The Complexity of Model Checking Higher Order Fixpoint Logic

Martin Lange[1] and Rafał Somla[2]

[1] Institut für Informatik, University of Munich, Germany
mlange@informatik.uni-muenchen.de
[2] IT department, Uppsala University, Sweden
rsomla@it.uu.se

Abstract. This paper analyzes the computational complexity of the model checking problem for Higher Order Fixpoint Logic – the modal μ-calculus enriched with a typed λ-calculus. It is hard for every level of the elementary time/space hierarchy and in elementary time/space when restricted to formulas of bounded type order.

1 Introduction

Temporal logics are well-established tools for the specification of correctness properties and their verification in hard- and software design processes. One of the most famous temporal logics is Kozen's modal μ-calculus $\mathcal{L}_\mu$ [16] which extends multi-modal logic with extremal fixpoint quantifiers. $\mathcal{L}_\mu$ subsumes many other temporal logics like CTL* [10], and with it CTL [9] and LTL [19], as well as PDL [12].

$\mathcal{L}_\mu$ is equi-expressive to the bisimulation-invariant fragment of Monadic Second Order Logic over trees or graphs [11,15]. Hence, properties expressed by formulas of the modal μ-calculus are only regular. There are, however, many interesting correctness properties of programs that are not regular. Examples include *uniform inevitability* [8] which states that a certain event occurs globally at the same time in all possible runs of the system; counting properties like "at any point in a run of a protocol there have never been more *send*- than *receive*-actions"; formulas saying that an unbounded number of data does not lose its order during a transmission process; etc.

When program verification was introduced to computer science, programs as well as their correctness properties were mainly specified in temporal logics. Hence, verification meant to check formulas of the form $\varphi \rightarrow \psi$ for validity, or equally formulas of the form $\varphi \wedge \psi$ for satisfiability. An intrinsic problem for this approach and non-regular properties is undecidability. Note that the intersection problem for context-free languages is already undecidable [1].

One of the earliest attempts at verifying non-regular properties of programs was *Non-Regular PDL* [13] which enriches ordinary PDL by context-free programs. Non-Regular PDL is highly undecidable, hence, the logic did not receive much attention for program verification purposes. Its model checking problem, however, remains decidable on finite transition systems.

Although the theoretical complexity of the model checking problem is normally below that of its satisfiability problem, it often requires a lot more time or space to do model checking. This is simply because the input to a model checker is usually a lot bigger compared to that of a satisfiability checker. Hence, the feasibility of model checking is very much limited by the state space explosion problem: real-world examples result in huge transition systems that are very hard to model check simply because of their sheer size. However, in recent years various clever techniques have been invented that can cope with huge state spaces, starting with *local model checking*, and resulting in symbolic methods like *BDD-based* [4] or *bounded model checking* [6]. They are also a reason for the shift in importance from the satisfiability checking to the model checking problem for program verification.

More expressive power naturally comes with higher complexities. But with good model checking techniques at hand, verifying non-regular properties has become worthwhile again. This is for example reflected in the introduction of *Fixpoint Logic with Chop*, FLC, [18] which extends $\mathcal{L}_\mu$ with a sequential composition operator. It is capable of expressing many non-regular – and even non-context-free – properties, and its model checking problem on finite transition systems is decidable in deterministic exponential time [17].

Another logic capable of expressing non-regular properties is the *Modal Iteration Calculus*, MIC, [7] which extends $\mathcal{L}_\mu$ with inflationary fixpoint quantifiers. Similar to FLC, the satisfiability checking problem for MIC is undecidable but its model checking problem is decidable in deterministic polynomial space [7].

In order to achieve non-regular effects in FLC, the original $\mathcal{L}_\mu$ semantics is lifted to a function from sets of states to sets of states. This idea has been followed consequently in the introduction of *Higher Order Fixpoint Logic*, HFL, [23] which incorporates a typed λ-calculus into the modal μ-calculus. This gives it even more expressive power than FLC. HFL is, for example, capable of expressing *assume-guarantee-properties*. Still, HFL's model checking problem on finite transition systems remains decidable. This has been stated in its introductory work [23] but no analysis of its computational complexity has been done so far.

Here we set out to answer the open question concerning the complexity of HFL's model checking problem. We start by recalling the logic and giving a few examples in Section 2. Section 3 presents a reduction from the satisfiability problem for First Order Logic over finite words to HFL's model checking problem. Consequently, the latter is hard for every level of the elementary time/space hierarchy. I.e. there is no model checking algorithm for HFL that runs in time given by a tower of exponentials whose height does not depend on the input formula. This is not too surprising because HFL incorporates the typed λ-calculus for which the problem of deciding whether a given term can be transformed into another given one, is also non-elementary [21]. This can be reduced to the model checking problem for HFL.

Here we provide a more fine-grained analysis of HFL's model checking problem. When restricted to terms of type order k, it is hard for $(k-3)$ExpSpace and included in $(k+1)$ExpTime. It remains to be seen whether this gap can be closed.

2 Preliminaries

Let $\mathcal{P} = \{p, q, \ldots\}$ be a set of atomic propositions, $\mathcal{A} = \{a, b, \ldots\}$ be a finite set of action names, and $\mathcal{V} = \{X, Y, X_1, \ldots\}$ a set of variable names. For simplicity, we fix $\mathcal{P}$, $\mathcal{A}$, and $\mathcal{V}$ for the rest of the paper.

A $v \in \{-, +, 0\}$ is called a variance. The set of HFL-types is the smallest set containing the atomic type Prop and being closed under function typing with variances, i.e. if σ and τ are HFL-types and v is a variance, then $\sigma^v \to \tau$ is an HFL-type. Formulas of HFL are given by the following grammar:

$$\varphi \ ::= \ q \mid X \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle a \rangle \varphi \mid \varphi\ \varphi \mid \lambda(X^v : \tau).\varphi \mid \mu(X : \tau).\varphi\ .$$

We use the standard abbreviations: $\mathtt{tt} := q \vee \neg q$ for some $q \in \mathcal{P}$, $\mathtt{ff} := \neg\mathtt{tt}$, $\varphi \wedge \psi := \neg(\neg\varphi \wedge \neg\psi)$, $[a]\psi := \neg\langle a \rangle \neg\psi$, and $\nu X.\varphi := \neg\mu X.\neg\varphi[\neg X/X]$. We will assume that any variable without an explicit type annotation is of the ground type Prop. Also, if a variance is omitted it is implicitly assumed to be 0.

A sequence Γ of the form $X_1^{v_1} : \tau_1, \ldots, X_n^{v_n} : \tau_n$ where X_i are variables, τ_i are types and v_i are variances is called a *context* (we assume all X_i are distinct). An HFL-formula φ has type τ in context Γ if the statement $\Gamma \vdash \varphi \colon \tau$ can be inferred using the rules of Figure 1. We say that φ is *well-formed* if $\Gamma \vdash \varphi \colon \tau$ for some Γ and τ.

For a variance v, we define its complement v^- as $+$ if $v = -$, as $-$, if $v = +$, and 0 otherwise. For a context $\Gamma = X_1^{v_1} : \tau_1, \ldots, X_n^{v_n} : \tau_n$, the complement Γ^- is defined as $X_1^{v_1^-} : \tau_1, \ldots, X_n^{v_n^-} : \tau_n$.

A (labeled) transition system is a structure $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a}\}, L)$ where $\mathcal{S}$ is a finite set of states, $\xrightarrow{a}$ is a binary relation on states for each $a \in \mathcal{A}$, and

$$(var)\quad \Gamma, X^v : \tau \vdash X : \tau \quad \text{if } v \in \{0, +\} \qquad\qquad (neg)\quad \frac{\Gamma^- \vdash \varphi \colon \mathsf{Prop}}{\Gamma \vdash \neg\varphi \colon \mathsf{Prop}}$$

$$(or)\quad \frac{\Gamma \vdash \varphi \colon \mathsf{Prop} \quad \Gamma \vdash \psi \colon \mathsf{Prop}}{\Gamma \vdash \varphi \vee \psi \colon \mathsf{Prop}} \qquad\qquad (mod)\quad \frac{\Gamma \vdash \varphi \colon \mathsf{Prop}}{\Gamma \vdash \langle a \rangle \varphi \colon \mathsf{Prop}}$$

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

$$(app^+)\quad \frac{\Gamma \vdash \varphi : (\sigma^+ \to \tau) \quad \Gamma \vdash \psi \colon \sigma}{\Gamma \vdash (\varphi\ \psi) : \tau} \qquad\qquad (app^-)\quad \frac{\Gamma \vdash \varphi : (\sigma^- \to \tau) \quad \Gamma^- \vdash \psi \colon \sigma}{\Gamma \vdash (\varphi\ \psi) : \tau}$$

$$(app^0)\quad \frac{\Gamma \vdash \varphi : (\sigma^0 \to \tau) \quad \Gamma \vdash \psi \colon \sigma \quad \Gamma^- \vdash \psi \colon \sigma}{\Gamma \vdash (\varphi\ \psi) : \tau} \qquad\qquad (prop)\quad \frac{}{\Gamma \vdash p \colon \mathsf{Prop}}$$

Fig. 1. Type inference rules for HFL

$$
\begin{aligned}
\mathcal{T}[\![\Gamma \vdash q : \mathsf{Prop}]\!]\eta &= \{s \in \mathcal{S} \mid q \in L(s)\} \\
\mathcal{T}[\![\Gamma \vdash X : \tau]\!]\eta &= \eta(X) \\
\mathcal{T}[\![\Gamma \vdash \neg\varphi : \mathsf{Prop}]\!]\eta &= \mathcal{S} - \mathcal{T}[\![\Gamma^- \vdash \varphi : \mathsf{Prop}]\!]\eta \\
\mathcal{T}[\![\Gamma \vdash \varphi \vee \psi : \mathsf{Prop}]\!]\eta &= \mathcal{T}[\![\Gamma \vdash \varphi : \mathsf{Prop}]\!]\eta \cup \mathcal{T}[\![\Gamma \vdash \psi : \mathsf{Prop}]\!]\eta \\
\mathcal{T}[\![\Gamma \vdash \langle a\rangle\varphi : \mathsf{Prop}]\!]\eta &= \{s \in \mathcal{S} \mid s \xrightarrow{a} t \text{ for some } t \in \mathcal{T}[\![\Gamma \vdash \varphi : \mathsf{Prop}]\!]\eta\} \\
\mathcal{T}[\![\Gamma \vdash \lambda(X^v : \tau).\varphi : \tau^v \to \tau']\!]\eta &= F \in \mathcal{T}[\![\tau^v \to \tau']\!] \text{ s.t. } \forall d \in \mathcal{T}[\![\tau]\!] \\
&\qquad F(d) = \mathcal{T}[\![\Gamma, X^v : \tau \vdash \varphi : \tau']\!]\eta[X \mapsto d] \\
\mathcal{T}[\![\Gamma \vdash \varphi\ \psi : \tau']\!]\eta &= \big(\mathcal{T}[\![\Gamma \vdash \varphi : \tau^v \to \tau']\!]\eta\big)\big(\mathcal{T}[\![\Gamma' \vdash \psi : \tau]\!]\eta\big) \\
\mathcal{T}[\![\Gamma \vdash \mu(X : \tau)\varphi : \tau]\!]\eta &= \textstyle\bigsqcap_{\mathcal{T}[\![\tau]\!]}\{d \in \tau \mid \\
&\qquad \mathcal{T}[\![\Gamma, X^+ : \tau \vdash \varphi : \tau]\!]\eta[X \mapsto d] \le_{\mathcal{T}[\![\tau]\!]} d\}
\end{aligned}
$$

Fig. 2. Semantics of HFL

$L : \mathcal{S} \to 2^{\mathcal{P}}$ is a labeling function denoting the set of propositional constants that are true in a state.

The semantics of a type w.r.t. a transition system $\mathcal{T}$ is a complete lattice, inductively defined on the type as

$$\mathcal{T}[\![\mathsf{Prop}]\!] = (2^{\mathcal{S}}, \subseteq)\ , \qquad \mathcal{T}[\![\sigma^v \to \tau]\!] = (\mathcal{T}[\![\sigma]\!])^v \to \mathcal{T}[\![\tau]\!]\ .$$

Here, for two partial orders $\bar{\tau} = (\tau, \le_\tau)$ and $\bar{\sigma} = (\sigma, \le_\sigma)$, $\bar{\sigma} \to \bar{\tau}$ denotes the partial order of all monotone functions ordered pointwise, and, $\bar{\tau}^v$ denotes $(\tau, \le_\tau^v)$. $\le_\tau^+$ is $\le_\tau$, $a \le_\tau^- b$ iff $b \le_\tau a$, and $\le_\tau^0 = \le_\tau \cap \le_\tau^-$.

An environment η is a possibly partial map on the variable set $\mathcal{V}$. For a context $\Gamma = X_1^{v_1} : \tau_1, \ldots, X_n^{v_n} : \tau_n$, we say that η respects Γ, denoted by $\eta \models \Gamma$, if $\eta(X_i) \in \mathcal{T}[\![\tau_i]\!]$ for $i \in \{1, \ldots, n\}$. We write $\eta[X \mapsto a]$ for the environment that maps X to a and otherwise agrees with η. If $\eta \models \Gamma$ and $a \in \mathcal{T}[\![\tau]\!]$ then $\eta[X \mapsto a] \models \Gamma, X : \tau$, where X is a variable that does not appear in Γ.

For any well-typed term $\Gamma \vdash \varphi : \tau$ and environment $\eta \models \Gamma$, Figure 2 defines the semantics of φ inductively to be an element of $\mathcal{T}[\![\tau]\!]$. In the clause for function application $(\varphi\ \psi)$ the context Γ' is Γ if $v \in \{+, 0\}$, and is Γ^- if $v = -$.

The model checking problem for HFL is the following: Given an HFL sentence $\varphi : \mathsf{Prop}$, a transition system $\mathcal{T}$ and a set of states A decide whether or not $\mathcal{T}[\![\varphi]\!]=A$.

We consider fragments of formulas that can be built using restricted type orders only. Let

$$\mathit{ord}(\mathsf{Prop}) \ := \ 0\ , \quad \mathit{ord}(\sigma \to \tau) \ := \ \max\{1 + \mathit{ord}(\sigma), \mathit{ord}(\tau)\}$$

and $\mathrm{HFL}^k := \{\ \varphi \in \mathrm{HFL} \mid\ \vdash \varphi : \mathsf{Prop}$ using types τ with $\mathit{ord}(\tau) \le k$ only $\}$.

Example 1. The following HFL formula expresses the non-regular property "on any path the number of a's seen at any time never exceeds the number of b's seen so far."

$$\nu X.[a]\mathtt{ff} \wedge [b]\big(\big(\nu(Z : \mathsf{Prop} \to \mathsf{Prop}).\lambda Y.([a]Y \wedge [b]\,(Z\,(Z\,Y)))\big)\ X\big)\ .$$

Note how function composition is used to "remember" in the argument to Z how many times a b-action has been seen along any path. Every b-action gives the potential to do another a-action later on which is remembered in the additional application of Z. a-action "uses up" one Z. If there have been as many a's as b's then the current state must be in the semantics of X again, hence, cannot do another a-action, etc.

Example 2. Let $\mathbf{2}_0^n := n$ and $\mathbf{2}_{m+1}^n := 2^{\mathbf{2}_m^n}$. For any $m \in \mathbb{N}$, there is an HFL formula φ_m expressing the fact that there is a maximal path of length $\mathbf{2}_m^1$ (number of states on this path) through a transition system. It can be constructed using a typed version of the Church numeral 2. Let $P_0 = \mathsf{Prop}$ and $P_{i+1} = P_i \to P_i$. For $i \geq 1$ define ψ_i of type P_{i+1} as $\lambda(F : P_i).\lambda(X : P_{i-1}).F\,(F\,X)$. Then

$$\varphi_m := \psi_m\ \psi_{m-1}\ \ldots\ \psi_1\ \big(\lambda X.\langle - \rangle X\big)\ \big([-]\,\mathtt{ff}\big)\ .$$

Note that for any $m \in \mathbb{N}$, φ_m is of size polynomial in m. This indicates that HFL is able to express computations of Turing Machines of arbitrary elementary complexity. The next section shows that this is indeed the case.

3 The Lower Complexity Bound

Let Σ be a finite alphabet. Formulas of FO in negation normal form over words in Σ^* are given by the following grammar.

$$\varphi \ ::= \ x \leq y \mid x < y \mid P_a(x) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi \mid \forall x.\varphi$$

where x, y are variables and $a \in \Sigma$.

A word $w \in \Sigma^*$ of length n is a function of type $\{0, \ldots, n-1\} \to \Sigma$. Thus, $w(i)$ denotes the i-th letter of w. FO formulas are interpreted over words in the usual way, written $w \models_\eta \varphi$ for a word w, a formula φ and an environment η evaluating the free variables in φ by positions in w.

Let Σ_0 and Π_0 be the set of all quantifier-free formulas of FO. Σ_{k+1} is the closure of $\Sigma_k \cup \Pi_k$ under the boolean operators and existential quantification. Similarly, Π_{k+1} is constructed from $\Sigma_k \cup \Pi_k$ using universal quantification instead.

Let DTime($f(n)$) and DSpace($f(n)$) be the classes of languages that can be decided by a deterministic Turing Machine in time, resp. space, $f(n)$ where n measures the length of the input word to the machine. The k-th levels of the elementary time/space hierarchy are

$$k\textsc{ExpTime} = \bigcup_{c \in \mathbb{N}} \mathrm{DTime}(\mathbf{2}_k^{c \cdot n}), \qquad k\mathrm{ExpSpace} = \bigcup_{c \in \mathbb{N}} \mathrm{DSpace}(\mathbf{2}_k^{c \cdot n})\ .$$

Furthermore, the elementary time/space hierarchy is

$$\textsc{ElTime} := \bigcup_{k \in \mathbb{N}} k\textsc{ExpTime} = \bigcup_{k \in \mathbb{N}} k\mathrm{ExpSpace}\ .$$

The standard translation of an FO formula into a finite automaton [3] and the encoding of space-bounded Turing Machine computations in FO [22,20] yield the following results.

Theorem 1. *An* FO *sentence* $\varphi \in \Sigma_{k+1}$ *of length* n *has a model iff it has a model of length* $\mathbf{2}_k^n$*.*

Theorem 2. *For all* $k \geq 1$*: The satisfiability problem for* FO *formulas in* Σ_{k+1} *is hard for* kExpSpace*.*

We will use these results to prove kExpSpace hardness of HFL. Our first step is to translate the problem of deciding whether a given binary word w of length $\mathbf{2}_k^n$ is a model of an FO sentence φ into the HFL model checking problem.

Let us fix n and define $\mathcal{T}_n$ to be a transition system with states $S_n = \{0, 1, \ldots, n-1\}$, an empty labeling (we will not use propositional constants) and a cyclic next state relation $\rightarrow \subseteq S_n \times S_n$ given by $0 \rightarrow n-1$ and $i \rightarrow i-1$ for $i = 1, ..., n-1$.

We represent the Boolean values *false* and *true* by the two elements of $\mathcal{B} = \{\varnothing, S_n\} \subset \mathcal{T}_n[\![\mathsf{Prop}]\!]$. In the sequel we will implicitly assume that all semantic interpretations are w.r.t $\mathcal{T}_n$ and omit it in front of semantic brackets. Hence we can write $[\![\mathtt{ff}]\!]$ and $[\![\mathtt{tt}]\!]$ for the representations of *false* and *true*, respectively.

Let $\mathcal{N}_0^n = \{\{0\}, \{1\}, \ldots, \{n-1\}\}$ and let $\mathcal{N}_{k+1}^n = \mathcal{N}_k^n \rightarrow \mathcal{B}$. Clearly, each $\mathcal{N}_k^n$ has exactly $\mathbf{2}_k^n$ elements which we will use to represent numbers in the range $0, \ldots, \mathbf{2}_k^n - 1$. We have also $\mathcal{N}_k^n \subseteq [\![N_k]\!]$ where $N_0 = \mathsf{Prop}$ and $N_{k+1} = N_k \rightarrow \mathsf{Prop}$.

Note that the elements of $\mathcal{N}_{k+1}^n = \mathcal{N}_k^n \rightarrow \mathcal{B}$ can be equivalently viewed as predicates over $\mathcal{N}_k^n$, subsets of $\mathcal{N}_k^n$ or binary words of length $\mathbf{2}_k^n$.

For a number $j \in \{0, 1, \ldots, \mathbf{2}_k^n - 1\}$ let $[\![j]\!]^k$ be the element of $\mathcal{N}_k^n$ representing j, defined inductively by $[\![j]\!]^0 = \{j\}$ and

$$[\![j]\!]^k(x) := \begin{cases} [\![\mathtt{tt}]\!] & \text{if } x = [\![i]\!]^{k-1} \text{ and } j_i = 1, \\ [\![\mathtt{ff}]\!] & \text{o.w.} \end{cases}$$

where j_i is the i-th bit in the binary representation of j. For a binary word w of length $\mathbf{2}_k^n$ let $[\![w]\!]^k := [\![\sum_i w_i \cdot 2^i]\!]^{k+1}$.

The possibility of a compact encoding of FO logic in HFL relies on the existence of polynomial size HFL formulas describing basic operations on numbers represented as elements of $\mathcal{N}_k^n$. We define

$$\mathsf{inc}_k : N_k \rightarrow N_k \ , \quad \mathsf{eq}_k : N_k \rightarrow N_k \rightarrow \mathsf{Prop} \ , \quad \mathsf{search}_k : (N_k \rightarrow \mathsf{Prop}) \rightarrow \mathsf{Prop}$$

adhering to the following specifications

$$[\![\mathsf{inc}_k]\!]\ [\![j]\!]^k = [\![j+1]\!]^k \ , \qquad [\![\mathsf{eq}_k]\!]\ [\![j]\!]^k\ [\![i]\!]^k = \begin{cases} [\![\mathtt{tt}]\!] \ , & \text{iff } j = i \\ [\![\mathtt{ff}]\!] \ , & \text{o.w.} \end{cases}$$

For a predicate $p \in \mathcal{N}_{k+1}^n$,

$$[\![\mathsf{search}_k]\!]\ p = \begin{cases} [\![\mathtt{tt}]\!] & \text{iff exists } x \in \mathcal{N}_k^n \text{ s.t. } p(x) = [\![\mathtt{tt}]\!] \\ [\![\mathtt{ff}]\!] & \text{o.w.} \end{cases}$$

The search function search_k can be implemented using inc_k and recursion. A helper function $\mathsf{search}'_k\ P\ x$ applies P to the successive numbers, starting from x, taking the union of the results.

$$\begin{aligned}\mathsf{search}_k &:= \lambda(P : N_k \to \mathsf{Prop}).\mathsf{search}'_k\ P\ \bot^k\ ,\\ \mathsf{search}'_k &:= \lambda(P : N_k \to \mathsf{Prop}).\ \mu(Z : N_k \to \mathsf{Prop}).\\ &\qquad \lambda(X : N_k).\ (P\,X) \vee (Z\,(\mathsf{inc}_k\,X))\ .\end{aligned}$$

Formula $\bot^k : N_k$ represents 0 and is defined as

$$\bot^0 := [-]\,\mathtt{ff}\ , \qquad \bot^k := \lambda(X : N_{k-1}).\,\mathtt{ff} \quad \text{for } k > 0\ .$$

Functions eq_k and inc_k are defined by induction on k. For $k = 0$ we set

$$\mathsf{eq}_0 := \lambda X.\lambda Y.(X \leftrightarrow Y)\ , \qquad \mathsf{inc}_0 := \lambda X.\langle - \rangle X\ .$$

For $k > 0$, function eq_k is implemented by searching for an argument at which two number representations differ:

$$\mathsf{eq}_k := \lambda(X : N_k).\lambda(Y : N_k).\,\neg\big(\mathsf{search}_{k-1}\ \lambda(I : N_{k-1}).\neg(X\,I \leftrightarrow Y\,I)\big)\ .$$

Function inc_k is the usual incrementation of a number in binary representation. The helper function $\mathsf{inc}'_k\,x\,i$ adds one to the i-th bit of n and possibly the following bits if the carry-over occurs.

$$\mathsf{inc}_k := \lambda(X : N_k).\,\mathsf{inc}'_k\ X\ \bot^{k-1}\ .$$

The value of $[\![\mathsf{inc}'_k]\!]\ [\![x]\!]^k\ [\![i]\!]^{k-1}$ is a function which for each j returns the j-th bit of $x+2^i$ (encoded as $[\![\mathtt{tt}]\!]$ or $[\![\mathtt{ff}]\!]$). For $j = i$ the corresponding bit is $\neg x_i$. If there is no carry-over ($x_i = 0$) then the remaining bits are unchanged. Otherwise the remaining bits are the same as in $x + 2^{i+1}$.

$$\begin{aligned}\mathsf{inc}'_k := \ &\lambda(X : N_k).\,\mu(Z : N_{k-1} \to N_k).\,\lambda(I : N_{k-1}).\\ &\quad\lambda(J : N_{k-1}).\ \mathsf{if}\ (\mathsf{eq}_{k-1}\ J\ I)\\ &\qquad\qquad (\neg(X\ I))\\ &\qquad\qquad \big(\mathsf{if}\ \neg(X\ I)\ (X\ J)\ (Z\ (\mathsf{inc}_{k-1}\,I)\ J)\big)\end{aligned}$$

where $\mathsf{if} := \lambda P.\lambda Q.\lambda R.\,(P \wedge Q) \vee (\neg P \wedge R)$.

Note that the lengths of inc_k, eq_k and search_k as strings can be exponential in k. However, the number of their subformulas is only polynomial in k.

Lemma 1. *For any $k \geq 0$, any $i \in \{0, \ldots, \mathbf{2}^n_k - 1\}$, and any $p \in \mathcal{N}^n_k \to \mathcal{B}$ we have: $[\![\mathsf{search}'_k]\!]\ p\ [\![i]\!]^k = [\![\mathtt{tt}]\!]$ iff $p([\![j]\!]^k) = [\![\mathtt{tt}]\!]$ for some $i \leq j < \mathbf{2}^n_k$.*

Proof. Simply because $[\![\mathsf{search}'_k]\!]\ p\ [\![i]\!]^k \equiv \bigcup_{j=i}^{\mathbf{2}^n_k - 1} p([\![j]\!]^k)$. □

Proposition 1. *For any k, $\mathsf{eq}_k, \mathsf{inc}_k \in \mathrm{HFL}^{k+1}$ and $\mathsf{search}_k, \mathsf{search}'_k \in \mathrm{HFL}^{k+2}$.*

Let φ be an FO sentence. For given k we translate φ into an HFL^{k+2} formula $tr_k(\varphi) : N_{k+1} \to \mathsf{Prop}$ s.t. for any word w of length $\mathbf{2}^n_k$, w is a model of φ iff $\big(\mathcal{T}_n[\![tr_k(\varphi)]\!]\big)\ (\mathcal{T}_n[\![w]\!]^k) = \mathcal{T}_n[\![\mathtt{tt}]\!]$.

$$\begin{array}{rcl}
tr_k(x \le y) &:=& \lambda(w : N_{k+1}).\,\mathsf{search}'_k\; x\; (\lambda(u : N_k).\,\mathsf{eq}_k\, u\, y)\;, \\
tr_k(P_0(x)) &:=& \lambda(w : N_{k+1}).\,\neg(w\; x)\;, \\
tr_k(P_1(x)) &:=& \lambda(w : N_{k+1}).\,(w\; x)\;, \\
tr_k(\exists x.\varphi) &:=& \lambda(w : N_{k+1}).\,\mathsf{search}_k\; (\lambda(x : N_k).\,tr_k(\varphi)\; w)\;, \\
tr_k(\neg\varphi) &:=& \lambda(w : N_{k+1}).\,\neg(tr_k(\varphi)\; w)\;, \\
tr_k(\varphi \vee \psi) &:=& \lambda(w : N_{k+1}).\,(tr_k(\varphi)\; w) \vee (tr_k(\psi)\; w)\;.
\end{array}$$

Conjunctions and universal quantifiers can be translated using negation and the above formulas. Note that free variables of φ become free variables of type N_k and variance 0 in $tr_k(\varphi)$.

Lemma 2. *For any* FO *sentence φ the translation $tr_k(\varphi)$ is a predicate. That is, $\mathcal{T}_n[\![tr_k(\varphi)]\!]$ is an element of $\mathcal{N}^n_{k+2}$ – a function which returns either $\mathcal{T}_n[\![\mathtt{tt}]\!]$ or $\mathcal{T}_n[\![\mathtt{ff}]\!]$ when applied to an argument from $\mathcal{N}^n_{k+1}$.*

Proof. This follows from the fact that, by their specifications, search_k and search'_k are predicates. Hence $tr_k(\varphi)$ is a predicate as a Boolean combination of predicates.

Proposition 2. *For any k and φ, $tr_k(\varphi) \in \mathrm{HFL}^{k+2}$.*

Lemma 3. *Let φ be an* FO *formula with variables $x_1, \ldots, x_l$. For any word w of length $\mathbf{2}^n_k$ and* FO*-environment η we have*

$$w \models_\eta \varphi \quad \textit{iff} \quad \big(\mathcal{T}_n[\![tr_k(\varphi)]\!]\rho\big)(\mathcal{T}_n[\![w]\!]^k) = \mathcal{T}_n[\![\mathtt{tt}]\!]$$

where ρ is an HFL *environment given by $\rho(x_i) = \mathcal{T}_n[\![\eta(x_i)]\!]^k$.*

Proof. By induction on the structure of φ. We fix n and k and as before omit $\mathcal{T}_n$ in front of semantic brackets.

Case $\varphi = x_i \le x_j$: Then $[\![tr_k(\varphi)]\!]\rho = [\![\mathsf{search}'_k]\!]\; [\![a]\!]^k\, p$ where $a = \eta(x_i)$, $b = \eta(x_j)$ and predicate p is given by $p(x) = [\![\mathsf{eq}_k]\!]\, x\, [\![b]\!]^k$. We have $w \models_\eta \varphi$ iff $a \le b$ iff exists $a \le c < \mathbf{2}^n_k$ s.t. $p([\![c]\!]^k) = [\![\mathtt{tt}]\!]$ iff $[\![\mathsf{search}'_k]\!]\; [\![a]\!]^k\, p = [\![\mathtt{tt}]\!]$, by Lemma 1.

Case $\varphi = \exists\, x.\psi$: Then $[\![tr_k(\varphi)]\!]\rho = [\![\mathsf{search}_k]\!]\, p$ where $p([\![i]\!]^k) = [\![tr_k(\psi)]\!]\rho[x \mapsto [\![i]\!]^k]\, [\![w]\!]^k$. By the specification of search_k and induction hypothesis we have $([\![tr_k(\varphi)]\!]\rho)\; [\![w]\!]^k = [\![\mathtt{tt}]\!]$ iff $p([\![i]\!]^k) = [\![\mathtt{tt}]\!]$ for some i iff $w \models_{\eta[x \mapsto i]} \psi$ iff $w \models_\eta \varphi$.

Case $\varphi = P_0(x_i)$: Then $w \models_\eta \varphi$ iff $w_{\eta(x_i)} = 0$ iff $[\![w]\!]^k\, [\![\eta(x_i)]\!]^k = [\![\mathtt{ff}]\!]$ iff $([\![tr_k(\varphi)]\!]\rho)\; [\![w]\!]^k = [\![\mathtt{tt}]\!]$.

All other cases are either analogous or follow immediately from the induction hypothesis when negation is used in formulas. □

Lemma 4. *The satisfiability problem for* FO *sentences in Σ_k is polynomially reducible to the model checking problem for* HFL^{k+2}.

Proof. First note that we can restrict ourselves to the satisfiability problem for FO^k over the binary alphabet $\{0, 1\}$ because any other alphabet can be encoded in it at a logarithmic expense only.

Given a Σ_k formula φ of length n we can construct in polynomial time and space an instance of the HFL^{k+2} model checking problem consisting of a formula $\varphi' = \mathsf{search}_k\ tr_{k-1}(\varphi)$, transition system $\mathcal{T}_n$ and the set $\mathcal{T}_n[\![\mathtt{tt}]\!] = \{0, 1, \dots, n-1\}$. Note that $\varphi' \in \mathrm{HFL}^{k+2}$ by Propositions 1 and 2. From Lemmas 1, 2 and 3 it follows that $\mathcal{T}_n[\![\varphi']\!] = \mathcal{T}_n[\![\mathtt{tt}]\!]$ iff φ has a model of size $\mathbf{2}^n_{k-1}$ which, by Theorem 1, is equivalent to φ having a model. □

Theorem 2 together with the reduction of Lemma 4 yields the following result.

Theorem 3. *The model checking problem for* HFL^{k+3} *is hard for* kExpSpace *under polynomial time reductions.*

Corollary 1. *The model checking problem for* HFL *is not in* ElTime.

Note that the reduction only uses modal formulas $\langle - \rangle\varphi$ and $[-]\varphi$ because $\mathbf{2}^n_k$ is an upper bound on a minimal model for an FO sentence in Σ_{k+1} of length n. However, $\mathbf{2}^n_k = \mathbf{2}^1_{k+\log^* n}$. This enables us to use modality-free formulas and transition systems of fixed size 1 in the reduction. The price to pay is that, in order to achieve kExpSpace-hardness, one needs formulas with unrestricted types. But it shows that HFL model checking is not in ElTime for fixed and arbitrarily small transition systems already.

4 The Upper Complexity Bound

In the following we will identify a type τ and its underlying complete lattice induced by a transition system $\mathcal{T}$ with state set $\mathcal{S}$. In order to simplify notation we fix $\mathcal{T}$ for the remainder of this section.

Suppose $|\mathcal{S}| = n$ for some $n \in \mathbb{N}$. We define the size $\#(\tau)$ of an HFL type τ, as well as $rp(\tau)$ – a space measure for a representation of one of its elements.

$$\#(\mathsf{Prop}) \ := \ 2^n\ , \qquad \#(\sigma \to \tau) \ := \ \#(\tau)^{\#(\sigma)}\ ,$$
$$rp(\mathsf{Prop}) \ := \ n\ , \qquad rp(\sigma \to \tau) \ := \ \#(\sigma) \cdot rp(\tau)\ .$$

Lemma 5. *For all* HFL *types* τ *we have:*

(a) There are only $\#(\tau)$ *many different elements of* τ.
(b) An element x *of* τ *can be represented using space* $O(rp(\tau))$.
(c) $\#(\tau) \leq \mathbf{2}^{O(n)}_{ord(\tau)+1}$.
(d) $rp(\tau) \leq \mathbf{2}^{O(n)}_{ord(\tau)}$.

Proof. Part (a) is standard. Part (b) is proved by induction on the structure of τ. The claim is easily seen to be true for $\tau = \mathsf{Prop}$. Let $\tau = \tau_1 \to \tau_2$, i.e. any element of τ is a function. Such a function can be represented by cascading tables where a table for type Prop consists of one entry only. For $\tau_1 \to \tau_2$ the table must contain for every element of τ_1 one entry of τ_2. Since $\#(\tau_1)$ is finite, we can assume to have a standard enumeration for all elements of τ_1. This enumeration can be used to determine the order in which the function values are stored in the table. With such an enumeration at hand, one does not need to write down the function arguments, and by (a) and the induction hypothesis the overall space needed is $O(rp(\tau_1 \to \tau_2))$.

We also prove (c) by induction on the structure of τ. For $\tau = \mathsf{Prop}$ this is true. Let $\tau = \tau_1 \to \tau_2$. Then we have

$$\begin{aligned} \#(\tau) &= \#(\tau_2)^{\#(\tau_1)} \leq (\mathbf{2}^{O(n)}_{ord(\tau_2)+1})^{\mathbf{2}^{O(n)}_{ord(\tau_1)+1}} = 2^{\mathbf{2}^{O(n)}_{ord(\tau_2)} \cdot \mathbf{2}^{O(n)}_{ord(\tau_1)+1}} \\ &\leq 2^{\mathbf{2}^{2 \cdot O(n)}_{\max\{ord(\tau_2), ord(\tau_1)+1\}}} = \mathbf{2}^{O(n)}_{ord(\tau)+1} . \end{aligned}$$

Again, the claim in (d) is easily seen to be true for $\tau = \mathsf{Prop}$. Let $\tau = \tau_1 \to \tau_2$, i.e. $ord(\tau) \geq 1$. Note that $ord(\tau_1) \leq ord(\tau) - 1$, and $ord(\tau_2) \leq ord(\tau)$. Then

$$rp(\tau) = \mathbf{2}^{O(n)}_{ord(\tau_1)+1} \cdot \mathbf{2}^{O(n)}_{ord(\tau_2)} \leq \mathbf{2}^{O(n)}_{ord(\tau)} \cdot \mathbf{2}^{O(n)}_{ord(\tau)} \leq \mathbf{2}^{O(n)}_{ord(\tau)}$$

by (c) and the hypothesis. □

Theorem 4. *For all $k \in \mathbb{N}$, the model checking problem for* HFL^k *and finite transition systems is in* $(k+1)$EXPTIME.

Proof. For a finite transition system $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \mathcal{A}\}, L)$ with $|\mathcal{S}| = n$ and an HFL formula φ of type τ, we describe an alternating procedure for finding the denotation of φ. The existential player $\exists$ proposes an element of τ as $[\![\varphi]\!]$ and the universal player $\forall$ challenges her choice. The game proceeds along the structure of φ in the following way.

If $\varphi = \lambda(X : \sigma).\psi$ then $\forall$ chooses an entry in the table written by $\exists$ as a value of φ. The row in which the entry is found determines the argument to φ, i.e. a value for X. $\forall$ can now invoke the verification protocol to check that this is the correct value of $\psi(X)$ when X has the value given by the entry.

For $\varphi = \psi_1\,\psi_2$, first player $\exists$ has to provide a table for ψ_1 and a denotation for ψ_2. The player $\forall$ can either check that the value in ψ_1's table in the row corresponding to ψ_2 is the previously guessed value for φ. Or he can proceed to challenge the denotation of ψ_2.

To verify a guess x of the value of $\varphi = \mu(X : \tau).\psi$ first $\exists$ writes a table of a function $\lambda(X : \tau).\psi$. Furthermore, she names a row in this table that determines a value for X. If the value in this row is not the same as the value of X she looses because she has not found a fixpoint. Then player $\forall$ can either challenge the whole table as above or name another smaller table row that defines a fixpoint. Note that it is always possible to require the entries in a table of type τ to respect the order $\leq_\tau$.

In all other cases φ is of type Prop and its value is a bit vector of length n. Correctness of Boolean operations can be easily verified by $\forall$ using additional $O(n)$ space for storing the values of the operands.

Clearly the space needed to perform the above procedure is bounded by the maximal $rp(\tau)$ where τ is a type of a subformula of φ. This includes using the enumeration function to find corresponding table rows. It is fair to assume that in order to enumerate $\mathbf{2}_k^{O(n)}$ elements one does not need more space than $\mathbf{2}_k^{O(n)}$. Hence, by Lemma 5 the model checking problem is in alternating $\mathbf{2}_k^{O(n)}$-space which equals $(k+1)$ExpTime [5]. □

5 Conclusion

We have shown that the model checking problem for HFL is hard for every kExpTime and consequently of non-elementary complexity. It is tempting to dismiss HFL as a specification formalism of any practical use. But the same argument would also rule out any practical implementation of a satisfiability checker for Monadic Second Order Logic over words or trees (MSO) since this problem has non-elementary complexity, too. However, the verification tool Mona [14] shows that in many cases satisfiability of MSO formulas can be checked efficiently. This is mainly because of the use of efficiently manipulable data structures like BDDs [2], and the fact that formulas used in practical cases do not coincide with those that witness the high complexity.

Thus, the theoretical complexity bounds proved in this paper need to be seen as a high alert warning sign for someone building a model checking tool based on HFL. This will certainly require the use of efficient data structures as well as other clever optimizations. However, only such an implementation will be able to judge the use of HFL as a specification formalism properly. Furthermore, Theorems 3 and 4 show that in order to reach high levels in ElTime, one needs formulas of high type order. But such types might not be needed in order to formulate natural correctness properties, cf. Example 1.

It remains to be seen whether the gap between $(k-3)$ExpSpace-hardness and inclusion in $(k+1)$ExpTime can be reduced and finally closed.

Acknowledgments. We would like to thank Martin Leucker for discussing HFL's model checking problem and an anonymous referee for pointing out Statman's result on the typed λ-calculus.

References

1. Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonologie, Sprachwissenschaft und Kommunikationsforschung*, 14:113–124, 1961.
2. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, August 1986.
3. J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Congress on Logic, Method, and Philosophy of Science*, pages 1–12, Stanford, CA, USA, 1962. Stanford University Press.
4. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.
5. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, January 1981.

6. E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.
7. A. Dawar, E. Grädel, and S. Kreutzer. Inflationary fixed points in modal logic. In L. Fribourg, editor, *Proc. 15th Workshop on Computer Science Logic, CSL'01*, LNCS, pages 277–291, Paris, France, September 2001. Springer.
8. E. A. Emerson. Uniform inevitability is tree automaton ineffable. *Information Processing Letters*, 24(2):77–79, January 1987.
9. E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30:1–24, 1985.
10. E. A. Emerson and J. Y. Halpern. "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, January 1986.
11. E. A. Emerson and C. S. Jutla. Tree automata, μ-calculus and determinacy. In *Proc. 32nd Symp. on Foundations of Computer Science*, pages 368–377, San Juan, Puerto Rico, October 1991. IEEE.
12. M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, April 1979.
13. D. Harel, A. Pnueli, and J. Stavi. Propositional dynamic logic of nonregular programs. *Journal of Computer and System Sciences*, 26(2):222–243, April 1983.
14. J. G. Henriksen, J. Jensen, M. Joergensen, and N. Klarlund. MONA: Monadic second-order logic in practice. *LNCS*, 1019:89–110, 1995.
15. D. Janin and I. Walukiewicz. On the expressive completeness of the propositional μ-calculus with respect to monadic second order logic. In U. Montanari and V. Sassone, editors, *Proc. 7th Conf. on Concurrency Theory, CONCUR'96*, volume 1119 of *LNCS*, pages 263–277, Pisa, Italy, August 1996. Springer.
16. D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, December 1983.
17. M. Lange and C. Stirling. Model checking fixed point logic with chop. In M. Nielsen and U. H. Engberg, editors, *Proc. 5th Conf. on Foundations of Software Science and Computation Structures, FOSSACS'02*, volume 2303 of *LNCS*, pages 250–263, Grenoble, France, April 2002. Springer.
18. M. Müller-Olm. A modal fixpoint logic with chop. In C. Meinel and S. Tison, editors, *Proc. 16th Symp. on Theoretical Aspects of Computer Science, STACS'99*, volume 1563 of *LNCS*, pages 510–520, Trier, Germany, 1999. Springer.
19. A. Pnueli. The temporal logic of programs. In *Proc. 18th Symp. on Foundations of Computer Science, FOCS'77*, pages 46–57, Providence, RI, USA, October 1977. IEEE.
20. K. Reinhardt. The complexity of translating logic to finite automata. In E. Grädel, W. Thomas, and Th. Wilke, editors, *Automata, Languages, and Infinite Games*, volume 2500 of *LNCS*, pages 231–238. Springer, 2002.
21. R. Statman. The typed λ-calculus is not elementary recursive. *Theoretical Computer Science*, 9:73–81, 1979.
22. L. Stockmeyer. *The Computational Complexity of Word Problems.* PhD thesis, MIT, 1974.
23. M. Viswanathan and R. Viswanathan. A higher order modal fixed point logic. In Ph. Gardner and N. Yoshida, editors, *Proc. 15th Int. Conf. on Concurrency Theory, CONCUR'04*, volume 3170 of *LNCS*, pages 512–528, London, UK, 2004. Springer.