On the relationship between higher-order recursion schemes and higher-order fixpoint logic
Naoki Kobayashi, Etienne Lozes, Florian Bruse

To cite this version:
Naoki Kobayashi, Etienne Lozes, Florian Bruse. On the relationship between higher-order recursion schemes and higher-order fixpoint logic. the 44th ACM SIGPLAN Symposium, Jan 2017, Paris, France. 10.1145/3009837. hal-01920615

HAL Id: hal-01920615
https://hal.archives-ouvertes.fr/hal-01920615
Submitted on 13 Nov 2018
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) [8, 12, 25], which asks, given a higher-order recursion scheme $G$ (which is a kind of a tree grammar) and a formula $\varphi$ of the modal $\mu$-calculus (or equivalently, an alternating parity tree automaton), whether the tree generated by $G$ satisfies $\varphi$. The other is higher-order modal fixpoint logic model checking of finite state systems (HFL model checking, for short) [34], which asks, given a finite state system $L$ and a formula $\varphi$ of the higher-order modal fixpoint logic (which is a higher-order extension of the modal $\mu$-calculus), whether $L$ satisfies $\varphi$. 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 [9, 16, 18, 19, 23, 26, 33, 35]. HFL model checking has been applied to assume-guarantee reasoning [34] and process equivalence checking [20]. 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 $\mu$-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. 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 [12, 13] and has been used for studies of practical algorithms [3, 10, 11, 24, 27], parameterized complexity [13, 14], decidability proofs [13, 32], etc. of HORS model checking. Our type-based characterization of HFL model checking is similar to (and actually simpler than) that for

---

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 [6, 13, 25, 28]) 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. Proofs omitted in the main text are found in Appendix.

2. Preliminaries

In this section, we first recall, in Section 2.1, the standard definitions of (infinite) trees, parity games and alternating games (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 $\mathbb{N}_+ = \{0, 1, 2, \ldots\}$ be the set of positive integers. Given a set $L$, an $L$-labeled tree $T$ is a partial map from integer labels to $L$ such that $\forall n \in \mathbb{N}_+, \forall i \in \mathbb{N}_+, \pi \cdot i \in \text{dom}(T) \implies \{\pi \cdot 0, \ldots, \pi \cdot (i - 1)\} \subseteq \text{dom}(T)$. An element of $\text{dom}(T)$ is called a node. For $n, n' \in \text{dom}(T)$, $n'$ is a child of $n$ if $n$ is the longest strict prefix of $n'$.

A ranked alphabet $\Sigma$ is a map from a finite set of symbols to the set of non-negative integers, called arities. A $\Sigma$-labeled tree $T$ is a ranked tree if for every node $n \in \text{dom}(T)$, the number of children of $n$ is $|\Sigma|$. A parity game is a two player game played by Player and Opponent and is defined by a tuple $G = (V_T, V_2, v_{\text{init}}, E, \Omega)$, where $V_T, V_2$ are disjoint sets of positions, $v_{\text{init}} \in V_T \cup V_2$ is the initial position, $E \subseteq (V_T \cup V_2)^2$ is a set of moves, and $\Omega : V_T \cup V_2 \to \{0, \ldots, p - 1\}$ assigns to each position a priority. Positions in $V_T$ are called Player’s positions, and positions in $V_2$ are called Opponent’s positions.

A play is a finite or infinite sequence of positions $v_0, v_1, \ldots$ such that $v_0 = v_{\text{init}}$ 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 $V_T$ is an Opponent’s position such that $v_n \in E(= \{v \mid (v_n, v) \in E\}) = \emptyset$, 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 $v \in V_T$ (Opponent’s moves remain unchanged), and for all $v \in V_2$, 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 $(V_T, V_2, v_{\text{init}}$, $E \cap W, \Omega)$ are won by Player.

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

$$B^+(X) \ni f := \text{tt} \mid \text{ff} \mid x \mid f_1 \lor f_2 \mid f_1 \land f_2,$$

where $x$ ranges over $X$.

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

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

Given an APT $A$ and a $\Sigma$-labeled ranked tree $T$, the acceptance game $G(T, A) = (V_T, V_2, v_{\text{init}}, E, \Omega)$ is the parity game defined by $V_T \cup V_2 := \{(n, f) \mid n \in \text{dom}(T), f\}$ is a subformula of $\delta(q, a)$ for some $(q, a) \in Q \times \text{dom}(\Sigma)$, with $(n, f) \in V_T$ iff $f$ is a conjunction or $tt$, $v_{\text{init}} := (\epsilon, \delta(q_{\text{init}}, T(e)))$.

$$E := \{(n, (f_1 \land f_2), (n, f_1)), (n, f_2)) \mid n \in \text{dom}(T), i \in \{1, 2\}, \epsilon \in \{\lor, \land\} \cup \{(n, (i, q)), (n, \delta(q, T(n, i)))\} | n, i \in \text{dom}(T), \Omega(n, (i, q)) = \Omega(q), \text{ and } \Omega(n, f \lor f') = \Omega(n, f \land f') = 0.$$

The language of $A$, written $L(A)$, is the set of trees for $T$ such that Player has a winning strategy for $G(T, A)$.

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 $f_1 \lor f_2$, Player picks $i$ and tries to show that the node $n$ satisfies $f_i$. If $f$ is a conjunction $f_1 \land f_2$, 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 $n$ 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 $A_0 = ((q_0, q_1), \Sigma, \delta, q_0, \Omega)$, where:

$$\Sigma = \{a \mapsto 2, b \mapsto 1, c \mapsto 0\}$$

$$\delta(q_0, a) = (1, q_0) \land (2, q_0), \delta(q_0, b) = (1, q_1), \delta(q_0, c) = tt$$

$$\Omega(q_0) = 1 \quad \Omega(q_1) = 2$$

Let $T$ be the tree where $\text{dom}(T) = (2.1)^* \cup (2.1)^*1 \cup (2.1)^*2$. $T(n) = a$ if $n \in (2.1)^*1$, $T(n) = c$ 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 $T = a \lor (b \cdot T)$. Let $D$ be $\text{dom}(T)$, the acceptance game $G(T, A_0)$ is $$(V_T, V_2, v_{\text{init}}, E, \Omega),$$

$$V_T = \{(n, (1, q_0) \land (2, q_0)), n \in D\} \cup \{(n, tt), n \in D\}$$

$$V_2 = \{(n, f), n \in D, f \in \{(1, q_0), (2, q_0), (1, q_1)\}\}$$

$$v_{\text{init}} = (\epsilon, (1, q_0) \land (2, q_0))$$

$$E = \{(n, (1, q_0) \land (2, q_0)), (n, (i, q)), (n, (1, q_0) \land (2, q_0))) \mid n \in (2.1)^*1, i \in \{0, 1\}\}$$

$$\cup \{(n, (2, q_0)), (n, (2, q_0)), n \in (2.1)^*1\}$$

$$\cup \{(n, (1, q_1)), (n, (1, q_1), tt) \mid n \in (2.1)^*1\}$$

$$\Omega(n, (i, q)) = j + 1 \text{ for } n \in D, j \in \{0, 1\}, \text{ and } i \in \{1, 2\}, \Omega(n, f) = 0 \text{ for } n \in D, f \in \{tt, (1, q_0), (2, q_0)\}.$$
\textbf{Model Checking of HORS}

In this section, we review the definition of higher-order recursion schemes (HORS) and the model checking problem on them [25]. 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 $\mu$-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 [25].

We first define types and terms. The set of simple types, ranged over by $\kappa$, is defined by:

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

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

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

A $\lambda$-term that does not contain $\lambda$ is called an applicative term. We often omit the type annotation and just write $Ax.e$ for $\lambda x : \kappa.e$. As usual, the type judgment relation $\mathcal{K} \vdash e : \kappa$, where $\mathcal{K}$ is a map$^3$ from a finite set of variables to the set of simple types, is defined as the least relation closed under the following rules:

$\mathcal{K}, x : \kappa \vdash x : \kappa$ \hspace{1cm} $\mathcal{K}, x : \kappa, e_1, e_2 : \kappa \vdash e_1 \cdot e_2 : \kappa$

$\mathcal{K} \vdash e_0 : \kappa_1 \rightarrow \kappa_2 \hspace{1cm} \mathcal{K} \vdash e_1 : \kappa_1 \hspace{1cm} \mathcal{K} \vdash \kappa_1 \rightarrow \kappa_2$

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

- $\Sigma$ is a ranked alphabet. The elements of $\Sigma$ 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 $\lambda$-terms (where both terminals and non-terminals are treated as variables). If $\mathcal{N}(A) = \kappa_1 \rightarrow \cdots \rightarrow \kappa_j \rightarrow \star$, then $\mathcal{R}(A)$ must be of the form $\lambda x_1 : \kappa_1, \ldots, \lambda x_t : \kappa_t.e$, where $e$ is an applicative term such that $\Sigma \cup \mathcal{N}, x_1 : \kappa_1, \ldots, x_t : \kappa_t \vdash e : \star$. Here, $\Sigma^1$ denotes:

$$\{a : \star \rightarrow \cdots \rightarrow \star \rightarrow \star \mid a \in \text{dom}(\Sigma)\}.$$

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

$^3$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.
2.3 HFL Model Checking

In this section we review Higher-Order Modal Fixpoint Logic [34] (HFL) and its model-checking problem. HFL is an extension of the modal μ-calculus with higher-order recursive predicates; HFL formulas \( \varphi \) and HFL types \( \eta \) are defined by the following grammar:

\[
\begin{align*}
\varphi & ::= \top \mid \bot \mid X \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid (\alpha)\varphi \mid [a]\varphi \mid \mu X.\varphi \mid \nu X.\varphi \mid \lambda X : \varphi \mid \varphi_1 \varphi_2 \\
\eta & ::= \bullet \mid \eta_1 \rightarrow \eta_2
\end{align*}
\]

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 greatest fixpoint operators \( \mu \) and \( \nu \); the difference is that they can be over higher-order predicates (created by a λ-abstraction \( \lambda X : \eta.\varphi \)). In its original formulation [34], 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 [21].

Each binder (\( \mu, \nu, \lambda \)) 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 \( \eta_1 \rightarrow \eta_2 \) describes functions from \( \eta_1 \) to \( \eta_2 \). The order of an HFL type \( \eta \) is defined by: \( \text{ord} (\bullet) = 0 \) and \( \text{ord}(\eta_1 \rightarrow \eta_2) = \max(\text{ord}(\eta_1) + 1, \text{ord}(\eta_2)) \). 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:

\[
\begin{align*}
\mathcal{H} \vdash \top & : \bullet \\
\mathcal{H} \vdash \bot & : \bullet \\
\mathcal{H}, X : \eta \vdash X : \eta \\
\mathcal{H} \vdash (a)\varphi & : \bullet \\
\mathcal{H} \vdash (\alpha)\varphi & : \bullet \\
\mathcal{H} \vdash \varphi_1 \lor \varphi_2 & : \bullet \\
\mathcal{H} \vdash \varphi_1 \land \varphi_2 & : \bullet \\
\mathcal{H}, X : \eta \vdash \varphi & : \eta \\
\mathcal{H} \vdash \mu X.\varphi & : \eta \\
\mathcal{H} \vdash \nu X.\varphi & : \eta \\
\mathcal{H} \vdash \varphi_1 : \eta_1 \rightarrow \eta_2 & : \eta \\
\mathcal{H} \vdash \varphi_2 : \eta_2 & : \eta \\
\end{align*}
\]

A closed formula \( \varphi \) is well-typed and has type \( \eta \) if the type judgment \( \mathcal{H} \vdash \varphi : \eta \) 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 \( \varphi \) is the largest order of the type of a subformula occurring in \( \varphi \). A formula is said to be a formula of the modal μ-calculus if its order is 0.

Let \( (U, A, \rightarrow, s_0) \) be a fixed LTS. The semantics of a formula of type \( \eta \) is an object of the lattice \( (D_{\eta}, \sqsubseteq, \sqcap, \sqcup, \top, \bot, \sqcap) \) defined by induction on \( \eta \). Define \( D_\varnothing = P(U) \) as the complete lattice of sets of states, and if \( \eta = \eta_1 \rightarrow \eta_2 \) then define \( D_\eta = D_{\eta_1} \rightarrow D_{\eta_2} \) as the complete lattice of monotone functions from \( D_{\eta_1} \) to \( D_{\eta_2} \). For every type \( \eta \) and function \( f \in D_{\eta_1 \rightarrow \eta} \) has a unique least fixpoint \( \text{LFP}_\eta(f) \in D_\eta \) and a unique greatest fixpoint \( \text{GFP}_\eta(f) \in D_\eta \), respectively defined as \( \{ x \in D_{\eta_1} \mid f(x) \sqsubseteq x \} \) and \( \{ x \in D_{\eta} \mid x \sqsubseteq f(x) \} \).

The interpretation \( \mathcal{H} \) of a type environment is the set of maps \( \rho \) such that \( \rho(X) \in D_{\mathcal{H}(X)} \) for each \( X \in \text{dom}(\rho) \). The interpretation

\[
[\mathcal{H} \vdash \varphi : \eta] = \text{a map from } [\mathcal{H}] \text{ to } D_{\eta} \text{ defined by induction on } \varphi \text{ as follows:}
\]

\[
\begin{align*}
\mathcal{H} \vdash \top & : \bullet \quad \mathcal{H} \vdash \bot : \bullet \\
\mathcal{H}, X : \eta \vdash X : \eta \quad \mathcal{H} \vdash (\alpha)\varphi : \bullet \\
\mathcal{H} \vdash (\alpha)\varphi : \bullet \\
\mathcal{H} \vdash \varphi_1 \lor \varphi_2 : \bullet \\
\mathcal{H} \vdash \varphi_1 \land \varphi_2 : \bullet \\
\mathcal{H}, X : \eta \vdash \varphi : \eta \\
\mathcal{H} \vdash \mu X.\varphi : \eta \\
\mathcal{H} \vdash \nu X.\varphi : \eta \\
\mathcal{H} \vdash \varphi_1 : \eta_1 \rightarrow \eta_2 \\
\mathcal{H} \vdash \varphi_2 : \eta_2 \\
\end{align*}
\]

Note that, in the last clause, \( \eta'_n \) is uniquely determined by \( \varphi \) and \( \eta_2 \).

We often omit \( \mathcal{H} \vdash \varphi : \eta \) 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 \( \varphi \) of type \( \bullet \), we simply write \([\varphi]\) for \([\mathcal{H} \vdash \varphi : \eta]\), where \( \rho_0 \) is the empty interpretation. We write \( \mathcal{L} \models \varphi \) if \( s_{\text{init}} \in [\varphi] \).

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 \( \varphi \) of type \( \bullet \) and a labeled transition system \( \mathcal{L} \).

Theorem 7 ([23], [34]). The HFL model checking problem is decidable [34]. It is \text{k-EXPTIME} complete for order-k HFL formulas [2].

Example 4. Consider the following HFL formula \( \varphi_0 \):

\[
(\nu F^{a \rightarrow b} \nu^a. \lambda X : \bullet. (\bullet) (X (F (\lambda Y : \bullet. (b (Y) X Y)))) (\lambda Y : \bullet. (b Y))
\]

It represents the property that there exists a transition sequence of the form: \( abababab \cdots \). In fact, if we replace \( F \) with \( \lambda X : \bullet. (\bullet) (X (F (\lambda Y : \bullet. (b (Y) X Y)))) \) infinitely often and reduce the \( \beta \)-redexes, we obtain the formula:

\[
(\bullet) (\bullet) (\bullet) (\bullet) (\bullet) (\bullet) (\bullet) \cdots
\]

Consider the LTS \( L_0 = (\{ s_0, s_1 \}, \{ a, b \}, \rightarrow, s_0) \), where \( \rightarrow \) is given by:

\[
s_0 \xrightarrow{a} s_1 \quad s_1 \xrightarrow{b} s_0 \quad s_0 \xrightarrow{b} s_1.
\]

Then we have \( L_0 \models \varphi_0 \).

Example 5. Consider the following formula \( \varphi_1 \) [20]:

\[
\mu E^{a \rightarrow b} \lambda X : \bullet. (\lambda Y : \bullet. (X \land Y) \lor E ((\lambda X) (b Y))
\]

The formula \( \varphi_1 X Y \) means “there exists \( n \geq 0 \) such that \( (\bullet)^n X \) and \((b)^n Y \) holds. For example, \( \varphi_2 := \varphi_1 \varphi_0 (\bullet) \) (where \( \varphi_0 \) is the one given in Example 4) means that there exists \( n \geq 0 \) such that a transition sequence of the form: \( abababab \cdots \) is possible after \( n \) steps of \( a \)-transitions, and no \( b \)-transition is possible after \( n \) steps of \( b \)-transitions. The LTS \( L_0 \) in Example 4 satisfies \( \varphi_2 \), 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 \( X^n_1 = \alpha_1 \varphi_1; \cdots; X^n_n = \alpha_n \varphi_n \), where each \( \alpha_i \) is \( \nu \) or \( \mu \), and for each \( i = 1, \ldots, n \), \( \varphi_i \) is a formula without fixpoint binders such that \( X_1 : \eta_1 \cdots X_n : \eta_n \vdash \varphi_1 : \eta_1 \).
For an HES $E = (X_1^{x_1} = a_1 \varphi_1; \cdots ; X_n^{x_n} = a_n \varphi_n)$, we write $E(X_1)$ for $\varphi$. We often omit the type annotation $\eta$. The HFL formula denoted by $\mathcal{E} := (X_1^{x_1} = a_1 \varphi_1; \cdots ; X_n^{x_n} = a_n \varphi_n)$ is defined inductively by:

$$
toHFL(X^a) = a X^a, \varphi \quad toHFL(E; \varphi) = toHFL(([a X^a \varphi] / X)E),$$

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

**Example 6.** The HFL formula $\varphi_0$ in Example 4 can be represented as the following HES:

$$S =_{\nu} F (\lambda Y : \bullet (b) Y); \quad F =_{\nu} \lambda X : \bullet \rightarrow \bullet (a) (X (F (\lambda Y : \bullet (b) (X Y)))).$$

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

$$S =_{\nu} F B; \quad F =_{\nu} \lambda X : \bullet \rightarrow \bullet \vdash (a) (X (F (G X))); \quad G =_{\nu} \lambda Y : \bullet \rightarrow \bullet (\lambda Y : \bullet (b) (X Y)); \quad B =_{\nu} \lambda Y : \bullet \rightarrow \bullet (b) Y.$$

**Example 7.** Consider the following HES $\mathcal{E}_1$:

$$S =_{\nu} X; \quad Y =_{\nu} \lambda Z. (\langle a \rangle (Z \land X)); \quad X =_{\nu} \langle a \rangle (Y X).$$

Then $\mathcal{E}_1$ is unsatisfiable. This can be checked by the following observations:

- $toHFL(\mathcal{E}_1)$ is the formula $\mu X. (\langle a \rangle \langle Z \land Y \langle a \rangle (Z \land X) \rangle)$.
- $\varphi$ implies $Z$, $toHFL(\mathcal{E}_1)$ implies $\mu X. (\langle a \rangle (\langle a \rangle X) \langle a \rangle X)$, which is unsatisfiable.

### 3. From HOR to HFL Model Checking

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

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

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

#### 3.1 Tree Automata Encoded as LTS

Let us fix an APT $A = (Q, \Sigma, \delta, q_{init}, \Omega)$ and construct the labeled transition system $\mathcal{L}_A$ encoding it. Intuitively, the control graph of $A$ becomes the LTS, but since the transition relation of $A$ uses positive Boolean formulas, these must be encoded as states of the transition system. Formally, the set of states of $\mathcal{L}_A$ is $Q \cup Q^l$, where $Q^l := \{ q \mid q \text{ is a subformula of } \delta(q, a) \}$ for some $(q, a) \in Q \times \text{dom}(\Sigma)$. 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

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

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

$$q \xrightarrow{\alpha \nu \eta \delta(q, a)} (d, q) \xrightarrow{d} q, \quad f_1 \land f_2 \xrightarrow{\text{and}} f_1 \lor f_2 \xrightarrow{\text{or}} f_1 \xrightarrow{\text{true}} q \xrightarrow{tt} \text{true}$$

for $q \in Q$, $a \in \text{dom}(\Sigma)$, and $i = 1, 2$. Note how the priority of a state $q$ is determined by the index $i$ on the label of any transition $q \xrightarrow{\text{true}}$ 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 $A_0$ be the APT of Example 1. The LTS $\mathcal{L}_A$ encoding $A_0$ is depicted on Figure 2.

![Figure 2](image-url)

**Figure 2.** The LTS $\mathcal{L}_A$ associated to the APT $A_0$ of Example 1, where $f = (1, q_0) \land (2, q_0)$, and $q_0$ is the initial state.

### 3.2 The Case of Trivial Automata

In order to get a better intuition of the encoding of $G$ into an HFL formula $\varphi_G$, we first discuss the special case where the automaton $A$ 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 [12].

As explained at the beginning of this section, $\varphi_G$ expresses the property that the automaton (or, the corresponding LTS $\mathcal{L}_A$ constructed above) has a successful run for the tree generated by $G$. Let us first consider a special case, namely where $G$ generates the finite tree $a_1 (b_1 c_1)$ (or $0$). Then, since the initial state of the automaton should be able to accept $a_1$, the corresponding LTS $\mathcal{L}_A$ should have a transition $a_01$; hence $\varphi_G$ should be of the form $(\nu a_0) \varphi_1$, where $\varphi_1$ describes the property that should be satisfied by the state $s = \delta(q_{init}, a)$. The formula $\varphi_1$ is not aware of the shape of $\delta(q_{init}, a)$, but knows that the state $s$ of the LTS after the $a$-transition is a positive Boolean formula. Thus, $\varphi_1$ 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 $A$ 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 $A$ for the second child $b$. $c$.
- If $s = f_1 \land f_2$, then any state after a $\xrightarrow{\text{and}}$-transition should satisfy $\varphi_1$ again.
- If $s = f_1 \lor f_2$, then some state after a $\xrightarrow{\text{or}}$-transition should satisfy $\varphi_1$ again.
- If $s = \text{true}$, i.e., if there is a $\xrightarrow{\text{true}}$-transition, then there is no further requirement.

Thus, $\varphi_1$ can be described as

$$\nu X. (1) \varphi_0 \lor (2) \varphi_0 \lor ((\text{and}) T \land [\text{and}] X) \lor ((\text{or}) X) \lor (\text{true}) T,$$
where \( \varphi_c \) and \( \varphi_{bc} \) describe the properties that the current state has transitions corresponding to accepting runs for \( c \) and \( bc \) respectively, which can be defined by:
\[
\varphi_c := \langle c_0 \rangle \nu X. (\langle and \rangle X \land \langle or \rangle X) \lor (\langle true \rangle \land (\langle and \rangle X \lor (\langle or \rangle X) \lor (\langle true \rangle) \land (\langle true \rangle). 
\]
By preparing the following formula \( L_\alpha \):
\[
\nu X. \lambda y_1, \ldots, y_n. \bigwedge_{i=1}^n (\langle and \rangle X \land \langle or \rangle X) \lor (\langle or \rangle X \land \langle true \rangle) \land (\langle true \rangle) 
\]
the formula \( \varphi_{bc} \) 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 \( \varphi_T \) that describes the property “the LTS \( L_A \) has transitions corresponding to a successful run of \( A \) that accepts \( T \)”, can be constructed inductively by:
\[
\varphi_{a_1 \cdots a_k} := (\langle a_0 \rangle (L_\ell \varphi_{a_1} \cdots \varphi_{a_k})). 
\]
In other words, the translation from a tree to the corresponding formula works as a homomorphism that replaces each tree constructor \( a \) of arity \( \ell \) with \( \lambda x_1, \ldots, \lambda x_\ell (a_0) (L_{x_1} \cdots L_{x_k}) \). Thus, we can naturally extend the translation to one from a HORS to a formula, as given below.

For a given HORS \( G = (\Sigma, N, R, S) \), let \( E_G \) be the HES \( A_0 = \nu (e_0)^1 \cdots A_m = \nu (e_m)^1 ; E_{aux} \) where \( (i) E_{aux} \) is the set of definitions for \( \nu L_n = \nu \lambda y_1, \ldots, y_n. \bigwedge_{i=1}^n (\langle and \rangle X \land \langle or \rangle X) \lor (\langle or \rangle X \land \langle true \rangle) \land (\langle true \rangle) \lor (\langle true \rangle) \lor (\langle true \rangle) \) for \( n \in \{1, \ldots, k\} \) with \( k \) being the largest arity; (ii) \( A_0, \ldots, A_m \) are the non-terminals of \( G \) with \( S = A_0 \); (iii) \( e_1 = R(A_1) \); and (iv) \( (e)^1 \) is defined by induction on \( e \) as follows:
- \( (\lambda y : \kappa, e)^1 = \lambda y : (\kappa, e)^1 \)
- \( (e_1 e_2)^1 = (e_1)^1 (e_2)^1 \)
- \( (z)^1 = z \) if \( z \) is either a non-terminal or a variable
- \( (a)^1 = \lambda y_1, \ldots, \lambda y_\Sigma(a) \cdot (a_0) (L_{y_1} \cdots L_{y_\Sigma(a)}) \)
- \( (\bullet)^1 = \bullet \)
- \( (\kappa_1 \mapsto \kappa_2)^1 = (\kappa_1)^1 \mapsto (\kappa_2)^1 \)

As in the case for the translation from trees to formulas, we just need to replace each tree constructor \( a \) of arity \( \ell \) with \( \lambda y_1, \ldots, y_\ell (a_0) \cdot L_{y_1} \cdots L_{y_\ell} \).

Example 9. Consider the HORS of Example 2. Then its encoding as a HFL formula is defined by the following HES (notice that some \( \beta \)-reductions have been done to ease readability):
- \( S = \nu F (\lambda x. (b_0 \cdot (L_1 z))) \)
- \( B = \nu \lambda g. (a_0) (L_2 (\langle c_0 \rangle L_0) (g ((B (\lambda x. (b_0 \cdot (L_1 z))))))) \)
- \( L_2 = \nu \lambda x. (b_0) (L_1 (g \, x)) \)
- \( L_1 = \nu \lambda x. (b_0) (L_0) \)

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 \( A \) and HORS \( G \), \( T_G \in L(A) \) if and only if \( L_A \models E_G \).

### 3.3 The General Case

In the general case where \( A \) is an APT with priorities \( \{0, \ldots, p-1\} \), we need to take into account the priority 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 \( G \) is a HORS where all non-terminals are of type \( * \) and all rules are of the form \( A \rightarrow a_1 \cdots a_{\Sigma(a)} \). For each \( A \), we prepare \( p \) fixpoint variables \( A^{p0}, \ldots, A^{pp-1} \), defined by
\[
A^{p_i} = a_i \lor (L_\alpha) (A_1^{p_1} \cdots A_{\Sigma(a)}^{p_1}), 
\]
where \( \alpha_i \) is \( \nu \) if \( i \) is even and \( \mu \) otherwise. As in the case of trivial automata, \( A^{p_i} \) expresses the property that the current state has transitions corresponding to a accepting run of \( A \) over the tree generated by \( A \); in addition, \( A^{p_i} \) 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 \( a_i \), hence the above definition of \( A^{p_i} \). If a priority \( i \) is visited infinitely often by the automaton, then a fixpoint variable of the form \( A^{p_i} \) is unfolded infinitely often. Thus, by letting \( E_G = (E_{p-1}; \ldots; E_0; E_{aux}) \) where \( E_i \) contains a declaration for \( A^{p_i} \) of the above form and \( E_{aux} \) is as given in the previous section, we can guarantee that the largest priority visited by \( A \) is even if and only if the largest index of the fixpoint variables expanded infinitely often is even. We thus obtain \( L_A \models E_G \) if and only if \( T \in L(A) \).

In the case of a HORS of an arbitrary order, each rule of the form \( A \rightarrow C[A_1, \ldots, A_k] \) should be replaced by a fixpoint equation of the form:
\[
A^{p_i} = a_i C[A_1^{p_i}, \ldots, A_k^{p_i}], 
\]
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 \( A \rightarrow B A_j \). 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 \( B^{p_0} A_1^{p_0} \cdots A_{p-1}^{p-1} \); here, \( B^{p_0} \) is defined so that it calls the \( i \)-th argument \( A_i^{p_i} \) 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 HORS \( E_G^{(p)} \) encoding the HORS \( G \) for any alternating parity automaton with priorities in \( \{0, \ldots, p-1\} \). It is defined by \( E_G^{(p)} := E_{p-1}; \ldots; E_0; E_{aux} \) where for each non-terminal \( A \) and for each priority \( i \), there is a definition \( A^{p_i} = \nu_i (R(A)^{p_0}) \) in \( E_i \), with \( (e)^1 \) to be defined soon, and again with \( \alpha_i = \nu \) if \( i \) is even and \( \mu \) otherwise.

For any term \( e \) and for any priority \( i \in \{0, \ldots, p-1\} \), let the formula \( (e)^1 \) be defined by induction on \( e \) as follows:
- \( (\lambda y : \kappa, e)^1 = \lambda y : (\kappa, e)^1 \)
- \( (e_1 e_2)^1 = (e_1)^1 (e_2)^1 \)
- \( (z)^1 = z \) if \( z \) is either a non-terminal or a variable
- \( (a)^1 = \lambda y_1, \ldots, \lambda y_\Sigma(a) \cdot (a_0) (L_{y_1} \cdots L_{y_\Sigma(a)}) \)
- \( (\bullet)^1 = \bullet \)
- \( (\kappa_1 \mapsto \kappa_2)^1 = (\kappa_1)^1 \mapsto (\kappa_2)^1 \)

where the \( L_n \)'s definitions are as before and introduced in \( E_{aux} \). Intuitively, \( i \) in \( (e)^1 \) 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 \( G_1 \) consisting of the rules:
- \( S \rightarrow FB \)
- \( Fg \rightarrow a \rightarrow c \rightarrow g \rightarrow F \)
- \( Bx \rightarrow bx \)
which is a simpler variant of \( G_0 \) in Example 2. It generates the regular tree \( T \) such that \( T = a \rightarrow c \rightarrow (bT) \). The HES \( E_G^{(0)} \) is:
- \( S^{p2} = \nu \varphi_s ; F^{p2} = \varphi_s ; B^{p2} = \varphi_s \)
- \( S^{p1} = \nu \varphi_s ; F^{p1} = \varphi_s ; B^{p1} = \varphi_s ; E^{p1} = \varphi_s \)
- \( S^{p0} = \nu \varphi_s ; E^{p0} = \varphi_s ; B^{p0} = \varphi_s ; E_{aux} \).
where
\[ \varphi_S = F_{20}B_{20}B_{21}B_{22} \]
\[ \varphi_F = \lambda g_{20}\lambda g_{21}\lambda g_{22}. \]
\[ (a_0)(L_1(c_0)L_0) \land (c_1)L_0 \lor (c_2)L_0) \varphi_F^{(0)}(F,g) \]
\[ \lor (a_1)(L_2(c_0)L_0) \land (c_1)L_0 \lor (c_2)L_0) \varphi_F^{(1)}(F,g) \]
\[ \lor (a_2)(L_3(c_0)L_0) \land (c_1)L_0 \lor (c_2)L_0) \varphi_F^{(2)}(F,g) \]
\[ \varphi_B = \lambda x_{20}\lambda x_{21}\lambda x_{22}. \]
\[ (b_0)(L_1{x_0}^{21}) \lor (b_1)(L_1{x_1}^{21}) \lor (b_2)(L_1{x_2}^{21}) \]
\[ \varphi_F^{(0)}(F,g) = g_{20}g_{21}g_{22} \]
\[ \varphi_F^{(1)}(F,g) = g_{20}g_{21}g_{22} \]
\[ \varphi_F^{(2)}(F,g) = g_{20}g_{21}g_{22} \]

For the LTS \( L_{A_0} \), in Figure 2, we can remove irrelevant parts of the formulas \( \varphi_S, \varphi_F \) and \( \varphi_B \) and simplify them to:
\[ \varphi_S' = F_{20}B_{21}B_{22} \]
\[ \varphi_F' = \lambda g_{21}\lambda g_{22}. \]
\[ (a_1)(L_2(c_1)L_0) \]
\[ (b_1)(L_1(a_2)(L_2(c_1)L_0) \]
\[ (b_1)(L_1(F_{22}B_{22}B_{22})) \]

The simplified version of \( S_{22} \) can be expanded (with some further simplification) to:
\[ (a_1)(L_2(c_1)L_0) \]
\[ (b_1)(L_1(a_2)(L_2(c_1)L_0) \]
\[ \cdots \lor (a_2)(L_2(c_1)L_0) \]
\[ (b_1)(L_1(F_{22}B_{22}B_{22})) \lor \cdots \]

The LTS in Figure 2 satisfies this property; note that \( F_{22} \) is defined by one of the outermost fixpoint operators \( \nu \).

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.

**Theorem 10.** Let \( A \) be an APT with priorities in \( \{0, \ldots, p-1\} \), and let \( \mathcal{G} \) be a HORS. Then \( T_0 \subseteq L(A) \) iff \( L_A = \Sigma_{G}^{\nu} \).

It might be noticed that the size of \( F_{22} \) is in \( O(p(n^0)(n)^2) \), where \( p \) is the number of priorities, and \( an(e) \) is the nesting of applications inside arguments, defined via \( an(e_1 e_2) = max(an(e_1),1+an(e_2)) \), \( an(\lambda y.e) = an(e) \), and \( an(A) = an(a) = an(y) = 0 \). This exponential blow-up might seem prohibitive, but it is easy to avoid. Indeed, by introducing some extra non-recursive, any HORS can be rewritten into an equivalent one with a linear blow-up such that for any non-terminal \( A, an(R(A)) \leq 2 \).

**Theorem 11.** For every HORS \( \mathcal{G} \) and \( p \geq 1 \), there is an HES \( E \) of size linear in the size of \( \mathcal{G} \) and polynomial in \( p \) such that for any APT \( A \) with priorities in \( \{0, \ldots, p-1\} \), \( T_0 \subseteq L(A) \) iff \( L_A = \Sigma_{E} \). Furthermore, \( E \) can be constructed in time polynomial in the size of \( \mathcal{G} \) and \( p \).

### 4. Intersection Types for HFL Model Checking

**Example 11.** Consider the HES \( E_0 \) of Example 6 and the LTS \( L_0 \) of Example 4. Let \( \Gamma = \{ G : (s_0 \rightarrow s_1) \rightarrow s_0 \rightarrow s_1, G : (s_1 \rightarrow \sigma) \rightarrow s_1 \rightarrow s_1, \} \\ F : (s_0 \rightarrow s_1) \land (s_1 \rightarrow s_1) \rightarrow s_0 \}. \) Then the type judgment \( \Gamma \vdash E(F) : (s_0 \rightarrow s_1) \land (s_1 \rightarrow s_1) \rightarrow s_0 \) holds (see the derivation in Figure 4).

5 For example, \( \top \) is actually annotated like \( \top^5 \). Without this assumption on the implicit annotation, \( S_{type}(\top) \) cannot be determined.
For an entire formula (represented in the form of an HES), we define typability in terms of a parity game.

Let \( \text{dep}(\mathcal{E}) \) be the number of switches between \( \nu \) and \( \mu \): 

\[
\text{dep}(\mathcal{E}) = 0 \\
\text{dep}(\mathcal{E} = \nu ; \mathcal{E}) = \left\{ \begin{array}{ll}
\text{dep}(\mathcal{E}) & \text{if } \text{dep}(\mathcal{E}) \text{ is even} \\
\text{dep}(\mathcal{E}) + 1 & \text{if } \text{dep}(\mathcal{E}) \text{ is odd}
\end{array} \right.
\]

\[
\text{dep}(\mathcal{E} = \mu ; \mathcal{E}) = \left\{ \begin{array}{ll}
\text{dep}(\mathcal{E}) & \text{if } \text{dep}(\mathcal{E}) \text{ is even} \\
\text{dep}(\mathcal{E}) + 1 & \text{if } \text{dep}(\mathcal{E}) \text{ is odd}
\end{array} \right.
\]

The priority of \( F_i \) in \( \mathcal{E} \), written \( \Omega_\mathcal{E}(F_i) \), is defined as \( \text{dep}(F_i) = \alpha_i \) \( \phi_i ; \mathcal{E}_2 \) if \( \mathcal{E} = (\mathcal{E}_1; F_i = \alpha_i \phi_i ; \mathcal{E}_2) \). For example, for the HES \( \mathcal{E}_7 \) of Example 7, \( \Omega_{\mathcal{E}_7}(S) = 3, \Omega_{\mathcal{E}_7}(Y) = 2 \), and \( \Omega_{\mathcal{E}_7}(X) = 1 \). When \( \mathcal{E} \) is clear from context, we omit the subscript and just write \( \Omega(F_i) \).

**Definition 12.** Let \( \mathcal{E} := (F_{11} = \alpha_{11} \phi_1; \cdots; F_{n\mathcal{\mu}} = \alpha_{n\mathcal{\mu}} \phi_n) \) be a fixpoint-free HES with \( \eta_1 = \bullet \), and \( \mathcal{L} = (U, A, \rightarrow, s_{\text{init}}) \) an LTS. The typability game \( \text{TG}(\mathcal{L}, \mathcal{E}) \) is the parity game \( (V_{\mathcal{G}}, V_{\text{init}}, V_{\text{init}}, \Omega, E, \Omega_\mathcal{G}) \), where:

- The set \( V_{\mathcal{G}} \) of Opponent’s positions is the set of intersection type environments \( \{ \Gamma \mid \text{dom}(\Gamma) \subseteq \{ F_1, \ldots, F_n \} \wedge \forall (\tau ; \mathcal{E}) \in \Gamma, \tau \leq \eta_1 \} \).
- The set \( V_{\text{init}} \) of Player’s positions is the set of type bindings that respect simple types, i.e., \( \{ (\mathcal{E}, \mathcal{F}, \tau) \mid \forall (\tau ; \mathcal{E}) \in \Gamma, \mathcal{F} \in \Gamma \} \).
- \( s_{\text{init}} \) is the initial position \( F_i = s_{\text{init}} \).
- \( E = E_1 \cup E_2 \), where \( E_1 \), the set of Player’s moves, is \( \{ (\mathcal{E} : \mathcal{F} ; \tau) \in \Gamma, E_2 \), the set of Opponent’s moves, is \( \{ (\mathcal{E} : \mathcal{F} ; \tau) \mid E_1 \} \).
- The priority function \( \Omega_\mathcal{G} \) is defined by: \( \Omega(\Gamma) = 0 \) for every \( \Gamma \in V_{\mathcal{G}} \), and \( \Omega(\mathcal{F} ; \tau) = \Omega_\mathcal{G}(F_i) \) for every \( \mathcal{F} ; \tau \in V_{\text{init}} \).

We write \( \mathcal{L} \models \mathcal{E} \) when Player wins the parity game \( \text{TG}(\mathcal{L}, \mathcal{E}) \).

Intuitively, in the game \( \text{TG}(\mathcal{L}, \mathcal{E}) \) Player tries to prove that \( \mathcal{L} \models \mathcal{E} \), and Opponent tries to disprove it. To this end, Player first shows that \( \phi_1 \), the righthand side of \( F_i \), has type \( s_{\text{init}} \) (i.e., the initial state of \( \mathcal{L} \) satisfies \( \phi_1 \)) under some type environment \( \Gamma \), and Opponent challenges it by picking a type binding \( F_j \) from \( \Gamma \), and asking why \( F_j \) has type \( \tau \). Player then shows that \( \phi_j \) has type \( \tau \) under some type environment \( \Gamma' \), and Opponent again challenges the assumption \( \Gamma' \), etc. Opponent gets stuck when Player’s assumption \( \Gamma' \) is empty, in which case Player wins; Player gets stuck when she fails to show why \( \phi_j \) has type \( \tau \), 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 \( \mathcal{E}_9 \) of Example 6 and the LTS \( \mathcal{L}_9 \) of Example 4. Let \( \Gamma \) be like in Example 11. Then Player has a winning strategy by always moving to the type environment \( \Gamma \) 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 \( \Gamma \).
- In any next round, Player is in a successor position of \( \Gamma \) chosen by Opponent, i.e., some type binding \( \lambda : \tau \) of \( \Gamma \). If \( A \) is either \( G \) or \( B \), Player can respond with the empty type environment, because \( \Gamma \vdash \mathcal{E}(A) : \tau \). Otherwise, Player is on position \( F \) with \( \mathcal{E} : \mathcal{F} \) with \( \mathcal{F} \vdash : \mathcal{F} \) holds in \( \mathcal{L}_9 \), so Player is allowed to move to \( \Gamma \).

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

**Example 13.** Consider the HFL formula \( \phi_2 \) in Example 5, which is equivalent to the following HES \( \mathcal{E}_2 \):

\[
S = \mu \lambda (F B) ((b)_B) ; \\
E = \mu \lambda X. \lambda Y. (X \land Y) \lor (\langle a \rangle X ((b)_Y)) ; \\
F = \mu \lambda X. \langle a \rangle X (F (G X)) ; \\
G = \mu \lambda X. \lambda Y. (b) (X Y) ; \\
B = \mu \lambda Y. (b) Y.
\]

Then, we have:

\[
E : s_0 \rightarrow s_0 \rightarrow s_0, F : (s_0 \rightarrow s_1) \land (s_1 \rightarrow s_2) \rightarrow s_0, \\
B : s_0 \rightarrow s_1, B : s_1 \rightarrow s_1 \vdash E_2(S) : s_0, \\
\emptyset \vdash E_2(E) : s_0 \rightarrow s_0, \\
\Gamma \vdash E_2(F) : (s_0 \rightarrow s_1) \land (s_1 \rightarrow s_2) \rightarrow s_0, \\
\emptyset \vdash E_2(G) : (s_0 \rightarrow s_1) \rightarrow s_0 \rightarrow s_1, \\
\emptyset \vdash E_2(G) : (s_1 \rightarrow s_1) \rightarrow s_1 \rightarrow s_1, \\
\emptyset \vdash E_2(B) : s_0 \rightarrow s_1, \emptyset \vdash E_2(B) : s_1 \rightarrow s_1
\]

where \( \Gamma \) is the one given in Example 11. These type judgments determine a winning strategy for Player.

**Example 14.** Consider the unsatisfiable HES \( \mathcal{E}_3 \) of Example 7; recall that \( \Omega_{\mathcal{E}_3}(S) = 3, \Omega_{\mathcal{E}_3}(Y) = 2 \), and \( \Omega_{\mathcal{E}_3}(X) = 1 \). Let \( \mathcal{L} = (\{ s \}, \{ [a], \cdots \}, s) \) with \( s \rightarrow s \). A strategy for Player in \( \text{TG}(\mathcal{L}, \mathcal{E}_3) \) is to always play \( \Gamma = \{ X : s; Y : s \rightarrow s \} \). 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 \( \mathcal{L} \models \mathcal{E} \).
The proof of the above theorem is given in the longer version [17]; here we just give an outline. The proof uses a semantic counterpart \( SG(\mathcal{L}, \mathcal{E}) \) of the typability game, which is obtained from \( TG(\mathcal{L}, \mathcal{E}) \) by replacing the player’s moves \( ((F_i : \tau, \Gamma) \mid \Gamma \models \varphi_i : \tau) \) with \( ((F_i : \tau, \Gamma) \mid \Gamma \models \varphi_i : \tau) \), where \( \Gamma \models \varphi_i : \tau \) is a semantic type judgment relation. Since \( \Gamma \models \varphi_i : \tau \) and only \( \Gamma \models \varphi_i : \tau \), the semantic typability game \( SG(\mathcal{L}, \mathcal{E}) \) is actually isomorphic to the (syntactic) typability game \( TG(\mathcal{L}, \mathcal{E}) \). 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 \( F_1 \rightarrow hfl(\mathcal{E}) \). Because \( SG(\mathcal{L}, F_1 \rightarrow hfl(\mathcal{E})) \) 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 \( \mathcal{E} \) (i.e., the number of nodes plus the size of the transition relation \( \rightarrow \)). 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 [13]. Under the assumption above, for each variable of type \( \eta \), the number of intersection types \( \tau \) 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 the size of the typability game. By the assumption that the depth \( \text{dep}(\mathcal{E}) \) 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 [13] (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 [15]. We fix an alternating parity automaton \( \mathcal{A} = (Q, \Sigma, \delta, q_{\text{init}}, \Theta) \). KO types are defined by the grammar

\[
\theta ::= q \mid \varsigma \mid \theta \wedge \varsigma \mid \bigwedge \{ (\theta_1, m_1), \ldots, (\theta_k, m_k) \}
\]

Here, \( q \) ranges over the set \( Q \) of states of the automaton, and \( m_i \) ranges over the set \( \{0, \ldots, p-1\} \) of priorities of \( \mathcal{A} \). In the case of intersection types for HFL, we often write \( (\theta_1, m_1) \wedge \cdots \wedge (\theta_k, m_k) \) or \( \wedge \{ (\theta_1, m_1), \ldots, (\theta_k, m_k) \} \) and \( \Theta \) for \( \bigwedge \). Intuitively, \( q \) is a type of a tree that is accepted by \( \mathcal{A} \) when \( q \) is taken as the initial state, whereas \( \varsigma \) is a tree that is accepted by \( \mathcal{A} \) when \( \varsigma \) is a node of the tree. \( \bigwedge \{ (\theta_1, m_1), \ldots, (\theta_k, m_k) \} \) is the type of an n-ary function that may use the \( j \)-th argument as a value of type \( \theta_j \), and \( \wedge \{ (\theta_1, m_1), \ldots, (\theta_k, m_k) \} \) expresses where the j-th argument may be used as a value of type \( \theta_j \); intuitively, \( (\theta_j, m_j) \) specifies that in constructing the output tree of type \( q \), the \( j \)-th argument may be used as a value of type \( \theta_j \) in a node of the tree in which the largest priority visited in the path from the root to this node is \( m_j \).

As in the type system of Section 4, we only consider KO types \( \varsigma \) that are refinements of simple types \( \kappa \) (which we write \( \varsigma ::= \kappa \), defined in a similar manner as in Section 4), and the empty intersection type that refines \( \kappa \) is written \( \top^\wedge \) or \( \top \) when \( \kappa \) is not meaningful. A type environment is a set \( \Theta \) 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 \( \mathcal{Q} \models f \) (where \( f \in \mathcal{B}^\ast \{1, \ldots, n\} \times Q \) and \( \mathcal{Q} = \{Q_1, \ldots, Q_n\} \) with \( Q_i \subseteq Q \) for each \( i \)) is defined by induction on \( f \): (i) \( \mathcal{Q} \models \text{tt} \), (ii) \( \mathcal{Q} \models \text{ff} \), (iii) \( \mathcal{Q} \models (i, q) \) if \( (q \in Q_i) \), (iv) \( \mathcal{Q} \models f_1 \lor f_2 \) if \( \mathcal{Q} \models f_1 \) or \( \mathcal{Q} \models f_2 \), and (v) \( \mathcal{Q} \models f_1 \land f_2 \) if \( \mathcal{Q} \models f_1 \) and \( \mathcal{Q} \models f_2 \). The operation \( \uparrow_{m} \) on type environments is defined by:

\[
\Theta_{\uparrow m} := \{ x : (\theta, \max(m, m')) \mid x : (\theta, m') \in \Theta \}
\]

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

- The set \( V_{\mathcal{G}} \) of Opponent’s positions is the set of intersection type environments \( \{ \Theta \mid \forall (F_i : \theta) \in \Theta : \theta \models \mathcal{N}(F_i) \} \).
We write $\text{KO}$, but the proof in [15] can be easily adapted to this variation. The type system presented in this section is actually a slight variation of the KO type system. For example, with $\ell_i$.

$\text{HORS}$ is the initial position of Player. $\Theta$ is the set of type bindings that are graph isomorphisms between the graphs of the arenas of the positions. By Lemma 17, the winners of $\text{HORS}$ for intersection types $\tau$ that refine a type of the form $\kappa^\ell$. We can show that the transformation preserves typing.

**Lemma 16.** Let $e$ be a term of a $\text{HORS}$. If $\Theta \vdash^{\text{HORS}} e : \theta$, then $(\Theta)^2 \vdash^{\text{HFL}} e^{\theta_2} : (\theta)^2$. Conversely, if $\Gamma \vdash^{\text{HFL}} e^{\theta_2} : \tau$, then $(\Gamma)^1 \vdash^{\text{HORS}} e : (\tau)^3$.

The following lemma guarantees that $\Gamma_\oplus$ if and only if it is a winning position of $\text{TQ}(\ell_i)$.

**Lemma 17.** Let $f$ be a subformula of $\delta(q, a)$ with $\Sigma(a) = \eta$, and $Q_1, \ldots, Q_n \in \Gamma$. Then $\Gamma \vdash^{\text{HFL}} L_n : \Lambda\eta_{Q_1} q \to \Lambda\eta_{Q_2} q \to \cdots \Lambda\eta_{Q_n} q \to f$ is a winning position of the $\text{HFL}$ typability game if and only if $(Q_1, \ldots, Q_n) \models f$.

We can now prove that the reduction preserves typability.

**Theorem 18.** Let $G$ be a $\text{HORS}$ and $A$ be an alternating parity tree automaton. Then, $\vdash^{\text{HORS}} G$ if and only if $\text{TQ}(\ell_i)$.

Proof. Let $G$ be the parity game obtained from $\text{TQ}(\ell_i)$ by removing Player’s positions of the form $L_n$, the edges from/to those positions. By Lemma 17, the winners of $\text{TQ}(\ell_i)$ and $G$ are the same. Notice that $(\cdot)^1$ and $(\cdot)^3$ are bijections between the positions of $G$ and the ones of $\text{KO}(\ell_i)$. 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, $\Omega(x^{\eta_\oplus} : \tau) = m = \Omega(x : (\tau, m))$ holds. So both games are isomorphic.

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

**Remark 2.** As mentioned in Section 1, since the decidability of $\text{HFL}$ model checking is straightforward, the decidability of $\text{HORS}$ 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 $\text{HORS}$ model checking, because it relies on the soundness and completeness of the $\text{KO}$ 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 $\alpha F^\psi$ with $\alpha \in \mu, \nu$ and $\eta = \eta_\mu \to \cdots \eta_\nu \to \bullet$ is equivalent to $F^\psi$ where

$$F^0 = \{ \lambda x_1 : \eta_\mu, \ldots \lambda x_n : \eta_\nu \| \alpha = \nu \}
\lambda x_1 : \eta_\mu, \ldots \lambda x_n : \eta_\nu \| \alpha = \mu
$$

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

6 The type system presented in this section is actually a slight variation of the original one [15], but the proof in [15] can be easily adapted to this variation.
and \( n \) is greater than the height of the lattice of \( D_n \). For \( \eta \) 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 \( F_{\eta} \), 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 \( \hat{E} \):

\[
S = \nu F((a)T); \quad F X = \mu X \lor (b)(F X).
\]

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

\[
S = \nu F^{(n)}((a)T); \quad F^{(n)} X = \mu X \lor (b)(F^{(n-1)} X); \quad \ldots \quad F^{(1)} X = \mu X \lor (b)(F^{(0)} X); \quad F^{(0)} X = \mu X.
\]

The annotations \( \nu \) and \( \mu \) in \( \hat{E}' \) above actually do not matter, because \( \hat{E}' \) does not contain any recursion. Now, by replacing each logical connective with the corresponding tree constructor, we obtain the following HORS \( G_E \), which generates the syntax tree of the formula obtained by reducing \( \hat{E}' \):

\[
S \to F^{(n)}((a)T) \quad F^{(n)} X \to \lor X ((b)(F^{(n-1)} X)) \quad \ldots \quad F^{(1)} X \to \lor X ((b)(F^{(0)} X)) \quad F^{(0)} X \to \lor X.
\]

Let \( \mathcal{L} = (U, A, \rightarrow, s_{init}) \) 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 \( T_g \) against \( \mathcal{E} \). Such an automaton \( A_E \) would be of the form \( (\{q_s \mid s \in U\}, \Sigma, \delta, q_{s-init}, \Omega) \), where \( q_s \) is a state for checking whether \( s \) satisfies the formula represented by the current subtree, the alphabet \( \Sigma \) consists of the tree constructors corresponding to logical connectives, and the transition function \( \delta \) is defined by:

\[
\delta(q_s, T) = \text{tt} \quad \delta(q_s, \bot) = \text{ff} \quad \delta(q_s, \lor) = ((1, q_1) \lor (2, q_2)) \quad \delta(q_s, (a)) = \lor((1, q_{s \rightarrow a}) \mid s \rightarrow s') \quad \ldots
\]

Then, we have \( \mathcal{L} \models \mathcal{E} \) if and only if \( G_E \models A_E \); thus we have reduced HFL model checking to HORS model checking.

The remaining problem is that \( G_E \) 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 \( F^{(i)} \) above by the number \( j \), and encode numbers as terms of HORS. Thus, the resulting HORS is obtained by:

\[
S \to F n ((a)T) \quad F j X \to \text{if } (\text{IsNull } j) \lor ((b)(F (j-1) X)).
\]

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 [7]. Let \( \exp_k(r) \) denote the \( k \)-fold exponent of \( r \), defined by \( \exp_0(r) = r \) and \( \exp_{k+1}(r) = 2^{\exp_k(r)} \).

For our purpose, we need to represent numbers up to \( \exp_k(r) \) by terms of order at most \( k-1 \) and of size polynomial in \( r \). Prepare \( B = \{0, 1\} \) and let \( \text{Num}_1 \) be defined by

\[
\text{Num}_1 = \text{Bit} \times \cdots \times \text{Bit}.
\]

For every \( i \), let \( \lfloor i \rfloor : \{0, \ldots, \exp_i(r) - 1\} \to \text{Num}_i \) 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 \( n \in \{0, \ldots, \exp_i(r) - 1\} \), \( n_{i+1} \) maps \( \lfloor n \rfloor \) to \( b_m \), where \( b_0 \ldots b_{\exp_i(r)-1} \) is the binary representation of \( n \).

In order to compute with bits, we represent bit expressions as \( \Sigma_{\text{Bit}} \)-labeled (possibly infinite) trees where \( \Sigma_{\text{Bit}} = \{1 \rightarrow 0, 0 \rightarrow 1; 0, 1 \rightarrow \} \). We define the relation \( T \downarrow b \) inductively by: (i) \( 1 \downarrow 1 \), (ii) \( 0 \downarrow 0 \), (iii) \( \text{if } T_1 T_2 \downarrow b \) if \( T_1 \uparrow 0 \) and \( T_2 \downarrow b \), and (iv) \( \text{if } T_1 T_2 \downarrow b \) if \( T_1 \uparrow 0 \) 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 \( A_{\text{Bit}} \) be the APT \((\{q_1, q_0\}, \Sigma_{\text{Bit}}, \delta, q_s, \Omega) \), with

\[
\delta(q, \text{if}) = ((1, q_1) \lor (2, q_2)) \land ((1, q_0) \land (3, q)) \quad \text{for every } q \in \{q_1, q_0\}.
\]

Lemma 19. \( A_{\text{Bit}} \) accepts a tree \( T \) from state \( q_1 \) (\( q_0 \), resp.) if and only if \( T \uparrow 1 \) (\( T \downarrow 0 \), resp.).

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

\[
\text{Not } x \to \text{if } x 0 1 \quad \text{Or}_1 x \to x \quad \text{Or}_\ell x_1 \cdots x_\ell \to \text{if } x_1 1 \text{ Or}_{\ell-1} x_2 \cdots x_\ell.
\]

We introduce the HORS types \( \text{Bit}^* = * \) and \( \text{Num}_i \), for all \( i \geq 2 \) as follows: \( \text{Num}_2 = \rightarrow \rightarrow \rightarrow * \to * \), and for all \( i \geq 2 \),

\[
\text{Num}_{i+1} = \text{Num}_i \rightarrow * \quad \text{(note that } \text{Num}_1 \text{ is undefined only because HORS types do not have product).}
\]

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

\[
\max_i : \text{Num}_i \rightarrow \text{Num}_i \quad \text{(which represents } \exp_i(r) - 1 \text{)}
\]

\[
\text{Dec}_i : \text{Num}_i \to \text{Num}_i \quad \text{(decrement function)}
\]

\[
\text{IsNull}_i : \text{Num}_i \to \text{Bit}^* \quad \text{(check if the argument is 0)}
\]
Lemma 21. Here, lefthand side is a non-terminal of HORS defined by the rewriting functions \{Dec, \ldots, f\} where \(\beta\) be an LTS \((0, \ldots, n)\). We assume that each \(\nu = \{0, \ldots, n\}\) only if \(\nu\) is considered as \((\land \varphi_1) \varphi_2\) in the above definition. In the image of the translation, those constants are treated as tree constructors of the HORS. The arguments \(y_1, \ldots, y_r\) are of type \(\text{Num}_{\text{int}}\); intuitively, \(F_{n[j]}(y_1, \ldots, y_r)\) corresponds to \(F^{\{y_1, \ldots, y_r\}}\).

We write \(G_{E, \mathcal{L}}\) for the HORS consisting of the above rules for \(F_j, S \to F_{n[j]}(\mathcal{M}_k)\) (where \(S\) is the start symbol), and the rules in Section 6.2 for encoding numbers.

**Example 15.** Recall the LTS \(L_0\) from Example 4, and the HES \(E_0\) from Example 6:

\[
S = F B, \quad F = \nu L X : \bullet \to \bullet. (x)(X(F(G X))); \quad G = \nu L X : \bullet \to \bullet. Y : \bullet(b)(Y); \quad B = \nu L Y : \bullet. (b) Y.
\]

We obtain the HORS \(G_{E_0, L_0}\) with

\[

S' \to S \mathcal{M}_k, \quad S y s \to if (IzZero y s) \top \to (F (Dec y s) \mathcal{M}_2 (B y s \mathcal{M}_2 \mathcal{M}_2 \mathcal{M}_2))
\]

where \(y_s'\) has been renamed to their respective nonterminal for ease of understanding and the parameters \(y_s\) have been renamed to lower case versions of their HFL constructors, and the rules for \(Dec_{2}\) and \(IzZero_{2}\) are as per their definitions.

Let \(A_{E}\) be the APT \((\{q_s \mid s \in U\} \cup \{q_0, q_0', \Sigma, \delta, q_{0\text{init}}, \Omega\})\) where:

\[
\Sigma = \Sigma_{\text{BR}} \cup \{\lor \to 2, \land \to 2, \top \to 0, \bot \to 0\}
\]

\[
\cup \\cup_{a \in A} \{(a) \to \{1, [a] \to \} \mid \delta(q_s, a) = \forall (q_0, s) \mid s \to a, s'
\]

\[
\delta(q_s, a) = \delta(q_s, a) \land \{q_s \to a, s'\}
\]

\[
\delta(q_s, \top) = \top \delta(q_s, \bot) = \bot
\]

\[
\delta(q_s, (a) \top) = \delta(q_s, (a) \bot) = \delta(q_s, (a) \top)
\]

\[
\delta(q_s, (a) \top) = \delta(q_s, (a) \bot) = \delta(q_s, (a) \top)
\]

\[
\delta(q_s, 1) = \top \delta(q_s, a) = \top \delta(q_s, a) = \bot
\]

\[
\delta(q_s, 0) = \bot \delta(q_s, a) = \bot \delta(q_s, a) = \top
\]

\[
\text{and } \Omega(q) = 1 \text{ for every } q. \text{ Note that } A_{E} \text{ is an extension of the automaton } A_{\text{HFL}} \text{ in the previous subsection.}
\]

**Theorem 22.** Let \(L\) be an LTS and let \(E\) be an HES. Then \(A_{E}\) accepts the tree generated by \(G_{E, L}\) if and only if \(L \models E\). The size \(\mathcal{L}\) is via \(r\).
of $G_{E, L}$ is polynomial in the size of $E$ and $L$, and $A_{E}$ has $m + 2$ states where $m$ is the number of states of $L$. Furthermore, they can be constructed in time polynomial in the size of $E$ and $L$ (assuming that the order $k$ of $E$ 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. [8] proved the decidability of the problem for HORS with the safety restriction, and Ong [25] 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 correct, a number of alternative proofs have been developed since then [6, 13, 28, 32]. Among others, Kobayashi and Ong [12, 13] 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, 10, 11, 24, 27]. 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 [17]) has been partially inspired by Salvati and Walukiewicz’s model theoretic approach to HORS model checking [29]. On the practical side, HORS model checking has been applied to automated verification of higher-order programs [9, 16, 18, 19, 23, 26, 33, 35].

Independently of the above line of work, Viswanathan and Viswanathan [34] introduced HFL, a higher-order extension of modal $\mu$-calculus, and showed that, while model checking remains decidable for finite state systems, HFL is strictly more expressive than modal $\mu$-calculus and FL and FLC (Modal Fixpoint Logic with Chop) [22], another extension of modal $\mu$-calculus. Axelfsson 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 [20], 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 [31].

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 [13]. 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 [30], 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 [7], 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 [27].

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

mu-calculus model checking of higher-order recursion schemes. A re-
vised and extended version of [13]. http://www-kb.is.s.u-tokyo.
for higher-order model checking. In Proceedings of PLDI 2011, pages
higher-order recursion schemes and higher-order fixpoint logic, 2016.
A longer version, available from the first author’s web page.
termination verification for higher-order functional programs. In
Proceedings of ESOP 2014, volume 8410 of Lecture Notes in Computer
tion and CEGAR for disproving termination of higher-order functional
programs. In Proceedings of CAV 2015, volume 9207 of Lecture Notes
[21] 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
[22] M. Müller-Olm. A modal fixpoint logic with chop. In Proceedings of
STACS 99, volume 1563 of Lecture Notes in Computer Science, pages
verification of higher-order functional programs. In Proceedings of
algorithm for higher-order model checking. In Proceedings of ICFP
pattern-matching algebraic data types. In Proceedings of POPL 2011,
ment approach to higher-order model checking. In Proceedings of
Proceedings of FoSSaCS 2015, volume 9034 of Lecture Notes in
of higher-order programs. In Proceedings of CONCUR 2014, volume 8704 of Lecture
[31] C. Stirling and D. Walker. Local model checking in the modal mu-
[32] T. Tsukada and C. L. Ong. Compositional higher-order model checking
via ω-regular games over Bohm trees. In Proceedings of CSL-LICS ’14,
[33] H. Unno, T. Terauchi, and N. Kobayashi. Automating relatively com-
plete verification of higher-order functional programs. In Proceedings of
[34] M. Viswanathan and R. Viswanathan. A higher order modal fixed point
logic. In Proceedings of CONCUR 2004, volume 3170 of Lecture Notes in
analysis for higher order concurrent programs by higher-order model
Appendix

A. Proof of Theorem 13

We fix an LTS $L = (U, A, \rightarrow, s_{0\alpha})$ and a HES:

$$E := (F_k^n = \alpha_0, \varphi_0; \cdots ; F_0^n = \alpha_0, \varphi_0),$$

where $\eta_\alpha = \bullet$.

We assume: (i) $\alpha_k$ is $\nu$ if $k$ is even and $\mu$ otherwise; and (ii) $F_k$ occurs in none of $\varphi_0, \cdots, \varphi_n$. Those assumptions do not lose generality, because (i) if $\alpha_i = \alpha_{i+1} = \mu (\alpha_i = \alpha_{i+1} = \nu, \text{resp.})$, then we can insert a dummy equation $F^\nu = \nu F^\mu (F^\nu = \mu F^\nu, \text{resp.})$ between the equations for $F_i$ and $F_{i+1}$, without changing the semantics and typability of $E$; and (ii) if $F_n$ occurs in $\varphi_i$, we can add $F_{n+1}^\nu = \alpha_{n+1} F_n$. By the assumption above, $\Omega(F_k) = k$.

As sketched in Section 4, we show Theorem 13 through semantic typability games. We first define the semantics of types and semantic typability games in Section A.1. We then introduce in Section A.2 the semantic typability game, a semantic counterpart of the typability game defined in Section 4, and show that it is equivalent to the (syntactic) typability game introduced in Section 4. We then show soundness and completeness of the semantic typability game (with respect to $E \models \Gamma$) in Sections A.3 and A.4 respectively.

A.1 Semantics of types

The semantics of types $D_\tau(\subseteq D_{\text{Stype}(\tau)})$ and $D_\sigma(\subseteq D_{\text{Stype}(\sigma)})$ are defined by:

$$D_\tau = \{ x \in D_\sigma \mid x \in \tau \}$$
$$D_{\tau_1 \wedge \cdots \wedge \tau_k} = D_{\tau_1} \cap \cdots \cap D_{\tau_k}$$
$$D_\tau = \{ f \in D_{\text{Stype}(\tau)} \in D_{\text{Stype}(\sigma)} \mid \forall x \in D_\sigma f(x) \in D_{\tau} \}$$

Recall that we are assuming that each $\tau$ or $\sigma$ is implicitly annotated with the corresponding simple type, so that $\text{Stype}(\tau)$ and $\text{Stype}(\sigma)$ are well defined. For each intersection type $\tau$, we define $\bot_\tau \subseteq D_\tau$ by:

$$\bot_\tau = \{ s \}$$
$$\bot_{\tau_1 \wedge \cdots \wedge \tau_k} = \bot_{\tau_1} \cap \cdots \cap \bot_{\tau_k}$$

When $\tau_\uparrow: \eta$ the restriction of $(D_{\eta_1}, \bot_{\eta_1}, \gamma_{\eta_1})$ to $D_\tau$ forms a complete sublattice, having $\bot_\tau$ as the least element.

Lemma 23. Suppose $\tau : \eta$. Then, the following conditions hold.

1. If $x, y \in D_\tau$, then $x \sqcap y \in \eta \Rightarrow y \sqcap x \in \eta \Rightarrow y \in D_\tau$.
2. $\bot_\tau$ is upward-closed, i.e., $x \in D_\tau$ and $x \sqsubseteq y \Rightarrow \Rightarrow y \in D_\tau$.
3. $\bot_\tau$ is the least element of $D_\tau$.
4. If $x \in D_{\eta_1}$, then $x \in D_\tau$ if and only if $\bot_\tau \subseteq \bot_{\eta_1}$.

Proof. The first property can be shown by induction on $\eta$.

1. If $\eta = \bullet$, then $\tau = \bigwedge_{0 \leq q \leq Q} q$ for some $Q \subseteq U$ with $Q \subseteq x$, $Q \subseteq y$. So $Q \subseteq x \cap y \subseteq \tau \cap y$, henceforth $x \cap y \subseteq D_\tau$.
2. Assume $\eta = \eta_1 \rightarrow \eta_2$ and the property holds for $\eta_2$. Let $x, y \in D_{\eta_1 \rightarrow \eta_2}$. For all $z \in D_{\eta_1}$. Then $(x \sqcap_{\eta_1 \rightarrow \eta_2} y)(z) = x(z) \cap_{\eta_2} y(z) \subseteq D_{\eta_2}$ and $(x \sqcap_{\eta_1 \rightarrow \eta_2} y)(z) = x(z) \cap_{\eta_2} y(z) \in D_{\eta_2}$ by induction, henceforth $x \cap_{\eta_1 \rightarrow \eta_2} y \sqsubseteq \eta_1 \rightarrow \eta_2$.

The second and third properties also follow by straightforward induction on $\eta$. The fourth property follows as an immediate corollary of the second and third properties.

Lemma 24. If $\sigma \subseteq \sigma'$ has a derivation then $D_\sigma \subseteq D_{\sigma'}$.

Proof. By straightforward induction on the derivation of $\sigma \subseteq \sigma'$ (see the three rules HFL-T-SUBT-BASE, HFL-T-SUBT-FUN, and HFL-T-SUBT-INT of Figure 4).}

Let $\rho$ be an interpretation (i.e., a map from a finite set of variables to $\bigcup_{\alpha} D_\alpha$). We write $\Gamma \models \rho$ if $\rho(X) \in D_\tau$ for every binding $X : \tau \in \text{dom}(\Gamma)$. We write $\Gamma \models \varphi : \tau$ ($\Gamma \models \varphi : \sigma$, resp.) if $[\varphi](\rho) \in D_\tau$ ($[\varphi](\rho) \in D_\sigma$, resp.) holds for every interpretation $\rho$ such that $\rho(\Gamma) = \Gamma$.

We shall show that, for any formula $\varphi$ that does not contain fixpoint operators, the syntactic type judgment $\Gamma \vdash \varphi : \tau$ is sound and complete with respect to the semantic type judgment $\Gamma \models \varphi : \tau$.

Lemma 25 (soundness of syntactic type judgment). Let $\varphi$ be a formula without fixpoint operators. Then, $\Gamma \vdash \varphi : \tau$ implies $\Gamma \models \varphi : \tau$.

Proof. By induction on the derivation of $\Gamma \vdash \varphi : \tau$.

1. Case HFL-T-TRUE: since $\tau = s \in U$, $\Gamma \models \tau$.
2. Case HFL-T-VAR: since $X : \tau \in \Gamma$, for any $\rho \models \Gamma$, $\rho = X : \tau$.
3. Case HFL-T-SOME: then $\varphi = \{a\} \varphi'$, $s = s \in U$, $s \in s'$, and $\Gamma \vdash \varphi' : s'$ for some $s'$. By induction, $\Gamma \models \varphi' : s'$, and by HFL semantics, $\Gamma \models \varphi : s$.
4. Case HFL-T-ALL: similar to previous case.
5. Case HFL-T-AND: then $\varphi = \varphi_1 \land \varphi_2$, $s = s \in U$, $\Gamma \vdash \varphi_1 : s$, and $\Gamma \vdash \varphi_2 : s$. By induction, $\Gamma \models \varphi_1 : s$, and $\Gamma \models \varphi_2 : s$, and by HFL semantics, $\Gamma \models \varphi_1 \land \varphi_2 : s$.
6. Case HFL-T-OR: similar to previous case.
7. Case HFL-T-ABS: then $\varphi = \lambda X : \eta \varphi'$, $\tau = \tau_1 \cdots \tau_k \rightarrow \tau'$, and $\Gamma, X : \tau_1, \cdots, X : \tau_k \vdash \varphi' : \tau'$ for some $X \not\in \text{dom}(\Gamma)$. Let $\rho$ be such that $\rho(\eta) = \Gamma$, and let $x \in D_{\eta_1}, \cdots, x \in D_{\eta_k}$.
8. Case HFL-T-APP: then $\varphi = \varphi_1 \varphi_2$, $\Gamma \vdash \varphi_1 : \tau'$ with $\tau' = \tau_1 \cdots \tau_k \rightarrow \tau$ and $\Gamma \vdash \varphi_2 : \tau_i$ for all $i = 1, \ldots, k$. By induction hypothesis, $\Gamma \models \varphi_1 : \tau_i \land \cdots \land \tau_k \rightarrow \tau$ and $\rho(\tau_i) = \rho(\tau)$ for all $i = 1, \ldots, k$. By definition of $\Gamma[\tau_1 \cdots \tau_k]$, it holds that $\rho(\varphi_1) = \tau_1 \cdots \tau_k$. Since this holds for all such $\rho$, $\Gamma \models \varphi_1 \varphi_2 : \tau$.

To prove the converse (completeness), we need some preparation. Given a type environment $\Gamma$, we define a canonical interpretation $\rho_\Gamma$ by:

$$\text{dom}(\rho_\Gamma) = \text{dom}(\Gamma) \quad \rho_\Gamma(X) = \bot_{\Gamma(X)}$$

We have:

Lemma 26. $\Gamma \models \varphi : \tau$ implies $[\varphi](\rho_\Gamma) \in D_\tau$. This follows immediately from the definition of $\Gamma \models \varphi : \tau$ and the fact $\rho_\Gamma(X) = \bot_{\Gamma(X)}$.

Proof. $\Gamma \models \varphi : \tau$ if $[\varphi](\rho_\Gamma) \in D_\tau$. This follows immediately from the definition of $\Gamma \models \varphi : \tau$ and the fact $\rho_\Gamma(X) = \bot_{\Gamma(X)}$.

Lemma 27. $\varphi \in D_\tau$ if and only if $\Gamma \models \varphi : \tau$. Suppose $\rho \models \Gamma$. Then, by the definition of $\rho_\Gamma$ and Lemma 23, we have $\rho_\Gamma(X) \subseteq \rho(X)$ for every $X \in \text{dom}(\Gamma)$. Thus, by the monotonicity of $[\varphi]$ and the upward-closedness of $D_\tau$ (Lemma 23), we have $[\varphi](\rho) \subseteq \rho_\Gamma$. Let

Proof. By straightforward induction on $\varphi$.
For each value $x \in D_\eta$, we define the corresponding type $\sigma_{x,\eta}$ by:
$$
\sigma_{x,\eta} = \bigwedge \{s \mid s \in x\}
$$
$$
\sigma_{x,\eta} \rightarrow \eta = \bigwedge_{s \in D_{\eta}} (\sigma_{y,\eta_1} \rightarrow \sigma_{x,y,\eta_2})
$$

Here, $\sigma_1 \rightarrow \sigma_2$ is defined by:
$$
\sigma_1 \rightarrow (\tau_1 \wedge \cdots \wedge \tau_k) = (\sigma_1 \rightarrow \tau_1) \wedge \cdots \wedge (\sigma_1 \rightarrow \tau_k).
$$

**Lemma 27.** If $x \in D_\eta$, then $x \succeq y$ if and only if $y \in D_{\sigma_{x,\eta}}$.

**Proof.** We first show that $x \in D_\eta$ implies $x = \bot_{\sigma_{x,\eta}}$ by induction on $\eta$.

- Case $\eta = \emptyset$: In this case, $x = \{s_1, \ldots, s_k\}$ and $\sigma_{x,\eta} = s_1 \wedge \cdots \wedge s_k$. Thus, $x \succeq \bot_{\sigma_{x,\eta}}$ follows immediately.

- Case $\eta = \eta_1 \rightarrow \eta_2$: We have:
$$
D_{\sigma_{x,\eta}} = \bigwedge_{x \in D_{\eta}} (\sigma_{y,\eta_1} \rightarrow \sigma_{x,y,\eta_2}).
$$

Suppose $y' \in \sigma_{x,\eta}$. We need to show
$$
xy' = \bigcup_{x \in D_{\eta}} \{\bot_{\sigma_{x,y,\eta_1}} \mid \sigma_{y,\eta_1} \subseteq y'\}.
$$

By the induction hypothesis, the right-hand side is equal to:
$$
\bigcup_{x \in D_{\eta}} \{\bot_{\sigma_{x,y,\eta_1}} \mid \sigma_{y,\eta_1} \subseteq y'\} = xy',
$$
as required.

Now, if $x \in D_\eta$ and $x \in \eta$, then $D_{\sigma_{x,\eta}} = x \subseteq \eta$. Thus, by Lemma 23, we have $y \in D_{\sigma_{x,\eta}}$. Conversely, if $x \in D_\eta$ and $y \in D_{\sigma_{x,\eta}}$, then $x = \bot_{\sigma_{x,\eta}} \subseteq \eta$, as required.

**Lemma 28.** If $\bot \subseteq \Sigma_\eta \lor \bot$, then $\tau' \leq \tau$.

**Proof.** We show that $\bot \subseteq \Sigma_\eta \lor \bot$ implies $\tau_i \leq \tau$ for some $i \in \{1, \ldots, k\}$.

- Case $\eta = \emptyset$: In this case, $\tau = s \leq s_i$. Thus, by the assumption $\bot \subseteq \eta \lor \bot$, we have $\{s\} \subseteq \{s_1, \ldots, s_k\}$, which implies $\tau = s = s_i = \tau_i$.

- Case $\eta = \eta_1 \rightarrow \eta_2$: In this case, $\tau = \sigma_i \rightarrow \tau'$ and $\tau_i = \sigma_i \rightarrow \tau_i$. By the condition $\bot \subseteq \eta \lor \bot$, we have
$$
\bot \subseteq \bot \lor \bot \subseteq \eta \lor \bot.
$$

The right-hand side is equal to:
$$
\bigcup_{\eta_2} \{\bot_{\eta_2}' \mid i \in \{1, \ldots, k\}, \bot_{\eta_2} \subseteq \eta_2\}.
$$

Thus, by the induction hypothesis, there must exist $i$ such that $\tau_i' \leq \tau'$ and $\bot_{\eta_2} \subseteq \eta_2$. Let $\sigma = \sigma_{\eta_2}' \wedge \cdots \wedge \sigma_{\eta_2}''$. Then $\bot_{\eta_2} \subseteq \bot_{\eta_2}' \subseteq \tau_i' \wedge \cdots \wedge \tau_i''$. For each $j \in \{1, \ldots, n\}$, the induction hypothesis, for each $j$, there exists $j' \in \{1, \ldots, m\}$ such that $\tau_{j'}' \leq \tau_{j'}''$. Thus, we have $\sigma \leq \sigma_i$. We have, therefore, $\tau_i \leq \tau$ as required.

We are now ready to prove the completeness of the syntactic type judgment.

**Lemma 29** (completeness of syntactic type judgment). Let $\varphi$ be a formula without fixpoint operators. Then, $\Gamma \models \varphi$ if and only if $\Gamma \vdash \varphi$.

**Proof.** The proof proceeds by induction on the structure of $\varphi$.

- Case $\varphi = \top$: Since $[\varphi](\mu) = U$, we have $U \subseteq D_\varphi$, which implies $\tau = s \in U$. Thus, by using HFL-T-TRUE we obtain $\Gamma \vdash \varphi : \tau$.

- Case $\varphi = \neg \psi$: By Lemma 28, we have $\Gamma \not\vdash \psi : \tau$. Therefore, we obtain $\Gamma \vdash \neg \psi$.

- Case $\varphi = \psi \land \psi'$: By Lemma 26, we have:
$$
\bot \subseteq \{\varphi(\rho) \mid \varphi(\rho) \in \{\psi(\rho'), \psi(\rho)\}\}.
$$

Thus, $\tau = s$ with $s \mapsto s'$ and $s' \in \{\varphi(\rho')\}$ for some $s, s'$. By $s' \in \{\varphi(\rho')\}$ and Lemma 26, we have $\Gamma \models \psi' : \tau$. By the induction hypothesis, we have $\Gamma \vdash \psi' : \tau$. Thus, by using HFL-T-SOME, we obtain $\Gamma \vdash \varphi : \tau$ as required.

- Case $\varphi = \psi \lor \psi'$: By Lemma 26, we have:
$$
\bot \subseteq \{\varphi(\rho) \mid \varphi(\rho) \in \{\psi(\rho'), \psi(\rho)\}\}.
$$

Thus, $\tau = s$ with $s \mapsto s'$ and $s' \in \{\varphi(\rho')\}$ for some $s, s'$. By $s' \in \{\varphi(\rho')\}$ and Lemma 26, we have $\Gamma \models \psi' : \tau$. By the induction hypothesis, we have $\Gamma \vdash \psi' : \tau$. Thus, by using HFL-T-ALL, we obtain $\Gamma \vdash \varphi : \tau$ as required.

- Case $\varphi = \neg \psi$: By Lemma 26, we have:
$$
\bot \subseteq \{\varphi(\rho) \mid \varphi(\rho) \in \{\psi(\rho'), \psi(\rho)\}\}.
$$

Thus, $\tau = s$ with $s \mapsto s'$ and $s' \in \{\varphi(\rho')\}$ for some $s, s'$. By $s' \in \{\varphi(\rho')\}$ and Lemma 26, we have $\Gamma \models \psi' : \tau$. By the induction hypothesis, we have $\Gamma \vdash \psi' : \tau$. Thus, by using HFL-T-AND, we obtain $\Gamma \vdash \varphi : \tau$ as required.

- Case $\varphi = \psi \land \psi'$: By Lemma 26, we have:
$$
\bot \subseteq \{\varphi(\rho) \mid \varphi(\rho) \in \{\psi(\rho'), \psi(\rho)\}\}.
$$

Thus, $\tau = s$ with $s \mapsto s'$ and $s' \in \{\varphi(\rho')\}$ for some $s, s'$. By $s' \in \{\varphi(\rho')\}$ and Lemma 26, we have $\Gamma \models \psi' : \tau$. By the induction hypothesis, we have $\Gamma \vdash \psi' : \tau$. Thus, by using HFL-T-OR, we obtain $\Gamma \vdash \varphi : \tau$ as required.

- Case $\varphi = \neg \psi$: By the assumption $\Gamma \models \neg \psi : \tau$, we have:
$$
[\psi(\mu)](\{[\psi(\mu)](\mu)\}) \in D_\varphi.
$$

Thus, by Lemma 26 and the induction hypothesis, we get $\Gamma \vdash \psi : \tau$ and $\Gamma \vdash \neg \psi : \tau$. Thus, by using HFL-T-AND, we obtain $\Gamma \vdash \varphi : \tau$ as required.

- Case $\varphi = \psi \lor \psi'$: By Lemma 26, we have:
$$
\bot \subseteq \{\varphi(\rho) \mid \varphi(\rho) \in \{\psi(\rho'), \psi(\rho)\}\}.
$$

Thus, $\tau = s$ with $s \mapsto s'$ and $s' \in \{\varphi(\rho')\}$ for some $s, s'$. By $s' \in \{\varphi(\rho')\}$ and Lemma 26, we have $\Gamma \models \psi' : \tau$. By the induction hypothesis, we have $\Gamma \vdash \psi' : \tau$. Thus, by using HFL-T-ALL, we obtain $\Gamma \vdash \varphi : \tau$ as required.

A.2 Semantic typability games

We call
$$
F_{\eta} = \eta \circ \alpha_1 \circ \cdots \circ \alpha_n
$$
an extended HES if \( \varphi_n \) may contain fixpoint operators. As for HES, we assume: (i) \( \alpha_k \) is \( \nu \) if \( k \) is even and \( \mu \) otherwise; and (ii) \( F_n \) occurs in none of \( \varphi_1, \ldots, \varphi_n \). Thus, \( \Omega_r(F_i) = i - j \) if \( j \) is even, and \( \Omega_r(F_i) = i + j + 1 \) otherwise.

The advantage of semantic type judgments introduced in the previous subsection is that we can define a typability game also for extended HES’s.

The semantic typability game for an extended HES

\[
E := (F^0_n = \alpha_n \varphi_n, \ldots; F^0_i = \alpha_i \varphi_i)
\]

and an LTS \( L = (U, A, \rightarrow, s_{init}) \), written \( SG(L, E) \), is a parity game \((V_s, V_T, s_{init}, \Omega_r, \psi)\), where:

- The set \( V_s \) of Opponent’s positions is the set of intersection type environments \( \{1 \mid \forall F_i : \tau \in \Gamma, \tau \vdash \eta_i\} \).
- The set \( V_T \) of Player’s positions is the set of type bindings that respect simple types, i.e., \( \{F_i : \tau \mid \tau \vdash \eta_i\} \).
- \( s_{init} \) is the initial position \( F : s_{init} \).
- \( E = E_1 \cup E_2 \), where \( E_1 \), the set of Player’s moves, is \( \{\Gamma, F_i : \tau \mid \Gamma \vdash \varphi_i \ : \tau\} \) and \( E_2 \), the set of Opponent’s moves, is \( \{\Gamma, F_i : \tau \mid \varphi_i : \tau \in \Gamma\} \).
- The priority function \( \Omega_r \) is defined by: \( \Omega_r(\Gamma) = 0 \) for every \( \Gamma \in V_s \) and \( \Omega_r(F_i : \tau) = \Omega_r(F_i) \) for every \( F_i : \tau \in V_T \).

For an ordinary HES (i.e., HES where fixpoint operators do not occur on the righthand side), the semantic typability game coincides with the (syntactic) typability game.

**Lemma 30.** Let \( E \) be an HES. Player wins \( TG(L, E) \) if and only if Player wins \( SG(L, E) \).

**Proof.** By the definition of the games, the sets of Opponent’s Player’s moves in \( TG(L, E) \) and \( SG(L, E) \) are identical. By Lemmas 25 and Lemmas 29, the sets of Player’s moves are also identical. Thus, the two games are isomorphic.

### A.3 Soundness of the Semantic Typability Game

We shall show that if Player wins the semantic typability game \( SG(L, E) \), then \( L \models E \) holds. To this end, we transform the semantic parity game step by step, until we obtain the trivial semantic parity game for \( E' := (F^0_n = \alpha_n \text{toHFL}(E)) \). Player winning the game means \( \emptyset \models \text{toHFL}(E) \). Let \( \alpha_i = \alpha_i \text{toHFL}(E) \). Hence, \( \text{toHFL}(E) \models \emptyset \). Thus, \( E' \) is obtained by removing the last equation \( F^0_n = \alpha_n \varphi_n \), and replacing \( F_i \) with \( \alpha_i \text{toHFL}(E) \). Note that \( E' \models (F^0_n = \alpha_n \text{toHFL}(E)) \) (recall that we assumed that \( F_n \) does not occur on the righthand side of \( E \)). We write \( E^{(i)} \) below for the righthand side of the equation for \( F_i \) in \( E^{(i)} \).

We shall show that the transformation from \( E^{(i)} \) to \( E^{(i+1)} \) preserves the winner of the semantic parity game. To this end, we construct a winning strategy for \( SG(L, E^{(i+1)}) \) from that for \( SG(L, E^{(i)}) \). Let \( W^{(i)} \) be a (memoryless) winning strategy for \( SG(L, E^{(i)}) \). For each winning position \( F : \tau \) of \( SG(L, E^{(i)}) \), we define the closure of \( F : \tau \), written \( \text{clos}_{W^{(i)}}(F : \tau) \), as the least type environment such that:

- \( W^{(i)}(F : \tau) \subseteq \text{clos}_{W^{(i)}}(F : \tau) \)
- If \( F_1 : \tau_1 \subseteq \text{clos}_{W^{(i)}}(F : \tau) \), then \( W^{(i)}(F_j : \tau_j) \subseteq \text{clos}_{W^{(i)}}(F : \tau) \).

For example, if \( W^{(i)}(F : \tau_1) = \{F : \tau_2, F : \tau_3\} \) and \( W^{(i)}(F : \tau_2) = \{F : \tau_3, F : \tau_4\} \), then \( \text{clos}_{W^{(i)}}(F : \tau_1) = \{F : \tau_2, F : \tau_3, F : \tau_4\} \).

We define Player’s memoryless strategy \( W^{(i+1)} \) for \( SG(L, E^{(i+1)}) \) by:

\( W^{(i+1)}(F_k : \tau) = \{F_k : \tau' \mid F_k : \tau' \subseteq \text{clos}_{W^{(i)}}(F_k : \tau), \tau' > \tau\} \)

if \( k > j \) and \( F_k : \tau \) is a winning position of \( \varphi^{(j)} \), and \( W^{(i+1)}(F_k : \tau) \) is undefined otherwise. We show that \( W^{(i+1)} \) is a valid strategy (i.e., \( (F_k : \tau, W^{(i+1)}(F_k : \tau)) \in E \)), and \( W^{(i+1)} \) is a winning strategy. To show that \( W^{(i+1)} \) is valid, it suffices to prove:

\( W^{(i+1)}(F_k : \tau) \models \varphi^{(i+1)} : \tau \)

We shall use the following lemma.

**Lemma 31 (semantic substitution lemma).** If \( F_0, F : \tau_1, \ldots, F : \tau_k \models \varphi \) with \( F \not\in \text{dom}(\Gamma_0) \) and \( \Gamma_i \models \varphi : \tau \) for each \( i \in \{1, \ldots, k\} \), then \( \Gamma_0, \Gamma_1, \ldots, \Gamma_k \models \varphi : \tau \).

**Proof.** This follows by straightforward induction on the structure of \( \varphi \).

Using the lemma above, we show that \( W^{(i+1)} \) is a valid strategy, by case analysis on \( \alpha_j \).

- Case \( \alpha_j = \mu \):
  Let us define \( \text{clos}_{W^{(i)}}(F_k : \tau) \) by:
  \( \text{clos}_{W^{(i)}}(F_k : \tau) = \{F : \tau \mid \text{clos}_{W^{(i)}}(F_k : \tau) \}

- Case \( \alpha_j = \nu \):
  Let \( \{F_1, \ldots, F_k\} \) be \( \{F_1 : \tau \mid F_k : \tau \subseteq \text{clos}_{W^{(i)}}(F_k : \tau)\} \). Then, we have:
  \( W^{(i+1)}(F_k : \tau) \models \varphi^{(j)} : \tau \),
  which implies
  \( W^{(i+1)}(F_k : \tau) \models \varphi^{(j)} : \tau \)

for every \( i \in \{1, \ldots, \ell\} \). By Lemma 26, we have:

\( W^{(i+1)}(F_k : \tau) \models \varphi^{(j)} : \tau \)

Thus, we have:

\( W^{(i+1)}(F_k : \tau) \models \varphi^{(j)} : \tau \)

from which we obtain

\( W^{(i+1)}(F_k : \tau) \models \varphi^{(j)} : \tau \)

by using Lemma 31, we have:

\( W^{(i+1)}(F_k : \tau) \models \varphi^{(j)} : \tau \)

as required.
Finally, to see that $W^{(j+1)}$ is a winning strategy, notice that for each segment $(F_k : \tau) (W^{(j+1)} ((F_k : \tau)) (F' : \tau'))$ of a play that conforms to the strategy $W^{(j+1)}$, there is a corresponding segment $(F_k : \tau) (W^{(j)} ((F_k : \tau)) (F' : \tau')) (W^{(j)} ((F_k : \tau))) \cdots (F' : \tau')$ of a play that conforms to the strategy $W^{(j)}$, where the largest priorities in the segments are the same. Thus, every play that conforms to $W^{(j+1)}$ is won by Player.

By the discussion above, we have:

**Lemma 32.** Let $E$ be an HES and $L$ be an LTS. If Player wins $SG(L, E)$, then $L \models E$.

### A.4 Completeness of the Semantic Typability Game

We show the converse of Lemma 32: if $L \models E$ then Player wins the semantic typability game $SG(E, L)$. Essentially, we just need to do the inverse of the argument for the soundness proof. We start with a winning strategy for the semantic typability game of $E^{(0)}$ and construct those for the semantic parity games of $E^{(n-1)}, \ldots, E^{(0)} = E$ step by step, where $E^{(0)}, \ldots, E^{(n)}$ are as defined in Section A.3.

Actually, we use a slightly different notion of semantic typability game. The *fat* semantic typability game for an extended $E$:

$$F_n = \alpha_n \varphi_n; \cdots; F_j = \alpha_j \varphi_j$$

(whit $\eta_n = \bullet$ and an LTS $L = (U, A, \rightarrow, s_{\text{init}})$ is a parity game $FG(L, E) = (V_c, V_s, V_e, \Omega)$, where:

- The set $V_c$ of Opponent’s positions is the set of intersection type environments $\{\Gamma | \forall F_i : \tau \in \Gamma \Rightarrow \eta_i\}$.
- The set $V_s$ of Player’s positions is the set of type bindings that respect simple types, i.e., $\{F_i : \sigma | \sigma :: \eta_i, \sigma :: \Gamma\}$.
- $V_{\text{init}}$ is the set of initial positions: $\{F_n : s_1 \land \cdots \land s_k | s_{\text{init}} \in \{s_1, \ldots, s_k\}\}$.
- $E = E_1 \cup E_2$, where $E_1$, the set of Player’s moves, is $\{(F_i : \sigma, \Gamma) | \Gamma \models \varphi_i : \sigma\}$; and $E_2$, the set of Opponent’s moves, is $\{(F_i, \Gamma : \sigma) | \sigma = \Gamma (\Gamma)\}$.
- The priority function $\Omega$ is defined by: $\Omega(\Gamma) = 0$ for every $\Gamma \in V_c$, and $\Omega(F_i : \Gamma) = \Omega_c(F_i)$ for every $F_i : \tau \in V_s$.

In the last but one clause, $\Gamma(F_j)$ denotes $\{\tau | F_i : \tau \in \Gamma\}$. Player wins if there is a winning strategy from one of the initial positions. The difference from the (non-fat) semantic typability game is that Player’s position is of the form $F : \sigma$ instead of $F : \tau$.

Assuming $L \models E$, we construct winning strategies for $FG(L, E^{(n)})$, $FG(L, E^{(n-1)})$, \ldots, $FG(L, E^{(0)})$ in this order. For $E^{(n)}$, there is a trivial winning strategy defined by: $W^{(n)} (F_n : \bot \rightarrow h_{\text{ReLU}}(E), \bullet) = \emptyset$.

Assume we are given a memoryless winning strategy $W^{(j+1)}$ for $FG(L, E^{(j+1)})$. Recall that $E^{(j+1)}$ is:

$$F_n = \alpha_n \varphi_n^{(j+1)}; \cdots; F_j + 1 = \alpha_j + 1 \varphi_j^{(j+1)} F_n,$$

where $\varphi_j^{(j+1)} = [\alpha_j F_j \varphi_j^{(j)} / F_j'] \varphi_j^{(j)}$. Without loss of generality, we assume that $W^{(j+1)}$ is defined only for Player’s winning positions of $FG(L, E^{(j+1)})$.

We define Player’s *history-sensitive strategy* $W^{(j)}$ for $FG(L, E^{(j)})$ as the partial function given by:

$$W^{(j)}(h(F_k : \sigma)) = \Gamma, F_j : \sigma_{\langle \alpha_j, F_j, \varphi_j^{(j)}, \langle \langle \Gamma, F_j, \sigma_j \rangle \rangle \rangle}$$

if $j < k$ and $W^{(j+1)}(F_k : \sigma) = \Gamma$.

- $W^{(j)}(h(\Gamma, F_j : \sigma_j) (F_j : \sigma_j)) = (\Gamma, F_j : \sigma_j)$ if $\alpha_j = \nu$ and $\sigma_j = \sigma_{\langle \nu F_j, \varphi_j^{(j)} \rangle (\langle \Gamma, F_j, \sigma_j \rangle)}$.

$W^{(j)}(h(\Gamma, F_j : \sigma_j)) = (\Gamma, F_j : \sigma_{j+1})$ if $\alpha_j = \mu$ and $\sigma_j, \tau_j = \sigma_{\langle \nu F_j, \varphi_j^{(j)} \rangle (\langle \Gamma, F_j, \sigma_j \rangle)}$.

Here, the formula $F_j^{(0)}$ occurring in the last clause is defined by:

$$F_j^{(0)} = \lambda X_1, \ldots, X_{\ell_j}. F_j^{(1)}.$$

Thus $\|F_j^{(1)}(\langle \emptyset \rangle)\|_\rho = \|\mu F_j, \varphi_j^{(j)}(\emptyset)\|_\rho$ for a sufficiently large $\ell$.

$W^{(j)}(h) is not defined if it does not match any of the three clauses above.

We show that $W^{(j)}(h)$ is a valid strategy, i.e., $W^{(j)}(h(F_k : \sigma)) = \Gamma$ implies $\Gamma \models \varphi_k : \sigma$. We perform case analysis on which clause has been used for deriving $W^{(j)}(h(F_k : \sigma)) = \Gamma$.

- **The first clause:**

  In this case, $\Gamma = \Gamma, F_j : \sigma_{\langle \alpha_j, F_j, \varphi_j^{(j)} \rangle (\langle \Gamma, F_j, \sigma_j \rangle)}$ with $k < j$ and $W^{(j+1)}(F_k : \sigma) = \Gamma'$. By the validity of the strategy $W^{(j+1)}$, we have $\Gamma' \models \varphi_k^{(j+1)} : \sigma$.

  Thus, we have

  $$\Gamma' = [\alpha_j F_j, \varphi_j^{(j)} / F_j] \varphi_k^{(j)} : \sigma$$

  as required.

- **The second clause:**

  In this case, $h = \nu \Gamma$ and $\Gamma = \Gamma, F_j : \sigma_j$ with $\alpha_j = \nu$ and $\varphi_j^{(j)} = \varphi_j^{(j+1)}$. Thus, we have $\Gamma' = \Gamma, F_j : \sigma_j \models \varphi_j^{(j)} : \sigma$ as required.

- **The third clause:**

  In this case, $h = \nu (F_j, F_j : \sigma_j)$ and $\Gamma = \Gamma, F_j : \sigma_j, \tau_j$ with $\alpha_j = \mu$ and $\sigma_j, \tau_j = \varphi_j^{(j)} = \varphi_j^{(j+1)}$. Since $\|F_j^{(1)}(\langle \emptyset \rangle)\|_\rho = \|\varphi_j^{(j)}(\emptyset)\|_\rho$, we have

  $$\Gamma' = [\alpha_j F_j, \varphi_j^{(j)} / F_j] \varphi_k^{(j)} : \sigma$$

  as required.

To check that $W^{(j)}$ is a winning strategy, it suffices to observe that (i) for each fragment $(F_k : \sigma) h(F_k', \sigma')$ of a play with $k, k' > j$, there exists a corresponding fragment of a play (consisting of two moves) $(F_k : \sigma) \Gamma(F_k', \sigma')$ conforming to $W^{(j+1)}$; (ii) if $h$ exists an infinite play that visits only $F_j$, then $\alpha_j$ must be even (since the third clause in the definition of $W^{(j)}$ can generate only finite plays); and (ii) Player never gets stuck (note that in the third clause, $\sigma_{j,0} = \Gamma$, and that in the first clause, $F_k : \sigma$ comes from the co-domain of $W^{(j+1)}$).

Now, by a standard theorem on parity games, there is also a memoryless winning strategy $W^{(j)}$.

---

8 Player’s history-sensitive strategy $W^{(j)}$ for a parity game is a partial map from $\{V_c \cup V_s\}^{<j} \times V_s$ to $V_c \cup V_s$. It is winning if Player wins every play that conforms to $W$, i.e., every play $v_{\text{init}} v_{\text{init}} v_{\text{init}} \cdots$ such that $\forall n, v_n \in V_s \Rightarrow v_{n+1} = W(v_{\text{init}} \cdots v_n)$. It is known that if there is a history-sensitive winning strategy, there also exists a memoryless winning strategy [5].
By repeating the above steps, we obtain a memoryless winning strategy $W_W$ for $FG(L, E)$. From $W_W$, we can construct a history-sensitive winning strategy $W$ for the non-fat semantic typability game $SG(L, E)$ as follows.

$$W'(F_n : s_m) = W_W(F_n : \sigma_0)$$

where $F_n : \sigma_0$ is an initial, winning position of the fat game.

$$W'(\Delta F : t) = W_W(F : \Gamma(F)).$$

We can further convert $W'$ to a memoryless winning strategy $W$ for $SG(L, E)$.

Thus, we have:

**Lemma 33.** Let $E$ be an HES and $L$ be an LTS. If $L \models E$, then Player wins $SG(L, E)$.

Theorem 13 follows as an immediate corollary of Lemmas 30, 32, and 33.

### B. Proofs in Sections 5

We show that the KO typing game $TG(q, A)$ is isomorphic to the HFL typing game $TG(L, E)$ where positions of the form $L_n : \tau$ have been omitted.

Let $\Gamma_{aux} = \{(L_n : \bigwedge_{\forall q_1 \in Q_1} q_1 \rightarrow \cdots \rightarrow \bigwedge_{\forall q_n \in Q_n} q_n \rightarrow f \mid (Q_1, \ldots, Q_n) \models f)\}.

The positions that are omitted precisely are the ones of $\Gamma_{aux}$. We first show that these are winning positions for Player.

**Proof of Lemma 17.** The claim is that always playing $\Gamma_{aux}$ is a winning strategy for Player in the typing game starting at position $L_n : \tau$.

To prove this, we reason by induction on $f$. Let $\gamma_{L_n} = \bigwedge_{\forall q_1 \in Q_1} q_1 \rightarrow \bigwedge_{\forall q_n \in Q_n} q_n \rightarrow f$.

If $f = (q_1, \ldots, q_n)$, then $f \rightarrow q_i \rightarrow q_i'$ for $i = 1, 2, \ldots, n$ since $q_i \models f$ and only if $q_i \in Q_i$, and if only if if $f_i \models f$.

If $f = f_1 \wedge f_2$, then $f \rightarrow f_1 \wedge f_2$, and if $f = f_1 \wedge f_2$ then $f \rightarrow f_1 \wedge f_2$.

If $f = f_1 \wedge f_2$, then $f \rightarrow f_1 \wedge f_2$, and if $f = f_1 \wedge f_2$ then $f \rightarrow f_1 \wedge f_2$.

Thus, $\Gamma_{aux} \models f$ and $f \rightarrow f$ is valid.

Hence, $\Gamma_{aux} \models f$.

Finally, $(\tau) \models \theta$, hence by $\text{T-Sub}^{\text{HORS}} e : (\tau)$.

**Lemma 35.** Let $e$ be term of a HORS. If $\Theta \models^{\text{HORS}} e : (\tau)$, then $(\Theta^m)^{\text{HORS}} e : (\tau)$.

Proof. By induction on $e$:

If $e = c_1 e_2$, then by $\text{T-App} \Theta = \Theta_0 \cup \bigcup_{i < j} \Theta_i \cup \Theta_j$ for some $\Theta_0$ and $\Theta_j$, and $\Theta \models^{\text{HORS}} e_1 : \Lambda_{\Theta_0} \Theta_j \cup \Theta_j \cup \Theta_j$.

By definition, $(\Theta^m)^{\text{HORS}} e_1 = (\Theta^m)^{\text{HORS}} e_1$ and $(\Theta^m)^{\text{HORS}} e_2 = (\Theta^m)^{\text{HORS}} e_2$.

By induction hypothesis, $(\Theta^m)^{\text{HORS}} e_1 = \Lambda_{\Theta_0} \Theta_j \cup \Theta_j \cup \Theta_j$.

Hence, $(\Theta^m)^{\text{HORS}} e_1 = (\Theta^m)^{\text{HORS}} e_1$ and $(\Theta^m)^{\text{HORS}} e_2 = (\Theta^m)^{\text{HORS}} e_2$.

By definition, $(\Theta^m)^{\text{HORS}} e_1 = (\Theta^m)^{\text{HORS}} e_1$ and $(\Theta^m)^{\text{HORS}} e_2 = (\Theta^m)^{\text{HORS}} e_2$.

Finally, $(\tau) \models ^\theta$, hence by $\text{T-Sub}^{\text{HORS}} e : (\tau)$.
We claim that we have to show that $\exp(x) - 1 - m''$-th bit of the binary representation of $\exp(x) - 1 - m$ is one for some $m''$, with $\exp(x) - 1 \geq m'' > m$, then, by the induction hypothesis, $T_{m''}^{m''} \not\parallel 1$, and the second clause of the if statement is relevant. 

In other words, if $T_{m'}^{m''}$, then $T_{m''}^{m''} \not\parallel b$, which is as desired since the $\exp(x) - 1 - m''$-th bit of $\exp(x) - 1 - m + 1$ must equal the same bit of $\exp(x) - 1 - m$, whence the claim holds for this case. If the $\exp(x) - 1 - m''$-th bit of the binary representation of $\exp(x) - 1 - m$ is zero for all $m''$ with $\exp(x) - 1 \geq m'' > m$, then the $\exp(x) - 1 - m''$-th bit of $\exp(x) - 1 - m + 1$ must be opposite to that of $\exp(x) - 1 - m$. By the induction hypothesis, $T_{m''}^{m''} \not\parallel 0$ whence, if $T_{m''}^{m''} \not\parallel b$, then $T_{m''}^{m''} \not\parallel b$.

It remains to show that $T_{m''}^{m''} \not\parallel 1$ if $T_{m''}^{m''} \not\parallel 1$ for some $m''$ with $\exp(x) - 1 \geq m'' > m$ and that $T_{m''}^{m''} \not\parallel 0$ if $T_{m''}^{m''} \not\parallel 0$ for all $m''$ with $\exp(x) - 1 \geq m'' > m$. By the claim of the lemma, if $i'' = \exp(x) - 1$, then the clause InZero, $g$ in the definition of ExistsOne, $h$ will generate a tree $T$ such that $T \not\parallel 1$ and $T_{m''}^{m''} \not\parallel 0$, which is correct since there is no valid $m'' > m$. The rest of the claim proceeds by induction over $m''$. Consider it proved for $m'' > 0$. We show that it holds for $m'' = 1$. By definition of ExistsOne, $h$, we have that $T_{m''}^{m'' - 1}$ is that generated by 

$$\text{if} (f (\text{Dec } g) 1 \text{ ExistsOne} h, f (\text{Dec } g))$$

where $f = \text{Dec}_{m+1}^m \text{ Max} h$, and $g = \text{Dec}_{m''}^{m'' - 1} \text{ Max}$, $T_{m''}^{m''} \not\parallel 1$ if the $\exp(x) - 1 - (m'' - 1)$-th bit of the binary representation is one, we get that $T_{m''}^{m''} \not\parallel 1$ if the $\exp(x) - 1 - (m'' - 1)$-th bit of the binary representation of $\exp(x) - 1 - (m + 1)$ is one. Since $T_{m''}^{m''} \not\parallel 0$ if the $\exp(x) - 1 - (m'' - 1)$-th bit of the binary representation is zero, we get that $T_{m''}^{m''} \not\parallel 0$ if $m'' > m$. By the induction hypothesis, $T_{m''}^{m''} \not\parallel 1$ if $m'' > m$. This finishes the induction.

Putting it all together, we obtain that, if $T$ is the tree generated by InZero, $h$ ExistsOne, $h$, then $T \not\parallel 1$ if $m'' = \exp(x) - 1$, which is the claim for the case $i'' + 1$ in the main induction. Hence, the lemma is proved.

Below we write $\text{FV}(\varphi)$ for the set of free variables occurring in $\varphi$.

Proof of Lemma 21. We define the substitution $\gamma_i (i \in \{0, \ldots, n, n + 1\})$ by:

$$\gamma_0 = []$$

(i.e., the empty substitution)

$$\gamma_i + 1 = [\alpha_i F_i, \gamma_i | \gamma_i]$$

Note that toHFL$(E) = \gamma_0 + 1 = E_0 \alpha \gamma_0 \varphi_0$.

For $\beta \in \{0, \ldots, n\}^{i'' - 1}$, we define the HFL formula $\varphi_\beta$ by:

$\varphi_\beta^{(m_0, \ldots, m_{i'' - 1}, 0)} = \lambda x_0, \ldots, x_{i''}. \varphi_\beta^{(0)}$

$\varphi_\beta^{(0)} = \omega^{(0)} F_{\alpha \beta \gamma_0 \gamma_0 \varphi_0}$

$\varphi_\beta^{(0)} = \omega^{(0)} F_{\alpha \beta \gamma_0 \gamma_0 \varphi_0}$

$\varphi_\beta^{(0)} = \omega^{(0)} F_{\alpha \beta \gamma_0 \gamma_0 \varphi_0}$

$\varphi_\beta^{(0)} = \omega^{(0)} F_{\alpha \beta \gamma_0 \gamma_0 \varphi_0}$

We shall show that $\varphi_\beta^{(0)} = \omega^{(0)} F_{\alpha \beta \gamma_0 \gamma_0 \varphi_0}$ by well-founded induction on $\beta$. Let $\beta = (\beta_0, \ldots, \beta_i)$.
• Case $\beta_j = 0$: The result follows immediately, since
  \[ \varphi^\beta_j = \lambda x_1, \ldots, x_{\ell_j} \, F_j. \]

• Case $\beta_j > 0$: We first show that
  \[ \varphi^\beta_j = \varphi^{\beta_j(0)} / F, \ldots, \varphi^{\beta_j(j)} / F_j \gamma_j F \]  
  holds for every $\ell < j$, by induction on $j - \ell > 0$. Since $\beta(0) = (\beta(1) + 1, mh)$, by the definition of $\varphi^\beta_j$, we have:
  \[ \varphi^{\beta_j(0)} = \varphi^{\beta_j(1)/F_0} = \varphi^{\beta_j(0)} / F_0 \gamma_0 F \]

By induction on $\ell < j$, for every $\beta$, we have
  \[ \varphi^{\beta_j(0)} / F = \varphi^{\beta_j(1)/F} \gamma_1 F \]

By induction, $\varphi^{\beta_j(0)} / F = \varphi^{\beta_j(0)/F} \gamma_j F$ as required.

Now, if $\beta_j < mh$, we have
  \[ F_j = [\varphi^{\beta_j(0)} / F_0, \ldots, \varphi^{\beta_j(0)/F_0} \gamma_j F] \]

By induction hypothesis, $[\varphi^{\beta_j(0)} / F] = \varphi^{\beta_j(0)/F} \gamma_j F$ for $\ell 

Thus, we have the required result.

The remaining is the case where $\beta_j = mh$. For any $\beta' = \beta_j, \ldots, \beta_j + 1, mh$ for $0 < m < mh$, we have
  \[ [\varphi^{\beta_j}] = [\varphi^{\beta_j(0)} / F_0, \ldots, \varphi^{\beta_j(0)/F_0} \gamma_j F] \]

By induction hypothesis, $\varphi^{\beta_j(0)} / F_0 = \varphi^{\beta_j(0)/F} \gamma_j F$ since $\gamma_j = \gamma_j (\beta_j)$ is closed.

Thus, we have:
  \[ [\varphi^{\beta_j}] = \varphi^{\beta_j(0)/F} \gamma_j F \]

for $f = ([\gamma_j F], \varphi^{\beta_j(0)(j+1)} / F_j + 1)(\gamma_j \gamma_j F)$ as required.

Finally, the required result follows as a special case of $[\varphi^\beta] = [\varphi^\beta]$, where $j = 0$ and $\beta = mh$.

We assume below that $\eta = \eta_{b,1} \rightarrow \cdots \rightarrow \eta_{b,}\rightarrow \bullet$. We define $\lambda$-terms $e^\beta_j$ for each $j \in \{0, \ldots, m\}$, $\beta \in \{0, \ldots, mh\}$ by induction on $\beta$ (with respect to the well-founded relation $\prec$):

\[ e_j^{(m_1, \ldots, m_j, +1)} = \lambda x_1, \ldots, x_{e_j}, [\eta_{b,1}^j, \ldots, x_{e_j}^j, e_j^{(m_1, \ldots, m_j, +1)}, \zeta_j] \]

if $\beta = (m_1, \ldots, m_j, +1)$ with $m_j > 0$.

Here, (3) translates HFL formulas and types to terms and types of HORS, by simply replacing the proposition type with the tree type, and every logical connective with the corresponding tree constructor:

\[ \bullet = \ast \quad \{ \eta \rightarrow \eta \} = \eta_{b,1} \rightarrow \eta_{b,2} \]

\[ c = c \quad \{ \lambda x \} = F_1 \]

We need to show $\varphi^\beta_1 = \varphi^\beta_{1,1} \sim e_1, e_1', e_1''$ for every $e_1, e_1', e_1''$ such that $e_1 \sim_{\eta_{b,1}} e_1'$.

Below we write $i^\#$ for Dec$^{mh-1}$Max. When $\beta = (\beta_1, \ldots, \beta_j)$, we also write $i^\#$ for the sequence $\beta_1^\#, \ldots, \beta_j^\#$. We show:

\[ e_j^{(\beta_1, \ldots, \beta_j)} = e_{j,1}^{(\beta_1, \ldots, \beta_j)} \]  

by well-founded induction on $\beta = (\beta_1, \ldots, \beta_j)$. We need to show $e_j^{(\beta_1, \ldots, \beta_j)} \sim e_{j,1}^{(\beta_1, \ldots, \beta_j)} e_{j,1}' e_{j,1}''$ for every $e_{j,1}, e_{j,1}', e_{j,1}''$ such that $e_{j,1} \sim_{\eta_{b,1}} e_{j,1}''$.

Case $\beta_j = 0$:

By the definition of $e_j^{(\beta_1, \ldots, \beta_j)}$, $T_{e_j^{(\beta_1, \ldots, \beta_j)}} e_{j,1} e_{j,1}'' = \zeta_j$. By the definition of $\zeta_j$,

\[ T_{e_j^{(\beta_1, \ldots, \beta_j)}} \beta_j^\# e_{j,1}^\# e_{j,1}'' = i \Rightarrow T_{e_j^{(\beta_1, \ldots, \beta_j)}} \beta_j^\# e_{j,1}^\# e_{j,1}'' \]
By the assumption $\beta_j = 0$ and by Lemma 20, $T_{{\text{IsZero}}(\beta_j \neq)}$ is accepted from $q_1$. Thus, the whole tree is accepted from $q_s$ if and only if $\hat{\alpha}_j$ is. Thus, we have the required result.

- **Case $\beta_j > 0$:**
  
  $$e_j^{(\beta_n \ldots \beta_j)} e_1 \ldots e_{\ell_j}$$  
  is reduced to:
  
  $$[e_0^{\beta(0)}/G_0, \ldots, e_n^{\beta(n)}/G_n, e_1/x_1, \ldots, e_{\ell_j}/x_{\ell_j}] \psi_j^i.$$
  
  On the other hand,
  
  $$G_j \beta_n \ldots \beta_j \# e_1 \ldots e_{\ell_j}$$  
  is reduced to:
  
  $$\begin{align*}
  \text{if } \text{IsZero}_k(\beta_j \#) \hat{\alpha}_j \\
  \left( [e_1'/x_1, \ldots, e_{\ell_j}/x_{\ell_j}] \psi_j \right) \beta_n \# \ldots (\beta_{j-1} \#) 
  \end{align*}$$

  The else part is actually equivalent to:
  
  $$[G_0 (\beta(0)) \# /G_0, \ldots, G_n (\beta(n)) \# /G_n, e_1'/x_1, \ldots, e_{\ell_j}/x_{\ell_j}] \psi_j^i.$$

  By the induction hypothesis, $e_i^{\beta(i)} \sim_{\eta^i} G_i (\beta(i)) \#$. Thus, by the standard logical relation argument, we obtain
  
  $$[e_0^{\beta(0)}/G_0, \ldots, e_n^{\beta(n)}/G_n, e_1/x_1, \ldots, e_{\ell_j}/x_{\ell_j}] \psi_j^i \sim_{\eta^i}$$

  $$[G_0 (\beta(0)) \# /G_0, \ldots, G_n (\beta(n)) \# /G_n, e_1'/x_1, \ldots, e_{\ell_j}/x_{\ell_j}] \psi_j^i.$$

  By the condition $\beta_j > 0$ and Lemma 20, $T_{{\text{IsZero}}_k(\beta_j \neq)}$ is accepted from $q_0$. Thus, we have the required result.

\[ \square \]

**Proof of Theorem 22.** This follows immediately from Lemmas 21, 36, and 37.

\[ \square \]