# A Type-Based HFL Model Checking Algorithm

Youkichi Hosoi, Naoki Kobayashi$^{(\boxtimes)}$, and Takeshi Tsukada

The University of Tokyo, Tokyo, Japan
hosoi@kb.is.s.u-tokyo.ac.jp
koba@is.s.u-tokyo.ac.jp
tsukada@kb.is.s.u-tokyo.ac.jp

**Abstract.** Higher-order modal fixpoint logic (HFL) is a higher-order extension of the modal $\mu$-calculus, and strictly more expressive than the modal $\mu$-calculus. It has recently been shown that various program verification problems can naturally be reduced to HFL model checking: the problem of whether a given finite state system satisfies a given HFL formula. In this paper, we propose a novel algorithm for HFL model checking: it is the first practical algorithm in that it runs fast for typical inputs, despite the hyper-exponential worst-case complexity of the HFL model checking problem. Our algorithm is based on Kobayashi et al.'s type-based characterization of HFL model checking, and was inspired by a saturation-based algorithm for HORS model checking, another higher-order extension of model checking. We prove the correctness of the algorithm and report on an implementation and experimental results.

## 1 Introduction

Higher-order modal fixpoint logic (HFL) has been proposed by Viswanathan and Viswanathan [19]. It is a higher-order extension of the modal $\mu$-calculus and strictly more expressive than the modal $\mu$-calculus; HFL can express non-regular properties of transition systems. There have recently been growing interests in HFL model checking, the problem of deciding whether a given finite state system satisfies a given HFL formula. In fact, Kobayashi et al. [10,20] have shown that various verification problems for higher-order functional programs can naturally be reduced to HFL model checking problems.

Unfortunately, however, the worst-case complexity of HFL model checking is $k$-EXPTIME complete (where $k$ is a parameter called the *order* of HFL formulas; order-0 HFL corresponds to the modal $\mu$-calculus) [2], and there has been no efficient HFL model checker. Kobayashi et al. [9] have shown that there are mutual translations between HFL model checking and HORS model checking (model checking of the trees generated by higher-order recursion schemes [15]). Since there are practical HORS model checkers available [4,7,8,16,18], one may expect to obtain an efficient HFL model checker by combining the translation from HFL to HORS model checking and a HORS model checker. That approach does not work, however, because the translation of Kobayashi et al. [9] from HFL to HORS model checking involves a complex encoding of natural numbers

as higher-order functions, which is impractical. Considering that the other translation from HORS to HFL model checking is simpler and more natural, we think that HFL model checking is a more primitive problem than HORS model checking. Also in view of applications to verification of concurrent programs [11,19] (in addition to the above-mentioned applications to higher-order program verification), a direct tool support for HFL model checking is important.

In the present paper, we propose a novel HFL model checking algorithm that is *practical* in the sense that it does not always suffer from the bottleneck of the worst-case complexity, and runs reasonably fast for typical inputs, as confirmed by experiments. To our knowledge, it is the first such algorithm for HFL model checking.

Our algorithm is based on Kobayashi et al.'s type-based characterization [9], which reduces HFL model checking to a typability game (which is an instance of parity games), and was inspired by the saturation-based algorithm for HORS model checking [18]. The detail of the algorithm is, however, different, and its correctness is quite non-trivial. Actually, the correctness proof for our algorithm is simpler and more streamlined than that for their algorithm [18].

We have implemented a prototype HFL model checker based on the proposed algorithm. We confirmed through experiments that the model checker works well for a number of realistic inputs obtained from program verification problems, despite the extremely high worst-case complexity of HFL model checking.

The rest of this paper is structured as follows. Section 2 recalls the definition of HFL model checking, and reviews its type-based characterization. Section 3 formalizes our type-based HFL model checking algorithm, and Section 4 gives an outline of its correctness proof. Section 5 is devoted to reporting on implementation and experimental results. Section 6 discusses related work, and Section 7 concludes the paper. Omitted details are found in Appendix.

## 2   Preliminaries

In this section, we review the notion of HFL model checking [19] and its type-based characterization. The latter forms the basis of our HFL model checking algorithm.

### 2.1   HFL Model Checking

We first review HFL model checking in this section.

A (finite) *labeled transition system* (LTS) $\mathcal{L}$ is a quadruple $(Q, \mathcal{A}, \longrightarrow, q_0)$, where $Q$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\longrightarrow \subseteq Q \times \mathcal{A} \times Q$ is a transition relation, and $q_0 \in Q$ is a designated initial state. We use the metavariable $a$ for actions. We write $q \xrightarrow{a} q'$ when $(q, a, q') \in \longrightarrow$.

The *higher-order modal fixpoint logic* (HFL) [19] is a higher-order extension of the modal $\mu$-calculus. The sets of (simple) types and formulas are defined by the following BNF.[1]

---

[1]   Following [9], we exclude out negations, without losing the expressive power [13].

$$\varphi \text{ (formulas)} ::= \mathtt{true} \mid \mathtt{false} \mid X \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2$$
$$\mid \langle a \rangle \varphi \mid [a]\, \varphi \mid \mu X^\eta . \varphi \mid \nu X^\eta . \varphi$$
$$\mid \lambda X^\eta . \varphi \mid \varphi_1\, \varphi_2$$
$$\eta \text{ (simple types)} ::= \mathtt{o} \mid \eta_1 \rightarrow \eta_2$$

The syntax of formulas on the first two lines is identical to that of the modal $\mu$-calculus formulas, except that the variable $X$ can range over *higher-order* predicates, rather than just propositions. Intuitively, $\mu X^\eta . \varphi$ ($\nu X^\eta . \varphi$, resp.) denotes the least (greatest, resp.) predicate of type $\eta$ such that $X = \varphi$. Higher-order predicates can be manipulated by using $\lambda$-abstractions and applications. The type $\mathtt{o}$ denotes the type of propositions. A *type environment* $\mathcal{H}$ for simple types is a map from a finite set of variables to the set of simple types. We often treat $\mathcal{H}$ as a set of type bindings of the form $X : \eta$, and write $X : \eta \in \mathcal{H}$ when $\mathcal{H}(X) = \eta$. A *type judgment relation* $\mathcal{H} \vdash \varphi : \eta$ is derived by the typing rules in Figure 1.

$$\frac{}{\mathcal{H} \vdash \mathtt{true} : \mathtt{o}} \qquad \frac{}{\mathcal{H} \vdash \mathtt{false} : \mathtt{o}} \qquad \frac{X : \eta \in \mathcal{H}}{\mathcal{H} \vdash X : \eta}$$

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

$$\frac{\mathcal{H} \cup \{X : \eta\} \vdash \varphi : \eta \quad X \notin dom(\mathcal{H})}{\mathcal{H} \vdash \mu X^\eta . \varphi : \eta} \qquad \frac{\mathcal{H} \cup \{X : \eta\} \vdash \varphi : \eta \quad X \notin dom(\mathcal{H})}{\mathcal{H} \vdash \nu X^\eta . \varphi : \eta}$$

$$\frac{\mathcal{H} \cup \{X : \eta_1\} \vdash \varphi : \eta_2 \quad X \notin dom(\mathcal{H})}{\mathcal{H} \vdash \lambda X^{\eta_1} . \varphi : \eta_1 \rightarrow \eta_2} \qquad \frac{\mathcal{H} \vdash \varphi_1 : \eta_2 \rightarrow \eta \quad \mathcal{H} \vdash \varphi_2 : \eta_2}{\mathcal{H} \vdash \varphi_1\, \varphi_2 : \eta}$$

**Fig. 1.** Typing rules for simple types

Note that, for each pair of a type environment $\mathcal{H}$ and an HFL formula $\varphi$, there is at most one simple type $\eta$ such that the type judgment relation $\mathcal{H} \vdash \varphi : \eta$ is derivable. We say an HFL formula $\varphi$ has type $\eta$ under a type environment $\mathcal{H}$ if the type judgment relation $\mathcal{H} \vdash \varphi : \eta$ is derivable.

For each simple type $\eta$, we define $order(\eta)$ inductively by: $order(\mathtt{o}) = 0$, $order(\eta_1 \rightarrow \eta_2) = \max(order(\eta_1) + 1, order(\eta_2))$. The order of an HFL formula $\varphi$ is the highest order of the types of the variables bound by $\mu$ or $\nu$ in $\varphi$. An order-0 HFL formula of type $\mathtt{o}$ can be viewed as a modal $\mu$-calculus formula, and vice versa. We write $FV(\varphi)$ for the set of free variables occurring in a formula $\varphi$. An HFL formula $\varphi$ is said to be *closed* if $FV(\varphi) = \emptyset$, and a closed formula is said to be *well-typed* if it has some simple type under the empty type environment.

*The semantics.* Let $\mathcal{L} = (Q, \mathcal{A}, \longrightarrow, q_0)$ be an LTS. The semantics of a well-typed HFL formula of type $\eta$ with respect to $\mathcal{L}$ is given as an element of a complete lattice $(D_{\mathcal{L},\eta}, \sqsubseteq_{\mathcal{L},\eta})$ defined by induction on the structure of $\eta$. For the base case, $(D_{\mathcal{L},\mathtt{o}}, \sqsubseteq_{\mathcal{L},\mathtt{o}})$ is defined by $D_{\mathcal{L},\mathtt{o}} = 2^Q$ and $\sqsubseteq_{\mathcal{L},\mathtt{o}} = \subseteq$, that is, $(D_{\mathcal{L},\mathtt{o}}, \sqsubseteq_{\mathcal{L},\mathtt{o}})$ is the

powerset lattice of the state set $Q$. For the step case, $D_{\mathcal{L},\eta_1 \to \eta_2}$ is defined as the set of monotonic functions from $D_{\mathcal{L},\eta_1}$ to $D_{\mathcal{L},\eta_2}$, and $\sqsubseteq_{\mathcal{L},\eta_1 \to \eta_2}$ is defined as the pointwise ordering over it.

For each type environment $\mathcal{H}$, we define $[\![\mathcal{H}]\!]_{\mathcal{L}}$ as the set of functions $\rho$ such that, for each $X \in dom(\mathcal{H})$, the image $\rho(X)$ is in the semantic domain of its type $\mathcal{H}(X)$, that is, $[\![\mathcal{H}]\!]_{\mathcal{L}} = \{\rho : dom(\mathcal{H}) \to \bigcup_\eta D_{\mathcal{L},\eta} \mid \forall X : \eta \in \mathcal{H}.\, \rho(X) \in D_{\mathcal{L},\eta}\}$. The interpretation of a type judgment relation $\mathcal{H} \vdash \varphi : \eta$ is a function $[\![\mathcal{H} \vdash \varphi : \eta]\!]_{\mathcal{L}} : [\![\mathcal{H}]\!]_{\mathcal{L}} \to D_{\mathcal{L},\eta}$ defined by induction on the derivation of $\mathcal{H} \vdash \varphi : \eta$ by:

$[\![\mathcal{H} \vdash \texttt{true} : \texttt{o}]\!]_{\mathcal{L}}(\rho) = Q$

$[\![\mathcal{H} \vdash \texttt{false} : \texttt{o}]\!]_{\mathcal{L}}(\rho) = \emptyset$

$[\![\mathcal{H} \vdash X : \eta]\!]_{\mathcal{L}}(\rho) = \rho(X)$

$[\![\mathcal{H} \vdash \varphi_1 \vee \varphi_2 : \texttt{o}]\!]_{\mathcal{L}}(\rho) = [\![\mathcal{H} \vdash \varphi_1 : \texttt{o}]\!]_{\mathcal{L}}(\rho) \cup [\![\mathcal{H} \vdash \varphi_2 : \texttt{o}]\!]_{\mathcal{L}}(\rho)$

$[\![\mathcal{H} \vdash \varphi_1 \wedge \varphi_2 : \texttt{o}]\!]_{\mathcal{L}}(\rho) = [\![\mathcal{H} \vdash \varphi_1 : \texttt{o}]\!]_{\mathcal{L}}(\rho) \cap [\![\mathcal{H} \vdash \varphi_2 : \texttt{o}]\!]_{\mathcal{L}}(\rho)$

$[\![\mathcal{H} \vdash \langle a \rangle \varphi : \texttt{o}]\!]_{\mathcal{L}}(\rho) = \{q \in Q \mid \exists q' \in [\![\mathcal{H} \vdash \varphi : \texttt{o}]\!]_{\mathcal{L}}(\rho).\, q \xrightarrow{a} q'\}$

$[\![\mathcal{H} \vdash [a] \varphi : \texttt{o}]\!]_{\mathcal{L}}(\rho) = \{q \in Q \mid \forall q' \in Q.\, q \xrightarrow{a} q' \Rightarrow q' \in [\![\mathcal{H} \vdash \varphi : \texttt{o}]\!]_{\mathcal{L}}(\rho)\}$

$[\![\mathcal{H} \vdash \mu X^\eta.\varphi : \eta]\!]_{\mathcal{L}}(\rho) = \bigsqcap_{\mathcal{L},\eta} \{d \in D_{\mathcal{L},\eta} \mid [\![\mathcal{H} \vdash \lambda X^\eta.\varphi : \eta \to \eta]\!]_{\mathcal{L}}(\rho)(d) \sqsubseteq_{\mathcal{L},\eta} d\}$

$[\![\mathcal{H} \vdash \nu X^\eta.\varphi : \eta]\!]_{\mathcal{L}}(\rho) = \bigsqcup_{\mathcal{L},\eta} \{d \in D_{\mathcal{L},\eta} \mid d \sqsubseteq_{\mathcal{L},\eta} [\![\mathcal{H} \vdash \lambda X^\eta.\varphi : \eta \to \eta]\!]_{\mathcal{L}}(\rho)(d)\}$

$[\![\mathcal{H} \vdash \lambda X^{\eta_1}.\varphi : \eta_1 \to \eta_2]\!]_{\mathcal{L}}(\rho) = \lambda d \in D_{\mathcal{L},\eta_1}.[\![\mathcal{H} \cup \{X : \eta_1\} \vdash \varphi : \eta_2]\!]_{\mathcal{L}}(\rho[X \mapsto d])$

$[\![\mathcal{H} \vdash \varphi_1\, \varphi_2 : \eta]\!]_{\mathcal{L}}(\rho) = [\![\mathcal{H} \vdash \varphi_1 : \eta_2 \to \eta]\!]_{\mathcal{L}}(\rho)([\![\mathcal{H} \vdash \varphi_2 : \eta_2]\!]_{\mathcal{L}}(\rho)).$

Here, $\rho[X \mapsto d]$ denotes the function $f$ such that $f(X) = d$ and $f(Y) = \rho(Y)$ for $Y \neq X$, and the unary operator $\bigsqcup_{\mathcal{L},\eta}$ ($\bigsqcap_{\mathcal{L},\eta}$, resp.) denotes the least upper bound (the greatest lower bound, resp.) with respect to $\sqsubseteq_{\mathcal{L},\eta}$.

Finally, for each closed HFL formula $\varphi$ of type $\eta$, we define the interpretation $[\![\varphi]\!]_{\mathcal{L}}$ by $[\![\varphi]\!]_{\mathcal{L}} = [\![\emptyset \vdash \varphi : \eta]\!]_{\mathcal{L}}(\rho_\emptyset)$, where $\rho_\emptyset$ is the empty map. We say that a closed propositional HFL formula $\varphi$ is *satisfied* by the state $q$ when $q \in [\![\varphi]\!]_{\mathcal{L}}$.

*Example 1.* Let $\varphi_1$ be $\mu F^{\texttt{o} \to \texttt{o}}.\lambda X^{\texttt{o}}.X \vee \langle a \rangle (F (\langle b \rangle X))$. The formula $\varphi_1 (\langle c \rangle \texttt{true})$ can be expanded to:

$$(\lambda X.\, X \vee \langle a \rangle (\varphi_1 (\langle b \rangle X))) (\langle c \rangle \texttt{true})$$
$$\equiv \langle c \rangle \texttt{true} \vee \langle a \rangle (\varphi_1 (\langle b \rangle \langle c \rangle \texttt{true}))$$
$$\equiv \langle c \rangle \texttt{true} \vee \langle a \rangle ((\lambda X.\, X \vee \langle a \rangle (\varphi_1 (\langle b \rangle X))) (\langle b \rangle \langle c \rangle \texttt{true}))$$
$$\equiv \langle c \rangle \texttt{true} \vee \langle a \rangle (\langle b \rangle \langle c \rangle \texttt{true} \vee \langle a \rangle (\varphi_1 (\langle b \rangle \langle b \rangle \langle c \rangle \texttt{true})))$$
$$\equiv \langle c \rangle \texttt{true} \vee \langle a \rangle \langle b \rangle \langle c \rangle \texttt{true} \vee \langle a \rangle \langle a \rangle \langle b \rangle \langle b \rangle \langle c \rangle \texttt{true} \vee \cdots .$$
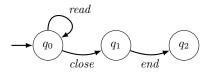
Thus, the formula $\varphi_1 (\langle c \rangle \texttt{true})$ describes the property that there exists a transition sequence of the form $a^n b^n c$ for some $n \geq 0$. As shown by this example, HFL is strictly more expressive than the modal $\mu$-calculus. $\qquad\square$

We write $\mathcal{L} \models \varphi$ when the initial state of $\mathcal{L}$ satisfies the property described by $\varphi$. The goal of HFL model checking is to decide, given $\mathcal{L}$ and $\varphi$ as input, whether $\mathcal{L} \models \varphi$ holds.

*Example 2.* To see how HFL model checking can be applied to program verification, let us consider the following OCaml-like program, which is a variation of the program considered in [10].

```
let rec f x k = if * then (close x; k())
                else (read x; read x; f x k) in
let d = open_in "foo" in f d (fun _ -> ())
```

Here, the asterisk `*` in the if-condition is a non-deterministic Boolean value. The program first opens the file `foo`, and then calls the function `f` with the opened file as an argument. The function `f` recursively reads the given file even times and closes it upon a non-deterministic condition.

Suppose we wish to check that the file `foo` is safely accessed as a read-only file. In the reduction methods by Kobayashi et al. [10], a program is transformed to an HFL formula that intuitively says "the behavior of the program conforms to the specification described as an LTS." In this case, the verification problem is reduced to the HFL model checking problem of deciding whether $\mathcal{L}_2 \models \varphi_2$ holds, where $\varphi_2 = (\nu F. \lambda k. \langle close \rangle k \wedge \langle read \rangle \langle read \rangle (F\, k))\, (\langle end \rangle \mathtt{true})$ and $\mathcal{L}_2$ is the following LTS, which models the access protocol for read-only files.



The formula $\varphi_2$ can be expanded to $\bigwedge_{n=0}^{\infty} \langle read \rangle^{2n} \langle close \rangle \langle end \rangle \mathtt{true}$, and checking whether $\mathcal{L}_2 \models \varphi_2$ holds is equivalent to checking whether (every prefix of) any sequence of the form $\mathtt{read}^{2n} \cdot \mathtt{close} \cdot \mathtt{end}$ belongs to the prefix-closure of $\mathtt{read}^* \cdot \mathtt{close} \cdot \mathtt{end}$, which is actually true. See [10] for systematic translations from program verification to HFL model checking.                                        □

## 2.2   Type-Based Characterization of HFL Model Checking

We now review the type-based characterization of the HFL model checking problem [9], which is going to be used as the basis of our algorithm given in Section 3.

To provide the type-based characterization, an HFL formula is represented in the form of a sequence of fixpoint equations $(F_1 : \eta_1 =_{\alpha_1} \varphi_1; \cdots; F_n : \eta_n =_{\alpha_n} \varphi_n)$, called a *hierarchical equation system* (HES). Here, for each $j \in \{1, \ldots, n\}$, $F_j$ is a distinct variable, $\alpha_j$ is either $\mu$ or $\nu$, and $\varphi_j$ is a fixpoint-free HFL formula that has type $\eta_j$ under the type environment $\{F_1 : \eta_1, \ldots, F_n : \eta_n\}$. We also require that if $\eta_j = \eta_{j,1} \to \cdots \to \eta_{j,\ell} \to \mathsf{o}$, then $\varphi_j$ is of the form $\lambda X_1^{\eta_{j,1}}. \cdots \lambda X_\ell^{\eta_{j,\ell}}. \psi_j$, where $\psi_j$ is a propositional formula that does not contain $\lambda$-abstractions. For each HES $\mathcal{E}$, we define a closed HFL formula $toHFL(\mathcal{E})$ inductively by:

$$toHFL(F : \eta =_\alpha \varphi) = \alpha F^\eta. \varphi$$
$$toHFL(\mathcal{E}; F : \eta =_\alpha \varphi) = toHFL([\alpha F^\eta. \varphi / F]\, \mathcal{E}),$$

where $[\varphi/X]\,\mathcal{E}$ denotes the HES obtained by replacing all free occurrences of the variable $X$ in $\mathcal{E}$ with the formula $\varphi$. Any HFL formula can be transformed to an HES, and vice versa. For example, $\nu X.\mu Y.(\langle a\rangle X \vee \langle b\rangle Y)$ can be expressed as an HES: $X =_\nu Y; Y =_\mu \langle a\rangle X \vee \langle b\rangle Y$. We write $\mathcal{L} \models \mathcal{E}$ when $\mathcal{L} \models toHFL(\mathcal{E})$ holds.

Given an LTS $\mathcal{L} = (Q, \mathcal{A}, \longrightarrow, q_0)$, the set of *(refinement) types* for HFL formulas, ranged over by $\tau$, is defined by:

$$\tau \ ::= \ q \mid \sigma \to \tau \qquad \sigma \ ::= \ \{\tau_1, \ldots, \tau_k\},$$

where $q$ ranges over $Q$. Intuitively, $q$ denotes the type of formulas that hold at state $q$. The type $\{\tau_1, \ldots, \tau_k\} \to \tau$ describes functions that take a value that has type $\tau_i$ for every $i \in \{1, \ldots, k\}$ as input, and return a value of type $\tau$ (thus, $\{\tau_1, \ldots, \tau_k\}$ is an intersection type). We often write $\top$ for $\emptyset$, and $\tau_1 \wedge \cdots \wedge \tau_k$ for $\{\tau_1, \ldots, \tau_k\}$. Henceforth, we just call $\tau$ and $\sigma$ *types*, and call those ranged over by $\eta$ *simple types* or *kinds*.

The *refinement relations* $\tau :: \eta$ and $\sigma :: \eta$, read "$\tau$ and $\sigma$ are refinements of $\eta$", are inductively defined by:

$$\frac{q \in Q}{q :: \mathsf{o}} \qquad \frac{\forall \tau \in \sigma.\, \tau :: \eta}{\sigma :: \eta} \qquad \frac{\sigma :: \eta_1 \quad \tau :: \eta_2}{\sigma \to \tau :: \eta_1 \to \eta_2}$$

Henceforth, we consider only those that are refinements of simple types, excluding out ill-formed types like $\{q, q \to q\} \to q$.

A *type environment* $\Gamma$ is a finite set of type bindings of the form $X : \tau$, where $X$ is a variable and $\tau$ is a type. Note that $\Gamma$ may contain more than one type binding for the same variable. We write $dom(\Gamma)$ for the set $\{X \mid \exists \tau.\, X : \tau \in \Gamma\}$ and $\Gamma(X)$ for the set $\{\tau \mid X : \tau \in \Gamma\}$. We also write $\{X : \sigma\}$ for the set $\{X : \tau_1, \ldots, X : \tau_k\}$ when $\sigma = \{\tau_1, \ldots, \tau_k\}$. The *type judgment relation* $\Gamma \vdash_{\mathcal{L}} \varphi : \tau$ for fixpoint-free formulas is defined by the typing rules in Figure 2.

The typability of an HES $\mathcal{E}$ is defined through the *typability game* $\mathbf{TG}(\mathcal{L}, \mathcal{E})$, which is an instance of parity games [5].

**Definition 1 (Typability Game).** *Let $\mathcal{L} = (Q, \mathcal{A}, \longrightarrow, q_0)$ be an LTS and $\mathcal{E} = (F_1 : \eta_1 =_{\alpha_1} \varphi_1; \cdots; F_n : \eta_n =_{\alpha_n} \varphi_n)$ be an HES with $\eta_1 = \mathsf{o}$. The typability game $\mathbf{TG}(\mathcal{L}, \mathcal{E})$ is a quintuple $(V_0, V_1, v_0, E_0 \cup E_1, \Omega)$, where:*

- $V_0 = \{F_j : \tau \mid j \in \{1, \ldots, n\}, \tau :: \eta_j\}$ *is the set of all type bindings.*
- $V_1 = \{\Gamma \mid \Gamma \subseteq V_0\}$ *is the set of all type environments.*
- $v_0 = F_1 : q_0 \in V_0$ *is the initial position.*
- $E_0 = \{(F_j : \tau,\ \Gamma) \in V_0 \times V_1 \mid \Gamma \vdash_{\mathcal{L}} \varphi_j : \tau\}.$
- $E_1 = \{(\Gamma,\ F_j : \tau) \in V_1 \times V_0 \mid F_j : \tau \in \Gamma\}.$
- $\Omega(F_j : \tau) = \Omega_j$ *for each $F_j : \tau \in V_0$, where $\Omega_j$ is inductively defined by: $\Omega_n = 0$ if $\alpha_n = \nu$, $\Omega_n = 1$ if $\alpha_n = \mu$; and for $i < n$, $\Omega_i = \Omega_{i+1}$ if $\alpha_i = \alpha_{i+1}$, and $\Omega_i = \Omega_{i+1} + 1$ if $\alpha_i \neq \alpha_{i+1}$. In other words, $\Omega_i$ $(1 \le i < n)$ is the least even (odd, resp.) number no less than $\Omega_{i+1}$ if $\alpha_i$ is $\nu$ ($\mu$, resp.).*
- $\Omega(\Gamma) = 0$ *for all $\Gamma \in V_1$.*

$$\frac{q \in Q}{\Gamma \vdash_{\mathcal{L}} \mathtt{true} : q} \quad \text{(T-True)} \qquad \frac{X : \tau \in \Gamma}{\Gamma \vdash_{\mathcal{L}} X : \tau} \quad \text{(T-Var)}$$

$$\frac{\Gamma \vdash_{\mathcal{L}} \varphi_i : q \text{ for some } i \in \{1, 2\}}{\Gamma \vdash_{\mathcal{L}} \varphi_1 \vee \varphi_2 : q} \quad \text{(T-Or)} \qquad \frac{\Gamma \vdash_{\mathcal{L}} \varphi_i : q \text{ for each } i \in \{1, 2\}}{\Gamma \vdash_{\mathcal{L}} \varphi_1 \wedge \varphi_2 : q} \quad \text{(T-And)}$$

$$\frac{\Gamma \vdash_{\mathcal{L}} \varphi : q' \text{ for some } q' \in Q \text{ with } q \xrightarrow{a} q'}{\Gamma \vdash_{\mathcal{L}} \langle a \rangle \varphi : q} \quad \text{(T-Some)}$$

$$\frac{\Gamma \vdash_{\mathcal{L}} \varphi : q' \text{ for each } q' \in Q \text{ with } q \xrightarrow{a} q'}{\Gamma \vdash_{\mathcal{L}} [a] \varphi : q} \quad \text{(T-All)}$$

$$\frac{\Gamma \cup \{X : \sigma\} \vdash_{\mathcal{L}} \varphi : \tau \quad X \notin dom(\Gamma) \quad \sigma :: \eta}{\Gamma \vdash_{\mathcal{L}} \lambda X^\eta.\varphi : \sigma \to \tau} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash_{\mathcal{L}} \varphi_1 : \sigma \to \tau \quad \Gamma \vdash_{\mathcal{L}} \varphi_2 : \tau' \text{ for each } \tau' \in \sigma}{\Gamma \vdash_{\mathcal{L}} \varphi_1 \, \varphi_2 : \tau} \quad \text{(T-App)}$$

$$\frac{\Gamma \vdash_{\mathcal{L}} \varphi : \tau \quad \tau \leq_{\mathcal{L}} \tau'}{\Gamma \vdash_{\mathcal{L}} \varphi : \tau'} \quad \text{(T-Sub)} \qquad \frac{q \in Q}{q \leq_{\mathcal{L}} q} \quad \text{(SubT-Base)}$$

$$\frac{\forall \tau' \in \sigma'. \, \exists \tau \in \sigma. \, \tau \leq_{\mathcal{L}} \tau'}{\sigma \leq_{\mathcal{L}} \sigma'} \quad \text{(SubT-Int)} \qquad \frac{\sigma' \leq_{\mathcal{L}} \sigma \quad \tau \leq_{\mathcal{L}} \tau'}{\sigma \to \tau \leq_{\mathcal{L}} \sigma' \to \tau'} \quad \text{(SubT-Fun)}$$

**Fig. 2.** Typing rules (where $\mathcal{L} = (Q, \mathcal{A}, \longrightarrow, q_0)$)

*A typability game is a two-player game played by player $0$ and player $1$. The set of positions $V_x$ belongs to player $x$. A* play *of a typability game is a sequence of positions $v_1 v_2 \ldots$ such that $(v_i, v_{i+1}) \in E_0 \cup E_1$ holds for each adjacent pair $v_i v_{i+1}$. A maximal finite play $v_1 v_2 \ldots v_k$ is won by player $x$ iff $v_k \in V_{1-x}$, and an infinite play $v_1 v_2 \ldots$ is won by player $x$ iff $\limsup_{i \to \infty} \Omega(v_i) = x \pmod 2$). We say a typability game is* winning *if the initial position $v_0$ is a winning position for player $0$, and call a winning strategy for her from $v_0$ simply a* winning strategy *of the game (such a strategy can be given as a partial function from $V_0$ to $V_1$).*

Intuitively, in the position $F_j : \tau$, player 0 is asked to show why $F_j$ has type $\tau$, by providing a type environment $\Gamma$ under which the body $\varphi_j$ of $F_j$ has type $\tau$. Player 1 then challenges player 0's assumption $\Gamma$, by picking a type binding $F' : \tau' \in \Gamma$ and asking why $F'$ has type $\tau'$. A play may continue indefinitely, in which case player 0 wins if the largest priority visited infinitely often is even.

The following characterization is the basis of our algorithm.

**Theorem 1 ([9]).** *Let $\mathcal{L}$ be an LTS and $\mathcal{E} = (F_1 : \eta_1 =_{\alpha_1} \varphi_1; \cdots; F_n : \eta_n =_{\alpha_n} \varphi_n)$ be an HES with $\eta_1 = \mathsf{o}$. Then, $\mathcal{L} \models \mathcal{E}$ if and only if the typability game $\mathbf{TG}(\mathcal{L}, \mathcal{E})$ is winning.*

*Example 3.* Let $\mathcal{E}_{\mathsf{ex}}$ be the following HES:

$$S =_\nu \langle a \rangle (F (\langle b \rangle S)); \quad F =_\mu \lambda X^\circ. X \vee \langle c \rangle S \vee \langle a \rangle (F (\langle b \rangle X)).$$

It expresses the property that there is an infinite sequence that can be partitioned into chunks of the form $a^k b^k$ or $a^k c$ (where $k \geq 1$), like $a^3 b^3 a^2 c a^2 b^2 a^3 c \cdots$.

Let $\mathcal{L}_{\mathtt{ex}}$ be an LTS shown on the left side of Figure 3. It satisfies the HES $\mathcal{E}_{\mathtt{ex}}$ as the sequence $abacabac\cdots$ is enabled at the initial state $q_0$. The corresponding typability game $\mathbf{TG}(\mathcal{L}_{\mathtt{ex}}, \mathcal{E}_{\mathtt{ex}})$ is defined as shown (partially) on the right side of Figure 3, and a winning strategy (depicted by two-headed arrows) is witnessed by the type judgments $\{S : q_2, F : q_1 \rightarrow q_1\} \vdash_{\mathcal{L}_{\mathtt{ex}}} \varphi_S : q_0$, $\{F : \top \rightarrow q_0\} \vdash_{\mathcal{L}_{\mathtt{ex}}} \varphi_S : q_2$, $\emptyset \vdash_{\mathcal{L}_{\mathtt{ex}}} \varphi_F : q_1 \rightarrow q_1$, and $\{S : q_0\} \vdash_{\mathcal{L}_{\mathtt{ex}}} \varphi_F : \top \rightarrow q_0$, where $\varphi_S$ and $\varphi_F$ denote the right-hand side formulas of the variables $S$ and $F$, respectively.                     □
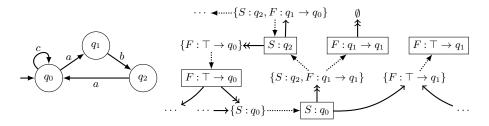


**Fig. 3.** LTS $\mathcal{L}_{\mathtt{ex}}$ (on the left side) and a part of the corresponding typability game $\mathbf{TG}(\mathcal{L}_{\mathtt{ex}}, \mathcal{E}_{\mathtt{ex}})$ (on the right side)

## 3   A Practical Algorithm for HFL Model Checking

We present our algorithm for HFL model checking in this section.

Theorem 1 immediately yields a naive model checking algorithm, which first constructs the typability game $\mathbf{TG}(\mathcal{L}, \mathcal{E})$ (note that $\mathbf{TG}(\mathcal{L}, \mathcal{E})$ is finite), and solves it by using an algorithm for parity game solving. Unfortunately, the algorithm does not work in practice, since the size of $\mathbf{TG}(\mathcal{L}, \mathcal{E})$ is too large; it is $k$-fold exponential in the size of $\mathcal{L}$ and $\mathcal{E}$, for order-$k$ HES.

The basic idea of our algorithm is to construct a subgame $\mathbf{TG}'(\mathcal{L}, \mathcal{E})$ of $\mathbf{TG}(\mathcal{L}, \mathcal{E})$, so that $\mathbf{TG}'(\mathcal{L}, \mathcal{E})$ is winning if and only if the original game $\mathbf{TG}(\mathcal{L}, \mathcal{E})$ is winning, and that $\mathbf{TG}'(\mathcal{L}, \mathcal{E})$ is often significantly smaller than $\mathbf{TG}(\mathcal{L}, \mathcal{E})$. The main question is of course how to construct such a subgame. Our approach is to consider a series of recursion-free[2] approximations $\mathcal{E}^{(0)}, \mathcal{E}^{(1)}, \mathcal{E}^{(2)}, \ldots$ of $\mathcal{E}$, which are obtained by unfolding fixpoint variables in $\mathcal{E}$ a certain number of times, and are free from fixpoint operators. The key observations are: (i) for sufficiently large $m$ (that may depend on $\mathcal{L}$ and $\mathcal{E}$), $\mathcal{L} \models \mathcal{E}^{(m)}$ if and only if $\mathcal{L} \models \mathcal{E}$, (ii) for such $m$, a winning strategy for $\mathbf{TG}(\mathcal{L}, \mathcal{E})$ can be constructed by using only the types used in a winning strategy for $\mathbf{TG}(\mathcal{L}, \mathcal{E}^{(m)})$, and (iii) (a superset of) the types needed in a winning strategy for $\mathbf{TG}(\mathcal{L}, \mathcal{E}^{(m)})$ can be computed effectively (and with reasonable efficiency for typical inputs), based on a method

---

[2]  We say an HES is recursion-free if there is no cyclic dependency on fixpoint variables, so that fixpoint variables can be completely eliminated by unfolding them; we omit the formal definition.

similar to saturation-based algorithms for HORS model checking [4,18]. (These observations are not trivial; they will be justified when we discuss the correctness of the algorithm in Section 4.)

In the rest of this section, we first explain more details about the intuitions behind our algorithm in Section 3.1.[3] We also introduce some definitions such as $\mathcal{E}^{(m)}$ during the course of explanation. These concepts are not directly used in the actual algorithm, but would help readers understand intuitions behind the algorithm. We then describe the algorithm in Section 3.2.

### 3.1   The Idea of the Algorithm

We first define a non-recursive HES as an approximation of $\mathcal{E}$. By the Kleene fixpoint theorem, we can approximate $\mathcal{E}$ by unfolding fixpoint variables finitely often, and the approximation becomes exact when the depth of unfolding is sufficiently large. Such an approximation can be naturally represented by a non-recursive HES $\mathcal{E}^{(m)}$ defined as follows.

**Definition 2 (Non-Recursive HES $\mathcal{E}^{(m)}$).** *Let $\mathcal{E} = (F_1 : \eta_1 =_{\alpha_1} \varphi_1; \cdots; F_n : \eta_n =_{\alpha_n} \varphi_n)$ be an HES and $m$ be a positive integer. We define $\mathcal{E}^{(m)} = (F_1^{(m)} : \eta_1 =_{\alpha_1} \varphi_1^{(m)}; \cdots)$ as a non-recursive HES consisting of equations of the form $F_j^{\boldsymbol{\beta}} : \eta_j =_{\alpha_j} \varphi_j^{\boldsymbol{\beta}}$.[4] Here, $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_j)$ is a tuple of integers satisfying $0 \leq \beta_k \leq m$ for each $k \in \{1, \ldots, j\}$, and $\varphi_j^{\boldsymbol{\beta}}$ is an HFL formula defined by:*

$$\varphi_j^{\boldsymbol{\beta}} = \begin{cases} \lambda X_1^{\eta_{j,1}}. \cdots \lambda X_\ell^{\eta_{j,\ell}}. \widehat{\alpha_j} & (\text{if } \beta_j = 0) \\ [F_1^{\boldsymbol{\beta}(1)}/F_1, \ldots, F_n^{\boldsymbol{\beta}(n)}/F_n] \varphi_j & (\text{if } \beta_j \neq 0). \end{cases}$$

*Here, $\eta_j = \eta_{j,1} \to \cdots \to \eta_{j,\ell} \to \mathsf{o}$, $\widehat{\nu} = \mathtt{true}$, $\widehat{\mu} = \mathtt{false}$, and $\boldsymbol{\beta}(k)$ is defined by:*

$$\boldsymbol{\beta}(k) = \begin{cases} (\beta_1, \ldots, \beta_k) & (\text{if } k < j) \\ (\beta_1, \ldots, \beta_j - 1, \underbrace{m, \ldots, m}_{(k-j) \ times}) & (\text{if } j \leq k). \end{cases}$$

*We call the superscript $\boldsymbol{\beta}$ an* index. *Intuitively, an index $\boldsymbol{\beta}$ indicates how many unfoldings are left to be done to obtain the formula represented by $\mathcal{E}^{(m)}$.*

*Example 4.* Recall the HES $\mathcal{E}_{\mathtt{ex}}$ in Example 3:

$$S =_\nu \langle a \rangle (F (\langle b \rangle S)); \quad F =_\mu \lambda X^\circ. X \vee \langle c \rangle S \vee \langle a \rangle (F (\langle b \rangle X)).$$

Then, a finite approximation $\mathcal{E}_{\mathtt{ex}}^{(1)}$ is:

$S^{(1)} =_\nu \langle a \rangle (F^{(0,1)} (\langle b \rangle S^{(0)})); \ S^{(0)} =_\nu \mathtt{true};$
$F^{(0,1)} =_\mu \lambda X^\circ. X \vee \langle c \rangle S^{(0)} \vee \langle a \rangle (F^{(0,0)} (\langle b \rangle X)); \ F^{(0,0)} =_\mu \lambda X^\circ. \mathtt{false};$
$F^{(1,1)} =_\mu \lambda X^\circ. X \vee \langle c \rangle S^{(1)} \vee \langle a \rangle (F^{(1,0)} (\langle b \rangle X)); \ F^{(1,0)} =_\mu \lambda X^\circ. \mathtt{false}. \quad \square$

---

[3]   Those intuitions may not be clear for non-expert readers. In such a case, readers may safely skip the subsection (except the definitions) and proceed to Section 3.2.

[4]   Since $\mathcal{E}^{(m)}$ does not contain recursion, the order of equations (other than the first one) does not matter.

For $\mathcal{E}^{(m)}$, its validity can be checked by "unfolding" all the fixpoint variables. The operation of unfolding is formally expressed by the following rewriting relation.

**Definition 3 (Rewriting Relation on HFL formulas).** *Let $\mathcal{E} = (F_1 : \eta_1 =_{\alpha_1} \varphi_1; \cdots; F_n : \eta_n =_{\alpha_n} \varphi_n)$ be an HES. The rewriting relation $\longrightarrow_{\mathcal{E}}$ is defined by the rule:*

$$C[F_j \, \chi_1 \cdots \chi_\ell] \longrightarrow_{\mathcal{E}} C[[\chi_1/X_1, \ldots, \chi_\ell/X_\ell] \, \psi_j] \; if \; \varphi_j = \lambda X_1. \cdots \lambda X_\ell.\psi_j.$$

*Here, $C$ ranges over the set of contexts defined by:*

$$C ::= [\,] \mid C \vee \chi \mid \chi \vee C \mid C \wedge \chi \mid \chi \wedge C \mid \langle a \rangle C \mid [a] \, C,$$

*and $C[\chi]$ denotes the formula obtained from $C$ by replacing $[\,]$ with $\chi$. We write $\longrightarrow_{\mathcal{E}}^*$ for the reflexive transitive closure of the relation $\longrightarrow_{\mathcal{E}}$.*

Note that the relation $\longrightarrow_{\mathcal{E}}$ preserves simple types and the semantics of formulas. By the strong normalization property of the simply-typed $\lambda$-calculus, if the HES $\mathcal{E}$ does not contain recursion, it is strongly-normalizing. Thus, for $\mathcal{E}^{(m)}$, the initial variable $F_1^{(m)}$ (which is assumed to have type $\mathsf{o}$) can be rewritten to a formula $\chi$ without any fixpoint variables, such that an LTS $\mathcal{L}$ satisfies $\chi$ if and only if $\mathcal{L}$ satisfies $\mathcal{E}^{(m)}$ (and for sufficiently large $m$, if and only if $\mathcal{L}$ satisfies $\mathcal{E}$). Furthermore, if the initial state $q_0$ of $\mathcal{L}$ satisfies $\chi$, then from the reduction sequence:

$$F_1^{(m)} = \chi_0 \longrightarrow_{\mathcal{E}^{(m)}} \chi_1 \longrightarrow_{\mathcal{E}^{(m)}} \cdots \longrightarrow_{\mathcal{E}^{(m)}} \chi_{m'} = \chi,$$

one can compute a series of type environments $\Gamma_{m'} = \emptyset, \Gamma_{m'-1}, \ldots, \Gamma_1, \Gamma_0 = \{F_1^{(m)} : q_0\}$ such that $\Gamma_i \vdash_{\mathcal{L}} \chi_i : q_0$ in a backward manner, by using the standard subject expansion property of intersection type systems (i.e., the property that typing is preserved by backward reductions) [3]. These type environments provide sufficient type information, so that a winning strategy for the typability game $\mathbf{TG}(\mathcal{L}, \mathcal{E}^{(m)})$ can be expressed only by using type bindings in $\Gamma^{(m)} = \Gamma_{m'} \cup \cdots \cup \Gamma_0$. If $m$ is sufficiently large, by using the same type bindings (but ignoring indices), we can also express a winning strategy for $\mathbf{TG}(\mathcal{L}, \mathcal{E})$. Thus, if we can compute (a possible overapproximation of) $\Gamma^{(m)}$ above, we can restrict the game $\mathbf{TG}(\mathcal{L}, \mathcal{E})$ to the subgame $\mathbf{TG}'(\mathcal{L}, \mathcal{E})$ consisting of only types occurring in $\Gamma^{(m)}$, without changing the winner.

The remaining issue is how to compute an overapproximation of $\Gamma^{(m)}$. It is unreasonable to compute it directly based on the definition above, as the "sufficiently large" $m$ is huge in general, and the number $m'$ of reduction steps may also be too large. Instead, we relax the rewriting relation $\longrightarrow_{\mathcal{E}}$ for the original HES $\mathcal{E}$ by adding the following rules:

$$C[F_j \, \chi_1 \cdots \chi_\ell] \longrightarrow_{\mathcal{E}}' \begin{cases} C[\mathtt{true}] & \text{if } \alpha_j = \nu \\ C[\mathtt{false}] & \text{if } \alpha_j = \mu. \end{cases}$$

The resulting relation $\longrightarrow_{\mathcal{E}}'$ simulates $\longrightarrow_{\mathcal{E}^{(m)}}$ for arbitrary $m$, in the sense that for any reduction sequence:

$$F_1^{(m)} = \chi_0 \longrightarrow_{\mathcal{E}^{(m)}} \chi_1 \longrightarrow_{\mathcal{E}^{(m)}} \cdots \longrightarrow_{\mathcal{E}^{(m)}} \chi_{m'} = \chi,$$

there exists a corresponding reduction sequence:

$$F_1 = \chi'_0 \longrightarrow'_\mathcal{E} \chi'_1 \longrightarrow'_\mathcal{E} \cdots \longrightarrow'_\mathcal{E} \chi'_{m'} = \chi,$$

where each $\chi'_i$ is the formula obtained by removing indices from $\chi_i$. Thus, to compute $\Gamma^{(m)}$, it suffices to compute type environments for $\chi'_i$'s, based on the subject expansion property. We can do so by using the function $\mathcal{F}_{\mathcal{L},\mathcal{E}}$ defined below, without explicitly constructing reduction sequences.

**Definition 4 (Backward Expansion Function $\mathcal{F}_{\mathcal{L},\mathcal{E}}$).** *Let $\mathcal{L}$ be an LTS, and $\mathcal{E} = (F_1 : \eta_1 =_{\alpha_1} \varphi_1; \cdots; F_n : \eta_n =_{\alpha_n} \varphi_n)$ be an HES. Let $\mathcal{T}_\mathcal{E}$ denote the set of all type environments for the fixpoint variables of $\mathcal{E}$, that is, $\mathcal{T}_\mathcal{E} = \{\, \Gamma \mid dom(\Gamma) \subseteq \{F_1, \ldots, F_n\}, \forall F_j : \tau \in \Gamma. \tau :: \eta_j \,\}$. The function $\mathcal{F}_{\mathcal{L},\mathcal{E}} : \mathcal{T}_\mathcal{E} \to \mathcal{T}_\mathcal{E}$ is a monotonic function defined by:*

$$\mathcal{F}_{\mathcal{L},\mathcal{E}}(\Gamma) = \Gamma \cup \{\, F_j : \tau \mid \tau :: \eta_j$$
$$\varphi_j = \lambda X_1. \cdots \lambda X_\ell.\psi_j,$$
$$\tau = \sigma_1 \to \cdots \to \sigma_\ell \to q,$$
$$\exists \Delta.\, dom(\Delta) \subseteq FV(\psi_j) \cap \{X_1, \ldots, X_\ell\},$$
$$\Gamma \cup \Delta \vdash_\mathcal{L} \psi_j : q, \forall k \in \{1, \ldots, \ell\}. \sigma_k = \Delta(X_k),$$
$$\forall X_i \in dom(\Delta). \exists \varphi \in Flow_\mathcal{E}(X_i). \forall \tau' \in \Delta(X_i). \Gamma \vdash_\mathcal{L} \varphi : \tau' \,\}.$$

*Here, $Flow_\mathcal{E}(X_i)$ denotes the set $\{\, \xi_i \mid F_1 \longrightarrow^*_\mathcal{E} C[F\,\xi_1 \cdots \xi_\ell] \,\}$, where $X_i$ is the i-th formal parameter of $F$ (i.e., the equation of $F$ is of the form $F =_\alpha \lambda X_1. \cdots \lambda X_\ell.\psi).$*[5]

The following lemma justifies the definition of $\mathcal{F}_{\mathcal{L},\mathcal{E}}$ (see Appendix A for a proof). It states that the function $\mathcal{F}_{\mathcal{L},\mathcal{E}}$ expands type environments in such a way that we can go backwards through the rewriting relation $\longrightarrow_\mathcal{E}$ without losing the typability.

**Lemma 1.** *If $F_1 \longrightarrow^*_\mathcal{E} \varphi \longrightarrow_\mathcal{E} \varphi'$ and $\Gamma \vdash_\mathcal{L} \varphi' : q$, then $\mathcal{F}_{\mathcal{L},\mathcal{E}}(\Gamma) \vdash_\mathcal{L} \varphi : q$.*

Let $\Gamma_0$ be the set of strongest type bindings (with respect to subtyping) for the $\nu$-variables of $\mathcal{E}$, that is, $\Gamma_0 = \{F_j : \tau \mid \alpha_j = \nu, \tau :: \eta_j, \tau = \top \to \cdots \to \top \to q\}$. The following lemma states that, if we are allowed to use the strongest type bindings contained in $\Gamma_0$, then we can also go backwards through the relaxed rewriting relation $\longrightarrow'_\mathcal{E}$ using the same function $\mathcal{F}_{\mathcal{L},\mathcal{E}}$.

**Lemma 2.** *If $F_1 \longrightarrow'^*_\mathcal{E} \varphi \longrightarrow'_\mathcal{E} \varphi'$, $\Gamma \vdash_\mathcal{L} \varphi':q$, and $\Gamma \supseteq \Gamma_0$, then $\mathcal{F}_{\mathcal{L},\mathcal{E}}(\Gamma) \vdash_\mathcal{L} \varphi:q$.*

Let us write $(\mathcal{F}_{\mathcal{L},\mathcal{E}})^\omega(\Gamma_0)$ for $\bigcup_{i\in\omega}(\mathcal{F}_{\mathcal{L},\mathcal{E}})^i(\Gamma_0)$. As an immediate corollary of Lemma 2, we have: if $F_1 = \chi'_0 \longrightarrow'_\mathcal{E} \chi'_1 \longrightarrow'_\mathcal{E} \cdots \longrightarrow'_\mathcal{E} \chi'_{m'} = \chi$ and $\emptyset \vdash_\mathcal{L} \chi : q_0$, then $(\mathcal{F}_{\mathcal{L},\mathcal{E}})^\omega(\Gamma_0) \vdash_\mathcal{L} \chi'_i : q_0$ for every $i$. Thus, $(\mathcal{F}_{\mathcal{L},\mathcal{E}})^\omega(\Gamma_0)$ serves as an overapproximation of $\Gamma^{(m)}$ mentioned above.

---
**Algorithm 1** The proposed HFL model checking algorithm

---
$\Gamma := \Gamma_0$
**while** $\Gamma \neq \mathcal{F}'_{\mathcal{L},\mathcal{E}}(\Gamma)$ **do**
    $\Gamma := \mathcal{F}'_{\mathcal{L},\mathcal{E}}(\Gamma)$
**end while**
**return** whether the subgame $\mathbf{SG}(\mathcal{L},\mathcal{E},\Gamma)$ is winning

---

### 3.2   The Algorithm

Based on the intuitions explained in Section 3.1, we propose the algorithm shown in Algorithm 1.

In the algorithm, the function $\mathcal{F}'_{\mathcal{L},\mathcal{E}}$ is an overapproximation of the function $\mathcal{F}_{\mathcal{L},\mathcal{E}}$, obtained by replacing $Flow_\mathcal{E}$ in the definition of $\mathcal{F}_{\mathcal{L},\mathcal{E}}$ with an overapproximation $Flow'_\mathcal{E}$ satisfying $\forall X.\, Flow_\mathcal{E}(X) \subseteq Flow'_\mathcal{E}(X)$. This is because it is in general too costly to compute the exact flow set $Flow_\mathcal{E}(X)$. The overapproximation $Flow'_\mathcal{E}$ can typically be computed by flow analysis algorithms for functional programs, such as 0-CFA [17]. The first four lines compute $\bigcup_{i\in\omega}(\mathcal{F}'_{\mathcal{L},\mathcal{E}})^i(\Gamma_0)$, which is an overapproximation of $\bigcup_{i\in\omega}(\mathcal{F}_{\mathcal{L},\mathcal{E}})^i(\Gamma_0)$ discussed in the previous subsection.

$\mathbf{SG}(\mathcal{L},\mathcal{E},\Gamma)$ on the last line denotes the subgame of $\mathbf{TG}(\mathcal{L},\mathcal{E})$, obtained by restricting the game arena. It is defined as follows.

**Definition 5 (Subgame).** *Let $\mathcal{L} = (Q,\mathcal{A},\longrightarrow,q_0)$ be an LTS, $\mathcal{E} = (F_1:\eta_1 =_{\alpha_1} \varphi_1;\cdots;F_n:\eta_n =_{\alpha_n} \varphi_n)$ be an HES with $\eta_1 = \mathsf{o}$, and $\Gamma \in \mathcal{T}_\mathcal{E}$ be a type environment for $\mathcal{E}$. The subgame $\mathbf{SG}(\mathcal{L},\mathcal{E},\Gamma)$ is a parity game defined the same as $\mathbf{TG}(\mathcal{L},\mathcal{E})$ except that the set of positions is restricted to the subsets of $\Gamma$. That is, for $\mathbf{TG}(\mathcal{L},\mathcal{E}) = (V'_0, V'_1, v'_0, E'_0 \cup E'_1, \Omega')$, $\mathbf{SG}(\mathcal{L},\mathcal{E},\Gamma)$ is the parity game $(V_0, V_1, v_0, E_0 \cup E_1, \Omega)$, where:*

- *$V_0 = \Gamma \cup \{v'_0\}$, $V_1 = \{\,\Gamma' \mid \Gamma' \subseteq \Gamma\,\}$,*
  *$v_0 = v'_0$, $E_0 = E'_0 \cap (V_0 \times V_1)$, $E_1 = E'_1 \cap (V_1 \times V_0)$.*
- *$\Omega$ is the restriction of $\Omega'$ to $V_0 \cup V_1$.*

The following theorem claims the correctness of the algorithm.

**Theorem 2 (Correctness).** *Let $\mathcal{L}$ be an LTS and $\mathcal{E} = (F_1:\eta_1 =_{\alpha_1} \varphi_1;\cdots;F_n:\eta_n =_{\alpha_n} \varphi_n)$ be an HES with $\eta_1 = \mathsf{o}$. Algorithm 1 terminates. Furthermore, it returns "yes" if and only if $\mathcal{L} \models \mathcal{E}$.*

*Example 5.* Recall the HES $\mathcal{E}_{\mathsf{ex}}$ and the LTS $\mathcal{L}_{\mathsf{ex}}$ in Example 3. The fixpoint computation from the initial type environment $\Gamma_0 = \{S:q_0,\, S:q_1,\, S:q_2\}$ by the function $\mathcal{F}_{\mathcal{L}_{\mathsf{ex}},\mathcal{E}_{\mathsf{ex}}}$ (with a few simple optimizations)[6] proceeds as shown in Table 1. Note that the flow set $Flow_{\mathcal{E}_{\mathsf{ex}}}(X)$ for the formal parameter $X$ of the

---
[5]   Without loss of generality, we assume that $X_1,\ldots,X_\ell$ are distinct from each other and do not occur in the other equations.

[6]   Using subtyping relations, we can refrain from unnecessary type derivations like $\{S:q_0, X:q_1\} \vdash_{\mathcal{L}_{\mathsf{ex}}} \psi_F : q_0$ in this example. See Appendix C for more details.

fixpoint variable $F$ is calculated as $\{\langle b\rangle S, \langle b\rangle\langle b\rangle S, \ldots\}$, and thus the only candidates for the type environment $\Delta$ in the algorithm are $\Delta = \emptyset$ and $\Delta = \{X : q_1\}$. The expansion reaches the fixpoint after two iterations,[7] and the algorithm returns "yes" as the resulting type environment contains sufficient type bindings to construct the winning strategy depicted in Figure 3 of Example 3.    □

**Table 1.** Fixpoint computation by the function $\mathcal{F}_{\mathcal{L}_{\mathrm{ex}},\mathcal{E}_{\mathrm{ex}}}$

| Iteration number $k$ | Type environment $\Gamma_k$ | Newly derivable type judgments |
|---|---|---|
| 0 | $\{S : q_0,\ S : q_1,\ S : q_2\}$ | $\{X : q_1\} \vdash_{\mathcal{L}_{\mathrm{ex}}} \psi_F : q_1$ <br> $\{S : q_0\} \vdash_{\mathcal{L}_{\mathrm{ex}}} \psi_F : q_0$ |
| 1 | $\Gamma_0 \cup \{F : q_1 \to q_1,\ F : \top \to q_0\}$ | $\{F : \top \to q_0\} \vdash_{\mathcal{L}_{\mathrm{ex}}} \psi_F : q_2$ |
| 2 | $\Gamma_1 \cup \{F : \top \to q_2\}$ | - |

## 4    Correctness of the Algorithm

We sketch a proof of Theorem 2 in this section. A more detailed proof is found in Appendix B. We discuss soundness and completeness (Theorems 3 and 4 below) separately, from which Theorem 2 follows.

### 4.1    Soundness of the Algorithm

The soundness of the algorithm follows immediately from the fact that the replacement of $\mathbf{TG}(\mathcal{L}, \mathcal{E})$ with the subgame $\mathbf{SG}(\mathcal{L}, \mathcal{E}, \Gamma)$ restricts only the moves of player 0, so that the resulting game is harder for her to win.

**Theorem 3 (Soundness).** *Let $\mathcal{L}$ be an LTS and $\mathcal{E} = (F_1 : \eta_1 =_{\alpha_1} \varphi_1; \cdots; F_n : \eta_n =_{\alpha_n} \varphi_n)$ be an HES with $\eta_1 = \mathsf{o}$. If $\mathcal{L} \nvDash \mathcal{E}$, then the algorithm returns "no", that is, the subgame $\mathbf{SG}(\mathcal{L}, \mathcal{E}, (\mathcal{F}'_{\mathcal{L},\mathcal{E}})^\omega(\Gamma_0))$ is not winning.*

*Proof.* We show the contraposition. Suppose that $\mathbf{SG}(\mathcal{L}, \mathcal{E}, (\mathcal{F}'_{\mathcal{L},\mathcal{E}})^\omega(\Gamma_0))$ is a winning game. Then, there exists a winning strategy $\varsigma$ of player 0 for the node $F_1 : q_0$ in that game. This strategy $\varsigma$ also gives a winning strategy of player 0 for the node $F_1 : q_0$ in the original typability game $\mathbf{TG}(\mathcal{L}, \mathcal{E})$; note that for each position $\Gamma \in V_1$ of $\mathbf{SG}(\mathcal{L}, \mathcal{E}, (\mathcal{F}'_{\mathcal{L},\mathcal{E}})^\omega(\Gamma_0))$, the set of possible moves of player 1 in $\mathbf{TG}(\mathcal{L}, \mathcal{E})$ is the same as that in $\mathbf{SG}(\mathcal{L}, \mathcal{E}, (\mathcal{F}'_{\mathcal{L},\mathcal{E}})^\omega(\Gamma_0))$. Therefore, $\mathcal{L} \models \mathcal{E}$ follows from Theorem 1.    □

---

[7]    Actually, our prototype model checker reported in Section 5 does not derive the type binding $F : \top \to q_2$, and thus the computation terminates in one iteration. This is because it uses information of types in flow analysis, which reveals that it does not affect the result whether formulas of the form $F\,\varphi$ have type $q_2$. See Appendix C.

### 4.2   Completeness of the Algorithm

The completeness of the algorithm is stated as Theorem 4 below.

**Theorem 4 (Completeness).** *Let $\mathcal{L}$ be an LTS and $\mathcal{E} = (F_1 : \eta_1 =_{\alpha_1} \varphi_1; \cdots; F_n : \eta_n =_{\alpha_n} \varphi_n)$ be an HES with $\eta_1 = \mathsf{o}$. If $\mathcal{L} \models \mathcal{E}$, then the algorithm returns "yes", that is, the subgame $\mathbf{SG}(\mathcal{L}, \mathcal{E}, (\mathcal{F}'_{\mathcal{L},\mathcal{E}})^\omega(\Gamma_0))$ is winning.*

The proof follows the intuitions provided in Section 3.1. Given a type environment $\Gamma$ for $\mathcal{E}^{(m)}$, we write $Forget(\Gamma)$ for the type environment obtained by removing all the indices from $\Gamma$. Theorem 4 follows immediately from Lemmas 3 and 4 below.

**Lemma 3.** *If the typability game $\mathbf{TG}(\mathcal{L}, \mathcal{E})$ is winning, then for sufficiently large $m$, the subgame $\mathbf{SG}(\mathcal{L}, \mathcal{E}, Forget((\mathcal{F}_{\mathcal{L},\mathcal{E}^{(m)}})^\omega(\emptyset)))$ is also winning.*

**Lemma 4.** *$Forget((\mathcal{F}_{\mathcal{L},\mathcal{E}^{(m)}})^\omega(\emptyset))) \subseteq (\mathcal{F}_{\mathcal{L},\mathcal{E}})^\omega(\Gamma_0)$.*

Note that Lemma 4 implies that the game $\mathbf{SG}(\mathcal{L}, \mathcal{E}, (\mathcal{F}'_{\mathcal{L},\mathcal{E}})^\omega(\Gamma_0))$ is more advantageous for player 0 than the game $\mathbf{SG}(\mathcal{L}, \mathcal{E}, Forget((\mathcal{F}_{\mathcal{L},\mathcal{E}^{(m)}})^\omega(\emptyset)))$, which is winning when $\mathcal{L} \models \mathcal{E}$ by Lemma 3.

Lemma 4 should be fairly obvious, based on the intuitions given in Section 3.1. Technically, it suffices to show that $F_j^{\boldsymbol{\beta}} : \tau \in (\mathcal{F}_{\mathcal{L},\mathcal{E}^{(m)}})^i(\emptyset)$ implies $F_j : \tau \in (\mathcal{F}_{\mathcal{L},\mathcal{E}})^i(\Gamma_0)$ by induction on $i$, with case analysis on $\beta_j$. If $\beta_j = 0$, then $\alpha_j = \nu$ and the body of $F_j^{\boldsymbol{\beta}}$ is $\lambda \widetilde{X}.\mathtt{true}$. Thus, $F_j : \tau = F_j : \top \to \cdots \to \top \to q \in \Gamma_0 \subseteq (\mathcal{F}_{\mathcal{L},\mathcal{E}})^i(\Gamma_0)$. If $\beta_j > 0$, then the body of $F_j^{\boldsymbol{\beta}}$ is the same as that of $F_j$ except indices. Thus, $F_j : \tau \in (\mathcal{F}_{\mathcal{L},\mathcal{E}})^i(\Gamma_0)$ follows from the induction hypothesis and the definition of the function $\mathcal{F}_{\mathcal{L},\mathcal{E}}$. See Appendix B for details.

To prove Lemma 3, we define another function *Regress* on type environments. Let $\preceq_k$ be the lexicographic ordering on the first $k$ elements of tuples of integers, and $\prec_k$ be its strict version. We write $\boldsymbol{\beta}_1 =_k \boldsymbol{\beta}_2$ if $\boldsymbol{\beta}_1 \preceq_k \boldsymbol{\beta}_2$ and $\boldsymbol{\beta}_2 \preceq_k \boldsymbol{\beta}_1$. For example, $(1,2) =_0 (1,2,3)$, $(1,2) =_1 (1,2,3)$, $(1,2) =_2 (1,2,3)$, and $(1,2) \prec_3 (1,2,3)$. Note that indices $\boldsymbol{\beta}$ combined with the order $\preceq_k$ can be used to witness a winning strategy of a parity game through a proper assignment of them (a *parity progress measure* [6]) to positions of the game. The function *Regress* is defined as follows.

**Definition 6 (Function *Regress*).** *First, we define $\Gamma_\mu^{\boldsymbol{\beta}}$ and $\Gamma_\nu^{\boldsymbol{\beta}}$ for a type environment $\Gamma$ for $\mathcal{E}^{(m)}$ and an index $\boldsymbol{\beta}$ of length $j$ by:*

$$\Gamma_\mu^{\boldsymbol{\beta}} = \{F_{j'} : \tau \mid \exists F_{j'}^{\boldsymbol{\beta}'} : \tau \in \Gamma. \, \boldsymbol{\beta}' \prec_j \boldsymbol{\beta}\}$$
$$\Gamma_\nu^{\boldsymbol{\beta}} = \{F_{j'} : \tau \mid \exists F_{j'}^{\boldsymbol{\beta}'} : \tau \in \Gamma. \, \boldsymbol{\beta}' \preceq_{j-1} \boldsymbol{\beta}\},$$

*that is, $\Gamma_\alpha^{\boldsymbol{\beta}}$ is a type environment for the original HES $\mathcal{E}$ consisting of all type bindings in $\Gamma$ with indices "smaller" than $\boldsymbol{\beta}$ (the meaning of "smaller" depends on the fixpoint operator $\alpha$). Using this $\Gamma_\alpha^{\boldsymbol{\beta}}$, we define Regress as a monotonic function on type environments for $\mathcal{E}^{(m)}$ by:*

$$Regress(\Gamma) = \{F_j^{\boldsymbol{\beta}} : \tau \in \Gamma \mid \Gamma_{\alpha_j}^{\boldsymbol{\beta}} \vdash_{\mathcal{L}} \varphi_j : \tau\},$$

that is, $Regress(\Gamma)$ consists of all $F_j^{\boldsymbol{\beta}}\!:\!\tau \in \Gamma$ such that the right-hand side formula $\varphi_j$ of $F_j$ in the original HES $\mathcal{E}$ has type $\tau$ under the type environment $\Gamma_{\alpha_j}^{\boldsymbol{\beta}}$.

Note that $Regress$ is a monotonic function on a finite domain. We write $Regress^\omega(\Gamma)$ for $\bigcap_{i\in\omega} Regress^i(\Gamma)$, which is the greatest $\Gamma'$ such that $\Gamma' \subseteq \Gamma$ and $Regress(\Gamma') = \Gamma'$. Lemma 3 follows immediately from the following two lemmas (Lemmas 5 and 6).

**Lemma 5.** *If $\Gamma \subseteq (\mathcal{F}_{\mathcal{L},\mathcal{E}^{(m)}})^\omega(\emptyset)$ is a fixpoint of Regress, then $Forget(\Gamma)$ is a subset of the winning region of player 0 for $\mathbf{SG}(\mathcal{L},\mathcal{E}, Forget((\mathcal{F}_{\mathcal{L},\mathcal{E}^{(m)}})^\omega(\emptyset)))$.*

This is intuitively because, for each $F_j\!:\!\tau \in Forget(\Gamma)$, we can find $F_j^{\boldsymbol{\beta}}\!:\!\tau \in \Gamma$ such that choosing $\Gamma_{\alpha_j}^{\boldsymbol{\beta}}$ at $F_j\!:\!\tau$ gives a winning strategy for player 0; see Appendix B for details. Now it remains to show:

**Lemma 6.** *If the typability game $\mathbf{TG}(\mathcal{L},\mathcal{E})$ is winning, then for sufficiently large $m$, $F_1 : q_0 \in Forget(Regress^\omega((\mathcal{F}_{\mathcal{L},\mathcal{E}^{(m)}})^\omega(\emptyset)))$.*

We prepare a few further definitions and lemmas (see Appendix B for proofs). Let $\Gamma_\omega^{(m)}$ be $(\mathcal{F}_{\mathcal{L},\mathcal{E}^{(m)}})^\omega(\emptyset)$ and $\Gamma_\omega'^{(m)}$ be $Regress^\omega(\Gamma_\omega^{(m)})$. For each $k = 1, 2, \ldots,$ we define $D_k$ as the set of type bindings removed by the $k$-th application of $Regress$ to $\Gamma_\omega^{(m)}$, that is, $D_k = Regress^{k-1}(\Gamma_\omega^{(m)}) \setminus Regress^k(\Gamma_\omega^{(m)})$.

**Lemma 7.** *If $F_j^{\boldsymbol{\beta}} : \tau \in D_k$ and $\beta_j = 0$, then $\alpha_j = \nu$.*

**Lemma 8.** *If $F_j^{\boldsymbol{\beta}}\!:\!\tau \in D_k$ and $\beta_j \neq 0$, then there exists $k'$ satisfying $1 \le k' < k$ such that $F_{j'}^{\boldsymbol{\beta}(j')} : \tau' \in D_{k'}$ holds for some $j'$ and $\tau'$.*

We are now ready to prove Lemma 6.

*Proof of Lemma 6.* We show the lemma by contradiction. Since the typability game $\mathbf{TG}(\mathcal{L},\mathcal{E})$ is winning, we have $F_1^{(m)}\!:\!q_0 \in \Gamma_\omega^{(m)}$ for sufficiently large $m$ (this is intuitively because $\mathbf{TG}(\mathcal{L},\mathcal{E}^{(m)})$ is also winning; see Appendix B, Lemma 13 for a formal proof). Suppose it were the case that $F_1^{(m)} : q_0 \notin \Gamma_\omega'^{(m)}$. Then there must be a positive integer $k$ such that $F_1^{(m)} : q_0 \in D_k$. Therefore, by Lemma 8, there exists a sequence of type bindings $F_1^{(m)}\!:\!q_0 = F_{j_0}^{\boldsymbol{\beta}_0}\!:\!\tau_0$, $F_{j_1}^{\boldsymbol{\beta}_1}\!:\!\tau_1$, $\ldots$, $F_{j_\ell}^{\boldsymbol{\beta}_\ell}\!:\!\tau_\ell$ such that (i) $\boldsymbol{\beta}_\ell$ ends with 0, and (ii) $\boldsymbol{\beta}_i = \boldsymbol{\beta}_{i-1}(j_i)$ and $F_{j_i}^{\boldsymbol{\beta}_i} : \tau_i \in D_{k_i}$ hold for each $i \in \{1, \ldots, \ell\}$, where $k = k_0 > k_1 > \cdots > k_\ell$. Moreover, we have $\alpha_{j_\ell} = \nu$ by Lemma 7. Let $\boldsymbol{\beta}_\ell = (\beta_1, \ldots, \beta_{j_\ell-1}, 0)$. Then, $(\beta_1, \ldots, \beta_{j_\ell-1}, m)$, $(\beta_1, \ldots, \beta_{j_\ell-1}, m-1)$, $\ldots$, and $(\beta_1, \ldots, \beta_{j_\ell-1}, 1)$ must exist in the sequence $\boldsymbol{\beta}_0, \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_{\ell-1}$ in this order (see Appendix B, Lemma 11).

For each $i \in \{0, \ldots, m\}$, let $\ell_i$ be the integer with $\boldsymbol{\beta}_{\ell_i} = (\beta_1, \ldots, \beta_{j_\ell-1}, i)$. Since the number of intersection types $\tau'$ satisfying $\tau' :: \eta_{j_\ell}$ is finite, there must exist duplicate types in the sequence $\tau_{\ell_0}, \tau_{\ell_1}, \ldots, \tau_{\ell_m}$ for sufficiently large $m$. Let $\tau_{\ell_a}$ and $\tau_{\ell_b}$ be such a pair with $\ell_a < \ell_b$. Then, we have $F_{j_\ell}^{\boldsymbol{\beta}_{\ell_a}} : \tau_{\ell_a} \in D_{k_{\ell_a}}$ and $F_{j_\ell}^{\boldsymbol{\beta}_{\ell_b}} : \tau_{\ell_b} \in D_{k_{\ell_b}}$. However, since $\alpha_{j_\ell} = \nu$ and $\boldsymbol{\beta}_{\ell_a} =_{j_\ell-1} \boldsymbol{\beta}_{\ell_b}$, we have

$\Gamma_{\alpha_{j_\ell}}^{\boldsymbol{\beta}_{\ell_a}} = \Gamma_{\alpha_{j_\ell}}^{\boldsymbol{\beta}_{\ell_b}}$ for any $\Gamma$. Therefore, by the definition of the function *Regress* and the assumption $\tau_{\ell_a} = \tau_{\ell_b}$, the type bindings $F_{j_\ell}^{\boldsymbol{\beta}_{\ell_a}} : \tau_{\ell_a}$ and $F_{j_\ell}^{\boldsymbol{\beta}_{\ell_b}} : \tau_{\ell_b}$ must be removed by *Regress* at the same time. This contradicts the assumption $\ell_a < \ell_b$. Therefore, $F_1^{(m)} : q_0 \in \Gamma_\omega'^{(m)}$ holds for sufficiently large $m$. □

## 5   Implementation and Experiments

We have implemented a prototype HFL model checker HOMUSAT[8] based on the algorithm discussed in Section 3. As mentioned in footnotes in Section 3, some optimization techniques are used to improve the performance of HOMUSAT. See Appendix C for a brief explanation on several of these optimizations.

We have carried out experiments to evaluate the efficiency of HOMUSAT. As benchmark problems, we used HORS model checking problems used as benchmarks for HORS model checkers TRAVMC2 [14], HORSATP [18], and HOR-SAT2 [8]. These benchmarks include many typical instances of higher-order model checking, such as the ones obtained from program verification problems. They were converted to HFL model checking problems via the translation by Kobayashi et al. [9]. The resulting set of benchmarks consists of 136 problems of orders up to 8. Whereas the LTS size is moderate (around 10) for most of the instances, there are several instances with large state sets (including those with $|Q| > 100$). HES sizes vary from less than 100 to around 10,000; note that in applications to higher-order program verification [10,20], the size of an HES corresponds to the size of a program to be verified. As to the number of alternations between $\mu$ and $\nu$ within the HES, over half of the instances (83 out of 136) have no alternation (that is, they are $\mu$-only or $\nu$-only), but there are a certain number of instances that have one or more alternations, up to a maximum of 4. See Appendix D for the distributions of the orders, the LTS sizes, and the numbers of alternations between $\mu$ and $\nu$ in the benchmark set. The experiments were conducted on a machine with 2.3 GHz Intel Core i5 processor and 8 GB memory. As a reference, we have compared the result with HORSATP, one of the state-of-the-art HORS model checkers,[9] run for the original problems.

The results are shown in Figures 4 and 5. Figure 4 compares the running times of HOMUSAT with those of HORSATP. As the figure shows, HOMUSAT often outperforms HORSATP. Although it is not that this result indicates the proposed algorithm is superior as a higher-order model checking algorithm to HORSATP (the two model checkers differ in the degree of optimization),[10] the fact that HOMUSAT works fast for various problems obtained via the mechanical

---

[8]  The source code and the benchmark problems used in the experiments are available at https://github.com/hopv/homusat.

[9]  For the restricted class of properties expressed by trivial automata, HORSAT2 is the state-of-the-art.

[10]  Actually, as the two algorithms are both based on type-based saturation algorithm [4], various type-oriented optimization techniques used in HOMUSAT can also be adapted to HORSATP and are expected to improve its performance.
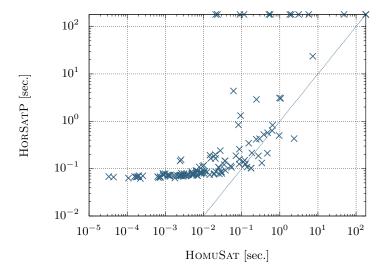
**Fig. 4.** The experimental results: comparison with HorSatP (timeout = 180 sec.)

conversion from HORS to HFL, which increases the size of inputs and thus makes them harder to solve, is promising. Evaluation of the efficiency of the proposed algorithm against a set of problems obtained directly as HFL model checking problems is left for future work.

Figure 5 shows the distribution of the running times of HomuSat with respect to the input HES size. As the figure shows, despite the $k$-EXPTIME worst-case complexity, the actual running times do not grow so rapidly. This is partially explained by the fact that the time complexity of HFL model checking is fixed-parameter polynomial in the size of HES [9].

## 6   Related Work

The logic HFL has been introduced by Viswanathan and Viswanathan [19]. Later, Lange and his colleagues studied its various theoretical properties [1,2,11]. In particular, they have shown that HFL model checking is $k$-EXPTIME complete for order $k$ HFL formulas. There has been, however, no practical HFL model checker. Lozes has implemented a prototype HFL model checker, but it is restricted to order-1 HFL, and scales only for LTS of size up to 10 or so [12].

Our algorithm is based on the type-based characterization of HFL model checking [9], and type-based saturation algorithms for HORS model checking [4,18]. In particular, the idea of restricting the arena of the typability game follows that of Suzuki et al. [18]. The detail of the algorithms are however different; in particular, the initial type environment in Suzuki et al. [18] contains $F : \top \to \cdots \to \top \to q$ for any recursive function $F$, whereas in our algorithm,
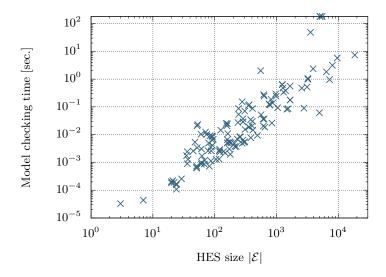
**Fig. 5.** HES size $|\mathcal{E}|$ versus time required for model checking (timeout $= 180$ sec.)

$\Gamma_0$ contains $F : \top \to \cdots \to \top \to q$ only for fixpoint variables bound by $\nu$. The use of a smaller initial type environment may be one of the reasons why our model checker tends to outperform theirs even for HORS model checking problems. Another difference is in the correctness proofs. In our opinion, our proof is significantly simpler and streamlined than theirs. Their proof manipulates infinite derivation trees. In contrast, our proof is a natural generalization of the correctness proof for the restricted fragment of HORS model checking (which corresponds to the $\mu$-only or $\nu$-only fragment of HFL model checking) [4], using the standard concept of parity progress measures.

## 7    Conclusion

We have proposed the first practical algorithm for HFL model checking, and proved its correctness. We have confirmed through experiments that, despite the huge worst-case complexity, our prototype HFL model checker runs fast for typical instances of higher-order model checking.

## References

1. Axelsson, R., Lange, M.: Model checking the first-order fragment of higher-order fixpoint logic. In: Proceedings of the 14th International Conference on Logic

for Programming, Artificial Intelligence, and Reasoning (LPAR 2007). LNCS, vol. 4790, pp. 62–76. Springer (2007). https://doi.org/10.1007/978-3-540-75560-9_7

2. Axelsson, R., Lange, M., Somla, R.: The complexity of model checking higher-order fixpoint logic. Logical Methods in Computer Science **3**(2:7), 1–33 (2007). https://doi.org/10.2168/LMCS-3(2:7)2007

3. Barendregt, H., Statman, R., Dekkers, W.: Lambda Calculus with Types. Cambridge University Press (2013). https://doi.org/10.1017/CBO9781139032636

4. Broadbent, C., Kobayashi, N.: Saturation-based model checking of higher-order recursion schemes. In: Proceedings of the 22nd EACSL Annual Conference on Computer Science Logic (CSL 2013). LIPIcs, vol. 23, pp. 129–148. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2013). https://doi.org/10.4230/LIPIcs.CSL.2013.129

5. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games: A Guide to Current Research, LNCS, vol. 2500. Springer (2002). https://doi.org/10.1007/3-540-36387-4

6. Jurdziński, M.: Small progress measures for solving parity games. In: Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2000). LNCS, vol. 1770, pp. 290–301. Springer (2000). https://doi.org/10.1007/3-540-46541-3_24

7. Kobayashi, N.: Model-checking higher-order functions. In: Proceedings of the 11th ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP 2009). pp. 25–36. ACM (2009). https://doi.org/10.1145/1599410.1599415

8. Kobayashi, N.: HorSat2: A model checker for HORS based on SATuration. https://github.com/hopv/horsat2 (2015)

9. Kobayashi, N., Lozes, E., Bruse, F.: On the relationship between higher-order recursion schemes and higher-order fixpoint logic. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017). pp. 246–259. ACM (2017). https://doi.org/10.1145/3009837.3009854

10. Kobayashi, N., Tsukada, T., Watanabe, K.: Higher-order program verification via HFL model checking. In: Proceedings of the 27th European Symposium on Programming (ESOP 2018). LNCS, vol. 10801, pp. 711–738. Springer (2018). https://doi.org/10.1007/978-3-319-89884-1_25

11. Lange, M., Lozes, É., Guzmán, M.V.: Model-checking process equivalences. Theoretical Computer Science **560**, 326–347 (2014). https://doi.org/10.1016/j.tcs.2014.08.020

12. Lozes, É.: Personal communication

13. Lozes, É.: A type-directed negation elimination. In: Proceedings of the 10th International Workshop on Fixed Points in Computer Science (FICS 2015). EPTCS, vol. 191, pp. 132–142 (2015). https://doi.org/10.4204/EPTCS.191.12

14. Neatherway, R.P., Ong, C.H.L.: TravMC2: Higher-order model checking for alternating parity tree automata. In: Proceedings of the 2014 International SPIN Symposium on Model Checking of Software (SPIN 2014). pp. 129–132. ACM (2014). https://doi.org/10.1145/2632362.2632381

15. Ong, C.H.L.: On model-checking trees generated by higher-order recursion schemes. In: Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science (LICS 2006). pp. 81–90. IEEE Computer Society (2006). https://doi.org/10.1109/LICS.2006.38

16. Ramsay, S.J., Neatherway, R.P., Ong, C.H.L.: A type-directed abstraction refinement approach to higher-order model checking. In: Proceedings of the 41st ACM

SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2014). pp. 61–72. ACM (2014). https://doi.org/10.1145/2535838.2535873

17. Shivers, O.: Control-Flow Analysis of Higher-Order Languages. Ph.D. thesis, Carnegie-Mellon University (1991)

18. Suzuki, R., Fujima, K., Kobayashi, N., Tsukada, T.: Streett automata model checking of higher-order recursion schemes. In: Proceedings of the 2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017). LIPIcs, vol. 84, pp. 32:1–32:18. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2017). https://doi.org/10.4230/LIPIcs.FSCD.2017.32

19. Viswanathan, M., Viswanathan, R.: A higher order modal fixed point logic. In: Proceedings of the 15th International Conference on Concurrency Theory (CONCUR 2004). LNCS, vol. 3170, pp. 512–528. Springer (2004). https://doi.org/10.1007/978-3-540-28644-8_33

20. Watanabe, K., Tsukada, T., Oshikawa, H., Kobayashi, N.: Reduction from branching-time property verification of higher-order programs to HFL validity checking. In: Proceedings of the 2019 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation (PEPM 2019). pp. 22–34. ACM (2019). https://doi.org/10.1145/3294032.3294077

# Appendix

## A    Proofs for Section 3

We only prove Lemma 1, since the other lemma (Lemma 2) is not used in the proofs in Section 4; it was introduced just to explain the intuitions behind the algorithm.

First, we prove that typing is closed under the inverse of substitutions.

**Lemma 9.** *Let $\varphi$ and $\chi$ be HFL formulas containing no fixpoint operators or $\lambda$-abstractions. If $\Gamma \vdash_{\mathcal{L}} [\chi/X]\,\varphi : \tau$, then there exists a type environment $\Delta$ such that $dom(\Delta) \subseteq FV(\varphi) \cap \{X\}$, $\Gamma \cup \Delta \vdash_{\mathcal{L}} \varphi : \tau$, and $\forall X : \tau' \in \Delta.\, \Gamma \vdash_{\mathcal{L}} \chi : \tau'$.*

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

- If $\varphi = \mathtt{true}/\mathtt{false}$, then $\Delta = \emptyset$ satisfies the condition.
- If $\varphi = Y$ and $Y \neq X$, then $\Delta = \emptyset$ satisfies the condition.
- If $\varphi = X$, then $\Delta = \{X : \tau\}$ satisfies the condition.
- If $\varphi = \varphi_1 \vee \varphi_2$, then $\tau = q \in Q$, and we have either $\Gamma \vdash_{\mathcal{L}} [\chi/X]\,\varphi_1 : q$ or $\Gamma \vdash_{\mathcal{L}} [\chi/X]\,\varphi_2 : q$. Suppose that $\Gamma \vdash_{\mathcal{L}} [\chi/X]\,\varphi_i : q$. By the induction hypothesis, there exists $\Delta_i$ such that $dom(\Delta_i) \subseteq FV(\varphi_i) \cap \{X\}$, $\Gamma \cup \Delta_i \vdash_{\mathcal{L}} \varphi_i : q$, and $\forall X : \tau' \in \Delta_i.\, \Gamma \vdash_{\mathcal{L}} \chi : \tau'$ hold. It is not difficult to see that this $\Delta_i$ satisfies the condition for $\Delta$.
- If $\varphi = \varphi_1 \wedge \varphi_2$, then $\tau = q \in Q$, and we have both $\Gamma \vdash_{\mathcal{L}} [\chi/X]\,\varphi_1 : q$ and $\Gamma \vdash_{\mathcal{L}} [\chi/X]\,\varphi_2 : q$. Therefore, by the induction hypothesis, there exists $\Delta_i$ for each $i = 1, 2$ such that $dom(\Delta_i) \subseteq FV(\varphi_i) \cap \{X\}$, $\Gamma \cup \Delta_i \vdash_{\mathcal{L}} \varphi_i : q$, and $\forall X : \tau' \in \Delta_i.\, \Gamma \vdash_{\mathcal{L}} \chi : \tau'$ hold. Hence, $\Delta = \Delta_1 \cup \Delta_2$ satisfies the condition.

- If $\varphi = \langle a \rangle \varphi'$, then $\tau = q \in Q$, and there exists $q' \in Q$ such that $q \xrightarrow{a} q'$ and $\Gamma \vdash_{\mathcal{L}} [\chi/X] \varphi' : q'$. Therefore, by the induction hypothesis, there exists $\Delta'$ such that $dom(\Delta') \subseteq FV(\varphi') \cap \{X\}$, $\Gamma \cup \Delta' \vdash_{\mathcal{L}} \varphi' : q'$, and $\forall X : \tau' \in \Delta'. \Gamma \vdash_{\mathcal{L}} \chi : \tau'$ hold. Hence, $\Delta = \Delta'$ satisfies the condition.
- If $\varphi = [a] \varphi'$, then $\tau = q \in Q$, and for each $q' \in Q$ with $q \xrightarrow{a} q'$, we have $\Gamma \vdash_{\mathcal{L}} [\chi/X] \varphi' : q'$. Let $Q' = \{q' \in Q \mid q \xrightarrow{a} q'\} = \bigcup_{i \in I} \{q_i\}$. By the induction hypothesis, there is $\Delta_i$ for each $i \in I$ such that $dom(\Delta_i) \subseteq FV(\varphi') \cap \{X\}$, $\Gamma \cup \Delta_i \vdash_{\mathcal{L}} \varphi' : q_i$, and $\forall X : \tau' \in \Delta_i. \Gamma \vdash_{\mathcal{L}} \chi : \tau'$ hold. Therefore, $\Delta = \bigcup_{i \in I} \Delta_i$ satisfies the condition.
- If $\varphi = \varphi_1 \varphi_2$, then there is an intersection type $\sigma = \{\tau_1, \ldots, \tau_k\}$ satisfying $\Gamma \vdash_{\mathcal{L}} [\chi/X] \varphi_1 : \sigma \to \tau$ and $\Gamma \vdash_{\mathcal{L}} [\chi/X] \varphi_2 : \tau_i$ for each $i \in \{1, \ldots, k\}$. Therefore, by the induction hypothesis, there exists $\Delta_0$ such that $dom(\Delta_0) \subseteq FV(\varphi_1) \cap \{X\}$, $\Gamma \cup \Delta_0 \vdash_{\mathcal{L}} \varphi_1 : \sigma \to \tau$, and $\forall X : \tau' \in \Delta_0. \Gamma \vdash_{\mathcal{L}} \chi : \tau'$. Moreover, for each $i \in \{1, \ldots, k\}$, there exists $\Delta_i$ such that $dom(\Delta_i) \subseteq FV(\varphi_2) \cap \{X\}$, $\Gamma \cup \Delta_i \vdash_{\mathcal{L}} \varphi_2 : \tau_i$, and $\forall X : \tau' \in \Delta_i. \Gamma \vdash_{\mathcal{L}} \chi : \tau'$. Therefore, $\Delta = \bigcup_{i=0}^{k} \Delta_i$ satisfies the condition.

$\square$

Now we are ready to prove Lemma 1.

*Proof of Lemma 1.* The proof is by induction on the derivation of $\varphi \longrightarrow_{\mathcal{E}} \varphi'$. For the base case, suppose that $\varphi = F_j \chi_1 \cdots \chi_\ell$ and $\varphi' = [\chi_1/X_1, \ldots, \chi_\ell/X_\ell] \psi_j$. By repeatedly applying Lemma 9, we can construct a type environment $\Delta$ such that $dom(\Delta) \subseteq FV(\psi_j) \cap \{X_1, \ldots, X_\ell\}$, $\Gamma \cup \Delta \vdash_{\mathcal{L}} \psi_j : q$, and $\forall X_k : \tau \in \Delta. \Gamma \vdash_{\mathcal{L}} \chi_k : \tau$. Let $\sigma_k = \{\tau \mid X_k : \tau \in \Delta\}$. Since $\chi_k \in Flow_{\mathcal{E}}(X_k)$ holds for every $k \in \{1, \ldots, \ell\}$, we have $F_j : \sigma_1 \to \cdots \to \sigma_\ell \to q \in \mathcal{F}_{\mathcal{L}, \mathcal{E}}(\Gamma)$ by the definition of the function $\mathcal{F}_{\mathcal{L}, \mathcal{E}}$. Therefore, by repeatedly applying the typing rule (T-App), we have $\mathcal{F}_{\mathcal{L}, \mathcal{E}}(\Gamma) \vdash_{\mathcal{L}} \varphi : q$. The induction steps are trivial. $\square$

# B   Proofs for Section 4

First, we see some properties of the truncated lexicographic order $\preceq_k$ on indices.

**Lemma 10.**

(i). *If $i \leq j$, then $\boldsymbol{\beta} \prec_i \boldsymbol{\beta}'$ implies that $\boldsymbol{\beta} \prec_j \boldsymbol{\beta}'$.*
(ii). *If $i \leq j$, then $\boldsymbol{\beta} \preceq_j \boldsymbol{\beta}'$ implies that $\boldsymbol{\beta} \preceq_i \boldsymbol{\beta}'$.*
(iii). *$\boldsymbol{\beta}(\ell) \prec_j \boldsymbol{\beta}$ and $\boldsymbol{\beta}(\ell) \preceq_{j-1} \boldsymbol{\beta}$ hold for each $\boldsymbol{\beta}$ of length $j$.*
(iv). *For a set of $\boldsymbol{\beta}$'s of length at most $n$, $\preceq_n$ gives a total order.*

*Proof.* Trivial. $\square$

The next lemma implies that, if we start from the initial index $\boldsymbol{\beta}_0 = (m)$ and inductively define $\boldsymbol{\beta}_i = \boldsymbol{\beta}_{i-1}(\ell_i)$ by a sequence of positive integers $\ell_1, \ell_2, \ldots$ until it reaches a final index $\boldsymbol{\beta}_k = (\beta_1, \ldots, \beta_{j-1}, 0)$, then the sequence $\boldsymbol{\beta}_0, \ldots, \boldsymbol{\beta}_k$ contains all $(\beta_1, \ldots, \beta_{j-1}, i)$'s for $i \in \{0, \ldots, m\}$.

**Lemma 11.** *Let $\boldsymbol{\beta}_0, \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_k$ be a sequence of indices such that, for each $i \in \{1, \ldots, k\}$, $\boldsymbol{\beta}_i = \boldsymbol{\beta}_{i-1}(\ell_i)$ holds for some $\ell_i$. If $\boldsymbol{\beta}_k = (\beta_1, \ldots, \beta_{j-1}, \beta_j)$, $\beta_j < m$, and $(\beta_1, \ldots, \beta_{j-1}, \beta_j + 1) \preceq_j \boldsymbol{\beta}_0$, then there exists $i \in \{0, \ldots, k-1\}$ such that $\boldsymbol{\beta}_i = (\beta_1, \ldots, \beta_{j-1}, \beta_j + 1)$.*

*Proof.* The proof proceeds by induction on the number of successors $k$. For the base case $k = 1$, suppose that $\boldsymbol{\beta}_0(\ell) = (\beta_1, \ldots, \beta_{j-1}, \beta_j)$. By the definition of the operation $\boldsymbol{\beta}(\ell)$ and the assumption $\beta_j < m$, there are only two possibilities: (i) $\boldsymbol{\beta}_0 = (\beta_1, \ldots, \beta_{j-1}, \beta_j + 1)$ and $\ell = j$, or (ii) $\boldsymbol{\beta}_0 = (\beta_1, \ldots, \beta_{j-1}, \beta_j, \ldots, \beta_{j'})$ and $\ell = j < j'$. Since we also have $(\beta_1, \ldots, \beta_{j-1}, \beta_j + 1) \preceq_j \boldsymbol{\beta}_0$, (i) must be the case, and thus $\boldsymbol{\beta}_0 = (\beta_1, \ldots, \beta_{j-1}, \beta_j + 1)$ holds.

For the step case, if $(\beta_1, \ldots, \beta_{j-1}, \beta_j + 1) \preceq_j \boldsymbol{\beta}_1$, then we are done by the induction hypothesis. Suppose otherwise, and let $\ell_0$ and $\ell_1$ be the lengths of $\boldsymbol{\beta}_0$ and $\boldsymbol{\beta}_1$, respectively. Then, since $\boldsymbol{\beta}_k = (\beta_1, \ldots, \beta_{j-1}, \beta_j) \preceq_j \boldsymbol{\beta}_1 \prec_j (\beta_1, \ldots, \beta_{j-1}, \beta_j + 1)$, we have $\boldsymbol{\beta}_1 =_j (\beta_1, \ldots, \beta_{j-1}, \beta_j)$, and thus $j \leq \ell_1$ holds. If $\ell_0 < j$, then we have $\beta_j = m$, and thus it contradicts the assumption $\beta_j < m$. Moreover, if $j < \ell_0$, then $\boldsymbol{\beta}_0 =_j \boldsymbol{\beta}_1$ must hold since we also have $j \leq \ell_1$. This contradicts the assumption $\boldsymbol{\beta}_1 \prec_j (\beta_1, \ldots, \beta_{j-1}, \beta_j + 1) \preceq_j \boldsymbol{\beta}_0$, and thus we have $\ell_0 = j$. Hence, it follows that $\boldsymbol{\beta}_0 = (\beta_1, \ldots, \beta_{j-1}, \beta_j + 1)$ from $\boldsymbol{\beta}_1 =_j (\beta_1, \ldots, \beta_{j-1}, \beta_j)$. $\qquad\square$

### B.1  Proof of Lemma 5

The following is a key lemma, which can be seen as defining a *parity progress measure* [6] for typability games.

**Lemma 12 (Parity Progress Measure for Typability Games).** *First, let $\mathbf{TG}(\mathcal{L}, \mathcal{E}) = (V_0, V_1, v_0, E, \Omega)$ be a typability game. Let $W_0$ be a subset of $V_0$ and $\varsigma : W_0 \to V_1$ be a strategy on $W_0$ such that $\varsigma(F_j : \tau) \subseteq W_0$ holds for every $F_j : \tau \in W_0$. If we can assign to each $F_j : \tau \in W_0$ an index $\xi(F_j : \tau)$ satisfying the following conditions:*

$$(1). \ \forall F_{j'} : \tau' \in \varsigma(F_j : \tau). \, \xi(F_{j'} : \tau') \prec_j \xi(F_j : \tau) \quad (\text{if } \alpha_j = \mu)$$
$$(2). \ \forall F_{j'} : \tau' \in \varsigma(F_j : \tau). \, \xi(F_{j'} : \tau') \preceq_{j-1} \xi(F_j : \tau) \ (\text{if } \alpha_j = \nu),$$

*then $\varsigma$ is a winning strategy of player 0 for each $F_j : \tau \in W_0$ in the typability game $\mathbf{TG}(\mathcal{L}, \mathcal{E})$, and also of the subgame $\mathbf{SG}(\mathcal{L}, \mathcal{E}, W_0)$.*

*Proof.* First, note that every finite maximal play starting from $v \in W_0$ and conforming with $\varsigma$ is winning for player 0 since $\varsigma$ is total. Moreover, since $\Omega(\Gamma) = 0$ for each $\Gamma \in V_1$, whether an infinite play satisfies the parity condition can be checked just by looking at the priorities of the positions in $V_0$.

Suppose that $\xi$ satisfies the conditions (1) and (2). Let $G = (W_0, E_\varsigma)$ be a directed graph such that $(v, w) \in E_\varsigma$ iff $w \in \varsigma(v)$. Since every play starting from $v \in W_0$ and conforming with $\varsigma$ corresponds to a walk in $G$ (by ignoring all positions in $V_1$), the strategy $\varsigma$ gives a winning strategy of player 0 for each $v \in W_0$ if every cycle in $G$ is even. Here we say that a cycle is *even* if the highest priority of its vertices is even, and otherwise the cycle is said to be *odd*.

Suppose that there is an odd cycle in $G$, and let $\mathcal{C}$ be such a cycle. Let $F_j : \tau$ be a vertex in $\mathcal{C}$ that has the minimum $j$ and thus the highest (odd) priority. Then, the cycle $\mathcal{C}$ can be written as $F_j : \tau = v_0 v_1 \cdots v_\ell = F_j : \tau$. Let $v_k = F_{j'} : \tau'$ be a vertex in $\mathcal{C}$ with $0 \le k < \ell$. If $\alpha_{j'} = \mu$, then $\xi(v_{k+1}) \prec_{j'} \xi(v_k)$ holds by the condition (1). Moreover, since $j \le j'$, we have $\xi(v_{k+1}) \preceq_j \xi(v_k)$ by Lemma 10 (ii). If $\alpha_{j'} = \nu$, then $\xi(v_{k+1}) \preceq_{j'-1} \xi(v_k)$ holds by the condition (2), and since $j \le j' - 1$, we have $\xi(v_{k+1}) \preceq_j \xi(v_k)$ again by Lemma 10 (ii). Therefore, $\xi(v_0) \succ_j \xi(v_1) \succeq_j \cdots \succeq_j \xi(v_\ell) = \xi(v_0)$ holds. This is a contradiction, and thus every cycle in $G$ is even. Hence, $\varsigma$ is a winning strategy of player 0 for each $v \in W_0$. Moreover, since $\varsigma(F_j : \tau) \subseteq W_0$ holds for each $F_j : \tau \in W_0$, it is also a winning strategy for the subgame $\mathbf{SG}(\mathcal{L}, \mathcal{E}, W_0)$.    □

We are now ready to prove Lemma 5.

*Proof of Lemma 5.* Let $W = \mathit{Forget}(\Gamma)$. To each $F_j : \tau \in W$, we assign an index $\xi(F_j : \tau)$ by $\xi(F_j : \tau) = \min_{\preceq_n}\{\boldsymbol{\beta} \mid F_j^{\boldsymbol{\beta}} : \tau \in \Gamma\}$. By the definition of the function *Regress*, for each $F_j : \tau \in W$, there exists a type environment $\Gamma' \subseteq \Gamma_{\alpha_j}^{\xi(F_j : \tau)}$ such that $\Gamma' \vdash_{\mathcal{L}} \varphi_j : \tau$ holds. We define $\varsigma(F_j : \tau)$ as such $\Gamma'$. Then, the function $\varsigma$ becomes a strategy on $W$ satisfying $\varsigma(v) \subseteq W$ for every $v \in W$.

Let $F_{j'} : \tau'$ be a type binding in $\varsigma(F_j : \tau)$. If $\alpha_j = \mu$, then there exists $F_{j'}^{\boldsymbol{\beta}'} : \tau' \in \Gamma$ satisfying $\boldsymbol{\beta}' \prec_j \xi(F_j : \tau)$ since $\varsigma(F_j : \tau) \subseteq \Gamma_{\alpha_j}^{\xi(F_j : \tau)}$ holds. Moreover, we have $\xi(F_{j'} : \tau') \preceq_n \boldsymbol{\beta}'$ by the definition of $\xi$, and thus $\xi(F_{j'} : \tau') \preceq_j \boldsymbol{\beta}'$ by Lemma 10 (ii). Hence $\xi(F_{j'} : \tau') \prec_j \xi(F_j : \tau)$ holds. Similarly, if $\alpha_j = \nu$, then we have $\xi(F_{j'} : \tau') \preceq_{j-1} \xi(F_j : \tau)$. Therefore, $W$ consists of only winning positions of $\mathbf{SG}(\mathcal{L}, \mathcal{E}, W)$ by Lemma 12, hence also of $\mathbf{SG}(\mathcal{L}, \mathcal{E}, \mathit{Forget}((\mathcal{F}_{\mathcal{L}, \mathcal{E}^{(m)}})^\omega(\emptyset)))$.    □

## B.2    Proofs for Lemma 6

Next, we prove the lemmas that are required to prove Lemma 6.

**Lemma 13.** *If* $\mathcal{L} \models \mathcal{E}$*, then* $F_1^{(m)} : q_0 \in \Gamma_\omega^{(m)}$ *holds for sufficiently large* $m$*.*

*Proof.* For sufficiently large $m$, $\mathcal{L} \models \mathcal{E}$ if and only if $\mathcal{L} \models \mathcal{E}^{(m)}$. Pick such $m$. Let $\varphi^{(m)}$ be a formula such that $F_1^{(m)} \longrightarrow_{\mathcal{E}^{(m)}}^* \varphi^{(m)} \not\longrightarrow_{\mathcal{E}^{(m)}}$. Since $\mathcal{L} \models \mathcal{E}^{(m)}$ and the reduction preserves the semantics, we have $\emptyset \vdash_{\mathcal{L}} \varphi^{(m)} : q_0$. Hence, we obtain $\Gamma_\omega^{(m)} \vdash_{\mathcal{L}} F_1^{(m)} : q_0$ and thus $F_1^{(m)} : q_0 \in \Gamma_\omega^{(m)}$ by repeatedly applying Lemma 1.    □

Given an HES $\mathcal{E} = (F_1 : \eta_1 =_{\alpha_1} \varphi_1; \cdots; F_n : \eta_n =_{\alpha_n} \varphi_n)$ and a type environment $\Gamma$ with $\mathit{dom}(\Gamma) \subseteq \{F_1, \ldots, F_n\}$, we write $\vdash_{\mathcal{L}} \mathcal{E} : \Gamma$ when $\Gamma \vdash_{\mathcal{L}} \varphi_j : \tau$ holds for every $F_j : \tau \in \Gamma$.

**Lemma 14.** *If* $\vdash_{\mathcal{L}} \mathcal{E} : \Gamma$*, then* $\vdash_{\mathcal{L}} \mathcal{E} : \mathcal{F}_{\mathcal{L}, \mathcal{E}}(\Gamma)$*.*

*Proof.* Suppose that $\vdash_{\mathcal{L}} \mathcal{E} : \Gamma$. Let $F_j : \tau$ be a type binding with $F_j : \tau \in \mathcal{F}_{\mathcal{L}, \mathcal{E}}(\Gamma)$, where $\tau = \sigma_1 \to \cdots \to \sigma_\ell \to q$ and $\varphi_j = \lambda X_1. \cdots \lambda X_\ell. \psi_j$. If $F_j : \tau \in \Gamma$, then, we have $\Gamma \vdash_{\mathcal{L}} \varphi_j : \tau$ since $\vdash_{\mathcal{L}} \mathcal{E} : \Gamma$, and thus $\mathcal{F}_{\mathcal{L}, \mathcal{E}}(\Gamma) \vdash_{\mathcal{L}} \varphi_j : \tau$ holds by $\Gamma \subseteq \mathcal{F}_{\mathcal{L}, \mathcal{E}}(\Gamma)$.

If $F_j : \tau \notin \Gamma$, then, by the definition of the function $\mathcal{F}_{\mathcal{L},\mathcal{E}}$, there exists a type environment $\Delta$ such that $dom(\Delta) \subseteq FV(\psi_j) \cap \{X_1, \ldots, X_\ell\}$, $\Gamma \cup \Delta \vdash_{\mathcal{L}} \psi_j : q$, and $\forall k \in \{1, \ldots, \ell\}.\, \sigma_k = \Delta(X_k)$. Therefore, by repeatedly applying the typing rule (T-ABS), we have $\Gamma \vdash_{\mathcal{L}} \varphi_j : \tau$, and thus $\mathcal{F}_{\mathcal{L},\mathcal{E}}(\Gamma) \vdash_{\mathcal{L}} \varphi_j : \tau$. ☐

**Lemma 15.** *Let $F_j^{\boldsymbol{\beta}} : \tau$ be a type binding in $\Gamma_\omega^{(m)}$. If $\beta_j = 0$, then we have $\emptyset \vdash_{\mathcal{L}} \varphi_j^{\boldsymbol{\beta}} : \tau$. Otherwise, we have $\Gamma_\omega^{(m)} \downarrow_{\{F_1^{\boldsymbol{\beta}(1)}, \ldots, F_n^{\boldsymbol{\beta}(n)}\}} \vdash_{\mathcal{L}} \varphi_j^{\boldsymbol{\beta}} : \tau$, where $\Gamma \downarrow_S$ denotes the restriction of $\Gamma$ by $S$, i.e., $\Gamma \downarrow_S = \{F : \tau \in \Gamma \mid F \in S\}$.*

*Proof.* Since $\vdash_{\mathcal{L}} \mathcal{E}^{(m)} : \emptyset$ is vacuously true, we have $\vdash_{\mathcal{L}} \mathcal{E}^{(m)} : \Gamma_\omega^{(m)}$ by repeatedly applying Lemma 14. Therefore, $\Gamma_\omega^{(m)} \vdash_{\mathcal{L}} \varphi_j^{\boldsymbol{\beta}} : \tau$ holds for each $F_j^{\boldsymbol{\beta}} : \tau \in \Gamma_\omega^{(m)}$. Let $F_j^{\boldsymbol{\beta}} : \tau$ be a type binding in $\Gamma_\omega^{(m)}$. Suppose that $\beta_j = 0$. Then, $\alpha_j$ must be $\nu$ because otherwise $\varphi_j^{\boldsymbol{\beta}} = \lambda \widetilde{X}.\mathtt{false}$ cannot be typed by any type environment. Therefore, $\varphi_j^{\boldsymbol{\beta}}$ is of the form $\lambda \widetilde{X}.\mathtt{true}$ and thus we have $\emptyset \vdash_{\mathcal{L}} \varphi_j : \tau$. When $\beta_j \neq 0$, we have $FV(\varphi_j^{\boldsymbol{\beta}}) \subseteq \{F_1^{\boldsymbol{\beta}(1)}, \ldots, F_n^{\boldsymbol{\beta}(n)}\}$ by the definition of the formula $\varphi_j^{\boldsymbol{\beta}}$, and thus $\Gamma_\omega^{(m)} \downarrow_{\{F_1^{\boldsymbol{\beta}(1)}, \ldots, F_n^{\boldsymbol{\beta}(n)}\}} \vdash_{\mathcal{L}} \varphi_j^{\boldsymbol{\beta}} : \tau$ always holds. ☐

We now restate and prove Lemmas 7 and 8.

**Lemma 7.** *If $F_j^{\boldsymbol{\beta}} : \tau \in D_k$ and $\beta_j = 0$, then $\alpha_j = \nu$.*

*Proof.* Let $F_j^{\boldsymbol{\beta}} : \tau$ be a type binding with $\beta_j = 0$ and $\alpha_j = \mu$. Then, $\varphi_j^{\boldsymbol{\beta}} = \lambda \widetilde{X}.\mathtt{false}$ by the definition of the formula $\varphi_j^{\boldsymbol{\beta}}$. If $F_j^{\boldsymbol{\beta}} : \tau \in \Gamma_\omega^{(m)}$, then $\Gamma_\omega^{(m)} \vdash_{\mathcal{L}} \varphi_j^{\boldsymbol{\beta}} : \tau$ holds by Lemma 15. However, since $\varphi_j^{\boldsymbol{\beta}}$ cannot be typed by any type environments, we have $\Gamma_\omega^{(m)} \nvdash_{\mathcal{L}} \varphi_j^{\boldsymbol{\beta}} : \tau$. Therefore, we have $F_j^{\boldsymbol{\beta}} : \tau \notin \Gamma_\omega^{(m)}$ and thus $F_j^{\boldsymbol{\beta}} : \tau \notin D_k$ for any $k$. ☐

**Lemma 8.** *If $F_j^{\boldsymbol{\beta}} : \tau \in D_k$ and $\beta_j \neq 0$, then there exists $k'$ satisfying $1 \leq k' < k$ such that $F_{j'}^{\boldsymbol{\beta}(j')} : \tau' \in D_{k'}$ holds for some $j'$ and $\tau'$.*

*Proof.* Let $F_j^{\boldsymbol{\beta}} : \tau$ be a type binding satisfying $F_j^{\boldsymbol{\beta}} : \tau \in D_k$ and $\beta_j \neq 0$. Since $D_k \subseteq \Gamma_\omega^{(m)}$, we have $F_j^{\boldsymbol{\beta}} : \tau \in \Gamma_\omega^{(m)}$. Therefore, by Lemma 15, there exists a type environment $\Gamma$ satisfying $\Gamma \subseteq \Gamma_\omega^{(m)} \downarrow_{\{F_1^{\boldsymbol{\beta}(1)}, \ldots, F_n^{\boldsymbol{\beta}(n)}\}}$ and $\Gamma \vdash_{\mathcal{L}} \varphi_j^{\boldsymbol{\beta}} : \tau$. For such $\Gamma$, we have $Forget(\Gamma) = \Gamma_{\alpha_j}^{\boldsymbol{\beta}}$ by Lemma 10 (iii). In addition, we have $\varphi_j^{\boldsymbol{\beta}} = [F_1^{\boldsymbol{\beta}(1)}/F_1, \ldots, F_n^{\boldsymbol{\beta}(n)}/F_n]\, \varphi_j$ since $\beta_j \neq 0$. Hence $\Gamma_{\alpha_j}^{\boldsymbol{\beta}} \vdash_{\mathcal{L}} \varphi_j : \tau$ holds. Moreover, we have $(Regress^{k-1}(\Gamma_\omega^{(m)}))_{\alpha_j}^{\boldsymbol{\beta}} \nvdash_{\mathcal{L}} \varphi_j : \tau$ since $F_j^{\boldsymbol{\beta}} : \tau \in D_k$. Therefore, $\Gamma \nsubseteq Regress^{k-1}(\Gamma_\omega^{(m)})$ holds, and thus there must exist a type binding $F_{j'}^{\boldsymbol{\beta}(j')} : \tau' \in \Gamma$ such that $F_{j'}^{\boldsymbol{\beta}(j')} : \tau' \notin Regress^{k-1}(\Gamma_\omega^{(m)})$. This implies that $F_{j'}^{\boldsymbol{\beta}(j')} : \tau' \in D_{k'}$ for some $k'$ satisfying $1 \leq k' < k$. ☐

### B.3   Proof of Lemma 4

Lemma 4 is an immediate corollary of the following lemma.

**Lemma 18.** *Let $\Gamma$ and $\Gamma'$ be type environments for $\mathcal{E}$ and $\mathcal{E}^{(m)}$, respectively. If $\Gamma_0 \subseteq \Gamma$ and $Forget(\Gamma') \subseteq \Gamma$, then $Forget(\mathcal{F}_{\mathcal{L},\mathcal{E}^{(m)}}(\Gamma')) \subseteq \mathcal{F}_{\mathcal{L},\mathcal{E}}(\Gamma)$.*

*Proof.* Let $\Gamma$ and $\Gamma'$ be type environments for $\mathcal{E}$ and $\mathcal{E}^{(m)}$, respectively, such that both $\Gamma_0 \subseteq \Gamma$ and $Forget(\Gamma') \subseteq \Gamma$ hold. Let $F_j^{\boldsymbol{\beta}} : \tau \in \mathcal{F}_{\mathcal{L},\mathcal{E}^{(m)}}(\Gamma')\backslash\Gamma'$. We show that $F_j : \tau \in \mathcal{F}_{\mathcal{L},\mathcal{E}}(\Gamma)$. Let $\tau = \sigma_1 \to \cdots \to \sigma_\ell \to q$, $\varphi_j = \lambda X_1. \cdots \lambda X_\ell.\psi_j$, and $\varphi_j^{\boldsymbol{\beta}} = \lambda X_1^{\boldsymbol{\beta}}. \cdots \lambda X_\ell^{\boldsymbol{\beta}}.\psi_j^{\boldsymbol{\beta}}$. By the definition of the function $\mathcal{F}_{\mathcal{L},\mathcal{E}^{(m)}}$, there is a type environment $\Delta'$ such that $dom(\Delta') \subseteq FV(\psi_j^{\boldsymbol{\beta}})\cap\{X_1^{\boldsymbol{\beta}},\dots,X_\ell^{\boldsymbol{\beta}}\}$, $\Gamma'\cup\Delta' \vdash_{\mathcal{L}} \psi_j^{\boldsymbol{\beta}}:q$, $\forall k \in \{1,\dots,\ell\}.\, \sigma_k = \Delta'(X_k^{\boldsymbol{\beta}})$, and $\forall X_k^{\boldsymbol{\beta}} \in dom(\Delta').\, \exists\varphi \in Flow_{\mathcal{E}^{(m)}}(X_k^{\boldsymbol{\beta}}).\, \forall\tau' \in \Delta'(X_k^{\boldsymbol{\beta}}).\, \Gamma' \vdash_{\mathcal{L}} \varphi:\tau'$.

Suppose that $\beta_j = 0$. If $\alpha_j = \mu$, then $\psi_j^{\boldsymbol{\beta}} = \texttt{false}$ and thus it cannot be typed by any type environments. Therefore, $\alpha_j$ must be $\nu$. Since $FV(\psi_j^{\boldsymbol{\beta}})$ is empty, $\Delta' = \emptyset$, and thus $F_j : \tau = F_j : \top \to \cdots \to \top \to q \in \Gamma_0 \subseteq \Gamma \subseteq \mathcal{F}_{\mathcal{L},\mathcal{E}}(\Gamma)$.

Suppose that $\beta_j \neq 0$, and let $\Delta = Forget(\Delta')$. Let us write $Forget(\varphi)$ for the formula obtained by removing all indices from $\varphi$. Then, since $Forget(\psi_j^{\boldsymbol{\beta}}) = \psi_j$ and $Forget(\Gamma' \cup \Delta') = Forget(\Gamma') \cup \Delta \subseteq \Gamma \cup \Delta$, we have $dom(\Delta) \subseteq FV(\psi_j) \cap \{X_1,\dots,X_\ell\}$ and $\Gamma \cup \Delta \vdash_{\mathcal{L}} \psi_j : q$. Moreover, since $\varphi \in Flow_{\mathcal{E}^{(m)}}(X_k^{\boldsymbol{\beta}})$ implies that $Forget(\varphi) \in Flow_{\mathcal{E}}(X_k)$, it follows that $\forall X_k \in dom(\Delta).\, \exists\varphi \in Flow_{\mathcal{E}}(X_k).\, \forall\tau' \in \Delta(X_k).\, \Gamma \vdash_{\mathcal{L}} \varphi:\tau'$. Therefore, $F_j : \tau \in \mathcal{F}_{\mathcal{L},\mathcal{E}}(\Gamma)$ holds by the definition of the function $\mathcal{F}_{\mathcal{L},\mathcal{E}}$.      □

*Proof of Lemma 4.* It suffices to show $Forget((\mathcal{F}_{\mathcal{L},\mathcal{E}^{(m)}})^i(\emptyset)) \subseteq (\mathcal{F}_{\mathcal{L},\mathcal{E}})^i(\Gamma_0)$ for any $i$. We prove it by induction on $i$. The base case where $i = 0$ is trivial. In the case where $i > 0$, we have $Forget((\mathcal{F}_{\mathcal{L},\mathcal{E}^{(m)}})^{i-1}(\emptyset)) \subseteq (\mathcal{F}_{\mathcal{L},\mathcal{E}})^{i-1}(\Gamma_0)$ by the induction hypothesis. Since $\Gamma_0 \subseteq (\mathcal{F}_{\mathcal{L},\mathcal{E}})^{i-1}(\Gamma_0)$, we have $Forget((\mathcal{F}_{\mathcal{L},\mathcal{E}^{(m)}})^i(\emptyset)) \subseteq (\mathcal{F}_{\mathcal{L},\mathcal{E}})^i(\Gamma_0)$ by Lemma 18.      □

## C   Several Optimizations

In the following, we report on several optimization techniques that are adopted in our prototype model checker HOMUSAT and thought to have improved its performance in the experiments in Section 5.

### C.1   Flow Analysis with Type Information

The function $\mathcal{F}_{\mathcal{L},\mathcal{E}}$ expands type environments in a backward direction of the rewriting sequence starting from the initial variable $F_1$. The objective of this expansion is to gather sufficient type bindings to derive the initial position $F_1 : q_0$ of the typability game, and thus what we really need is to expand type environments in a backward direction of the *derivation* of the initial position. This fact enables us to restrict the expansion using information on types.

For example, if the initial equation of the HES is $F_1 =_\alpha \langle a \rangle A \vee \langle b \rangle B$, then $\Gamma \vdash_{\mathcal{L}} \varphi_1 : q_0$ holds if and only if we have either $A : q \in \Gamma$ for some $q$ with $q_0 \xrightarrow{a} q$ or $B : q \in \Gamma$ for some $q$ with $q_0 \xrightarrow{b} q$. If $F_1$, $A$, and $B$ do not occur in the body of any fixpoint variable other than $F_1$, then we can safely exclude $F_1 : q$ with $q \neq q_0$, $A : q$ with $q_0 \xlongnotrightarrow{a} q$, and $B : q$ with $q_0 \xlongnotrightarrow{b} q$ from consideration. Although we do not describe it in detail here, we can gather such information on types during flow analysis using the idea of abstract configuration graphs [16] adapted to the context of HFL model checking.

## C.2    Restriction on the Initial Type Environment

In Algorithm 1, the initial type environment $\Gamma_0$ contains the strongest type bindings for all $\nu$-variables. However, as the completeness proof in Appendix B indicates, it is sufficient to have the strongest type bindings for the $\nu$-variables approximated as $\lambda \widetilde{X}.\widehat{\nu} = \lambda \widetilde{X}.\mathtt{true}$ in a formula $\varphi$ such that $F_1^{(m)} \longrightarrow^*_{\mathcal{E}^{(m)}} \varphi$.

Such variables can be found by analysing a *call graph* of the HES. A call graph $G = (V, E)$ of an HES $\mathcal{E}$ is a directed graph whose vertex set $V$ is the set of fixpoint variables of $\mathcal{E}$, and $(F_j, F_k) \in E$ holds if and only if the variable $F_k$ occurs in the body of the variable $F_j$. It is sufficient for the initial type environment $\Gamma_0$ to contain the strongest type bindings for the $\nu$-variables $F$ such that there is a cycle in $G$ whose highest priority is given by $F$.[11]

## C.3    Normalization through Subtyping

The objective of backward expansions by the function $F_{\mathcal{L},\mathcal{E}}$ is to construct a subgame that is winning for player 0 whenever the full typability game is winning for her. The actual implementation of the expansion process reflects this view, and it constructs not only vertices but at the same time edges of the arena of a subgame. Some edges (moves) of a typability game are comparable in terms of their "strengths" from the viewpoint of player 0, and this fact enables us to decrease the size of subgames and possibly reduce the computational cost of expansion processes.

For an illustration, let us consider the situation in which we are to derive new types for a variable $F$, whose body is $\varphi_F = \lambda X^\circ. G(G\,X)$, in the expansion process of the type environment $\Gamma = \{G : \top \to q, G : q \to q\}$. Then, we can derive $F : \sigma \to q$ for any $\sigma$ because $\Gamma$ contains a strongest type binding $G : \top \to q$ which makes the derivation $\{G : \top \to q\} \vdash_{\mathcal{L}} \varphi_F : \sigma \to q$ possible.

However, we can also derive $\varphi_F : q \to q$ from the type environment $\{G : q \to q\}$, and the edge from $F : q \to q$ to $\{G : q \to q\}$ is always stronger than the edge from $F : q \to q$ to $\{G : \top \to q\}$ in a typability game; if $\{G : \top \to q\}$ is a winning position of player 0, then so is $\{G : q \to q\}$, because $\top \to q \leq_{\mathcal{L}} q \to q$ and thus $\Gamma' \vdash_{\mathcal{L}} \varphi_G : \top \to q$ implies $\Gamma' \vdash_{\mathcal{L}} \varphi_G : q \to q$. Hence, player 0 can safely choose

---

[11]    Moreover, we can restrict the use of such strongest type bindings to the type derivations for the fixpoint variables that call $F$ in such a cycle.

$\{G : q \to q\}$ instead of $\{G : \top \to q\}$ from $F : q \to q$. As a result, we can refrain from constructing the edge from $F : q \to q$ to $\{G : \top \to q\}$ without undermining the completeness of the algorithm.

Moreover, although $F : \sigma \to q$ is derivable for any $\sigma$, we can refrain from adding type bindings other than $F : \top \to q$ and $F : q \to q$ to $\Gamma$ at this point of time. This is because for any $\Gamma'$ and $\sigma$ such that $\Gamma' \subseteq \Gamma$, $\sigma \neq \emptyset, \{q\}$, and $\Gamma' \vdash_{\mathcal{L}} \varphi_F : \sigma \to q$, we also have $\Gamma' \vdash_{\mathcal{L}} \varphi_F : \top \to q$ and the edge from $F : \sigma \to q$ to $\Gamma'$ is always weaker than the edge from $F : \top \to q$ to $\Gamma'$; if there is a strategy $\varsigma$ with $\varsigma(F : \sigma \to q) = \Gamma'$ that gives a winning strategy for player 0 from $F : \sigma \to q$, then the position $F : \top \to q$ is also winning, and thus player 0 can safely use $F : \top \to q$ instead of $F : \sigma \to q$ by exploiting the relation $\top \to q \leq_{\mathcal{L}} \sigma \to q$.

In general, when the body of a fixpoint variable $F$ is of the form $\lambda X . \psi_F$, an edge obtained from a type derivation $\Gamma \vdash_{\mathcal{L}} \psi_F : q$ is stronger than an edge obtained from a derivation $\Gamma' \vdash_{\mathcal{L}} \psi_F : q$ if and only if $\Gamma$ is weaker as a type environment than $\Gamma'$ (that is, we have $\forall X \in dom(\Gamma). \Gamma'(X) \leq_{\mathcal{L}} \Gamma(X)$). In our prototype model checker HOMUSAT, we use the inclusion relation $\Gamma \subseteq \Gamma'$ instead of $\forall X \in dom(\Gamma). \Gamma'(X) \leq_{\mathcal{L}} \Gamma(X)$ as it is computationally cheaper, although the relations are not equivalent and it requires further analysis to decide if the choice actually improves the performance of the model checker in general cases.

## D    Content of the Benchmark Set

In the following, we report on some details on the content of the benchmark set we used in the experiments in Section 5.[12]

Figure 6 shows a plot of the order of HES versus the size of LTS. As can be seen, the LTS size is moderate (up to around 10) for most of the instances, but some instances have large state sets. For example, `xhtmlf-div-2`, which was taken from the HORSAT2 benchmark set, consists of an HES of order-2 and an LTS of size 220. Although for such an instance the possible number of types is enormously huge, HOMUSAT solved it in fewer than 2 seconds.

Figure 7 shows the distribution of the numbers of alternations between $\mu$ and $\nu$ within the benchmark set. Whereas over half of the instances (83 out of 136) have no alternation, there are a certain number of instances that have one or more alternations, up to a maximum of 4. Note that, although the number of alternations is one of the factors that affect the difficulty of input problems, it is not so significant at least for the problems we used. Actually, all three problems that HOMUSAT timed out (`file-e`, `file~2`, and `intro`) have no alternation. All of them are $\nu$-only problems with order 3, $|Q| = 4$, and $|\mathcal{E}| \approx 5,000$, taken from HORSAT2 benchmarks.

---

[12]  It is available at `https://github.com/hopv/homusat/tree/master/bench/exp`.
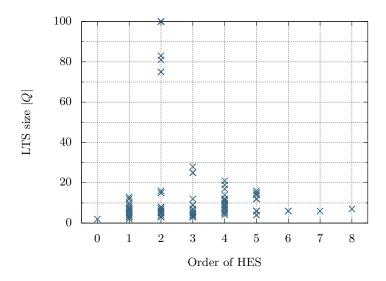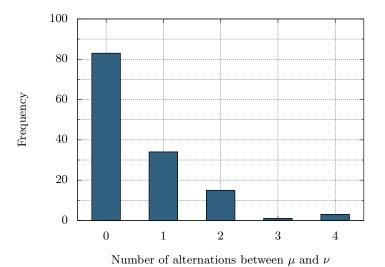
**Fig. 6.** Order of HES versus LTS size $|Q|$



**Fig. 7.** Distribution of the numbers of alternations