

Saturation-Based Model Checking of Higher-Order Recursion Schemes

Christopher Broadbent¹ and Naoki Kobayashi²

- 1 The University of Tokyo
chbroadbent@gmail.com
- 2 The University of Tokyo
koba@is.s.u-tokyo.ac.jp

Abstract

Model checking of higher-order recursion schemes (HORS) has recently been studied extensively and applied to higher-order program verification. Despite recent efforts, obtaining a scalable model checker for HORS remains a big challenge. We propose a new model checking algorithm for HORS, which combines two previous, independent approaches to higher-order model checking. Like previous type-based algorithms for HORS, it directly analyzes HORS and outputs intersection types as a certificate, but like Broadbent et al.'s saturation algorithm for collapsible pushdown systems (CPDS), it propagates information backward, in the sense that it starts with target configurations and iteratively computes their pre-images. We have implemented the new algorithm and confirmed that the prototype often outperforms TRECS and CSH0Re, the state-of-the-art model checkers for HORS.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Model checking, higher-order recursion schemes, intersection types

1 Introduction

The model checking of higher-order recursion schemes (higher-order model checking for short) [10, 22] is a generalization of finite-state and pushdown model checking, and has recently been applied to automated verification of higher-order programs [13, 23, 19]. Higher-order recursion schemes (HORS) are higher-order grammars describing possibly infinite trees, and can also be seen as simply-typed functional programs with recursion and tree constructors. Thus they serve as natural models for higher-order programs, and various verification problems for functional programs can be easily reduced to higher-order model checking [13, 11, 23, 26].

Despite the very bad worst-case complexity of higher-order model checking (k -EXPTIME complete for order- k HORS [22, 18]), several *practical* model checking algorithms [11, 14, 21, 4] have been developed, which do not immediately suffer from the hyper-exponential bottleneck. The state-of-the-art model checker TRECS can handle a few hundred lines of HORS generated from various program verification problems. It is, however, not scalable enough to support automated verification of thousands or millions of lines of code. Thus, obtaining a better higher-order model checker is a grand challenge in the field, and that is also the general goal of the present work.

The previous algorithms for higher-order model checking can be roughly classified into two radically different approaches, as shown in Table 1. The algorithms of Kobayashi [11, 14] and Neatherway et al. [21] directly work on HORS and check properties expressed by trivial tree automata [1] (Büchi tree automata where all the states are accepting, which can be used for describing safety properties stating that certain (bad) states are unreachable). For finitely



© Christopher Broadbent and Naoki Kobayashi;
licensed under Creative Commons License CC-BY

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

■ **Table 1** Classification of higher-order model checking algorithms

algorithms	models	properties	state representation	propagation
Type-based approach [11, 14, 21]	HORS	trivial	types	forward
Saturation-based approach [4, 5]	CPDS	co-trivial	stack automata	backward
HORSAT	HORS	co-trivial	types/automata	backward
HORSAT ^T	HORS	trivial	types/automata	backward

representing infinite states, they use intersection types [13, 17]. Information is propagated in the *forward* direction, in the sense that they start from the requirement that the start symbol must have the initial state of the trivial automaton as its type, and compute the “post-image” to collect type information needed to conclude that bad states are unreachable. On the other hand, the recent algorithm of Broadbent et al. [4, 5] works on collapsible pushdown systems (CPDS) (which are equi-expressive with HORS although the mutual translations are rather complex [9, 6]) and deals with properties expressed by *co-trivial* tree automata (Büchi tree automata where no states are accepting, which can be used for describing the complement of a safety property, stating that certain (bad) states are reachable). Their algorithm generalizes the saturation algorithm for pushdown system model checking [2, 7]. It finitely represents infinite states of CPDS by using stack automata, and propagates information in the *backward* direction, in the sense that it starts with final (bad) states, computes the “pre-image”, and checks whether the start state is in the pre-image. Broadbent et al. [5] recently report that with an optimization based on forward static analysis, a saturation-based model checker **CSHORE** can compete with TRECS [11, 12]. Due to the huge gap as summarized in Table 1, however, the two approaches have been of independent use, and it was difficult to transfer or integrate the techniques.

The present paper fills the gap between the two approaches, and proposes a new model-checking algorithm HORSAT and its variation HORSAT^T for HORS. As indicated in Table 1, the new algorithms are classified somewhere between the two approaches. Like the previous algorithms of Kobayashi and Neatherway et al., HORSAT works directly on HORS. Like Broadbent et al.’s saturation algorithm for CPDS [4, 5], however, HORSAT deals with *co-trivial* automata, and propagates information *backwards*, starting from final states and iteratively computing the pre-image. We use intersection types to represent the pre-image, but they can equally be interpreted as alternating tree automata accepting (tree representations of) the terms in the pre-image, which are somehow related to the stack automata representation used in Broadbent et al.’s saturation algorithm. We have implemented the new algorithms and obtained promising experimental results.

Besides filling the gap, the main advantage of the new algorithms over the previous algorithms for HORS model checking is the efficiency (both in theory and in practice). Unlike TRECS [11] or TravMC [21], they satisfy the fixed-parameter polynomial time complexity; thus it is expected to scale to large inputs better than TRECS. The previous fixed-parameter polynomial time algorithms GTRECS [14] for HORS did not scale well due to a large constant factor. According to the experiments, the new algorithm clearly outperforms GTRECS.

Compared with the saturation algorithm for CPDS [4, 5], the new algorithms have the following advantages:

- The new algorithms are much easier to understand, implement, and optimize. For example, the saturation algorithm for CPDS [4] uses non-standard automata called stack

automata to represent a set of CPDS states, while we can use standard tree automata (or equivalently intersection types) to represent a set of states (which are just applicative terms).

- It is much easier to verify the result of model checking HORS. It is partly because our algorithm works directly on HORS without a detour to CPDS, but also because the variant HORSAT of our algorithm can generate intersection types as a certificate, which is completely compatible with the certificate used by other model checking algorithms [11, 14], and whose validity can be easily checked (by an independent tool) based on the type-based characterization of trivial automata model checking [13].

In Section 2 we recall the nomenclature and foundational results of model checking with intersection types. Section 3 introduces the HORSAT algorithm and its application to co-trivial automata model-checking. We then proceed in Section 4 to the variant HORSAT, which can be used in the trivial case and generates a certificate when the HORS is safe. We follow up with some experimental results in Section 5.

2 Preliminaries

In this section, we review HORS, and tree automata (for infinite trees). We then define trivial/co-trivial automata model checking of HORS, and provide their characterizations in terms of intersection types.

We write $\text{dom}(f)$ and $\text{codom}(f)$ for the domain and co-domain of a map f . A *ranked alphabet*, denoted often by Σ , is a mapping from symbols to their arities. An (unlabeled) *tree* D is a subset of $\{1, \dots, m\}^*$ such that $\epsilon \in D$, and for every $\pi \in \{1, \dots, m\}$ and $k \in \{1, \dots, m\}$, $\pi k \in D$ implies $\{\pi\} \cup \{\pi i \mid 1 \leq i \leq k\} \subseteq D$. For a set L of symbols, an S -labeled tree is a map from a tree to S . For a ranked alphabet Σ , a Σ -labeled ranked tree T is a $\text{dom}(\Sigma)$ -labeled tree such that for every $\pi \in \text{dom}(T)$, $\{k \mid \pi k \in \text{dom}(T)\} = \{k \mid 1 \leq k \leq \Sigma(T(\pi))\}$.

Next we define *applicative terms*. The set of *sorts* is:

$$\kappa \text{ (sorts)} ::= \mathbf{o} \mid \kappa_1 \rightarrow \kappa_2$$

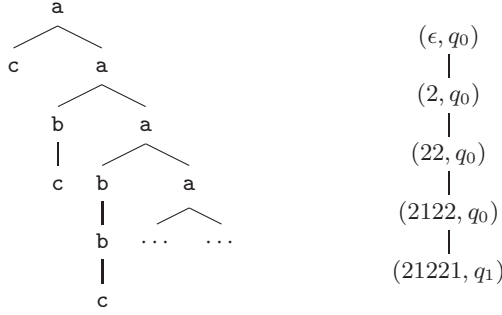
Here, the sort \mathbf{o} describes ranked trees. The *order* and *arity* of a sort κ , written $\text{ord}(\kappa)$ and $\text{ar}(\kappa)$ respectively, are:

$$\begin{aligned} \text{ord}(\mathbf{o}) &= 0 & \text{ord}(\kappa_1 \rightarrow \kappa_2) &= \max(\text{ord}(\kappa_1) + 1, \text{ord}(\kappa_2)) \\ \text{ar}(\mathbf{o}) &= 0 & \text{ar}(\kappa_1 \rightarrow \kappa_2) &= \text{ar}(\kappa_2) + 1 \end{aligned}$$

A *sort environment* is a finite map from variables to sorts. The set $\mathbf{ATerms}_{\Gamma, \Sigma, \kappa}$ of *applicative terms* having sort κ under a sort environment \mathcal{K} is defined inductively by: (i) $a \in \mathbf{ATerms}_{\Gamma, \Sigma, \mathbf{o}}$, (ii) $x \in \mathbf{ATerms}_{\Gamma, \Sigma, \kappa}$ if $\Gamma(x) = \kappa$, and (iii) $t_1 t_2 \in \mathbf{ATerms}_{\Gamma, \Sigma, \kappa}$ if $t_1 \in \mathbf{ATerms}_{\Gamma, \Sigma, \kappa' \rightarrow \kappa}$ and $t_2 \in \mathbf{ATerms}_{\Gamma, \Sigma, \kappa'}$ for some κ' .

► **Definition 1** (HORS). A (deterministic) higher-order recursion scheme (HORS), written \mathcal{G} , is a quadruple $(\Sigma, \mathcal{N}, \mathcal{R}, S)$, where

1. Σ is a *ranked alphabet*. The elements of $\text{dom}(\Sigma)$ are called *terminals*.
2. \mathcal{N} is a map from a finite set of symbols called *non-terminals* to sorts. $\text{dom}(\Sigma)$ and $\text{dom}(\mathcal{N})$ must be disjoint.
3. \mathcal{R} is a map from the set of non-terminals (i.e. $\text{dom}(\mathcal{N})$) to terms of the form $\lambda x_1. \dots \lambda x_\ell. t$, where (i) t is an applicative term, (ii) $\mathcal{N}(F)$ is of the form $\kappa_1 \rightarrow \dots \rightarrow \kappa_\ell \rightarrow \mathbf{o}$, and (iii) $t \in \mathbf{ATerms}_{\mathcal{N}\{x_1 \mapsto \kappa_1, \dots, x_\ell \mapsto \kappa_\ell\}, \Sigma, \mathbf{o}}$.



■ **Figure 1** $\text{Tree}(\mathcal{G}_1)$ (left) and a run-tree of \mathcal{A}_1 over $\text{Tree}(\mathcal{G}_1)$ (right).

4. S is a non-terminal called the *start symbol*. We require that $S \in \text{dom}(\mathcal{N})$ and $\mathcal{N}(S) = \mathbf{o}$. The *order* of a non-terminal F , written $\text{ord}(F)$, is the order of its sort, i.e. $\text{ord}(\mathcal{N}(F))$. The *order* of a HORS $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$, written $\text{ord}(\mathcal{G})$, is the highest order of its non-terminals.

Intuitively, a HORS $\mathcal{G} = (\Sigma, \mathcal{N}, \{F_1 \mapsto t_1, \dots, F_k \mapsto t_k\}, F_1)$ is a tree-generating, simply-typed call-by-name functional program, given by the recursive function definitions $F_1 = t_1, \dots, F_k = t_k$ with the main function F_1 and the tree constructors Σ . We sometimes write $\Sigma_{\mathcal{G}}, \mathcal{N}_{\mathcal{G}}, \mathcal{R}_{\mathcal{G}}, S_{\mathcal{G}}$ for the four components of \mathcal{G} . The reduction relation $\longrightarrow_{\mathcal{R}}$ on terms of sort \mathbf{o} is defined by:

$$\begin{aligned} F t_1 \cdots t_k &\longrightarrow_{\mathcal{R}} [t_1/x_1, \dots, t_k/x_k]t && \text{if } \mathcal{R}(F) = \lambda x_1 \cdots \lambda x_k. t \\ a t_1, \dots, t_i \cdots t_k &\longrightarrow_{\mathcal{R}} a t_1, \dots, t'_i \cdots t_k && \text{if } t_i \longrightarrow_{\mathcal{R}} t'_i \end{aligned}$$

Here, $[t_1/x_1, \dots, t_k/x_k]t$ denotes the term obtained from t by replacing all the (free) occurrences of x_1, \dots, x_k with t_1, \dots, t_k respectively. When $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$, we also write $s \longrightarrow_{\mathcal{G}} t$ for $s \longrightarrow_{\mathcal{R}} t$.

For an applicative term of sort \mathbf{o} , $(\Sigma \cup \{\perp \mapsto 0\})$ -labeled tree t^{\perp} (in the term representation) is defined by: (i) $(a t_1 \cdots t_k)^{\perp} = a t_1^{\perp} \cdots t_k^{\perp}$ and (ii) $(F t_1 \cdots t_k)^{\perp} = \perp$. The value tree [22] of \mathcal{G} , written $\mathbf{Tree}(\mathcal{G})$, is the $\Sigma \cup \{\perp \mapsto 0\}$ -labeled ranked trees obtained as the least upper bound of the set $\{t^{\perp} \mid S \longrightarrow_{\mathcal{G}} t\}$ with respect to the least precongruence \sqsubseteq that satisfies $\perp \sqsubseteq T$ for every tree T .

► **Example 2.** Consider $\mathcal{G}_1 = (\Sigma = \{a \mapsto 2, b \mapsto 1, c \mapsto 0\}, \mathcal{N}, \mathcal{R}, S)$ where:

$$\mathcal{N} = \{S \mapsto \mathbf{o}, F \mapsto \mathbf{o} \rightarrow \mathbf{o}\} \quad \mathcal{R} = \{S \mapsto F c, F \mapsto \lambda x. a x (F(b x))\}$$

S is reduced as follows, generating the tree in Figure 1.

$$S \longrightarrow_{\mathcal{G}_1} F c \longrightarrow_{\mathcal{G}_1} a c (F(b c)) \longrightarrow_{\mathcal{G}_1} \cdots$$

□

Next, we introduce tree automata, which are used for describing properties on trees.

► **Definition 3** (trivial/co-trivial ATA). An *alternating tree automaton* (ATA) is a quadruple $\mathcal{A} = (\Sigma, Q, \Delta, q_I)$, where Σ is a ranked alphabet, Q is a set of states, $q_I \in Q$ is the initial state, and $\Delta \subset Q \times \text{dom}(\Sigma) \times 2^{\{1, \dots, m\} \times Q}$ is a transition function where m is the largest arity in Σ . We require that if $(q, a, S) \in \Delta$, then $S \subseteq \{1, \dots, \Sigma(a)\} \times Q$. For a (possibly

infinite) Σ -labeled ranked tree T , a $(\text{dom}(T) \times Q)$ -labeled tree R is a *run-tree* of \mathcal{A} over T if (i) $R(\epsilon) = (\epsilon, q_I)$, and (ii) if $R(\pi) = (\pi', q)$, then there exists $S = \{(j_1, q_{k_1}), \dots, (j_\ell, q_{k_\ell})\}$ such that $(q, T(\pi'), S) \in \Delta$ and $\{i \mid \pi i \in \text{dom}(R)\} = \{1, \dots, \ell\}$ with $R(\pi i) = (\pi' j_i, q_{k_i})$ for each $i \in \{1, \dots, \ell\}$. A (possibly infinite) Σ -labeled ranked tree T is *accepted by \mathcal{A} in trivial mode* if there is a (possibly infinite) run-tree of \mathcal{A} over T , and T is accepted by \mathcal{A} in *co-trivial mode* if there is a *finite* run-tree of \mathcal{A} over T . ATA is called *trivial* (resp. *co-trivial*) if it accepts input trees in trivial (resp. co-trivial) mode. For an ATA $\mathcal{A} = (\Sigma, Q, \Delta, q_I)$ (with $\perp \notin \text{dom}(\Sigma)$), we write \mathcal{A}^\top for the ATA $(\Sigma \cup \{\perp \mapsto 0\}, Q, \Delta \cup \{(q, \perp, \emptyset)\}, q_I)$, and \mathcal{A}^\perp for $(\Sigma \cup \{\perp \mapsto 0\}, Q, \Delta, q_I)$,

► **Example 4.** Let $\mathcal{A}_1 = (\Sigma, \{q_0, q_1\}, \Delta_1, q_0)$ where Δ_1 is:

$$\{(q_0, \mathbf{a}, \{(1, q_0)\}), (q_0, \mathbf{a}, \{(2, q_0)\}), (q_0, \mathbf{b}, \{(1, q_1)\}), (q_1, \mathbf{b}, \emptyset), (q_1, \mathbf{a}, \{(1, q_0)\}), (q_1, \mathbf{a}, \{(2, q_0)\})\}.$$

In the co-trivial mode, \mathcal{A}_1 accepts trees that have a path containing two consecutive occurrences of **b**. Figure 1 shows a run-tree of \mathcal{A}_1 over $\mathbf{Tree}(\mathcal{G}_1)$. In the trivial mode, \mathcal{A}_1 additionally accepts trees having an infinite path. \square

► **Remark.** A trivial (co-trivial) ATA is a special case of alternating parity tree automaton [8], where all the states have priority 0 (resp. 1). Therefore, from a trivial automaton \mathcal{A} , one can construct a co-trivial automaton $\overline{\mathcal{A}}$ that accepts the complement of the trees accepted by \mathcal{A} , and vice versa.

► **Example 5.** Recall ATA \mathcal{A}_1 in Example 4. A tree is accepted by \mathcal{A}_1 in the co-trivial (resp. trivial) mode if and only if it is not accepted by the following ATA $\overline{\mathcal{A}}_1 = (\Sigma, \{\overline{q_0}, \overline{q_1}\}, \overline{\Delta}_1, \overline{q_0})$ in the trivial (resp. co-trivial) mode.

$$\overline{\Delta}_1 = \{(\overline{q_0}, \mathbf{a}, \{(1, \overline{q_0}), (2, \overline{q_0})\}), (\overline{q_0}, \mathbf{b}, \{(1, \overline{q_1})\}), (\overline{q_0}, \mathbf{c}, \emptyset), (\overline{q_1}, \mathbf{a}, \{(1, \overline{q_0}), (2, \overline{q_0})\}), (\overline{q_1}, \mathbf{c}, \emptyset)\}$$

In the present paper, we are interested in the following model checking problems.

► **Definition 6** (trivial/co-trivial ATA model checking). A trivial (resp. co-trivial) ATA model checking problem for HORS is the decision problem: “Given a HORS \mathcal{G} and an ATA \mathcal{A} as input, is $\mathbf{Tree}(\mathcal{G})$ accepted by \mathcal{A}^\top (resp. \mathcal{A}^\perp) in trivial (resp. co-trivial) mode?”

By Remark 2, the decidability of trivial/co-trivial ATA model checking follows immediately from that of alternating parity tree automata (APT) model checking for HORS [22], and a trivial ATA model checking problem can be reduced to a co-trivial ATA model checking, and vice versa.

Following Kobayashi and Ong’s type systems for HORS [13, 17], we provide below a type-based characterization of trivial/co-trivial ATA model checking for HORS. Fix an ATA $\mathcal{A} = (\Sigma, Q, \Delta, q_I)$. The set of types is given by:

$$\tau \text{ (types)} ::= q \mid \sigma \rightarrow \tau \quad \sigma \text{ (intersections)} ::= \bigwedge \{\tau_1, \dots, \tau_k\}$$

Intuitively, the type $q \in Q$ describes a tree accepted by \mathcal{A} from the state q (i.e., accepted by (Σ, Q, Δ, q)). The type $\sigma \rightarrow \tau$ describes a function that takes an element of (intersection) type σ as input, and returns an element of type τ . The intersection $\bigwedge \{\tau_1, \dots, \tau_k\}$ (where k may be 0) describes an element of the intersection of the sets denoted by τ_1, \dots, τ_k . When $k = 0$, we write \top for $\bigwedge \{\tau_1, \dots, \tau_k\}$. We often write $\tau_1 \wedge \dots \wedge \tau_k$ or $\bigwedge_{i \in \{1, \dots, k\}} \tau_i$ for

$\bigwedge\{\tau_1, \dots, \tau_k\}$. We assume that \bigwedge binds tighter than \rightarrow , so that $q_0 \wedge q_1 \rightarrow q_2$ means $(q_0 \wedge q_1) \rightarrow q_2$.

A type τ is called a *refinement* of a sort κ , written $\tau :: \kappa$ if $\tau :: \kappa$ is derivable by the following rules:

$$\frac{q \in Q}{q :: \circ} \quad \frac{\tau :: \kappa \quad \tau_i :: \kappa' \text{ for each } i \in I}{(\bigwedge_{i \in I} \tau_i \rightarrow \tau) :: \kappa' \rightarrow \kappa}$$

A *type environment* is a set of type bindings of the form $x : \tau$. For a type environment Γ , We write $\text{dom}(\Gamma)$ for the set $\{x \mid x : \tau \in \Gamma\}$. Unlike ordinary type systems, a type environment may contain multiple type bindings for the variable, like $\{x : q_0, x : q_1\}$. We sometimes omit curly brackets and just write $x_1 : \tau_1, \dots, x_n : \tau_n$ for $\{x_1 : \tau_1, \dots, x_n : \tau_n\}$. When $\sigma = \bigwedge\{\tau_1, \dots, \tau_n\}$, we also write $x : \sigma$ for $x : \tau_1, \dots, x : \tau_n$.

The rules for a type judgment $\Gamma \vdash_{\mathcal{A}, \mathcal{G}} t : \tau$ are:

$$\begin{array}{c} \frac{}{\Gamma \cup \{x : \tau\} \vdash_{\mathcal{A}, \mathcal{G}} x : \tau} \text{ (T-VAR)} \quad \frac{(q, a, \{(i, q_j) \mid 1 \leq i \leq \Sigma(a), j \in I_i\}) \in \Delta_{\mathcal{A}}}{\Gamma \vdash_{\mathcal{A}, \mathcal{G}} a : \bigwedge_{j \in I_1} q_j \rightarrow \dots \rightarrow \dots \bigwedge_{j \in I_{\Sigma(a)}} q_j \rightarrow q} \text{ (T-CON)} \\[10pt] \frac{\Gamma \vdash_{\mathcal{A}, \mathcal{G}} t_1 : \bigwedge_{i \in I} \tau_i \rightarrow \tau \quad \Gamma \vdash_{\mathcal{A}, \mathcal{G}} t_2 : \tau_i \text{ (for each } i \in I)}{\Gamma \vdash_{\mathcal{A}, \mathcal{G}} t_1 t_2 : \tau} \text{ (T-APP)} \quad \frac{\{x_i : \tau_j \mid 1 \leq i \leq k, j \in I_i\} \vdash_{\mathcal{A}, \mathcal{G}} t : q \quad \mathcal{R}_{\mathcal{G}}(F) = \lambda x_1. \dots \lambda x_k. t}{(\bigwedge_{j \in I_1} \tau_j \rightarrow \dots \rightarrow \bigwedge_{j \in I_k} \tau_j \rightarrow q) :: \mathcal{N}_{\mathcal{G}}(F)} \text{ (T-NT)} \end{array}$$

We sometimes omit the subscripts \mathcal{A} and \mathcal{G} when they are clear from the context.

The following are special cases of the soundness and completeness of Kobayashi and Ong's type system for APT model checking [17] (where the priorities are restricted to 0 and 1 respectively for Clauses (i) and (ii) of Theorem 7.¹

► **Theorem 7.** (i) $\text{Tree}(\mathcal{G})$ is accepted by \mathcal{A}^\top in the trivial mode if and only if there is a possibly infinite derivation tree for $\emptyset \vdash_{\mathcal{A}, \mathcal{G}} S : q_I$. (ii) $\text{Tree}(\mathcal{G})$ is accepted by \mathcal{A}^\perp in the co-trivial mode if and only if there is a finite derivation tree for $\emptyset \vdash_{\mathcal{A}, \mathcal{G}} S : q_I$.

The existing practical² model checking algorithms for HORS [11, 14, 21] deal with trivial ATA model checking, and try to construct an infinite derivation tree for $\emptyset \vdash_{\mathcal{A}, \mathcal{G}} S : q_I$ [21], or infer a set of types of non-terminals occurring in such a derivation tree [11, 14]. All of the algorithms run in the *forward* direction, in the sense that they start with $S : q_I$, and expand non-terminals to construct types/derivation trees while checking that invalid trees cannot be generated from S .

3 Co-Trivial ATA Model Checking

This section presents a co-trivial ATA model checking algorithm that runs in the backward manner. We first note the following property (see Appendix B for a proof).

► **Lemma 8.** Let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ be a HORS. $\text{Tree}(\mathcal{G})$ is accepted by a co-trivial ATA \mathcal{A}^\perp if and only if there exists a term t such that $S \rightarrow_{\mathcal{G}}^* t$ and t^\perp is accepted by \mathcal{A}^\perp .

¹ Kobayashi and Ong [17] do not consider HORS's that generate trees containing \perp ; see Section A in Appendix on how to derive the result below for \perp -generating HORS.

² Since the worst-case complexity of trivial/co-trivial ATA model checking is n -EXPTIME complete [22, 18], we call a model checking algorithm *practical* if it terminates in a realistic time (say, in a few minutes, rather than in several years) for typical inputs.

By the lemma above, for co-trivial ATA model checking, it suffices to start with the set \mathcal{T}_0 of terms:

$$\{t \in \mathbf{ATerms}_{\mathcal{N}, \Sigma, \circ} \mid t^\perp \text{ is accepted by } \mathcal{A}^\perp \text{ in the co-trivial mode}\},$$

expand it to the set \mathcal{T} of all the terms that can be reduced to a term in \mathcal{T}_0 , and check whether $S \in \mathcal{T}$. Since \mathcal{T} may be *infinite* in general, we represent a (possibly infinite) set of terms by using a *finite* type environment Γ for non-terminals. We write \mathbf{Terms}_Γ for the set $\{t \mid \Gamma \vdash_{\mathcal{A}}^- t : q_I\}$ of terms. Here, $\Gamma \vdash_{\mathcal{A}}^- t : \tau$ means that there is a finite derivation for $\Gamma \vdash_{\mathcal{A}, \mathcal{G}} t : \tau$ that does not use rule T-NT (so that \mathcal{G} does not matter), where non-terminals are treated as variables. Then, the initial set \mathcal{T}_0 above is represented by the empty type environment \emptyset , i.e., $\mathbf{Terms}_\emptyset = \mathcal{T}_0$, as stated by the following lemma: see Appendix B for a proof.

► **Lemma 9.** $\emptyset \vdash_{\mathcal{A}}^- t : q_I$ if and only if t^\perp is accepted by \mathcal{A}^\perp in the co-trivial mode.

We shall construct below a monotonic function \mathcal{F} on type environments that satisfies the following conditions:

- (I) $\{t \in \mathbf{ATerms}_{\mathcal{N}, \Sigma, \circ} \mid \exists t'. t \rightarrow_{\mathcal{G}} t' \in \mathbf{Terms}_\Gamma\} \subseteq \mathbf{Terms}_{\mathcal{F}(\Gamma)}$; and
- (II) $\mathbf{Terms}_{\mathcal{F}^i(\emptyset)} \subseteq \{t \mid \exists t'. t \rightarrow_{\mathcal{G}}^* t' \in \mathcal{T}_0\}$ for every i .

Then, we have $\mathbf{Terms}_{\bigcup_{i \in \omega} \mathcal{F}^i(\emptyset)} = \mathcal{T}$. Since \mathcal{F} is monotonic and a type environment ranges over a finite set $\{\Gamma \mid \text{dom}(\Gamma) \subseteq \text{dom}(\mathcal{N}) \text{ and } \forall F : \tau \in \Gamma. \tau :: \mathcal{N}(F)\}$, we can obtain $\bigcup_{i \in \omega} \mathcal{F}^i(\emptyset)$ by computing $\mathcal{F}(\emptyset)$, $\mathcal{F}^2(\emptyset)$, $\mathcal{F}^3(\emptyset)$, ... until it converges; we can then check whether $S : q_I \in \bigcup_{i \in \omega} \mathcal{F}^i(\emptyset)$ holds to decide whether $\mathbf{Tree}(\mathcal{G})$ is accepted by \mathcal{A}^\perp in the co-trivial mode, as stated below.

► **Lemma 10.** Suppose that a monotonic function \mathcal{F} on type environments satisfies the two conditions above. Then, $S : q_I \in \bigcup_{i \in \omega} \mathcal{F}^i(\emptyset)$ if and only if $\mathbf{Tree}(\mathcal{G})$ is accepted by \mathcal{A} .

Proof. By the first condition of \mathcal{F} , we have $\{t \mid \exists t'. t \rightarrow_{\mathcal{G}}^i t' \in \mathbf{Terms}_\emptyset\} \subseteq \mathbf{Terms}_{\mathcal{F}^i(\emptyset)}$. Thus, we have $\mathcal{T} = \mathbf{Terms}_{\bigcup_{i \in \omega} \mathcal{F}^i(\emptyset)}$. The required result follows by Lemma 8. \square

It remains to construct \mathcal{F} that satisfies the conditions (I) and (II) above. The following lemma provides a clue as to how to construct \mathcal{F} . It states that typing is closed under the inverse of substitutions: see Section B for a proof.

► **Lemma 11.** Suppose $s \in \mathbf{ATerms}_{\mathcal{K} \cup \{x_1 : \kappa_1, \dots, x_\ell : \kappa_\ell\}, \Sigma, \circ}$ and $t_i \in \mathbf{ATerms}_{\mathcal{K}, \Sigma, \kappa_i}$ for each $i \in \{1, \dots, \ell\}$. If $\Gamma \vdash_{\mathcal{A}}^- [t_1/x_1, \dots, t_\ell/x_\ell]s : q$ with $\Gamma :: \mathcal{K}$, then there exist (possibly empty) sets I_i and $\{\tau_j \mid j \in I_i\}$ for each $i \in \{1, \dots, \ell\}$ such that: (i) $\Gamma \cup \{x_i : \tau_j \mid i \in \{1, \dots, \ell\}, j \in I_i\} \vdash_{\mathcal{A}}^- s : q$; (ii) $\Gamma \vdash_{\mathcal{A}}^- t_i : \tau_j$ for each $i \in \{1, \dots, \ell\}$ and $j \in I_i$; and (iii) $\tau_j :: \kappa_i$ for every $j \in I_i$.

The above lemma implies that if $F t_1 \cdots t_\ell \rightarrow_{\mathcal{G}} [t_1/x_1, \dots, t_\ell/x_\ell]s$ and $\Gamma \vdash_{\mathcal{A}}^- [t_1/x_1, \dots, t_\ell/x_\ell]s : q$, then $\Gamma \cup \{F : \bigwedge_{j \in I_1} \tau_1 \rightarrow \cdots \rightarrow \bigwedge_{j \in I_\ell} \tau_j \rightarrow q\} \vdash_{\mathcal{A}} F t_1 \cdots t_\ell : q$ holds, where I_i and τ_j are as given by the lemma above. This motivates us to define $\mathcal{F}_{\mathcal{G}, \mathcal{A}, \mathcal{R}}$ (where $\mathcal{R} \subseteq \mathcal{R}_{\mathcal{G}}$) as follows.

$$\begin{aligned} \mathcal{F}_{\mathcal{G}, \mathcal{A}, \mathcal{R}}(\Gamma) &= \Gamma \cup \{F : \Gamma_V(x_1) \rightarrow \cdots \rightarrow \Gamma_V(x_\ell) \rightarrow q \mid \mathcal{R}(F) = \lambda x_1. \cdots \lambda x_\ell. t, \\ &\quad \mathcal{N}(F) = \kappa_1 \rightarrow \cdots \rightarrow \kappa_\ell \rightarrow \circ, \text{ and } x_1 : \kappa_1, \dots, x_\ell : \kappa_\ell; \Gamma \vdash_{\mathcal{A}} t : q \Rightarrow \Gamma_V\}. \end{aligned}$$

Here, $\Gamma(x)$ is an abbreviation of $\bigwedge\{\tau \mid x : \tau \in \Gamma\}$, and the relation $\mathcal{K}; \Gamma_N \vdash_{\mathcal{A}} t : \tau \Rightarrow \Gamma_V$ is defined by:


```

 $\Gamma := \emptyset;$ 
while not( $\mathcal{F}_{\mathcal{G},\mathcal{A}}(\Gamma) = \Gamma$ ) do ( $\Gamma := \mathcal{F}_{\mathcal{G},\mathcal{A}}(\Gamma)$ ; if  $S : q_I \in \Gamma$  then return TRUE);
return FALSE

```

■ **Figure 2** Co-trivial ATA model checking algorithm HORSAT.

$$\begin{array}{c}
\frac{x : \tau \in \text{Inhabited}(\mathcal{K}, \Gamma_N)}{\mathcal{K}; \Gamma_N \vdash_{\mathcal{A}} x : \tau \Rightarrow x : \tau} \quad \frac{\emptyset \vdash_{\mathcal{A},\mathcal{G}} a : \tau}{\mathcal{K}; \Gamma_N \vdash_{\mathcal{A}} a : \tau \Rightarrow \emptyset} \quad \frac{}{\mathcal{K}; \Gamma_N \cup \{F : \tau\} \vdash_{\mathcal{A}} F : \tau \Rightarrow \emptyset} \\
\\
\frac{\mathcal{K}; \Gamma_N \vdash_{\mathcal{A}} t_1 : \bigwedge_{i \in I} \tau_i \rightarrow \tau \Rightarrow \Gamma_0 \quad \mathcal{K}; \Gamma_N \vdash_{\mathcal{A}} t_2 : \tau_i \Rightarrow \Gamma_i \text{ (for each } i \in I) \quad \Gamma_0 \cup \bigcup_{i \in I} \Gamma_i \in \text{Inhabited}(\mathcal{K}, \Gamma_N)}{\mathcal{K}; \Gamma_N \vdash_{\mathcal{A}} t_1 t_2 : \tau \Rightarrow \Gamma_0 \cup \bigcup_{i \in I} \Gamma_i}
\end{array}$$

where $\text{Inhabited}(\mathcal{K}, \Gamma_N)$ is the set of type environments for which the corresponding term environments can be constructed from non-terminals in Γ_N , i.e.:

$$\{\Gamma \mid \Gamma :: \mathcal{K} \text{ and } \forall x \in \text{dom}(\Gamma). \exists s \in \mathbf{ATerms}_{\mathcal{N}, \Sigma, \mathcal{K}(x)}. \forall x : \tau \in \Gamma. \Gamma_N \vdash_{\mathcal{A}}^- s : \tau\}.$$

The relation $\mathcal{K}; \Gamma_N \vdash_{\mathcal{A}} t : \tau \Rightarrow \Gamma_V$ intuitively means that $\Gamma_N, \Gamma_V \vdash t : \tau$ holds and that Γ_V is inhabited, as stated by the following lemma (which follows by straightforward induction).

► **Lemma 12.** *If $\mathcal{K}; \Gamma_N \vdash t : \tau \Rightarrow \Gamma_V$, then $\Gamma_N \cup \Gamma_V \vdash t : \tau$ and $\Gamma_V \in \text{Inhabited}(\mathcal{K}, \Gamma_N)$. Conversely, if $\Gamma_N \cup \Gamma_V \vdash_{\mathcal{A}}^- t : \tau$ and $\Gamma_V \in \text{Inhabited}(\mathcal{K}, \Gamma_N)$, then $\mathcal{K}; \Gamma_N \vdash t : \tau \Rightarrow \Gamma'_V$ for some Γ'_V such that $\Gamma'_V \subseteq \Gamma_V$.*

Reading the rules for $\mathcal{K}; \Gamma_N \vdash_{\mathcal{A}} t : \tau \Rightarrow \Gamma_V$ in a bottom-up manner, we can interpret them as an algorithm which, given \mathcal{K}, Γ_N and τ as input, outputs Γ_V such that $\Gamma_N, \Gamma_V \vdash_{\mathcal{A}}^- t : \tau$ and $\Gamma_V \in \text{Inhabited}(\mathcal{K}, \Gamma_N)$. For checking the inhabitation condition $\Gamma \in \text{Inhabited}(\mathcal{K}, \Gamma_N)$, we can use Rehof and Urzyczyn's reduction to the emptiness problem of alternating tree automata [25]: see Appendix C.

We write $\mathcal{F}_{\mathcal{G},\mathcal{A}}$ for $\mathcal{F}_{\mathcal{G},\mathcal{A},\mathcal{R}_{\mathcal{G}}}$, and also omit the subscripts when they are clear from the context. The following lemma justifies the definition of $\mathcal{F}_{\mathcal{G},\mathcal{A},\mathcal{R}}$.

► **Lemma 13.** *Suppose $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}', S)$ and $\mathcal{R} \subseteq \mathcal{R}'$. If $\Gamma \vdash_{\mathcal{A}}^- t : q$ and $s \rightarrow_{\mathcal{R}} t$, then $\mathcal{F}_{\mathcal{G},\mathcal{A},\mathcal{R}}(\Gamma) \vdash_{\mathcal{A}}^- s : q$.*

Proof. The proof proceeds by induction on the derivation of $t \rightarrow_{\mathcal{R}} t'$. Since the induction step is trivial, we show only the base case, where $t = F t_1 \cdots t_\ell$ and $t' = [t_1/x_1, \dots, t_\ell/x_\ell]s$ with $\lambda x_1. \cdots \lambda x_\ell. s = \mathcal{R}(F)$. Let $\mathcal{N}(F) = \kappa_1 \rightarrow \cdots \rightarrow \kappa_\ell \rightarrow \circ$. By Lemma 11, we have:

$$\Gamma \cup \{x_i : \tau_j \mid i \in \{1, \dots, \ell\}, j \in I_i\} \vdash_{\mathcal{A}}^- s : q \quad \Gamma \vdash_{\mathcal{A}}^- t_i : \tau_j \text{ for each } i \in \{1, \dots, \ell\}, j \in I_i$$

and $\tau_j :: \kappa_i$ for every $j \in I_i$. By Lemma 12, there exists Γ_V such that

$$\{x_1 : \kappa_1, \dots, x_\ell : \kappa_\ell\}; \Gamma \vdash_{\mathcal{A}}^- s : q \Rightarrow \Gamma_V \quad \Gamma \vdash_{\mathcal{A}}^- t_i : \tau \text{ for each } x_i : \tau \in \Gamma_V.$$

Thus, we have $F : \Gamma_V(x_1) \rightarrow \cdots \rightarrow \Gamma_V(x_\ell) \rightarrow q \in \mathcal{F}_{\mathcal{G},\mathcal{A},\mathcal{R}}(\Gamma)$, which implies $\mathcal{F}_{\mathcal{G},\mathcal{A},\mathcal{R}}(\Gamma) \vdash_{\mathcal{A}}^- t_1 \cdots t_\ell : q$ as required. \square

The whole algorithm is given in Figure 2. The following theorem guarantees the soundness and completeness of the algorithm.

► **Theorem 14.** *Let the function $\mathcal{F}_{\mathcal{G}, \mathcal{A}}$ be as defined above. Then, $S : q_I \in \bigcup_{i \in \omega} \mathcal{F}_{\mathcal{G}, \mathcal{A}}^i(\emptyset)$ if and only if $\mathbf{Tree}(\mathcal{G})$ is accepted by \mathcal{A}^\perp in the co-trivial mode.*

Proof. By Lemma 10, it suffices to show that $\mathcal{F} = \mathcal{F}_{\mathcal{G}, \mathcal{A}}$ satisfies the two conditions (I) and (II). Condition (I) follows immediately from Lemma 13.

To check condition (II), suppose $t \in \mathbf{Terms}_{\mathcal{F}^m(\emptyset)}$, i.e., $\mathcal{F}^m(\emptyset) \vdash_{\mathcal{A}}^- t : q_I$. We construct the following HORS $\mathcal{G}^{(m)}$ as an approximation of \mathcal{G} :

$$\begin{aligned} \mathcal{G}^{(m)} &= (\Sigma \cup \{\perp \mapsto 0\}, \mathcal{N}^{(m)}, \mathcal{R}^{(m)}, F_1^{(m)}) & \mathcal{N}^{(m)} &= \{F_i^{(j)} \mid i \in \{1, \dots, n\}, j \in \{0, \dots, m\}\} \\ \mathcal{R}^{(m)} &= \{F_i^{(0)} \mapsto \lambda x_1 \dots \lambda x_\ell. \perp \mid \mathbf{ar}(\mathcal{N}(F_i)) = \ell\} \\ &\quad \cup \{F_i^{(j)} \mapsto [F_1^{(j-1)} / F_1, \dots, F_n^{(j-1)} / F_n] \mathcal{R}(F_i) \mid j \geq 1\} \end{aligned}$$

$\mathcal{G}^{(m)}$ is a HORS without recursion, obtained by unfolding non-terminals of \mathcal{G} . We write $t^{(m)}$ for the term obtained by replacing each non-terminal F in t with $F^{(m)}$. By the assumption $\mathcal{F}^m(\emptyset) \vdash_{\mathcal{A}}^- t : q_I$, we have $\emptyset \vdash_{\mathcal{A}^\perp, \mathcal{G}^{(m)}} t^{(m)} : q_I$: see Lemma 23 in Appendix B. By the strong normalization of the simply-typed λ -calculus, we have $t^{(m)} \rightarrow_{\mathcal{G}^{(m)}}^* u$ for some $(\Sigma \cup \{\perp \mapsto 0\})$ -labeled tree (in the term representation) u . By the type preservation property (Lemma 25 in Appendix B), we have $\emptyset \vdash_{\mathcal{A}^\perp, \mathcal{G}^{(m)}} u : q_I$. By Lemma 9, u is accepted by \mathcal{A}^\perp . By the construction of $t^{(m)}$, there exists a term t' such that $t \rightarrow_{\mathcal{G}}^* t'$ and $t'^\perp = u$. Thus, we have $t \in \{s \mid \exists t'. s \rightarrow_{\mathcal{G}}^* t' \in \mathcal{T}_0\}$ as required. \square

► **Example 15.** Let \mathcal{G} be \mathcal{G}_1 in Example 2 and \mathcal{A} be \mathcal{A}_1 in Example 4. Then, since $\emptyset \vdash_{\mathcal{A}} \mathbf{a} x (F(\mathbf{b} x)) : q_0 \Rightarrow x : q_0$ and $\emptyset \vdash_{\mathcal{A}} \mathbf{a} x (F(\mathbf{b} x)) : q_1 \Rightarrow x : q_0$, we have:

$$\mathcal{F}(\emptyset) = \{F : q_0 \rightarrow q_0, F : q_0 \rightarrow q_1\}.$$

Similarly, $\mathcal{F}^2(\emptyset), \mathcal{F}^3(\emptyset), \dots$ are computed as follows.

$$\begin{aligned} \mathcal{F}^2(\emptyset) &= \mathcal{F}(\emptyset) \cup \{F : q_1 \rightarrow q_0, F : q_1 \rightarrow q_1\} & \mathcal{F}^3(\emptyset) &= \mathcal{F}^2(\emptyset) \cup \{F : \top \rightarrow q_0, F : \top \rightarrow q_1\} \\ \mathcal{F}^4(\emptyset) &= \mathcal{F}^3(\emptyset) \cup \{S : q_0, S : q_1\} = \mathcal{F}^5(\emptyset) \end{aligned}$$

Thus, $\bigcup_{i \in \omega} \mathcal{F}^i(\emptyset) = \mathcal{F}^4(\emptyset)$ contains $S : q_0$, which implies that $\mathbf{Tree}(\mathcal{G}_1)$ is accepted by \mathcal{A}_1 in the co-trivial mode. \square

► **Remark.** The inhabitation check needed for computing $\mathcal{F}(\Gamma)$ can be quite costly: in fact, the inhabitation problem is EXPTIME-complete [25]. In order to avoid the drawback, we can actually replace $\mathbf{Inhabited}(\mathcal{K}, \Gamma_N)$ in the rules for the relation $\mathcal{K}; \Gamma_N \vdash_{\mathcal{A}} t : \tau \Rightarrow \Gamma_V$ with an over-approximation $\mathbf{Inhabited}'(\mathcal{K}, \Gamma_N)$, such that

$$\mathbf{Inhabited}(\mathcal{K}, \Gamma_N) \subseteq \mathbf{Inhabited}'(\mathcal{K}, \Gamma_N) \subseteq \{\Gamma \mid \Gamma :: \mathcal{K}\}.$$

By the proof of Theorem 14, our co-trivial model checking algorithm remains sound and complete for any such over-approximation.

► **Remark.** Like Kobayashi's GTRecS algorithm [14], the co-trivial ATA model checking algorithm above runs in time polynomial in the size of HORS under the assumption that the other parameters (the size of the co-trivial automaton, the order of HORS, and the largest arity of non-terminals in HORS). First, the number of iterations is also linear in the size of HORS since the largest size of type environments is linear in the size of HORS (under the fixed-parameter assumption above). The cost for computing $\mathcal{F}(\Gamma)$ is also linear in the size of HORS, as the inhabitation check can be performed in a constant time (again, under the fixed-parameter assumption above): see Appendix C.

4 Trivial ATA Model Checking

This section presents a saturation-based algorithm for *trivial* ATA model checking. *Co-trivial* model checking is more natural for the saturation-based method, but many typical program verification problems are reduced to trivial ATA model checking problems for HORS [13, 11, 19], so that a program is safe if and only if the tree generated by a corresponding HORS is accepted by a trivial ATA. Although trivial ATA model checking can be reduced to co-trivial model checking by negating the trivial ATA (so that the co-trivial ATA accepts “error configurations” of a program), that approach is not good at *certifying* that a co-trivial property is *not* satisfied. When a co-trivial property is satisfied, then based on Theorem 7, one can generate a derivation tree for $\emptyset \vdash_{\mathcal{A}, \mathcal{G}} S : q_I$ or the types of non-terminals required in the derivation tree as a certificate. When a co-trivial property is not satisfied, however, the possible witness is a fixedpoint of \mathcal{F} , which can be very large. It is also difficult to validate the witness independently of the model checking algorithm, especially when the saturation algorithm is combined with other static analysis as in [5] and our implementation reported in Section 5. This is in contrast with the existing trivial automata model checking algorithms for HORS [11, 14, 21], which can output the types of non-terminals as a certificate when a property is satisfied. Checking the certificates amounts to type checking for an intersection type system, which can be performed independently of the model checking algorithms. This section modifies the saturation-based algorithm so that it can deal with trivial model checking directly (rather than negating the property) and generate certificates.

We first clarify the notion of “certificates”. Define the function $Shrink_{\mathcal{G}}$ on type environments by:

$$Shrink_{\mathcal{G}}(\Gamma) = \{F : \sigma_1 \rightarrow \cdots \rightarrow \sigma_k \rightarrow q \in \Gamma \mid \mathcal{R}(F) = \lambda x_1. \cdots \lambda x_\ell. t \text{ and } \Gamma, x_1 : \sigma_1, \dots, x_\ell : \sigma_\ell \vdash_{\mathcal{A}} t : q\}.$$

Intuitively, $Shrink_{\mathcal{G}}(\Gamma)$ picks each type binding $F : \sigma_1 \rightarrow \cdots \rightarrow \sigma_k \rightarrow q$ in Γ , checks whether it is valid in the sense that the body of F has the same type, and filters out invalid ones. We omit the subscript \mathcal{G} when it is clear from context. We write $\vdash \mathcal{G} : \Gamma$ if $Shrink_{\mathcal{G}}(\Gamma) = \Gamma$. We also write $\Gamma \vdash (\mathcal{G}, t) : \tau$ if $\vdash \mathcal{G} : \Gamma$ and $\Gamma \vdash t : \tau$ hold. The following theorem is due to Kobayashi [13]. It also follows immediately from Theorem 7: see Appendix B.

► **Theorem 16.** *Tree(\mathcal{G}) is accepted by \mathcal{A}^\top in trivial mode if and only if $\Gamma \vdash (\mathcal{G}, S) : q_I$.*

Based on Theorem 16, a type environment Γ such that $\Gamma \vdash (\mathcal{G}, S) : q_I$ serves as a certificate for **Tree**(\mathcal{G}) being accepted by \mathcal{A} .

► **Example 17.** Recall \mathcal{G}_1 in Example 2. Let $\mathcal{A}_2 = (\{\mathbf{a} \mapsto 2, \mathbf{b} \mapsto 1, \mathbf{c} \mapsto 0\}, \{q_0, q_1\}, \Delta_2, q_0)$ where

$$\Delta_2 = \{(q_0, \mathbf{a}, \{(1, q_0), (2, q_0)\}), (q_0, \mathbf{b}, \{(1, q_1)\}), (q_0, \mathbf{c}, \emptyset), (q_1, \mathbf{b}, \{(1, q_1)\}), (q_1, \mathbf{c}, \emptyset)\}.$$

\mathcal{A}_2 describes the property that \mathbf{a} cannot occur below \mathbf{b} . **Tree**(\mathcal{G}_1) is accepted by \mathcal{A}_2 , and the type environment $\{S : q_0, F : q_0 \wedge q_1 \rightarrow q_0\}$ serves as a certificate. \square

The existing algorithms for trivial automata model checking [11, 14] first compute an overapproximation Γ' of a possible certificate, compute the fixedpoint $\Gamma = \bigcap_{j \in \omega} Shrink^j(\Gamma')$, and check whether $S : q_I \in \Gamma$. We show below that such an overapproximation Γ' can be computed by using \mathcal{F} . For that purpose, we just need to replace the initial type environment \emptyset with the following type environment Γ_0 :

$$\Gamma_0 = \{F : \underbrace{\top \rightarrow \cdots \rightarrow \top}_{\text{ar}(\mathcal{N}(F))} \rightarrow q \mid F \in \text{dom}(\mathcal{N})\}$$

```

 $\Gamma' := \Gamma_0;$ 
while not( $\mathcal{F}_{\mathcal{G}, \mathcal{A}}(\Gamma') = \Gamma'$ ) do
  ( $\Gamma' := \mathcal{F}_{\mathcal{G}, \mathcal{A}}(\Gamma')$ ;  $\Gamma := \Gamma'$ ; (while not( $\text{Shrink}(\Gamma) = \Gamma$ ) do  $\Gamma := \text{Shrink}(\Gamma)$ );
  if  $S : q_I \in \Gamma$  then return  $\Gamma$  );
return FALSE

```

■ **Figure 3** Trivial ATA model checking algorithm HORSAT_T.

Terms_{Γ_0} is the set of terms t such that t^\perp is accepted by \mathcal{A}^\top . Thus, for $\Gamma' = \bigcup_{i \in \omega} \mathcal{F}^i(\Gamma_0)$, $\text{Terms}_{\Gamma'}$ is the set $\{t \mid t \xrightarrow{*}_{\mathcal{G}} t' \text{ and } t'^\perp \text{ is accepted by } \mathcal{A}^\top\}$. The type environment Γ' itself is not a certificate: in fact, *any* term t that has a non-terminal as its head is an element of $\text{Terms}_{\Gamma'}$, since $t^\perp = \perp$ and \perp is accepted by \mathcal{A}^\top . As the following theorem shows, however, Γ' serves as an overapproximation of a possible certificate.

► **Theorem 18.** $S : q_I \in \bigcap_{j \in \omega} \text{Shrink}^j(\bigcup_{i \in \omega} \mathcal{F}^i(\Gamma_0))$ if and only if $\text{Tree}(\mathcal{G})$ is accepted by \mathcal{A}^\top in the trivial mode.

Based on the theorem, we obtain the trivial ATA model checking algorithm shown in Figure 3. The outer loop repeatedly computes $\mathcal{F}(\Gamma_0), \mathcal{F}^2(\Gamma_0), \dots$, until it converges to a fixedpoint, or finds a certificate. The inner loop computes $\Gamma = \bigcap_{j \in \omega} \text{Shrink}^j(\mathcal{F}^i(\Gamma_0))$. If $S : q_I \in \Gamma$ then the algorithm terminates and returns Γ as a certificate. Otherwise, the algorithm eventually returns **FALSE** when a fixedpoint of \mathcal{F} is reached but $S : q_I \in \Gamma$ does not hold. Like the co-trivial ATA model checking algorithm in Figure 2, the algorithm terminates as soon as a certificate is found.

► **Example 19.** Recall \mathcal{G}_1 in Example 2 and \mathcal{A}_2 in Example 17. $\Gamma_0, \mathcal{F}(\Gamma_0), \mathcal{F}^2(\Gamma_0), \dots$ are computed as follows.

$$\begin{aligned} \Gamma_0 &= \{S : q_0, S : q_1, F : \top \rightarrow q_0, F : \top \rightarrow q_1\} \\ \mathcal{F}(\Gamma_0) &= \Gamma_0 \cup \{F : q_0 \rightarrow q_0\} \quad \mathcal{F}^2(\Gamma_0) = \mathcal{F}(\Gamma_0) \cup \{F : q_0 \wedge q_1 \rightarrow q_0\} = \mathcal{F}^3(\Gamma_0) \end{aligned}$$

For $\Gamma' = \mathcal{F}^2(\Gamma_0)$, $\text{Shrink}^i(\Gamma')$ ($i = 1, 2, \dots$) are:

$$\begin{aligned} \text{Shrink}(\Gamma') &= \{S : q_0, S : q_1, F : q_0 \rightarrow q_0, F : q_0 \wedge q_1 \rightarrow q_0\} \\ \text{Shrink}^2(\Gamma') &= \{S : q_0, F : q_0 \wedge q_1 \rightarrow q_0\} = \text{Shrink}^3(\Gamma'). \end{aligned}$$

Since $S : q_0 \in \text{Shrink}^2(\Gamma') = \bigcap_{j \in \omega} \text{Shrink}^j(\Gamma')$, we can conclude that $\text{Tree}(\mathcal{G})$ is accepted by \mathcal{A}_2 in trivial mode.

► **Remark.** GTRECS algorithm [14] is obtained by replacing \mathcal{F} in Figure 3 with the function *Expand* in [14]. The functions *Expand* and \mathcal{F} are quite different, however; the former uses a *game-semantic* idea [14, 24] to propagate the requirement that S should have type q_I in the *forward* direction, while the latter propagates information backwards using *purely type-based* techniques. Although both algorithms enjoy the fixed-parameter linear time complexity [14], according to experiments (see Section 5), GTRECS tends to be much slower. This is attributed to the game-semantics interpretation of intersection types. For example, a function of type $q_1 \rightarrow \dots \rightarrow q_n \rightarrow q$ may have any type of the form $\sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow q$ for $\sigma_i \in \{\top, q_i\}$ in the interpretation of [14]; thus the number of possible types.

5 Experiments

We have implemented a new higher-order model checker incorporating both the co-trivial HORSAT and trivial HORSATT model checking algorithms described respectively in Sections 3 and 4. As in [5], we have optimized the algorithms by using forward flow analysis, to exclude irrelevant type bindings; more precisely, the definition of $Inhabited(\mathcal{K}, \Gamma_N)$ in Section 3 has been replaced by

$$\{\Gamma \mid \Gamma :: \mathcal{K} \text{ and } \forall x \in dom(\Gamma). \exists s \in \mathbf{ATerms}_{\mathcal{N}, \Sigma, \mathcal{K}(x)} \cap \mathbf{Flow}(x). \forall x : \tau \in \Gamma. \Gamma_N \vdash_{\mathcal{A}} s : \tau\}.$$

where $\mathbf{Flow}(x)$ is an overapproximation of the terms that may flow to x in a reduction sequence from the start symbol S . We use OCFA to compute $\mathbf{Flow}(x)$. The implementation was compared to other type-based model checkers for HORS: TRECS [11], GTRECS2 [15] (which is a successor of GTRECS [14]) and TravMC [21], and a saturation-based model checker CSHORe [5] for CPDS. Since GTRECS2 has different variants of the algorithm for proving safety properties and their complements (eventually giving up if it cannot prove the property) we only ran the appropriate version of GTRECS2 on each example.

The benchmark suite consists of five categories of inputs (separated by horizontal lines), which have been collected from different applications of trivial automata model checking of HORS [20, 23, 19, 26, 16]: the first one from the HMTT verification tool [20], the second from the PMRS model checker [23], the third from software model checker MoChi [19], the fourth from exact flow analysis [26], and the fifth from applications to data compression [16]. The inputs have been automatically generated from program verification problems except for those in the fifth category. We have chosen relatively large programs from each category, so the benchmarks represent “hard instances”. The benchmarks marked by “(neg)” expect the output of model checking to be “No”; “Yes” is expected for the others. Some tools such as TRECS and TravMC can take certain extensions of HORS as input; benchmarks with such extensions were reformulated as a pure HORS in every case, which might result in a longer run-time than on the original. The implementation and benchmarks are available at <http://www-kb.is.s.u-tokyo.ac.jp/~koba/horsat/>.

We ran the experiments on an Acer Aspire TimeLineX 4820T laptop computer with an Intel Core i5 M430 CPU and 4GB of RAM. The operating system was GNU/Linux (Fedora 17) with kernel version 3.6.2-4. TRECS, GTRECS2 and HORSAT were compiled with OCaml version 3.12.1, CSHORe was run with OpenJDK IcedTea version 1.7.0 and TravMC using Mono version 3.0.2. A memout of 2GB and a timeout of 300s was given to each tool for each run. The order and size of the recursion scheme are respectively listed in the O and Sz columns. The T, G, TMC and C columns give the total run-times (in seconds) for the TRECS, GTRECS2, TravMC and CSHORe tools as each tool reports for itself. The HS and HST columns give the run-times as reported by HORSAT and HORSATT.

Overall the benchmark results are favorable to our new algorithms HORSAT and HORSATT. They are the best two tools in terms of the number of time-outs, although the state-of-the-art model checker TRECS is often better in terms of the run-times when it terminates. (The only time-out of HORSAT is for `fibstring`, which is a pathological case where a huge string is concisely expressed by a HORS.) HORSAT outperforms another saturation-based model checker CSHORe except for one instance (`search-e-church`); this confirms the advantage of directly working on HORS without a detour to CPDS.

Benchmark file	Ord	Sz	T	G	TMC	C	HS	HST
jwig-cal_main	2	7627	0.090	0.902	0.081	—	13.137	5.306
specialize_cps_coerce1-c	3	2731	—	—	—	5.145	1.702	0.956
xhtmlf-div-2 (neg)	2	3003	0.327	51.8	—	—	12.392	2.697
xhtmlf-m-church	2	3027	0.331	18.558	—	—	9.282	2.496
fold_fun_list	7	1346	0.724	—	—	4.152	0.180	0.729
fold_right	5	1310	—	—	—	2.996	34.796	7.958
search-e-church (neg)	6	837	0.011	—	0.489	9.547	0.403	0.921
zip	4	2952	—	—	—	19.299	3.501	—
filepath	2	5956	—	—	—	1.059	0.586	6.860
filter-nonzero (neg)	5	482	0.008	0.486	0.206	3.337	0.067	0.147
filter-nonzero-1	5	888	0.272	—	—	11.116	0.223	1.203
map-plusone-2	5	704	1.227	—	20.080	5.518	0.113	0.609
cfa-life2	14	7648	—	—	—	—	2.860	—
cfa-matrix-1	8	2944	—	—	—	—	0.450	5.345
cfa-psdes	7	1819	—	—	—	6.761	0.185	1.410
tak (neg)	8	451	—	—	7.763	—	1.570	0.429
dna	2	411	0.029	0.072	0.467	—	0.126	0.335
g45	4	55	—	2.576	—	—	0.019	0.017
fibstring	4	29	—	0.179	102.583	—	—	—
1	3	35	—	0.010	23.322	0.439	0.002	0.006

■ **Table 2** Comparison of model-checking tools.

6 Related Work

Early model-checking algorithms for HORS [10, 1, 22, 9] were mainly used for showing the decidability of model checking problems, and suffer from the k -EXPTIME worst-case complexity [22] for almost all inputs. To our knowledge, Kobayashi [11] proposed the first practical algorithm for trivial automata model checking, and implemented a model checker TRECS [12]. His algorithm reduces the start symbol S in a finite number of steps, and infers the types of non-terminals by observing how each non-terminal symbol is used in the partial reduction sequence. The inferred type environment is then used as an over-approximation of the fixedpoint of *Shrink* in Section 4. Some other practical algorithms [14, 21] have since been developed. Except GTRECS in a pathological case (the fifth category in the benchmark), however, they failed to show a clear practical advantage over TRECS. Those algorithms are all based on a type-based characterization of the problem, and propagate information *forwards*, starting with the goal to prove that S has type q_I . Broadbent et al. [4, 5] have recently proposed a quite different algorithm for CPDS, which uses *backward* propagation of information.

As already noted in Section 1, our new algorithms HORSAT and HORSAT^T bridge the gap between the two families of model checking algorithms mentioned above. On the one hand, HORSAT and HORSAT^T are strongly related to the type-based algorithms in that they use Kobayashi’s type-based characterization of model checking [13], and the algorithms can be viewed as an (optimized) fixed-point computation for a function on intersection type environments. On the other hand, HORSAT is also related to the saturation algorithm for CPDS, in that both propagate information backwards, starting from the set of error

configurations. Our representation of a set of terms as a type environment is superficially quite different from Broadbent et al.’s stack automata used to represent CPDS configurations (which are variations of alternating automata). Based on Rehof and Urzyczyn’s result [25], however, a type environment can also be regarded as an alternating tree automaton that accepts the set \mathbf{Terms}_Γ of terms well-typed under Γ (see Section C in Appendix). Thus, both approaches essentially represent the set of states reachable to accepting configurations by using (variants of) alternating automata. A more precise connection on this point will be discussed in a companion paper [3], by using a streamlined version of CPDS.

As mentioned in Section 1, the model checking of HORS has recently been applied to automated program analysis and verification [13, 11, 23, 26]. Those applications should benefit from the performance advantage of HORSAT; in fact, we have recently replaced the underlying model checker TRECS with HORSAT in the work on exact flow analysis [26] and observed a speed up by an order of magnitude in several cases.

7 Conclusion

We have presented the first algorithm for model-checking HORS using intersection types that employs a *backward* mode of inference, à la saturation algorithm for CPDS [4]; previous type-based algorithms use forward inference. We have also implemented a prototype model checker and confirmed that it often outperforms previous model checkers for HORS.

This paper lays the foundation for further work on backward mode saturation-like algorithms using types. It is worth mentioning that the set of (*forwards*)-*reachable* terms is irregular (when viewed as a language consisting of abstract syntax trees) and existing forward-mode algorithms must in some sense approximate the set of reachable terms. On the other hand, Rehof and Urzyczyn’s construction [25] applied to the type environment computed by saturation shows that the set of terms *backward-reachable* from error-terms is *regular*. In this respect backward algorithms are arguably more natural.

References

- 1 K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(3), 2007.
- 2 A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *International Conference on Concurrency Theory*, pages 135–150, 1997.
- 3 C. Broadbent and N. Kobayashi. Streamlining collapsible pushdown systems and their model checking, 2013. To be submitted. The title is tentative. The paper will be made available at <http://www-kb.is.s.u-tokyo.ac.jp/~koba/horsat/> as soon as it is submitted.
- 4 C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. A saturation method for collapsible pushdown systems. In *Proceedings of ICALP 2012*, volume 7392 of *LNCS*, pages 165–176. Springer-Verlag, 2012.
- 5 C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. A collapsible approach to verifying higher-order programs. submitted <http://h1900900.stratoserver.net/Web/>, 2013.
- 6 A. Carayol and O. Serre. Collapsible pushdown automata and labeled recursion schemes: Equivalence, safety and effective selection. In *Proceedings of LICS 2012*, pages 165–174. IEEE, 2012.
- 7 A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *Proc. 2nd Int. Workshop on Verification of Infinite State Systems (INFINITY’97)*, *Bologna, Italy, July 11–12, 1997*, volume 9 of *ENTCS*. Elsevier, 1997.

- 8 E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*. Springer-Verlag, 2002.
- 9 M. Hague, A. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *Proceedings of 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 452–461. IEEE Computer Society, 2008.
- 10 T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS 2002*, volume 2303 of *LNCS*, pages 205–222. Springer-Verlag, 2002.
- 11 N. Kobayashi. Model-checking higher-order functions. In *Proceedings of PPDP 2009*, pages 25–36. ACM Press, 2009.
- 12 N. Kobayashi. TRECS: A type-based model checker for recursion schemes. <http://www-kb.is.s.u-tokyo.ac.jp/~koba/treecs/>, 2009.
- 13 N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proc. of POPL*, pages 416–428, 2009.
- 14 N. Kobayashi. A practical linear time algorithm for trivial automata model checking of higher-order recursion schemes. In *Proceedings of FoSSaCS 2011*, volume 6604 of *LNCS*, pages 260–274. Springer-Verlag, 2011.
- 15 N. Kobayashi. GTRECS2: A model checker for recursion schemes based on games and types. A tool available at <http://www-kb.is.s.u-tokyo.ac.jp/~koba/gtreecs2/>, 2012.
- 16 N. Kobayashi, K. Matsuda, A. Shinohara, and K. Yaguchi. Functional programs as compressed data. *Higher-Order and Symbolic Computation*, 2013. To appear. A preliminary version appeared in Proceedings of PEPM12.
- 17 N. Kobayashi and C.-H. L. Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of LICS 2009*, pages 179–188. IEEE Computer Society Press, 2009.
- 18 N. Kobayashi and C.-H. L. Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. *Logical Methods in Computer Science*, 7(4), 2011.
- 19 N. Kobayashi, R. Sato, and H. Unno. Predicate abstraction and CEGAR for higher-order model checking. In *Proc. of PLDI*, pages 222–233, 2011.
- 20 N. Kobayashi, N. Tabuchi, and H. Unno. Higher-order multi-parameter tree transducers and recursion schemes for program verification. In *Proc. of POPL*, pages 495–508, 2010.
- 21 R. P. Neatherway, S. J. Ramsay, and C.-H. L. Ong. A traversal-based algorithm for higher-order model checking. In *ACM SIGPLAN International Conference on Functional Programming (ICFP '12)*, pages 353–364, 2012.
- 22 C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS 2006*, pages 81–90. IEEE Computer Society Press, 2006.
- 23 C.-H. L. Ong and S. Ramsay. Verifying higher-order programs with pattern-matching algebraic data types. In *Proc. of POPL*, pages 587–598, 2011.
- 24 C.-H. L. Ong and T. Tsukada. Two-level game semantics, intersection types, and recursion schemes. In *Proceedings of ICALP 2012*, volume 7392 of *LNCS*, pages 325–336. Springer-Verlag, 2012.
- 25 J. Rehof and P. Urzyczyn. Finite combinatory logic with intersection types. In *Proceedings of TLCA 2011*, volume 6690 of *LNCS*, pages 169–183. Springer, 2011.
- 26 Y. Tobita, T. Tsukada, and N. Kobayashi. Exact flow analysis by higher-order model checking. In *Proceedings of FLOPS 2012*, volume 7294 of *LNCS*, pages 275–289. Springer, 2012.

A

 On Theorem 7

Kobayashi and Ong's result [17] directly imply the following special cases of Theorem 7.

► **Theorem 20.** *Suppose $\mathbf{Tree}(\mathcal{G})$ does not contain \perp . Then, $\mathbf{Tree}(\mathcal{G})$ is accepted by \mathcal{A} in the trivial mode if and only if there is a possibly infinite derivation tree for $\emptyset \vdash_{\mathcal{A}, \mathcal{G}} S : q_I$.*

► **Theorem 21.** *Suppose $\mathbf{Tree}(\mathcal{G})$ does not contain \perp . Then, $\mathbf{Tree}(\mathcal{G})$ is accepted by \mathcal{A} in the co-trivial mode if and only if there is a finite derivation tree for $\emptyset \vdash_{\mathcal{A}, \mathcal{G}} S : q_I$.*

We sketch how to derive Theorem 7 from Theorems 20 and 21. Suppose $\mathbf{Tree}(\mathcal{G})$ may contain \perp . Let \mathcal{G}' be a HORS that is obtained by adding a special terminal *loop* of arity 1, and replacing each rule $F x_1 \cdots x_k \rightarrow t$ of \mathcal{G} with $F x_1 \cdots x_k \rightarrow \text{loop}(t)$. Then, $\mathbf{Tree}(\mathcal{G}')$ does not contain \perp , and $\mathbf{Tree}(\mathcal{G})$ is obtained from $\mathbf{Tree}(\mathcal{G}')$ by removing every infinite sequence *loop* with \perp and removing other occurrences of *loop*. Let \mathcal{A}' be the ATA obtained from \mathcal{A} by adding the transition rule $(q, \text{loop}, \{(1, q)\})$ for every state q . Then, we have:

$$\begin{aligned}
 & \mathbf{Tree}(\mathcal{G}) \text{ is accepted by } \mathcal{A} \text{ in the co-trivial mode} \\
 \Leftrightarrow & \mathbf{Tree}(\mathcal{G}') \text{ is accepted by } \mathcal{A}' \text{ in the co-trivial mode} \\
 \Leftrightarrow & \text{there is a finite derivation for } \emptyset \vdash_{\mathcal{A}', \mathcal{G}'} S : q_I. \\
 \Leftrightarrow & \text{there is a finite derivation for } \emptyset \vdash_{\mathcal{A}, \mathcal{G}} S : q_I.
 \end{aligned}$$

Similarly, we have:

$$\begin{aligned}
 & \mathbf{Tree}(\mathcal{G}) \text{ is accepted by } \mathcal{A} \text{ in the trivial mode} \\
 \Leftrightarrow & \mathbf{Tree}(\mathcal{G}') \text{ is accepted by } \mathcal{A}' \text{ in the trivial mode} \\
 \Leftrightarrow & \text{there is a possibly infinite derivation for } \emptyset \vdash_{\mathcal{A}', \mathcal{G}'} S : q_I. \\
 \Leftrightarrow & \text{there is a possibly infinite derivation for } \emptyset \vdash_{\mathcal{A}, \mathcal{G}} S : q_I.
 \end{aligned}$$

B

 Proofs

B.1 Proofs for Section 3

Proof of Lemma 8

Suppose that $\mathbf{Tree}(\mathcal{G})$ is accepted by \mathcal{A} . Then there exists a finite run-tree R of \mathcal{A} over $\mathbf{Tree}(\mathcal{G})$. Let D be the relevant part of the domain of $\mathbf{Tree}(\mathcal{G})$, i.e., $D = \{\pi \mid (\pi, a) \in \text{codom}(R)\}$. By the definition of $\mathbf{Tree}(\mathcal{G}) (= \bigsqcup \{t^\perp \mid S \rightarrow_{\mathcal{G}}^* t\})$, there exists t such that $S \rightarrow_{\mathcal{G}}^* t$ and $U \subseteq \text{dom}(t^\perp)$ with $\mathbf{Tree}(\mathcal{G})(\pi) = t^\perp(\pi)$ for every $\pi \in U$. Then, R is also a run-tree of \mathcal{A}^\perp over t^\perp . Thus, t satisfies the required property.

Conversely, suppose that $S \rightarrow_{\mathcal{G}}^* t$ and t^\perp is accepted by \mathcal{A}^\perp . Since \mathcal{A}^\perp has no transition rule on \perp , a run-tree of \mathcal{A}^\perp over t^\perp must also be a run-tree of \mathcal{A} over $\mathbf{Tree}(\mathcal{G})$. Thus, $\mathbf{Tree}(\mathcal{G})$ is accepted by \mathcal{A} . \square

Proof of Lemma 9

Let $\mathcal{A} = (\Sigma, Q, \Delta, q_I)$. We show that $\emptyset \vdash_{\mathcal{A}}^- t : q$ if and only if t^\perp is accepted by (Σ, Q, Δ, q) , by induction on the structure of t . If $\emptyset \vdash_{\mathcal{A}}^- t : q$, then t must be of the form $a t_1 \cdots t_k$ and:

$$\begin{aligned}
 & (q, a, \{(i, q_j) \mid i \in \{1, \dots, k\}, j \in I_i\}) \in \Delta \\
 & \emptyset \vdash_{\mathcal{A}}^- t_i : q_j \text{ for each } i \in \{1, \dots, k\}, j \in I_i
 \end{aligned}$$

By the induction hypothesis, t_i^\perp is accepted by (Σ, Q, Δ, q_j) for each $i \in \{1, \dots, k\}, j \in I_i$. Thus, $t^\perp = a t_1^\perp \dots t_k^\perp$ is accepted by (Σ, Q, Δ, q) .

Conversely, suppose that t^\perp is accepted by (Σ, Q, Δ, q) . Then t must be of the form $a t_1 \dots t_k$. So, we have: $(q, a, \{(i, q_j) \mid i \in \{1, \dots, k\}, j \in I_i\}) \in \Delta$ for some I_1, \dots, I_k and t_i^\perp is accepted by (Σ, Q, Δ, q_j) for each $i \in \{1, \dots, k\}, j \in I_i$. By the induction hypothesis, $\emptyset \vdash_{\mathcal{A}}^- t_i : q_j$ for each $i \in \{1, \dots, k\}, j \in I_i$. Thus, we have $\emptyset \vdash_{\mathcal{A}}^- t : q$ as required. \square

► **Lemma 22.** *Let s and t be applicative terms. If $\Gamma \vdash_{\mathcal{A}}^- [t/x]s : \tau$, then there exist a (possibly empty) set I and $\{\tau_i \mid i \in I\}$ (where ℓ may be 0) such that $\Gamma \cup \{x : \tau_i \mid i \in I\} \vdash_{\mathcal{A}}^- s : \tau$ and $\Gamma \vdash_{\mathcal{A}}^- t : \tau_i$ for each $i \in I$.*

Proof. The proof proceeds by induction on the structure of s .

- Case $s = a$ or $s = y \neq x$: The required result holds for $I = \emptyset$.
- Case $s = x$: The result holds for $I = \{1\}$ and $\tau_1 = \tau$.
- Case $s = s_1 s_2$: In this case, we have

$$\begin{aligned} \Gamma \vdash_{\mathcal{A}}^- [t/x]s_1 : \bigwedge_{j \in J} \tau'_j \rightarrow \tau \\ \Gamma \vdash_{\mathcal{A}}^- [t/x]s_2 : \tau'_j \text{ for each } j \in J \end{aligned}$$

By the induction hypothesis, we have:

$$\begin{aligned} \Gamma \cup \{x : \tau_i \mid i \in I_0\} \vdash_{\mathcal{A}}^- s_1 : \bigwedge_{j \in J} \tau'_j \rightarrow \tau \\ \Gamma \cup \{x : \tau_i \mid i \in I_j\} \vdash_{\mathcal{A}}^- s_2 : \tau'_j \text{ for each } j \in J \\ \Gamma \vdash_{\mathcal{A}}^- t : \tau_i \text{ for each } i \in I \cup \bigcup_{j \in J} I_j. \end{aligned}$$

Let $I = I_0 \cup \bigcup_{j \in J} I_j$. Then, we have $\Gamma \cup \{x : \tau_i \mid i \in I\} \vdash_{\mathcal{A}}^- s : \tau$ as required.

\square

Proof of Lemma 11

This follows by repeated applications of Lemma 22. \square

► **Lemma 23.** *Let $\mathcal{G}^{(m)}$ and $t^{(m)}$ as defined in the proof of Theorem 14. If $\mathcal{F}^m(\emptyset) \cup \Gamma \vdash_{\mathcal{A}}^- t : \tau$ then $\Gamma \vdash_{\mathcal{A}^\perp, \mathcal{G}^{(m)}} t^{(m)} : \tau$.*

Proof. This follows by double induction on m and the derivation of $\mathcal{F}^m(\emptyset) \cup \Gamma \vdash_{\mathcal{A}}^- t : \tau$, with case analysis on the last rule used for deriving $\mathcal{F}^m(\emptyset) \cup \Gamma \vdash_{\mathcal{A}}^- t : \tau$. Since the other cases are trivial, we show only the case where $t = F$, with $F : \tau \notin \Gamma$. In this case, $F : \tau \in \mathcal{F}^m(\emptyset)$ holds for some $m \geq 1$. By the definition of \mathcal{F} and Lemmas 12, we have

$$\begin{aligned} \tau &= \Gamma_V(x_1) \rightarrow \dots \rightarrow \Gamma_V(x_k) \rightarrow q \\ \mathcal{R}(F) &= \lambda x_1. \dots \lambda x_\ell. s \\ \mathcal{F}^{m-1}(\emptyset) \cup \Gamma_V &\vdash_{\mathcal{A}}^- t : q \end{aligned}$$

By the induction hypothesis, we have $\Gamma_V \vdash_{\mathcal{A}^\perp, \mathcal{G}} t^{(m-1)} : q$. By using T-NT, we obtain $\Gamma \vdash_{\mathcal{A}^\perp, \mathcal{G}} F^{(m)} : \tau$ as required.

\square

► **Lemma 24** (substitution). *If $\Gamma \cup \{x : \tau_i \mid i \in I\} \vdash_{\mathcal{A}, \mathcal{G}} s : q$ and $\Gamma \vdash_{\mathcal{A}, \mathcal{G}} t : \tau_i$ for every $i \in I$, with $x \notin \text{dom}(\Gamma)$, then $\Gamma \vdash_{\mathcal{A}, \mathcal{G}} [t/x]s : q$.*

Proof. This follows by straightforward induction on the structure of s . \square

► **Lemma 25** (type preservation). *If $\emptyset \vdash_{\mathcal{A}, \mathcal{G}} t : q$ and $t \rightarrow_{\mathcal{G}} t'$, then $\emptyset \vdash_{\mathcal{A}, \mathcal{G}} t' : q$.*

Proof. This follows by induction on the derivation of $t \rightarrow_{\mathcal{G}} t'$. Since the induction step is trivial, we show only the case where $t = F t_1 \cdots t_k$ and $t' = [t_1/x_1, \dots, t_k/x_k]s$ with $\mathcal{R}(F) = \lambda x_1 \cdots \lambda x_k.s$. In this case, we have:

$$\begin{aligned} & \{x_i : \tau_j \mid i \in \{1, \dots, k\}, j \in I_i\} \vdash_{\mathcal{A}}^- \mathcal{G}s : q \\ & \emptyset \vdash_{\mathcal{A}}^- \mathcal{G}t_i : \tau_j \text{ for each } i \in \{1, \dots, k\}, j \in I_i \end{aligned}$$

By the substitution lemma (Lemma 24), we have $\emptyset \vdash_{\mathcal{A}}^- \mathcal{G}[t_1/x_1, \dots, t_k/x_k]s : q$ as required. \square

B.2 Proofs for Section 4

Proof of Theorem 16

This follows immediately from Theorem 7. Note that there is a fixedpoint Γ of $\text{Shrink}_{\mathcal{G}}$ such that $S : q_I \in \Gamma$ if and only if there is a possibly infinite derivation tree for $\emptyset \vdash_{\mathcal{A}} S : q_I$. To see this, note that if there is a possibly infinite derivation tree for $\emptyset \vdash_{\mathcal{A}} S : q_I$, then the set $\{F : \tau \mid \emptyset \vdash_{\mathcal{A}} F : \tau \text{ occurs in the derivation tree}\}$ is a fixed-point of Shrink , and conversely, one can construct an infinite derivation tree for $\emptyset \vdash_{\mathcal{A}} S : q_I$ from a fixedpoint of Shrink . \square

The rest of this subsection is devoted to the proof of Theorem 18.

► **Lemma 26.** *Suppose $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}', S)$ and $\mathcal{R} \subseteq \mathcal{R}'$. If $\Gamma \vdash (\mathcal{G}, t) : q$ and $s \rightarrow_{\mathcal{R}} t$, then $\mathcal{F}_{\mathcal{G}, \mathcal{A}, \mathcal{R}}(\Gamma) \vdash (\mathcal{G}, s) : q$.*

Proof. By Lemma 13, we have $\mathcal{F}_{\mathcal{G}, \mathcal{A}, \mathcal{R}}(\Gamma) \vdash s : q$. Thus, it remains to show that $\text{Shrink}(\mathcal{F}_{\mathcal{G}, \mathcal{A}, \mathcal{R}}(\Gamma)) = \mathcal{F}_{\mathcal{G}, \mathcal{A}, \mathcal{R}}(\Gamma)$. Suppose $F : \sigma_1 \rightarrow \cdots \rightarrow \sigma_k \rightarrow q \in \mathcal{F}_{\mathcal{G}, \mathcal{A}, \mathcal{R}}(\Gamma) \setminus \Gamma$ and $\mathcal{R}(F) = \lambda x_1 \cdots \lambda x_k.t$. By the definition of \mathcal{F} and Lemma 12, we have $\Gamma, x_1 : \sigma_1, \dots, x_k : \sigma_k \vdash_{\mathcal{A}} t : q$. Thus, we have $\mathcal{F}_{\mathcal{G}, \mathcal{A}, \mathcal{R}}(\Gamma), x_1 : \sigma_1, \dots, x_k : \sigma_k \vdash_{\mathcal{A}} t : q$ as required. \square

► **Lemma 27.** *If $\text{Tree}(\mathcal{G})$ is accepted by \mathcal{A}^\top in the trivial mode, then $S : q_I \in \bigcap_{j \in \omega} \text{Shrink}^j(\bigcup_{i \in \omega} \mathcal{F}^i(\Gamma_0))$.*

Proof. Let \mathcal{G} be $(\Sigma, \mathcal{N}, \mathcal{R}, S)$, where $\text{dom}(\mathcal{N}) = \{F_1, \dots, F_n\}$ and $S = F_1$. Let Γ_{max} be the largest type environment that conforms to \mathcal{N} , i.e., $\{F : \tau \mid \tau :: \mathcal{N}(F)\}$. Let m be the number of type bindings $|\Gamma_{\text{max}}|$. Let $\mathcal{G}^{(m)}$ be the HORS $(\Sigma \cup \{\perp \mapsto 0\}, \mathcal{N}^{(m)}, \mathcal{R}^{(m)}, F_1^{(m)})$ constructed as an approximation of \mathcal{G} in the proof of Theorem 14. Let $\mathcal{R}'^{(m)}$ be the following subset of $\mathcal{R}^{(m)}$:

$$\mathcal{R}'^{(m)} = \{F_i^{(j)} \mapsto [F_1^{(j-1)} / F_1, \dots, F_n^{(j-1)} / F_n] \mathcal{R}(F_i)\}$$

By the strong normalization of the simply-typed λ -calculus, $F_1^{(m)} \rightarrow_{\mathcal{R}'^{(m)}}^* t \not\rightarrow_{\mathcal{R}'^{(m)}}$ for some t , and there is a corresponding reduction sequence $F_1 \rightarrow_{\mathcal{G}}^* t'$ of \mathcal{G} such that $t'^\perp = t^\perp$. Thus, t^\perp is accepted by \mathcal{A}^\top . By Lemma 9, we have $\emptyset \vdash_{\mathcal{A}}^- t^\perp : q_I$. Thus, we have $\Gamma_0^{(0)} \vdash_{\mathcal{A}}^- t : q_I$, where $\Gamma_0^{(0)} = \{F^{(0)} : \tau \mid F : \tau \in \Gamma_0\}$. By Lemma 26, we have $\Gamma' \vdash (\mathcal{G}^{(m)}, F_1^{(m)}) : q_I$ for $\Gamma' = \bigcup_{i \in \omega} \mathcal{F}_{\mathcal{G}^{(m)}, \mathcal{A}^\top, \mathcal{R}'^{(m)}}^i(\Gamma_0^{(0)})$. Let Γ_j' be: $\{F_i : \tau \mid F_i^{(j')} : \tau \in \Gamma' \wedge j' \geq j\}$. By the condition $\Gamma' = \bigcup_{i \in \omega} \mathcal{F}_{\mathcal{G}^{(m)}, \mathcal{A}^\top, \mathcal{R}'^{(m)}}^i(\Gamma_0^{(0)})$, we have $\Gamma'_0 \subseteq \bigcup_{i \in \omega} \mathcal{F}_{\mathcal{G}, \mathcal{A}, \mathcal{R}}^i(\Gamma_0)$. Furthermore, Γ_j' forms a monotonically decreasing sequence: $\Gamma'_0 \supseteq \Gamma'_1 \supseteq \cdots \supseteq \Gamma'_m$. Since $|\Gamma'_0| \leq m$ and $|\Gamma'_m| \geq |\{F_1^{(m)} : q_I\}| = 1$, there exists $k (< m)$ such that $\Gamma'_k = \Gamma'_{k+1}$. By the condition $\Gamma' \vdash (\mathcal{G}^{(m)}, F_1^{(m)}) : q_I$, we have $\vdash \mathcal{G}^{(m)} : \Gamma'$, which implies:

$$\Gamma \downarrow_{\{F_1^{(k)}, \dots, F_n^{(k)}\}} x_1 : \sigma_1, \dots, x_k : \sigma_k \vdash_{\mathcal{A}}^- [F_1^{(k)} / F_1, \dots, F_n^{(k)} / F_n] t : \tau$$

for every $F^{(k+1)} : \tau \in \Gamma'$ and $\mathcal{R}(F) = \lambda x_1. \dots \lambda x_\ell. t$. Here, $\Gamma \downarrow_S$ denotes $\{F : \tau \in \Gamma \mid F \in S\}$. Since $\Gamma'_k = \Gamma'_{k+1}$, the above condition implies: $\Gamma'_k, x_1 : \sigma_1, \dots, x_\ell : \sigma_\ell \vdash_{\mathcal{A}}^- t : \tau$ for every $F : \tau \in \Gamma'_k$ and $\mathcal{R}(F) = \lambda x_1. \dots \lambda x_\ell. t$, i.e., $\text{Shrink}_{\mathcal{G}}(\Gamma'_k) = \Gamma'_k$. Therefore, we have:

$$S : q_I \in \Gamma'_m \subseteq \Gamma'_k = \bigcap_{j \in \omega} \text{Shrink}^j(\Gamma'_k) \subseteq \bigcap_{j \in \omega} \text{Shrink}^j(\Gamma'_0) \subseteq \bigcap_{j \in \omega} \text{Shrink}^j(\bigcup_{i \in \omega} \mathcal{F}^i(\Gamma_0)).$$

□

Proof of Theorem 18

The “only if” direction follows immediately from Theorem 16. The “if” direction follows from Lemma 27. □

C An algorithm to check the inhabitation condition

C.1 The construction

Computing $\mathcal{F}(\Gamma)$ requires a procedure to check $\Gamma \stackrel{?}{\in} \text{Inhabited}(\mathcal{K}, \Gamma_N)$. Following Rehof and Urzyczyn [25], we can reduce it the emptiness problem for an alternating tree automaton. This also serves to demonstrate how our algorithm can be interpreted as manipulating alternating tree automata (the stack automata manipulated by the CPDS algorithm can also be seen as a kind of alternating tree automaton). Given Γ_N , define an ATA $\mathcal{A}' = (\Sigma', Q', \Delta', q'_I)$ by:

$$\begin{aligned} \Sigma' &= \{a \mapsto 0 \mid a \in \text{dom}(\Sigma)\} \cup \{F \mapsto 0 \mid F \in \text{dom}(\mathcal{N})\} \\ &\quad \cup \{\textcircled{a} \mapsto 2\} \\ Q' &= \{(\tau, \kappa) \mid \tau :: \kappa, \text{ and } \tau \text{ occurs (as a sub-expression)} \\ &\quad \text{in } \Gamma_N \text{ or a type of a constant}\} \\ \Delta' &= \{((\tau, \kappa), a, \emptyset) \mid \emptyset \vdash_{\mathcal{A}}^- a : \tau \text{ and } \kappa = \underbrace{\circ \rightarrow \dots \rightarrow \circ}_{\Sigma(a)} \rightarrow \circ\} \\ &\quad \cup \{((\tau, \mathcal{N}(F)), F, \emptyset) \mid F : \tau \in \Gamma_N\} \\ &\quad \cup \{((\tau, \kappa), \textcircled{a}), \\ &\quad \quad \{(1, (\bigwedge_{i \in I} \tau_i \rightarrow \tau, \kappa' \rightarrow \kappa))\} \cup \{(2, (\tau_i, \kappa')) \mid i \in I\} \\ &\quad \quad \mid (\bigwedge_{i \in I} \tau_i \rightarrow \tau, \kappa' \rightarrow \kappa) \in Q\} \\ q'_I &= (q_I, \circ) \end{aligned}$$

By the construction, t^\sharp is accepted by $(\Sigma', Q', \Delta', (\tau, \kappa))$ if and only if $t \in \mathbf{ATerms}_{\mathcal{N}, \Sigma, \kappa}$ and $\Gamma_N \vdash_{\mathcal{A}}^- t : \tau$, where t^\sharp is the tree representation of term t , with an application $t_1 t_2$ is expressed by a tree:



Thus, to check $\Gamma \in \text{Inhabited}(\mathcal{K}, \Gamma_N)$, it suffices to check that for every $x \in \text{dom}(\Gamma)$, the intersection:

$$\bigcap_{x : \tau \in \Gamma} \mathcal{L}(\Sigma', Q', \Delta', (\tau, \mathcal{K}(x)))$$

is non-empty, where $\mathcal{L}(\mathcal{A})$ denotes the set of trees accepted by \mathcal{A} (in the co-trivial mode).

During the computation of $\mathcal{F}^m(\emptyset)$ (where $m = 1, 2, \dots$), the automaton \mathcal{A}' for $\Gamma_N = \mathcal{F}^m(\emptyset)$ can be constructed incrementally.

C.2 A remark on the complexity of the emptiness check

For the purposes of HORS model-checking, we are interested in the complexity of our algorithms when the arity of sorts (and hence types) and the sizes of the property automaton (and in particular Q) are bounded. This is because these tend to be small compared to the size of the HORS itself.

In this subsection we explain why the inhabitation check used in the HORSAT algorithm can be regarded (under the assumptions above) as a constant-time operation when considering the theoretical worst-case complexity. However, this is not intended as a practical algorithm (the constant is still generally very large) but just to demonstrate that theoretically inhabitation checking does not negate the fixed parameter tractability of the algorithm.

First observe that, for the purposes of inhabitation checking, it is only necessary to have a single non-terminal bound to any given type. More precisely, let Γ_N be a set of type bindings for non-terminals, and \mathcal{K} a kind-environment such that $\Gamma_N :: \mathcal{K}$. Suppose for each sort κ and each set $T \subseteq \mathbf{ITypes}_\kappa$ we have a fresh non-terminal $F_{\kappa,T}$. Define:

$$\begin{aligned} U(\mathcal{K}) &:= \{F_{\kappa,T} : \kappa \mid \exists G. \forall \tau \in T. G : \tau \in \Gamma_N \text{ with } \tau :: \kappa\} \\ &\quad \cup \{F_{\kappa,T} : \kappa \mid \exists a \in \text{dom}(\Sigma). \forall \tau \in T. \emptyset \vdash_{\mathcal{A},\mathcal{G}} a : \tau \\ &\quad \quad \text{with } \tau :: \kappa\} \\ U(\Gamma_N) &:= \{F_{\kappa,T} : \tau \mid F_{\kappa,T} \in U(\mathcal{K}) \text{ and } \tau \in T\} \end{aligned}$$

It should be clear that

$$\text{Inhabited}(\mathcal{K}, \Gamma_N) = \text{Inhabited}(U(\mathcal{K}), U(\Gamma_N))$$

After all, given a term t witnessing $\{x : \tau \mid \tau \in T\} \in \text{Inhabited}(\mathcal{K}, \Gamma_N)$ we can construct a term t' witnessing $\{x : \tau \mid \tau \in T\} \in \text{Inhabited}(U(\mathcal{K}), U(\Gamma_N))$ by replacing each non-terminal G occurring in t with $F_{\kappa,R}$ where R is the maximal set satisfying $\{G : \tau \mid \tau \in R\} \subseteq \Gamma$ and $R :: \kappa$, and replacing every terminal a with $F_{\kappa,R}$ where R is the maximal set satisfying $\emptyset \vdash_{\mathcal{A},\mathcal{G}} a : \tau$ for every $\tau \in R$, with $R :: \kappa$. Conversely, given a term t' witnessing $\{x : \tau \mid \tau \in T\} \in \text{Inhabited}(U(\mathcal{K}), U(\Gamma_N))$, we can construct a term t witnessing $\{x : \tau \mid \tau \in T\} \in \text{Inhabited}(\mathcal{K}, \Gamma_N)$ by replacing every occurrence $F_{\kappa,R}$ in t' by either a G such that $G : \tau \in \Gamma$ for every $\tau \in R$ or $a \in \text{dom}(\Sigma)$ such that $\emptyset \vdash_{\mathcal{A},\mathcal{G}} a : \tau$ for every $\tau \in R$ (one of which must exist by definition).

Note that the size of $U(\Gamma_N)$ is bounded by a constant, namely the number of different well-sorted sets of intersection types. Thus in particular the emptiness of $\text{Inhabited}(U(\mathcal{K}), U(\Gamma_N))$ can be determined in constant time (for example by the algorithm sketched in the previous section whose run-time depends only on $|U(\Gamma_N)|$ and the number of intersection types). So assuming that $U(\Gamma_N)$ has already been computed, $\text{Inhabited}(\mathcal{K}, \Gamma_N)$ can be computed in constant time.

We now observe how $U(\Gamma_N)$ can be grown incrementally together with Γ_N as the algorithm progresses (rather than being recomputed on each inhabitation check). Since $\Gamma_N := \emptyset$ at the start of the algorithm, $U(\Gamma_N)$ is correspondingly initialized to $U(\emptyset)$.

As an aid the HORSAT algorithm could maintain a table H with $\text{dom}(\mathcal{N})$ as keys where:

$$H(G) = \{P \subseteq \mathbf{ITypes}_\kappa \mid G : \kappa \in \mathcal{K} \text{ and } \forall \tau \in P. G : \tau \in \Gamma\}$$

This table grows as Γ_N is grown with the progression of the HORSAT algorithm. Whenever a new set P' is added to $H(G)$ for some $G \in \text{dom}(\mathcal{N}) \cup \text{dom}(\Sigma)$, P' is also added to $U(\Gamma_N)$.

Whenever a new type binding for a non-terminal $G : \tau$ is added to Γ_N , this must be processed against every member P of $H(G)$ with $P \cup \{\tau\}$ then being added to $H(G)$.

Since the number of intersection types is fixed (and so the number of sets of intersection types is fixed) this must involve only constantly many comparisons. Thus assuming that $H(G)$ can be looked up in constant time, the overhead to HORSAT for maintaining $H(G)$ would also only be constant for each addition to Γ_N (and it is only upon each addition to Γ_N that the inhabitation check is performed).

D More about Implementation and Experiments

This section reports more details about the implementation and discusses experimental results.

D.1 Implementation

We first justify the optimization of \mathcal{F} mentioned in Section 5, where $Inhabited(\mathcal{K}, \Gamma_N)$ is replaced with:

$$\{\Gamma \mid \Gamma :: \mathcal{K} \text{ and } \forall x \in \text{dom}(\Gamma). \exists s \in \mathbf{ATerms}_{\mathcal{N}, \Sigma, \mathcal{K}(x)} \cap \mathbf{Flow}(x). \forall x : \tau \in \Gamma. \Gamma_N \vdash_{\mathcal{A}} s : \tau\}.$$

Let \mathcal{F}_{opt} be the optimized version of \mathcal{F} , and write **Reachable** for $\{t \mid S \longrightarrow_{\mathcal{G}}^* t\}$. Then, \mathcal{F}_{opt} satisfies:

$$\{t \in \mathbf{ATerms}_{\mathcal{N}, \Sigma, \circ} \cap \mathbf{Reachable} \mid \exists t'. t \longrightarrow_{\mathcal{G}} t' \in \mathbf{Terms}_{\Gamma}\} \subseteq \mathbf{Terms}_{\mathcal{F}_{opt}(\Gamma)}$$

instead of condition (I) in Section 4. From this, we obtain the following theorem (cf. Theorem 14).

► **Theorem 28.** $S : q_I \in \bigcup_{i \in \omega} \mathcal{F}_{opt}^i(\emptyset)$ if and only if **Tree**(\mathcal{G}) is accepted by \mathcal{A}^\perp in the co-trivial mode.

Proof. The “only if” direction follows from Theorem 14 and the fact that $\mathcal{F}_{opt}(X) \subseteq \mathcal{F}(X)$. To show the “if” direction, suppose that **Tree**(\mathcal{G}) is accepted by \mathcal{A}^\perp in the co-trivial mode. By Lemmas 8 and 9, there exists t_n, t_{n-1}, \dots, t_0 such that

$$S = t_n \longrightarrow_{\mathcal{G}} t_{n-1} \longrightarrow_{\mathcal{G}} \dots \longrightarrow_{\mathcal{G}} t_0$$

and $\emptyset \vdash_{\mathcal{A}} t_0^\perp : q_I$. By the induction on i and the property of \mathcal{F}_{opt} above, we have $t_i \in \mathbf{Terms}_{\mathcal{F}_{opt}^i(\emptyset)}$. Thus, we have $S \in \mathbf{Terms}_{\mathcal{F}_{opt}^n(\emptyset)} \subseteq \mathbf{Terms}_{\bigcup_{i \in \omega} \mathcal{F}_{opt}^i(\emptyset)}$, i.e., $S : q_I \in \bigcup_{i \in \omega} \mathcal{F}_{opt}^i(\emptyset)$. \square

Next, we discuss optimizations specific to the trivial model checking algorithm HORSAT^T. The naive implementation of the HORSAT^T algorithm in Section 4 suffers from the explosion of the size of type environments, due to the initial type environment Γ_0 . Based on the proof of Theorem 18, we actually replace the rules $\mathcal{R} = \{F_1 \mapsto \lambda \tilde{x}_1. t_1, \dots, F_\ell \mapsto \lambda \tilde{x}_\ell. t_\ell\}$ of a given HORS with the following “unfolded” rules (where $m = 2$ in the actual implementation):

$$\begin{aligned} & \{F_i^{(m)} \mapsto \lambda \tilde{x}_1. \mathbf{br} ([F_1^{(m)} / F_1, \dots, F_\ell^{(m)} / F_\ell] t_i) \mid ([F_1^{(m-1)} / F_1, \dots, F_\ell^{(m-1)} / F_\ell] t_i) \mid i \in \{1, \dots, \ell\}\} \\ & \cup \{F_i^{(j)} \mapsto \lambda \tilde{x}_1. ([F_1^{(j-1)} / F_1, \dots, F_\ell^{(j-1)} / F_\ell] t_i) \mid i \in \{1, \dots, \ell\}, j \in \{1, \dots, m-1\}\} \end{aligned}$$

and replace Γ_0 with the set:

$$\Gamma_0 = \{F_i^{(0)} : \underbrace{\top \rightarrow \dots \rightarrow \top}_{\text{ar}(\mathcal{N}(F_i))} \rightarrow q \mid i \in \{1, \dots, \ell\}\}.$$

Note that there are no rules for $F_i^{(0)}$'s, and the initial type environment Γ_0 contains only type bindings on $F_i^{(0)}$'s.

D.2 Discussions on Experiments

In this subsection, we provide some rationals for the experimental results. Some familiarity with the previous model checking algorithms is required for understanding the comments below.

- TRECS/TravMC vs HORSAT/HORSAT^T

The better overall results of HORSAT and HORSAT^T can be largely explained by their fixed-parameter polynomial time complexity (in the size of \mathcal{G} , where the fixed parameters are the size of the largest sort of non-terminals in \mathcal{G} and the number of the states of \mathcal{A}). Neither TRECS nor TravMC satisfies this property, which means that some redundant computation occurs during model checking (or intersection type inference). In the case of TRECS [11], the redundant computation occurs when it collects type information by partially reducing the start symbol S . If there are two applications $F t_1$ and $F t_2$, both terms are reduced to check how the actual arguments t_1 and t_2 are used, and to construct the types of F . Thus, the types of F constructed from $F t_1$ and $F t_2$ may end up to be the same, in which case, one of the reductions is redundant. HORSAT can avoid this redundancy thanks to the backward propagation. With the forward propagation of TRECS, it is difficult to predict which reduction is redundant, and TRECS applies some heuristics to schedule the reductions. According to our previous experiments with TRECS and the new experiments in the present paper, the heuristic scheduling of TRECS works fairly well for small examples, but does not scale well to larger examples like those in our benchmark suite. The function *ElimTE* of the TRECS algorithm [11] also makes TRECS fail to scale to HORS of large orders (like the benchmark programs in the fourth category).

- GTRECS2 vs HORSAT/HORSAT^T

The GTRECS2 algorithm also satisfies the fixed-parameter polynomial time complexity, but as already mentioned in Remark 4, GTRECS2 (and its predecessor GTRECS [14]) suffers from the explosion of the number of types. GTRECS2 uses a game-semantic interpretation of types, where each type represents a possible state of a play of a term. For example, $\top \rightarrow q_0$ represents the state where the environment has requested a tree of type q_0 , and $q_1 \rightarrow q_0$ represents the state where the function requests the argument of type q_1 in response. GTRECS2 (and GTRECS) collects those types and supplies them to the *Shrink* function in Section 4, instead of $\mathcal{F}^i(\Gamma_0)$ of HORSAT^T. The main problem of GTRECS2 is that it needs to consider all the types corresponding to intermediate states of a possible play. As mentioned in Remark 4 if the actual type of a function is $q_1 \rightarrow \dots \rightarrow q_n \rightarrow q$, the types corresponding to intermediate states may include $\sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow q$ for every $\sigma_i \in \{\top, q_i\}$. This may cause k -fold exponential blow-up for an order- k function. According to our previous experience with GTRECS2, it quickly slows down with the increase of the order of HORS.

For **fibstring**, GTRECS2 is better than HORSAT and HORSAT^T. This appears to be due to HORSAT's imprecise approximation of forward flow information, which makes HORSAT propagate irrelevant type information, causing the explosion of the size of the type environment. This suggests that it may be better to use a more precise flow analysis to compute an overapproximation of **Flow**(x) in HORSAT.

- CSHORe vs HORSAT

We think that the better performance of HORSAT comes from the advantage of directly working with HORS. Although HORS and CPDS are equi-expressive (as tree generators), CPDS can in general exhibit behaviors that have no natural interpretation in terms of HORS evaluation and are in particular unnecessary for translating HORS to CPDS. Since

the CSHORe algorithm is designed for arbitrary CPDS both its forward flow analysis and saturation phase must track enough information to soundly handle all possible CPDS behaviours, which constitutes extra work. It is also an advantage that by directly working with HORS, we can use a standard, well-established flow analysis algorithm (OCFA, in the case of our implementation) to obtain forward information.

For one instance `fold_right`, CSHORe outperforms HORSAT. For this case, CSHORe's forward flow analysis is precise enough to get the result without performing saturation. This also suggests that it may be better to use a more precise flow analysis for HORSAT.

- HORSAT vs HORSATT

The winner is case-by-case, and there is no clear superiority between the two, except that HORSATT is better in the first category, and HORSAT is in the third and fourth categories. There are a few significant differences that may explain the results.

- As a point that favors HORSAT, the HORSAT algorithm starts saturation with a smaller initial environment (\emptyset , instead of Γ_0), and it does not need to apply the *Shrink* function.
- As another point that favors HORSAT, as soon as a given co-trivial property is found to be satisfied (i.e., in the “neg” instances in the benchmark), HORSAT can terminate (in the “then” part on Line 2 in Figure 2).
- As a point that favors HORSATT, as soon as a given trivial property is satisfied, HORSATT can terminate (by the “then” part on Line 4 in Figure 3).
- The automata (hence the types of constants used for saturation) are different: HORSATT uses a trivial automaton, while HORSAT uses its complement. (In all the benchmark programs, properties are expressed by using trivial automata, and HORSAT constructs its complement internally.) This difference affects the size of the saturated type environment, analogously to the difference between the sizes of Boolean formulas in DNF and CNF.

The experiments in Section 5 seem to be mainly affected by the first and fourth points above. The second and third points did not always bring advantages. In fact, HORSATT outperforms in two of the four negative instances, contrary to the second point above. Turning off the check $S:q_I \in \Gamma$ in the algorithm in Figure 2 did not affect the performance in those two instances. Appropriate scheduling of backward propagation might however make the second and third points more significant.