rho_0 \dots \rho_{2^n_k} \in T^*$ be a row of width 2^n_k for some $k \geq 1$. The coding $\mathsf{row}_k(\rho)$ of ρ is the function that maps $\mathsf{jones}_{k-1}(i)$ to $\{j\}$ where j is the number of i-th tile of the row, *i.e.* $\rho_i = t_j$. For example, the initial row of a tiling problem has the form $t_I\, t_\Box \cdots t_\Box$, i.e., the initial tile followed by $2^n_k - 1$ instances of $t_\Box$. The function encoding it would return the set $\{0\}$ of tiles labeled by t_I at argument $\mathsf{jones}_{k-1}(0)$ and return the set $\{|T|-2\}$ of tiles labeled by $t_\Box$ at arguments $\mathsf{jones}_{k-1}(1), \dots, \mathsf{jones}_{k-1}(2^n_k - 1)$.

Consider then the following formulas.

$$
\begin{aligned}
\mathsf{isTile} &= \lambda(x\colon \bullet),.\ [\mathsf{e}]\Big(x \Rightarrow \big(([\mathsf{u}]\neg x) \wedge ([\mathsf{d}]\neg x) \wedge (p_F \vee \langle \mathsf{u}\rangle p_F)\big)\Big)\\
\mathsf{isRow}_k &= \lambda(r\colon \tau_k).\ \mathsf{forall}_{k-1}\big(\lambda(m\colon \tau_{k-1}).\ \mathsf{isTile}\ (r\ m)\big)\\
\mathsf{isZero}_0 &= \lambda(m\colon \tau_0).\ [\mathsf{e}]\neg m\\
\mathsf{isZero}_{k+1} &= \lambda(m\colon \tau_{k+1}).\ \mathsf{forall}_k\ \big(\lambda(o\colon \tau_k).\ \mathsf{isZero}_0(m\ o)\big)\\
\mathsf{init}_k &= \lambda(m\colon \tau_{k-1}).\ \mathsf{ite}\ (\mathsf{isZero}_k\ m)\ p_I\ p_\Box\\
\mathsf{isFinal}_k &= \lambda(r\colon \tau_k).\ [\mathsf{e}]\big((r\ \mathsf{zero}_{k-1}) \Rightarrow p_F\big)\\
\mathsf{horiz}_k &= \lambda(r\colon \tau_k).\ \mathsf{forall}_{k-1}\ \Big(\lambda(m\colon \tau_{k-1}).\\
&\qquad [\mathsf{e}]\big((r\ m) \Rightarrow \big((\mathsf{isZero}_{k-1}\ (\mathsf{next}_{k-1}\ m)) \vee \langle \mathsf{h}\rangle(r\ (\mathsf{next}_{k-1}\ m))\big)\big)\Big)\\
\mathsf{vert}_k &= \lambda(r_1, r_2\colon \tau_k).\ \mathsf{forall}_{k-1}\ \Big(\lambda(m\colon \tau_{k-1}).\ [\mathsf{e}]\big((r_1\ m) \Rightarrow \langle \mathsf{v}\rangle(r_2\ m)\big)\Big)
\end{aligned}
$$

The function isTile checks whether its argument uniquely identifies a tile by verifying that it is a singleton set, and that it is not a state of index greater than $|T|-1$. The function isRow checks whether its argument r is a proper encoding of a row by verifying that $r\,m$ returns the encoding of a tile for each $m \in \{\mathsf{jones}_{k-1}(0), \dots, \mathsf{jones}_{k-1}(2^n_k - 1)\}$. The function init_k returns the initial row encoded as described in the previous paragraph, while $\mathsf{isFinal}_k$ verifies that its argument is a final row, i.e., a row where the tile in position 0 is t_F. Moreover, the function horiz_k verifies that the row r satisfies the horizontal matching condition. This is achieved by checking that, for each $m \in \{\mathsf{jones}_{k-1}(0), \dots, \mathsf{jones}_{k-1}(2^n_k - 1)\}$, either m is $\mathsf{jones}_{k-1}(2^n_k)$ (whence the value $\mathsf{isZero}_{k-1}(\mathsf{next}_{k-1}\ m)$ is $\top_{\mathcal{S}}$) or that there is a h-transition from the singleton set $(r\ m)$ into the singleton set $r\ (\mathsf{next}_{k-1}\ m)$. Finally, vert_k verifies that two rows satisfy the vertical matching condition in a similar way.

Lemma 8. *The following hold:*

1. $[\![\mathsf{isTile}\ x]\!]_\eta$ *evaluates to* $\top_{\mathcal{S}}$ *if* $\eta(x) = \{i\}$ *for some* $i \in \{0, \dots, |T| - 1\}$, *otherwise it evaluates to* $\bot_{\mathcal{S}}$.
2. $[\![\mathsf{isRow}_k\ x]\!]_\eta$ *evaluates to* $\top_{\mathcal{S}}$ *iff* $\eta(x) = \mathsf{row}_k(\rho)$ *for some row* ρ *of width* 2^n_k, *otherwise it evaluates to* $\bot_{\mathcal{S}}$.
3. $[\![\mathsf{init}_k]\!]$ *evaluates to* $\mathsf{row}_k(t_I \cdot t_\Box \cdots t_\Box)$.
4. *Assume* $\eta(r) = \mathsf{row}_k(\rho)$ *and* $\eta(r') = \mathsf{row}_k(\rho')$ *for some rows* $\rho = \rho_0 \dots \rho_{2^n_k}$ *and* $\rho' = \rho'_0 \dots \rho'_{2^n_k}$. *Then*
   (a) $[\![\mathsf{isFinal}_k\ r]\!]_\eta$ *evaluates to* $\top_{\mathcal{S}}$ *if* $\rho_0 = t_F$, *otherwise it evaluates to* $\bot_{\mathcal{S}}$.
   (b) $[\![\mathsf{horiz}_k\ r]\!]_\eta$ *evaluates to* $\top_{\mathcal{S}}$ *if* $(\rho_i, \rho_{i+1}) \in H$ *for all* $i \in \{0, \dots, 2^n_k - 1\}$, *otherwise it evaluates to* $\bot_{\mathcal{S}}$.
   (c) $[\![\mathsf{vert}_k\ r\ r']\!]_\eta$ *evaluates to* $\top_{\mathcal{S}}$ *if* $(\rho_i, \rho'_i) \in V$ *for all* $i \in \{0, \dots, 2^n_k - 1\}$, *otherwise it evaluates to* $\bot_{\mathcal{S}}$.

We now have introduced all the pieces we need for defining φ_k. Intuitively, φ_k should check for the existence of a solution to the order-k corridor tiling problem by performing an iteration that starts with a representation of the initial row in a solution and then guesses the next rows, each time checking that they match the previous one vertically. The iteration stops when a row is found that begins with the final tile. Let $\varphi_k =$

$$\big(\mu(P\colon \tau_{k+1}).\, \lambda(r_1\colon \tau_k).(\mathsf{isFinal}_k\ r_1) \lor (\mathsf{exists\_succ}_k\ r_1\ P)\big)\ \mathsf{init}_k$$

where $\mathsf{exists\_succ}_k =$

$$\lambda(r_1\colon \tau_k, p\colon \tau_{k+1}).\ \mathsf{exists}_k\ \big(\lambda(r_2\colon \tau_k).\ (\mathsf{horiz}_k\ r_2) \land (\mathsf{vert}_k\ r_1\ r_2) \land (p\ r_2)\big).$$

Here, $\mathsf{exists\_succ}$ consumes a row r_1 of type τ_k, and a function p of type τ_{k+1}. It guesses a row r_2 using exists_k, verifies that it matches r_1 vertically from above, and then applies p to r_2. Of course, p in this settting is the fixpoint P which generates new rows using $\mathsf{exists\_succ}$ until one of them is a final row, or ad infinitum, if the tiling problem is unsolvable.

Theorem 9. *Let* $k \geq 0$. *The model-checking problem of* $\mathrm{HFL}^{k+1}_{\mathsf{tail}}$ *is* k-EXPSPACE-*hard in data complexity.*

Proof. For $k = 0$ this is already known: there is a simple and fixed HFL^1 formula φ_0 that expresses the universality problem for NFA [2], a problem known to be

PSPACE-hard, i.e. 0-EXPSPACE-hard. It is easy to check that this φ_0 is in fact tail-recursive.

Let $k \geq 1$. The problem of deciding whether $\mathcal{T}_{n,\mathcal{K}} \models \varphi_k$ is equivalent to the problem of deciding whether $(\mathcal{K}, n)$ has a solution to the order-k corridor tiling problem. Therefore, we only need to give a formula ψ_k that is tail-recursive and equivalent to φ_k. Note indeed that φ_k is *not* tail recursive, because the recursive variable P of type τ_{k+1} appears as an argument of $\mathsf{exists\_succ}_k$. However, after β-reduction of $\mathsf{exists\_succ}_k\ r_1\ P$ and then $\mathsf{exists}_k(\lambda r_2 \dots)$, we get a formula ψ_k equivalent to φ_k and of the form

$$\Big(\mu(P\colon \tau_{k+1}).\,\lambda(r_1\colon \tau_k).\\ (\dots) \vee \big(\mu(F\colon \tau_{k+1}).\lambda(r_2\colon \tau_k)\ ((\dots) \wedge (P\ r_2)) \vee (F\ (\mathsf{next}_k\ r_2))\big)\ r_1\Big)\ \mathsf{init}_k$$

where the $(\dots)$ parts do not contain the recursive variables P and F, hence this formula is tail-recursive. □

The uppper bound and the fact that the lower one holds for the data complexity already yield a hierarchy of expressive power within $\mathrm{HFL}_{\mathsf{tail}}$.

Corollary 10. *For all $k \geq 0$, $\mathrm{HFL}^k_{\mathsf{tail}} \lneq \mathrm{HFL}^{k+1}_{\mathsf{tail}}$.*

Proof. Suppose this was not the case. Then there would be a $k \geq 0$ such that $\mathrm{HFL}^k_{\mathsf{tail}} \equiv \mathrm{HFL}^{k+1}_{\mathsf{tail}}$. We need to distinguish the cases $k = 0$ and $k > 0$.

Let $k = 0$. Note that $\mathrm{HFL}^0_{\mathsf{tail}}$ is a fragment of the modal μ-calculus which can only express regular properties. On the other hand, $\mathrm{HFL}^1_{\mathsf{tail}}$ contains formulas that express non-regular properties, for instance uniform inevitability [2].

Now let $k > 0$ and suppose that for every $\varphi \in \mathrm{HFL}^{k+1}_{\mathsf{tail}}$ there would exist a $\widehat{\varphi} \in \mathrm{HFL}^k_{\mathsf{tail}}$ such that $\widehat{\varphi} \equiv \varphi$. Take the formula φ_{k+1} as constructed above and used in the proof of Thm. 9. Fix some function *enc* which represents a transition system and a state as a string over some suitable alphabet. According to Thm. 9, $L := \{enc(\mathcal{T}, s) \mid \mathcal{T}, s \models \varphi_{k+1}\}$ is a k-EXPSPACE-hard language.

On the other hand, consider $\widehat{\varphi_{k+1}}$ which, by assumption, belongs to $\mathrm{HFL}^k_{\mathsf{tail}}$ and is equivalent to φ_{k+1}. Hence, $L = \{enc(\mathcal{T}, s) \mid \mathcal{T}, s \models \widehat{\varphi_{k+1}}\}$. According to Thm. 5, we have $L \in (k{-}1)$-EXPSPACE and therefore k-EXPSPACE $= (k-1)$-EXPSPACE which contradicts the space hierarchy theorem [16]. □

5 Conclusion

We have presented a fragment of HFL that, given equal type order, is more efficient to model-check than regular HFL: Instead of $(k + 1)$-fold exponential time, model-checking an order $k + 1$ tail-recursive formula requires only k-fold exponential space. We have shown that this is optimal. Moreover, since the result already holds for data complexity, the space hierarchy theorem yields a strict hierarchy of expressive power within $\mathrm{HFL}_{\mathsf{tail}}$.

The definition of tail recursion presented in this paper was designed for clarity and can be extended with some syntactic sugar. For example, we take advantage of the free nondeterminism available due to Savitch's Thereom to resolve disjunctions and modal diamonds. One can, of course, also design a tail-recursive fragment that uses co-nondeterminism, allows unrestricted use of conjunctions and modal boxes, but restricts use of their duals. For symmetry reasons this fragment enjoys the same complexity theoretic properties as the fragment presented here. In fact, it is even possible to mix both fragments: tail recursion demands that (some) subformulas under operators that are not covered by Savitch's Theorem be *safe* in the sense that they have no free fixpoint variables. It is completely reasonable to allow a switch from nondeterministic tail recursion to co-nondeterministic tail recursion, and vice versa, at such safe points. Since clever use of negation can emulate this in the fragment presented in this paper, we have chosen not to introduce such switches in this paper for reasons of clarity. Making co-nondeterminism available can be helpful if formulas in negation normal form, which HFL admits, are needed.

An open question is how much the restrictions of tail recursion can be lifted for fixpoint definitions of order below the maximal order in a formula. A naïve approach would conclude that one can lift tail recursion for fixpoints of low order, since there is enough space available to compute their semantics via traditional fixpoint iteration. However, this can have undesired effects when lower-order fixpoints are nested with higher-order ones, breaking tail recursion. Outlining the definite border on what is possible with respect to lower-order fixpoints is a direction for future work.

References

1. H. R. Andersen. A polyadic modal μ-calculus. Technical Report ID-TR: 1994-195, Dept. of Computer Science, Technical University of Denmark, Copenhagen, 1994.
2. R. Axelsson and M. Lange. Model checking the first-order fragment of higher-order fixpoint logic. In *Proc. 14th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR'07*, volume 4790 of *LNCS*, pages 62–76. Springer, 2007.
3. R. Axelsson, M. Lange, and R. Somla. The complexity of model checking higher-order fixpoint logic. *Logical Methods in Computer Science*, 3:1–33, 2007.
4. E. A. Emerson. Uniform inevitability is tree automaton ineffable. *Information Processing Letters*, 24(2):77–79, 1987.
5. D. Harel, A. Pnueli, and J. Stavi. Propositional dynamic logic of nonregular programs. *Journal of Computer and System Sciences*, 26(2):222–243, 1983.
6. J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Trans. AMS*, 117:285–306, 1965.
7. D. Janin and I. Walukiewicz. On the expressive completeness of the propositional μ-calculus with respect to monadic second order logic. In *CONCUR*, pages 263–277, 1996.
8. N. D. Jones. The expressive power of higher-order types or, life without CONS. *Journal of Func. Prog.*, 11(1):5–94, 2001.
9. D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, Dec. 1983.
10. M. Lange. Model checking propositional dynamic logic with all extras. *Journal of Applied Logic*, 4(1):39–49, 2005.

11. M. Lange. Temporal logics beyond regularity, 2007. Habilitation thesis, University of Munich, BRICS research report RS-07-13.
12. M. Lange and É. Lozes. Capturing bisimulation-invariant complexity classes with higher-order modal fixpoint logic. In *Proc. 8th Int. IFIP Conf. on Theoretical Computer Science, TCS'14*, volume 8705 of *LNCS*, pages 90–103. Springer, 2014.
13. M. Lange and R. Somla. Propositional dynamic logic of context-free programs and fixpoint logic with chop. *Information Processing Letters*, 100(2):72–75, 2006.
14. M. Otto. Bisimulation-invariant PTIME and higher-dimensional μ-calculus. *Theor. Comput. Sci.*, 224(1-2):237–265, 1999.
15. W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.
16. R. E. Stearns, J. Hartmanis, and P. M. Lewis II. Hierarchies of memory limited computations. In *Proc. 6th Ann. Symp. on Switching Circuit Theory and Logical Design*, pages 179–190. IEEE, 1965.
17. P. van Emde Boas. The convenience of tilings. In A. Sorbi, editor, *Complexity, Logic, and Recursion Theory*, volume 187 of *Lecture notes in pure and applied mathematics*, pages 331–363. Marcel Dekker, Inc., 1997.
18. M. Viswanathan and R. Viswanathan. A higher order modal fixed point logic. In *Proc. 15th Int. Conf. on Concurrency Theory, CONCUR'04*, volume 3170 of *LNCS*, pages 512–528. Springer, 2004.