Fundamenta Informaticae 178 (2021) 1–30
DOI 10.3233/FI-2021-1996
IOS Press

The Complexity of Model-Checking Tail-Recursive Higher-Order Fixpoint Logic

Florian Bruse*, **Martin Lange**
University of Kassel, Kassel, Germany
florian.bruse@uni-kassel.de, martin.lange@uni-kassel.de

Etienne Lozes
University of Nice Sophia Antipolis, I3S, CNRS, France
etienne.lozes@unice.fr

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) [1] is an extension of the modal μ-calculus [2] 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 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.

*Address for correspondence: University of Kassel, School of Electrical Engineering and Computer Science, Kassel, Germany

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 [3]. HFL is still bisimulation-invariant [1] but it has the power to express non-regular, i.e. non-MSO-definable properties like certain assume-guarantee properties [1]; all context-free [4] and even some context-sensitive reachability properties [5]; structural properties like being a balanced tree, being bisimilar to a word, [5] 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-calculus. The aforementioned examples are all expressible in fragments of low order, namely in HFL^1 or in exceptional cases only HFL^2. Later on, we give an example of a reachability property expressible in HFL^3 which is unlikely to be expressible in 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 [6]. The case of $k = 0$ is reasonably simple since the expressive power of the modal μ-calculus – which equals HFL^0 – is 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 [7]. For $k = 1$, the strict increase in expressive power was shown using a diagonalisation argument [1]. For arbitrary $k > 0$, strictness follows from considerations in computational complexity: model checking HFL^k is k-EXPTIME-complete [6] 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 [8] 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, called *tail recursion*, which causes space-efficiency in a relative sense. This criterion had 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 [9] captures the bisimulation-invariant fragment of polynomial time [10], $\mathrm{PHFL}^0 \equiv \mathrm{P}/{\sim}$ in short, it was shown that $\mathrm{PHFL}^1 \equiv \mathrm{EXPTIME}/{\sim}$ [11], 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, $\mathrm{PHFL}^1_{\mathsf{tail}} \equiv \mathrm{PSPACE}/{\sim}$ [11]. The same holds for the tail-recursive fragment of the modal μ-calculus and NLOGSPACE over graphs equipped with an additional particular partial order, but it is unlikely to hold for the class of all graphs [11].

These results can indeed be seen as an indication that tail recursion is 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 [12].

The computational complexity of the tail-recursive fragments of each HFL^k has been identified to be $(k-1)$-EXPSPACE in a preliminary version of this paper [13]. Here we give a weaker definition which results in larger tail-recursive fragments, i.e. it gives better space-efficient complexity bounds for more properties than the previous version. While this previous version allowed properties to either result in a nondeterministic or a co-nondeterministic evaluation procedure, the extended definition here makes use of the fact that bounded alternation between existential and universal nondeterminism can be eliminated to result in a deterministic procedure without exceeding the space bounds [14].

The paper is organised as follows. In Sect. 2 we recall HFL and define its tail-recursive fragments. In Sect. 3 we present upper bounds; matching lower bounds are presented in Sect. 4. Sect. 5 concludes the paper with remarks on further work in this area.

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, $\xrightarrow{a} \subseteq S^2$ 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. Contrary to the situation for the modal μ-calculus, which only allows subsets of a given LTS to be defined, HFL can also define objects of higher type orders. This is captured by the following system of simple types. It defines types inductively from a ground type via function forming. In order to trace the role of negation, which is important to guarantee monotonicity for fixpoints to be well-defined, *variances* are used to indicate that a function is monotone $(+)$, antitone $(-)$, or unrestricted (0) in its argument. Syntactically, this means the argument is used exclusively positively, exclusively negatively or possibly in both ways.

The set of HFL-types is given by the grammar

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

where $v \in \{+, -, 0\}$.

The order $\mathsf{ord}(\tau)$ of a type τ is defined inductively as $\mathsf{ord}(\bullet) = 0$, and $\mathsf{ord}(\tau_1^v \to \tau_2) = \max(1 + \mathsf{ord}(\tau_1), \mathsf{ord}(\tau_2))$. The function type constructor $\to$ is right-associative. Thus, every type can be written as $\tau_1^{v_1} \to \dots \tau_m^{v_m} \to \bullet$.

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 $f \in [\![\tau]\!]$ if f inhabits the underlying set of $[\![\tau]\!]$. These lattices are then inductively defined as follows:

- $[\![\bullet]\!]^{\mathcal{T}}$ is the lattice $\mathfrak{P}(\mathcal{S})$ ordered by the inclusion relation $\subseteq$,
- $[\![\tau_1^v \to \tau_2]\!]^{\mathcal{T}}$ is the lattice whose domain is the set of all (if $v = 0$), resp. all monotone (if $v = +$), resp. all antitone (if $v = -$) functions of type $[\![\tau_1]\!]^{\mathcal{T}} \to [\![\tau_2]\!]^{\mathcal{T}}$ ordered pointwise, i.e. $f \sqsubseteq_{\tau_1^v \to \tau_2} g$ iff $f(x) \sqsubseteq_{\tau_2} g(x)$ for all $x \in [\![\tau_1]\!]^{\mathcal{T}}$.

Note that the set of all functions into a complete lattice is a complete lattice again when ordered pointwise.

The height the *height* $\mathsf{ht}(\tau)$ of such a type lattice is defined as the longest strictly ascending chain in that lattice. Clearly, the height of $[\![\bullet]\!]^{\mathcal{T}}$ is $|S| + 1$ if S is the underlying set of $\mathcal{T}$. The height of a function type lattice $[\![\tau_1 \to \tau_2]\!]^{\mathcal{T}}$ is k-fold exponential in the size of the underlying set of $\mathcal{T}$ for $k = \mathsf{ord}(\tau_1 \to \tau_2)$. For precise estimates, see [6].

Example 2.1. Let $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a}\}_{a \in \mathcal{A}}, \ell)$ be an LTS, and let $T, T' \subseteq S$. Then the first-order function

$$X \mapsto \{s \mid \text{ ex. } t \in X \text{ s.t. } s \xrightarrow{a} t\}$$

is an element of $[\![\bullet^+ \to \bullet]\!]$. Moreover, the second-order function $f \mapsto \mathcal{S} \setminus f(f\,T)$ is an element of $[\![(\bullet^v \to \bullet)^- \to \bullet]\!]$ independently of the value of v and the second-order function $f \mapsto \mathcal{S} \setminus f(f\,T) \cup f(T')$ is an element of $[\![(\bullet^v \to \bullet)^0 \to \bullet]\!]$ independently of the value of v.

Finally, if $T \subseteq T'$ the functions g_T and $g_{T'}$ defined as $f \mapsto S \setminus f(f\,T)$, respectively $f \mapsto S \setminus f(f\,T')$ compare in the following way: in $[\![(\bullet^+ \to \bullet)^- \to \bullet]\!]$, we have that $g_{T'} \sqsubseteq_{(\bullet^+ \to \bullet)^- \to \bullet} g_T$, since $g_{T'}(x) \subseteq g_T(x)$ for all $x \subseteq S$, due to the monotonicity of f. On the other hand, also $g_{T'} \sqsubseteq_{(\bullet^- \to \bullet)^- \to \bullet} g_T$ holds, since the twofold application of the antimonotone f yields that the term $f(f\,T)$ is monotone in T.

Remark 2.2. The only role of the variances is to ensure that the semantics of HFL formulas is well-defined. Hence, the precise workings of the variances, and the typing system in the following paragraph, are not central to the paper, since we only work with formulas that have well-defined semantics.

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

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

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. Note that fixpoint variables have no variance decoration; instead the typing rules ensure that they will be used positively only.

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 reason for this is that in the tail-recursive fragment (to be defined below), the presence of negation requires that the formula to be negated contains no free fixpoint variables. Hence, formulas that clearly are permitted if $\wedge$ etc. are built-in would be illegal simply due to the implicit negation required by rewriting a formula to only use the minimal set of operators.

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. Several consecutive λ abstractions will be compressed in favour of readability; e.g. $\lambda(x\colon \tau).\ \lambda(y\colon \sigma).\ \psi$ will be written as $\lambda(x\colon \tau, y\colon \sigma).\ \psi$ or just $\lambda(x, y\colon \tau).\ \psi$ if $\tau = \sigma$. Sometimes, we omit type and/or variance annotations.

The set of subformulas $\mathsf{sub}(\varphi)$ of a formula φ is defined in the usual way. 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 subformula φ' in the unique occurrence of the form $\sigma(X\colon \tau).\ \varphi'$ with $\sigma \in \{\mu, \nu\}$. Note that this also introduces a partial order $\succ$ on the fixpoint variables via $X \succ Y$ iff $\sigma Y.\mathsf{fp}(Y)$ is a subformula of $\mathsf{fp}(X)$. 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 $j > i$ since every partial order can be flattened into a total order. A fomula, or a subformula of a formula, is called *fixpoint-closed* if it has no free fixpoint variables, i.e. if it contains no variables not bound by a fixpoint binder in the (sub)formula in question.

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

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

$$
\frac{\Gamma, x^v : \tau_1 \vdash \varphi : \tau_2}{\Gamma \vdash \lambda(x^v : \tau_1).\ \varphi : \tau_1^v \to \tau_2} \qquad
\frac{\Gamma, X^+ : \tau \vdash \varphi : \tau}{\Gamma \vdash \mu(X : \tau).\ \varphi : \tau} \qquad
\frac{\Gamma, X^+ : \tau \vdash \varphi : \tau}{\Gamma \vdash \nu(X : \tau).\ \varphi : \tau}
$$

$$
\frac{\Gamma \vdash \varphi : \tau_1^+ \to \tau_2 \qquad \Gamma \vdash \psi : \tau_1}{\Gamma \vdash \varphi\ \psi : \tau_2} \qquad
\frac{\Gamma \vdash \varphi : \tau_1^- \to \tau_2 \qquad \overline{\Gamma} \vdash \psi : \tau_1}{\Gamma \vdash \varphi\ \psi : \tau_2}
$$

$$
\frac{\Gamma \vdash \varphi : \tau_1^0 \to \tau_2 \qquad \Gamma \vdash \psi : \tau_1 \qquad \overline{\Gamma} \vdash \psi : \tau_1}{\Gamma \vdash \varphi\ \psi : \tau_2}
$$

Figure 1. The HFL typing system.

A sequence of the form $X_1^{v_1}\colon \tau_1, \dots, X_n^{v_n}\colon \tau_n, x_1^{v'_1}\colon \tau'_1, \dots, x_m^{v'_m}\colon \tau'_m$ 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 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 $\mu(X : \bullet).\ \neg\neg X$.

Let φ be a formula, and let $\Gamma \vdash \psi : \tau$ be a judgment for a subformula of φ derived in the proof that φ is well-formed. Then $\mathsf{type}(\psi)$ is τ in the context of φ. Since this context is usually clear, we will not make it explicit from now on. The *order* of φ is the maximal type order k of any subformula of φ and, hence of any type used in a proof of $\emptyset \vdash \varphi : \bullet$. 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 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 $\Gamma \vdash x : \tau$ and $\eta(X) \in [\![\tau']\!]$ if $\Gamma \vdash X : \tau'$. 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 $\eta[X \mapsto f](Y) = f$ if $X = Y$. Updates for λ-variables are defined analogously.

$$
\begin{aligned}
[\![\Gamma \vdash p : \bullet]\!]_\eta &= \{s \in \mathcal{S} \mid p \in \ell(s)\} \\
[\![\Gamma \vdash \varphi \vee \psi : \bullet]\!]_\eta &= [\![\Gamma \vdash \varphi : \bullet]\!]_\eta \cup [\![\Gamma \vdash \psi : \bullet]\!]_\eta \\
[\![\Gamma \vdash \varphi \wedge \psi : \bullet]\!]_\eta &= [\![\Gamma \vdash \varphi : \bullet]\!]_\eta \cap [\![\Gamma \vdash \psi : \bullet]\!]_\eta \\
[\![\Gamma \vdash \langle a \rangle \varphi : \bullet]\!]_\eta &= \{s \in \mathcal{S} \mid \text{ ex. } t \in [\![\Gamma \vdash \varphi : \bullet]\!]_\eta \text{ s.t. } s \xrightarrow{a} t\} \\
[\![\Gamma \vdash [a] \varphi : \bullet]\!]_\eta &= \{s \in \mathcal{S} \mid \text{ f.a. } t \in \mathcal{S} \text{ with } s \xrightarrow{a} t \text{ holds } t \in [\![\Gamma \vdash \varphi : \bullet]\!]_\eta\} \\
[\![\Gamma \vdash x : \tau]\!]_\eta &= \eta(x) \\
[\![\Gamma \vdash X : \tau]\!]_\eta &= \eta(X) \\
[\![\Gamma \vdash \lambda(x^v : \tau_1) : \tau_1^v \to \tau_2]\!]_\eta &= f \in [\![\tau_1^v \to \tau_2]\!] \text{ s.t.} \\
&\qquad\qquad \text{f.a. } y \in [\![\tau_1]\!].\ f(y) = [\![\Gamma, x^v : \tau_1 \vdash \varphi : \tau_2]\!]_{\eta[x \mapsto y]} \\
[\![\Gamma \vdash \varphi\, \psi : \tau_2]\!]_\eta &= [\![\Gamma \vdash \varphi : \tau_1^v \to \tau_2]\!]_\eta\, ([\![\Gamma \vdash \psi : \tau_1]\!]_\eta) \\
[\![\Gamma \vdash \mu(X : \tau).\varphi : \tau]\!]_\eta &= \bigsqcap \{d \in [\![\tau]\!] \mid [\![\Gamma, X : \tau^+ \vdash \varphi : \tau]\!]_{\eta[X \mapsto d]} \sqsubseteq_\tau d\} \\
[\![\Gamma \vdash \nu(X : \tau).\varphi : \tau]\!]_\eta &= \bigsqcup \{d \in [\![\tau]\!] \mid d \sqsubseteq_\tau [\![\Gamma, X : \tau^+ \vdash \varphi : \tau]\!]_{\eta[X \mapsto d]}\}
\end{aligned}
$$

Figure 2. Semantics of HFL.

The semantics of an HFL formula is defined inductively as per Fig. 2. Note that the least and greatest fixpoints defined in the last two lines of the figure exist since the typing system enforces that $d \mapsto [\![\Gamma, X : \tau^+ \vdash \varphi : \tau]\!]_{\eta[X \mapsto d]}$ is monotone in d. Since $[\![\tau]\!]$ is a complete lattice, so is $[\![\tau \to \tau]\!]$, whence the fixpoints exist by the Knaster-Tarski Theorem. For more details we refer to [1].

Given an LTS $\mathcal{T}$ and a state s in its underlying set, we write $\mathcal{T}, s \models_\eta \varphi : \tau$ if $s \in [\![\Gamma \vdash \varphi : \tau]\!]_\eta$ for suitable Γ and abbreviate the special case with a closed formula of ground type writing $\mathcal{T}, s \models \varphi$ instead of $\mathcal{T}, s \models_\emptyset \varphi : \bullet$.

Tail recursion. 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 [11], but restricted to orders 0 and 1, and later for the monadic case of full HFL [13]. The following definition is an expanded version of the latter definition.

Here we define tail recursion with respect to some type order k. Intuitively, order-k tail-recursion restricts the occurrence of free fixpoint variables in the syntax of formulas such that

- free fixpoint variables do not occur in an operand position,
- subformulas with free fixpoint variables can have fully unrestricted nondetermistic operators, i.e. $\vee, \langle a \rangle$ but limited universal branching, i.e. $\wedge, [a]$, or fully unrestricted universal operators, but limited branching in nondeterministic operators,
- negation is only allowed for fixpoint-closed formulas and
- subformulas that are fixpoint-closed and do not contain fixpoint binders of order k are not restricted in the position of free fixpoint variables.

The intuition behind these restrictions is the following: a free fixpoint variable in the operand of an application constitutes a recursive call that is not tail recursive in the classical sense, since, in applicative evaluation order, the operator of this application cannot be evaluated until the value of the operand is known. Boolean and modal connectives constitute branching: if both sides of a boolean connective contain free fixpoint variables, both recursive calls have to be evaluated independently, which clearly is not tail recursive at first hand. The reasoning for the modal operators is similar. However, since we aim for space-restricted complexity results, Savitch's Theorem yields either free nondeterministic or free universal choice, but not both at the same time. This requires branching to be limited to either nondeterministic or universal operators.

Moreover, a fixpoint-closed subformula cannot contain recursive calls to fixpoint definitions outside of itself, whence the semantics of said formula can be computed independently of the rest, and then freely negated without introducing hidden unrestricted alternation between nondeterministic and universal choice. Hence, negation at fixpoint-free subformulas constitutes alternation between nondeterministic and universal choice, but the restriction to fixpoint-closed subformulas means the alternation is bounded by the size of the formula and, hence, permissible by iterated application of Savitch's Theorem. Finally, the last point concerns subformulas that, due to their low type-theoretic order, can be evaluated in the desired complexity bounds already using a standard HFL model checker.

Definition 2.3. Let S_1, S_2 be sets. We write $S_1 \xleftrightarrow{\emptyset} S_2$ to denote that at least one of S_1 and S_2 is empty. Note that, in particular, $\emptyset \xleftrightarrow{\emptyset} \emptyset$ holds.

An HFL formula φ of order at most k is *order-k tail-recursive* if the statement $\mathsf{tail}^k(\varphi, \emptyset, A)$ for some $A \in \{\mathsf{N}, \mathsf{U}, \mathsf{L}\}$ can be derived via the rules in Fig. 3. $\mathrm{HFL}^k_{\mathsf{tail}}$ consists of all order-k tail-recursive formulas in HFL^k.

$$(\mathsf{alter})\ \frac{A \in \{\mathsf{N},\mathsf{U},\mathsf{L}\} \qquad A' \in \{\mathsf{N},\mathsf{U}\} \qquad \mathsf{tail}^k(\varphi,\emptyset,A)}{\mathsf{tail}^k(\varphi,\emptyset,A')} \qquad (\mathsf{prop})\ \frac{A \in \{\mathsf{N},\mathsf{U},\mathsf{L}\}}{\mathsf{tail}^k(p,\emptyset,A)}$$

$$(\mathsf{var})\ \frac{A \in \{\mathsf{N},\mathsf{U},\mathsf{L}\}}{\mathsf{tail}^k(x,\emptyset,A)} \qquad (\mathsf{fvar})\ \frac{A \in \{\mathsf{N},\mathsf{U},\mathsf{L}\}}{\mathsf{tail}^k(X,\{X\},A)} \qquad (\neg)\ \frac{A, \in \{\mathsf{N},\mathsf{U}\} \qquad \mathsf{tail}^k(\varphi,\emptyset,A)}{\mathsf{tail}^k(\neg\varphi,\emptyset,A)}$$

$$(\neg_{\mathsf{L}})\ \frac{\mathsf{tail}^k(\varphi,\mathcal{X},\mathsf{L})}{\mathsf{tail}^k(\neg\varphi,\mathcal{X},\mathsf{L})} \qquad (\vee)\ \frac{A \in \{\mathsf{N},\mathsf{L}\} \qquad \mathsf{tail}^k(\varphi_1,\mathcal{X}_1,A) \qquad \mathsf{tail}^k(\varphi_2,\mathcal{X}_2,A)}{\mathsf{tail}^k(\varphi_1 \vee \varphi_2,\mathcal{X}_1 \cup \mathcal{X}_2,A)}$$

$$(\vee_{\mathsf{U}})\ \frac{\mathsf{tail}^k(\varphi_1,\mathcal{X}_1,\mathsf{U}) \qquad \mathsf{tail}^k(\varphi_2,\mathcal{X}_2,\mathsf{U}) \qquad \mathcal{X}_1 \overset{\emptyset}{\longleftrightarrow} \mathcal{X}_2}{\mathsf{tail}^k(\varphi_1 \vee \varphi_2,\mathcal{X}_1 \cup \mathcal{X}_2,\mathsf{U})}$$

$$(\wedge)\ \frac{A \in \{\mathsf{U},\mathsf{L}\} \qquad \mathsf{tail}^k(\varphi_1,\mathcal{X}_1,A) \qquad \mathsf{tail}^k(\varphi_2,\mathcal{X}_2,A)}{\mathsf{tail}^k(\varphi_1 \wedge \varphi_2,\mathcal{X}_1 \cup \mathcal{X}_2,A)}$$

$$(\wedge_{\mathsf{N}})\ \frac{\mathsf{tail}^k(\varphi_1,\mathcal{X}_1,\mathsf{N}) \qquad \mathsf{tail}^k(\varphi_2,\mathcal{X}_2,\mathsf{N}) \qquad \mathcal{X}_1 \overset{\emptyset}{\longleftrightarrow} \mathcal{X}_2}{\mathsf{tail}^k(\varphi_1 \wedge \varphi_2,\mathcal{X}_1 \cup \mathcal{X}_2,\mathsf{N})} \qquad (\langle a\rangle)\ \frac{A \in \{\mathsf{N},\mathsf{L}\} \qquad \mathsf{tail}^k(\varphi,\mathcal{X},A)}{\mathsf{tail}^k(\langle a\rangle\varphi,\mathcal{X},A)}$$

$$(\langle a\rangle_{\mathsf{U}})\ \frac{\mathsf{tail}^k(\varphi,\emptyset,\mathsf{U})}{\mathsf{tail}^k(\langle a\rangle\varphi,\emptyset,\mathsf{U})} \qquad ([a])\ \frac{A \in \{\mathsf{U},\mathsf{L}\} \qquad \mathsf{tail}^k(\varphi,\mathcal{X},A)}{\mathsf{tail}^k([a]\varphi,\mathcal{X},A)} \qquad ([a]_{\mathsf{N}})\ \frac{\mathsf{tail}^k(\varphi,\emptyset,\mathsf{N})}{\mathsf{tail}^k([a]\varphi,\emptyset,\mathsf{N})}$$

$$(\mathsf{app})\ \frac{A \in \{\mathsf{N},\mathsf{U}\} \qquad A' \in \{\mathsf{N},\mathsf{U}\,\mathsf{L}\} \qquad \mathsf{tail}^k(\varphi_1,\mathcal{X},A) \qquad \mathsf{tail}^k(\varphi_2,\emptyset,A')}{\mathsf{tail}^k(\varphi_1\,\varphi_2,\mathcal{X},A)}$$

$$(\mathsf{app}_{\mathsf{L}})\ \frac{\mathsf{tail}^k(\varphi_1,\mathcal{X}_1,\mathsf{L}) \qquad \mathsf{tail}^k(\varphi_2,\mathcal{X}_2,\mathsf{L})}{\mathsf{tail}^k(\varphi_1\,\varphi_2,\mathcal{X}_1 \cup \mathcal{X}_2,\mathsf{L})} \qquad (\lambda)\ \frac{A \in \{\mathsf{N},\mathsf{U},\mathsf{L}\} \qquad \mathsf{tail}^k(\varphi,\mathcal{X},A)}{\mathsf{tail}^k(\lambda(x^v : \tau).\,\varphi,\mathcal{X},A)}$$

$$(\mathsf{fp})\ \frac{\sigma \in \{\mu,\nu\} \qquad A \in \{\mathsf{N},\mathsf{U}\} \qquad \mathsf{tail}^k(\varphi,\mathcal{X},A)}{\mathsf{tail}^k(\mu(X : \tau).\,\varphi,\mathcal{X} \setminus \{X\},A)}$$

$$(\mathsf{fp}_{\mathsf{L}})\ \frac{\sigma \in \{\mu,\nu\} \qquad \mathit{ord}(\tau) < k \qquad \mathsf{tail}^k(\varphi,\mathcal{X},\mathsf{L})}{\mathsf{tail}^k(\sigma(X : \tau).\,\varphi,\mathcal{X} \setminus \{X\},\mathsf{L})}$$

Figure 3. Derivation rules for establishing order-k tail-recursiveness.

We often omit the order and speak of just a tail-recursive formula. In this case, it is asumed to be order-k tail-recursive where k is the order of said formula. Note that if $\mathsf{tail}^k(\varphi,\mathcal{X},A)$ can be derived, then $\mathcal{X}$ is exactly the set of free fixpoint variables of φ. The three *modes* N, U and L indicate whether nondeterministic operators (N) or universal operators (U) can be used without restriction. For example, rules $(\vee)$ and $(\vee_{\mathsf{U}})$ govern the behavior of tail recursion around disjunctions: if the subformula in question is in mode N, then both subformulas of a disjunction may contain free fixpoint variables, assuming a judgment for them in mode N can be derived. On the other hand, if a disjunction is in mode U, then at most one subformula can have free fixpoint variables. Note that, via rule (alter), subformulas without free fixpoint variables can always add modes N and U if there is a derivation for them for at least one mode. This is due to the above observation that fixpoint-closed subformulas cannot contain undesired recursive dependencies outside of themselves.

A derivation for a subformula in mode L (for *lower*) means that it does not contain fixpoint binders of order k. Intuitively, the formula is equivalent to one in HFL^{k-1} and, hence, harmless. For this reason, occurrences of free fixpoint variables are completely unrestricted in a subformula that has a derivation for mode L. This includes fixpoint variables in operand position. However, note that a derivation for mode L is only useful if it ends in some subformula that is fixpoint-variable closed.

We give some examples of tail-recursive, resp. non-tail-recursive formulas at various (low) type orders.

Example 2.4. The tail-recursive HFL^1 formula $\big(\nu F.\lambda x.\lambda y.(x \Rightarrow y) \wedge (F\ \langle a\rangle x\ \langle b\rangle y)\big)\ \top\ \langle b\rangle\top$ expresses a form of assume-guarantee property [1].

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

A similar pattern is being employed by $(\nu(F\colon \bullet \to \bullet).\lambda(X\colon \bullet).X \wedge \bigwedge_{a\in\Sigma} F\ \langle a\rangle X)\ p_{\mathsf{acc}}$ which expresses the existence of a path to a p_{acc}-state with edge labels forming a word w, for any $w \in \{a,b\}^*$. Hence, it expresses universality for NFA over Σ.

Example 2.5. The HFL^1-formula $\big(\mu(F\colon \bullet \to \bullet).\,\lambda(x\colon \bullet).\langle a\rangle(F\,[b]x \wedge F\,[c]x)\big)\,p$ defines the set of all states from which an a-path starts, at the end of which any path of the same length labeled by a combination of b and c ends in a node labeled by P. This formula is not tail recursive, since F appears simultaneously under $\langle a\rangle$ and a conjunction. However, it can be rewritten into a tail-recursive formula of order 2 using the type $\tau = (\bullet \to \bullet) \to \bullet \to \bullet$, namely

$$\Big(\mu(F\colon \tau).\,\lambda(f\colon \bullet \to \bullet, x\colon \bullet).\,(F\,\lambda(y\colon \bullet).\langle a\rangle f\,y)\,[b]x \wedge (F\,\lambda(z\colon \bullet).\langle a\rangle f\,z)\,[c]x\Big)\,p$$

The HFL^2-formula

$$\big(\mu(F\colon (\bullet \to \bullet) \to \bullet).\,\lambda(f\colon \bullet \to \bullet).\,p \vee f\,(F\,\lambda(x\colon \bullet).\,f(f\,x))\big)\,\lambda(y\colon\ \bullet\,\tau\bullet).\,\langle a\rangle y$$

defines the set of all states from which a state labeled by P is reachable via a sequence of $2^n - 1$ a-transitions for some n. It is not tail recursive, since F appears in an operand position. This formula, too, can be rewritten into a tail recursive one using an extra argument.

Example 2.6. Consider the reachability problem "there is a maximal path labelled with a word from L" for some formal language $L \subseteq \Sigma^*$, particularly for $L = \{a^n b^n c^n \mid n \geq 1\}$. It can be formalised by the tail-recursive 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 types $x\colon \bullet$; $f,g,h\colon \tau_1 := \bullet^+ \to \bullet$; $F\colon \tau_1^+ \to \tau_1^+ \to \tau_1^+ \to \bullet^+ \to \bullet$, and where $id := \lambda x.\,x$.

Consider the same reachability question for $L = \{(ab^n)^n \mid n \geq 0\}$. This can be specified in tail-recursive HFL^3 as follows. For better readability we omit type annotations for the moment. We also use the abbreviations $\langle\alpha\rangle := \lambda(x\colon \bullet^+).\langle\alpha\rangle x$ for $\alpha \in \mathcal{A}$, $id := \lambda(x\colon \bullet^+).x$, $\varphi \circ \psi := \lambda(x\colon \bullet^+).\varphi\ (\psi\ x)$ and $\varphi \sqcup \psi := \lambda(x^+\colon \bullet).(\varphi\ x) \vee (\psi\ x)$.

$$\Phi := \Big(\mu Z.\lambda f.\lambda g.(g\ \langle a\rangle\ f) \sqcup \big(Z\ ((\langle b\rangle \circ f)\ (\lambda x,y.x \circ y \circ (g\ x\ y))\big)\Big)\ id\ (\lambda x,y.id)\ p$$

The omitted types are $f,x,y : \tau_1 = \bullet^+ \to \bullet$, $g : \tau_2 = \tau_1^+ \to \tau_1^+ \to \tau_1$ and $Z : \tau_1^+ \to \tau_2^+ \to \tau_1$.

In order to see that Φ indeed formalises the property described above, note that its fixpoint formula is of the simple form $\mu Z.\lambda\_.\chi \vee (Z\ \varphi\ \psi)$, and consider the shape that the arguments to Z take on in each iteration of a fixpoint unfolding: in the i-th iteration they become $\langle b\rangle^i$ and $\lambda x, y.(x \circ y)^i$, respectively. By instantiating and composing them in a suitable way, one can obtain a formula that expresses the existence of a path of the form $(ab^i)^i$.

3. Upper bounds in the exponential space hierarchy

3.1. Verifying tail recursion

In preparation for the result, we first show that it can be verified in time linear in the size of the syntax tree of a formula whether it is order-k tail-recursive. For this we use a bottom-up procedure VERTR that collects all those modes such that a derivation in the respective mode is possible for the given subformula. Crucially, this does not necessarily mean that the derivation can be extended to one for the full formula. Intuitively, this is the same principle as the powerset construction used to determinise finite automata, although it only operates on the set $modes = \{\mathsf{N}, \mathsf{U}, \mathsf{L}\}$. The procedure MODETC that is used in the definition of VERTR emulates the derivation rule (alter), i.e. it adds modes N and U to subformulas that are fixpoint-closed and have at least one successful derivation. The information gathered by VERTR will also be used later to establish an actual derivation tree for a formula that is known to be order-k tail-recursive. Knowledge of the full derivation tree is needed in order to meet the desired complexity bounds for the model checking algorithm.

Lemma 3.1. Let φ be an HFL formula. Then $\mathsf{tail}^k(\varphi, \mathcal{X}, A)$ is derivable for $A \in \{\mathsf{N}, \mathsf{U}, \mathsf{L}\}$ if and only if $\text{MODETC}(\text{VERTR}(\varphi, k)) = (\mathcal{X}, modes)$ and $A \in modes$. Hence, it is decidable in time $\mathcal{O}(|\varphi|)$ whether φ is order-k tail-recursive.

Proof:
Procedure VERTR in Algorithm 1 checks whether an HFL-formula φ is order-k tail-recursive. We prove by induction on the syntax of φ that $\mathsf{tail}^k(\varphi, \mathcal{X}, A)$ for $A \in \{\mathsf{N}, \mathsf{U}, \mathsf{L}\}$ is derivable if and only if $\text{MODETC}(\text{VERTR}(\varphi, k))$ returns $(\mathcal{X}, modes)$ and $A \in modes$. Let ψ be a subformula of φ and assume that the statement has been proved for all proper subformulas of ψ. Depending on the form of ψ, the argument proceeds as follows. The cases of p, X, x or $\lambda x.\, \psi'$ are straight-forward.

Case $\psi = \psi_1 \vee \psi_2$. Let $(\mathcal{X}_i, modes_i)$ be the return value of $\text{MODETC}(\text{VERTR}(\varphi_i, k))$ for $i \in \{1, 2\}$. If either of the $modes_i$ is empty, the algorithm correctly returns $(\_, \emptyset)$.[1] Otherwise, if $A \in modes_i$, then $\mathsf{tail}^k(\varphi_i, \mathcal{X}_i, A)$ is derivable by the induction hypothesis. Note that the only two applicable rules are $(\vee)$ and $(\vee_\mathsf{U})$. If $\mathsf{N} \in modes_1$ and $\mathsf{N} \in modes_2$, then $\mathsf{tail}^k(\varphi, \mathcal{X}_1 \cup \mathcal{X}_2, \mathsf{N})$ is derivable, and the same holds for L. In both cases, both return statements of $\text{VERTR}(\psi, k)$ return $(\mathcal{X}_1 \cup \mathcal{X}_2, modes)$ with $\mathsf{N} \in modes$, respectively $\mathsf{L} \in modes$ since $modes_1 \cap modes_2$ contains N, respectively L.

Moreover, if at least one of the $\mathcal{X}_i$ is $\emptyset$, then rule $(\vee_\mathsf{U})$ is potentially applicable. If both $\mathcal{X}_i$ are empty, then $modes_1 \cap modes_2$ contains U and rule $(\vee_\mathsf{U})$ allows to derive $\mathsf{tail}^k(\psi, \emptyset, \mathsf{U})$, and also

[1] Here, and in the following, the underscore denotes a value that is otherwise irrelevant.

Algorithm 1 Checking for order-k tail-recursion

procedure $\textsc{VerTr}(\varphi, k)$ $\triangleright$ Returns $(\mathcal{X}, \mathit{modes})$
  switch φ **do**
    case $\varphi = p$ **return** $(\emptyset, \{\mathsf{N}, \mathsf{U}, \mathsf{L}\})$
    case $\varphi = X$ **return** $(\{X\}, \{\mathsf{N}, \mathsf{U}, \mathsf{L}\})$
    case $\varphi = x$ **return** $(\emptyset, \{\mathsf{N}, \mathsf{U}, \mathsf{L}\})$
    case $\varphi = \varphi_1 \vee \varphi_2$
      $(\mathcal{X}_i, \mathit{modes}_i) \leftarrow \textsc{ModeTc}(\textsc{VerTr}(\varphi_i, k))$, $i \in \{1, 2\}$
      if $\mathcal{X}_1 \xleftrightarrow{\emptyset} \mathcal{X}_2$ **then return** $(\mathcal{X}_1 \cup \mathcal{X}_2, \mathit{modes}_1 \cap \mathit{modes}_2)$
      else return $(\mathcal{X}_1 \cup \mathcal{X}_2, (\mathit{modes}_1 \cap \mathit{modes}_2) \setminus \{\mathsf{U}\})$
    case $\varphi = \varphi_1 \wedge \varphi_2$
      $(\mathcal{X}_i, \mathit{modes}_i) \leftarrow \textsc{ModeTc}(\textsc{VerTr}(\varphi_i, k))$, $i \in \{1, 2\}$
      if $\mathcal{X}_1 \xleftrightarrow{\emptyset} \mathcal{X}_2$ **then return** $(\mathcal{X}_1 \cup \mathcal{X}_2, \mathit{modes}_1 \cap \mathit{modes}_2)$
      else return $(\mathcal{X}_1 \cup \mathcal{X}_2, (\mathit{modes}_1 \cap \mathit{modes}_2) \setminus \{\mathsf{N}\})$
    case $\varphi = \langle a \rangle \varphi'$
      $(\mathcal{X}, \mathit{modes}) \leftarrow \textsc{ModeTc}(\textsc{VerTr}(\varphi', k))$
      if $\mathcal{X} = \emptyset$ **then return** $(\mathcal{X}, \mathit{modes})$
      else return $(\mathcal{X}, (\mathit{modes} \setminus \{\mathsf{U}\})$
    case $\varphi = [a]\varphi'$
      $(\mathcal{X}, \mathit{modes}) \leftarrow \textsc{ModeTc}(\textsc{VerTr}(\varphi', k))$
      if $\mathcal{X} = \emptyset$ **then return** $(\mathcal{X}, \mathit{modes})$
      else return $(\mathcal{X}, (\mathit{modes} \setminus \{\mathsf{N}\})$
    case $\varphi = \neg\varphi'$
      $(\mathcal{X}, \mathit{modes}) \leftarrow \textsc{ModeTc}(\textsc{VerTr}(\varphi', k))$
      if $\mathcal{X} = \emptyset$ **then return** $(\mathcal{X}, \mathit{modes})$
      else return $(\mathcal{X}, (\mathit{modes} \cap \{\mathsf{L}\})$
    case $\varphi = \lambda x.\, \varphi'$ **return** $\textsc{VerTr}(\varphi', k)$
    case $\varphi = \varphi_1\, \varphi_2$
      $(\mathcal{X}_i, \mathit{modes}_i) \leftarrow \textsc{ModeTc}(\textsc{VerTr}(\varphi_i, k))$, $i \in \{1, 2\}$
      if $\mathcal{X}_2 = \emptyset$ and $\mathit{modes}_2 \neq \emptyset$ **then return** $(\mathcal{X}_1, \mathit{modes}_1)$
      else return $(\mathcal{X}_1 \cup \mathcal{X}_2, \mathit{modes}_1 \cap \mathit{modes}_2 \cap \{\mathsf{L}\})$
    case $\varphi = \sigma(X : \tau).\, \varphi'$
      $(\mathcal{X}, \mathit{modes}) \leftarrow \textsc{ModeTc}(\textsc{VerTr}(\varphi, k))$
      if $\mathit{ord}(\tau) < k$ **then return** $(\mathcal{X} \setminus \{X\}, \mathit{modes})$
      else return $(\mathcal{X} \setminus \{X\}, \mathit{modes} \setminus \{\mathsf{L}\})$
procedure $\textsc{ModeTc}((\mathcal{X}, \mathit{modes}))$
  if $\mathcal{X} = \emptyset$ and $\mathit{modes} \neq \emptyset$ **then return** $(\mathcal{X}, \mathit{modes} \cup \{\mathsf{N}, \mathsf{U}\})$
  else return $(\mathcal{X}, \mathit{modes})$

$\textsc{VerTr}(\psi, k)$ returns $(\emptyset, modes)$ with $\mathsf{U} \in modes$. If there is $j \in \{1, 2\}$ such that $\mathcal{X}_j = \emptyset$ but $\mathcal{X}_{1-j} \neq \emptyset$, then rule $(\vee_\mathsf{U})$ is applicable if and only if $modes_{1-j}$ contains U and $modes_j \neq \emptyset$. In this case rule (alter) implies $\mathsf{U} \in modes_j$ and rule $(\vee_\mathsf{U})$ allows to derive $\mathsf{tail}^k(\psi, \mathcal{X}_1 \cup \mathcal{X}_2, \mathsf{U})$, whence $\textsc{VerTr}(\psi, k)$ returns $(\mathcal{X}_1 \cup \mathcal{X}_2, modes)$ with $\mathsf{U} \in modes$.

Cases $\psi = \psi_1 \wedge \psi_2$, $\psi = \langle a \rangle \psi'$, $\psi = [a]\psi'$ and $\psi = \neg\psi'$. These follow a similar pattern as the case $\psi = \psi_1 \vee \psi_2$.

Case $\psi = \psi_1\, \psi_2$. Let $(\mathcal{X}_i, modes_i)$ be the return value of $\textsc{ModeTc}(\textsc{VerTr}(\varphi_i, k))$ for $i \in \{1, 2\}$. The case where either of the $modes_i$ is empty is again easily seen to work correctly. Otherwise, if $A \in modes_i$, then $\mathsf{tail}^k(\varphi_i, \mathcal{X}_i, A)$ is derivable by the induction hypothesis. Note that the only applicable rules are (app) and $(\mathsf{app}_\mathsf{L})$. Rule (app) is only applicable if $\mathcal{X}_2 = \emptyset$, in which case $\mathsf{tail}^k(\psi, \mathcal{X}_1, A)$ is derivable if $A \in modes_1$ and $modes_2 \neq \emptyset$. In this case, the return value of $\textsc{VerTr}(\psi, k)$ is obtained via the first return call and is $(\mathcal{X}_1, modes)$ with $A \in modes$ if and only if $A \in modes_1$. Note that this also contains the case that $modes_1 \cap modes_2$ contains L, in which case also rule $(\mathsf{app}_\mathsf{L})$ would be applicable to derive $\mathsf{tail}^k(\psi, \mathcal{X}_1, \mathsf{L})$.

If $\mathcal{X}_2 \neq \emptyset$, then only rule $(\mathsf{app}_\mathsf{L})$ is applicable and $\mathsf{tail}^k(\psi, \mathcal{X}_1 \cup \mathcal{X}_2, \mathsf{L})$ is only derivable if $\mathsf{L} \in modes_1$ and $\mathsf{L} \in modes_2$. In this case, the return value of $\textsc{VerTr}(\psi, k)$ is obtained via the second return call and is $(\mathcal{X}_1 \cup \mathcal{X}_2, \{\mathsf{L}\})$ if and only if $\mathsf{L} \in modes_1 \cap modes_2$.

Case $\psi = \sigma(X\colon \tau).\, \psi'$. Let $(\mathcal{X}, modes')$ be the return value of $\textsc{ModeTc}(\textsc{VerTr}(\psi', k))$. The case of an empty $modes'$ is as before. Otherwise, if $A \in modes'$ then $\mathsf{tail}^k(\psi', \mathcal{X}, A)$ is derivable by the induction hypothesis. Note that only rules (fp) and (fp_L) are applicable. If $\mathsf{ord}(\tau) < k$ then $\mathsf{tail}^k(\psi, \mathcal{X} \setminus \{X\}, A)$ is derivable via rule (fp_L) if and only if $\mathsf{tail}^k(\psi', \mathcal{X}, A)$ is derivable. Procedure $\textsc{VerTr}(\psi, k)$ mirrors this in its first return call by returning $(\mathcal{X} \setminus \{X\}, modes')$.

However, if $\mathsf{ord}(\tau) = k$ then only rule (fp) is applicable, and $\mathsf{tail}^k(\psi, \mathcal{X} \setminus \{X\}, A)$ is derivable via rule (fp) if and only if $\mathsf{tail}^k(\psi', \mathcal{X}, A)$ is derivable and $A \neq \mathsf{L}$. Procedure $\textsc{VerTr}(\psi, k)$ mirrors this in its first return call by returning $(\mathcal{X} \setminus \{X\}, modes' \setminus \{\mathsf{L}\})$.

By applying the result of the induction to (φ, k), we obtain that $\mathsf{tail}^k(\varphi, \emptyset, A)$ is derivable for $A \in \{\mathsf{N}, \mathsf{U}, \mathsf{L}\}$ if and only if $\textsc{ModeTc}(\textsc{VerTr}(\varphi, k)) = (\emptyset, modes)$ with $A \in modes$. For the complexity results, note that procedure $\textsc{VerTr}$ performs exactly one recursive call per subformula of φ and calls $\textsc{ModeTc}$ at most once per subformula. The latter procedure runs in constant time, while the former procedure has a constant inner loop. Hence, the overall procedure runs in $\mathcal{O}(|\varphi|)$. ☐

As said before, after running $\textsc{VerTr}(\varphi, k)$, we have not yet established a derivation tree for order-k tail-recursiveness of φ, we just know that, for some subformula ψ of φ, if $(\mathcal{X}, modes)$ is the return value of $\textsc{VerTr}(\psi, k)$, then there is a successful derivation of $\mathsf{tail}^k(\psi, \mathcal{X}, A)$ for each $A \in modes$. However, this is sufficient to generate a derivation tree for order-k tail-recursiveness of φ in a top-down fashion. We do this by generating, for each subformula ψ in the syntax *tree* of φ a triple[2] $\mathsf{info}(\psi) = (\mathcal{X}, A, A')$. The intuition here is as follows: $\mathcal{X}$ just stores the free fixpoint variables of ψ for notational convenience. The other two store which facts exactly are used with respect to ψ in the derivation of order-k tail-recursiveness of φ: each subformula, with the exception of φ itself, atomic subformulas,

[2]This triple is generated for ψ as a subformula of φ. In order to avoid notational clutter, we do not display this since φ is clear from context and fixed throughout the presentation.

and another exception made clear below, appears in the derivation tree twice, once in the premises of a derivation rule, and once in the conclusion. The middle entry, A, signals that $\mathsf{tail}^k(\psi, \mathcal{X}, A)$ is used as the premise of a rule, while the last entry, A', signals that $\mathsf{tail}^k(\psi, \mathcal{X}, A')$ is used as the conclusion of a rule. For example, if the triples $\mathsf{info}(\psi_1 \vee \psi_2) = (\mathcal{X}, \_, \mathsf{U})$, $\mathsf{info}(\psi_1) = (\mathcal{X}, \mathsf{U}, \_)$ and $\mathsf{info}(\psi_2) = (\emptyset, \mathsf{U}, \_)$, get generated then we can conclude that the following instance of rule $(\vee_{\mathsf{U}})$ was used to derive order-k tail-recursiveness of φ:

$$(\vee_{\mathsf{U}})\ \frac{\mathsf{tail}^k(\psi_1, \mathcal{X}, \mathsf{U}) \qquad \mathsf{tail}^k(\psi_2, \emptyset, \mathsf{U}) \qquad \mathcal{X} \xleftrightarrow{\emptyset} \emptyset}{\mathsf{tail}^k(\psi_1 \vee \psi_2, \mathcal{X} \cup \emptyset, \mathsf{U})}$$

Note that $A \neq A'$ can only occur if $\mathcal{X} = \emptyset$. This also signals that an application of rule (alter) was used on ψ, which is the exception to the rule that each subformula that is not φ itself or atomic occurs exactly twice in the derivation tree.

During the procedure to generate these triples, we will temporarily allow the last entry to be ε, which means that it is as of yet undetermined. We begin the procedure with the tuple $\mathsf{info}(\varphi) = (\mathcal{X}, A, \varepsilon)$ such that $A \in \mathit{modes}$, where $(\mathcal{X}, \mathit{modes}) = \textsc{ModeTc}(\textsc{VerTr}(\varphi, k))$. Given a tuple of the form $\mathsf{info}(\psi) = (\mathcal{X}, A, \varepsilon)$, by assumption $\mathsf{tail}^k(\psi, \mathcal{X}, A)$ is derivable. Update $\mathsf{info}(\psi)$ to $(\mathcal{X}, A, A')$ depending on the form of ψ as follows:

Case ψ is p or X or x. Then ψ is a leaf formula. Update $\mathsf{info}(\psi)$ to $(\mathcal{X}, A, A)$. Note that the axiom rules (prop), (var) and (fvar) are applicable with premise A.

Case ψ is of the form $\neg\psi'$, $\langle a\rangle\psi'$, $[a]\psi'$ or $\lambda x.\ \psi'$. Let $(\mathcal{X}, \mathit{modes}) = \textsc{ModeTc}(\textsc{VerTr}(\psi, k))$ and let $(\mathcal{X}', \mathit{modes}') = \textsc{ModeTc}(\textsc{VerTr}(\psi', k))$. Note that necessarily $\mathcal{X} = \mathcal{X}'$ since both formulas have the same free fixpoint variables, and that $\mathit{modes} = \mathit{modes}'$: If $A' \in \mathit{modes}'$, then by rules $(\neg)$, $(\neg_{\mathsf{L}})$, $(\langle a\rangle)$, $(\langle a\rangle_{\mathsf{U}})$, $([a])$, $([a]_{\mathsf{N}})$ and (λ), the premise $\mathsf{tail}^k(\psi', \mathcal{X}', A')$ allows $\mathsf{tail}^k(\psi, \mathcal{X}, A')$ to be derived. On the other hand, if $A' \in \mathit{modes}$ then $\mathsf{tail}^k(\psi, \mathcal{X}, A')$ is derivable, either because $\mathsf{tail}^k(\psi', \mathcal{X}', A')$ is derivable and one of the above rules applies, or because $\mathsf{tail}^k(\psi, \mathcal{X}, A'')$ is derivable via such a rule from $\mathsf{tail}^k(\psi', \mathcal{X}', A'')$ for $A \in \{\mathsf{N}, \mathsf{U}\} \cap \mathit{modes}'$ and $\mathcal{X} = \emptyset$ and rule (alter) allows $\mathsf{tail}^k(\psi, \emptyset, A')$ to be derived from $\mathsf{tail}^k(\psi, \emptyset, A'')$. Since also $\mathcal{X}' = \emptyset$, we have that $\mathsf{tail}^k(\psi', \emptyset, A')$ is derivable from $\mathsf{tail}^k(\psi', \emptyset, A'')$. Update $\mathsf{info}(\psi)$ to $(\mathcal{X}, A, A)$ and continue with $\mathsf{info}(\psi') = (\mathcal{X}', A, \varepsilon)$.

Case ψ is of the form $\sigma(X\colon \tau).\,\psi'$. Let $(\mathcal{X}, \mathit{modes}) = \textsc{ModeTc}(\textsc{VerTr}(\psi, k))$. Also, let $(\mathcal{X}', \mathit{modes}')$ equal $\textsc{ModeTc}(\textsc{VerTr}(\psi', k))$. Note that $\mathcal{X}' = \mathcal{X} \cup \{X\}$. There are two cases: If $\mathsf{ord}(\tau) < k$, then $\mathit{modes}' \subseteq \mathit{modes}'$ since, if $A' \in \mathit{modes}'$, rules (fp) and $(\mathsf{fp}_{\mathsf{L}})$ allows $\mathsf{tail}^k(\psi, \mathcal{X}, A')$ to be derived from $\mathsf{tail}^k(\psi', \mathcal{X}', A')$. If $A \in \mathit{modes}'$, update $\mathsf{info}(\psi)$ to $(\mathcal{X}, A, A)$ and continue with $\mathsf{info}(\psi') = (\mathcal{X}', A, \varepsilon)$. If $A \notin \mathit{modes}'$, there must be $A' \in \mathit{modes}'$ with $A' \neq A$ for if $\mathit{modes}' = \emptyset$, also $\mathit{modes} = \emptyset$ since the only applicable rules with conclusion $\mathsf{tail}^k(\psi, \mathcal{X}, A')$ are (fp), $(\mathsf{fp}_{\mathsf{L}})$ and (alter). The first two are not applicable if $\mathit{modes}' = \emptyset$, and rule (alter) requires a premise of the form $\mathsf{tail}^k(\psi, \mathcal{X}, A'')$ with $A'' \neq A'$ and that premise must necessarily be derived via a rule different from (alter), which does not exist. Hence, there is $A' \in \mathit{modes}'$ with $A' \neq A$. By the same reasoning, $\mathcal{X} = \emptyset$. Since also $A' \in \mathit{modes}$, the premise $\mathsf{tail}^k(\psi', \mathcal{X}', A')$, which is derivable by the definition of $\textsc{ModeTc}(\textsc{VerTr})$, allows $\mathsf{tail}^k(\psi, \mathcal{X}, A')$ to be derived via rule (fp) or $(\mathsf{fp}_{\mathsf{L}})$. Since $\mathcal{X} = \emptyset$, rule (alter) allows $\mathsf{tail}^k(\psi, \emptyset, A)$ to be derived. Update $\mathsf{info}(\psi)$ to $(\mathcal{X}, A, A')$ and continue with $\mathsf{info}(\psi') = (\mathcal{X}', A', \varepsilon)$.

The second case is that $\mathsf{ord}(\tau) = k$. Note that if $A' \in \{\mathsf{N}, \mathsf{U}\} \cap \mathit{modes}'$, then $A' \in \mathit{modes}$ since if $A' \in \mathit{modes}'$ then rules (fp) allows $\mathsf{tail}^k(\psi, \mathcal{X}, A')$ to be derived from $\mathsf{tail}^k(\psi', \mathcal{X}', A')$. If $A \in \mathit{modes}'$, update $\mathsf{info}(\psi)$ to $(\mathcal{X}, A, A)$ and continue with $\mathsf{info}(\psi') = (\mathcal{X}', A, \varepsilon)$. If $A \notin \mathit{modes}'$, via reasoning similar to the case of $\mathsf{ord}(\tau) < k$ we obtain that $\mathit{modes} = \emptyset$ and there is $A' \in \mathit{modes}' \setminus \{\mathsf{L}\}$ with $A' \neq A$. Then also $A' \in \mathit{modes}$ and the premise $\mathsf{tail}^k(\psi', \mathcal{X}', A')$, which is derivable by the definition of $\textsc{ModeTc}(\textsc{VerTr})$, allows $\mathsf{tail}^k(\psi, \mathcal{X}, A')$ to be derived via rule (fp). Since $\mathcal{X} = \emptyset$, rule (alter) allows $\mathsf{tail}^k(\psi, \emptyset, A)$ to be derived. Update $\mathsf{info}(\psi)$ to $(\mathcal{X}, A, A')$ and continue with $\mathsf{info}(\psi') = (\mathcal{X}', A', \varepsilon)$.

Case ψ is of the form $\psi_1\, \psi_2$. Let $(\mathcal{X}, \mathit{modes}) = \textsc{ModeTc}(\textsc{VerTr}(\psi, k))$ and let $(\mathcal{X}_i, \mathit{modes}_i)$ equal $\textsc{ModeTc}(\textsc{VerTr}(\psi_i))$ for $i \in \{1,2\}$. If $A = \mathsf{L}$, then $\mathsf{tail}^k(\psi_i, \mathcal{X}_i, \mathsf{L})$ is derivable for $i \in \{1,2\}$ since the only rule with conclusion $\mathsf{tail}^k(\psi, \mathcal{X}, \mathsf{L})$ is rule $(\mathsf{app}_\mathsf{L})$, which has premises $\mathsf{tail}^k(\psi_i, \mathcal{X}_i, \mathsf{L})$ for $i \in \{1,2\}$. Update $\mathsf{info}(\psi)$ to $(\mathcal{X}, \mathsf{L}, \mathsf{L})$ and continue with both of $\mathsf{info}(\psi_1) = (\mathcal{X}_1, \mathsf{L}, \varepsilon)$ and $\mathsf{info}(\psi_2) = (\mathcal{X}_2, \mathsf{L}, \varepsilon)$.

If $A \neq \mathsf{L}$, then there are two cases: if $\mathcal{X} = \emptyset$, then also $\mathcal{X}_i = \emptyset$ for $i \in \{1,2\}$. By the same reasoning as in the case for negation, modal operators, etc., we have that A also in modes_i for $i \in \{1,2\}$, whence rule (app) is applicable with premises $\mathsf{tail}^k(\psi_1, \emptyset, A)$ and $\mathsf{tail}^k(\psi_2, \emptyset, A)$. Update $\mathsf{info}(\psi)$ to $(\mathcal{X}, A, A)$ and continue with both $\mathsf{info}(\psi_i) = (\mathcal{X}_1, A, \varepsilon)$ and $\mathsf{info}(\psi_2) = (\mathcal{X}, A, \varepsilon)$. If $\mathcal{X} \neq \emptyset$, note that there is no premise such that the conclusion of (alter) yields $\mathsf{tail}^k(\psi, \mathcal{X}, A)$, whence $\mathsf{tail}^k(\psi, \mathcal{X}, A)$ is derived from (app). Hence, $\mathcal{X}_1 = \mathcal{X}$ since $\mathcal{X}_2 = \emptyset$ for otherwise rule (app) would not be applicable. It follows that $A \in \mathit{modes}_1$ and that $\mathit{modes}_2 \neq \emptyset$, whence rule (app) is applicable with premises $\mathsf{tail}^k(\psi_1, \mathcal{X}_1, A)$ and $\mathsf{tail}^k(\psi_2, \emptyset, A')$ with $A' \in \mathit{modes}$. Update $\mathsf{info}(\psi)$ to $(\mathcal{X}, A, A)$ and continue with $\mathsf{info}(\psi_1) = (\mathcal{X}_1, A, \varepsilon)$ and $\mathsf{info}(\psi_2) = (\mathcal{X}_2, A', \varepsilon)$.

Case ψ is of the form $\psi_1 \vee \psi_2$ or $\psi_1 \wedge \psi_2$. Let $(\mathcal{X}, \mathit{modes}) = \textsc{ModeTc}(\textsc{VerTr}(\psi, k))$ and let $(\mathcal{X}_i, \mathit{modes}_i) = \textsc{ModeTc}(\textsc{VerTr}(\psi_i))$ for $i \in \{1,2\}$. Without loss of generality, $\psi = \psi_1 \vee \psi_2$, the case for $\wedge$ is completely symmetric. If $A = \mathsf{L}$, then $\mathsf{L} \in \mathit{modes}_i$ for $i \in \{1,2\}$ since the only rule with conclusion $\mathsf{tail}^k(\psi, \mathcal{X}, \mathsf{L})$ is rule $(\vee)$ with premises $\mathsf{tail}^k(\psi_i, \mathcal{X}_i, \mathsf{L})$ for $i \in \{1,2\}$. Update $\mathsf{info}(\psi)$ to $(\mathcal{X}, \mathsf{L}, \mathsf{L})$ and continue with $\mathsf{info}(\psi_i) = (\mathcal{X}_i, \mathsf{L}, \varepsilon)$ for $i \in \{1,2\}$.

If $A = \mathsf{N}$, then $\mathsf{N} \in \mathit{modes}_i$ for $i \in \{1,2\}$. For the sake of contradiction, assume that $\mathsf{N} \notin \mathit{modes}_i$ for some $i \in \{1,2\}$. Then $\mathcal{X}_i \neq \emptyset$, for otherwise $\mathit{modes}_i = \emptyset$, which is a contradiction, or (alter) would be applicable to derive $\mathsf{tail}^k(\psi_i, \emptyset, \mathsf{N})$ from $\mathsf{tail}^k(\psi_i, \emptyset, A')$ for some $A' \in \mathit{modes}_i$. But if $\mathcal{X}_i \neq \emptyset$, then also $\mathcal{X} \neq \emptyset$, whence there is no possible premise such that rule (alter) derives $\mathsf{tail}^k(\psi, \mathcal{X}, \mathsf{N})$. But since $\mathsf{N} \notin \mathit{modes}_i$, rule $(\vee)$ is also not available, contradicting that $\mathsf{tail}^k(\psi, \mathcal{X}, \mathsf{N})$ is derivable. Hence, $\mathsf{N} \in \mathit{modes}_i$ for $i \in \{1,2\}$ and rule $(\vee)$ is applicable with premises $\mathsf{tail}^k(\psi_i, \mathcal{X}_i, \mathsf{N})$ for $i \in \{1,2\}$ and derives $\mathsf{tail}^k(\psi, \mathcal{X}, \mathsf{N})$. Update $\mathsf{info}(\psi)$ to $(\mathcal{X}, \mathsf{N}, \mathsf{N})$ and continue with $\mathsf{info}(\psi_i) = (\mathcal{X}_i, \mathsf{N}, \varepsilon)$ for $i \in \{1,2\}$.

If $A = \mathsf{U}$, there are two cases. If $\mathcal{X} = \emptyset$, then $\mathsf{N} \in \mathit{modes}$. Use rule (alter) to derive $\mathsf{tail}^k(\psi, \emptyset, \mathsf{U})$ from $\mathsf{tail}^k(\psi, \emptyset, \mathsf{N})$ and refer to the previous case. Update $\mathsf{info}(\psi)$ to $(\mathcal{X}, \mathsf{U}, \mathsf{N})$ and and continue with $\mathsf{info}(\psi_i) = (\mathcal{X}_i, \mathsf{N}, \varepsilon)$ for $i \in \{1,2\}$. If $\mathcal{X} \neq \emptyset$, then rule $\mathsf{tail}^k(\psi, \mathcal{X}, \mathsf{N})$ can not be the conclusion of rule (alter), whence the only rule with this conclusion must be rule $(\vee_\mathsf{U})$. It follows that there is i such that $\mathcal{X}_i = \emptyset$, and, moreover, $\mathsf{N} \in \mathit{modes}_1$ and $\mathsf{N} \in \mathit{modes}_2$, for otherwise $\mathsf{tail}^k(\psi, \mathcal{X}, \mathsf{U})$ would not be derivable. Update $\mathsf{info}(\psi)$ to $(\mathcal{X}, \mathsf{U}, \mathsf{U})$ and continue with $\mathsf{info}(\psi_i) = (\mathcal{X}_i, \mathsf{U}, \varepsilon)$ for $i \in \{1,2\}$.

Remark 3.2. The above double pass algorithm to determine the exact derivation for an order-k tail-recursive formula φ might seem over-engineered at first, but it is necessary to obtain an algorithm that generates the derivation in time linear in the size of the syntax tree of φ. A single-pass linear time algorithm would have to cope with the following problems: a bottom-up algorithm will not know which mode to assign to a fixpoint variable node, and a top-down algorithm cannot distinguish which mode to use given a boolean connective.

Consider the formula $\mu(X\colon \bullet).\,(X \vee X) \wedge p$. A bottom-up algorithm cannot determine whether $\mathsf{tail}^k(p, \emptyset, \mathsf{L})$ will be used in the derivation since the subformula p is of order 0, whether $\mathsf{tail}^k(p, \emptyset, \mathsf{U})$ or $\mathsf{tail}^k(p, \emptyset, \mathsf{N})$ will be used. All of them are derivable, but only $\mathsf{tail}^k(p, \emptyset, \mathsf{N})$ leads to a successful derivation. Of course, one can collect all the possible derivations that are still possible, which is what procedure VERTR does. However, this does not fix a derivation tree.

On the other hand, a top-down algorithm would have to determine whether tail recursiveness of the subformula $(X \vee X) \wedge p$ is to be derived using rule $(\wedge)$, since it is a conjunction, or rule $(\wedge_{\mathsf{N}})$, since the right conjunct has no free fixpoint variables. The former would be a bad choice, since the conjunct then contains the disjunction $X \vee X$, which definitely requires rule $(\vee)$. Of course, a top-down algorithm is possible if one pre-computes which modes have successful derivations for each subformula, and this is exactly what procedure CHECKTR does.

3.2. Model checking tail-recursive formulas

We construct a bounded-alternation k-EXPSPACE algorithm in order to model check order-$(k+1)$ HFL formulas that are order-$(k+1)$ tail-recursive. The recursion of least and greatest fixpoints is handled by an invocation of Kleene's Fixpoint Theorem, which entails that, on a given finite LTS, each fixpoint is equivalent to some finite approximation, depending on the height of the lattice in question. We implement this by using a counter for each fixpoint variable: upon reaching a fixpoint definition, the counter is initialised to the height of the lattice relevant to the fixpoint variable in question, indicating how many times it can be unfolded. Every time the variable is reached the algorithm will continue with the defining formula of the fixpoint, but decrease the counter by one. Once the counter reaches 0, the algorithm terminates with the default value of false or true, depending on whether the variable in question is a least or greatest fixpoint. In order to connect this procedure to the semantics of HFL, consider the following definition:

Definition 3.3. Let $\mathcal{T}$ be an LTS, let φ be an order-k tail-recursive formula with fixpoint variables in $\mathcal{X}$. For $X \in \mathcal{X}$, let $\sigma_X X.\,\mathsf{fp}(X)$ be the defining formula for X.

Define approximations for all $X \in \mathcal{X}$ via the substitution instances

$$X^0 := \mathsf{fp}(X)[\bot^{\tau_X}/X] \text{ if } \sigma_X = \mu \ , \quad X^0 := \mathsf{fp}(X)[\top^{\tau_X}/X] \text{ if } \sigma_X = \nu \ , \quad X^{i+1} := \mathsf{fp}(X)[X^i/X]$$

where $\bot^{\tau_1 \to \cdots \to \tau_n \to \bullet} = \lambda(x_1\colon \tau_1, \ldots, x_n\colon \tau_n).\,\bot$ and $\top^{\tau_1 \to \cdots \to \tau_n \to \bullet} = \lambda(x_1\colon \tau_1, \ldots, x_n\colon \tau_n).\,\top$

Given an environment η and a mapping $\mathsf{cnt}\colon \mathcal{X} \to \mathbb{N}$, define η^{cnt} as

$$\eta^{\mathsf{cnt}}(x) := \eta(x) \ , \quad \eta^{\mathsf{cnt}}(X) := [\![X^{\mathsf{cnt}(X)}]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}.$$

Note that, even though the definition looks circular, η^{cnt} is well-defined since $[\![X^{\mathsf{cnt}}]\!]^{\mathcal{T}}$ does not contain X anymore.

For $\mathsf{cnt}\colon \mathcal{X} \to \mathbb{N}$ and $\mathsf{cnt}'\colon \mathcal{X} \to \mathbb{N}$, we define $\mathsf{cnt}' < \mathsf{cnt}$ as follows: $\mathsf{cnt}' < \mathsf{cnt}$ if there is a variable X such that $\mathsf{cnt}'(X) < \mathsf{cnt}(X)$ and $\mathsf{cnt}'(Y) = \mathsf{cnt}(Y)$ for all Y with $Y \succ X$. Moreover, if $\mathsf{cnt}(X) \neq 0$, we define $\mathsf{cnt}[X\texttt{--}]$ as

$$\mathsf{cnt}[X\texttt{--}](Y) = \begin{cases} \mathsf{cnt}(Y) & \text{, if } Y \neq X \\ \mathsf{cnt}(X) - 1 & \text{, if } Y = X. \end{cases}$$

Lemma 3.4. Let $\mathcal{T}$ be a finite LTS. Let φ be an HFL formula of order k that is order-k tail-recursive, let $\psi = \sigma(X : \tau).\,\psi'$ be a subformula of φ such that $\mathsf{info}(\psi) = (\mathcal{X}, \_, \_)$ and let $\mathsf{cnt}\colon \mathcal{X} \to \mathbb{N}$ be a mapping. Then $[\![\sigma(X : \tau).\,\psi']\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}} = [\![\psi']\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}[X \mapsto \mathsf{ht}(\tau)]}}$, where $\mathsf{ht}(\tau)$ is the height of the lattice $[\![\tau]\!]$.

This follows from the Kleene Fixpoint Theorem. Generally, $\mathsf{ht}(\tau)$ is k-fold exponential in the size of $|\mathcal{S}|$ for $k = \mathit{ord}(\tau)$ [6]. Note that a k-fold exponentially large number can be represented by $(k-1)$-fold exponentially many bits.

We claim that procedure `McTr` in Algorithm 2 encodes a valid model checking procedure for order-$(k+1)$ tail-recursive formulas. However, one can see that already storing the counters for order-$(k+1)$ fixpoint variables requires k-fold exponentially many bits. Moreover, the algorithm contains alternation between nondeterministic and universal choice. This clearly exhausts the desired complexity bounds, but *bounded* alternation does not, as explained below. Fortunately, the algorithm is only required to switch between universal and nondeterministic choice at subformulas that appear in an instance of rule (alter). Since application of this rule is restricted to fixpoint-closed subformulas, switches of the modes of alternation do not interact with the recursion and, hence, are bounded in depth by the height of the formula. We combine these switches of the alternation mode with the maximal depth of non-tail-recursive calls the algorithm has to make, into the following measure:

Definition 3.5. Let $\varphi \in \mathrm{HFL}^k_{\mathsf{tail}}$. The *recursion depth* $\mathit{rd}(\psi)$ of a subformula ψ of φ is defined as $\mathit{rd}(\psi) = 0$ if $\mathsf{info}(\psi) = (\emptyset, \_, \mathsf{L})$ and otherwise as follows.

$$\begin{array}{ll}
\mathit{rd}(\psi) = 0 & \text{if } \psi = p \text{ or } \psi = x \text{ or } \psi = X \\
\mathit{rd}(\psi) = \mathit{rd}(\psi') & \text{if } \psi = \langle a \rangle \psi' \text{ and } \mathsf{info}(\psi) = (_, _, \mathsf{N}), \text{ or} \\
& \quad \psi = [a]\psi' \text{ and } \mathsf{info}(\psi) = (_, _, \mathsf{U}), \text{ or} \\
& \quad \psi = \sigma X.\,\psi' \text{ or } \psi = \lambda x.\psi' \\
\mathit{rd}(\psi) = 1 + \mathit{rd}(\psi') & \text{if } \psi = \langle a \rangle \psi' \text{ and } \mathsf{info}(\psi) = (_, _, \mathsf{U}), \text{ or} \\
& \quad \psi = [a]\psi' \text{ and } \mathsf{info}(\psi) = (_, _, \mathsf{N}), \text{ or} \\
& \quad \psi = \neg\psi' \\
\mathit{rd}(\psi) = \max\{r_1, r_2\} & \text{if } \psi = \psi_1 \vee \psi_2 \text{ or } \psi = \psi_1 \wedge \psi_2 \text{ where for } i \in \{1, 2\}: \\
& \quad r_i = \begin{cases} \mathit{rd}(\psi_i) & \text{if } \mathsf{info}(\psi_i) = (\mathcal{X}, _, _) \text{ and } \mathcal{X} \neq \emptyset \\ 1 + \mathit{rd}(\psi_i) & \text{if } \mathsf{info}(\psi_i) = (\emptyset, _, _) \end{cases} \\
\mathit{rd}(\psi) = \max\{\mathit{rd}(\psi_1), 1 + \mathit{rd}(\psi_2)\} & \text{if } \psi = \psi_1\,\psi_2
\end{array}$$

Algorithm 2 Efficient model checking for order-$(k+1)$ tail-recursive HFL formulas

procedure $\mathtt{McTr}(A, s, \psi, (f_1, \dots, f_k), \eta, \mathsf{cnt})$ $\triangleright$ Inputs $\mathcal{T}$ and k not explicitly given
  $(\mathcal{X}, \_, A') \leftarrow \mathsf{info}(\psi)$
  if $A \neq A'$ **then return** $\mathtt{McTr}(A', s, \psi, (f_1, \dots, f_k), \eta, \emptyset)$ $\triangleright$ $\mathcal{X} = \emptyset$ guaranteed
  if $A = \mathsf{L}$ **then**
    $f \leftarrow [\![\psi]\!]^{\mathcal{T}}_{\eta}$ $\triangleright$ use a conventional model checker
    if $s \in ((f\, f_1) \cdots f_k)$ **then return** true
    else return false
  switch ψ **do**
    case $\psi = p$
      if $\mathcal{T}, s \models p$ **then return** true
      else return false
    case $\psi = x$
      $f \leftarrow \eta(x)$
      if $s \in ((f\, f_1) \cdots f_k)$ **then return** true
      else return false
    case $\psi = \psi_1 \vee \psi_2$
      if $A = \mathsf{N}$ **then**
        guess $i \in \{1, 2\}$
        return $\mathtt{McTr}(A, s, \psi_i, (f_1, \dots, f_k), \eta, \mathsf{cnt})$
      else if $A = \mathsf{U}$ **then**
        $(\mathcal{X}_i, \_, \_) \leftarrow \mathsf{info}(\psi_i), i \in \{1, 2\}$
        choose $i \in \{1, 2\}$ s.t. $\mathcal{X}_i = \emptyset$
        $b \leftarrow \mathtt{McTr}(\mathsf{U}, s, \psi_i, (f_1, \dots, f_k), \eta, \emptyset)$
        if $b = \mathsf{true}$ **then return** true
        else return $\mathtt{McTr}(A, s, \psi_{1-i}, (f_1, \dots, f_k), \eta, \mathsf{cnt})$
    case $\psi = \psi_1 \wedge \psi_2$
      if $A = \mathsf{U}$ **then**
        choose $i \in \{1, 2\}$
        return $\mathtt{McTr}(A, s, \psi_i, (f_1, \dots, f_k), \eta, \mathsf{cnt})$
      else if $A = \mathsf{N}$ **then**
        $(\mathcal{X}_i, \_, \_) \leftarrow \mathsf{info}(\psi_i), i \in \{1, 2\}$
        guess $i \in \{1, 2\}$ s.t. $\mathcal{X}_i = \emptyset$
        $b \leftarrow \mathtt{McTr}(\mathsf{N}, s, \psi_i, (f_1, \dots, f_k), \eta, \emptyset)$
        if $b = \mathsf{false}$ **then return** false
        else return $\mathtt{McTr}(A, s, \psi_{1-i}, (f_1, \dots, f_k), \eta, \mathsf{cnt})$
    case $\psi = \neg\psi'$
      $b \leftarrow \mathtt{McTr}(A, s, \psi', (f_1, \dots, f_k), \eta, \emptyset)$
      if $b = \mathsf{true}$ **then return** false
      else return true

case $\psi = \langle a \rangle \psi'$
if $A = \mathsf{N}$ **then**
guess t with $s \xrightarrow{a} t$
return $\mathtt{McTr}(A, t, \psi', (f_1, \ldots, f_k), \eta, \mathsf{cnt})$
else if $A = \mathsf{U}$ **then**
for t with $s \xrightarrow{a} t$ **do**
$b \leftarrow \mathtt{McTr}(A, t, \psi', (f_1, \ldots, f_k), \eta, \emptyset)$
if $b = \mathsf{true}$ **then return** true
return false
case $\psi = [a]\psi'$
if $A = \mathsf{U}$ **then**
choose t with $s \xrightarrow{a} t$
return $\mathtt{McTr}(A, t, \psi', (f_1, \ldots, f_k), \eta, \mathsf{cnt})$
else if $A = \mathsf{N}$ **then**
for t with $s \xrightarrow{a} t$ **do**
$b \leftarrow \mathtt{McTr}(A, t, \psi', (f_1, \ldots, f_k), \eta, \emptyset)$
if $b = \mathsf{false}$ **then return** false
return true
case $\psi = \psi_1\, \psi_2$
$(\emptyset, \_, A_2) \leftarrow \mathsf{info}(\psi_2)$
if $A_2 = \mathsf{L}$ **then**
$f \leftarrow [\![\psi_2]\!]^{\mathcal{T}}_{\eta}$ ▷ use a conventional model checker
return $\mathtt{McTr}(A, s, \psi_1, (f, f_1, \ldots, f_k), \eta, \mathsf{cnt})$
else
$(\tau_1 \to \cdots \to \tau_n \to \bullet) \leftarrow \mathsf{type}(\psi_2)$
$f \leftarrow \emptyset$
for $g_1, \ldots, g_n \in [\![\tau_1]\!]^{\mathcal{T}} \times \cdots \times [\![\tau_n]\!]^{\mathcal{T}}$ **do**
$\mathcal{S}' \leftarrow \emptyset$
for $t \in \mathcal{S}$ **do**
$b \leftarrow \mathtt{McTr}(A_2, t, \psi_2, (g_1, \ldots, g_n), \eta, \emptyset)$
if $b = \mathsf{true}$ **then** $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{t\}$
$f \leftarrow f[(g_1, \ldots, g_n) \mapsto \mathcal{S}']$
return $\mathtt{McTr}(A, s, \psi_1, (f, f_1, \ldots, f_k), \eta, \mathsf{cnt})$
case $\psi = \lambda x.\, \psi'$ **return** $\mathtt{McTr}(A, s, \psi', (f_2, \ldots, f_k), \eta[x \mapsto f_1], \mathsf{cnt})$
case $\psi = \sigma(X : \tau).\, \psi'$ **return** $\mathtt{McTr}(A, s, \psi', (f_1, \ldots, f_k), \eta, \mathsf{cnt}[X \mapsto \mathsf{ht}(\tau)])$
case $\psi = X$
if $\mathsf{cnt}(X) \neq 0$ **then**
return $\mathtt{McTr}(A, s, \mathsf{fp}(X), (f_1, \ldots, f_k), \eta, \mathsf{cnt}[X\texttt{--}])$
else if $\sigma_X = \mu$ **then return** false
else return true

We associate recursion depth 0 to all formulas ψ with $\mathsf{info}(\psi) = (\emptyset, \_, \mathsf{L})$ since model checking of these is handled by a conventional model checker. The other cases are as described: recursion depth increases at negations, at subformulas where (alter) has been used, and at applications.

We give a sketch of how the algorithm works. It is a top-down bounded-alternation algorithm, which means that it proceeds through the formula-tree in a top-down manner, while returning to the defining formula of a fixpoint in case it reaches a fixpoint variable. The algorithm's arguments $(A, s, \psi, (f_1, \ldots, f_k), \eta, \mathsf{cnt})$ are used as follows.

- A determines whether nondeterministic or universal choice is available at the moment;
- s is the state of the underying LTS at which ψ is evaluated;
- $(f_1, \ldots, f_k)$ stores values for the arguments to ψ in case it is of higher order;
- η is a variable assignment to the free λ-variables in ψ;
- cnt records the number of unfoldings that are left for each fixpoint variable.

For a subformula ψ of φ such that $\mathsf{info}(\psi) = (\_, A, A')$, i.e. one such that $\mathsf{tail}^k(\psi, \_, A')$ is used as the conclusion of the derivation rule connecting ψ with its subformulas, the algorithm uses nondeterministic choice if $A' = \mathsf{N}$ and universal choice if $A' = \mathsf{U}$. If $A' = \mathsf{L}$, the problem is forwarded to a conventional model checker, since that one will not exhaust the complexity bounds due to lower type order, but can handle non-tail-recursive subformulas. If $A \neq A'$, i.e. if the current formula to be model checked requires the opposite mode (out of N and U), due to the requirements of tail recursiveness, it must be fixpoint-closed and, hence, the algorithm changes modes mimicking rule (alter) (line 3). This is the point where we need to know the full derivation tree for $\mathsf{tail}^k(\varphi, \emptyset, A)$. At boolean and modal operators, the algorithm proceeds by nondeterministic or universal choice, depending on the mode. The latter also advance the current LTS state of evaluation. If the algorithm is in the mode opposite to the nature of the operator in question, e.g. N and $\wedge$, one of the subformulas in question is fixpoint-closed and its semantics is computed outright (e.g. line 23). Since this outright computation concerns a formula of lower recursion depth, such calls cannot be nested indefinitely.[3] At applications, the semantics of the operand is computed outright (lines 61 and 66). This is possible since it must be of order at most $k - 1$. Again, this constitutes a non-tail-recursive call. The same holds as for the case of mismatched boolean operators. Lambda abstraction is handled by transferring one of the arguments from the arguments to the variable assignment η.

Fixpoints are handled by Kleene's Fixpoint Theorem: when a fixpoint definition is passed, a counter is initialised with the height of the lattice in question. Each time the fixpoint variable in question is passed, the definition of the fixpoint is unfolded (line 77), but the counter is decreased by one. If it reaches 0, the default value of the fixpoint in question is returned. Note that passing a fixpoint variable higher in $\succ$ implicitly resets the counter for a fixpoint variable, since, if it is unfolded again, the defining formula is passed again in line 74 which resets the counter.

We are now ready to give the correctness proof. Termination of `McTr` is relatively easy to see due to the use of the counters that get decreased in each recursive call.

[3]In a practical implementation, such calls would also be cached in case they are needed again for a different iteration of a fixpoint. Since the value of such a formula is fixed and does not depend on fixpoint variables, caching is possible

Lemma 3.6. Let $\mathcal{T}, s_I$ be be an LTS and let φ be a closed HFL formula of ground type that is order-k tail-recursive. Let $\mathsf{info}(\varphi) = (\emptyset, A, \_)$. Then $\mathtt{McTr}(A, s_I, \varphi, \varepsilon, \emptyset, \emptyset)$ returns $\mathtt{true}$ if and only if $\mathcal{T}, s_I \models \varphi$.

Proof:
We will prove the following statement by induction: if ψ is a subformula of φ of type $\tau_1 \to \cdots \to \tau_n \to \bullet$ and $\mathsf{info}(\psi) = (\mathcal{X}, \_, \_)$ and $\mathsf{cnt}\colon \mathcal{X} \to \mathbb{N}$ and $f_1, \dots, f_n$ with $f_i \in [\![\tau_i]\!]^{\mathcal{T}}$ for $1 \le i \le n$ and $A \in \{\mathsf{N}, \mathsf{U}, \mathsf{L}\}$ then

$$s \in (\cdots([\![\psi]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}\, f_1)\cdots\, f_n) \quad \text{iff} \quad \mathtt{McTr}(A, s, \psi, (f_1, \dots, f_n), \eta, \mathsf{cnt}) \text{ returns } \mathtt{true}\,.$$

The statement of the lemma then follows with $\psi = \varphi$, $s = s_I$, $n = 0$, $\eta = \emptyset$ and cnt_I defined for all X as $\mathsf{cnt}_I(X) = \mathsf{ht}(\tau_X)$, where τ_X is the type of X.

The induction has four induction parameters: rd, ψ, cnt and A. If $\mathtt{McTr}(A', \_, \psi', \_, \_, \mathsf{cnt}')$ is called tail-recursively during evaluation of $\mathtt{McTr}(A, \_, \psi, \_, \_, \mathsf{cnt})$, i.e. as a call of the form **return** $\mathtt{McTr}(\dots)$, then either

- $\mathsf{cnt} = \mathsf{cnt}'$, $\psi = \psi'$ and $\mathsf{info}(\psi) = (\_, A, A')$ or
- $\mathsf{cnt} = \mathsf{cnt}'$ and ψ' is a proper subformula of ψ or
- $\mathsf{cnt}' < \mathsf{cnt}$.

Moreover, no such call during the algorithm will increase recursion depth, and if such a call of $\mathtt{McTr}(A', \_, \psi', \_, \_, \mathsf{cnt}')$ is not tail recursive in an algorithmic sense, i.e., if it is of the form $b \leftarrow \mathtt{McTr}(\dots)$ then $rd(\psi') < rd(\psi)$.

Let $\mathtt{McTr}(A, s, \psi, (f_1, \dots, f_k), \eta, \mathsf{cnt})$ be a call of $\mathtt{McTr}$ and let $(\mathcal{X}, A', A'') = \mathsf{info}(\psi)$. If $A \neq A''$ then the algorithm returns the value of $\mathtt{McTr}(A'', s, \psi, (f_1, \dots, f_n), \eta, \mathsf{cnt})$ which, by the induction hypothesis is $\mathtt{true}$ if and only if $s \in [\![\psi]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}$, which is also the claim of the lemma for $\mathtt{McTr}(A, s, \psi, (f_1, \dots, f_k), \eta, \mathsf{cnt})$. Now assume that $A = A''$. If $A = \mathsf{L}$ then $\mathcal{X} = \emptyset$ and $\mathtt{McTr}$ calls a conventional model checker to compute $[\![\psi]\!]^{\mathcal{T}}_{\eta}$, which works correctly by assumption. The claim of the lemma then follows.

If $A \neq \mathsf{L}$, the argument depends on the form of ψ. If ψ is of the forms p, x or $\lambda x.\, \psi'$ then the claim of the lemma is immediate.

Case $\psi = \psi_1 \vee \psi_2$. There are two cases: If $A = \mathsf{N}$, then $\mathtt{McTr}$ guesses $i \in \{1, 2\}$ and returns the value of $\mathtt{McTr}(A, s, \psi_i, \varepsilon, \eta, \mathsf{cnt})$. Hence, $\mathtt{McTr}$ returns $\mathtt{true}$ if and only if there is i such that $\mathtt{McTr}(A, s, \psi_i, \varepsilon, \eta, \mathsf{cnt})$ returns $\mathtt{true}$, which, by the induction hypothesis is the case if and only if $s \in [\![\psi_i]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}$. By the definition of HFL semantics, $s \in [\![\psi]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}$ if and only if $s \in [\![\psi_i]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}$ for at least one $i \in \{1, 2\}$ whence $\mathtt{McTr}(\mathsf{N}, s, \psi, \varepsilon, \eta, \mathsf{cnt})$ returns true if and only if $s \in [\![\psi]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}$.

If $A = \mathsf{U}$, then $\mathtt{McTr}$ universally chooses $i \in \{1, 2\}$ such that $\mathsf{info}(\psi_i) = (\mathcal{X}_i, \_, A_i)$ and $\mathcal{X}_i = \emptyset$ and calculates $b = \mathtt{McTr}(A_i, s, \psi_i, \varepsilon, \eta, \emptyset)$. By the induction hypothesis, $b = \mathtt{true}$ if and only if $s \in [\![\psi_i]\!]^{\mathcal{T}}_{\eta^{\emptyset}}$ which also entails $s \in [\![\psi]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}$. So in case $b = \mathtt{true}$, the algorithm works as claimed in the lemma. Note that also, because $\mathcal{X}_i = \emptyset$, we have that $rd(\psi_i) < rd(\psi)$ so the condition on non-tail-recursive calls is satisfied. In case $b = \mathsf{false}$, the algorithm returns the value of $\mathtt{McTr}(A, s, \psi_{1-i}, \varepsilon, \eta, \mathsf{cnt})$. By the induction hypothesis, this return value is true if and only if $s \in [\![\psi_{i-1}]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}$ and, by the definition of HFL semantics, this is the case if and only if $s \in [\![\psi]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}$, which settles the claim of the lemma.

Case $\psi = \langle a \rangle \psi'$. There are two cases. If $A = \mathsf{N}$, then $\mathtt{McTr}$ guesses t with $s \xrightarrow{a} t$ and returns the value of $\mathtt{McTr}(A, t, \psi', \varepsilon, \eta, \mathsf{cnt})$. By the induction hypothesis, that value is true if and only if $t \in [\![\psi']\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}$ which, by the definition of HFL semantics, entails that $\mathtt{McTr}(\mathsf{N}, s, \psi, \varepsilon, \eta, \mathsf{cnt})$ returns true if and only if $s \in [\![\psi]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}$.

If $A = \mathsf{U}$, the algorithm iterates through all t with $s \xrightarrow{a} t$ and returns true if and only if the call $\mathtt{McTr}(A, t, \psi', \varepsilon, \eta, \emptyset)$ returns true for some t. If this is the case, then by the induction hypothesis, $t \in [\![\psi']\!]^{\mathcal{T}}_{\eta^{\emptyset}}$ and, by the definition of HFL semantics, also $s \in [\![\psi]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}$. Hence, $\mathtt{McTr}(\mathsf{U}, s, \psi, \varepsilon, \eta, \mathsf{cnt})$ returns true if and only if $s \in [\![\psi]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}$. Note that necessarily $\mathsf{info}(\psi') = (\emptyset, \_, \_)$ and, hence, that $rd(\psi') < rd(\psi)$, so the condition on non-tail-recursive calls is satisfied.

Case $\psi = [a]\psi'$ and **case** $\psi = \psi_1 \wedge \psi_2$. These are analogous to the two previous ones.

Case $\psi = \neg\psi'$. Correctness of the algorithm in this case is straight-forward. Note that necessarily $\mathsf{info}(\psi') = (\emptyset, \_, \_)$ and, hence $rd(\psi') < rd(\psi)$ whence the condition on non-tail-recursive calls is satisfied.

Case $\psi = \psi_1\, \psi_2$. Let $(\emptyset, \_, A_2) = \mathsf{info}(\psi_2)$. If $A_2 = \mathsf{L}$, then the algorithm calls a conventional model checker to determine $f = [\![\psi_2]\!]^{\mathcal{T}}_{\eta^{\emptyset}}$.

If $A_2 \neq \mathsf{L}$, let $\tau_1 \to \cdots \to \tau_n \to \bullet$ be the type of ψ_2. Let $(\emptyset, \_, A_2) = \mathsf{info}(\psi_2)$. The algorithm then computes, for each $g_1, \ldots, g_n$ in $[\![\tau_1]\!]^{\mathcal{T}} \times \cdots \times [\![\tau_n]\!]^{\mathcal{T}}$, and each state t, whether $\mathtt{McTr}(A_2, t, \psi_2, (g_1, \ldots, g_n), \eta, \emptyset)$ returns true. Since necessarily $rd(\psi_2) < rd(\psi)$, the criterion on non-tail-recursive calls is satisfied and, by the induction hypothesis this call returns true if and only if $t \in [\![\psi_2]\!]^{\mathcal{T}}_{\eta^{\emptyset}}$. Hence, $T = \{t \mid \mathtt{McTr}(A_2, t, \psi_2, (g_1, \ldots, g_k), \eta, \emptyset) = \mathsf{true}\}$ is equal to $(\cdots([\![\psi_2]\!]^{\mathcal{T}}_{\eta^{\emptyset}}\, g_1) \cdots\, g_n)$, and updating f to map $g_1, \ldots, g_n$ to T yields that $(\cdots(f\, g_1) \cdots\, g_n) = (\cdots([\![\psi_2]\!]^{\mathcal{T}}_{\eta^{\emptyset}}\, g_1) \cdots\, g_n)$. By repeating this process for all $g_1, \ldots, g_n \in [\![\tau_1]\!]^{\mathcal{T}} \times \cdots \times [\![\tau_n]\!]^{\mathcal{T}}$, we obtain that $f = [\![\psi_2]\!]^{\mathcal{T}}_{\eta^{\emptyset}}$.

In either case, the algorithm returns the value of $\mathtt{McTr}(A, s, \psi_1, (f, f_1, \ldots, f_k), \eta, \mathsf{cnt})$, which by the induction hypothesis is true if and only if $s \in (\cdots(([\![\psi_1]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}\, f)\, f_1) \cdots\, f_k)$, and the latter holds if and only if $s \in (\cdots([\![\psi]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}\, f_1) \cdots)\, f_k)$. The claim of the lemma follows.

Case $\psi = \sigma(X\colon \tau).\, \psi'$. The algorithm returns what $\mathtt{McTr}(A, s, \psi', (f_1, \ldots, f_k), \eta, \mathsf{cnt}[X \mapsto \mathsf{ht}(\tau)])$ returns. The latter returns true if and only if $s \in (\cdots([\![\psi']\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}[X \mapsto \mathsf{ht}(\tau)]}}\, f_1) \cdots\, f_k)$ by the induction hypothesis. By Lemma 3.4, this is equivalent to $s \in (\cdots([\![\sigma(X\colon \tau).\, \psi']\!]^{\mathcal{T}}_{\mathsf{cnt}}\, f_1)\, \cdots\, f_k)$.

Case $\psi = X$. If $\mathsf{cnt}(X) = 0$ then note that $[\![X]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}} = X^{\mathsf{cnt}(X)} = X^0$. If $\sigma_X = \mu$ then $X^0 = \bot^\tau$ where τ is the type of X. Hence, $s \notin (\cdots([\![\bot^\tau]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}\, f_1) \cdots\, f_k)$, and the algorithm correctly returns false. The case for $\sigma_X = \nu$ is analogous.

If $\mathsf{cnt}(X) \neq 0$, then the algorithm returns the value of $\mathtt{McTr}(A, s, \mathsf{fp}(X), \eta, \mathsf{cnt}[X\texttt{--}])$. Note that $[\![X]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}} = \eta^{\mathsf{cnt}}(X) = [\![X^{\mathsf{cnt}(X)}]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}} = [\![\mathsf{fp}(X)[X^{\mathsf{cnt}(X)-1}/X]]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}$. However, since X does not appear freely in $\mathsf{fp}(X)[X^{\mathsf{cnt}(X)-1}/X]$, we can replace cnt by $\mathsf{cnt}[X\texttt{--}]$ without altering semantics, i.e., $[\![\mathcal{T}]\!]^{\eta^{\mathsf{cnt}}}_{=}[\![\mathsf{fp}(X)[X^{\mathsf{cnt}(X)-1}/X]]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}$. Finally, since $X^{\mathsf{cnt}(X)-1} = \mathsf{cnt}[X\texttt{--}]$, the latter is equivalent to $[\![\mathsf{fp}(X)]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}[X\texttt{--}]}}$. It follows that $s \in (\cdots([\![X]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}\, f_1)\cdots\, f_k)$ if and only if we have that $s \in (\cdots([\![\mathsf{fp}(X)]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}[X\texttt{--}]}}\, f_1)\cdots\, f_k)$. Since $\mathsf{cnt}[X\texttt{--}] < \mathsf{cnt}$, by the induction hypothesis this is true if and only if $\texttt{McTr}(A, s, \mathsf{fp}(X), (f_1, \cdots, f_k), \eta, \mathsf{cnt}[X\texttt{--}])$ returns true. It follows that if $\mathsf{cnt}(X) \neq 0$, then $\texttt{McTr}(A, s, \psi, (f_1, \ldots, f_k), \eta, \mathsf{cnt})$ returns the same as $\texttt{McTr}(A, s, \mathsf{fp}(X), (f_1, \ldots, f_k), \eta, \mathsf{cnt}[X\texttt{--}])$.

□

Theorem 3.7. The model checking problem for $\mathrm{HFL}^{k+1}_{\mathsf{tail}}$ is in k-EXPSPACE.

Proof:
Calls to a conventional model checker are made for formulas that do not contain order-$(k+1)$ fixpoint definitions. While such formulas are not necessarily of order k or lower, their only order-$(k+1)$ elements are λ-abstractions. Using a strategy that evaluates arguments first, the subformula in question is essentially of order k. Hence, we can safely assume that any calls to a conventional model checker conclude in k-EXPTIME, well within the desired complexity bounds.

The information required to evaluate $\texttt{McTr}(A, s, \varphi, (f_1, \ldots, f_n), \eta, \mathsf{cnt})$ takes k-fold exponential space: references to a mode, a state and a subformula take logarithmic space; each of the function tables $f_1, \ldots, f_n$ 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 at most $|\mathcal{V}_{\mathsf{fp}}|$ 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, McTr 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 McTr as described above, with an added logarithmically sized counter in case of $[a]\varphi$ and $\langle a\rangle\varphi$) has to be preserved only at steps of the form $b \leftarrow \texttt{McTr}(\ldots)$, in which case the call goes to a subformula with strictly smaller recursion depth. Since the recursion depth of an order-$(k+1)$ tail-recursive 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, respectively universal k-fold exponential space. It follows that McTr runs in k-fold exponential space.

Regarding alternation, note that alternation occurs on two places: the first case are calls of the form $\texttt{McTr}(A, s, \varphi, (f_1, \ldots, f_k), \eta, \mathsf{cnt})$ where $\mathsf{info}(\varphi) = (\emptyset, A, A')$ with $A' \neq A$, and the second case are non-tail-recursive calls. Since the first kind of alternation necessarily occurs in subformulas that are fixpoint variable closed, and the latter occurs only for fixpoint-closed subformulas with strictly decreasing recursion depth, the maximum nesting depth of alternation is bounded by the size of the input formula. Hence, Theorem 4.2 from [15] (a generalisation of Savitch's Theorem attributed to Borodin) allows us to conclude that McTr can be simulated in deterministic k-EXPSPACE. □

4. Lower bounds in the exponential space hierarchy

For tail-recursive formulas of order 1 a matching lower bound of PSPACE is easily obtained.

Theorem 4.1. The model checking problem of $\mathrm{HFL}^1_{\mathsf{tail}}$ is PSPACE-hard in data complexity.

Proof:
This follows immediately from Example 2.4 showing that NFA universality, known to be PSPACE-complete, can be expressed in $\mathrm{HFL}^1_{\mathsf{tail}}$. I.e. there is a fixed $\mathrm{HFL}^1_{\mathsf{tail}}$ formula whose set of models (representing NFA as LTS in the most straight-forward way) is PSPACE-hard. □

There is no straight-forward extension of this construction to higher orders and the complexity classes of k-EXPSPACE; hence, we prove lower bounds by different reductions. A typical k-EXPSPACE-complete problem is the order-k corridor tiling problem [16]: a tiling system is of the form $\mathcal{K} = (T, H, V, t_I, t_\square, t_F)$ 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 $t_I, t_\square, t_F \in T$ 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 $\rho_0, \ldots, \rho_{m-1}$ of words over the alphabet T, with $|\rho_i| = 2^n_k$ for all $i \in \{0, \ldots, m-1\}$, such that the following four conditions hold. We write $\rho(j)$ for the j-th letter of the word ρ, beginning with $j = 0$.

- $\rho_0 = t_I t_\square \ldots t_\square$
- For each $i = 0, \ldots, m-1$ and $j = 0, \ldots, 2^n_k - 2$ we have $(\rho_i(j), \rho_i(j+1)) \in H$.
- For each $i = 0, \ldots, m-2$ and $j = 0, \ldots, 2^n_k - 1$ 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 4.2. ([16])
For each $k \geq 0$, the order-k corridor tiling problem is k-EXPSPACE-hard.

In the following, we show that the model checking problem for $\mathrm{HFL}^{k+1}_{\mathsf{tail}}$ is k-EXPSPACE-hard in data complexity for $k \geq 0$. More precisely, we devise a formula $\varphi_k \in \mathrm{HFL}^{k+1}_{\mathsf{tail}}$ and, for any given instance $(\mathcal{K}, n)$ of the order-k tiling problem, an LTS $\mathcal{T}_{\mathcal{K},n}$ of size $\mathcal{O}(|\mathcal{K}| + n)$ such that

$$\mathcal{T}_{\mathcal{K},n} \models \varphi_k \qquad \text{iff} \qquad \text{the order-}k\text{ tiling problem on input } \mathcal{K}, n \text{ has a solution.}$$

Note that φ_k will not depend on $\mathcal{K}$ or n but only on k. Moreover, φ_k only works for $k \geq 1$, which is why Theorem 4.1 needed to be stated separately.

Of course, an LTS of size n does not contain enough states to encode a tiling problem of width 2^n_k via objects of low order. The idea of the reduction is to encode a solution of the order-k corridor tiling problem as a sequence of functions of domain $0, \ldots, 2^n_k - 1$ that map number i to the ith tile in the row, which is encoded as a singleton set in the LTS. The domain of such a function can be identified with a suitable space of order-$(k-1)$ functions, whence the encoding of a row is then an order-k function. The latter can then be iterated over in tail-recursive HFL of order $k+1$ using a generalised version of binary incrementation. The first step of the reduction is to fix the general shape of the transition systems witnessing k-EXPSPACE-hardness of $\mathrm{HFL}^{k+1}_{\mathsf{tail}}$.

Fix a tiling system $\mathcal{K} = (T, H, V, t_I, t_\square, t_F)$ 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 $t_0 = t_I$, $t_{|T|-2} = t_\square$, and $t_{|T|-1} = t_F$.

We define the transition system $\mathcal{T}_{\mathcal{K},n} = (\mathcal{S}, \{\xrightarrow{a}\}_{a\in\mathcal{A}}, \ell)$ via $\mathcal{S} = \{0, \dots, n-1\}$ and $\mathcal{A} = \{\mathsf{h}, \mathsf{v}, \mathsf{e}, \mathsf{u}, \mathsf{d}\}$ with

$$\begin{aligned}
\xrightarrow{\mathsf{h}} &= \{(i,j) \mid (t_i, t_j) \in H\} && \text{(for ``horizontal''),}\\
\xrightarrow{\mathsf{v}} &= \{(i,j) \mid (t_i, t_j) \in V\} && \text{(for ``vertical''),}\\
\xrightarrow{\mathsf{e}} &= \{0, \dots, n-1\} \times \{0, \dots, n-1\} && \text{(for ``everywhere''),}\\
\xrightarrow{\mathsf{u}} &= \{(i,j) \mid 0 \leq i < j \leq n-1\} && \text{(for ``up'') and}\\
\xrightarrow{\mathsf{d}} &= \{(i,j) \mid 0 \leq j < i \leq n-1\} && \text{(for ``down'').}
\end{aligned}$$

The state labelling is given by $\ell(0) = \{p_I\}$, $\ell(|T|-2) = \{p_\square\}$, and $\ell(|T|-1) = \{p_F\}$.

The states of this transition system play two roles. On the one hand, they encode the different tiles of the tiling problem $\mathcal{K}$, with the special tiles $t_I, t_\square, t_F$ identified by propositional labeling, while the rest remain anonymous. Since $n \geq |T|$, there are enough states for this. The horizontal and vertical matching relations are encoded by the accessibility relations h and v, respectively. On the other hand, the states are used as the digits in a representation of large numbers (remember that n is a parameter to the tiling problem). 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. This allows some sort of quantification over the respective groups of digits, for example, $[\mathsf{d}]X$ holds at all states such that all states representing digits of lower significance are in X.

In order to define φ_k, we need to encode the rows of a tiling as functions of order k. Such a row can be seen as a function associating to each column number a given tile, therefore a given state. In order to achieve an order of k for this function, we therefore seek a representation of any column number in $\{0, \dots, 2^n_k - 1\}$ as a function of order $k-1$; we achieve this by means of a functional encoding of large numbers popularised by Jones [17].

4.1. Functional encoding of large numbers

Let $\top_{\mathcal{S}} = [\![\top]\!]^{\mathcal{T}_{\mathcal{K},n}} = \{0, \dots, n-1\}$ and $\bot_{\mathcal{S}} = [\![\bot]\!]^{\mathcal{T}_{\mathcal{K},n}} = \emptyset$. Let $\tau_0 = \bullet$ and $\tau_{k+1} = \tau_k \to \bullet$ for all $k \geq 0$. For all $k \geq 0$ and $i \in \{0, \dots, 2^n_{k+1} - 1\}$, let

$$\mathsf{jones}_k \colon [\![\tau_k]\!]^{\mathcal{T}_{\mathcal{K},n}} \to \{0, \dots, 2^n_{k+1} - 1\}$$

be the map defined as follows.

- For $S \subseteq \mathcal{S} = \{0, \dots, n-1\}$, $\mathsf{jones}_0(S)$ is the number $m = \sum_{i\in S} 2^i$.

- Let $k \geq 0$ and $\mathcal{X} \in [\![\tau_{k+1}]\!]^{\mathcal{T}_{\mathcal{K},n}}$; the i-th bit of $\mathsf{jones}_{k+1}(\mathcal{X})$, where $i \in \{0, \ldots, 2^n_k - 1\}$, is set (respectively unset) if for all $\mathcal{Y} \in [\![\tau_k]\!]^{\mathcal{T}_{\mathcal{K},n}}$ such that $\mathsf{jones}_k(\mathcal{Y}) = i$, $\mathcal{X}(\mathcal{Y}) = \top_{\mathcal{S}}$ (respectively, for all $\mathcal{Y} \in [\![\tau_k]\!]^{\mathcal{T}_{\mathcal{K},n}}$ such that $\mathsf{jones}_k(\mathcal{Y}) = i$, $\mathcal{X}(\mathcal{Y}) = \bot_{\mathcal{S}}$); $\mathsf{jones}_k(\mathcal{X})$ is defined if for all i, its i-th bit is either set or unset; in that case, $\mathsf{jones}_k(\mathcal{X}) = \sum_{i \in S} 2^i$, where $S \subseteq \{0, \ldots, 2^n_k - 1\}$ is the set of bits that are set in $\mathcal{X}$.

Note that $\mathsf{jones}_k \colon [\![\tau_k]\!]^{\mathcal{T}_{\mathcal{K},n}} \to \{0, \ldots, 2^n_{k+1} - 1\}$ is a surjective, partial map.

Lemma 4.3. Consider the following formulas:

$$\begin{aligned}
\mathsf{ite} &= \lambda(b\colon \bullet), (x\colon \bullet), (y\colon \bullet).\ (b \wedge x) \vee (\neg b \wedge y) \\
\mathsf{zero}_0 &= \bot \\
\mathsf{zero}_{k+1} &= \lambda(i\colon \tau_k).\ \bot \\
\mathsf{gt}_0 &= \lambda(m_1, m_2\colon \tau_0).\ \langle \mathsf{e} \rangle \big(m_2 \wedge \neg m_1 \wedge [\mathsf{u}](m_1 \Rightarrow m_2)\big) \\
\mathsf{next}_0 &= \lambda(m\colon \bullet).\ \mathsf{ite}\ m\ (\langle \mathsf{d} \rangle \neg m)\ ([\mathsf{d}] m)
\end{aligned}$$

The following holds.

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. $\mathsf{jones}_k([\![\mathsf{zero}_k]\!]) = 0$
3. Assume $\mathsf{jones}_0(\eta(x_1)) = m_1$ and $\mathsf{jones}_0(\eta(x_2)) = m_2$. If $m_1 < m_2$, then $[\![\mathsf{gt}_0\ x_1\ x_2]\!]_\eta = \top_{\mathcal{S}}$, otherwise $[\![\mathsf{gt}_0\ x_1\ x_2]\!]_\eta = \bot_{\mathcal{S}}$.
4. Assume $\mathsf{jones}_k(\eta(x)) = m$. Then $\mathsf{jones}_0([\![\mathsf{next}_0\ x]\!]_\eta) = m + 1$ modulo 2^n.

Proof:
(1) and (2) are straightforward. For (3), check that 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 $\mathsf{jones}_0(m_2)$. For (4), the claim follows from the following observation: if m is any given number in $\{0, \ldots, 2^n_k - 1\}$ and $m' = m + 1$ modulo 2^n_k, if i is any bit position in the binary representation of m and m', and if b and b' are the bits at position i in m and m', then the following holds:

- when b is set, b' is set if and only if there is a bit of lesser significance that is not set in m,
- when b is not set, b' is set if and only if all lower bits are set in m. $\square$

In other words, we just devised a function next_0 that defines a successor function over the (encodings of) numbers in $\{0, \ldots, 2^n - 1\}$. We are now going to present a formula next_k by induction that defines such a successor function over the (encodings of) numbers in $\{0, \ldots, 2^n_{k+1} - 1\}$. For this, we will need to introduce a quantification over (encodings of) numbers, which would only make sense for some predicates.

Definition 4.4. Let $k \geq 0$ be fixed. A function $p\colon \tau_k \to \bullet$ is an *arithmetic predicate* if for all $m \in \{0, \ldots, 2^n_{k+1} - 1\}$, one of the two holds:

- either for all $\mathcal{X} \in [\![\tau_k]\!]^{\mathcal{T}_{\mathcal{K},n}}$ such that $\mathsf{jones}_k(\mathcal{X}) = m$, $[\![p\ x]\!]_{\eta[x\mapsto\mathcal{X}]} = \top_{\mathcal{S}}$;
- or for all $\mathcal{X} \in [\![\tau_k]\!]^{\mathcal{T}_{\mathcal{K},n}}$ such that $\mathsf{jones}_k(\mathcal{X}) = m$, $[\![p\ x]\!]_{\eta[x\mapsto\mathcal{X}]} = \bot_{\mathcal{S}}$.

For instance, $\lambda(m\colon \bullet).\mathsf{gt}_0\ \mathsf{zero}_0\ m$ is an arithmetic predicate. When p is an arithmetic predicate, we write $p(m)$ if $[\![p\ x]\!]_{\eta[x\mapsto\mathcal{X}]} = \top_{\mathcal{S}}$ for all $\mathcal{X}$ such that $\mathsf{jones}_k(\mathcal{X}) = m$.

In the following lemma, we devise a quantifier over (encodings of) numbers for arithmetic predicates, assuming that we have a successor function for these numbers at hand.

Lemma 4.5. Let $k \geq 0$ be fixed, and assume some fixed formula $\mathsf{next}_k : \tau_k \to \tau_k$ such that $\mathsf{jones}_k([\![\mathsf{next}_k\ x]\!]_\eta) = \mathsf{jones}_k(\eta(x)) + 1$ modulo 2^n_{k+1}. Consider the formulas

$$\begin{aligned}
\mathsf{exists}_k &= \lambda(p\colon \tau_{k+1}).\ \Big(\big(\mu(F\colon \tau_k \to \bullet)\ .\ \lambda(m\colon \tau_k)\ .\ (p\ m) \vee F\ (\mathsf{next}_k\ m)\big)\ \mathsf{zero}_k\Big)\\
\mathsf{forall}_k &= \lambda(p\colon \tau_{k+1}).\ \neg\,\mathsf{exists}_k\ (\neg p)
\end{aligned}$$

Let p be an arithmetic predicate. The following holds:

- $[\![\mathsf{exists}_k\ p]\!]_\eta = \top_{\mathcal{S}}$ if there exists $m \in \{0, \ldots, 2^n_{k+1} - 1\}$ such that $p(m)$; otherwise we have that $[\![\mathsf{exists}_k\ p]\!]_\eta = \bot_{\mathcal{S}}$.
- $[\![\mathsf{forall}_k\ p]\!]_\eta = \top_{\mathcal{S}}$ if for all $m \in \{0, \ldots, 2^n_{k+1} - 1\}$, $p(m)$; otherwise $[\![\mathsf{forall}_k\ p]\!]_\eta = \bot_{\mathcal{S}}$.

Proof:
By fixpoint unfolding, $\mathsf{exists}_k\ p$ is equivalent to the infinitary formula

$$\bigvee_{m\geq 0} \big(p\ (\underbrace{\mathsf{next}_k(\mathsf{next}_k(\ldots(\mathsf{next}_k}_{m \text{ times}}\ \mathsf{zero}_k)\ldots))))\big).$$

By hypothesis on next_k, $[\![\mathsf{next}_k^m\ \mathsf{zero}_k]\!]$ is some $\mathcal{X}$ such that $\mathsf{jones}_k(\mathcal{X}) = m$ modulo 2^n_{k+1}. Since p is arithmetic, $[\![p\ (\mathsf{next}_k^m\ \mathsf{zero}_k)]\!]$ is either $\top_{\mathcal{S}}$ or $\bot_{\mathcal{S}}$, and the former holds only if $p(m)$, which ends the proof. $\square$

We are now ready to define the successor function next_k by induction.

Lemma 4.6. Let gt_k and next_k be defined by mutual recursion as

$$\begin{aligned}
\mathsf{gt}_{k+1} &= \lambda(m_1, m_2\colon \tau_{k+1}).\ \mathsf{exists}_k\Big(\lambda(i\colon \tau_k).\ (m_2\ i) \wedge \neg(m_1\ i) \wedge\\
&\qquad \mathsf{forall}_k\big(\lambda(j\colon \tau_k).\ (\mathsf{gt_k}\ i\ j) \Rightarrow (m_1\ j) \Rightarrow (m_2\ j)\big)\Big)\\
\mathsf{next}_{k+1} &= \lambda(m\colon \tau_{k+1}, i\colon \tau_k).\ \mathsf{ite}\ (m\ i)\\
&\qquad \Big(\mathsf{exists}_k\big(\lambda(j_1\colon \tau_k).\ (\mathsf{gt}_k\ i\ j_1) \wedge \neg(m\ j_1)\big)\Big)\\
&\qquad \Big(\mathsf{forall}_k\big(\lambda(j_2\colon \tau_k).\ (\mathsf{gt}_k\ i\ j_2) \Rightarrow (m\ j_2)\big)\Big)
\end{aligned}$$

starting from gt_0 and next_0 as in Lemma 4.3. Then the following holds.

1. Assume $\mathsf{jones}_k(\eta(x_1)) = m_1$ and $\mathsf{jones}_k(\eta(x_2)) = m_2$. If $m_1 < m_2$, then $[\![\mathsf{gt}_k\ x_1\ x_2]\!]_\eta = \top_{\mathcal{S}}$, otherwise $[\![\mathsf{gt}_k\ x_1\ x_2]\!]_\eta = \bot_{\mathcal{S}}$.
2. Assume $\mathsf{jones}_k(\eta(x)) = m$. Then $\mathsf{jones}_k([\![\mathsf{next}_k\ x]\!]_\eta) = m + 1$ modulo 2^n_{k+1}.

Proof:
The proof is based on the same observations that were made for Lemma 4.3, except that the bit positions are now level-k 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, the quantification over all bit positions uses the functions forall_k and exists_k instead of the relation e. $\square$

4.2. Encoding the tiling problem

We are now ready to define the encoding of rows of width 2^n_k as functions in the space $[\![\tau_k]\!]^{\mathcal{T}_{\mathcal{K},n}}$. Remember that $\mathcal{S}$ contains a state j for each tile t_j in T. In the following we identify t_j with the state j. Let $\rho = \rho_0 \dots \rho_{2^n_k - 1} \in T^*$ be a row in the tiling problem of width 2^n_k for some $k \geq 1$. We represent ρ by the function $\mathsf{row}_k(\rho)$ that maps any order-$(k-1)$ encoding of a column number $i \in \{0, \dots, 2^n_k - 1\}$ to the singleton predicate $\{\rho_i\}$. In other words, an encoding of a row is an order-k function that consumes order-$(k-1)$-functions, which encode the numbers $0, \dots, 2^n_k - 1$, and maps each of them to a singleton representing a tile.

Consider then the following formulas.

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

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. The function horiz_k verifies that 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 $\mathsf{jones}_{k-1}(2^n_k - 1)$ (whence the value $\mathsf{isZero}_{k-1}(\mathsf{next}_{k-1}\ m)$ is $\top_{\mathcal{S}}$) or that there is an 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 4.7. 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 $\varphi_k =$

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

where $\mathsf{existsSucc}_k = \lambda(r_1\colon \tau_k, p\colon \tau_{k+1}).\, \mathsf{exists}_k\ \big(\lambda(r_2\colon \tau_k).\, (\mathsf{horiz}_k\ r_2) \wedge (\mathsf{vert}_k\ r_1\ r_2) \wedge (p\ r_2)\big)$. It 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 setting is the fixpoint P which generates new rows using $\mathsf{existsSucc}$ until one of them is a final row, or ad infinitum, if the tiling problem is unsolvable.

Theorem 4.8. The model checking problem of $\mathrm{HFL}^{k+1}_{\mathsf{tail}}$ is k-EXPSPACE-hard in data complexity for $k > 1$.

Proof:
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 $\mathsf{existsSucc}_k$. However, after β-reduction of $\mathsf{existsSucc}_k\ r_1\ P$ and then $\mathsf{exists}_k(\lambda r_2 \dots)$, we get a formula ψ_k equivalent to φ_k and of the form

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

where α, β do not contain the recursive variables P and F, hence this formula is tail-recursive.

With the observation that φ_k only depends on k but not on $\mathcal{K}$ or n we get the lower bound not only for the general model checking problem but for data complexity already. $\square$

The upper 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 4.9. For all $k \geq 0$, $\mathrm{HFL}^{k}_{\mathsf{tail}} \lneqq \mathrm{HFL}^{k+1}_{\mathsf{tail}}$.

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 [18].

Now let $k > 0$ and suppose that for every $\varphi \in \mathrm{HFL}^{k+1}_{\mathsf{tail}}$ there was 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. 4.8. According to Theorems 4.1 and 4.8, $\{(\mathcal{T}, s) \mid \mathcal{T}, s \models \varphi_{k+1}\}$ is a k-EXPSPACE-hard set. 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}. Then the set of its models could be decided in $(k-1)$-EXPSPACE according to Theorem. 3.7, which contradicts the space hierarchy theorem [12]. $\square$

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 $\mathrm{HFL}_{\mathsf{tail}}$.

While we have restricted ourselves to the monadic case in this paper, the results can be extended to the polyadic extension PHFL of HFL: the definitions generalise accordingly and the lower bounds presented in Section 4 carry over straight-forwardly by virtue of 1-ary PHFL being just ordinary HFL. The transfer of the upper bound can be achieved by reducing the model checking problem of polyadic HFL to the one for ordinary HFL via a product construction (cf. [19]).

Proposition 5.1. The model checking problem for PHFL^{k+1} is k-EXPSPACE-complete for $k \geq 0$.

References

[1] Viswanathan M, Viswanathan R. A Higher Order Modal Fixed Point Logic. In: CONCUR'04, volume 3170 of *LNCS*. Springer, 2004 pp. 512–528. doi:10.1007/978-3-540-28644-8_33.

[2] Kozen D. Results on the Propositional μ-calculus. *TCS*, 1983. **27**:333–354. doi:10.1007/BFb0012782.

[3] Janin D, Walukiewicz I. On the Expressive Completeness of the Propositional μ-Calculus with Respect to Monadic Second Order Logic. In: CONCUR. 1996 pp. 263–277. doi:10.1007/3-540-61604-7_60.

[4] Lange M, Somla R. Propositional Dynamic Logic of Context-Free Programs and Fixpoint Logic with Chop. *Information Processing Letters*, 2006. **100**(2):72–75. doi:10.1016/j.ipl.2006.04.019.

[5] Lange M. Temporal Logics Beyond Regularity, 2007. Habilitation thesis, University of Munich, BRICS research report RS-07-13.

[6] Axelsson R, Lange M, Somla R. The Complexity of Model Checking Higher-Order Fixpoint Logic. *Logical Methods in Computer Science*, 2007. **3**:1–33. doi:10.2168/LMCS-3(2:7)2007.

[7] Emerson EA. Uniform inevitability is tree automaton ineffable. *Information Processing Letters*, 1987. **24**(2):77–79. doi:10.1016/0020-0190(87)90097-4.

[8] Hartmanis J, Stearns RE. On the Computational Complexity of Algorithms. *Trans. AMS*, 1965. **117**:285–306. doi:10.1090/S0002-9947-1965-0170805-7.

[9] Andersen HR. A Polyadic Modal μ-Calculus. Technical Report ID-TR: 1994-195, Dept. of Computer Science, Technical University of Denmark, Copenhagen, 1994. doi:10.1.1.42.1859.

[10] Otto M. Bisimulation-invariant PTIME and higher-dimensional μ-calculus. *Theor. Comput. Sci.*, 1999. **224**(1-2):237–265. doi:10.1016/S0304-3975(98)00314-4.

[11] Lange M, Lozes É. 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*. Springer, 2014 pp. 90–103. doi:10.1007/978-3-662-44602-7.

[12] Stearns RE, Hartmanis J, Lewis II PM. Hierarchies of memory limited computations. In: Proc. 6th Ann. Symp. on Switching Circuit Theory and Logical Design. IEEE, 1965 pp. 179–190.

[13] Bruse F, Lange M, Lozes E. Space-Efficient Fragments of Higher-Order Fixpoint Logic. In: Proc. 11th Workshop on Reachability Problems, RP'17, volume 10506 of *LNCS*. Springer, 2017 pp. 26–41. doi:10.1007/978-3-319-67089-8_3.

[14] Savitch WJ. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 1970. **4**:177–192.

[15] Chandra AK, Kozen D, Stockmeyer LJ. Alternation. *J. ACM*, 1981. **28**(1):114–133. doi:10.1145/322234.322243.

[16] van Emde Boas P. The Convenience of Tilings. In: Sorbi A (ed.), Complexity, Logic, and Recursion Theory, volume 187 of *Lecture notes in pure and applied mathematics*, pp. 331–363. Marcel Dekker, Inc., 1997.

[17] Jones ND. The expressive power of higher-order types or, life without CONS. *Journal of Func. Prog.*, 2001. **11**(1):5–94. doi:10.1017/S0956796800003889.

[18] Axelsson R, Lange M. 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*. Springer, 2007 pp. 62–76. doi:10.1007/978-3-540-75560-9_7.

[19] Lange M, Lozes E. Model Checking the Higher-Dimensional Modal μ-Calculus. In: Proc. 8th Workshop on Fixpoints in Computer Science, FICS'12, volume 77 of *Electr. Proc. in Theor. Comp. Sc.* 2012 pp. 39–46. doi:10.4204/EPTCS.77.