Saturation-Based Model Checking of Higher-Order Recursion Schemes

Christopher Broadbent[1] **and Naoki Kobayashi**[2]

1 **The Technische Universität München**
broadben@in.tum.de
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 `CSHORe`, 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

Digital Object Identifier 10.4230/LIPIcs.CSL.2013.129

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

Computer Science Logic 2013 (CSL'13).
Editor: Simona Ronchi Della Rocca; pp. 129–148

Leibniz International Proceedings in Informatics
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
HorSatT	HORS	trivial	types/automata	backward

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). They use intersection types [13, 17] to finitely represent infinite states. 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 HorSatT 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 HorSatT 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 HorSatT, 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 $dom(f)$ and $codom(f)$ 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 $\{\pi\} \cup \{\pi i \mid 1 \le i \le k\} \subseteq D$. For a set S 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 $dom(\Sigma)$-labeled tree such that for every $\pi \in dom(T)$, $\{k \mid \pi k \in dom(T)\} = \{k \mid 1 \le k \le \Sigma(T(\pi))\}$.

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

$$\kappa \text{ (sorts)} \ ::= \mathtt{o} \mid \kappa_1 \to \kappa_2$$

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

$$\mathtt{ord}(\mathtt{o}) = 0 \qquad \mathtt{ord}(\kappa_1 \to \kappa_2) = max(\mathtt{ord}(\kappa_1)+1, \mathtt{ord}(\kappa_2))$$
$$\mathtt{ar}(\mathtt{o}) = 0 \qquad \mathtt{ar}(\kappa_1 \to \kappa_2) = \mathtt{ar}(\kappa_2)+1$$

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) $a \in \mathbf{ATerms}_{\mathcal{K},\Sigma,\underbrace{\mathtt{o}\to\cdots\to\mathtt{o}}_{\Sigma(a)}\to\mathtt{o}}$, (ii) $x \in \mathbf{ATerms}_{\mathcal{K},\Sigma,\kappa}$ if $\mathcal{K}(x) = \kappa$, and (iii) $t_1 t_2 \in \mathbf{ATerms}_{\mathcal{K},\Sigma,\kappa}$ if $t_1 \in \mathbf{ATerms}_{\mathcal{K},\Sigma,\kappa'\to\kappa}$ and $t_2 \in \mathbf{ATerms}_{\mathcal{K},\Sigma,\kappa'}$ 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 $dom(\Sigma)$ are called *terminals*.
2. $\mathcal{N}$ is a map from a finite set of symbols called *non-terminals* to sorts. $dom(\Sigma)$ and $dom(\mathcal{N})$ must be disjoint.

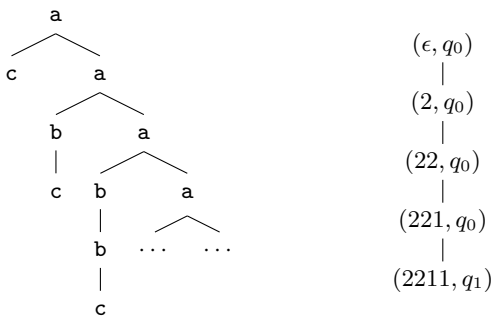

Figure 1 $\mathbf{Tree}(\mathcal{G}_1)$ (left) and a run-tree of $\mathcal{A}_1$ over $\mathbf{Tree}(\mathcal{G}_1)$ (right).

3. $\mathcal{R}$ is a map from the set of non-terminals (i.e. $dom(\mathcal{N})$) to terms of the form $\lambda x_1.\cdots\lambda x_\ell.t$, where t is an applicative term. For each $F \in dom(\mathcal{N})$, if $\mathcal{R}(F) = \lambda x_1.\cdots\lambda x_\ell.t$, then $\mathcal{N}(F)$ must be of the form $\kappa_1 \to \cdots \to \kappa_\ell \to \mathtt{o}$, and $t \in \mathbf{ATerms}_{\mathcal{N}\cup\{x_1\mapsto\kappa_1,\ldots,x_\ell\mapsto\kappa_\ell\},\Sigma,\mathtt{o}}$.
4. S is a non-terminal called the *start symbol*. We require that $S \in dom(\mathcal{N})$ and $\mathcal{N}(S) = \mathtt{o}$.

The *order* of a non-terminal F, written $\mathtt{ord}(F)$, is the order of its sort, i.e. $\mathtt{ord}(\mathcal{N}(F))$. The *order* of a HORS $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$, written $\mathtt{ord}(\mathcal{G})$, is the highest order of its non-terminals.

Intuitively, a HORS $\mathcal{G} = (\Sigma, \mathcal{N}, \{F_1 \mapsto t_1, \ldots, F_k \mapsto t_k\}, F_1)$ is a tree-generating, simply-typed call-by-name functional program, given by the recursive function definitions $F_1 = t_1, \ldots, F_k = t_k$ 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 $\mathtt{o}$ is defined by:

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

Here, $[t_1/x_1,\ldots,t_k/x_k]t$ denotes the term obtained from t by replacing all the (free) occurrences of $x_1,\ldots,x_k$ with $t_1,\ldots,t_k$ 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 t of sort $\mathtt{o}$, the $(\Sigma \cup \{\bot \mapsto 0\})$-labeled tree $t^\bot$ (in the term representation) is defined by: (i) $(a\,t_1\,\cdots\,t_k)^\bot = a\,t_1^\bot\,\cdots\,t_k^\bot$ and (ii) $(F\,t_1\,\cdots\,t_k)^\bot = \bot$. The value tree [22] of $\mathcal{G}$, written $\mathbf{Tree}(\mathcal{G})$, is the $\Sigma \cup \{\bot \mapsto 0\})$-labeled ranked tree obtained as the least upper bound of the set $\{t^\bot \mid S \longrightarrow_\mathcal{G} t\}$ with respect to the least precongruence $\sqsubseteq$ that satisfies $\bot \sqsubseteq T$ for every tree T.

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

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

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

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

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 $\Delta \subset Q \times dom(\Sigma) \times 2^{\{1,\ldots,m\}\times Q}$ is a transition function where m is the largest arity in Σ. We require that if $(q, a, S) \in \Delta$, then $S \subseteq \{1, \ldots, \Sigma(a)\} \times Q$. For a (possibly infinite) Σ-labeled ranked tree T, a $(dom(T) \times Q)$-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 $S = \{(j_1, q_{k_1}), \ldots, (j_\ell, q_{k_\ell})\}$ such that $(q, T(\pi'), S) \in \Delta$ and $\{i \mid \pi i \in dom(R)\} = \{1, \ldots, \ell\}$ with $R(\pi i) = (\pi' j_i, q_{k_i})$ for each $i \in \{1, \ldots, \ell\}$. 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. An 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 $\perp \notin dom(\Sigma)$), we write $\mathcal{A}^\top$ for the ATA $(\Sigma \cup \{\perp \mapsto 0\}, Q, \Delta \cup \{(q, \perp, \emptyset) \mid q \in Q\}, q_I)$, 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 $\Sigma = \{\mathtt{a} \mapsto 2, \mathtt{b} \mapsto 1, \mathtt{c} \mapsto 0\}$ and

$$\Delta_1 = \{(q_0, \mathtt{a}, \{(1, q_0)\}), (q_0, \mathtt{a}, \{(2, q_0)\}), (q_0, \mathtt{b}, \{(1, q_1)\}),\\ (q_1, \mathtt{b}, \emptyset), (q_1, \mathtt{a}, \{(1, q_0)\}), (q_1, \mathtt{a}, \{(2, q_0)\})\}.$$

In the co-trivial mode, $\mathcal{A}_1$ accepts trees that have a path containing two consecutive occurrences of $\mathtt{b}$. 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. ◀

▶ **Remark 5.** *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 6.** 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.

$$\overline{\Delta_1} = \{(\overline{q_0}, \mathtt{a}, \{(1, \overline{q_0}), (2, \overline{q_0})\}), (\overline{q_0}, \mathtt{b}, \{(1, \overline{q_1})\}), (\overline{q_0}, \mathtt{c}, \emptyset), (\overline{q_1}, \mathtt{a}, \{(1, \overline{q_0}), (2, \overline{q_0})\}), (\overline{q_1}, \mathtt{c}, \emptyset)\}$$

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

▶ **Definition 7** (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 5, 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:

$$\tau \text{ (types)} ::= q \mid \sigma \to \tau \qquad \sigma \text{ (intersections)} ::= \bigwedge\{\tau_1, \ldots, \tau_k\}$$

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 $\sigma \to \tau$ describes a function that takes an element of (intersection) type

σ as input, and returns an element of type τ. The intersection $\bigwedge\{\tau_1,\ldots,\tau_k\}$ (where k may be 0) describes an element of the intersection of the sets denoted by $\tau_1,\ldots,\tau_k$. When $k=0$, we write $\top$ for $\bigwedge\{\tau_1,\ldots,\tau_k\}$. We often write $\tau_1\wedge\cdots\wedge\tau_k$ or $\bigwedge_{i\in\{1,\ldots,k\}}\tau_i$ for $\bigwedge\{\tau_1,\ldots,\tau_k\}$. We assume that $\bigwedge$ binds tighter than $\to$, so that $q_0\wedge q_1\to q_2$ means $(q_0\wedge q_1)\to q_2$.

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

$$\frac{q\in Q}{q :: o} \qquad \frac{\tau :: \kappa \qquad \tau_i :: \kappa' \text{ for each } i\in I}{(\bigwedge_{i\in I}\tau_i\to\tau) :: \kappa'\to\kappa}$$

A *type environment* is a *set* of type bindings of the form $x:\tau$. For a type environment Γ, We write $dom(\Gamma)$ 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 $x_1:\tau_1,\ldots,x_n:\tau_n$ for $\{x_1:\tau_1,\ldots,x_n:\tau_n\}$. When $\sigma=\bigwedge\{\tau_1,\ldots,\tau_n\}$, we also write $x:\sigma$ for $x:\tau_1,\ldots,x:\tau_n$.

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

$$\frac{}{\Gamma\cup\{x:\tau\}\vdash_{\mathcal{A},\mathcal{G}} x:\tau}\ (\text{T-Var}) \qquad \frac{(q,a,\{(i,q_j)\mid 1\le i\le\Sigma(a), j\in I_i\})\in\Delta_{\mathcal{A}}}{\Gamma\vdash_{\mathcal{A},\mathcal{G}} a:\bigwedge_{j\in I_1}q_j\to\cdots\to\cdots\bigwedge_{j\in I_{\Sigma(a)}}q_j\to q}\ (\text{T-Con})$$

$$\frac{\Gamma\vdash_{\mathcal{A},\mathcal{G}} t_1:\bigwedge_{i\in I}\tau_i\to\tau \quad \Gamma\vdash_{\mathcal{A},\mathcal{G}} t_2:\tau_i \text{ (for each } i\in I)}{\Gamma\vdash_{\mathcal{A},\mathcal{G}} t_1t_2:\tau}\ (\text{T-App}) \qquad \frac{\{x_i:\tau_j\mid 1\le i\le k, j\in I_i\}\vdash_{\mathcal{A},\mathcal{G}} t:q \quad \mathcal{R}_{\mathcal{G}}(F)=\lambda x_1.\cdots\lambda x_k.t \quad (\bigwedge_{j\in I_1}\tau_j\to\cdots\to\bigwedge_{j\in I_k}\tau_j\to q)::\mathcal{N}_{\mathcal{G}}(F)}{\Gamma\vdash_{\mathcal{A},\mathcal{G}} F:\bigwedge_{j\in I_1}\tau_j\to\cdots\to\bigwedge_{j\in I_k}\tau_j\to q}\ (\text{T-NT})$$

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 8.[1]

▶ **Theorem 8.** *(i)* $\mathbf{Tree}(\mathcal{G})$ *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)* $\mathbf{Tree}(\mathcal{G})$ *is accepted by* $\mathcal{A}^\bot$ *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 practical[2] 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).

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

▶ **Lemma 9.** *Let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ be a HORS. $\mathbf{Tree}(\mathcal{G})$ is accepted by a co-trivial ATA $\mathcal{A}^{\perp}$ if and only if there exists a term t such that $S \longrightarrow_{\mathcal{G}}^{*} t$ and $t^{\perp}$ is accepted by $\mathcal{A}^{\perp}$.*

By the lemma above, for co-trivial ATA model checking, it suffices to start with the set of terms $\mathcal{T}_0 = \{t \in \mathbf{ATerms}_{\mathcal{N},\Sigma,\mathsf{o}} \mid t^{\perp} \text{ is accepted by } \mathcal{A}^{\perp} \text{ in the co-trivial mode}\}$, 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 $\mathbf{Terms}_{\Gamma}$ 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 10.** *$\emptyset \vdash^{-}_{\mathcal{A}} t : q_I$ 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) $\{t \in \mathbf{ATerms}_{\mathcal{N},\Sigma,\mathsf{o}} \mid \exists t'.t \longrightarrow_{\mathcal{G}} t' \in \mathbf{Terms}_{\Gamma}\} \subseteq \mathbf{Terms}_{\mathcal{F}(\Gamma)}$; and
(II) $\mathbf{Terms}_{\mathcal{F}^i(\emptyset)} \subseteq \{t \mid \exists t'.t \longrightarrow_{\mathcal{G}}^{*} t' \in \mathcal{T}_0\}$ 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 $\{\Gamma \mid dom(\Gamma) \subseteq dom(\mathcal{N}) \text{ and } \forall F : \tau \in \Gamma.\tau :: \mathcal{N}(F)\}$, we can obtain $\bigcup_{i\in\omega} \mathcal{F}^i(\emptyset)$ by computing $\mathcal{F}(\emptyset)$, $\mathcal{F}^2(\emptyset)$, $\mathcal{F}^3(\emptyset), \ldots$ 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 11.** *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 $\{t \mid \exists t'.t \longrightarrow_{\mathcal{G}}^{i} t' \in \mathbf{Terms}_{\emptyset}\} \subseteq \mathbf{Terms}_{\mathcal{F}^i(\emptyset)}$. Thus, we have $\mathcal{T} = \mathbf{Terms}_{\bigcup_{i\in\omega} \mathcal{F}^i(\emptyset)}$. The required result follows by Lemma 9. ◀

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

The above lemma implies that if $F\,t_1 \cdots t_\ell \longrightarrow_{\mathcal{G}} [t_1/x_1,\ldots,t_\ell/x_\ell]s$ and $\Gamma \vdash^{-}_{\mathcal{A}} [t_1/x_1,\ldots,t_\ell/x_\ell]s : q$, then $\Gamma \cup \{F : \bigwedge_{j\in I_1} \tau_1 \to \cdots \to \bigwedge_{j\in I_\ell} \tau_j \to q\} \vdash_{\mathcal{A}} F\,t_1 \cdots t_\ell : q$ 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.

$$\mathcal{F}_{\mathcal{G},\mathcal{A},\mathcal{R}}(\Gamma) = \Gamma \cup \{F : \Gamma'(x_1) \to \cdots \to \Gamma'(x_\ell) \to q \mid \mathcal{R}(F) = \lambda x_1.\cdots\lambda x_\ell.t, \\ \mathcal{N}(F) = \kappa_1 \to \cdots \to \kappa_\ell \to \mathsf{o}, \text{ and } x_1 : \kappa_1, \ldots, x_\ell : \kappa_\ell; \Gamma \vdash_{\mathcal{A}} t : q \Rightarrow \Gamma'\}.$$

Here, $\Gamma(x)$ is an abbreviation of $\bigwedge\{\tau \mid x : \tau \in \Gamma\}$. The relation $\mathcal{K}; \Gamma_N \vdash_{\mathcal{A}} t : \tau \Rightarrow \Gamma_V$ (where Γ_N and Γ_V are meta-variables for type environments on non-terminals and variables respectively) is defined by:

Γ := $\emptyset$;
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.

$$\frac{x : \tau \in \mathit{Inhabited}(\mathcal{K}, \Gamma_N)}{\mathcal{K}; \Gamma_N \vdash_{\mathcal{A}} x : \tau \Rightarrow x : \tau} \qquad \frac{\emptyset \vdash_{\mathcal{A},\mathcal{G}} a : \tau}{\mathcal{K}; \Gamma_N \vdash_{\mathcal{A}} a : \tau \Rightarrow \emptyset} \qquad \frac{}{\mathcal{K}; \Gamma_N \cup \{F : \tau\} \vdash_{\mathcal{A}} F : \tau \Rightarrow \emptyset}$$

$$\frac{\mathcal{K}; \Gamma_N \vdash_{\mathcal{A}} t_1 : \bigwedge_{i \in I} \tau_i \to \tau \Rightarrow \Gamma_0 \qquad \mathcal{K}; \Gamma_N \vdash_{\mathcal{A}} t_2 : \tau_i \Rightarrow \Gamma_i \text{ (for each } i \in I) \qquad \Gamma_0 \cup \bigcup_{i \in I} \Gamma_i \in \mathit{Inhabited}(\mathcal{K}, \Gamma)}{\mathcal{K}; \Gamma_N \vdash_{\mathcal{A}} t_1 t_2 : \tau \Rightarrow \Gamma_0 \cup \bigcup_{i \in I} \Gamma_i}$$

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

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

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 13.** *If $\mathcal{K}; \Gamma_N \vdash t : \tau \Rightarrow \Gamma_V$, then $\Gamma_N \cup \Gamma_V \vdash t : \tau$ and $\Gamma_V \in \mathit{Inhabited}(\mathcal{K}, \Gamma_N)$. Conversely, if $\Gamma_N \cup \Gamma_V \vdash^{-}_{\mathcal{A}} t : \tau$ and $\Gamma_V \in \mathit{Inhabited}(\mathcal{K}, \Gamma_N)$, 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 $\mathcal{K}, \Gamma_N$ and τ as input, outputs Γ_V such that $\Gamma_N, \Gamma_V \vdash^{-}_{\mathcal{A}} t : \tau$ and $\Gamma_V \in \mathit{Inhabited}(\mathcal{K}, \Gamma_N)$. For checking the inhabitance condition $\Gamma \in \mathit{Inhabited}(\mathcal{K}, \Gamma_N)$, 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 14.** *Suppose $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}', S)$ and $\mathcal{R} \subseteq \mathcal{R}'$. If $\Gamma \vdash^{-}_{\mathcal{A}} t : q$ and $s \longrightarrow_{\mathcal{R}} t$, 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 $t \longrightarrow_{\mathcal{R}} t'$. Since the induction step is trivial, we show only the base case, where $t = F\, t_1 \cdots t_\ell$ and $t' = [t_1/x_1, \ldots, t_\ell/x_\ell]s$ with $\lambda x_1. \cdots \lambda x_\ell. s = \mathcal{R}(F)$. Let $\mathcal{N}(F) = \kappa_1 \to \cdots \to \kappa_\ell \to \mathsf{o}$. By Lemma 12, we have:

$$\Gamma \cup \{x_i : \tau_j \mid i \in \{1, \ldots, \ell\}, j \in I_i\} \vdash^{-}_{\mathcal{A}} s : q \qquad \Gamma \vdash^{-}_{\mathcal{A}} t_i : \tau_j \text{ for each } i \in \{1, \ldots, \ell\}, j \in I_i$$

and $\tau_j :: \kappa_i$ for every $j \in I_i$. By Lemma 13, there exists Γ_V such that

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

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

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

▶ **Theorem 15.** *Let the function $\mathcal{F}_{\mathcal{G},\mathcal{A}}$ be as defined above. Then, $S : q_I \in \bigcup_{i\in\omega} \mathcal{F}^i_{\mathcal{G},\mathcal{A}}(\emptyset)$ if and only if* $\mathbf{Tree}(\mathcal{G})$ *is accepted by $\mathcal{A}^\perp$ in the co-trivial mode.*

Proof. By Lemma 11, 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 14.

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 $\mathcal{G}$:

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

$\mathcal{G}^{(m)}$ 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 27 in Appendix B. By the strong normalization of the simply-typed λ-calculus, we have $t^{(m)} \longrightarrow^*_{\mathcal{G}^{(m)}} u$ for some $(\Sigma\cup\{\perp \mapsto 0\})$-labeled tree (in the term representation) u. By the type preservation property (Lemma 29 in Appendix B), we have $\emptyset \vdash_{\mathcal{A}^\perp,\mathcal{G}^{(m)}} u : q_I$. By Lemma 10, u is accepted by $\mathcal{A}^\perp$. By the construction of $t^{(m)}$, there exists a term t' such that $t \longrightarrow^*_{\mathcal{G}} t'$ and $t'^\perp = u$. Thus, we have $t \in \{s \mid \exists t'.s \longrightarrow^*_{\mathcal{G}} t' \in \mathcal{T}_0\}$ as required. ◀

▶ **Example 16.** Let $\mathcal{G}$ be $\mathcal{G}_1$ in Example 2 and $\mathcal{A}$ be $\mathcal{A}_1$ in Example 4. Then, since $\emptyset \vdash_{\mathcal{A}} \mathtt{a}\,x\,(F(\mathtt{b}\,x)) : q_0 \Rightarrow x : q_0$ and $\emptyset \vdash_{\mathcal{A}} \mathtt{a}\,x\,(F(\mathtt{b}\,x)) : q_1 \Rightarrow x : q_0$, we have:

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

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

$$\mathcal{F}^2(\emptyset) = \mathcal{F}(\emptyset) \cup \{F : q_1 \to q_0, F : q_1 \to q_1\} \qquad \mathcal{F}^3(\emptyset) = \mathcal{F}^2(\emptyset) \cup \{F : \top \to q_0, F : \top \to q_1\}$$
$$\mathcal{F}^4(\emptyset) = \mathcal{F}^3(\emptyset) \cup \{S : q_0, S : q_1\} = \mathcal{F}^5(\emptyset)$$

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. ◀

▶ **Remark 17.** *The inhabitance check needed for computing $\mathcal{F}(\Gamma)$ can be quite costly: in fact, the inhabitance problem is EXPTIME-complete [25]. In order to avoid the drawback, we can actually replace $\mathit{Inhabited}(\mathcal{K}, \Gamma_N)$ in the rules for the relation $\mathcal{K}; \Gamma_N \vdash_{\mathcal{A}} t : \tau \Rightarrow \Gamma_V$ with an over-approximation $\mathit{Inhabited}'(\mathcal{K}, \Gamma_N)$, such that*

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

By the proof of Theorem 15, our co-trivial model checking algorithm remains sound and complete for any such over-approximation.

▶ **Remark 18.** *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) are fixed. 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 inhabitance 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 8, 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 analyses 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 $\mathit{Shrink}_{\mathcal{G}}$ on type environments by:

$$\mathit{Shrink}_{\mathcal{G}}(\Gamma) = \{F : \sigma_1 \to \cdots \to \sigma_k \to q \in \Gamma \mid \mathcal{R}(F) = \lambda x_1.\cdots\lambda x_\ell.t \text{ and } \Gamma, x_1 : \sigma_1, \ldots, x_\ell : \sigma_\ell \vdash_{\mathcal{A}} t : q\}.$$

Intuitively, $\mathit{Shrink}_{\mathcal{G}}(\Gamma)$ picks each type binding $F : \sigma_1 \to \cdots \to \sigma_k \to q$ 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 $\mathit{Shrink}_{\mathcal{G}}(\Gamma) = \Gamma$, 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 8: see Appendix B.

▶ **Theorem 19.** $\mathbf{Tree}(\mathcal{G})$ *is accepted by* $\mathcal{A}^\top$ *in trivial mode if and only if* $\Gamma \vdash (\mathcal{G}, S) : q_I$ *for some* Γ.

Based on Theorem 19, a type environment Γ such that $\Gamma \vdash (\mathcal{G}, S) : q_I$ serves as a certificate for $\mathbf{Tree}(\mathcal{G})$ being accepted by $\mathcal{A}$.

▶ **Example 20.** Recall $\mathcal{G}_1$ in Example 2. Let $\mathcal{A}_2 = (\{\mathtt{a} \mapsto 2, \mathtt{b} \mapsto 1, \mathtt{c} \mapsto 0\}, \{q_0, q_1\}, \Delta_2, q_0)$ where $\Delta_2 = \{(q_0, \mathtt{a}, \{(1, q_0), (2, q_0)\}), (q_0, \mathtt{b}, \{(1, q_1)\}), (q_0, \mathtt{c}, \emptyset), (q_1, \mathtt{b}, \{(1, q_1)\}), (q_1, \mathtt{c}, \emptyset)\}$. $\mathcal{A}_2$ describes the property that $\mathtt{a}$ cannot occur below $\mathtt{b}$. $\mathbf{Tree}(\mathcal{G}_1)$ is accepted by $\mathcal{A}_2$, and the type environment $\{S : q_0, F : q_0 \wedge q_1 \to q_0\}$ serves as a certificate. ◀

The existing algorithms for trivial automata model checking [11, 14] first compute an overapproximation Γ' of a possible certificate, compute the fixedpoint $\Gamma = \bigcap_{j\in\omega} \mathit{Shrink}^j(\Gamma')$, 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 type environment $\Gamma_0 = \{F : \underbrace{\top \to \cdots \to \top}_{\mathtt{ar}(\mathcal{N}(F))} \to q \mid F \in \mathit{dom}(\mathcal{N})\}$. $\mathbf{Terms}_{\Gamma_0}$ is the set of terms t such that $t^\perp$ is accepted by $\mathcal{A}^\top$. Thus, for $\Gamma' = \bigcup_{i\in\omega} \mathcal{F}^i(\Gamma_0)$, $\mathbf{Terms}_{\Gamma'}$ is

```
Γ′ := Γ0;
while not(F_{G,A}(Γ′)=Γ′) do
  (Γ′ := F_{G,A}(Γ′); Γ := Γ′; (while not(Shrink(Γ)=Γ) do Γ := Shrink(Γ));
   if S : q_I ∈ Γ then return Γ );
return FALSE
```

Figure 3 Trivial ATA model checking algorithm HorSatT.

the set $\{t \mid t \longrightarrow_{\mathcal{G}}^{*} t' \text{ and } t'^{\perp} \text{ is accepted by } \mathcal{A}^{\top}\}$. 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 $\mathbf{Terms}_{\Gamma'}$, 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 21.** *$S : q_I \in \bigcap_{j\in\omega} \mathit{Shrink}^j(\bigcup_{i\in\omega} \mathcal{F}^i(\Gamma_0))$ if and only if $\mathbf{Tree}(\mathcal{G})$ 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 $\mathcal{F}(\Gamma_0), \mathcal{F}^2(\Gamma_0), \ldots$, until it converges to a fixedpoint, or finds a certificate. The inner loop computes $\Gamma = \bigcap_{j\in\omega} \mathit{Shrink}^j(\mathcal{F}^i(\Gamma_0))$. 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 22.** Recall $\mathcal{G}_1$ in Example 2 and $\mathcal{A}_2$ in Example 20. $\Gamma_0, \mathcal{F}(\Gamma_0), \mathcal{F}^2(\Gamma_0), \ldots$ are computed as follows.

$$\Gamma_0 = \{S : q_0, S : q_1, F : \top \to q_0, F : \top \to q_1\}$$
$$\mathcal{F}(\Gamma_0) = \Gamma_0 \cup \{F : q_0 \to q_0\} \qquad \mathcal{F}^2(\Gamma_0) = \mathcal{F}(\Gamma_0) \cup \{F : q_0 \wedge q_1 \to q_0\} = \mathcal{F}^3(\Gamma_0)$$

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

$$\mathit{Shrink}(\Gamma') = \{S : q_0, S : q_1, F : q_0 \to q_0, F : q_0 \wedge q_1 \to q_0\}$$
$$\mathit{Shrink}^2(\Gamma') = \{S : q_0, F : q_0 \wedge q_1 \to q_0\} = \mathit{Shrink}^3(\Gamma').$$

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

▶ **Remark 23.** 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 $q_1 \to \cdots \to q_n \to q$ may have any type of the form $\sigma_1 \to \cdots \to \sigma_n \to q$ for $\sigma_i \in \{\top, q_i\}$ in the interpretation of [14]; thus the number of possible types blows up.*

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 $\mathit{Inhabited}(\mathcal{K}, \Gamma_N)$ in Section 3 has been replaced by

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

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 0CFA 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 software model checker MoCHi [19], the third from the PMRS model checker [23], 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 HORS are respectively listed in the O and Sz columns (where the size of a HORS is defined as the total number of occurrences of symbols in the righthand side of the rewriting rules). 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 (`fold_right`); this confirms the advantage of directly working on HORS without a detour to CPDS. According to further experiments (by Steven Ramsay) using the benchmark $\mathcal{G}_{m,n}$ [14], the running time of HorSat is not linear in the size of $\mathcal{G}_{m,n}$, even if we exclude out the time for 0CFA (whose worst case complexity is cubic time) to compute $\mathbf{Flow}(x)$. We believe that this is due the naiveness of the current implementation, and that an improved implementation would make the running time almost linear in the size of $\mathcal{G}_{m,n}$.

Table 2 Comparison of model-checking tools.

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`	3	2731	—	—	—	5.145	1.702	**0.956**
`xhtmlf-div-2` **(neg)**	2	3003	**0.327**	51.8	—	—	12.392	2.697
`xhtmlf-m-ch`	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-ch` **(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	—	—	—
`l`	3	35	—	0.010	23.322	0.439	**0.002**	0.006

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.

As already noted in Section 1, our new algorithms HorSat and HorSatT bridge the gap between the two families of model checking algorithms mentioned above. On the one hand, HorSat and HorSatT 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 $\mathbf{Terms}_\Gamma$ 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.

Acknowledgment. We thank anonymous referees for useful comments. This work was partially supported by Kakenhi 23220001. The first author was supported by JSPS and AvH Postdoc. Fellowships.

References

1 Klaus 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 Christopher Broadbent and Naoki Kobayashi. Streamlining collapsible pushdown systems and their model checking, 2013. Draft.

4 Christopher H. Broadbent, Arnaud Carayol, Matthew Hague, and Olivier Serre. A saturation method for collapsible pushdown systems. In *Proceedings of ICALP 2012*, volume 7392 of *LNCS*, pages 165–176. Springer-Verlag, 2012.

5 Christopher H. Broadbent, Arnaud Carayol, Matthew Hague, and Olivier Serre. C-SHORe: A collapsible approach to verifying higher-order programs. In *Proceedings of ICFP 2013*, 2013.

6 Arnaud Carayol and Olivier 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 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*. Springer-Verlag, 2002.

9 Matthew Hague, Andrzej Murawski, C.-H. Luke Ong, and Olivier 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 Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS 2002*, volume 2303 of *LNCS*, pages 205–222. Springer-Verlag, 2002.

11 Naoki Kobayashi. Model-checking higher-order functions. In *Proceedings of PPDP 2009*, pages 25–36. ACM Press, 2009.

12 Naoki Kobayashi. TRecS: A type-based model checker for recursion schemes. `http://www-kb.is.s.u-tokyo.ac.jp/~koba/trecs/`, 2009.

13 Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proc. of POPL*, pages 416–428, 2009.

14 Naoki 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 Naoki 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/gtrecs2/`, 2012.

16 Naoki Kobayashi, Kazutaka Matsuda, Ayumi Shinohara, and Kazuya Yaguchi. Functional programs as compressed data. *Higher-Order and Symbolic Computation*, 2013. To appear. A preliminary version appeared in Proceedings of PEPM12.

17 Naoki Kobayashi and C.-H. Luke 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 Naoki Kobayashi and C.-H. Luke Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. *Logical Methods in Computer Science*, 7(4), 2011.

19 Naoki Kobayashi, Ryosuke Sato, and Hiroshi Unno. Predicate abstraction and CEGAR for higher-order model checking. In *Proc. of PLDI*, pages 222–233, 2011.

20 Naoki Kobayashi, Naoshi Tabuchi, and Hiroshi Unno. Higher-order multi-parameter tree transducers and recursion schemes for program verification. In *Proc. of POPL*, pages 495–508, 2010.

21 Robin P. Neatherway, Steven James Ramsay, and C.-H. Luke 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. Luke 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. Luke Ong and Steven Ramsay. Verifying higher-order programs with pattern-matching algebraic data types. In *Proc. of POPL*, pages 587–598, 2011.

24 C.-H. Luke Ong and Takeshi 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 Jakob Rehof and Pawel Urzyczyn. Finite combinatory logic with intersection types. In *Proceedings of TLCA 2011*, volume 6690 of *LNCS*, pages 169–183. Springer, 2011.

26 Yoshihiro Tobita, Takeshi Tsukada, and Naoki 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 8

Kobayashi and Ong's result [17] directly imply the following special cases of Theorem 8.

▶ **Theorem 24.** *Suppose* $\mathbf{Tree}(\mathcal{G})$ *does not contain* $\bot$*. 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 25.** *Suppose* $\mathbf{Tree}(\mathcal{G})$ *does not contain* $\bot$*. 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 8 from Theorems 24 and 25. Suppose $\mathbf{Tree}(\mathcal{G})$ may contain $\bot$. Let $\mathcal{G}'$ be a HORS that is obtained by adding a special terminal *loop* of arity 1, and replacing each rule $F\,x_1 \cdots x_k \to t$ of $\mathcal{G}$ with $F\,x_1 \cdots x_k \to \mathit{loop}(t)$. Then, $\mathbf{Tree}(\mathcal{G}')$ does not contain $\bot$, and $\mathbf{Tree}(\mathcal{G})$ is obtained from $\mathbf{Tree}(\mathcal{G}')$ by removing every infinite sequence *loop* with $\bot$ and removing other occurrences of *loop*. Let $\mathcal{A}'$ be the ATA obtained from $\mathcal{A}$ by adding the transition rule $(q, \mathit{loop}, \{(1, q)\})$ 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 9

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., $D = \{\pi \mid (\pi, a) \in \mathit{codom}(R)\}$. By the definition of $\mathbf{Tree}(\mathcal{G})(= \bigsqcup\{t^\bot \mid S \longrightarrow_{\mathcal{G}}^* t\})$, there exists t such that $S \longrightarrow_{\mathcal{G}}^* t$ and $U \subseteq \mathit{dom}(t^\bot)$ with $\mathbf{Tree}(\mathcal{G})(\pi) = t^\bot(\pi)$ for every $\pi \in U$. Then, R is also a run-tree of $\mathcal{A}^\bot$ over $t^\bot$. Thus, t satisfies the required property.

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

Proof of Lemma 10

Let $\mathcal{A} = (\Sigma, Q, \Delta, q_I)$. We show that $\emptyset \vdash_{\mathcal{A}}^- t : q$ if and only if $t^\bot$ 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:

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

By the induction hypothesis, ${t_i}^\bot$ is accepted by (Σ, Q, Δ, q_j) for each $i \in \{1, \ldots, k\}, j \in I_i$. Thus, $t^\bot = a\,{t_1}^\bot \cdots {t_k}^\bot$ 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 $a\,t_1 \cdots t_k$. So, we have: $(q, a, \{(i, q_j) \mid i \in \{1, \ldots, k\}, j \in I_i\}) \in \Delta$ for some $I_1, \ldots, I_k$ and ${t_i}^{\perp}$ is accepted by (Σ, Q, Δ, q_j) for each $i \in \{1, \ldots, k\}, j \in I_i$. By the induction hypothesis, $\emptyset \vdash^{-}_{\mathcal{A}} t_i : q_j$ for each $i \in \{1, \ldots, k\}, j \in I_i$. Thus, we have $\emptyset \vdash^{-}_{\mathcal{A}} t : q$ as required. ◀

▶ **Lemma 26.** *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

$$\Gamma \vdash^{-}_{\mathcal{A}} [t/x]s_1 : \textstyle\bigwedge_{j \in J} \tau'_j \to \tau \qquad \Gamma \vdash^{-}_{\mathcal{A}} [t/x]s_2 : \tau'_j \text{ for each } j \in J$$

By the induction hypothesis, we have:

$$\Gamma \cup \{x : \tau_i \mid i \in I_0\} \vdash^{-}_{\mathcal{A}} s_1 : \textstyle\bigwedge_{j \in J} \tau'_j \to \tau \qquad \Gamma \cup \{x : \tau_i \mid i \in I_j\} \vdash^{-}_{\mathcal{A}} s_2 : \tau'_j \text{ for each } j \in J$$
$$\Gamma \vdash^{-}_{\mathcal{A}} t : \tau_i \text{ for each } i \in I \cup \textstyle\bigcup_{j \in J} I_j.$$

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.

◀

Proof of Lemma 12

This follows by repeated applications of Lemma 26. ◀

▶ **Lemma 27.** *Let $\mathcal{G}^{(m)}$ and $t^{(m)}$ as defined in the proof of Theorem 15. 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 $\mathcal{F}^m(\emptyset) \vdash^{-}_{\mathcal{A}} t : \tau$, 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 13, we have

$$\tau = \Gamma_V(x_1) \to \cdots \to \Gamma_V(x_k) \to q \qquad \mathcal{R}(F) = \lambda x_1. \cdots \lambda x_\ell. s \qquad \mathcal{F}^{m-1}(\emptyset) \cup \Gamma_V \vdash^{-}_{\mathcal{A}} t : q$$

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. ◀

▶ **Lemma 28** (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 $x \notin dom(\Gamma)$, then $\Gamma \vdash_{\mathcal{A}, \mathcal{G}} [t/x]s : q$.*

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

▶ **Lemma 29** (type preservation). *If $\emptyset \vdash_{\mathcal{A}, \mathcal{G}} t : q$ and $t \longrightarrow_{\mathcal{G}} t'$, then $\emptyset \vdash_{\mathcal{A}, \mathcal{G}} t' : q$.*

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

$$\{x_i : \tau_j \mid i \in \{1, \ldots, k\}, j \in I_i\} \vdash^{-}_{\mathcal{A}} \mathcal{G}s : q \qquad \emptyset \vdash^{-}_{\mathcal{A}} \mathcal{G}t_i : \tau_j \text{ for each } i \in \{1, \ldots, k\}, j \in I_i$$

By the substitution lemma (Lemma 28), we have $\emptyset \vdash^{-}_{\mathcal{A}} \mathcal{G}[t_1/x_1, \ldots, t_k/x_k]s : q$ as required. ◀

B.2 Proofs for Section 4

Proof of Theorem 19

This follows immediately from Theorem 8. Note that there is a fixedpoint Γ of $\mathit{Shrink}_{\mathcal{G}}$ 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. ◀

The rest of this subsection is devoted to the proof of Theorem 21.

▶ **Lemma 30.** *Suppose $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}', S)$ and $\mathcal{R} \subseteq \mathcal{R}'$. If $\Gamma \vdash (\mathcal{G}, t) : q$ and $s \longrightarrow_{\mathcal{R}} t$, then $\mathcal{F}_{\mathcal{G},\mathcal{A},\mathcal{R}}(\Gamma) \vdash (\mathcal{G}, s) : q$.*

Proof. By Lemma 14, we have $\mathcal{F}_{\mathcal{G},\mathcal{A},\mathcal{R}}(\Gamma) \vdash s : q$. So, it remains to show $\mathit{Shrink}(\mathcal{F}_{\mathcal{G},\mathcal{A},\mathcal{R}}(\Gamma)) = \mathcal{F}_{\mathcal{G},\mathcal{A},\mathcal{R}}(\Gamma)$. Suppose $F : \sigma_1 \to \cdots \to \sigma_k \to q \in \mathcal{F}_{\mathcal{G},\mathcal{A},\mathcal{R}}(\Gamma) \setminus \Gamma$ and $\mathcal{R}(F) = \lambda x_1. \cdots \lambda x_\ell.t$. By the definition of $\mathcal{F}$ and Lemma 13, we have $\Gamma, x_1 : \sigma_1, \ldots, x_\ell : \sigma_\ell \vdash_{\mathcal{A}} t : q$. Thus, we have $\mathcal{F}_{\mathcal{G},\mathcal{A},\mathcal{R}}(\Gamma), x_1 : \sigma_1, \ldots, x_\ell : \sigma_\ell \vdash_{\mathcal{A}} t : q$ as required. ◀

▶ **Lemma 31.** *If $\mathbf{Tree}(\mathcal{G})$ is accepted by $\mathcal{A}^{\top}$ in the trivial mode, then $S : q_I$ is an element of $\bigcap_{j\in\omega} \mathit{Shrink}^j(\bigcup_{i\in\omega} \mathcal{F}^i(\Gamma_0))$.*

Proof. Let $\mathcal{G}$ be $(\Sigma, \mathcal{N}, \mathcal{R}, S)$, where $\mathit{dom}(\mathcal{N}) = \{F_1, \ldots, F_n\}$ and $S = F_1$. Let $\Gamma_{\mathbf{max}}$ 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 $|\Gamma_{\mathbf{max}}|$. Let $\mathcal{G}^{(m)}$ be the HORS $(\Sigma \cup \{\bot \mapsto 0\}, \mathcal{N}^{(m)}, \mathcal{R}^{(m)}, F_1^{(m)})$ constructed as an approximation of $\mathcal{G}$ in the proof of Theorem 15. Let $\mathcal{R}'^{(m)}$ be the following subset of $\mathcal{R}^{(m)}$:

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

By the strong normalization of the simply-typed λ-calculus, $F_1^{(m)} \longrightarrow^*_{\mathcal{R}'^{(m)}} t \not\longrightarrow_{\mathcal{R}'^{(m)}}$ for some t, and there is a corresponding reduction sequence $F_1 \longrightarrow^*_{\mathcal{G}} t'$ of $\mathcal{G}$ such that $t'^{\bot} = t^{\bot}$. Thus, $t^{\bot}$ is accepted by $\mathcal{A}^{\top}$. By Lemma 10, we have $\emptyset \vdash^-_{\mathcal{A}} t^{\bot} : q_I$. 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 30, we have $\Gamma' \vdash (\mathcal{G}^{(m)}, F_1^{(m)}) : q_I$ for $\Gamma' = \bigcup_{i\in\omega} \mathcal{F}^i_{\mathcal{G}^{(m)},\mathcal{A}^{\top},\mathcal{R}'^{(m)}}(\Gamma_0^{(0)})$. Let Γ'_j be: $\{F_i : \tau \mid F_i^{(j')} : \tau \in \Gamma' \wedge j' \geq j\}$. By the condition $\Gamma' = \bigcup_{i\in\omega} \mathcal{F}^i_{\mathcal{G}^{(m)},APT^{\top},\mathcal{R}'^{(m)}}(\Gamma_0^{(0)})$, we have $\Gamma'_0 \subseteq \bigcup_{i\in\omega} \mathcal{F}^i_{\mathcal{G},\mathcal{A},\mathcal{R}}(\Gamma_0)$. 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:

$$\Gamma{\downarrow}_{\{F_1^{(k)},\ldots,F_n^{(k)}\}}, x_1 : \sigma_1, \ldots, x_\ell : \sigma_\ell \vdash^-_{\mathcal{A}} [F_1^{(k)}/F_1, \ldots, F_n^{(k)}/F_n]t : \tau$$

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

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

◀

Proof of Theorem 21

The "only if" direction follows immediately from Theorem 19. The "if" direction follows from Lemma 31. ◀

C An algorithm to check the inhabitance condition

C.1 The construction

Computing $\mathcal{F}(\Gamma)$ requires a procedure to check $\Gamma \overset{?}{\in} \mathit{Inhabited}(\mathcal{K}, \Gamma_N)$. 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:

$$
\begin{aligned}
\Sigma' &= \{a \mapsto 0 \mid a \in \mathit{dom}(\Sigma)\} \cup \{F \mapsto 0 \mid F \in \mathit{dom}(\mathcal{N})\} \\
&\quad \cup \{@ \mapsto 2\} \\
Q' &= \{(\tau, \kappa) \mid \tau :: \kappa, \text{ and } \tau \text{ occurs (as a sub-expression)} \\
&\qquad \text{in } \Gamma_N \text{ or a type of a constant}\} \\
\Delta' &= \{((\tau,\kappa), a, \emptyset) \mid \emptyset \vdash^-_{\mathcal{A}} a : \tau \text{ and } \kappa = \underbrace{\mathsf{o} \to \cdots \to \mathsf{o}}_{\Sigma(a)} \to \mathsf{o}\} \\
&\quad \cup \{((\tau, \mathcal{N}(F)), F, \emptyset) \mid F : \tau \in \Gamma_N\} \\
&\quad \cup \{((\tau,\kappa), @, \\
&\qquad \{(1, (\textstyle\bigwedge_{i\in I} \tau_i \to \tau, \kappa' \to \kappa))\} \cup \{(2, (\tau_i, \kappa')) \mid i \in I\}) \\
&\qquad\quad \mid (\textstyle\bigwedge_{i\in I} \tau_i \to \tau, \kappa' \to \kappa) \in Q\} \\
q_I' &= (q_I, \mathsf{o})
\end{aligned}
$$

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 $\Gamma \in \mathit{Inhabited}(\mathcal{K}, \Gamma_N)$, it suffices to check that for every $x \in \mathit{dom}(\Gamma)$, 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 $m = 1, 2, \ldots$), 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 inhabitance 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 inhabitance checking does not negate the fixed parameter tractability of the algorithm.

First observe that, for the purposes of inhabitance 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:

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

It should be clear that

$$\mathit{Inhabited}(\mathcal{K},\Gamma_N) = \mathit{Inhabited}(U_{\mathcal{K},\Gamma_N}, \Delta_{\mathcal{K},\Gamma_N})$$

After all, given a term t witnessing $\{x : \tau \mid \tau \in T\} \in \mathit{Inhabited}(\mathcal{K},\Gamma_N)$ we can construct a term t' witnessing $\{x : \tau \mid \tau \in T\} \in \mathit{Inhabited}(U_{\mathcal{K},\Gamma_N}, \Delta_{\mathcal{K},\Gamma_N})$ 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 $\{x : \tau \mid \tau \in T\} \in \mathit{Inhabited}(U_{\mathcal{K},\Gamma_N}, \Delta_{\mathcal{K},\Gamma_N})$, we can construct a term t witnessing $\{x : \tau \mid \tau \in T\} \in \mathit{Inhabited}(\mathcal{K},\Gamma_N)$ 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 $a \in \mathit{dom}(\Sigma)$ 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 $\Delta_{\mathcal{K},\Gamma_N}$ is bounded by a constant, namely the number of different well-sorted sets of intersection types. Thus in particular the emptiness of $\mathit{Inhabited}(U_{\mathcal{K},\Gamma_N}, \Delta_{\mathcal{K},\Gamma_N})$ can be determined in constant time (for example by the algorithm sketched in the previous section whose run-time depends only on $|\Delta_{\mathcal{K},\Gamma_N}|$ and the number of intersection types). So assuming that $\Delta_{\mathcal{K},\Gamma_N}$ has already been computed, $\mathit{Inhabited}(\mathcal{K},\Gamma_N)$ can be computed in constant time.

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

As an aid the HorSat algorithm could maintain a table H with $\mathit{dom}(\mathcal{N})$ 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 $G \in \mathit{dom}(\mathcal{N}) \cup \mathit{dom}(\Sigma)$, P' is also added to $\Delta_{\mathcal{K},\Gamma_N}$.

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 inhabitance check is performed).