On the Relationship between Higher-Order Recursion Schemes and Higher-Order Fixpoint Logic

Naoki Kobayashi
The University of Tokyo, Japan
koba@is.s.u-tokyo.ac.jp

Étienne Lozes
LSV, ENS Paris-Saclay, CNRS, France
lozes@lsv.ens-cachan.fr

Florian Bruse
University of Kassel, Germany
florian.bruse@uni-kassel.de

Abstract

We study the relationship between two kinds of higher-order extensions of model checking: HORS model checking, where models are extended to higher-order recursion schemes, and HFL model checking, where the logic is extended to higher-order modal fixpoint logic. These extensions have been independently studied until recently, and the former has been applied to higher-order program verification, while the latter has been applied to assume-guarantee reasoning and process equivalence checking. We show that there exist (arguably) natural reductions between the two problems. To prove the correctness of the translation from HORS to HFL model checking, we establish a type-based characterization of HFL model checking, which should be of independent interest. The results reveal a close relationship between the two problems, enabling cross-fertilization of the two research threads.

Categories and Subject Descriptors F.4.1 [*Mathematical Logic and Formal Languages*]: Mathematical Logic

Keywords higher-order recursion schemes, higher-order modal fixpoint logic, model checking

1. Introduction

Inspired by the great success of finite state model checking [4], two kinds of its higher-order extensions have been studied recently. One is model checking of higher-order recursion schemes (HORS model checking, for short) [7, 11, 24], which asks, given a higher-order recursion scheme $\mathcal{G}$ (which is a kind of a tree grammar) and a formula φ of the modal μ-calculus (or equivalently, an alternating parity tree automaton), whether the tree generated by $\mathcal{G}$ satisfies φ. The other is higher-order modal fixpoint logic model checking of finite state systems (HFL model checking, for short) [33], which asks, given a finite state system $\mathcal{L}$ and a formula φ of the higher-order modal fixpoint logic (which is a higher-order extension of the modal μ-calculus), whether $\mathcal{L}$ satisfies φ. Thus, in HORS model checking, systems to be verified are higher-order, whereas in HFL model checking, properties to be checked are higher-order. HORS model checking has recently been successfully applied to verification of higher-order programs [8, 15, 17, 18, 22, 25, 32, 34]. HFL model checking has been applied to assume-guarantee reasoning [33] and process equivalence checking [19]. In general, HORS model checking is useful for precisely modeling and verifying certain *infinite* state systems, whereas HFL model checking is useful for checking *non-regular* properties of systems that cannot be expressed in ordinary modal logics such as LTL, CTL, and modal μ-calculus.

Unfortunately, the two problems (i.e., HORS/HFL model checking) have been studied independently by different research communities, and little has been known on their relationship. Interestingly, both problems are k-EXPTIME complete, where k is the largest type-theoretic order of functions used in HORS or HFL formulas. Thus, there should exist translations between order-k HORS model checking problems and order-k HFL model checking, but no direct (i.e., without going via Turing machines) translations were known.

In the present paper, we present direct, mutual translations between the HORS and HFL model checking problems. Interestingly, the roles of systems and properties are switched by the translations; in the HORS-to-HFL translation, a HORS (which is a description of a system to be verified) is translated to an HFL formula, and an automaton (which is a description of a property to be checked) is translated to a transition system, whereas in the converse translation, an HFL formula is translated to a HORS and a transition system is translated to an automaton. The translations are non-trivial. For the HORS-to-HFL translation, we have to replace the parity acceptance condition on the tree generated by HORS with proper alternation of least and greatest fixpoint operators of HFL. For the converse translation, we have to emulate the calculation of least and greatest fixpoint operators by HORS, which requires a tricky encoding of numbers.

The correctness of the HORS-to-HFL translation is also non-trivial.[1] To this end, we provide a type-based characterization of HFL model checking, so that an HFL formula is typable in the type system parameterized by a finite transition system if and only if the transition system satisfies the formula. We then prove that a HORS is typable in (a variation of) Kobayashi and Ong's type system for characterizing the HORS model checking if and only if the corresponding HFL formula is typable in the aforementioned type system. Thus, the correctness of the HORS-to-HFL formula follows from that of Kobayashi and Ong's type system.

The type-based characterization of HFL model checking mentioned above should be of independent interest. A type-based characterization of HORS model checking is well established [11, 12] and has been used for studies of practical algorithms [3, 9, 10, 23, 26], parameterized complexity [12, 13], decidability proofs [12, 31], etc. of HORS model checking. Our type-based characterization of HFL model checking is similar to (and actually simpler than) that for

POPL'17, January 15–21, 2017, Paris, France
© 2017 ACM. 978-1-4503-4660-3/17/01...$15.00
http://dx.doi.org/10.1145/3009837.3009854

[1] It is necessarily so because the decidability of HORS model checking is non-trivial (and in fact, it has been the subject of many papers [5, 12, 24, 27]) whereas that of HFL model checking is straightforward; a proof of the correctness of the HORS-to-HFL translation would therefore serve as an alternative proof of the decidability of HORS model checking.

HORS model checking. Thus, the type-based characterization clarifies the similarity and difference of HORS/HFL model checking. We also expect that the type-based approach to HFL will allow us to develop practical algorithms for HFL model checking, following the success of the corresponding approach to HORS model checking.

The rest of the paper is structured as follows. Section 2 reviews the definitions of HORS/HFL model checking problems. Section 3 presents a translation from HORS model checking to HFL model checking. Section 4 provides a type-based characterization of HFL model checking, and Section 5 uses it to prove the correctness of the translation of Section 3. Section 6 presents a translation from HFL model checking to HORS model checking, and proves its correctness. Section 7 discusses related work and Section 8 concludes the paper. Omitted proofs are given in the longer version of the paper [16].

2. Preliminaries

In this section, we first recall, in Section 2.1, the standard definitions of (infinite) trees, parity games and tree automata (that are required for defining HORS and HFL), and then review the definitions of higher-order recursion schemes (HORS) and higher-order modal fixpoint logic (HFL), and model checking problems on them in Sections 2.2 and 2.3.

2.1 Trees, Parity Games, and Alternating Parity Tree Automata

Let $\mathbb{N}_+$ be the set of positive integers. Given a set L, an *L-labeled tree* T is a partial map from $\mathbb{N}_+^*$ to L such that $\forall \pi \in \mathbb{N}_+^*.\forall i \in \mathbb{N}_+.\ \pi \cdot i \in \mathtt{dom}(T) \implies \{\pi, \pi\cdot 1, \dots \pi \cdot (i-1)\} \subseteq \mathtt{dom}(T)$. An element of $\mathtt{dom}(T)$ is called a *node*. For $n, n' \in \mathtt{dom}(T)$, n' is a child of n if n is the longest strict prefix of n'.

A *ranked alphabet* Σ is a map from a finite set of symbols to the set of non-negative integers, called *arities*. A Σ-labeled tree T is a *ranked tree* if for every node $n \in \mathtt{dom}(T)$, the number of children of n is $\Sigma(T(n))$.

A *parity game* is a two player game played by Player and Opponent and is defined by a tuple $\mathbf{G} = (V_\forall, V_\exists, v_{\mathsf{init}}, E, \Omega)$, where $V_\forall, V_\exists$ are disjoint sets of *positions*, $v_{\mathsf{init}} \in V_\forall \cup V_\exists$ is the initial position, $E \subseteq (V_\forall \cup V_\exists)^2$ is a set of *moves*, and $\Omega : V_\forall \cup V_\exists \to \{0, \dots, p-1\}$ assigns to each position a *priority*. Positions in $V_\exists$ are called Player's positions, and positions in $V_\forall$ are called Opponent's positions.

A play is a finite or infinite sequence of positions $v_0, v_1, \dots$ such that $v_0 = v_{\mathsf{init}}$ and $(v_i, v_{i+1}) \in E$ for all $i \geq 0$. The play is won by Player if either it is finite and the last position $v_n \in V_\forall$ is an Opponent's position such that $v_n E (= \{v \mid (v_n, v) \in E\}) = \emptyset$, or the play is infinite and the largest priority occurring infinitely often (i.e., $\limsup_{i\to\infty} \Omega(v_i)$) is even. A memoryless strategy for Player is $W \subseteq E$ such that $vW = vE$ for all $v \in V_\forall$ (Opponent's moves remain unchanged), and for all $v \in V_\exists$, there is at most one v' such that $(v, v') \in W$ (Player's moves are uniquely determined by the current position); it is a winning strategy for Player if all plays in the game $(V_\forall, V_\exists, v_{\mathsf{init}}, E \cap W, \Omega)$ are won by Player.

Given a finite set X, the set $\mathsf{B}^+(X)$ of positive Boolean formulas over X is defined by

$$\mathsf{B}^+(X) \ni f ::= \mathtt{tt} \mid \mathtt{ff} \mid x \mid f_1 \vee f_2 \mid f_1 \wedge f_2,$$

where x ranges over X.

Definition 1 (alternating parity tree automata)**.** *An* alternating parity tree automaton (APT) *is a quintuple $\mathcal{A} = (Q, \Sigma, \delta, q_{\mathsf{init}}, \Omega)$ such that:*

- *Q is a finite set of states with a distinguished initial state $q_{\mathsf{init}} \in Q$.*
- *Σ is a ranked alphabet.*
- *$\delta : Q \times \Sigma \to \mathsf{B}^+(\{1, \dots, m\} \times Q)$ is a transition function, where m is the largest arity of symbols in $\mathtt{dom}(\Sigma)$.*
- *$\Omega : Q \to \{0, \dots, p-1\}$ assigns a priority to each state.*

Given an APT $\mathcal{A}$ and a Σ-labeled ranked tree T, the acceptance game $\mathbf{G}(T, \mathcal{A}) = (V_\forall, V_\exists, v_{\mathsf{init}}, E, \Omega)$ is the parity game defined by $V_\forall \cup V_\exists := \{(n, f) \mid n \in \mathtt{dom}(T),\ f$ is a subformula of $\delta(q, a)$ for some $(q, a) \in Q \times \mathtt{dom}(\Sigma)\}$, with $(n, f) \in V_\forall$ iff f is a conjunction or $\mathtt{tt}$, $v_{\mathsf{init}} := (\epsilon, \delta(q_{\mathsf{init}}, T(\epsilon)))$, $E := \{((n, f_1 * f_2), (n, f_i)) \mid n \in \mathtt{dom}(T), i \in \{1, 2\}, * \in \{\vee, \wedge\}\} \cup \{((n, (i, q)), (n.i, \delta(q, T(n.i)))) \mid n, n.i \in \mathtt{dom}(T)\}$, $\Omega(n, (i, q)) = \Omega(q)$, and $\Omega(n, f \vee f') = \Omega(n, f \wedge f') = 0$. The language of $\mathcal{A}$, written $L(\mathcal{A})$, is the set of trees T such that Player has a winning strategy for $\mathbf{G}(T, \mathcal{A})$.

Intuitively, a position (n, f) of the game above represents a state where Player tries to prove that the node n satisfies f, and Opponent tries to disprove it. If f is a disjunction $f_1 \vee f_2$, Player picks i and tries to show that the node n satisfies f_i. If f is a conjunction $f_1 \wedge f_2$, Opponent picks i and tries to disprove that the node n satisfies f_i. If $f = (i, q)$, then Player tries to show that the child $n.i$ satisfies $\delta(q, T(n.i))$ (i.e., is accepted from q by the automaton). When a play continues indefinitely, Player wins iff the largest priority of states visited infinitely often is even.

Example 1. *Consider the APT $\mathcal{A}_0 = (\{q_0, q_1\}, \Sigma, \delta, q_0, \Omega)$, where:*

$$\Sigma = \{a \mapsto 2, b \mapsto 1, c \mapsto 0\}$$
$$\delta(q_i, a) = (1, q_0) \wedge (2, q_0) \quad \delta(q_i, b) = (1, q_1) \quad \delta(q_i, c) = \mathtt{tt} \quad (\textit{for } i \in \{0, 1\})$$
$$\Omega(q_0) = 1 \qquad \Omega(q_1) = 2$$

Let T be the tree where $\mathtt{dom}(T) = (2.1)^* \cup (2.1)^*.1 \cup (2.1)^*.2$, $T(n) = a$ if $n \in (2.1)^*$, $T(n) = c$ if $n \in (2.1)^*.1$, and $T(n) = b$ if $n \in (2.1)^*.2$. (Thus, T is the regular infinite tree defined by $T = a\ c\ (b\ T)$. Let D be $\mathtt{dom}(T)$. The acceptance game $\mathbf{G}(T, \mathcal{A}_0)$ is $(V_\forall, V_\exists, v_{\mathsf{init}}, E, \Omega')$, where:

$$\begin{aligned}
V_\forall &= \{(n, (1, q_0) \wedge (2, q_0)) \mid n \in D\} \cup \{(n, \mathtt{tt}) \mid n \in D\}\\
V_\exists &= \{(n, f) \mid n \in D, f \in \{(1, q_0), (2, q_0), (1, q_1)\}\}\\
v_{\mathsf{init}} &= (\epsilon, (1, q_0) \wedge (2, q_0))\\
E &= \{((n, (1, q_0) \wedge (2, q_0)), (n, (i, q_0))) \mid n \in D, i \in \{1, 2\}\}\\
&\quad \cup \{((n, (1, q_i)), (n.1, (1, q_0) \wedge (2, q_0))) \mid\\
&\qquad\qquad n \in (2.1)^*.2, i \in \{0, 1\}\}\\
&\quad \cup \{((n, (2, q_0)), (n.2, (1, q_1))) \mid n \in (2.1)^*\}\\
&\quad \cup \{((n, (1, q_1)), (n.1, \mathtt{tt})) \mid n \in (2.1)^*\}\\
\Omega'(n, (i, q_j)) &= j + 1 \textit{ for } n \in D,\ j \in \{0, 1\}, \textit{ and } i \in \{1, 2\}\\
\Omega'(n, f) &= 0 \textit{ for } n \in D,\ f \in \{\mathtt{tt}, (1, q_0) \wedge (2, q_0)\}.
\end{aligned}$$

E itself is a winning strategy for $\mathbf{G}(T, \mathcal{A}_0)$; so, T is accepted by $\mathcal{A}_0$. In general, a tree is accepted by $\mathcal{A}_0$ if and only if every infinite path of the tree contains infinitely many occurrences of b. □

Remark 1. *The acceptance of a tree by an APT can also be understood as follows, without using parity games. Let $\mathcal{A} = (Q, \Sigma, \delta, q_{\mathsf{init}}, \Omega)$ be an APT. The automaton has subformulas of $\delta(q, a)$ as "intermediate" states. Given a tree T, $\mathcal{A}$ runs a thread for reading the root with the initial state q_{init}. Whenever a thread visits a node labeled with a at state q, it transits to an intermediate state $\delta(q, a)$. A thread in an intermediate state f performs the following actions, depending on the shape of f.*

- *Case $f = f_1 \wedge f_2$: the thread splits into two threads with states f_1 and f_2.*
- *Case $f = f_1 \vee f_2$: the thread moves to either state f_1 or f_2.*
- *Case $f = (i, q)$: the thread visits the i-th child of the current node with state q.*
- *Case $f = \mathtt{tt}$: the thread terminates successfully.*

- *Case $f = \texttt{ff}$: the thread fails.*

An APT $\mathcal{A}$ accepts a tree T if there is a run in which no thread fails, and for every non-terminating thread, the largest priority of states visited infinitely often is even.

A *labeled transition system* (LTS) $\mathcal{L}$ is a quadruple $(U, A, \longrightarrow, s_{\mathsf{init}})$, where U is a finite set of states, A is a finite set of actions, $\longrightarrow\ \subseteq U \times A \times U$ is a transition relation, and s_{init} is the initial state. We write $s \xrightarrow{a} s'$ when $(s, a, s') \in \longrightarrow$.

2.2 Model Checking of HORS

In this section, we review the definition of higher-order recursion schemes (HORS) and the model checking problem on them [24]. A HORS is a simply-typed, higher-order tree grammar for generating a labeled tree, and the model checking problem on it asks whether the tree generated by a given HORS satisfies a given property (expressed in terms of an alternating tree automaton or a modal μ-calculus formula). When a tree is viewed as a transition system (where a node is regarded as a state and an edge as a transition), a HORS is considered a (possibly infinite) transition system. The trees generated by order-0 HORS's are regular, which correspond to finite state transition systems, whereas the trees generated by order-1 HORS's are those generated by pushdown systems. In that sense, the HORS model checking may be considered a strict extension of finite state model checking and pushdown model checking. Yet, the model checking problem remains decidable [24].

We first define types and terms. The set of *simple types*, ranged over by κ, is defined by:

$$\kappa ::= \star \mid \kappa_1 \to \kappa_2.$$

The base type $\star$ is used as the type of trees below. The *order* of a type κ is defined by: $\mathsf{ord}(\star) = 0$ and $\mathsf{ord}(\kappa_1 \to \kappa_2) = \max(\mathsf{ord}(\kappa_1) + 1, \mathsf{ord}(\kappa_2))$. The set of *(simply-typed) λ-terms*, ranged over by e, is defined by:

$$e ::= x \mid e_1 e_2 \mid \lambda x : \kappa.e.$$

A λ-term that does not contain λ is called an *applicative term*. We often omit the type annotation and just write $\lambda x.e$ for $\lambda x : \kappa.e$. As usual, the type judgment relation $\mathcal{K} \vdash e : \kappa$, where $\mathcal{K}$ is a map[2] from a finite set of variables to the set of simple types, is defined as the least relation closed under the following rules:

$$\frac{}{\mathcal{K}, x : \kappa \vdash x : \kappa} \qquad \frac{\mathcal{K}, x : \kappa_1 \vdash e : \kappa_2}{\mathcal{K} \vdash \lambda x : \kappa_1.e : \kappa_1 \to \kappa_2}$$

$$\frac{\mathcal{K} \vdash e_0 : \kappa_1 \to \kappa_2 \qquad \mathcal{K} \vdash e_1 : \kappa_1}{\mathcal{K} \vdash e_0\, e_1 : \kappa_2}$$

Definition 2 (HORS)**.** *A higher-order recursion scheme (HORS, for short) $\mathcal{G}$ is a quadruple $(\Sigma, \mathcal{N}, \mathcal{R}, S)$, where:*

- *Σ is a ranked alphabet. The elements of Σ are called* terminals.
- *$\mathcal{N}$ is a map from a finite set of symbols (called* non-terminals*) to the set of simple types.*
- *$\mathcal{R}$ is a map from the set of non-terminals to the set of λ-terms (where both terminals and non-terminals are treated as variables). If $\mathcal{N}(A) = \kappa_1 \to \cdots \to \kappa_\ell \to \star$, then $\mathcal{R}(A)$ must be of the form $\lambda x_1 : \kappa_1. \cdots \lambda x_\ell : \kappa_\ell.e$, where e is an applicative term such that $\Sigma^! \cup \mathcal{N}, x_1 : \kappa_1, \ldots, x_\ell : \kappa_\ell \vdash e : \star$. Here, $\Sigma^!$ denotes:*

$$\{a : \underbrace{\star \to \cdots \to \star}_{\Sigma(a)} \to \star \mid a \in \mathtt{dom}(\Sigma)\}.$$

- *S is a non-terminal such that $\mathcal{N}(S) = \star$.*

[2] Following the usual convention, we write $x_1 : \kappa_1, \ldots, x_n : \kappa_n$ instead of $\{x_1 \mapsto \kappa_1, \ldots, x_n \mapsto \kappa_n\}$ for a type environment.

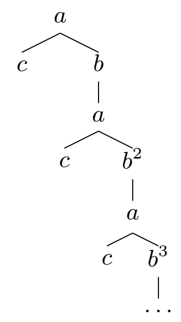

Figure 1. The tree generated by $\mathcal{G}_0$ in Example 2.

The order *of a HORS is* $\max(\{\mathsf{ord}(\mathcal{N}(A)) \mid A \in \mathtt{dom}(\mathcal{N})\})$*. The rewriting relation $e \longrightarrow_{\mathcal{G}} e'$ is the least relation closed under the following rules:*

- *$A\, e_1 \ldots e_\ell \longrightarrow_{\mathcal{G}} [e_1/x_1, \ldots, e_\ell/x_\ell]e$ if $\mathcal{R}(A) = \lambda x_1 : \kappa_1. \cdots \lambda x_\ell : \kappa_\ell.e$.*
- *$a\, e_1 \cdots e_i \cdots e_\ell \longrightarrow_{\mathcal{G}} a\, e_1 \cdots e_i' \cdots, e_\ell$ if $e_i \longrightarrow_{\mathcal{G}} e_i'$ and $\Sigma(a) = \ell$.*

We often represent $\mathcal{R}$ in the form of rewriting rules, writing $A\ x_1\ \cdots\ x_\ell \to e$ for $\mathcal{R}(A) = \lambda x_1 : \kappa_1. \cdots \lambda x_\ell : \kappa_\ell.e$.

The tree generated by $\mathcal{G}$ is the one obtained from S by (possibly) infinite rewriting. Formally, it is defined as follows.

Definition 3 ($T_{\mathcal{G}}$)**.** *For an applicative term e of type $\star$, the $(\Sigma \cup \{\bot \mapsto 0\})$-labeled tree $e^{\bot}$ is defined by:*

$$(a\, e_1 \cdots e_k)^{\bot} = a\, e_1{}^{\bot} \cdots e_k{}^{\bot} \qquad (A\, e_1 \cdots e_k)^{\bot} = \bot$$

We define the relation $\sqsubseteq$ on trees by: $T_1 \sqsubseteq T_2$ iff $\mathtt{dom}(T_1) \subseteq \mathtt{dom}(T_2)$ and for every $n \in \mathtt{dom}(T_1)$, $T_1(n) = \bot$ or $T_1(n) = T_2(n)$. The tree generated by $\mathcal{G}$, written $T_{\mathcal{G}}$, is $\bigsqcup\{e^{\bot} \mid S \longrightarrow_{\mathcal{G}}^{*} e\}$, where $\bigsqcup U$ denotes the least upper bound of the trees in U with respect to $\sqsubseteq$.[3]

Example 2. *Consider the HORS $\mathcal{G}_0 = (\{a \mapsto 2, b \mapsto 1, c \mapsto 0\}, \mathcal{N}, \mathcal{R}, S)$, where $\mathcal{N} = \{S : \star, F : (\star \to \star) \to \star, B : (\star \to \star) \to \star \to \star\}$, and $\mathcal{R}$ consists of the following rewriting rules:*

$$S \to F\, b \qquad F\ g \to a\ c\ (g(F(B\, b))) \qquad B\ g\ x \to b(g\, x).$$

S is reduced as follows:

$$S \longrightarrow Fb \longrightarrow ac(b(F(B\, b))) \longrightarrow ac(b(ac(Bb(F(B(Bb))))))$$
$$\longrightarrow ac(b(ac(b(b(F(B(B\, b))))))) \longrightarrow \cdots.$$

The tree generated by $\mathcal{G}_0$ (i.e., $T_{\mathcal{G}_0}$) is shown in Figure 1, where b^i denotes i repetitions of b.

Definition 4 (model checking of HORS)**.** *We write $\mathcal{G} \models \mathcal{A}$ if $T_{\mathcal{G}} \in L(\mathcal{A})$. The HORS model checking problem is the problem of deciding whether $\mathcal{G} \models \mathcal{A}$, given a HORS $\mathcal{G}$ and an alternating parity tree automaton $\mathcal{A}$.*

Example 3. *Consider the APT $\mathcal{A}_0$ in Example 1 and the HORS $\mathcal{G}_0$ in Example 2. Then, $\mathcal{G}_0 \models \mathcal{A}_0$ holds.* □

Theorem 5 (Ong [24])**.** *The HORS model checking problem is k-EXPTIME complete for order-k HORS.*

As in [12, 24], in the rest of this paper, we assume that $T_{\mathcal{G}}$ does not contain $\bot$. Given $\mathcal{G}$ and $\mathcal{A}$, we can always transform them to $\mathcal{G}'$ and $\mathcal{A}'$ such that (i) $\mathcal{G} \models \mathcal{A}$ if and only if $\mathcal{G}' \models \mathcal{A}'$ and (ii) $T_{\mathcal{G}'}$ does not contain $\bot$.

[3] The least upper bound always exists, as $\longrightarrow$ is confluent.

2.3 HFL Model Checking

In this section we review Higher-Order Modal Fixpoint Logic [33] (HFL) and its model-checking problem. HFL is an extension of the modal μ-calculus with higher-order recursive predicates; HFL formulas φ and HFL types η are defined by the following grammar

$$\begin{aligned}\varphi ::= &\top \mid \bot \mid X \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \langle a\rangle\varphi \mid [a]\varphi \\ &\mid \mu X^{\eta}.\varphi \mid \nu X^{\eta}.\varphi \mid \lambda X:\eta.\varphi \mid \varphi_1\ \varphi_2 \\ \eta ::= &\bullet \mid \eta_1 \to \eta_2\end{aligned}$$

The syntax of the formulas except the last two components (λ-abstractions and applications) is almost identical to that of the modal μ-calculus; in particular, as in the modal μ-calculus, we have the least and great fixpoint operators μ and ν; the difference is that they can be over *higher-order* predicates (created by a λ-abstraction $\lambda X:\eta.\varphi$). In its original formulation [33], HFL includes negations. In our setting, these are disallowed for simplicity, which is not a restriction since any closed HFL formula can be transformed to an equivalent negation-free formula [20].

Each binder (μ, ν, λ) is annotated with the type of the bound variable (we may sometimes omit this annotation when it is clear from the context). The type $\bullet$ describes propositions, and the type $\eta_1 \to \eta_2$ describes functions from η_1 to η_2. The *order* of an HFL type η is defined by: $\mathsf{ord}(\bullet) = 0$ and $\mathsf{ord}(\eta_1 \to \eta_2) = \max(\mathsf{ord}(\eta_1)+1, \mathsf{ord}(\eta_2))$. A type judgment relation is of the form $\mathcal{H} \vdash \varphi : \eta$, where $\mathcal{H}$ is a map from a finite set of variables to the set of HFL types. Type judgments are derived from the following rules.

$$\frac{}{\mathcal{H} \vdash \top : \bullet} \qquad \frac{}{\mathcal{H} \vdash \bot : \bullet} \qquad \frac{}{\mathcal{H}, X:\eta \vdash X : \eta}$$

$$\frac{\mathcal{H} \vdash \varphi : \bullet}{\mathcal{H} \vdash \langle a\rangle\varphi : \bullet} \qquad \frac{\mathcal{H} \vdash \varphi : \bullet}{\mathcal{H} \vdash [a]\varphi : \bullet} \qquad \frac{\mathcal{H} \vdash \varphi_1 : \bullet \quad \mathcal{H} \vdash \varphi_2 : \bullet}{\mathcal{H} \vdash \varphi_1 \vee \varphi_2 : \bullet}$$

$$\frac{\mathcal{H} \vdash \varphi_1 : \bullet \quad \mathcal{H} \vdash \varphi_2 : \bullet}{\mathcal{H} \vdash \varphi_1 \wedge \varphi_2 : \bullet} \qquad \frac{\mathcal{H}, X:\eta \vdash \varphi : \eta}{\mathcal{H} \vdash \mu X^{\eta}.\ \varphi : \eta}$$

$$\frac{\mathcal{H}, X:\eta \vdash \varphi : \eta}{\mathcal{H} \vdash \nu X^{\eta}.\ \varphi : \eta} \qquad \frac{\mathcal{H}, X:\eta_1 \vdash \varphi : \eta_2}{\mathcal{H} \vdash \lambda X:\eta_1.\ \varphi : \eta_1 \to \eta_2}$$

$$\frac{\mathcal{H} \vdash \varphi_1 : \eta_2 \to \eta \quad \mathcal{H} \vdash \varphi_2 : \eta_2}{\mathcal{H} \vdash \varphi_1\ \varphi_2 : \eta}$$

A closed formula φ is well-typed and has type η if the type judgment $\emptyset \vdash \varphi : \eta$ is derivable from the above rules. In the remainder, we always implicitly assume that all the (closed) formulas are well-typed.

The *order* of a formula φ is the largest order of the type of a subformula occurring in φ. A formula is said to be a formula of the modal μ-calculus if its order is 0.

Let $(U, A, \longrightarrow, s_{\mathsf{init}})$ be a fixed LTS. The semantics of a formula of type η is an object of the lattice $(D_\eta, \sqcup_\eta, \sqcap_\eta)$ defined by induction on η: Define $D_\bullet = \mathcal{P}(U)$ as the complete lattice of sets of states, and if $\eta = \eta_1 \to \eta_2$ then define $D_\eta = D_{\eta_1} \to D_{\eta_2}$ as the complete lattice of monotone functions from D_{η_1} to D_{η_2}. For every type η and function $f \in D_{\eta\to\eta}$, f has a unique least fixpoint $\mathsf{LFP}_\eta(f) \in D_\eta$ and a unique greatest fixpoint $\mathsf{GFP}_\eta(f) \in D_\eta$, respectively defined as $\bigsqcap\{x \in D_\eta \mid f(x) \sqsubseteq x\}$ and $\bigsqcup\{x \in D_\eta \mid x \sqsubseteq f(x)\}$.

The interpretation $[\![\mathcal{H}]\!]$ of a type environment is the set of maps ρ such that $\rho(X) \in D_{\mathcal{H}(X)}$ for each $X \in \mathtt{dom}(\rho)$. The interpretation $[\![\mathcal{H} \vdash \varphi : \eta]\!]$ is a map from $[\![\mathcal{H}]\!]$ to D_η defined by induction on φ as follows:

$$\begin{aligned}
&[\![\mathcal{H} \vdash \top : \bullet]\!](\rho) = U\\
&[\![\mathcal{H} \vdash \bot : \bullet]\!](\rho) = \emptyset\\
&[\![\mathcal{H}, X:\eta \vdash X : \eta]\!](\rho) = \rho(X)\\
&[\![\mathcal{H} \vdash \langle a\rangle\varphi : \bullet]\!](\rho) = \{s \mid \exists s' \in [\![\mathcal{H} \vdash \varphi : \bullet]\!](\rho).\ s \xrightarrow{a} s'\}\\
&[\![\mathcal{H} \vdash [a]\varphi : \bullet]\!](\rho)\\
&\quad = \{s \mid \forall s' \in S.\ (s \xrightarrow{a} s' \text{ implies } s' \in [\![\mathcal{H} \vdash \varphi : \bullet]\!](\rho))\}\\
&[\![\mathcal{H} \vdash \varphi_1 \vee \varphi_2 : \bullet]\!](\rho) = [\![\mathcal{H} \vdash \varphi_1 : \bullet]\!](\rho) \cup [\![\mathcal{H} \vdash \varphi_2 : \bullet]\!](\rho)\\
&[\![\mathcal{H} \vdash \varphi_1 \wedge \varphi_2 : \bullet]\!](\rho) = [\![\mathcal{H} \vdash \varphi_1 : \bullet]\!](\rho) \cap [\![\mathcal{H} \vdash \varphi_2 : \bullet]\!](\rho)\\
&[\![\mathcal{H} \vdash \mu X^{\eta}.\varphi : \eta]\!](\rho) = \mathsf{LFP}_\eta([\![\mathcal{H} \vdash \lambda X:\eta.\ \varphi]\!](\rho))\\
&[\![\mathcal{H} \vdash \nu X^{\eta}.\varphi : \eta]\!](\rho) = \mathsf{GFP}_\eta([\![\mathcal{H} \vdash \lambda X:\eta.\ \varphi]\!](\rho))\\
&[\![\mathcal{H} \vdash \lambda X:\eta_1.\ \varphi : \eta_1 \to \eta_2]\!](\rho)\\
&\quad = \{V \mapsto [\![\mathcal{H}, X:\eta_1 \vdash \varphi : \eta_2]\!](v[X \mapsto V]) \mid V \in D_{\eta_1}\}\\
&[\![\mathcal{H} \vdash \varphi_1\ \varphi_2 : \eta]\!](\rho)\\
&\quad = [\![\mathcal{H} \vdash \varphi_1 : \eta' \to \eta]\!](\rho)([\![\mathcal{H} \vdash \varphi_2 : \eta']\!](\rho))
\end{aligned}$$

Note that, in the last clause, η' is uniquely determined by $\mathcal{H}$ and φ_2.

We often omit $\mathcal{H} \vdash \cdot : \eta$ and just write $[\![\varphi]\!]$ for $[\![\mathcal{H} \vdash \varphi : \eta]\!]$, with the understanding that each subformula is implicitly annotated with its type. For a closed formula φ of type $\bullet$, we simply write $[\![\varphi]\!]$ for $[\![\emptyset \vdash \varphi : \bullet]\!](\rho_\emptyset)$, where $\rho_\emptyset$ is the empty interpretation. We write $\mathcal{L} \models \varphi$ if $s_{\mathsf{init}} \in [\![\varphi]\!]$.

We now review the definition of HFL model checking and the decidability/complexity result.

Definition 6 (HFL model checking)**.** *The HFL model checking problem is the problem of deciding whether $\mathcal{L} \models \varphi$, given a closed HFL formula φ of type $\bullet$ and a labeled transition system $\mathcal{L}$.*

Theorem 7 ([2, 33])**.** *The HFL model checking problem is decidable [33]. It is k-EXPTIME complete for order-k HFL formulas [2].*

Example 4. *Consider the following HFL formula φ_0:*

$$(\nu F^{(\bullet\to\bullet)\to\bullet}.\lambda X:\bullet\to\bullet.\langle a\rangle(X(F\,(\lambda Y:\bullet.\langle b\rangle(X\,Y)))))\\ (\lambda Y:\bullet.\langle b\rangle Y).$$

It represents the property that there exists a transition sequence of the form: $abab^2ab^3ab^4\cdots$. In fact if we replace F with $\lambda X:\bullet\to\bullet.\langle a\rangle(X(F\,(\lambda Y:\bullet.\langle b\rangle(X\,Y))))$ infinitely often and reduce the β-redexes, we obtain the formula:

$$\langle a\rangle\langle b\rangle\langle a\rangle\langle b\rangle^2\langle a\rangle\langle b\rangle^3\langle a\rangle\langle b\rangle^4\cdots.$$

Consider the LTS $\mathcal{L}_0 = (\{s_0, s_1\}, \{a, b\}, \longrightarrow, s_0)$, where $\longrightarrow$ is given by:

$$s_0 \xrightarrow{a} s_1 \quad s_1 \xrightarrow{b} s_0 \quad s_1 \xrightarrow{b} s_1.$$

Then we have $\mathcal{L}_0 \models \varphi_0$. □

Example 5. *Consider the following formula φ_1 [19]:*

$$\mu E^{\bullet\to\bullet\to\bullet}.\lambda X:\bullet.\lambda Y:\bullet.(X \wedge Y) \vee E\,(\langle a\rangle X)\,(\langle b\rangle Y).$$

The formula $\varphi_1\ X\ Y$ means "there exists $n \geq 0$ such that $\langle a\rangle^n X$ and $\langle b\rangle^n Y$ holds. For example, $\varphi_2 := \varphi_1\ \varphi_0\ ([b]\bot)$ (where φ_0 is the one given in Example 4) means that there exists $n \geq 0$ such that a transition sequence of the form: $abab^2ab^3ab^4\cdots$ is possible after n steps of a-transitions, and no b-transition is possible after n steps of b-transitions. The LTS $\mathcal{L}_0$ in Example 4 satisfies φ_2, since the property is satisfied for $n = 0$.

For discussing transformations between HFL and HORS, it is convenient to express HFL formulas in the form of systems of equations, called HES.

Definition 8 (HES)**.** *A hierarchical equation system (HES) is a sequence of equations of the form $X_1^{\eta_1} =_{\alpha_1} \varphi_1; \cdots; X_n^{\eta_n} =_{\alpha_n} \varphi_n$. where each α_i is ν or μ, and for each $i = 1, \ldots, n$, φ_i is a formula without fixpoint binders such that $X_1:\eta_1 \ldots, X_n:\eta_n \vdash \varphi_i:\eta_i$.*

For an HES $\mathcal{E} = (X_1^{\eta_1} =_{\alpha_1} \varphi_1; \cdots ; X_n^{\eta_n} =_{\alpha_n} \varphi_n)$, we write $\mathcal{E}(X_i)$ for φ_i. We often omit the type annotation η_i. The HFL formula denoted by $\mathcal{E} := (X_1^{\eta_1} =_{\alpha_1} \varphi_1; \cdots ; X_n^{\eta_1} =_{\alpha_n} \varphi_n)$ is defined inductively by:

$$\begin{aligned} &\mathit{toHFL}(X^\eta =_\alpha \varphi) = \alpha X^\eta.\varphi \\ &\mathit{toHFL}(\mathcal{E}; X^\eta =_\alpha \varphi) = \mathit{toHFL}([\alpha X^\eta.\varphi/X]\mathcal{E}). \end{aligned}$$

We write $\mathcal{L} \models \mathcal{E}$ if $\mathcal{L} \models \mathit{toHFL}(\mathcal{E})$. We sometimes write $X\ y_1 \cdots y_k =_\alpha \varphi$ for $X =_\alpha \lambda y_1. \cdots \lambda y_k.\varphi$.

Example 6. *The HFL formula φ_0 in Example 4 can be represented as the following HES.*

$$\begin{aligned} &S =_\nu F\,(\lambda Y : \bullet.\langle b\rangle Y); \\ &F =_\nu \lambda X : \bullet \to \bullet.\langle a\rangle(X(F\,(\lambda Y : \bullet.\langle b\rangle(X\,Y)))). \end{aligned}$$

We can also restrict HES so that λ occurs only at the top-level. For example, the HES above can further be transformed to the following equivalent HES $\mathcal{E}_0$.

$$\begin{aligned} &S =_\nu F\,B; \quad F =_\nu \lambda X : \bullet \to \bullet.\langle a\rangle(X(F\,(G\ X))); \\ &G =_\nu \lambda X : \bullet \to \bullet.\lambda Y : \bullet.\langle b\rangle(X\,Y); \quad B =_\nu \lambda Y : \bullet.\langle b\rangle Y. \end{aligned}$$

Since the equations for S, G, B are not recursive, it does not matter whether they are annotated with μ or ν. □

Example 7. *Consider the following HES $\mathcal{E}_\bot$:*

$$S =_\mu X; \quad Y =_\nu \lambda Z.\langle a\rangle(Z \wedge X); \quad X =_\mu \langle a\rangle(Y\ X).$$

Then $\mathcal{E}_\bot$ is unsatisfiable. This can be checked by making the following observations:

- *$\mathit{toHFL}(\mathcal{E}_\bot)$ is the formula $\mu X.\langle a\rangle(\varphi\ X)$ where φ is the HFL formula $\nu Y.\lambda Z.\langle a\rangle\big(Z \wedge (\mu X'.\langle a\rangle(Y\ X'))\big)$.*
- *since $\varphi\ Z$ implies $\langle a\rangle Z$, $\mathit{toHFL}(\mathcal{E}_\bot)$ implies $\mu X.\langle a\rangle\langle a\rangle X$, which is unsatisfiable.*

3. From HORS to HFL Model Checking

We introduce a reduction from HORS model checking to HFL model checking. The reduction proceeds by exchanging the roles of the model and the specification:

- the alternating parity tree automaton $\mathcal{A}$ of an instance of a HORS model-checking problem is encoded as the labeled transition system $\mathcal{L}_\mathcal{A}$ of an instance of the HFL model-checking problem; and
- similarly, the HORS $\mathcal{G}$ is encoded as a HFL formula $\varphi_\mathcal{G}$.

Intuitively, $\mathcal{L}_\mathcal{A}$ represents the transitions that can be made by the automaton $\mathcal{A}$ (according to the behavior of $\mathcal{A}$ described in Remark 1), and the formula $\varphi_\mathcal{G}$ describes that $\mathcal{L}_\mathcal{A}$ has transitions corresponding to a successful run of $\mathcal{A}$ for the tree generated by $\mathcal{G}$. We now present these encodings; we prove their soundness in Section 5.

3.1 Tree Automata Encoded as LTS

Let us fix an APT $\mathcal{A} = (Q, \Sigma, \delta, q_{\mathsf{init}}, \Omega)$ and construct the labeled transition system $\mathcal{L}_\mathcal{A}$ encoding it. Intuitively, the control graph of $\mathcal{A}$ becomes the LTS, but since the transition relation of $\mathcal{A}$ uses positive Boolean formulas, these must be encoded as states of the transition system. Formally, the set of states of $\mathcal{L}_A$ is $Q \cup Q_f$, where $Q_f := \{f \mid f$ is a subformula of $\delta(q, a)$ for some $(q, a) \in Q \times \mathtt{dom}(\Sigma)\}$. The rest of the encoding makes sure that the transition relation of the automaton and the state priorities are represented by the labeling of the transitions. The set of labels of $\mathcal{L}_A$ is the set

$$\begin{aligned} &\{a_i \mid a \in \mathtt{dom}(\Sigma), i \in \{0, 1, \ldots, p-1\}\} \\ \cup\ &\{d \mid d \in \{1, \ldots, m\}\} \\ \cup\ &\{\mathtt{and}, \mathtt{or}, \mathtt{true}\} \end{aligned}$$

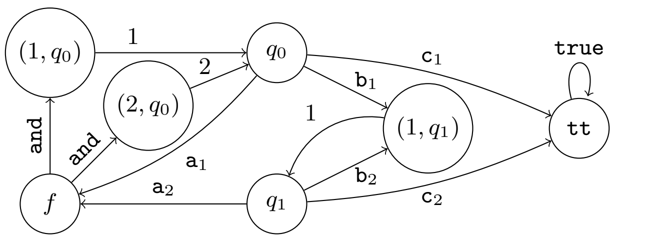

Figure 2. The LTS $\mathcal{L}_{\mathcal{A}_0}$ associated to the APT $\mathcal{A}_0$ of Example 1, where $f = (1, q_0) \wedge (2, q_0)$, and q_0 is the initial state.

where $p - 1$ is the largest priority, and m is the largest arity. The initial state q_{init} of the automaton is also the initial state of the transition system, and the transition relation is defined by

$$\begin{aligned} &q \xrightarrow{a_{\Omega(q)}} \delta(q, a) \qquad (d, q) \xrightarrow{d} q \\ &f_1 \wedge f_2 \xrightarrow{\mathtt{and}} f_i \quad f_1 \vee f_2 \xrightarrow{\mathtt{or}} f_i \quad \mathtt{tt} \xrightarrow{\mathtt{true}} \mathtt{tt} \end{aligned}$$

for $q \in Q$, $a \in \mathtt{dom}(\Sigma)$, and $i = 1, 2$. Note how the priority of a state q is determined by the index i on the label of any transition $q \xrightarrow{\mathtt{a}_i}$ starting at q. The positive Boolean formulas are represented by their syntax tree, with each leaf having an outgoing transition towards the automaton state associated to it.

Example 8. *Let $\mathcal{A}_0$ be the APT of Example 1. The LTS $\mathcal{L}_\mathcal{A}$ encoding $\mathcal{A}_0$ is depicted on Figure 2.*

3.2 The Case of Trivial Automata

In order to get a better intuition of the encoding of $\mathcal{G}$ into an HFL formula $\varphi_\mathcal{G}$, we first discuss the special case where the automaton $\mathcal{A}$ is a trivial tree automaton [1], i.e., an alternating parity tree automaton where all the states have priority 0. This class of automata has been used to verify higher-order programs against safety properties [11].

As explained at the beginning of this section, $\varphi_\mathcal{G}$ expresses the property that the automaton (or, the corresponding LTS $\mathcal{L}_\mathcal{A}$ constructed above) has a successful run for the tree generated by $\mathcal{G}$. Let us first consider a special case, namely where $\mathcal{G}$ generates the finite tree $a\,c\,(b\,c)$. Then, since the initial state of the automaton should be able to accept a, the LTS $\mathcal{L}_\mathcal{A}$ should have a transition a_0; hence $\varphi_\mathcal{G}$ should be of the form $\langle a_0\rangle\varphi_1$, where φ_1 describes the property that should be satisfied by the state $s = \delta(q_{\mathsf{init}}, a)$. The formula φ_1 is not aware of the shape of $\delta(q_{\mathsf{init}}, a)$, but knows that the state s of the LTS after the a-transition is a positive Boolean formula. Thus, φ_1 asserts the following property:

- If $s = (1, q)$, i.e., if there is a $\xrightarrow{1}$-transition, then the next state (corresponding to q) should have transitions corresponding to an accepting run of $\mathcal{A}$ for the first child c.
- If $s = (2, q)$, i.e., if there is a $\xrightarrow{2}$-transition, then the next state should have transitions corresponding to an accepting run of $\mathcal{A}$ for the second child $b\,c$.
- If $s = f_1 \wedge f_2$, then any state after a $\xrightarrow{\mathtt{and}}$-transition should satisfy φ_1 again.
- If $s = f_1 \vee f_2$, then some state after a $\xrightarrow{\mathtt{or}}$-transition should satisfy φ_1 again.
- If $s = \mathtt{tt}$, i.e., if there is a $\xrightarrow{\mathtt{true}}$-transition, then there is no further requirement.

Thus, φ_1 can be described as

$$\nu X.\langle 1\rangle\varphi_c \vee \langle 2\rangle\varphi_{b\,c} \vee (\langle \mathtt{and}\rangle\top \wedge [\mathtt{and}]X) \vee (\langle \mathtt{or}\rangle X) \vee \langle \mathtt{true}\rangle\top,$$

where φ_c and $\varphi_{b\,c}$ describe the properties that the current state has transitions corresponding to accepting runs for c and $b\,c$ respectively, which can be defined by:

$$\varphi_c := \langle c_0\rangle \nu X.((\langle \mathtt{and}\rangle\top \wedge [\mathtt{and}]X) \vee (\langle \mathtt{or}\rangle X) \vee \langle \mathtt{true}\rangle\top$$
$$\varphi_{b\,c} := \langle b_0\rangle \nu X.\langle 1\rangle\varphi_c \vee ((\langle \mathtt{and}\rangle\top \wedge [\mathtt{and}]X) \vee (\langle \mathtt{or}\rangle X) \vee \langle \mathtt{true}\rangle\top.$$

By preparing the following formula L_n:

$$\nu X.\lambda y_1,\dots,y_n.\ \bigvee_{j=1}^{n}\langle j\rangle y_j \vee (\langle \mathtt{and}\rangle\top \wedge [\mathtt{and}](X\ y_1\dots y_n)) \vee\langle \mathtt{or}\rangle(X\ y_1\dots y_n) \vee \langle \mathtt{true}\rangle\top$$

the formula $\varphi_{\mathcal{G}}$ can be simplified to:

$$\langle a_0\rangle(L_2\ (\langle c\rangle L_0)\ (\langle b_0\rangle(L_1(\langle c_0\rangle L_0)))).$$

In general, for a finite tree T, the formula φ_T that describes the property "the LTS $\mathcal{L}_{\mathcal{A}}$ has transitions corresponding to a successful run of $\mathcal{A}$ that accepts T", can be constructed inductively by:

$$\varphi_{a\,T_1\cdots T_\ell} = \langle a_0\rangle(L_\ell\,\varphi_{T_1}\ \cdots\ \varphi_{T_\ell}).$$

In other words, the translation from a tree T to the corresponding formula works as a homomorphism that replaces each tree constructor a of arity ℓ with $\lambda x_1.\cdots\lambda x_\ell.\langle a_0\rangle(L_\ell\,x_1\ \cdots\ x_\ell)$. Thus, we can naturally extend the translation to one from a HORS to a formula, as given below.

For a given HORS $\mathcal{G} = (\Sigma,\mathcal{N},\mathcal{R},S)$, let $\mathcal{E}_{\mathcal{G}}$ be the HES $A_0 =_\nu (e_0)^\dagger;\dots;A_m =_\nu (e_m)^\dagger;\mathcal{E}_{aux}$ where (i) $\mathcal{E}_{aux}$ is the set of definitions for L_n:

$$L_n =_\nu\quad \lambda y_1,\dots,y_n.\ \bigvee_{j=1}^{n}\langle j\rangle y_j \vee (\langle \mathtt{and}\rangle\top \wedge [\mathtt{and}](L_n\ y_1\dots y_n)) \vee\langle \mathtt{or}\rangle(L_n\ y_1\dots y_n) \vee \langle \mathtt{true}\rangle\top$$

for $n \in \{1,\dots,k\}$ with k being the largest arity; (ii) $A_0,\dots,A_m$ are the non-terminals of $\mathcal{G}$ with $S = A_0$; (iii) $e_i = \mathcal{R}(A_i)$; and (iv) $(e)^\dagger$ is defined by induction on e as follows.

$(\lambda y:\kappa.\ e)^\dagger = \lambda y:(\kappa)^\dagger.\ (e)^\dagger$
$(e_1\ e_2)^\dagger = (e_1)^\dagger\ (e_2)^\dagger$
$(z)^\dagger = z$ if z is either a non-terminal or a variable
$(\mathtt{a})^\dagger = \lambda y_1:\bullet.\cdots\lambda y_{\Sigma(\mathtt{a})}:\bullet.\ \langle \mathtt{a}_0\rangle(L_{\Sigma(\mathtt{a})}\ y_1\dots y_{\Sigma(\mathtt{a})})$
$(\star)^\dagger = \bullet$
$(\kappa_1 \to \kappa_2)^\dagger = (\kappa_1)^\dagger \to (\kappa_2)^\dagger.$

As in the case for the translation from trees to formulas, we just need to replace each tree constructor a of arity ℓ with $\lambda y_1,\dots y_\ell.\langle \mathtt{a}_0\rangle(L_\ell\ y_1\dots y_\ell)$.

Example 9. *Consider the HORS of Example 2. Then its encoding as a HFL formula is defined by the following HES (notice that some β-reductions have been done to ease readability).*

$S =_\nu\quad F\ (\lambda x.\ \langle \mathtt{b}_0\rangle(L_1\ x));$
$F =_\nu\quad \lambda g.\ \langle \mathtt{a}_0\rangle\Big(L_2\ (\langle \mathtt{c}_0\rangle L_0)\ \big(g\,(F\,(B\,(\lambda x.\ \langle \mathtt{b}_0\rangle(L_1\,x)))))\Big);$
$B =_\nu\quad \lambda g.\lambda x.\ \langle \mathtt{b}_0\rangle(L_1\,(g\,x));$
$L_2 =_\nu \dots; L_1 =_\nu \dots; L_0 =_\nu \dots$

The following theorem states the correctness of the translation above. We omit the proof, since it is a special case of Theorem 10 given later.

Theorem 9. *For any trivial automaton $\mathcal{A}$ and HORS $\mathcal{G}$, $T_{\mathcal{G}} \in L(\mathcal{A})$ if and only if $\mathcal{L}_{\mathcal{A}} \models \mathcal{E}_{\mathcal{G}}$.*

3.3 The General Case

In the general case where $\mathcal{A}$ is an APT with priorities $\{0,\dots,p-1\}$, we need to take into account the parity acceptance condition and it must be reflected somehow in the resulting HFL formula. Let us first examine the case of an order-0 HORS. Assume $\mathcal{G}$ is a HORS where all non-terminals are of type $\star$ and all rules are of the form $A \to \mathtt{a}\ A_1\dots A_{\Sigma(\mathtt{a})}$. For each A, we prepare p fixpoint variables $A^{\sharp 0},\dots,A^{\sharp p-1}$, defined by

$$A^{\sharp i} =_{\alpha_i} \bigvee_{i'=0,\dots,p-1} \langle \mathtt{a}_{i'}\rangle(L_{\Sigma(\mathtt{a})}\ A_1^{\sharp i'}\dots A_{\Sigma(\mathtt{a})}^{\sharp i'}),$$

where α_i is ν if i is even and μ otherwise. As in the case of trivial automata, $A^{\sharp i}$ expresses the property that the current state has transitions corresponding to a accepting run of $\mathcal{A}$ over the tree generated by A; in addition, $A^{\sharp i}$ remembers that the priority of the previous state is i (this intuition will be refined later). The priority of the previous state of the automaton is recorded in the subscript of the transition label $\mathtt{a}_{i'}$, hence the above definition of $A^{\sharp i}$. If a priority i is visited infinitely often by the automaton, then a fixpoint variable of the form $A^{\sharp i}$ is unfolded infinitely often. Thus, by letting $\mathcal{E}_{\mathcal{G}}^{(p)} = (\mathcal{E}_{p-1};\dots,\mathcal{E}_0;\mathcal{E}_{aux})$ where $\mathcal{E}_i$ contains a declaration for $A^{\sharp i}$ of the above form and $\mathcal{E}_{aux}$ is as given in the previous section, we can guarantee that the largest priority visited by $\mathcal{A}$ is even if and only if the largest index of the fixpoint variables expanded infinitely often is even. We thus obtain $\mathcal{L}_{\mathcal{A}} \models \mathcal{E}_{\mathcal{G}}^{(p)}$ if and only if $T_{\mathcal{G}} \in L(\mathcal{A})$.

In the case of a HORS of an arbitrary order, each rule of the form $A \to C[A_1,\dots,A_k]$ should be replaced by a fixpoint equation of the form:

$$A^{\sharp i} =_{\alpha_i} C'[A_1^{\sharp i_1},\dots,A_k^{\sharp i_k}],$$

where each i_j is the largest priority visited since the unfolding of A before A_j is unfolded. The main difficulty arises when A_j occurs as an argument of another non-terminal, as in $A \to B\,A_j$. In this case, only B knows the largest priority visited before A_j is unfolded. Thus, we replicate the argument of B and translate $B\,A_j$ to $B^{\sharp 0}\ A_j^{\sharp 0}\ \cdots\ A_j^{\sharp p-1}$; here, $B^{\sharp 0}$ is defined so that it calls the i-th argument $A_j^{\sharp i}$ when the largest priority visited before unfolding A_j inside the body of B is i.

Let us present now the general construction of the HES $\mathcal{E}_{\mathcal{G}}^{(p)}$ encoding the HORS $\mathcal{G}$ for any alternating parity automaton with priorities in $\{0,\dots,p-1\}$. It is defined by $\mathcal{E}_{\mathcal{G}}^{(p)} := \mathcal{E}_{p-1};\dots;\mathcal{E}_0;\mathcal{E}_{aux}$ where for each non-terminal A and for each priority i, there is a definition $A^{\sharp i} =_{\alpha_i} (\mathcal{R}(A))^{\sharp 0}$ in $\mathcal{E}_i$, with $(.)^{\sharp(.)}$ to be defined soon, and again with $\alpha_i = \nu$ if i is even and μ otherwise.

For any term e and for any priority $i \in \{0,\dots,p-1\}$, let the formula $(e)^{\sharp i}$ be defined by induction on e as follows:

$(\lambda y:\kappa.e)^{\sharp i} = \lambda y^{\sharp 0}:\kappa^\sharp\dots\lambda y^{\sharp p-1}:\kappa^\sharp.\ e^{\sharp i}$
$(e_1\ e_2)^{\sharp i} = e_1^{\sharp i}\ e_2^{\sharp\max(0,i)}\ e_2^{\sharp\max(1,i)}\dots e_2^{\sharp\max(p-1,i)}$
$(z)^{\sharp i} = z^{\sharp i}$ if z is either a non-terminal or a variable
$(\mathtt{a})^{\sharp i} = \lambda y_1^{\sharp 0}:\bullet.\cdots\lambda y_1^{\sharp p-1}:\bullet.\cdots\lambda y_{\Sigma(\mathtt{a})}^{\sharp 0}:\bullet.\cdots\lambda y_{\Sigma(\mathtt{a})}^{\sharp p-1}:\bullet.$
$\quad\bigvee_{i'=0,\dots,p-1}\langle \mathtt{a}_{i'}\rangle(L_{\Sigma(\mathtt{a})}\ y_1^{\sharp i'}\dots y_{\Sigma(\mathtt{a})}^{\sharp i'})$
$(\star)^\sharp = \bullet$
$(\kappa_1 \to \kappa_2)^\sharp = \underbrace{(\kappa_1)^\sharp \to\cdots\to (\kappa_1)^\sharp}_{p\text{ times}} \to (\kappa_2)^\sharp$

where the L_n's definitions are as before and introduced in $\mathcal{E}_{aux}$. Intuitively, i in $(e)^{\sharp i}$ denotes the largest priority visited before the tree generated by e is visited (since the last unfolding of a non-terminal).

Example 10. *Consider the HORS $\mathcal{G}_1$ consisting of the rules:*

$$S \ \to\ F\,B \qquad F\,g \to \mathtt{a}\ \mathtt{c}\ (g\,(F\,g)) \qquad B\,x \to \mathtt{b}\,x,$$

which is a simpler variant of $\mathcal{G}_0$ in Example 2. It generates the regular tree T such that $T = \mathtt{a}\ \mathtt{c}\ (\mathtt{b}\,T)$. The HES $\mathcal{E}_{\mathcal{G}}^{(3)}$ is:

$$S^{\sharp 2} =_\nu \varphi_S; F^{\sharp 2} =_\nu \varphi_F; B^{\sharp 2} =_\nu \varphi_B;$$
$$S^{\sharp 1} =_\mu \varphi_S; F^{\sharp 1} =_\mu \varphi_F; B^{\sharp 1} =_\mu \varphi_B;$$
$$S^{\sharp 0} =_\nu \varphi_S; F^{\sharp 0} =_\nu \varphi_F; B^{\sharp 0} =_\nu \varphi_B; \mathcal{E}_{aux},$$

where

$$\begin{aligned}
&\varphi_S = F^{\sharp 0}\, B^{\sharp 0}\, B^{\sharp 1}\, B^{\sharp 2}\\
&\varphi_F = \lambda g^{\sharp 0}.\lambda g^{\sharp 1}.\lambda g^{\sharp 2}.\\
&\qquad\qquad \langle \mathtt{a}_0\rangle\, (L_2\, (\langle \mathtt{c}_0\rangle L_0 \vee \langle \mathtt{c}_1\rangle L_0 \vee \langle \mathtt{c}_2\rangle L_0)\, \varphi^{(0)}_{g(F\,g)})\\
&\qquad\qquad \vee\langle \mathtt{a}_1\rangle\, (L_2\, (\langle \mathtt{c}_0\rangle L_0 \vee \langle \mathtt{c}_1\rangle L_0 \vee \langle \mathtt{c}_2\rangle L_0)\, \varphi^{(1)}_{g(F\,g)})\\
&\qquad\qquad \vee\langle \mathtt{a}_2\rangle (L_2\, (\langle \mathtt{c}_0\rangle L_0 \vee \langle \mathtt{c}_1\rangle L_0 \vee \langle \mathtt{c}_2\rangle L_0)\, \varphi^{(2)}_{g(F\,g)})\\
&\varphi_B = \lambda x^{\sharp 0}.\lambda x^{\sharp 1}.\lambda x^{\sharp 2}.\\
&\qquad\qquad \langle \mathtt{b}_0\rangle (L_1\, x^{\sharp 0}) \vee \langle \mathtt{b}_1\rangle (L_1\, x^{\sharp 1}) \vee \langle \mathtt{b}_2\rangle (L_1\, x^{\sharp 2})\\
&\varphi^{(0)}_{g(F\,g)} = g^{\sharp 0}\, \varphi^{(0)}_{F\,g}\, \varphi^{(1)}_{F\,g}\, \varphi^{(2)}_{F\,g} \qquad \varphi^{(0)}_{F\,g} = F^{\sharp 0}\, g^{\sharp 0}\, g^{\sharp 1}\, g^{\sharp 2}\\
&\varphi^{(1)}_{g(F\,g)} = g^{\sharp 1}\, \varphi^{(1)}_{F\,g}\, \varphi^{(1)}_{F\,g}\, \varphi^{(2)}_{F\,g} \qquad \varphi^{(1)}_{F\,g} = F^{\sharp 1}\, g^{\sharp 1}\, g^{\sharp 1}\, g^{\sharp 2}\\
&\varphi^{(2)}_{g(F\,g)} = g^{\sharp 2}\, \varphi^{(2)}_{F\,g}\, \varphi^{(2)}_{F\,g}\, \varphi^{(2)}_{F\,g} \qquad \varphi^{(2)}_{F\,g} = F^{\sharp 2}\, g^{\sharp 2}\, g^{\sharp 2}\, g^{\sharp 2}.
\end{aligned}$$

For the LTS $\mathcal{L}_{\mathcal{A}_0}$ in Figure 2, we can remove irrelevant parts of the formulas φ_S,φ_F and φ_B and simplify[4] *them to:*

$$\begin{aligned}
&\varphi'_S = F^{\sharp 0}\, B^{\sharp 1}\, B^{\sharp 2}\\
&\varphi'_F = \lambda g^{\sharp 1}.\lambda g^{\sharp 2}.\\
&\qquad\qquad \langle \mathtt{a}_1\rangle\, (L_2\, (\langle \mathtt{c}_1\rangle L_0)\, (g^{\sharp 1}\, (F^{\sharp 1}\, g^{\sharp 1}\, g^{\sharp 2})\, (F^{\sharp 2}\, g^{\sharp 2}\, g^{\sharp 2})))\\
&\qquad\qquad \vee\langle \mathtt{a}_2\rangle (L_2\, (\langle \mathtt{c}_1\rangle L_0)\, (g^{\sharp 2}\, (F^{\sharp 2}\, g^{\sharp 2}\, g^{\sharp 2})\, (F^{\sharp 2}\, g^{\sharp 2}\, g^{\sharp 2})))\\
&\varphi'_B = \lambda x^{\sharp 1}.\lambda x^{\sharp 2}.(\langle \mathtt{b}_1\rangle (L_1\, x^{\sharp 1}) \vee \langle \mathtt{b}_2\rangle (L_1\, x^{\sharp 2})).
\end{aligned}$$

The simplified version of $S^{\sharp 2}$ can be expanded (with some further simplification) to:

$$\begin{aligned}
&\langle \mathtt{a}_1\rangle (L_2\, (\langle \mathtt{c}_1\rangle L_0)\\
&\qquad\quad (\langle \mathtt{b}_1\rangle (L_1 (\langle \mathtt{a}_2\rangle (L_2\, (\langle \mathtt{c}_1\rangle L_0)\\
&\qquad\qquad\qquad\qquad (\langle \mathtt{b}_1\rangle (L_1 (F^{\sharp 2}\, B^{\sharp 2}\, B^{\sharp 2}))))))))
\end{aligned}$$

and $F^{\sharp 2}\, B^{\sharp 2}\, B^{\sharp 2}$ may further be expanded to

$$\begin{aligned}
&\cdots \vee \langle \mathtt{a}_2\rangle (L_2\, (\langle \mathtt{c}_1\rangle L_0)\\
&\qquad\qquad (\langle \mathtt{b}_1\rangle (L_1 (F^{\sharp 2}\, B^{\sharp 2}\, B^{\sharp 2})) \vee \cdots)).
\end{aligned}$$

The LTS in Figure 2 satisfies this property; note that $F^{\sharp 2}$ is defined by one of the outermost fixpoint operators ν.

The correctness of the translation is stated in the theorem below. We prove it in Section 5, after preparing a type-based characterization of HFL model checking in Section 4.

Theorem 10. *Let $\mathcal{A}$ be an APT with priorities in $\{0, \ldots, p-1\}$, and let $\mathcal{G}$ be a HORS. Then $T_{\mathcal{G}} \in L(\mathcal{A})$ iff $\mathcal{L}_{\mathcal{A}} \models \mathcal{E}^{(p)}_{\mathcal{G}}$.*

It might be noticed that the size of $e^{\sharp i}$ is in $\mathcal{O}(p^{\mathsf{an}(e)}|e|)$, where p is the number of priorities, and $\mathsf{an}(e)$ is the nesting of applications inside arguments, defined via $\mathsf{an}(e_1\ e_2) = \max(\mathsf{an}(e_1), 1 + \mathsf{an}(e_2))$, $\mathsf{an}(\lambda y.e) = \mathsf{an}(e)$, and $\mathsf{an}(A) = \mathsf{an}(\mathtt{a}) = \mathsf{an}(y) = 0$. This exponential blow-up might seem prohibitive, but it is easy to avoid. Indeed, by introducing some extra non-terminals, any HORS can be rewritten into an equivalent one with a linear blow-up such that for all non-terminal A, $\mathsf{an}(\mathcal{R}(A)) \leq 2$.

Theorem 11. *For every HORS $\mathcal{G}$ and every $p \geq 1$, there is an HES $\mathcal{E}$ of size linear in the size of $\mathcal{G}$ and polynomial in p such that for any APT $\mathcal{A}$ with priorities in $\{0, \ldots, p-1\}$, $T_{\mathcal{G}} \in L(\mathcal{A})$ iff $\mathcal{L}_{\mathcal{A}} \models \mathcal{E}$. Furthermore, $\mathcal{E}$ can be constructed in time polynomial in the size of $\mathcal{G}$ and p.*

4. Intersection Types for HFL Model Checking

Inspired by Kobayashi and Ong's type system [12] for characterizing HORS model checking, this section develops a type system for characterizing HFL model checking. It is parameterized by an LTS $\mathcal{L}$, and an HFL formula φ that is typable in the type system if and only if $\mathcal{L} \models \varphi$. We shall use this type-based characterization for proving the correctness of the translation from HORS model checking to HFL model checking presented in Section 3 (Theorem 10). We expect that the type-based characterization is also useful for constructing a practical model checker for HFL.

We fix an LTS $\mathcal{L} = (U, A, \longrightarrow, s_{\mathsf{init}})$. We define the set of intersection types by:

$$\tau ::= s \mid \sigma \rightarrow \tau \qquad \sigma ::= \bigwedge\{\tau_1, \ldots, \tau_k\}.$$

Here, s ranges over the set U of states of $\mathcal{L}$. We often write $\tau_1 \wedge \cdots \wedge \tau_k$ or $\bigwedge_{i\in\{1,\ldots,k\}} \tau_i$ for $\bigwedge\{\tau_1, \ldots, \tau_k\}$, and $\top$ for $\bigwedge \emptyset$.

Intuitively, the type s describes propositions that are true in state s, and the type $\tau_1 \wedge \cdots \wedge \tau_k \rightarrow \tau$ describes functions that take formulas having type τ_i for every i, and return a formula of type τ. For example, the logical connective $\wedge$ (when viewed as a function that takes two propositions and returns a proposition) has type $s \rightarrow s \rightarrow s$ for any s, because given formulas φ_1 and φ_2 that are both true in state s, $\varphi_1 \wedge \varphi_2$ is also true in state s. Similarly, $\vee$ has types $s \rightarrow \top \rightarrow s$ and $\top \rightarrow s \rightarrow s$ for every $s \in U$.

Each intersection type should be regarded as a refinement of a simple type κ (constructed from $\bullet$ and $\rightarrow$, as introduced in Section 2.3). It does not make sense, for example, to consider an intersection type like $s \wedge (s_1 \rightarrow s_2)$, where the part s describes propositions whereas the part $s_1 \rightarrow s_2$ describes functions on propositions. To exclude such an ill-formed intersection type, we define the refinement relations $\tau :: \kappa$ (which should be read "τ is a refinement of κ") and $\sigma :: \kappa$ inductively using the following rules:

$$\frac{s \in U}{s :: \bullet} \qquad \frac{\tau_i :: \kappa \text{ for each } i \in \{1, \ldots, k\}}{\tau_1 \wedge \cdots \wedge \tau_k :: \kappa} \qquad \frac{\sigma :: \kappa \qquad \tau :: \kappa'}{(\sigma \rightarrow \tau) :: (\kappa \rightarrow \kappa')}$$

Henceforth, we consider only intersection types that are refinements of some simple types. We assume that each intersection type τ or σ is implicitly annotated with the corresponding simple type (i.e., κ such that $\tau :: \kappa$ or $\sigma :: \kappa$) and write $\mathit{Stype}(\tau)$ or $\mathit{Stype}(\sigma)$ for κ.[5]

We assume below that an HFL formula is given in the form of an HES

$$\mathcal{E} := (X_1 =_{\alpha_1} \varphi_1; \cdots ; X_n =_{\alpha_n} \varphi_n).$$

A type judgment for (fixpoint-free) HFL formulas is of the form $\Gamma \vdash \varphi : \tau$, where Γ, called an (intersection) *type environment*, is a set of type bindings of the form $X : \tau$. A type environment may contain multiple bindings for the same variable. We write $\Gamma(X)$ for $\tau_1 \wedge \cdots \wedge \tau_k$ if $\{\sigma \mid X : \sigma \in \Gamma(X)\} = \{\tau_1, \ldots, \tau_k\}$. The type judgment relation is inductively defined by the typing rules in Figure 4. Note that in the rules HFL-T-ABS and HFL-T-APP above, k may be 0.

Most of the typing rules should be easy to understand, based on the intuition that s is the type of a formula that is satisfied by the state s. For example, the rule HFL-T-SOME says that s satisfies $\langle a\rangle\varphi$ if there exists a state s' and a transition $s \xrightarrow{a} s'$ such that s' satisfies φ. The rules HFL-T-ABS and HFL-T-APP are the standard typing rules for abstractions and applications. The subtyping relation $\tau \leq \tau'$ means, as usual, that a value of type τ may also be used as a value of type τ'.

Example 11. *Consider the HES $\mathcal{E}_0$ of Example 6 and the LTS $\mathcal{L}_0$ of Example 4. Let $\Gamma = \{G : (s_0 \rightarrow s_1) \rightarrow s_0 \rightarrow s_1, G : (s_1 \rightarrow s_1) \rightarrow s_1 \rightarrow s_1, F : \big((s_0 \rightarrow s_1) \wedge (s_1 \rightarrow s_1)\big) \rightarrow s_0\}$. Then the type judgment $\Gamma \vdash \mathcal{E}(F) : \big((s_0 \rightarrow s_1) \wedge (s_1 \rightarrow s_1)\big) \rightarrow s_0$ holds (see the derivation in Figure 4).*

[4] For example, since the priority 0 does not occur, we can eliminate the first argument $g^{\sharp 0}$ of F. Similarly, we can also eliminate $\langle \mathtt{c}_2\rangle L_0$ from φ_F because the $\mathtt{c}_2$ transition cannot be taken at the end of a path labeled with $(\mathtt{a}_0 + \mathtt{a}_1)(1 + 2 + \mathtt{and} + \mathtt{or})^*$.

[5] Thus, for example, $\top$ is actually annotated like $\top^{\kappa}$. Without this assumption on the implicit annotation, $\mathit{Stype}(\top)$ cannot be determined.

$$\frac{s \in U}{\Gamma \vdash \top : s} \quad \text{(HFL-T-TRUE)}$$

$$\frac{}{\Gamma, X : \tau \vdash X : \tau} \quad \text{(HFL-T-VAR)}$$

$$\frac{s \xrightarrow{a} s' \qquad \Gamma \vdash \varphi : s'}{\Gamma \vdash \langle a \rangle \varphi : s} \quad \text{(HFL-T-SOME)}$$

$$\frac{\Gamma \vdash \varphi : s' \text{ for every } s' \text{ such that } s \xrightarrow{a} s'}{\Gamma \vdash [a]\varphi : s} \quad \text{(HFL-T-ALL)}$$

$$\frac{\Gamma \vdash \varphi_1 : s \qquad \Gamma \vdash \varphi_2 : s}{\Gamma \vdash \varphi_1 \wedge \varphi_2 : s} \quad \text{(HFL-T-AND)}$$

$$\frac{\Gamma \vdash \varphi_i : s \text{ for some } i \in \{1,2\}}{\Gamma \vdash \varphi_1 \vee \varphi_2 : s} \quad \text{(HFL-T-OR)}$$

$$\frac{\begin{array}{c}\Gamma, X : \tau_1, \dots, X : \tau_k \vdash \varphi : \tau \qquad X \notin \mathtt{dom}(\Gamma) \\ \tau_i :: \eta \text{ for each } i \in \{1, \dots, k\}\end{array}}{\Gamma \vdash \lambda X : \eta . \varphi : \tau_1 \wedge \dots \wedge \tau_k \to \tau} \quad \text{(HFL-T-ABS)}$$

$$\frac{\begin{array}{c}\Gamma \vdash \varphi_1 : \tau_1 \wedge \dots \wedge \tau_k \to \tau \\ \Gamma \vdash \varphi_2 : \tau_i \text{ for each } i \in \{1, \dots, k\}\end{array}}{\Gamma \vdash \varphi_1 \varphi_2 : \tau} \quad \text{(HFL-T-APP)}$$

$$\frac{\Gamma \vdash \varphi : \tau \qquad \tau \leq \tau'}{\Gamma \vdash \varphi : \tau'} \quad \text{(HFL-T-SUB)}$$

$$\frac{}{s \leq s} \quad \text{(HFL-SUBT-BASE)}$$

$$\frac{\sigma' \leq \sigma \qquad \tau \leq \tau'}{\sigma \to \tau \leq \sigma' \to \tau'} \quad \text{(HFL-SUBT-FUN)}$$

$$\frac{\forall j \in \{1, \dots, \ell\}. \exists i \in \{1, \dots, k\}. \tau_i \leq \tau'_j}{\tau_1 \wedge \dots \wedge \tau_k \leq \tau'_1 \wedge \dots \wedge \tau'_\ell} \quad \text{(HFL-SUBT-INT)}$$

Figure 3. Typing rules for HFL formulas.

For an entire formula (represented in the form of an HES), we define typability in terms of a parity game.

Let $\mathtt{dep}(\mathcal{E})$ be the number of switches between ν and μ:

$$\mathtt{dep}(\epsilon) = 0$$

$$\mathtt{dep}(F =_\nu \varphi; \mathcal{E}) = \begin{cases} \mathtt{dep}(\mathcal{E}) & \text{if } \mathtt{dep}(\mathcal{E}) \text{ is even} \\ \mathtt{dep}(\mathcal{E}) + 1 & \text{if } \mathtt{dep}(\mathcal{E}) \text{ is odd} \end{cases}$$

$$\mathtt{dep}(F =_\mu \varphi; \mathcal{E}) = \begin{cases} \mathtt{dep}(\mathcal{E}) & \text{if } \mathtt{dep}(\mathcal{E}) \text{ is odd} \\ \mathtt{dep}(\mathcal{E}) + 1 & \text{if } \mathtt{dep}(\mathcal{E}) \text{ is even} \end{cases}$$

The priority of F_i in $\mathcal{E}$, written $\Omega_\mathcal{E}(F_i)$ is defined as $\mathtt{dep}(F_i =_{\alpha_i} \varphi_i; \mathcal{E}_2)$ if $\mathcal{E} = (\mathcal{E}_1; F_i =_{\alpha_i} \varphi_i; \mathcal{E}_2)$. For example, for the HES $\mathcal{E}_\perp$ of Example 7, $\Omega_{\mathcal{E}_\perp}(S) = 3$, $\Omega_{\mathcal{E}_\perp}(Y) = 2$, and $\Omega_{\mathcal{E}_\perp}(X) = 1$. When $\mathcal{E}$ is clear from context, we omit the subscript and just write $\Omega(F_i)$.

Definition 12. *Let $\mathcal{E} := (F_1^{\eta_1} =_{\alpha_1} \varphi_1; \cdots ; F_n^{\eta_n} =_{\alpha_n} \varphi_n)$ be a fixpoint-free HES with $\eta_1 = \bullet$, and $\mathcal{L} = (U, A, \longrightarrow, s_{\mathsf{init}})$ an LTS. The* typability game $\mathbf{TG}(\mathcal{L}, \mathcal{E})$ *is the parity game $(V_\forall, V_\exists, v_{\mathsf{init}}, E, \Omega)$, where:*

- *The set $V_\forall$ of Opponent's positions is the set of intersection type environments $\{\Gamma \mid \mathtt{dom}(\Gamma) \subseteq \{F_1, \dots, F_n\} \wedge \forall (F_i : \tau) \in \Gamma. \tau :: \eta_i\}$.*
- *The set $V_\exists$ of Player's positions is the set of type bindings that respect simple types, i.e., $\{F_i : \tau \mid \tau :: \eta_i\}$.*
- *v_{init} is the initial position $F_1 : s_{\mathsf{init}}$.*
- *$E = E_1 \cup E_2$, where E_1, the set of Player's moves, is $\{(F_i : \tau, \Gamma) \mid \Gamma \vdash \varphi_i : \tau\}$; and E_2, the set of Opponent's moves, is $\{(\Gamma, F_i : \tau) \mid F_i : \tau \in \Gamma\}$.*
- *The priority function Ω, is defined by: $\Omega(\Gamma) = 0$ for every $\Gamma \in V_\forall$, and $\Omega(F_i : \tau) = \Omega_\mathcal{E}(F_i)$ for every $F_i : \tau \in V_\exists$.*

We write $\mathcal{L} \vdash \mathcal{E}$ when Player wins the parity game $\mathbf{TG}(\mathcal{L}, \mathcal{E})$.

Intuitively, in the game $\mathbf{TG}(\mathcal{L}, \mathcal{E})$ Player tries to prove that $\mathcal{L} \models \mathcal{E}$, and Opponent tries to disprove it. To this end, Player first shows that φ_1, the righthand side of F_1, has type s_{init} (i.e., the initial state of $\mathcal{L}$ satisfies φ_1) under some type environment Γ, and Opponent challenges it by picking a type binding $F_j : \tau$ from Γ, and asking why F_j has type τ. Player then shows that φ_j has type τ under some type environment Γ', and Opponent again challenges the assumption Γ', etc. Opponent gets stuck when Player's assumption Γ' is empty, in which case Player wins; Player gets stuck when she fails to show why φ_j has type τ, in which case Opponent wins. A play may continue indefinitely, in which case the winner is determined by the largest priority visited infinitely often.

Example 12. *Consider again the HES $\mathcal{E}_0$ of Example 6 and the LTS $\mathcal{L}_0$ of Example 4. Let Γ be like in Example 11. Then Player has a winning strategy by always moving to the type environment Γ or the empty type environment (in which case Player wins).*

- *In the first round, Player is in position $S : s_0$, but it holds that $\Gamma \vdash \mathcal{E}(S) : s_0$, so Player can move to Γ.*
- *In any next round, Player is in a successor position of Γ chosen by Opponent, i.e. some type binding $A : \tau$ of Γ. If A is either G or B, Player can respond with the empty type environment, because $\emptyset \vdash \mathcal{E}(A) : \tau$. Otherwise, Player is on position $F : \tau_F$ with $\tau_F = \big((s_0 \to s_1) \wedge (s_1 \to s_1)\big) \to s_0$. We saw that $\Gamma \vdash F : \tau_F$ holds in Example 11, so Player is allowed to move to Γ.*

Since the only infinite play according to this strategy is the one where Player's position (except the initial position) is always $F : \tau_F$, and since F has priority 0, Player's strategy is a winning one.

Example 13. *Consider the HFL formula φ_2 in Example 5, which is equivalent to the following HES $\mathcal{E}_2$:*

$$\begin{array}{l} S =_\mu E\,(F\,B)\,([b]\bot); \\ E =_\mu \lambda X. \lambda Y. (X \wedge Y) \vee E\,(\langle a \rangle X)\,(\langle b \rangle Y); \\ F =_\nu \lambda X. \langle a \rangle (X (F\,(G\,X)));\ G =_\nu \lambda X. \lambda Y. \langle b \rangle (X\,Y); \\ B =_\nu \lambda Y. \langle b \rangle Y. \end{array}$$

Then, we have:

$$\begin{array}{l} E : s_0 \to s_0 \to s_0, F : ((s_0 \to s_1) \wedge (s_1 \to s_1)) \to s_0, \\ \qquad\qquad B : s_0 \to s_1, B : s_1 \to s_1 \vdash \mathcal{E}_2(S) : s_0 \\ \emptyset \vdash \mathcal{E}_2(E) : s_0 \to s_0 \to s_0 \\ \Gamma \vdash \mathcal{E}_2(F) : ((s_0 \to s_1) \wedge (s_1 \to s_1)) \to s_0 \\ \emptyset \vdash \mathcal{E}_2(G) : (s_0 \to s_1) \to s_0 \to s_1 \\ \emptyset \vdash \mathcal{E}_2(G) : (s_1 \to s_1) \to s_1 \to s_1 \\ \emptyset \vdash \mathcal{E}_2(B) : s_0 \to s_1 \qquad \emptyset \vdash \mathcal{E}_2(B) : s_1 \to s_1 \end{array}$$

where Γ is the one given in Example 11. These type judgments determine a winning strategy for Player.

Example 14. *Consider the unsatisfiable HES $\mathcal{E}_\perp$ of Example 7; recall that $\Omega_{\mathcal{E}_\perp}(S) = 3$, $\Omega_{\mathcal{E}_\perp}(Y) = 2$, and $\Omega_{\mathcal{E}_\perp}(X) = 1$. Let $\mathcal{L} = (\{s\}, \{a\}, \longrightarrow, s)$ with $s \xrightarrow{a} s$. A strategy for Player in $\mathbf{TG}(\mathcal{L}, \mathcal{E}_\perp)$ is to always play $\Gamma = \{X : s, Y : s \to s\}$. This strategy can be seen as a cyclic type derivation that is depicted in Figure 5. It is not a winning strategy: the dashed cycle has the largest priority 2, but the self loop on $X : s$ (depicted with a thick line) has the largest priority 1, hence Opponent can force an infinite play with the largest priority 1.*

We now prove that the type-based characterization is sound and complete.

Theorem 13 (soundness and completeness of the type-based characterization)**.** *Let $\mathcal{E}$ be a fixpoint-free HES and $\mathcal{L}$ an LTS. Then, $\mathcal{L} \vdash \mathcal{E}$ if and only if $\mathcal{L} \models \mathcal{E}$.*

$$\dfrac{\dfrac{\dfrac{X : s_0 \to s_1 \vdash X : s_0 \to s_1 \qquad \dfrac{\Gamma \vdash F : \big((s_0 \to s_1) \wedge (s_1 \to s_1)\big) \to s_0 \quad \dfrac{\begin{array}{c}\Gamma \vdash G : (s_0 \to s_1) \to s_0 \to s_1 \\ X : s_0 \to s_1 \vdash X : s_0 \to s_1\end{array}}{\Gamma, X : s_0 \to s_1 \vdash G\,X : s_0 \to s_1} \quad \dfrac{\begin{array}{c}\Gamma \vdash G : (s_1 \to s_1) \to s_1 \to s_1 \\ X : s_1 \to s_1 \vdash X : s_1 \to s_1\end{array}}{\Gamma, X : s_1 \to s_1 \vdash G\,X : s_1 \to s_1}}{\Gamma, X : s_0 \to s_1, X : s_1 \to s_1 \vdash F(G\,X) : s_0}}{\Gamma, X : s_0 \to s_1, X : s_1 \to s_1 \vdash X(F(G\,X)) : s_1}}{\Gamma, X : s_0 \to s_1, X : s_1 \to s_1 \vdash \langle \mathsf{a} \rangle \big(X(F(G\,X))\big) : s_0}}{\Gamma \vdash \lambda X.\, \langle \mathsf{a} \rangle \big(X(F(G\,X))\big) : \big((s_1 \to s_1) \wedge (s_0 \to s_1)\big) \to s_0}$$

Figure 4. Type derivation for $\Gamma \vdash \mathcal{E}(F) : \big((s_1 \to s_1) \wedge (s_0 \to s_1)\big) \to s_0$ in Example 11.

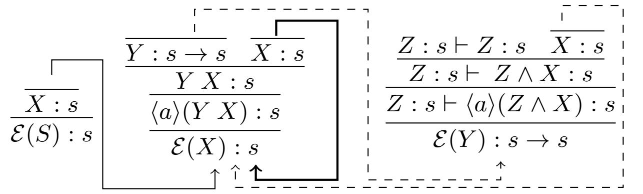
$X : s$
$\mathcal{E}(S) : s$
$Y : s \to s$
$X : s$
$Y\,X : s$
$\langle a \rangle (Y\,X) : s$
$\mathcal{E}(X) : s$
$Z : s \vdash Z : s$
$X : s$
$Z : s \vdash\ Z \wedge X : s$
$Z : s \vdash \langle a \rangle (Z \wedge X) : s$
$\mathcal{E}(Y) : s \to s$

Figure 5. A Player's strategy in the typing game of Example 14.

The proof of the above theorem is given in the longer version [16]; here we just give an outline. The proof uses a semantic counterpart $\mathbf{SG}(\mathcal{L}, \mathcal{E})$ of the typability game, which is obtained from $\mathbf{TG}(\mathcal{L}, \mathcal{E})$ by replacing the player's moves $\{(F_i : \tau, \Gamma) \mid \Gamma \vdash \varphi_i : \tau\}$ with $\{(F_i : \tau, \Gamma) \mid \Gamma \models \varphi_i : \tau\}$, where $\Gamma \models \varphi_j : \tau$ is a semantic type judgment relation. Since $\Gamma \vdash \varphi_j : \tau$ if and only $\Gamma \models \varphi_j : \tau$, the semantic typability game $\mathbf{SG}(\mathcal{L}, \mathcal{E})$ is actually isomorphic to the (syntactic) typability game $\mathbf{TG}(\mathcal{L}, \mathcal{E})$. We can then transform the semantic typability game step by step, preserving the winner, until we get the semantic typability game for the *extended* HES (where fixpoint binders may occur in definitions) consisting of the single equation $F_1 = \mathit{toHFL}(\mathcal{E})$. Because $\mathbf{SG}(\mathcal{L}, F_1 = \mathit{toHFL}(\mathcal{E}))$ is winning for Player if and only if $\mathcal{L} \models \mathcal{E}$, we have the required result.

As a corollary of Theorem 13, we also have the following parameterized complexity result.

Theorem 14. *Let $\mathcal{E}$ be a HES and $\mathcal{L}$ an LTS. Suppose that the following parameters are bounded above by constants: (i) the depth of $\mathcal{E}$; (ii) the size of the largest (simple) type in $\mathcal{E}$; and (iii) the size of $\mathcal{L}$ (i.e., the number of states plus the size of the transition relation $\overset{?}{\longrightarrow}$). Then, $\mathcal{L} \models \mathcal{E}$ can be decided in time polynomial in the size of $\mathcal{E}$.*

The theorem follows from the same reasoning as that for the parameterized complexity result for HORS model checking [12]. Under the assumption above, for each variable of type η, the number of intersection types τ such that $\tau :: \eta$ is bounded above by a constant. Thus, the size of each type environment in the typability game is linear in the size of $\mathcal{E}$, hence also is the size of the typability game. By the assumption that the depth $\mathtt{dep}(\mathcal{E})$ is fixed, the game can be solved in time polynomial in the size of the game, hence also in the size of $\mathcal{E}$.

5. Correctness of the HORS-to-HFL Reduction (Proof of Theorem 10)

In this section, we establish the correctness of the HORS-to-HFL reduction (Theorem 13) we presented in Section 3. The proof relies on the type-based characterization of HORS model-checking based on Kobayashi and Ong's type system [12] (KO type system, for short). Below we first briefly review KO type system in Section 5.1. Then we show that the typability of a HORS model-checking instance in the KO type system is equivalent to the typability of its HFL translation in the type system of Section 4.

5.1 KO Type System

We review here (a variation of) KO type system for characterizing HORS model checking [14]. We fix an alternating parity automaton $\mathcal{A} = (Q, \Sigma, \delta, q_{\mathsf{init}}, \Omega)$. KO types are defined by the grammar

$$\theta ::= q \mid \varsigma \to \theta \qquad \varsigma ::= \bigwedge\{(\theta_1, m_1), \dots, (\theta_k, m_k)\}.$$

Here, q ranges over the set Q of states of the automaton, and m_j ranges over the set $\{0, \dots, p-1\}$ of priorities of $\mathcal{A}$. As in the case of intersection types for HFL, we often write $(\theta_1, m_1) \wedge \dots \wedge (\theta_k, m_k)$ or $\bigwedge_{i \in \{1,\dots,k\}} (\theta_i, m_i)$ for $\wedge\{(\theta_1, m_1), \dots, (\theta_k, m_k)\}$, and $\top$ for $\bigwedge \emptyset$.

Intuitively, q is the type of a tree that is accepted by $\mathcal{A}$ when q is taken as the initial state, whereas $\varsigma_1 \to \dots \varsigma_n \to q$ with $\varsigma_j = (\theta_{j,1}, m_{j,1}) \wedge \dots \wedge (\theta_{j,k_j}, m_{j,k_j})$ is the type of an n-ary function that may use the j-th argument as a value of types $\theta_{j,1}, \dots, \theta_{j,k_j}$ and generates a tree of type q. The part $m_{j,\ell}$ ($\ell \in \{1, \dots, k_j\}$) expresses *where* the j-th argument may be used as a value of type $\theta_{j,\ell}$; intuitively, $(\theta_{j,\ell}, m_{j,\ell})$ specifies that in constructing the output tree of type q, the j-th argument may be used as a value of type $\theta_{j,\ell}$ in a node of the tree in which the largest priority visited in the path from the root to this node is $m_{j,\ell}$. For the space restriction, we refer the reader to [14] for more intuitions on KO types. A slight difference between the original KO type system and the one presented here is that by "the largest priority visited in the path from the root", we exclude the priority of the current node, whereas the original type system included it. This change is just for technical convenience for matching the HFL type system in the previous section with KO type system.

As in the type system of Section 4, we only consider KO types ς that are refinements of simple types κ (which we write $\varsigma :: \kappa$, defined in a similar manner as in Section 4), and the empty intersection type that refines κ is written $\top^{\kappa}$ or just $\top$ when κ is not meaningful. A type environment is a set Θ of bindings $x : (\varsigma, m)$ where x is either a non-terminal or a term variable, and m is a priority.

The typing rules of KO type system are given in Figure 6. In the rule KO-T-CONST, the relation $\mathbf{Q} \models f$ (where $f \in \mathsf{B}^+(\{1, \dots, n\} \times Q)$ and $\mathbf{Q} = (Q_1, \dots, Q_n)$ with $Q_i \subseteq Q$ for each i) is defined by induction on f: (i) $\mathbf{Q} \models \mathtt{tt}$, (ii) $\mathbf{Q} \not\models \mathtt{ff}$, (iii) $\mathbf{Q} \models (i, q)$ if ($q \in Q_i$), (iv) $\mathbf{Q} \models f_1 \vee f_2$ if $\mathbf{Q} \models f_1$ or $\mathbf{Q} \models f_2$, and (v) $\mathbf{Q} \models f_1 \wedge f_2$ if $\mathbf{Q} \models f_1$ and $\mathbf{Q} \models f_2$. The operation $\cdot\!\uparrow_m$ on type environments is defined by:

$$\Theta\!\uparrow_m := \{x : (\theta, \max(m, m')) \mid x : (\theta, m') \in \Theta\}.$$

The KO typability game $\mathbf{KG}(\mathcal{G}, \mathcal{A})$ for a HORS $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ and an APT $\mathcal{A} = (Q, \Sigma, \delta, q_{\mathsf{init}}, \Omega)$ is a parity game $(V_\forall, V_\exists, v_{\mathsf{init}}, E, \Omega')$, where:

- The set $V_\forall$ of Opponent's positions is the set of intersection type environments $\{\Theta \mid \forall (F_i : \theta) \in \Theta. \theta :: \mathcal{N}(F_i)\}$.

$$\frac{}{x:(\theta,0) \vdash^{\texttt{HORS}}_{\mathcal{A}} x:\theta} \quad \text{(KO-T-VAR)}$$

$$\frac{(Q_1,\dots,Q_n) \models \delta_{\mathcal{A}}(q,a)}{\emptyset \vdash^{\texttt{HORS}}_{\mathcal{A}} a : \bigwedge_{q_1\in Q_1}(q_1,\Omega(q)) \to \cdots \to \bigwedge_{q_n \in Q_n}(q_n,\Omega(q)) \to q} \quad \text{(KO-T-CONST)}$$

$$\frac{\Theta_0 \vdash^{\texttt{HORS}}_{\mathcal{A}} e_0 : \bigwedge_{i\in I}(\theta_i,m_i) \to \theta \qquad \Theta_i \vdash^{\texttt{HORS}}_{\mathcal{A}} e_1 : \theta_i \text{ for each } i \in I}{\Theta_0 \cup \bigcup_{i\in I}(\Theta_i{\uparrow}_{m_i}) \vdash^{\texttt{HORS}}_{\mathcal{A}} e_0\ e_1 : \theta} \quad \text{(KO-T-APP)}$$

$$\frac{\Theta \cup \{x:(\theta_i,m_i) \mid i\in I\} \vdash^{\texttt{HORS}}_{\mathcal{A}} e:\theta \qquad x \text{ does not occur in } \Theta}{\Theta \vdash^{\texttt{HORS}}_{\mathcal{A}} \lambda x.e : \bigwedge_{i\in I}(\theta_i,m_i)\to\theta} \quad \text{(KO-T-ABS)}$$

$$\frac{\Theta \vdash^{\texttt{HORS}}_{\mathcal{A}} e:\theta \qquad \theta \le \theta'}{\Theta \vdash^{\texttt{HORS}}_{\mathcal{A}} e:\theta'} \quad \text{(KO-T-SUB)}$$

$$\frac{}{q \le q} \quad \text{(KO-SUBT-BASE)}$$

$$\frac{\theta \le \theta' \qquad \forall i \in I.\exists j \in J.(\theta'_j \le \theta_i \wedge m'_j = m_i)}{\bigwedge_{i\in I}(\theta_i,m_i) \to \theta \le \bigwedge_{j\in J}(\theta'_j,m'_j)\to\theta'} \quad \text{(KO-SUBT-FUN)}$$

Figure 6. KO type system.

- The set $V_\exists$ of Player's positions is the set of type bindings that respect simple types, i.e., $\{F_i : (\theta,m) \mid \tau :: \mathcal{N}(F_i)\}$.
- v_{init} is the initial position $F : (q_{\mathsf{init}}, \Omega(q_{\mathsf{init}}))$.
- $E = E_1 \cup E_2$, where E_1, the set of Player's moves, is $\{(F_i : (\theta,m),\Theta) \mid \Theta \vdash \mathcal{R}(F_i) : \theta\}$; and E_2, the set of Opponent's moves, is $\{(\Theta, F_i : (\theta,m)) \mid F_i : (\theta,m) \in \Theta\}$.
- The priority function Ω', is defined by: $\Omega'(\Theta) = 0$ for every $\Theta \in V_\forall$, and $\Omega'(F_i : (\theta,m)) = m$ for every $F_i : (\theta,m) \in V_\exists$.

We write $\vdash^{\texttt{HORS}}_{\mathcal{A}} \mathcal{G}$ if Player has a winning strategy for $\mathbf{KG}(\mathcal{G},\mathcal{A})$.

The following theorem states the soundness and completeness of KO type system.[6]

Theorem 15 (Kobayashi and Ong [14]). *Suppose that $T_{\mathcal{G}}$ does not contain $\perp$. Then, $\vdash^{\texttt{HORS}}_{\mathcal{A}} \mathcal{G}$ if and only if $T_{\mathcal{G}} \in L(\mathcal{A})$.*

5.2 Preservation of the Typability

Fix a HORS $\mathcal{G}$ and an APT $\mathcal{A}$ with the set $\{0,\dots,p-1\}$ of priorities. We want to relate the typing game for $\mathbf{KG}(\mathcal{G},\mathcal{A})$ to the typing game $\mathbf{TG}(\mathcal{L}_{\mathcal{A}}, \mathcal{E}^{(p)}_{\mathcal{G}})$. To avoid confusion, we write below $\Gamma \vdash^{\texttt{HFL}} \varphi : \tau$ for the type judgment in HFL.

We first identify types for HFL model-checking (Section 4) and KO types. We define the translation $(\cdot)^\sharp$ of KO types to the types in Section 4.

$$\begin{aligned}
&(q)^\sharp = q\\
&(\textstyle\bigwedge_{j\in J}(\theta_j,m_j) \to \theta)^\sharp =\\
&\qquad \textstyle\bigwedge_{j\in J, m_j=0}(\theta_j)^\sharp \to \cdots \to \bigwedge_{j\in J, m_j = p-1}(\theta_j)^\sharp \to (\theta)^\sharp
\end{aligned}$$

For example, with $p = 2$, $((q_0,0)\wedge(q_1,0)\wedge(q_1,1) \to q)^\sharp = q_0 \wedge q_1 \to q_1 \to q$. Note that $\theta :: \kappa$ implies $(\theta)^\sharp :: \kappa^\sharp$, and that for any HFL intersection type τ, there is a KO type θ such that $\tau = (\theta)^\sharp$ if and only if $\tau :: \kappa^\sharp$ for some κ, and in that case, θ is unique. We write $(\tau)^\flat$ for this θ. In particular, it holds that

$$\begin{aligned}
&(q)^\flat = q\\
&(\textstyle\bigwedge_{j\in I_0}\tau_j \to \cdots \to \bigwedge_{j\in I_{p-1}}\tau_j \to \tau)^\flat =\\
&\qquad \textstyle\bigwedge_{i\in\{0,\dots,p-1\}, j\in I_i}((\tau_j)^\flat, i) \to (\tau)^\flat.
\end{aligned}$$

We extend $(.)^\sharp$ and $(.)^\flat$ to type environments:

$$\begin{aligned}
&(\Gamma)^\flat = \{A : ((\tau)^\flat, i) \mid A^{\sharp i} : \tau \in \Gamma\}\\
&(\Theta)^\sharp = \{A^{\sharp i} : (\theta)^\sharp \mid A : (\theta,i) \in \Theta\} \cup \Gamma_{aux},
\end{aligned}$$

where $\Gamma_{aux} = \{L_n : \bigwedge_{q_1\in Q_1} q_1 \to \cdots \bigwedge_{q_n \in Q_n} q_n \to f \mid (Q_1,\dots,Q_n) \models f\}$ is the type environment for all L_n. Note that $(\Gamma)^\flat$ is well defined for the type environments used in the typing game of $\mathbf{TG}(\mathcal{L}_{\mathcal{A}}, \mathcal{E}^{(p)}_{\mathcal{G}})$ because it only contains bindings $A^{\sharp i} : \tau$ for intersection types τ that refine a type of the form $\kappa^\sharp$.

We can show that the transformation preserves typing.

Lemma 16. *Let e be a term of a HORS. If $\Theta \vdash^{\texttt{HORS}} e : \theta$, then $(\Theta)^\sharp \vdash^{\texttt{HFL}} e^{\sharp 0} : (\theta)^\sharp$. Conversely, if $\Gamma \vdash^{\texttt{HFL}} e^{\sharp 0} : \tau$, then $(\Gamma)^\flat \vdash^{\texttt{HORS}} e : (\tau)^\flat$.*

The following lemma guarantees that $L_n : \tau \in \Gamma_{aux}$ if and only if it is a winning position of $\mathbf{TG}(\mathcal{L}_{\mathcal{A}}, \mathcal{E}^{(p)}_{\mathcal{G}})$.

Lemma 17. *Let f be a subformula of $\delta(q,a)$ with $\Sigma(a) = n$, and $Q_1,\dots,Q_n \subseteq Q$. Then $\vdash^{\texttt{HFL}} L_n : \bigwedge_{q\in Q_1} q \to \bigwedge_{q \in Q_2} q \to \cdots \to \bigwedge_{q\in Q_n} q \to f$ is a winning position of the HFL typability game if and only if $(Q_1,\dots,Q_n) \models f$.*

We can now prove that the reduction preserves typability.

Theorem 18. *Let $\mathcal{G}$ be a HORS and $\mathcal{A}$ be an alternating parity tree automaton. Then, $\vdash^{\texttt{HORS}}_{\mathcal{A}} \mathcal{G}$ if and only if $\mathcal{L}_{\mathcal{A}} \vdash^{\texttt{HFL}} \mathcal{E}^{(p)}_{\mathcal{G}}$.*

Proof. Let $\mathbf{G}$ be the parity game obtained from $\mathbf{TG}(\mathcal{L}_{\mathcal{A}}, \mathcal{E}^{(p)}_{\mathcal{G}})$ by removing Player's positions of the form $L_n{:}\tau$, and the edges from/to those positions. By Lemma 17, the winners of $\mathbf{TG}(\mathcal{L}_{\mathcal{A}}, \mathcal{E}^{(p)}_{\mathcal{G}})$ and $\mathbf{G}$ are the same.

Notice that $(.)^\sharp$ and $(.)^\flat$ are bijections between the positions of $\mathbf{G}$ and the ones of $\mathbf{KG}(\mathcal{G},\mathcal{A})$. By Lemma 16, these bijections are graph isomorphisms between the graphs of the arenas of the games. Moreover, the priority of every Opponent's position is 0 in both games, and for Player's positions, $\Omega(x^{\sharp m} : \tau) = m = \Omega\big(x : (\tau, m)\big)$ holds. So both games are isomorphic. $\square$

Theorem 10 is an immediate corollary of Theorems 13, 15, and 18.

Remark 2. *As mentioned in Section 1, since the decidability of HFL model checking is straightforward, the decidability of HORS model checking is an immediate corollary of Theorem 10. Our proof of Theorem 10 in this section, however, does not qualify as a new proof of the decidability of HORS model checking, because it relies on the soundness and completeness of the KO type system.*

6. From HFL to HORS Model Checking

In this section, we present a reduction from HFL model checking to HORS model checking.

Recall that, over a (finite) LTS, by the Kleene Fixpoint Theorem, any fixpoint formula $\alpha F^\eta.\psi$ with $\alpha \in \{\mu,\nu\}$ and $\eta = \eta_1 \to \cdots \to \eta_\ell \to \bullet$ is equivalent to F^n where

$$F^0 = \begin{cases} \lambda x_1 : \eta_1.\cdots\lambda x_\ell : \eta_\ell.\top & \text{if } \alpha = \nu \\ \lambda x_1 : \eta_1.\cdots\lambda x_\ell : \eta_\ell.\perp & \text{if } \alpha = \mu \end{cases}$$

$$F^{i+1} = [F^i/F]\psi$$

[6] The type system presented in this section is actually a slight variation of the original one [14], but the proof in [14] can be easily adapted to this variation.

and n is greater than the height of the lattice of D_η. For η of order k, this height is a number k-fold exponential in the number of states of the LTS. Precise bounds can be found in [2]. Our aim is to create a HORS that generates the syntax tree of $F^{(n)}$, and then runs it against an alternating automaton that encodes the LTS in question.

6.1 Overview of the Translation

We first give an overview of the translation using an example. Let us consider the following HES $\mathcal{E}$:

$$S =_\nu F\ (\langle a\rangle\top); \quad F\ X =_\mu X \vee \langle b\rangle(F\ X).$$

It represents the property that the action a may be enabled after finitely many b transitions. For a sufficiently large number n, $\mathcal{E}$ is equivalent to the following HES $\mathcal{E}'$, obtained by unfolding F n times.

$$\begin{array}{l} S =_\nu F^{(n)}\ (\langle a\rangle\top); \\ F^{(n)}\ X =_\mu X \vee \langle b\rangle(F^{(n-1)}\ X); \\ \cdots \\ F^{(1)}\ X =_\mu X \vee \langle b\rangle(F^{(0)}\ X); \\ F^{(0)}\ X =_\mu \bot. \end{array}$$

The annotations ν and μ in $\mathcal{E}'$ above actually do not matter, because $\mathcal{E}'$ does not contain any recursion. Now, by replacing each logical connective with the corresponding tree constructor, we obtain the following HORS $\mathcal{G}_\mathcal{E}$, which generates the syntax tree of the formula obtained by reducing $\mathcal{E}'$:

$$\begin{array}{l} S \to F^{(n)}\ (\langle a\rangle\top) \\ F^{(n)}\ X \to \vee\ X\ (\langle b\rangle(F^{(n)}\ X)) \\ \cdots \\ F^{(1)}\ X \to \vee\ X\ (\langle b\rangle(F^{(0)}\ X)) \\ F^{(0)}\ X \to \bot. \end{array}$$

Let $\mathcal{L} = (U, A, \longrightarrow, s_{\mathsf{init}})$ be an LTS. To check whether $\mathcal{L} \models \mathcal{E}'$ (hence also $\mathcal{L} \models \mathcal{E}$) holds, it suffices to run a tree automaton to evaluate (the formula represented by) the tree $T_{\mathcal{G}_\mathcal{E}}$ against $\mathcal{L}$. Such an automaton $\mathcal{A}_\mathcal{L}$ would be of the form $(\{q_s \mid s \in U\}, \Sigma, \delta, q_{s_{\mathsf{init}}}, \Omega)$ where q_s is a state for checking whether s satisfies the formula represented by the current subtree, the alphabet Σ consists of the tree constructors corresponding to logical connectives, and the transition function δ is defined by:[7]

$$\begin{array}{l} \delta(q_s, \top) = \mathtt{tt} \quad \delta(q_s, \bot) = \mathtt{ff} \quad \delta(q_s, \vee) = (1, q_s) \vee (2, q_s) \\ \delta(q_s, \langle a\rangle) = \vee\{(1, q_{s'} \mid s \xrightarrow{a} s'\} \qquad \cdots . \end{array}$$

Then, we have $\mathcal{L} \models \mathcal{E}$ if and only if $\mathcal{G}_\mathcal{E} \models \mathcal{A}_\mathcal{L}$; thus we have reduced HFL model checking to HORS model checking.

The remaining problem is that $\mathcal{G}_\mathcal{E}$ is too large, because the required number n of unfoldings is in general k-fold exponential in the size of $\mathcal{L}$ for an order-k HES. To address the problem, we parameterize each non-terminal $F^{(j)}$ above by the number j, and encode numbers as terms of HORS. Thus, the resulting HORS is given by:

$$\begin{array}{l} S \to F\ n\ (\langle a\rangle\top) \\ F\ j\ X \to \mathtt{if}\ (\mathtt{IsZero}\ j)\ \bot\ (\vee\ X\ (\langle b\rangle(F\ (j-1)\ X))). \end{array}$$

Below, we first prepare an encoding of numbers in Section 6.2. We then present the general translation from HFL model checking to HORS model checking in Section 6.3.

6.2 Counting with HORS

As a first step, we show how to implement large numbers in HORS. Our encoding follows that of Jones [6]. Let $\mathbf{exp}_k(r)$ denote the k-fold exponent of r, defined by $\mathbf{exp}_0(r)$ and $\mathbf{exp}_{i+1}(r) = 2^{\mathbf{exp}_i(r)}$.

[7] The full definition is given later in Section 6.3.

For our purpose, we need to represent numbers up to $\mathbf{exp}_k(r)$ by terms of order at most $k-1$ and of size polynomial in r. Prepare $\mathbf{Bit} = \{\mathtt{0}, \mathtt{1}\}$ and let $\mathbf{Num}_i$ be defined by

$$\begin{array}{l} \mathbf{Num}_1 = \underbrace{\mathbf{Bit} \times \cdots \times \mathbf{Bit}}_{r} \\ \mathbf{Num}_{i+1} = \mathbf{Num}_i \to \mathbf{Bit}. \end{array}$$

For every i, let $[\![.]\!]_i : \{0, \ldots, \mathbf{exp}_i(r) - 1\} \to \mathbf{Num}_i$ be the bijection defined as follows: (i) for every $n \in \{1, \ldots, 2^r - 1\}$, $[\![n]\!]_1 = (b_0, \ldots, b_{r-1})$, where $b_0 \ldots b_{r-1}$ is the binary representation of n starting with b_0 as the least significant bit; (ii) for every $n \in \{0, \ldots, \mathbf{exp}_{i+1}(r) - 1\}$, for every $m \in \{0, \ldots, \mathbf{exp}_i(r) - 1\}$ $[\![n]\!]_{i+1}$ maps $[\![m]\!]_i$ to b_m, where $b_0 \ldots b_{\mathbf{exp}_i(r)-1}$ is the binary representation of n.

In order to compute with bits, we represent bit expressions as $\Sigma_{\mathbf{Bit}}$-labeled (possibly infinite) trees where $\Sigma_{\mathbf{Bit}} = \{\mathtt{1} \mapsto 0, \mathtt{0} \mapsto 0, \mathtt{if} \mapsto 3\}$. We define the relation $T \Downarrow b$ inductively, by: (i) $\mathtt{1} \Downarrow \mathtt{1}$, (ii) $\mathtt{0} \Downarrow \mathtt{0}$, (iii) $\mathtt{if}\ T_0\ T_1\ T_2 \Downarrow b$ if $T_0 \Downarrow \mathtt{1}$ and $T_1 \Downarrow b$, and (iv) $\mathtt{if}\ T_0\ T_1\ T_2 \Downarrow b$ if $T_0 \Downarrow \mathtt{0}$ and $T_2 \Downarrow b$. We call b the value of T when $T \Downarrow b$ holds. Note that a bit expression T may or may not have a value if T is infinite.

We prepare an automaton to evaluate bit expressions. Let $\mathcal{A}^{\mathbf{Bit}}$ be the APT $(\{q_1, q_0\}, \Sigma_{\mathbf{Bit}}, \delta, q_1, \Omega)$, with

$$\begin{array}{l} \delta(q, \mathtt{if}) = ((1, q_1) \wedge (2, q)) \vee ((1, q_0) \wedge (3, q)) \\ \qquad\qquad\qquad \text{for every } q \in \{q_1, q_0\} \\ \delta(q_1, \mathtt{1}) = \delta(q_0, \mathtt{0}) = \mathtt{tt} \\ \delta(q_1, \mathtt{0}) = \delta(q_0, \mathtt{1}) = \mathtt{ff} \\ \Omega(q_1) = \Omega(q_0) = 1. \end{array}$$

Lemma 19. *$\mathcal{A}^{\mathbf{Bit}}$ accepts a tree T from state q_1 (q_0, resp.) if and only if $T \Downarrow \mathtt{1}$ ($T \Downarrow \mathtt{0}$, resp.).*

We assume below that other bit operations are represented as order-1 non-terminals of HORS. For example, the bit complement $\mathtt{Not}$ and ℓ-ary disjunction $\mathtt{OR}_\ell$ can be defined by the following rewriting rules:

$$\begin{array}{l} \mathtt{Not}\ x \to \mathtt{if}\ x\ \mathtt{0}\ \mathtt{1} \\ \mathtt{OR}_1\ x \to x \qquad \mathtt{OR}_\ell\ x_1\ \cdots\ x_\ell \to \mathtt{if}\ x_1\ \mathtt{1}\ (\mathtt{OR}_{\ell-1}\ x_2\ \cdots\ x_\ell) \end{array}$$

We introduce the HORS types $\mathbf{Bit}^\star = \star$ and $\mathbf{Num}_i^\star$ for all $i \geq 2$ as follows: $\mathbf{Num}_2^\star = \underbrace{\star \to \cdots \to \star}_{r} \to \star$, and for all $i \geq 2$, $\mathbf{Num}_{i+1}^\star = \mathbf{Num}_i^\star \to \star$ (note that $\mathbf{Num}_1^\star$ is undefined only because HORS types do not have product).

For the purpose of encoding HFL formulas, we need to prepare the following terms of HORS:

$\mathtt{Max}_i : \mathbf{Num}_i^\star$	(which represents $\mathbf{exp}_i(r) - 1$)
$\mathtt{Dec}_i : \mathbf{Num}_i^\star \to \mathbf{Num}_i^\star$	(decrement function)
$\mathtt{IsZero}_i : \mathbf{Num}_i^\star \to \mathbf{Bit}^\star$	(check if the argument is 0)

for all $i \geq 2$. They are defined as follows, using the auxiliary functions $\mathtt{ExistsOne}_i$ and $\mathtt{DecSub}_j$:

$$
\begin{array}{l}
\mathtt{Max}_1 \equiv (\mathtt{1}, \dots, \mathtt{1}) \qquad \mathtt{Max}_{i+1}\ g \to \mathtt{1} \\
\mathtt{Dec}_1\ (b_0, \dots, b_{r-1}) \equiv \\
\qquad\qquad (\mathtt{DecSub}_0\ b_0, \dots, \mathtt{DecSub}_{r-1}\ b_0\ \cdots\ b_{r-1}) \\
\mathtt{DecSub}_0\ b_0 \to \mathtt{Not}\ b_0 \\
\mathtt{DecSub}_j\ b_0\ \cdots\ b_j \to \\
\quad \text{(* Flip } b_j \text{ only if } b_0, \dots, b_{j-1} \text{ are all 0 *)} \\
\qquad \mathtt{if}\ (\mathtt{OR}_j\ b_0\ \cdots\ b_{j-1})\ b_j\ (\mathtt{Not}\ b_j) \\
\mathtt{Dec}_{i+1}\ f\ n \to \\
\quad \text{(* Flip the } n\text{-th bit of } f \text{ only if all the lower bits are 0.*)} \\
\qquad \mathtt{if}\ (\mathtt{ExistsOne}_{i+1}\ f\ n)\ (f\ n)\ (\mathtt{Not}(f\ n)) \\
\mathtt{ExistsOne}_{i+1}\ f\ n \to \\
\quad \text{(* Check whether some bit of } f \text{ lower than the n-th bit is 0 *)} \\
\qquad \mathtt{if}\ (\mathtt{IsZero}_i\ n)\ \mathtt{0} \\
\qquad\qquad (\mathtt{OR}_2\ (f\ (\mathtt{Dec}_i\ n))\ \big(\mathtt{ExistsOne}_{i+1}\ f\ (\mathtt{Dec}_i\ n)\big)) \\
\mathtt{IsZero}_1\ (b_0, \dots, b_{r-1}) \to \mathtt{Not}(\mathtt{OR}_r\ b_0\ \cdots\ b_{r-1}) \\
\mathtt{IsZero}_{i+1}\ f \to \mathtt{Not}(\mathtt{OR}_2\ (f\ \mathtt{Max}_i)\ (\mathtt{ExistsOne}_{i+1}\ f\ \mathtt{Max}_i)).
\end{array}
$$

Here, $\equiv$ indicates that the lefthand side is a shorthand (or a macro) for the righthand side, and $\to$ indicates that the head symbol on the lefthand side is a non-terminal of HORS defined by the rewriting rule. The meta-variable i ranges over $\{1, \dots, k-1\}$, and j ranges over $\{1, \dots, r\}$. The encodings above should be easy to understand; $\mathtt{Max}_i$ represents the number whose bit representation is $\underbrace{11 \cdots 1}_{\mathbf{exp}_{i-1}(r)}$, hence defined as a function that always returns $\mathtt{1}$.

The following lemma states the correctness of our number encoding.

Lemma 20. *Let T be the tree generated by $\mathtt{IsZero}_i(\mathtt{Dec}_i^m\ \mathtt{Max}_i)$. Then, (i) if $m = \mathbf{exp}_i(r) - 1$, then $T \Downarrow \mathtt{1}$; (ii) if $m < \mathbf{exp}_i(r) - 1$, then $T \Downarrow \mathtt{0}$.*

6.3 The Translation

Let $\mathcal{L}$ be an LTS $(U, A, \longrightarrow, s_{\mathsf{init}})$, and $\mathcal{E}$ be an order-k HES $F_n =_{\alpha_n} \varphi_n; \cdots ; F_0 =_{\alpha_0} \varphi_0$ where F_i is of type η_i (and thus $\eta_n = \bullet$). We assume that each φ_j is of the form $\lambda x_1. \cdots \lambda x_{\ell_j}.\psi_j$ such that ψ_j does not contain lambda abstractions.

Let h_j be the height of the lattice of D_{η_j}, and M the largest arity of types occurring in $\eta_0, \dots, \eta_n$. By [2], Lemma 3.5, $\mathbf{exp}_k(r) - 1 \geq \max(h_0, \dots, h_n)$ for $r > \log |U| + |U| \cdot (M + k)^k$. Let $\mathbf{mh}$ be $\mathbf{exp}_k(r) - 1$ for the least such natural number r. Note that r is polynomial in $|U|$ and M, assuming that the order k of $\mathcal{E}$ is a constant.

Let $\beta = (\beta_n, \dots, \beta_j)$ be a collection of non-negative integers. If $\beta_j > 0$, define

$$
\begin{array}{ll}
\beta(\ell) = (\beta_n, \dots, \beta_\ell) & \text{if } \ell > j \\
\beta(\ell) = (\beta_n, \dots, \beta_j - 1, \underbrace{\mathbf{mh}, \dots, \mathbf{mh}}_{j-\ell \text{ times}}) & \text{if } \ell \leq j
\end{array}
$$

Let $<$ be the lexicographic order on β's, i.e., the least transitive relation that satisfies: $(\beta_n, \dots, \beta_{j+1}) < (\beta_n, \dots, \beta_{j+1}, \beta_j)$ and $(\beta_n, \dots, \beta_{j+1}, \beta_j) < (\beta_n, \dots, \beta_{j+1}, \beta_j + 1)$. We define the HFL formula $F_j^{(m_n, \dots, m_j)}$ for each $j \in \{0, \dots, n\}, m_n, \dots, m_j \in \{0, \dots, \mathbf{mh}\}$ as follows, by well-founded induction on $<$.

$$
\begin{array}{l}
F_j^{(m_n, \dots, m_{j+1}, 0)} = \lambda x_1. \cdots \lambda x_{\ell_j}.\widehat{\alpha_j} \\
F_j^\beta = [F_0^{\beta(0)}/F_0, \dots, F_n^{\beta(n)}/F_n]\varphi_j \\
\qquad \text{if } \beta = (m_n, \dots, m_j) \text{ with } m_j > 0.
\end{array}
$$

Here, $\widehat{\alpha_j} = \top$ if $\alpha_j = \nu$ and $\widehat{\alpha_j} = \bot$ if $\alpha_j = \mu$. By the Kleene Fixpoint Theorem, we have:

Lemma 21. $[\![\mathit{toHFL}(\mathcal{E})]\!] = [\![F_n^{(\mathbf{mh})}]\!]$.

Since $F_n^{(\mathbf{mh})}$ contains no fixpoint operators, we can reduce it to a formula in basic modal logic. Below we create a HORS that generates the syntax tree of this formula.

For each $F_j (j \in \{0, \dots, n\})$ of $\mathcal{E}$, we prepare a non-terminal of the same name F_j of a HORS, and the following rewriting rule:

$$
\begin{array}{l}
F_j\ y_n, \dots, y_j, x_1, \dots, x_{\ell_j} \to \\
\qquad \mathtt{if}\ (\mathtt{IsZero}_k\ y_j)\ \widehat{\alpha_j}\ ([\![\psi_j]\!]_{y_n, \dots, y_{j+1}, \mathtt{Dec}_k(y_j)}).
\end{array}
$$

Here, $[\![\psi_j']\!]_{y_n, \dots, y_j}$ is defined by induction on formulas:

$$
\begin{array}{l}
[\![c]\!]_{y_n, \dots, y_j} = c \qquad [\![x_\ell]\!]_{y_n, \dots, y_j} = x_\ell \\
[\![F_\ell]\!]_{y_n, \dots, y_j} = \begin{cases} F_\ell\, y_n\ \dots\ y_\ell & \text{if } \ell \geq j \\ F_\ell\, y_n\ \dots\ y_j\ \underbrace{\mathtt{Max}_k\ \dots\ \mathtt{Max}_k}_{j-l \text{ times}} & \text{if } \ell < j \end{cases} \\
[\![\varphi_1 \varphi_2]\!]_{y_n, \dots, y_j} = [\![\varphi_1]\!]_{y_n, \dots, y_j}\ [\![\varphi_2]\!]_{y_n, \dots, y_j}
\end{array}
$$

Here, c ranges over $\vee, \wedge, \langle a \rangle, [a], \top, \bot$; so, for example, $\varphi_1 \wedge \varphi_2$ is considered as $(\wedge\ \varphi_1)\ \varphi_2$ in the above definition. In the image of the translation, those constants are treated as tree constructors of the HORS. The arguments $y_1, \dots, y_j$ are of type $\mathbf{Num}_k^\star$; intuitively, $F_j\ [\![n_1]\!]_k\ \cdots\ [\![n_j]\!]_k$ corresponds to $F_j^{(n_1, \dots, n_j)}$.

We write $\mathcal{G}_{\mathcal{E},\mathcal{L}}$[8] for the HORS consisting of the above rules for F_j, $S \to F_n\ \mathtt{Max}_k$ (where S is the start symbol), and the rules in Section 6.2 for encoding numbers.

Example 15. *Recall the LTS $\mathcal{L}_0$ from Example 4, and the HES $\mathcal{E}_0$ from Example 6:*

$$
\begin{array}{l}
S =_\nu F\, B; \quad F =_\nu \lambda X : \bullet \to \bullet.\langle a \rangle (X (F\, (G\ X))); \\
G =_\nu \lambda X : \bullet \to \bullet.\lambda Y : \bullet.\langle b \rangle (X\, Y); \quad B =_\nu \lambda Y : \bullet.\langle b \rangle Y.
\end{array}
$$

We obtain the HORS $\mathcal{G}_{\mathcal{E}_0,\mathcal{L}_0}$ with

$$
\begin{array}{l}
S' \to S\ \mathtt{Max}_2 \\
S\ y_S \to \mathtt{if}\ (\mathtt{IsZero}_2\ y_S)\ \top \\
\qquad\qquad (F\, (\mathtt{Dec}_2\, y_S)\, \mathtt{Max}_2\, (B\, y_S\ \mathtt{Max}_2\ \mathtt{Max}_2\ \mathtt{Max}_2)) \\
F\ y_S\ y_F\ x \to \\
\qquad \mathtt{if}\ (\mathtt{IsZero}_2\ y_F)\ \top \\
\qquad (\langle a \rangle\ (x\, (F\, y_S\ (\mathtt{Dec}_2\, y_F)\ (G\, y_S\ (\mathtt{Dec}_2\, y_F)\, \mathtt{Max}_2\ x)))) \\
G\ y_S\ y_F\ y_G\ x\ y \to \mathtt{if}\ (\mathtt{IsZero}_2\ y_G)\ \top\ (\langle b \rangle\ (x\ y)) \\
B\ y_S\ y_F\ y_G\, y_B\ y \to \mathtt{if}\ (\mathtt{IsZero}_2\ y_B)\ \top\ (\langle b \rangle\ y)
\end{array}
$$

where the y_j's have been renamed to their respective nonterminal for ease of understanding and the parameters x_j have been renamed to lower case versions of their HFL correspondents, and the rules for $\mathtt{Dec}_2$ and $\mathtt{IsZero}_2$ are as per their definition. □

Let $\mathcal{A}_\mathcal{L}$ be the APT $(\{q_s \mid s \in U\} \cup \{q_1, q_0\}, \Sigma, \delta, q_{s_{\mathsf{init}}}, \Omega)$ where:

$$
\begin{array}{l}
\Sigma = \Sigma_{\mathbf{Bit}} \cup \{\vee \mapsto 2, \wedge \mapsto 2, \top \mapsto 0, \bot \mapsto 0\} \\
\qquad \cup \bigcup_{a \in A} \{\langle a \rangle \mapsto 1, [a] \mapsto 1\} \\
\delta(q_s, \langle a \rangle) = \vee\{(1, q_{s'}) \mid s \xrightarrow{a} s'\} \\
\delta(q_s, [a]) = \wedge\{(1, q_{s'}) \mid s \xrightarrow{a} s'\} \\
\delta(q_s, \top) = \mathtt{tt} \qquad \delta(q_s, \bot) = \mathtt{ff} \\
\delta(q_s, \vee) = (1, q_s) \vee (2, q_s) \qquad \delta(q_s, \wedge) = (1, q_s) \wedge (2, q_s) \\
\delta(q_s, \mathtt{1}) = \delta(q_s, \mathtt{0}) = \mathtt{ff} \qquad\qquad (\text{for each } s \in U) \\
\delta(q, \mathtt{if}) = ((1, q_1) \wedge (2, q)) \vee ((1, q_0) \wedge (3, q)) \\
\qquad (\text{for every } q \in \{q_s \mid s \in U\} \cup \{q_1, q_0\}) \\
\delta(q_1, \mathtt{1}) = \mathtt{tt} \qquad \delta(q_1, a) = \mathtt{ff} \text{ if } a \notin \{\mathtt{1}, \mathtt{if}\} \\
\delta(q_0, \mathtt{0}) = \mathtt{tt} \qquad \delta(q_0, a) = \mathtt{ff} \text{ if } a \notin \{\mathtt{0}, \mathtt{if}\}
\end{array}
$$

and $\Omega(q) = 1$ for every q. Note that $\mathcal{A}_\mathcal{L}$ is an extension of the automaton $\mathcal{A}^{\mathbf{Bit}}$ in the previous subsection.

Theorem 22. *Let $\mathcal{L}$ be an LTS and let $\mathcal{E}$ be an HES. Then $\mathcal{A}_\mathcal{L}$ accepts the tree generated by $\mathcal{G}_{\mathcal{E},\mathcal{L}}$ if and only if $\mathcal{L} \models \mathcal{E}$. The size*

[8] The only dependence of $\mathcal{G}_{\mathcal{E},\mathcal{L}}$ on $\mathcal{L}$ is via r.

of $\mathcal{G}_{\mathcal{E},\mathcal{L}}$ is polynomial in the size of $\mathcal{E}$ and $\mathcal{L}$; and $\mathcal{A}_{\mathcal{L}}$ has $m + 2$ states where m is the number of states of $\mathcal{L}$. Furthermore, they can be constructed in time polynomial in the size of $\mathcal{E}$ and $\mathcal{L}$ (assuming that the order k of $\mathcal{E}$ is a constant).

By the above theorem, the reduction combined with an optimal algorithm for HORS model checking yields an k-EXPTIME HFL model checking algorithm, which is optimal [2].

7. Related Work

The model checking problem for HORS has been studied since around 2000. Knapik et al. [7] proved the decidability of the problem for HORS with the safety restriction, and Ong [24] proved the decidability for arbitrary HORS, without the safety restriction and showed that the problem is k-EXPTIME complete for order-k HORS. Since Ong's proof was complex, a number of alternative proofs have been developed since then [5, 12, 27, 31]. Among others, Kobayashi and Ong [11, 12] have provided a type-based characterization of HORS model checking, which inspired our type system for HFL model checking in Section 4. The type-based characterization of HORS model checking has lead to development of practical algorithms for HORS model checking [3, 9, 10, 23, 26]. We therefore expect that our type-based characterization of HFL model checking also yields practical algorithms for HFL model checking. The proof of the correctness of our type-based characterization (found in the longer version [16]) has been partially inspired by Salvati and Walukiewicz's model theoretic approach to HORS model checking [28]. On the practical side, HORS model checking has been applied to automated verification of higher-order programs [8, 15, 17, 18, 22, 25, 32, 34].

Independently of the above line of work, Viswanathan and Viswanathan [33] introduced HFL, a higher-order extension of modal μ-calculus, and showed that, while model checking remains decidable for finite state systems, HFL is strictly more expressive than modal μ-calculus and FLC (Modal Fixpoint Logic with Chop) [21], another extension of modal μ-calculus. Axelsson et al. [2] proved that the model checking problem for order-k HFL formulas is k-EXPTIME complete. The state of the art on practical algorithms for HFL model checking is much behind that on HORS model checking algorithms. In [19], the authors sketch a global model-checking algorithm that does not compute the entire representation of functions, but relies on neededness analysis in order to partially represent them. By contrast, the typing game presented in this paper may be seen as a higher-order extension of *local* model-checking [30].

Somewhat surprisingly, despite that both problems are higher-order extensions of finite state model checking that have been introduced and studied in the 2000's, and despite that both are k-EXPTIME complete for the order-k fragment, we are not aware of any previous work that studies the connection between HORS and HFL model checking. The translation from HORS to HFL in Section 3 has been partially inspired by Kobayashi and Ong's type system for HORS model checking [12]. Their type system statically keeps track of the largest priority of states visited using types, whereas our translation dynamically keeps that information by duplicating arguments. This fact is reflected in the translation from their types to our types for HFL presented in Section 3. The translation from HORS to HFL model checking may also have some connection to Salvati and and Walukiewicz's recent work [29], which uses a model-theoretic approach to reduce HORS model checking to nested least/greatest fixpoint computations. In the translation from HFL to HORS, the key challenge was how to encode big numbers into order-$(k-1)$ terms of HORS. Our encoding may be seen as a combination of Jones' encoding of big numbers as functions [6], and encoding of Boolean expressions into order-0 terms (with an added automaton to evaluate these expressions); the latter encoding was used in the benchmark of the HORS model checker PREFACE [26].

8. Conclusion

We have presented mutual translations between the HORS and HFL model checking problems, both higher-order extensions of finite state model checking. We have also proved the correctness of both translations. These translations preserve complexity, in the sense that the translation followed by an optimal algorithm for the target problem yields an optimal (i.e., k-EXPTIME) algorithm for the source problem. The results reveal the close connection between the two problems, enabling the cross-fertilization of the two threads of research. The type-based characterization of HFL model checking developed in Section 4 may be seen as the first outcome of such cross-fertilization, which may yield a practical algorithm for HFL model checking.

Acknowledgments

We would like to thank Martin Lange, Takeshi Tsukada, and anonymous referees for useful comments. This work was partially supported by JSPS KAKENHI Grant Number JP15H05706.

References

[1] K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(3), 2007.

[2] R. Axelsson, M. Lange, and R. Somla. The complexity of model checking higher-order fixpoint logic. *Logical Methods in Computer Science*, 3(2), 2007.

[3] C. H. Broadbent and N. Kobayashi. Saturation-based model checking of higher-order recursion schemes. In *Proceedings of CSL 2013*, volume 23 of *LIPIcs*, pages 129–148. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[4] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

[5] M. Hague, A. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *Proceedings of LICS 2008*, pages 452–461. IEEE Computer Society, 2008.

[6] N. D. Jones. The expressive power of higher-order types or, life without CONS. *Journal of Functional Programming*, 11(1):5–94, 2001.

[7] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002.

[8] N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proceedings of POPL 2009*, pages 416–428. ACM, 2009.

[9] N. Kobayashi. Model-checking higher-order functions. In *Proceedings of PPDP 2009*, pages 25–36. ACM, 2009.

[10] N. Kobayashi. A practical linear time algorithm for trivial automata model checking of higher-order recursion schemes. In *Proceedings of FoSSaCS 2011*, volume 6604 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2011.

[11] N. Kobayashi. Model checking higher-order programs. *Journal of the ACM*, 60(3), 2013.

[12] N. Kobayashi and C.-H. L. Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of LICS 2009*, pages 179–188. IEEE Computer Society, 2009.

[13] N. Kobayashi and C.-H. L. Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. *Logical Methods in Computer Science*, 7(4), 2011.

[14] N. Kobayashi and C.-H. L. Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. A re-

vised and extended version of [12]. `http://www-kb.is.s.u-tokyo.ac.jp/~koba/papers/lics09-full.pdf`, 2012.

[15] N. Kobayashi, R. Sato, and H. Unno. Predicate abstraction and CEGAR for higher-order model checking. In *Proceedings of PLDI 2011*, pages 222–233. ACM, 2011.

[16] N. Kobayashi, É. Lozes, and F. Bruse. On the relationship between higher-order recursion schemes and higher-order fixpoint logic, 2016. A longer version, available from the first author's web page.

[17] T. Kuwahara, T. Terauchi, H. Unno, and N. Kobayashi. Automatic termination verification for higher-order functional programs. In *Proceedings of ESOP 2014*, volume 8410 of *Lecture Notes in Computer Science*, pages 392–411. Springer, 2014.

[18] T. Kuwahara, R. Sato, H. Unno, and N. Kobayashi. Predicate abstraction and CEGAR for disproving termination of higher-order functional programs. In *Proceedings of CAV 2015*, volume 9207 of *Lecture Notes in Computer Science*, pages 287–303. Springer, 2015.

[19] M. Lange, É. Lozes, and M. V. Guzmán. Model-checking process equivalences. *Theor. Comput. Sci.*, 560:326–347, 2014.

[20] É. Lozes. A type-directed negation elimination. In R. Matthes and M. Mio, editors, *Proceedings of Tenth International Workshop on Fixed Points in Computer Science (FICS 2015)*, volume 191 of *EPTCS*, pages 132–142, 2015.

[21] M. Müller-Olm. A modal fixpoint logic with chop. In *Proceedings of STACS 99*, volume 1563 of *Lecture Notes in Computer Science*, pages 510–520. Springer, 1999.

[22] A. Murase, T. Terauchi, N. Kobayashi, R. Sato, and H. Unno. Temporal verification of higher-order functional programs. In *Proceedings of POPL 2016*, pages 57–68. ACM, 2016.

[23] R. P. Neatherway, S. J. Ramsay, and C.-H. L. Ong. A traversal-based algorithm for higher-order model checking. In *Proceedings of ICFP '12*, pages 353–364, 2012.

[24] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *Proceedings of LICS 2006*, pages 81–90. IEEE Computer Society, 2006.

[25] C.-H. L. Ong and S. Ramsay. Verifying higher-order programs with pattern-matching algebraic data types. In *Proceedings of POPL 2011*, pages 587–598. ACM, 2011.

[26] S. Ramsay, R. Neatherway, and C.-H. L. Ong. An abstraction refinement approach to higher-order model checking. In *Proceedings of POPL 2014*, pages 61–72. ACM, 2014.

[27] S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In *Proceedings of ICALP 2011*, volume 6756 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 2011.

[28] S. Salvati and I. Walukiewicz. Typing weak MSOL properties. In *Proceedings of FoSSaCS 2015*, volume 9034 of *Lecture Notes in Computer Science*, pages 343–357. Springer, 2015.

[29] S. Salvati and I. Walukiewicz. A model for behavioural properties of higher-order programs. In *Proceedings of CSL 2015*, volume 41 of *LIPIcs*, pages 229–243. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[30] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89(1):161–177, 1991.

[31] T. Tsukada and C. L. Ong. Compositional higher-order model checking via ω-regular games over böhm trees. In *Proceedings of CSL-LICS '14*, pages 78:1–78:10. ACM, 2014.

[32] H. Unno, T. Terauchi, and N. Kobayashi. Automating relatively complete verification of higher-order functional programs. In *Proceedings of POPL 2013*, pages 75–86. ACM, 2013.

[33] M. Viswanathan and R. Viswanathan. A higher order modal fixed point logic. In *Proceedings of CONCUR 2004*, volume 3170 of *Lecture Notes in Computer Science*, pages 512–528. Springer, 2004.

[34] K. Yasukata, N. Kobayashi, and K. Matsuda. Pairwise reachability analysis for higher order concurrent programs by higher-order model checking. In *Proceedings of CONCUR 2014*, volume 8704 of *Lecture Notes in Computer Science*, pages 312–326. Springer, 2014.