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

Florian Bruse, Martin Lange, Etienne Lozes

Fundamenta Informaticae XX (2021) 1–29
DOI 10.3233/FI-2016-0000
IOS Press

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

Florian Bruse
University of Kassel, Germany

Martin Lange
University of Kassel, Germany

Etienne Lozes
University of Nice Sophia Antipolis, I3S, UMR 7271, CNRS, Nice, France

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

1. Introduction

Higher-Order Modal Fixpoint Logic (HFL) [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.

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 stan-

dard 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-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, i.e. HFL^0 is quite well understood, including examples of properties that are known not to be expressible in it. The aforementioned tree property of being balanced is such an example [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.

Said criterion has been developed for PHFL, a polyadic extension of HFL, in the context of descriptive complexity. Extending Otto's result showing that a polyadic version of the modal μ-calculus [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 complexity bounds [14].

The paper is organised as follows. In Sect. 2 we recall HFL and apply the concept of tail-recursion, originally developed for a polyadic extension, to this monadic logic. In Sect. 3 we present upper bounds; matching lower bounds are presented in Sect. 4. 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, $\xrightarrow{a}$ is a binary relation for each $a \in \mathcal{A}$ and $\ell\colon \mathcal{S} \to \mathfrak{P}(\mathcal{P})$ is a function assigning, to each state, the set of propositions that are satisfied in it. We write $s \xrightarrow{a} t$ to denote that $(s,t) \in \xrightarrow{a}$.

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

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

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

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

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

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

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

$$\mid X \mid \mu(X\colon \tau).\ \varphi \mid \nu(X\colon \tau).\ \varphi$$

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

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

A sequence of the form $X_1^{v_1}\colon \tau_1, \ldots, X_n^{v_n}\colon \tau_n, x_1^{v_1'}\colon \tau_1', \ldots, x_j^{v_j'}\colon \tau_j'$ where the X_i are fixpoint variables, the x_j are λ-variables, the τ_i, τ_j' are types and the v_i, v_j' are variances, is called a *context*. We assume that each fixpoint variable and each λ-variable occurs only once per context. The context $\overline{\Gamma}$ is obtained from Γ by replacing all typing hypotheses of the form $X^+\colon \tau$ by $X^-\colon \tau$ and vice versa, and doing the same for λ-variables. An HFL-formula φ has type τ in context Γ if $\Gamma \vdash \varphi\colon \tau$ can be derived via the typing rules in Fig. 1. A formula φ is *well-formed* if $\Gamma \vdash \varphi\colon \tau$ can be derived for some Γ and τ. Note that, while fixpoint variables may only be used in a monotonic fashion, contexts with fixpoint variables of negative variance are still necessary to type formulas of the form $\mu(X\colon \bullet).\ \neg\neg X$. In some examples, we may sometimes omit type and/or variance anotations.

Moreover, we also assume that in a well-formed formula φ, each fixpoint variable $X \in \mathcal{V}_{\mathsf{fp}}$ is bound at most once, i.e., there is at most one subformula of the form $\mu(X\colon \tau).\ \psi$ or $\nu(X\colon \tau).\ \psi$. Then there is a function $\mathsf{fp}\colon \mathcal{V}_{\mathsf{fp}} \to \mathsf{sub}(\varphi)$ such that $\mathsf{fp}(X)$ is the unique subformula $\sigma(X\colon \tau).\ \varphi'$ with $\sigma \in \{\mu, \nu\}$. Note that this also introduces a partial order $\succ$ on the fixpoint variables via $X \succ Y$ iff $\sigma Y.\psi$ is a subformula of $\mathsf{fp}(X)$.

Note that it is possible to order the fixpoints in such a formula as $X_1, \ldots, X_n$ such that $\mathsf{fp}(X_i) \notin \mathsf{sub}(\mathsf{fp}(X_j))$ for $j > i$.

Let φ be a formula, and let $\Gamma \vdash \psi{:}\tau$ be a judgement for a subformula of φ derived in the proof that φ is well-formed. Then the type $\mathsf{type}(\psi)$ of ψ is τ. 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 $\emptyset \vdash \varphi\colon \bullet$. With HFL^k we denote the set of all well-formed HFL formulas of ground type whose order is at most k. In particular, HFL^0 is the modal μ-calculus $\mathcal{L}_\mu$. The notion of order of a formula can straightforwardly be applied to formulas which are not of ground type $\bullet$. We will therefore also speak of the order of some arbitrary subformula of an HFL formula.

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

- $[\![\bullet]\!]^{\mathcal{T}}$ is the lattice $\mathfrak{P}(\mathcal{S})$ ordered by the inclusion relation $\subseteq$,

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

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

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

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

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

Figure 1. The HFL typing system

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

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

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

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

The notion of tail-recursion has been transposed to the framework of higher-order fixpoint logics, originally for a polyadic extension of HFL [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.

For this paper, we define tail recursion with respect to some type theoretic order k, which usually is the order of the formula. 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 vice versa,

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

Figure 2. Semantics of HFL

- negation is only allowed for fixpoint closed formulas
- subformulas that are fixpoint closed and do not contain fixpoint binders of order k are not restricted in the position of free fixpoint variables

Let S_1, S_2 be sets. We write $S_1 \leftarrow\!\emptyset\!\rightarrow S_2$ to denote that at least one of S_1 and S_2 is empty. Note that, in particular, $\emptyset \leftarrow\!\emptyset\!\rightarrow \emptyset$ holds.

Definition 2.1. An HFL formula φ of order at most k is *order k tail-recursive* if the statement $\mathsf{tail}^k(\varphi, \emptyset,)$ 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.

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 F indicate whether nondeterministic operators (N) or universal operators (U) can be used without restriction. For example, rules $(\vee)$ and $(\vee_U)$ 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 judgement 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 their is a derivation for them for at least one mode.

Finally, a derivation for a subformula in mode F means that it does not contain fixpoint binders of order k. Intuitively this means that 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

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

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

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

$$(\vee_U)\ \frac{\mathsf{tail}^k(\varphi_1,\mathcal{X}_1,U) \qquad \mathsf{tail}^k(\varphi_2,\mathcal{X}_2,U) \qquad \mathcal{X}_1 \leftarrow\!\!\emptyset\!\!\rightarrow \mathcal{X}_2}{\mathsf{tail}^k(\varphi_1 \vee \varphi_2,\mathcal{X}_1 \cup \mathcal{X}_2,U)}$$

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

$$(\wedge_N)\ \frac{\mathsf{tail}^k(\varphi_1,\mathcal{X}_1,N) \qquad \mathsf{tail}^k(\varphi_2,\mathcal{X}_2,N) \qquad \mathcal{X}_1 \leftarrow\!\!\emptyset\!\!\rightarrow \mathcal{X}_2}{\mathsf{tail}^k(\varphi_1 \wedge \varphi_2,\mathcal{X}_1 \cup \mathcal{X}_2,N)}$$

$$(\langle a\rangle)\ \frac{A \in \{N,F\} \qquad \mathsf{tail}^k(\varphi,\mathcal{X},A)}{\mathsf{tail}^k(\langle a\rangle\varphi,\mathcal{X},A)} \qquad (\langle a\rangle_U)\ \frac{\mathsf{tail}^k(\varphi,\emptyset,U)}{\mathsf{tail}^k(\langle a\rangle\varphi,\emptyset,U)}$$

$$([a])\ \frac{A \in \{U,F\} \qquad \mathsf{tail}^k(\varphi,\mathcal{X},A)}{\mathsf{tail}^k([a]\varphi,\mathcal{X},A)} \qquad ([a]_N)\ \frac{\mathsf{tail}^k(\varphi,\emptyset,N)}{\mathsf{tail}^k([a]\varphi,\emptyset,N)}$$

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

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

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

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

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

has a derivation for mode F. This includes fixpoint variables in operand position. However, note that a derivation for mode F is only useful if it ends in some subformula that is fixpoint variable closed.

Example 2.2. The HFL^1 formula

$$\big(\nu F.\lambda x.\lambda y.(x \Rightarrow y) \wedge (F\ \langle a\rangle x\ \langle b\rangle y)\big)\ \top\ \langle b\rangle\top$$

has been introduced for expressing a form of assume-guarantee property [1]. This formula is tail-recursive, as one can easily check.

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

In the next section, we will see that these properties and any other expressible in $\mathsf{HFL}^1_{\mathsf{tail}}$ can be checked in polynomial space, thus improving a known exponential time upper bound [6, 15].

Example 2.3. Consider reachability properties of the form "there is a maximal path labelled with a word from L" where $L \subseteq \Sigma^*$ is some formal language. For context-free languages the logic formalising such properties is Propositional Dynamic Logic of Context-Free Programs [16]. It can be model checked in polynomial time [17]. However, formal-language constrained reachability is not restricted to context-free languages only. Consider the reachability problem above for $L = \{a^n b^n c^n \mid n \geq 1\}$. It can be formalised by the HFL^2 formula

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

with type $x : \bullet$; $f, g, h : \tau_1 := \bullet^+ \to \bullet$ and $F : \tau_1^+ \to \tau_1^+ \to \tau_1^+ \to \bullet^+ \to \bullet$. Again, one can check that this formula is tail-recursive. Since it is of order 2, Thm. 3.7 yields that the corresponding reachability problem can be checked using exponential space.

At last, we give an example of a property that can be specified in the third-order fragment of HFL.

Example 2.4. Let $\mathcal{A} = \{a, b\}$ and $L = \{(ab^n)^n) \mid n \geq 0\}$. Consider the property "there is a path with labels in L that ends in a state satisfying p". This can be specified in 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 the following.

$$f, x, y : \underbrace{\bullet^+ \to \bullet}_{\tau_1}\ ,\quad g : \underbrace{\tau_1^+ \to \tau_1^+ \to \tau_1}_{\tau_2}\ ,\quad Z : \underbrace{\tau_1^+ \to \tau_2^+ \to \tau_1}_{\tau_3}$$

Note that the order of each type τ_i is i which means that Φ is indeed of order 3. It is not hard to check it is tail-recursive. Thus, $\Phi \in \mathsf{HFL}^3_{\mathsf{tail}}$.

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. This is shown in the following table, using the equivalence $\circ id \equiv \psi$ and the abbreviation $\zeta^i := \zeta \circ \ldots \circ \zeta$ (i times).

iteration i	first argument φ_i	second argument ψ_i
0	id	$\lambda x, y.id$
1	$\langle b \rangle$	$\lambda x, y.x \circ y$
2	$\langle b \rangle \circ \langle b \rangle$	$\lambda x, y.x \circ y \circ x \circ y$
$\vdots$	$\vdots$	$\vdots$
i	$\langle b \rangle^i$	$\lambda x, y.(x \circ y)^i$

Noting that the fixpoint unfolds to a large disjunction in which each disjunct is formed by the subformula $g\ \langle a \rangle\ f$ such that the formal parameters f and g of the i-th unfolding are given by the formulas φ_1 and ψ_1 in the table above, we get that

$$\Phi \equiv (\bigsqcup_{i \geq 0} \psi_i\ \langle a \rangle\ \varphi_i)\ p \equiv (\bigsqcup_{i \geq 0} (\lambda x, y.(x \circ y)^i)\ \langle a \rangle\ \langle b \rangle^i)\ p \equiv \bigvee_{i \geq 0} \underbrace{\langle a \rangle \langle b \rangle^i \ldots \langle a \rangle \langle b \rangle^i}_{i \text{ times}} p\,.$$

3. Upper Bounds in the Exponential Space Hierarchy

In order to reduce notational clutter, given some tuple where only some values are relevant in the given context, we suppress displaying the other values. For example, in the tuple $(a, b, \_)$ the third component of the tuple holds some value that is not relevant in the given context.

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 CHECKTR that collects all those modes such that a derivation in the respective mode is possible for the given subformula. This does not 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 $modes = \{N, U, F\}$. The procedure MODETC that is used in the definition of CHECKTR 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.

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

Proof:
Procedure CHECKTR 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 $A \in \{N, U, F\}$ is derivable if and only if MODETC(CHECKTR(φ, 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 ψ being of the form p, X, x or $\lambda x.\, \psi'$ are straight-forward to see.

Algorithm 1 Checking for order-k tail recursion

procedure CHECKTR(φ, k) ▷ Returns $(\mathcal{X}, modes)$
  switch φ **do**
    case $\varphi = p$ **return** $(\emptyset, \{N, U, F\})$
    case $\varphi = X$ **return** $(\{X\}, \{N, U, F\})$
    case $\varphi = x$ **return** $(\emptyset, \{N, U, F\})$
    case $\varphi = \varphi_1 \vee \varphi_2$
      $(\mathcal{X}_i, modes_i) \leftarrow$ MODETC(CHECKTR(φ_i, k)), $i \in \{1, 2\}$
      if $\mathcal{X}_1 \leftarrow\!\!\!\!\!\!\!\!\rightarrow \mathcal{X}_2$ **then return** $(\mathcal{X}_1 \cup \mathcal{X}_2, modes_1 \cap modes_2)$
      else return $(\mathcal{X}_1 \cup \mathcal{X}_2, (modes_1 \cap modes_2) \setminus \{U\})$
    case $\varphi = \varphi_1 \wedge \varphi_2$
      $(\mathcal{X}_i, modes_i) \leftarrow$ MODETC(CHECKTR(φ_i, k)), $i \in \{1, 2\}$
      if $\mathcal{X}_1 \leftarrow\!\!\!\!\!\!\!\!\rightarrow \mathcal{X}_2$ **then return** $(\mathcal{X}_1 \cup \mathcal{X}_2, modes_1 \cap modes_2)$
      else return $(\mathcal{X}_1 \cup \mathcal{X}_2, (modes_1 \cap modes_2) \setminus \{N\})$
    case $\varphi = \langle a \rangle \varphi'$
      $(\mathcal{X}, modes) \leftarrow$ MODETC(CHECKTR(φ', k))
      if $\mathcal{X} = \emptyset$ **then return** $(\mathcal{X}, modes)$
      else return $(\mathcal{X}, (modes \setminus \{U\})$
    case $\varphi = [a] \varphi'$
      $(\mathcal{X}, modes) \leftarrow$ MODETC(CHECKTR(φ', k))
      if $\mathcal{X} = \emptyset$ **then return** $(\mathcal{X}, modes)$
      else return $(\mathcal{X}, (modes \setminus \{N\})$
    case $\varphi = \neg \varphi'$
      $(\mathcal{X}, modes) \leftarrow$ MODETC(CHECKTR(φ', k))
      if $\mathcal{X} = \emptyset$ **then return** $(\mathcal{X}, modes)$
      else return $(\mathcal{X}, (modes \cap \{F\})$
    case $\varphi = \lambda x.\, \varphi'$ **return** CHECKTR(φ', k)
    case $\varphi = \varphi_1\, \varphi_2$
      $(\mathcal{X}_i, modes_i) \leftarrow$ MODETC(CHECKTR(φ_i, k)), $i \in \{1, 2\}$
      if $\mathcal{X}_2 = \emptyset$ and $modes_2 \neq \emptyset$ **then return** $(\mathcal{X}_1, modes_1)$
      else return $(\mathcal{X}_1 \cup \mathcal{X}_2, modes_1 \cap modes_2 \cap \{F\})$
    case $\varphi = \sigma(X{:}\tau).\, \varphi'$
      $(\mathcal{X}, modes) \leftarrow$ MODETC(CHECKTR(φ, k))
      if $ord(\tau) < k$ **then return** $(\mathcal{X}_1 \setminus \{X\}, modes_1)$
      else return $(\mathcal{X}_1 \setminus \{X\}, modes_1 \setminus \{F\})$
procedure MODETC($(\mathcal{X}, modes)$)
  if $\mathcal{X} = \emptyset$ and $modes \neq \emptyset$ **then return** $(\mathcal{X}, modes \cup \{N, U\})$
  else return $(\mathcal{X}, modes)$

Case $\psi = \psi_1 \vee \psi_2$. Let $(\mathcal{X}_i, modes_i)$ be the return value of MODETC(CHECKTR(φ_i, k)) for $i \in \{1, 2\}$. By the induction hypothesis, we have that if $A \in modes_i$, then $\mathsf{tail}^k(\varphi_i, \mathcal{X}_i, A)$ is derivable. Note that the only two applicable rules are $(\vee)$ and $(\vee_U)$. If $N \in modes_1$ and $N \in modes_2$, then $\mathsf{tail}^k(\varphi, \mathcal{X}_1 \cup \mathcal{X}_2, N)$ is derivable, and the same holds for F. In both cases, both return statements of CHECKTR(ψ, k) return $(\mathcal{X}_1 \cup \mathcal{X}_2, modes)$ with $N \in modes$, respectively $F \in modes$ since $modes_1 \cap modes_2$ contains N, respectively F.

Moreover, if at least one of the $\mathcal{X}_i$ is $\emptyset$, then rule $(\vee_U)$ is potentially applicable. If $\mathcal{X}_i$ is $\emptyset$ for both $i = 1$ and $i = 2$, then $modes_1 \cap modes_2$ contains U and rule $(\vee_U)$ allows to derive $\mathsf{tail}^k(\psi, \emptyset, U)$, and also CHECKTR$(\psi, k)$ returns $(\emptyset, modes)$ with $U \in modes$. If there is $j \in \{1, 2\}$ such that $\mathcal{X}_j = \emptyset$ but $\mathcal{X}_{1-j} \neq \emptyset$, then rule $(\vee_U)$ is applicable if and only if $modes_{1-j}$ contains U and $modes_j \neq \emptyset$. In this case rule (alter) implies $U \in modes_j$ and rule $(\vee_U)$ allows to derive $\mathsf{tail}^k(\psi, \mathcal{X}_1 \cup \mathcal{X}_2, U)$, whence CHECKTR$(\psi, k)$ returns $(\mathcal{X}_1 \cup \mathcal{X}_2, modes)$ with $U \in modes$.

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 MODETC(CHECKTR(φ_i, k)) for $i \in \{1, 2\}$. By the induction hypothesis, we have that if $A \in modes_i$, then $\mathsf{tail}^k(\varphi_i, \mathcal{X}_i, A)$ is derivable. Note that the only applicable rules are (app) and (app_F). 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 CHECKTR(ψ, 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, in particular, this contains the case that $modes_1 \cap modes_2$ contains F, in which case also rule (app_F) would be applicable to derive $\mathsf{tail}^k(\psi, \mathcal{X}_1, F)$.

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

Case $\psi = \sigma(X{:}\tau).\, \psi'$. Let $(\mathcal{X}, modes')$ be the return value of MODETC(CHECKTR(ψ', k)). By the induction hypothesis, we have that if $A \in modes'$ then $\mathsf{tail}^k(\psi', \mathcal{X}, A)$ is derivable. Note that only rules (fp) and (fp_F) are applicable. If $\mathsf{ord}(\tau) < k$ then $\mathsf{tail}^k(\psi, \mathcal{X} \setminus \{X\}, A)$ is derivable via rule (fp_F) if and only if $\mathsf{tail}^k(\psi', \mathcal{X}, A)$ is derivable. Procedure CHECKTR(ψ, 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 $A \neq F$. Procedure CHECKTR(ψ, k) mirrors this in its first return call by returning $(\mathcal{X} \setminus \{X\}, modes' \setminus \{F\})$.

By applying the result of the induction to (φ, k), we obtain that $\mathsf{tail}^k(\varphi, \emptyset, A)$ is derivable for $A \in \{N, U, F\}$ if and only if MODETC(CHECKTR(φ, k)) $= (\emptyset, modes)$ with $A \in modes$. For the complexity results, note that procedure CHECKTR does exactly one recursive call per subformula of φ and calls 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 time in $\mathcal{O}(|\varphi|)$. □

In order to extend the verification of tail recursion to a full proof, for each formula in the syntax *tree* of ψ we generate a triple $\mathsf{info}(\psi) = (\mathcal{X}, A, A')$ where MODETC(CHECKTR(ψ, k)) $= (\mathcal{X}, modes)$ and $A, A' \in modes$ or $A' = \epsilon$, which means A' is as of yet undetermined. The intended semantics of such a triple is that the derivation rule that connects ψ with its predecessor in the syntax tree of φ, if

it exists, uses the fact that $\mathsf{tail}^k(\psi, \mathcal{X}, A)$ is derivable and the rule that connects ψ with its successors uses the fact that $\mathsf{tail}^k(\psi, \mathcal{X}, A')$ is derivable. As a stipulation, $A \neq A'$ only if $\mathcal{X} = \emptyset$. This is a valid constraint since rule (alter) is only applicable if $\mathcal{X} = \emptyset$.

Begin the procedure with the tuple $\mathsf{info}(\psi) = (\mathcal{X}, A, \epsilon)$ such that $A \in \mathit{modes}$, where $(\mathcal{X}, \mathit{modes}) = \textsc{modetc}(\textsc{checktr}(\varphi, k))$. Given a tuple of the form $\mathsf{info}(\psi) = (\mathcal{X}, A, \epsilon)$, 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 $(\mathcal{X}, \mathit{modes}) = \textsc{modetc}(\textsc{checktr}(\psi, k))$ and let $(\mathcal{X}', \mathit{modes}') = \textsc{modetc}(\textsc{checktr}(\psi', k))$. 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 $(\neg)$, $(\neg_F)$, $(\langle a\rangle)$, $(\langle a\rangle_U)$, $([a])$, $([a]_N)$ and (λ), the premise $\mathsf{tail}^k(\psi', \mathcal{X}', A')$ allows to derive $\mathsf{tail}^k(\psi, \mathcal{X}, A')$. 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 $A \in \{N, U\} \cap \mathit{modes}'$ and $\mathcal{X} = \emptyset$ and rule (alter) allows to derive $\mathsf{tail}^k(\psi, \emptyset, A')$ 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 $\mathsf{info}(\psi') = (\mathcal{X}', A, \epsilon)$.

Case ψ is of the form $\sigma(X{:}\tau).\, \psi'$. Let $(\mathcal{X}, \mathit{modes}) = \textsc{modetc}(\textsc{checktr}(\psi, k))$ and let $(\mathcal{X}', \mathit{modes}') = \textsc{modetc}(\textsc{checktr}(\psi', k))$. 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 (fp_F) allow to derive $\mathsf{tail}^k(\psi, \mathcal{X}, A')$ 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 $\mathsf{info}(\psi') = (\mathcal{X}', A, \epsilon)$. 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), (fp_F) 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 $\textsc{modetc}(\textsc{checktr})$, allows to derive $\mathsf{tail}^k(\psi, \mathcal{X}, A')$ via rule (fp) or (fp_F). Since $\mathcal{X} = \emptyset$, rule (alter) allows to derive $\mathsf{tail}^k(\psi, \emptyset, A)$. Update $\mathsf{info}(\psi)$ to $(\mathcal{X}, A, A')$ and continue with $\mathsf{info}(\psi') = (\mathcal{X}', A', \epsilon)$.

The second case is that $\mathsf{ord}(\tau) = k$. Note that if $A' \in \{N, U\} \cap \mathit{modes}'$, then $A' \in \mathit{modes}$ since if $A' \in \mathit{modes}'$ then rules (fp) allows to derive $\mathsf{tail}^k(\psi, \mathcal{X}, A')$ 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 $\mathsf{info}(\psi') = (\mathcal{X}', A, \epsilon)$. 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 $A' \in \mathit{modes}' \setminus \{F\}$ 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 $\textsc{modetc}(\textsc{checktr})$, allows to derive $\mathsf{tail}^k(\psi, \mathcal{X}, A')$ via rule (fp). Since $\mathcal{X} = \emptyset$, rule (alter) allows to derive $\mathsf{tail}^k(\psi, \emptyset, A)$. Update $\mathsf{info}(\psi)$ to $(\mathcal{X}, A, A')$ and continue with $\mathsf{info}(\psi') = (\mathcal{X}', A', \epsilon)$.

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

If $A \neq F$, then there are two cases: If $\mathcal{X} = \emptyset$, then also $\mathcal{X}_i = \emptyset$ for $i = 1, 2$. By the same reasoning as in the case for negation, modal operators, etc., we have that A also in $modes_i$ for $i = 1, 2$, 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 $\mathsf{info}(\psi_i) = (\mathcal{X}_1, A, \epsilon)$ and $\mathsf{info}(\psi_2) = (\mathcal{X}, A, \epsilon)$. 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 $A \in modes_1$ and that $modes_2 \neq \emptyset$, 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 $A' \in modes$. Update $\mathsf{info}(\psi)$ to $(\mathcal{X}, A, A)$ and continue with $\mathsf{info}(\psi_1) = (\mathcal{X}_1, A, \epsilon)$ and $\mathsf{info}(\psi_2) = (\mathcal{X}_2, A', \epsilon)$.

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

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

If $A = U$, there are two cases. If $\mathcal{X} = \emptyset$, then $N \in modes$. Use rule (alter) to derive $\mathsf{tail}^k(\psi, \emptyset, U)$ from $\mathsf{tail}^k(\psi, \emptyset, N)$ and refer to the previous case. Update $\mathsf{info}(\psi)$ to $(\mathcal{X}, U, N)$ and and continue with $\mathsf{info}(\psi_i) = (\mathcal{X}_i, N, \epsilon)$ for $i = 1, 2$. If $\mathcal{X} \neq \emptyset$, then rule $\mathsf{tail}^k(\psi, \mathcal{X}, N)$ can not be the conclusion of rule (alter), whence the only rule with this conclusion must be rule $(\vee_U)$. It follows that there is i such that $\mathcal{X}_i = \emptyset$, and, moreover, $N \in modes_1$ and $N \in modes_2$, for otherwise $\mathsf{tail}^k(\psi, \mathcal{X}, U)$ would not be derivable. Update $\mathsf{info}(\psi)$ to $(\mathcal{X}, U, U)$ and continue with $\mathsf{info}(\psi_i) = (\mathcal{X}_i, U, \epsilon)$ for $i = 1, 2$.

Proceeding like this yields, for each subformula ψ in the formula tree of φ, a triple $\mathsf{info}(\psi) = (\mathcal{X}, A, A')$ such that A is the relevant mode when connecting ψ with its predecessor, if it exists, and A' is the relevant mode when connective ψ with its successors, if they exist. In particular, at least one rule is applicable with the selected modes in either direction.

Remark 3.2. The 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 $\mu(X{:}\bullet).\, (X \vee X) \wedge p$. A bottom-up approach would not know which mode to assign to the subformulas X, and a top down approach would not know whether the conjunction should be assigned mode U, or whether advantage should be taken of the fact that the right conjunct is

fixpoint free whence mode N is also possible. Our approach solves this by first running Algorithm 1. It generates all possible partial derivations bottom-up, even those that cannot be continued to a derivation for the full formula. Then it uses a top-down local approach to extract the exact derivation steps at each subformula, aided by information already collected by Algorithm 1 on which rules modes for the involved subformulas can actually be completed to a successful derivation.

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 a counter: Upon reaching a fixpoint definition, a counter is added 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 termines with the default value of true or false, depending on the polarity of the 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 $\sigma_X.\,\mathsf{fp}(X)$ be the defining formula for X.

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

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

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

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

$$\eta^{\mathsf{cnt}}(x) := \eta(x) \ , \quad \eta^{\mathsf{cnt}}(X) := [\![X^{\mathsf{cnt}(X)}]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}} \text{ if } X \in \mathcal{X}' \ .$$

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 $\mathsf{cnt}'\colon \mathcal{X}' \to \mathbb{N}$ such that $\mathcal{X} \subseteq \mathcal{X}'$, 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 $\mathsf{cnt}[X{-}{-}]$ as

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

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, and let $\psi = \sigma(X{:}\tau).\,\psi'$ be a subformula of φ such that $\mathsf{info}(\psi) = (\mathcal{X}, \_, \_)$. Let $\mathsf{cnt}\colon \mathcal{X}' \to \mathbb{N}$ be such that $\mathcal{X} \subseteq \mathcal{X}'$. Then $[\![\sigma(X{:}\tau.\,\psi']\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}} = [\![\psi']\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}[X\mapsto \mathsf{ht}(\tau)]}}$,

It is known that over a finite LTS $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a}\}, \ell)$, ψ is equivalent to X^m, where m is the height $\mathsf{ht}(\tau)$ of the lattice of τ. Generally, $\mathsf{ht}(\tau)$ is k-fold exponential in the size of $|\mathcal{S}|$ for $k = ord(\tau)$ [6].

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

procedure MCTR($A, s, \psi, (f_1, \ldots, f_k), \eta, \mathsf{cnt}$) ▷ Inputs $\mathcal{T}$ and k not explicitely given
  $(\mathcal{X}, A'', A') \leftarrow \mathsf{info}(\psi)$
  if $A \neq A'$ **then return** $MCtr(A', s, \psi, (f_1, \ldots, f_k), \eta, \emptyset)$
  if $A = F$ **then**
    $f \leftarrow [\![\psi]\!]^{\mathcal{T}}_{\eta}$ ▷ 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 $A = N$ **then**
        guess $i \in \{1, 2\}$
        return $MCtr(A, s, \psi_i, (f_1, \ldots, f_k), \eta, \mathsf{cnt})$
      else if $A = U$ **then**
        $(\mathcal{X}_i, \_, A_i) \leftarrow \mathsf{info}(\psi_i), i \in \{1, 2\}$
        choose $i \in \{1, 2\}$ s.t. $\mathcal{X}_i = \emptyset$
        $b \leftarrow MCtr(A_i, s, \psi_i, (f_1, \ldots, f_k), \eta, \emptyset)$
        if $b = \mathsf{true}$ **then return** true
        else return $MCtr(A, s, \psi_{1-i}, (f_1, \ldots, f_k), \eta, \mathsf{cnt})$
    case $\psi = \psi_1 \wedge \psi_2$
      if $A = U$ **then**
        choose $i \in \{1, 2\}$
        return $MCtr(A, s, \psi_i, (f_1, \ldots, f_k), \eta, \mathsf{cnt})$
      else if $A = N$ **then**
        $(\mathcal{X}_i, \_, A_i) \leftarrow \mathsf{info}(\psi_i), i \in \{1, 2\}$
        guess $i \in \{1, 2\}$ s.t. $\mathcal{X}_i = \emptyset$
        $b \leftarrow MCtr(A_i, s, \psi_i, (f_1, \ldots, f_k), \eta, \emptyset)$
        if $b = \mathsf{false}$ **then return** false
        else return $MCtr(A, s, \psi_{1-i}, (f_1, \ldots, f_k), \eta, \mathsf{cnt})$
    case $\psi = \neg\psi'$
      $b \leftarrow checkTR(A, s, \psi', (f_1, \ldots, f_k), \eta, \emptyset)$
      if $b = \mathsf{true}$ **then return** false
      else return true

Algorithm 3 Efficient model-checking for order k tail recursive HFL formulas (cont.).

40: **case** $\psi = \langle a \rangle \psi'$
41: **if** $A = N$ **then**
42: **guess** t with $s \xrightarrow{a} t$
43: **return** $MCtr(A, t, \psi', (f_1, \ldots, f_k), \eta, \mathsf{cnt})$
44: **else if** $A = U$ **then**
45: **for** t with $s \xrightarrow{a} t$ **do**
46: $b \leftarrow MCtr(A, t, \psi', (f_1, \ldots, f_k), \eta, \emptyset)$
47: **if** $b = \mathsf{true}$ **then return** true
48: **return** false
49: **case** $\psi = [a] \psi'$
50: **if** $A = U$ **then**
51: **choose** t with $s \xrightarrow{a} t$
52: **return** $MCtr(A, t, \psi', (f_1, \ldots, f_k), \eta, \mathsf{cnt})$
53: **else if** $A = N$ **then**
54: **for** t with $s \xrightarrow{a} t$ **do**
55: $b \leftarrow MCtr(A, t, \psi', (f_1, \ldots, f_k), \eta, \emptyset)$
56: **if** $b = \mathsf{false}$ **then return** false
57: **return** true
58: **case** $\psi = \psi_1\, \psi_2$
59: $(\_, A'') \leftarrow \mathsf{info}(\psi_2)$
60: **if** $A_2' = F$ **then**
61: $f \leftarrow [\![\psi_2]\!]_\eta^{\mathcal{T}}$ ▷ use a conventional model checker
62: **return** $checkTR(A, s, \psi_1, (f, f_1, \ldots, f_k), \eta, \mathsf{cnt})$
63: **else**
64: $(\emptyset, \_, A_2) \leftarrow \mathsf{info}(\psi_2)$
65: $(\tau_1 \to \cdots \to \tau_n \to \bullet) \leftarrow \mathsf{type}(\psi_2)$
66: $f \leftarrow \emptyset$
67: **for** $g_1, \ldots, g_n \in [\![\tau_1]\!]^{\mathcal{T}} \times \cdots \times [\![\tau_n]\!]^{\mathcal{T}}$ **do**
68: $\mathcal{S}' \leftarrow \emptyset$
69: **for** $t \in \mathcal{S}$ **do**
70: $b \leftarrow checkTR(A_2, t, \psi_2, (g_1, \ldots, g_n), \eta, \emptyset)$
71: **if** $b = \mathsf{true}$ **then** $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{t\}$
72: $f \leftarrow f[g_1, \ldots, g_n \mapsto \mathcal{S}']$
73: **return** $checkTR(A, s, \psi_1, (f, f_1, \ldots, f_k), \eta, \mathsf{cnt})$
74: **case** $\psi = \lambda x.\, \psi'$ **return** $checkTR(A, s, (\psi_1, \mathcal{X}_1, A_1'), (f_2, \ldots, f_k), \eta[x \mapsto f_1], \mathsf{cnt})$
75: **case** $\psi = \sigma(X{:}\tau).\, \psi'$ **return** $checkTR(A, s, \psi', (f_1, \ldots, f_k), \eta, \mathsf{cnt}[X \mapsto \mathsf{ht}(\tau)])$
76: **case** $\psi = X$
77: **if** $\mathsf{cnt}(X) \neq 0$ **then**
78: **return** $checkTR(A, s, (\mathsf{fp}\varphi X, (f_1, \ldots, f_k), \eta, \mathsf{cnt}[X{-}{-}])$
79: **else if** $\sigma_X = \mu$ **then return** $\bot$
80: **else return** $\top$

Note that a k-fold exponentially large number can be represented by $(k-1)$-fold exponentially many bits.

We claim that procedure $checkTR$ in Algorithm 2 encodes a valid model-checking procedure for order k rail recursive formulas. In order to give the correctness proof, we need a measure of the degree to which a formula diverges from a recursion in the relevant mode, e.g. by having mode N conjunctions and vice versa, negations, applications etc. This measures how many non tail-recursive calls are necessary during the algorithm.

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

- $rd(\psi) = 0$ if $\psi = p$ or $\psi = x$ or $\psi = X$
- $rd(\psi) = rd(\psi')$ if $\psi = \langle a \rangle \psi'$ and $\mathsf{info}(\psi) = (\_, \_, N)$ or if $\psi = [a]\psi'$ and $\mathsf{info}(\psi) = (\_, \_, U)$ or if $\psi = \sigma X.\, \psi'$ or if $\psi = \lambda x.\psi'$
- $rd(\psi) = 1 + rd(\psi')$ if $\psi = \langle a \rangle \psi'$ and $\mathsf{info}(\psi) = (\_, \_, U)$ or if $\psi = [a]\psi'$ and $\mathsf{info}(\psi) = (\_, \_, N)$ or if $\psi = \neg\psi'$
- $rd(\psi) = \max\{r_1, r_2\}$ if $\psi = \psi_1 \vee \psi_2$ or if $\psi = \psi_1 \wedge \psi_2$

  $$r_i = \begin{cases} rd(\psi_i) \text{ if } \mathsf{info}(\psi_i) = (\mathcal{X}, \_, \_) \text{ and } \mathcal{X} \neq \emptyset \\ 1 + rd(\psi_i) \text{ if } \mathsf{info}(\psi_i) = (\emptyset, \_, \_) \end{cases}$$

  for $i \in \{1, 2\}$.
- $rd(\psi) = \max\{rd(\psi_1), 1 + rd(\psi_2)\}$ if $\psi = \psi_1\, \psi_2$.

We are now ready to give the correctness proof.

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 $checkTR(A, s_I, \varphi, \epsilon, \emptyset, \emptyset)$ terminates and returns $\top$ if and only if $\mathcal{T}, s_I \models \varphi$.

Proof:
We are now going to prove by induction the following statement: 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 $\mathsf{cnt}\colon \mathcal{X}' \to \mathbb{N}$ with $\mathcal{X} \subseteq \mathcal{X}'$ and $f_1, \ldots, f_n$ with $f_i \in [\![\tau_i]\!]^{\mathcal{T}}$ for $1 \leq i \leq n$ and $A \in \{N, U, F\}$ then

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

and terminates.

The statement of the lemma then follows with $\psi = \varphi$, $s = s_I$, $n = 0$, $\eta = \emptyset$ and $\mathsf{cnt} = \emptyset$.

The induction has four induction parameters: rd, ψ, cnt and A. If $checkTR(A', \_, \psi', \_, \_, \mathsf{cnt}')$ is called tail-recursively during evaluation of $checkTR(A, \_, \psi, \_, \_, \mathsf{cnt})$, i.e., in the form **return** $checkTR(\ldots)$, 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 $checkTR(A', \_, \psi', \_, \_, \mathsf{cnt}')$ is not tail recursive in an algorithmic sense, i.e., if it is of the form $b \leftarrow checkTR(\dots)$ then $rd(\psi') < rd(\psi)$.

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

If $A \neq F$, 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 $A = N$, then $checkTR$ guesses $i \in \{1, 2\}$ and returns the value of $checkTR(A, s, \psi_i, \epsilon, \eta, \mathsf{cnt})$. Hence, $checkTR$ returns true if and only if there is i such that $checkTR(A, s, \psi_i, \epsilon, \eta, \mathsf{cnt})$ returns 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 $checkTR(N, s, \psi, \epsilon, \eta, \mathsf{cnt})$ returns true if and only if $s \in [\![\psi]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}$. If $A = U$, then $checkTR$ universally chooses $i \in \{1, 2\}$ such that $\mathsf{info}(\psi_i) = (\mathcal{X}_i, \_, A_i)$ and $\mathcal{X}_i = \emptyset$ and calculates $b = checkTR(A_i, s, \psi_i, \epsilon, \eta, \emptyset)$. By the induction hypothesis, $b = \mathsf{true}$ 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 $b = \mathsf{true}$, 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 $checkTR(A, s, \psi_{1-i}, \epsilon, \eta, \mathsf{cnt})$. 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 $\psi = \psi_1 \wedge \psi_2$**.** Again, this is analogous to the previous case.

Case $\psi = \langle a \rangle \psi'$**.** There are two cases. If $A = N$, then $checkTR$ guesses t with $s \xrightarrow{a} t$ and returns the value of $checkTR(A, t, \psi', \epsilon, \eta, \mathsf{cnt})$. 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 $checkTR(N, s, \psi, \epsilon, \eta, \mathsf{cnt})$ returns true if and only if $s \in [\![\psi]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}$.

If $A = U$, the algorithm iterates through all t with $s \xrightarrow{a} t$ and returns true if and only if $checkTR(A, t, \psi', \epsilon, \eta,$ returns true. If this is the case for some such t, 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, $checkTR(U, s, \psi, \epsilon, \eta, \mathsf{cnt})$ 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 $\psi = [a]\psi'$**.** Again, this is analogous to the previous case.

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 $A_2 = F$, then the algorithm calls a conventional model-checker to determine $f = [\![\psi_2]\!]^{\mathcal{T}}_{\eta^\emptyset}$.

If $A_2 \neq F$, 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 $checkTR(A_2, t, \psi_2, (g_1, \ldots, g_n), \eta, \emptyset)$ 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, $T = \{t \mid checkTR(A_2, t, \psi_2, (g_1, \ldots, g_k), \eta, \emptyset) = \mathsf{true}\}$ 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 $checkTR(A, s, \psi_1, (f, f_1, \ldots, f_k), \eta, \mathsf{cnt})$, which by the induction hypothesis is true if and $s \in (\cdots((([\![\psi_2]\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}}}\, f)\, f_1)\cdots\, f_k)$, 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 $\psi = \sigma(X{:}\tau).\,\psi'$**.** The algorithm returns the value of $checkTR(A, s, \psi', (f_1, \ldots, f_k), \eta, \mathsf{cnt}[X \mapsto \mathsf{ht}(\tau)])$. By the induction hypothesis, this value is true if and only if $s \in (\cdots([\![\psi']\!]^{\mathcal{T}}_{\eta^{\mathsf{cnt}[X\mapsto\mathsf{ht}(\tau)]}}\, f_1)\cdots\, f_k)$, which, by Lemma3.4 is equivalent to $s \in (\cdots([\![\sigma(X{:}\tau).\,\psi']\!]^{\mathcal{T}}_{\mathsf{cnt}}$.

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.

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

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 lambda abstractions. Using an argument first model-checking strategy, 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 $checkTR(A, s, \varphi, (f_1, \ldots, f_n), \eta, \mathsf{cnt})$ takes k-fold exponential space: references to a mode, a state and a subformula take linear 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, **check** operates in a tail-recursive fashion for most operators, which means that no stack has to be maintained and the space needed is restricted to what is described in the previous paragraph. A calling context (which is just an instance of **check** as described above, with an added logarithmically sized counter in case of $[a]\varphi$ and $\langle a\rangle\varphi$) has to be preserved only at steps of the form $b \leftarrow checkTR(\dots)$, in which case the call goes to a subformula with strictly smaller recursion depth. Since the recursion depth of an $\mathrm{HFL}^k_{\mathsf{tail}}$-formula is linear in the size of the formula, only linearly many such calling contexts have to be stored at any given point during the evaluation, which does not exceed nondeterministic k-fold exponential space. It follows that $checkTR$ runs in k-fold exponential space.

Regarding alternation, note that alternation occurs on two places: If $checkTR(A, s, \varphi, (f_1, \dots, f_k), \eta, \mathsf{cnt})$ is called such that $\mathsf{info}(\varphi) = (\emptyset, A, A')$ with $A' \neq A$, and potentially for 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 [18] (a generalisation of Savitch's Theorem attributed to Borodin) allows us to conclude that $checkTR$ can be simulated in deterministic k-EXPSPACE. □

4. Lower Bounds in the Exponential Space Hierarchy

A typical k-EXPSPACE-complete problem (for $k \geq 1$) is the order-k corridor tiling problem [19]: 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 $\rho_0, \dots, \rho_{m-1}$ of words over the alphabet T, with $|\rho_i| = 2^n_k$ for all $i \in \{0, \dots, m-1\}$, and such that the following four conditions hold. We write $\rho(j)$ for the j-th letter of the word ρ, beginning with $j = 0$.

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

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

Proposition 4.1. ([19])
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

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

Fix a tiling system $\mathcal{K} = (T, H, V, t_I, t_F)$ and an $n \geq 1$. W.l.o.g. we assume $|T| \leq n$, and we fix an enumeration $T = \{t_0, \ldots, t_{|T|-1}\}$ 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 $\mathcal{T}_{\mathcal{K},n} = (\mathcal{S}, \{\xrightarrow{a}\}_{a \in \mathcal{A}}, \ell)$ as follows:

- $\mathcal{S} = \{0, \ldots, n-1\}$,
- $\mathcal{A} = \{\mathsf{h}, \mathsf{v}, \mathsf{e}, \mathsf{u}, \mathsf{d}\}$ with $\xrightarrow{\mathsf{h}} = \{(i,j) \mid (t_i, t_j) \in H\}$ (for "horizontal"), $\xrightarrow{\mathsf{v}} = \{(i,j) \mid (t_i, t_j) \in V\}$ (for "vertical"), $\xrightarrow{\mathsf{e}} = \{0, \ldots, n-1\} \times \{0, \ldots, n-1\}$ (for "everywhere"), $\xrightarrow{\mathsf{u}} = \{(i,j) \mid 0 \leq i < j \leq n-1\}$ (for "up") and $\xrightarrow{\mathsf{d}} = \{(i,j) \mid 0 \leq j < i \leq n-1\}$ (for "down").
- $\ell(0) = \{p_I\}$, $\ell(|T|-2) = \{p_\square\}$, 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. The horizontal and vertical matching relations are encoded by the accessibility relations h and v, respectively. On the other hand, the states of this transition systems are used as the digits in a represention of large numbers. The relation u connects a digit to all digits of higher significance, d connects to all digits of lower significance, and e is the global accessibility relation.

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 k for this function, we therefore seek a representation of any column number in $\{0, \ldots, 2^n_k - 1\}$ as a function of order $k-1$; we achieve this by means of a functional encoding of large numbers popularized by Jones [20].

4.1. Functional Encoding of Large Numbers

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

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

be the map defined as follows

- for $S \subseteq \mathcal{S} = \{0, \ldots, n-1\}$, $\mathsf{jones}_0(S)$ 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 $i \in \{0, \ldots, 2^n_{k+1} - 1\}$, 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 $\mathcal{Y} \in [\![\tau_{k-1}]\!]^{\mathcal{T}_{\mathcal{K},n}}$ 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 $S \subseteq \{0, \ldots, 2^n_{k+1} - 1\}$ is the set of bits that are set in $\mathcal{X}$.

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

Lemma 4.2. Consider the following formulas:

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

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: 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 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.

□

In other words, we just devised a function next_0 that defines a successor function over the (encondings of) numbers in $\{0, \ldots, 2^n - 1\}$. We are now going to define by induction a formula next_k 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 4.3. (arithmetic predicate)
Let $k \geq 0$ be fixed. A function $p : \tau_k \to \bullet$ 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, $\lambda(m : \bullet).\mathsf{gt}_0\ \mathsf{zero}_0\ m$ 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 at hand a successor function for these numbers.

Lemma 4.4. 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 $[\![\mathsf{exists}_k\ p]\!]_\eta = \bot_{\mathcal{S}}$.
- $[\![\mathsf{forall}_k\ p]\!]_\eta = \top_{\mathcal{S}}$ 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

$$\bigvee_{m \geq 0} \big(p\ (\mathsf{next}_k^m\ \mathsf{zero}_k)\big).$$

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. □

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

Lemma 4.5. Let gt_k and next_k be defined by the mutual recursive definition

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

starting from gt_0 and next_0 as in Lemma 4.2.

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 4.2, 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. □

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 sequel, we identify t_j with the state j. Let $\rho = \rho_0 \dots \rho_{2^n_k} \in T^*$ be a row 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 $\{\rho_i\}$.

Consider then the following formulas.

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

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

Lemma 4.6. The following hold:

1. $[\![\mathsf{isTile}\ x]\!]_\eta$ evaluates to $\top_{\mathcal{S}}$ if $\eta(x) = \{i\}$ for some $i \in \{0, \ldots, |T| - 1\}$, 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 $\mathsf{row}_k(t_I \cdot t_\square \cdots t_\square)$.
4. Assume $\eta(r) = \mathsf{row}_k(\rho)$ and $\eta(r') = \mathsf{row}_k(\rho')$ for some rows $\rho = \rho_0 \ldots \rho_{2^n_k}$ and $\rho' = \rho'_0 \ldots \rho'_{2^n_k}$. 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 $i \in \{0, \ldots, 2^n_k - 1\}$, 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 $i \in \{0, \ldots, 2^n_k - 1\}$, 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 $\varphi_k =$

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

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

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

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

Theorem 4.7. The model-checking problem of $\mathrm{HFL}^{k+1}_{\mathsf{tail}}$ is k-EXPSPACE-hard in data complexity for $k \geq 0$.

Proof:
We treat the case $k = 0$ separately (remember that the order k corridor problem is k-EXPSPACE complete for $k \geq 1$). For $k = 0$, we consider the HFL^1 formula

$$(\nu(F : \bullet \to \bullet).\lambda(X : \bullet).X \wedge \bigwedge_{a \in \Sigma} F\ \langle a \rangle X)\ p_{\mathsf{acc}}.$$

This formula is to be interpreted on a given NFA over alphabet Σ, seen as a LTS as follows: p_{acc} is true on all accepting states of the automaton, and $\langle a \rangle$ is the successor relation associated to the letter a. This formula evaluates to true on the initial state if and only if the NFA is universal (see also [15]). Moreover, It is easy to check that this formula tail-recursive. Since the universality problem for NFA is PSPACE-hard, i.e. 0-EXPSPACE-hard, it shows the claim for $k = 0$.

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

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

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

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

Corollary 4.8. For all $k \geq 0$, $\text{HFL}^k_{\mathsf{tail}} \lneq \text{HFL}^{k+1}_{\mathsf{tail}}$.

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

Let $k = 0$. Note that $\text{HFL}^0_{\mathsf{tail}}$ is a fragment of the modal μ-calculus which can only express regular properties. On the other hand, $\text{HFL}^1_{\mathsf{tail}}$ contains formulas that express non-regular properties, for instance uniform inevitability [15].

Now let $k > 0$ and suppose that for every $\varphi \in \text{HFL}^{k+1}_{\mathsf{tail}}$ there would exist a $\widehat{\varphi} \in \text{HFL}^k_{\mathsf{tail}}$ such that $\widehat{\varphi} \equiv \varphi$. Take the formula φ_{k+1} as constructed above and used in the proof of Thm. 4.7. Fix some function enc which represents a transition system and a state as a string over some suitable alphabet. According to Thm. 4.7, $L := \{enc(\mathcal{T}, s) \mid \mathcal{T}, s \models \varphi_{k+1}\}$ is a k-EXPSPACE-hard language.

On the other hand, consider $\widehat{\varphi_{k+1}}$ which, by assumption, belongs to $\text{HFL}^k_{\mathsf{tail}}$ and is equivalent to φ_{k+1}. Hence, $L = \{enc(\mathcal{T}, s) \mid \mathcal{T}, s \models \widehat{\varphi_{k+1}}\}$. According to Thm. 3.7, we have $L \in (k-1)$-EXPSPACE and therefore k-EXPSPACE $= (k-1)$-EXPSPACE which contradicts the space hierarchy theorem [12]. □

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

Tail recursion was originally proposed in the context of polyadic HFL [11]. While we have restricted ourselves to the monadic case in this paper, the results can be extended to PHFL: The definitions generalise accordingly and the lower bounds presented in Section 4 carry over straight-forwardly by virtue of 1-ary HFL 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. [21]).

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

Recall Example 2.4 showing how to specify the reachability property "there is a path of the form $(ab^n)^n$ for some $n \in \mathbb{N}$". From previous results [6] one can conclude that this property can be checked in triply exponential time. Since the HFL^3 formula given in this example is tail-recursive, one can conclude with Thm. 3.7 that it can even be checked in doubly exponential space. However, this is not optimal. With some combinatorial reasoning one can see that whenever a state satisfies this property then it must also satisfy it for some "small" n, namely one that is at most exponential in the number of states of the transition system.. One can then devise a simple nondeterministic procedure that uses at most polynomial space and checks this property. With Savitch's Theorem [14] we therefore get that it is already in PSPACE which is obviously considerably better than the bound derived from Thm. 3.7. Note, however, that the logical approach via Thm. 3.7 makes a stronger statement about the *combined complexity* of model checking $\mathrm{HFL}^k_{\mathsf{tail}}$, whereas the argument about the PSPACE bound obtained via combinatorial and simple algorithmic reasoning is concerned with the complexity of checking one particular such property only.

Still, similar considerations can be made for various other reachability properties of the form $\langle L \rangle p$ where L is some, preferably non-regular, formal language. In many cases it is possible to obtain better bounds by devising algorithms directly compared to those that are obtained by the logical, declarative approach using their specification in HFL. The fact that combined complexity cannot be better than complexity for each fixed formula does not quite explain this discrepancy in the obtained upper bounds. We suspect that, instead, the formulas specifiying these reachability problems fall into some fragments which have yet to be discovered formally and which have at least exponentially better model checking complexity. It is, for instance noteworthy that the formalisations of such reachability problems rarely make use of negative variances or combinations of $[-]$-operators with disjunctions. This observation could be the starting point for the identification of fragments within each HFL^k with even better model checking complexity than k-EXPTIME as the general case, resp. $(k-1)$-EXPSPACE as the tail-recursive fragment.

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.

[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.

[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.

[7] Emerson EA. Uniform inevitability is tree automaton ineffable. *Information Processing Letters*, 1987. **24**(2):77–79.

[8] Hartmanis J, Stearns RE. On the Computational Complexity of Algorithms. *Trans. AMS*, 1965. **117**:285–306.

[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] 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.

[16] Harel D, Pnueli A, Stavi J. Propositional Dynamic Logic of Nonregular Programs. *Journal of Computer and System Sciences*, 1983. **26**(2):222–243.

[17] Lange M. Model Checking Propositional Dynamic Logic with All Extras. *Journal of Applied Logic*, 2005. **4**(1):39–49.

[18] Chandra AK, Kozen D, Stockmeyer LJ. Alternation. *J. ACM*, 1981. **28**(1):114–133. doi:10.1145/322234.322243. URL `http://doi.acm.org/10.1145/322234.322243`.

[19] 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.

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

[21] 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.