Abstract

We propose a practical algorithm for Streett automata model checking of higher-order recursion schemes (HORS, for short) [13, 25], which checks whether the tree generated by a given HORS is accepted by a given Streett automaton. The Streett automata model checking of HORS is useful in the context of liveness verification of higher-order functional programs. The previous approach to Streett automata model checking converted Streett automata to parity automata and then invoked a parity tree automata model checker. We show through experiments that our direct approach outperforms the previous approach. Besides being able to directly deal with Streett automata, our algorithm is the first practical Streett or parity automata model checking algorithm that runs in time polynomial in the size of HORS, assuming that the other parameters are fixed. Previous practical fixed-parameter polynomial time algorithms for HORS could only deal with the class of trivial tree automata. We have confirmed through experiments that (a parity automata version of) our model checker outperforms previous parity automata model checkers for HORS.

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

Keywords and phrases Higher-order model checking, higher-order recursion schemes, Streett automata

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.32

1 Introduction

Model checking of higher-order recursion schemes (HORS, for short) [13, 25] has been actively studied and applied to automated verification of higher-order programs [17, 21, 24, 26, 34, 23, 41]. The model checking problem asks whether the tree generated by a given HORS is accepted by a given tree automaton. Despite the extremely high complexity (\(k\)-EXPTIME complete for order-\(k\) HORS), practical model checkers that work reasonably well for typical inputs have been developed [14, 16, 3, 31, 29, 7, 28]. In particular, the state-of-the-art trivial automata model checkers for HORS (i.e., model checkers which handle the restricted class of automata called trivial tree automata [1]) [3, 18, 31] can handle thousands of lines of input in a few seconds. The state-of-the-art model checkers for the full class of tree automata (that
is equi-expressive to the modal $\mu$-calculus and MSO) have, however, been much behind the trivial automata model checkers. Indeed, whilst the state-of-the-art trivial automata model checkers for HORS employ fixed-parameter polynomial time algorithms, existing parity tree automata model checkers for HORS \[7, 28\] do not. Another limitation of the state-of-the-art model checkers for HORS is that the class of automata is restricted to trivial or parity tree automata. Whilst parity tree automata are equi-expressive to other classes of tree automata like Streett, Rabin, and Muller automata, the translation from those automata to parity tree automata significantly increases the number of states. Thus, it may be desirable for model checkers to support other classes of automata directly.

To address the limitations above, we propose a practical Streett automata model checking algorithm for HORS, which checks whether the tree generated by a given HORS is accepted by a given Streett automaton. Compared with the previous model checking algorithms for HORS that can deal with the full class of tree automata \[7, 28\], our new algorithm has the following advantages: (i) It can directly deal with Streett automata, which naturally arise in the context of liveness verification of higher-order programs \[41\]. (ii) More importantly, it runs in time polynomial in the size of HORS, assuming that the other parameters (the size of the automaton and the largest order and arity of non-terminals in HORS) are fixed. The previous parity automata model checkers for HORS \[7, 28\] did not satisfy this property, and suffered from hyper-exponential time complexity in the size of HORS.

We develop the algorithm in two steps. First, following Kobayashi and Ong’s type system for parity automata model checking of HORS \[19\], we prepare a type system for Streett automata model checking such that the tree generated by a HORS $G$ is accepted by a Streett automaton $A$ if and only if $G$ is typable in the type system parameterized by $A$. We prove its correctness by showing that the type system can actually be viewed as an instance of Tsukada and Ong’s type system \[38\]. Secondly, we develop a practical algorithm for checking the typability. The algorithm has been inspired by Broadbent and Kobayashi’s saturation-based algorithm \[3\] for trivial automata model checking; in fact, the algorithm is a simple modification of their HorSatT algorithm. The proof of the correctness of our algorithm is, however, non-trivial and much more involved than the correctness proof for HorSatT. The correctness proof is one of the main contributions of the present paper.

We have implemented a new model checker HorSatS based on the proposed algorithm and its variation, called HorSatP,\(^1\) for parity tree automata model checking, and experimentally confirmed the two advantages above. For the advantage (i), we have confirmed that HorSatS is often faster than the combination of a converter from Streett to parity tree automata, and HorSatP. For (ii), we have confirmed that HorSatP often outperforms previous parity automata model checkers \[7, 28\].

The rest of the paper is organized as follows. Section 2 reviews basic definitions. Section 3 provides a type-based characterization of Streett automata model checking of HORS and proves its correctness. Section 4 develops a practical algorithm for Streett automata model checking and proves its correctness. Section 5 reports experimental results. Section 6 discusses related work and Section 7 concludes the paper. Proofs omitted in this paper are found in a longer version of the paper \[36\].

\(^1\) Actually, HorSatP has been implemented in 2015 and used in Watanabe et al.’s work \[41\]. It has not been properly formalized, however.
2 Preliminaries

In this section we review the definitions of higher-order recursion schemes (HORS) [12, 25, 17] and (alternating) Streett tree automata [35, 8] to define Streett automata model checking of HORS. We write dom(f) for the domain of a map f, and over a sequence $x_1, x_2, \ldots, x_m$ (for some $m$). Given a set $\Sigma$ of symbols (which we call terminals), a $\Sigma$-labeled (unranked) tree is a partial map $T$ from $\{1, \ldots, N\}^*$ (for some fixed $N \in \mathbb{N}$) to $\Sigma$ such that if $\pi i \in \text{dom}(T)$, then $\pi \in \text{dom}(T)$ and $\pi j \in \text{dom}(T)$ for all $1 \leq j \leq i$. If $\Sigma$ is a ranked alphabet (i.e., a map from terminals to natural numbers), a $\Sigma$-labeled ranked tree $T$ is a dom($\Sigma$)-labeled tree such that for each $\pi \in \text{dom}(T)$, $\{i | \pi i \in \text{dom}(T)\} = \{1, \ldots, \Sigma(\pi)\}$.

2.1 Higher-Order Recursion Schemes

The set of sorts, ranged over by $\kappa$, is defined by $\kappa ::= \circ | \kappa_1 \rightarrow \kappa_2$. We sometimes abbreviate $\kappa_1 \rightarrow \cdots \rightarrow \kappa \rightarrow \kappa'$ to $\kappa^n \rightarrow \kappa'$. The sorts can be viewed as the types of the simply-typed $\lambda$-calculus with the single base type $\circ$, with the type of trees. We write $\text{Terms}_{\Sigma, \kappa}$ for the set of simply-typed $\lambda$-terms that have the sort $\kappa$ under the environment $\Sigma$. The order and arity of each sort, denoted by $\text{ord}(\kappa)$ and $\text{arity}(\kappa)$, are defined inductively by: $\text{ord}(\circ) = \text{arity}(\circ) = 0$, $\text{ord}(\kappa_1 \rightarrow \kappa_2) = \max(\text{ord}(\kappa_1) + 1, \text{ord}(\kappa_2))$, and $\text{arity}(\kappa_1 \rightarrow \kappa_2) = \text{arity}(\kappa_2) + 1$.

Definition 1 (higher-order recursion scheme). A higher-order recursion scheme (HORS) is a quadruple $G = (\Sigma, N, \mathcal{R}, S)$ where: (i) $\Sigma$ is a ranked alphabet. (ii) $N$ is a map from symbols called non-terminals to sorts. (iii) $\mathcal{R}$ is a map from non-terminals to simply-typed terms of the form $\lambda x.t$, where $t$ does not include $\lambda$-abstractions and has the sort $\circ$. It is required that for each $F \in \text{dom}(N)$, $\lambda x.t \in \text{Terms}_{\Sigma, \kappa}$ where $\kappa$ is the set of terms in $\Sigma \cup \{\bot \}$. (i) $N(S) = \circ$.

The rewriting relation $\rightarrow_G$ on terms of $G$ is defined inductively by: (i) $F s_1 \ldots s_m \rightarrow_G [s_1/x_1, \ldots, s_m/x_m]$ if $F \in \text{dom}(N)$, $\lambda x_1 \ldots x_m.t$ and $s \rightarrow_G s'$ if $s \rightarrow_G s'$, and (ii) $s \rightarrow_G s'$ if $t \rightarrow_G t'$. Here, $[s_1/x_1, \ldots, s_m/x_m]$ the term obtained from $t$ by replacing each $x_i$ with $s_i$. The tree $[G]$ generated by $G$, called the value tree of $G$, is defined as the least upper bound of $\{t^\perp | S \rightarrow_G t\}$ with respect to $\sqsubseteq$ where $t^\perp$ is defined by (i) $t^\perp = t$ if $t$ is a terminal, (ii) $(t_1 t_2)^\perp = t_1^\perp t_2^\perp$ if $t_1^\perp \neq \perp$, and (iii) $t^\perp = \perp$ otherwise. Here, the partial order $\sqsubseteq$ is defined by $t_1 \sqsubseteq t_2$ if $t_2$ is obtained by replacing some of $\perp$’s in $t_1$ with some trees. The value tree $[G]$ is a $(\Sigma \cup \{\perp \})$-labeled ranked tree.

Example 2 (HORS). Let $G_1 = (\Sigma, N, \mathcal{R}, S)$ where $\Sigma = \{a \rightarrow 2, b \rightarrow 1, c \rightarrow 0\}$, $N = \{F \mapsto ((\circ \rightarrow \circ) \rightarrow \circ), B \mapsto ((\circ \rightarrow \circ) \rightarrow \circ \rightarrow \circ), I \mapsto (\circ \rightarrow \circ), S \mapsto \circ\}$, and $\mathcal{R} = \{F \mapsto (\lambda f. \, b \, (a \, (f \, c) \, (F \, (B \, f)))\}, B \mapsto (\lambda f. \, b \, (f \, x)), I \mapsto (\lambda x. \, x), S \mapsto (F \, I)\}$. From the start non-terminal $S$, the reduction proceeds as follows.

\[
S \rightarrow_G F I \rightarrow_G I \, (a \, (I \, c) \, (F \, (B \, I))) \rightarrow_G a \, c \, (F \, (B \, I)) \\
\rightarrow_G a \, c \, (b \, (a \, (b \, c) \, (F \, (B \, (B \, I)))))) \rightarrow_G a \, c \, (b \, (a \, (b \, c) \, (F \, (B \, (B \, I)))))) \\
\rightarrow_G a \, c \, (b \, (a \, (b \, c) \, (F \, (B \, (B \, I)))))) \\
\rightarrow_G \ldots
\]

The value tree $[G_1]$ has a finite path $abab^2 \ldots ab^n ab^n c$ for each $n \in \mathbb{N}$ and also has an infinite path $abab^2 \ldots$.

2.2 Streett Tree Automata

Given a set $X$, the set $B^+(X)$ of positive boolean formulas over $X$, ranged over by $\varphi$, is defined by $\varphi ::= \text{true} \mid \text{false} \mid x \mid \varphi_1 \land \varphi_2 \mid \varphi_1 \lor \varphi_2$ where $x$ ranges over $X$. Given a subset
Streett Automata Model Checking of Higher-Order Recursion Schemes

Y of X, one can calculate the boolean value [φ]_Y of φ by assigning true to the elements of Y and false to those of X \ Y. We say Y satisfies φ if [φ]_Y = true.

Definition 3 (Streett tree automaton [35, 8]). A Streett tree automaton is a tuple \( \mathcal{A} = (\Sigma, Q, \delta, q_0, C) \) where \( \Sigma \) is a ranked alphabet, \( Q \) is a finite set of states, \( \delta \) is a map from \( Q \times \text{dom}(\Sigma) \) to \( \mathcal{B}^+ (\mathbb{N} \times Q) \) called a transition function such that \( \delta(q, a) \in \mathcal{B}^+ (\{1, \ldots, \Sigma(a)\} \times Q) \) for each \( q \) and \( a \), and \( q_0 \in Q \) is a special state called the initial state, and \( C \) is a Streett acceptance condition of the form \( C = \{(E_1, F_1), \ldots, (E_k, F_k)\} \) where \( E_i, F_i \subseteq Q \) for each \( i \).

Given a \( \Sigma \)-labeled ranked tree \( T \), a run-tree of \( \mathcal{A} \) over \( T \) is a \((\text{dom}(T) \times Q)\)-labeled (unranked) tree \( R \) such that (i) \( \varepsilon \in \text{dom}(R) \), (ii) \( R(\varepsilon) = (\varepsilon, q_0) \), (iii) for every \( \pi \in \text{dom}(R) \) with \( R(\pi) = (\xi, q) \) and \( j \in \{j \in \mathbb{N} \mid \pi j \in \text{dom}(R)\} \), there exist \( i \in \mathbb{N} \) and \( q' \in Q \) such that \( R(\pi j) = (\xi i, q') \), and (iv) for every \( \pi \in \text{dom}(R) \) with \( R(\pi) = (\xi, q) \), \( \{i, q'\} \) satisfies \( \delta(q, T(\xi)) \). Let \( \text{Paths}(R) \) be defined by \( \text{Paths}(R) = \{\pi \in \mathbb{N}^\omega \mid \text{every (finite) prefix of } \pi \text{ is in } \text{dom}(R)\} \) and \( \text{Inf}_R : \text{Paths}(R) \rightarrow 2^\mathbb{Q} \) be defined by \( \text{Inf}_R(\pi) = \{q \in Q \mid \text{the state label of } R(\pi) \text{ is } q \text{ for infinitely many } i\} \) where \( \pi_i \) is the prefix of \( \pi \) of length \( i \). A run-tree \( R \) is accepting if for every \( \pi \in \text{Paths}(R) \) and every \( (E_i, F_i) \in C \), \( \text{Inf}_R(\pi) \cap E_i \neq \emptyset \Rightarrow \text{Inf}_R(\pi) \cap F_i = \emptyset \) holds. A Streett automaton \( \mathcal{A} \) accepts a \( \Sigma \)-labeled ranked tree \( T \) if there exists an accepting run-tree of \( \mathcal{A} \) over \( T \).

Example 4 (Streett tree automaton). Let \( A_1 = (\Sigma, Q, \delta, q_0, C) \) be a Streett tree automaton where \( \Sigma = \{a \mapsto 2, b \mapsto 1, c \mapsto 0\} \), \( Q = \{q_0, q_a, q_b\} \), \( \delta(q, a) = (1, q_a) \land (2, q_b) \) for each \( q \), \( \delta(q, b) = (1, q_b) \) for each \( q \), and \( \delta(q, c) = \text{true} \) for each \( q \), and \( C = \{(E_1, F_1)\} \) where \( E_1 = \{q_a\} \) and \( F_1 = \{q_b\} \). This automaton accepts \( \Sigma \)-labeled ranked trees such that for each infinite path, if \( a \) appears infinitely often in it, then \( b \) also appears infinitely often in it.

2.3 Streett Automata Model Checking Problem for HORS

We can now define the Streett automata model checking problem for HORS.

Definition 5 (Streett automata model checking problem for HORS). The Streett model checking problem for HORS is a decision problem to check whether \( \mathcal{G} \) is accepted by \( \mathcal{A} \), for a given HORS \( \mathcal{G} \) and a Streett automaton \( \mathcal{A} \).

The need for Streett automata model checking of HORS naturally arises in the context of verifying liveness properties of higher-order programs. A popular method for verification of temporal properties of programs is to use Vardi’s reduction to fair termination [40], the problem of checking whether all the fair execution sequences of a given program are terminating [5, 2, 27, 41]. Here, a fairness constraint is of the form \( \{(A_1, B_1), \ldots, (A_n, B_n)\} \), which means that if the event \( A_1 \) occurs infinitely often, so does \( B_1 \), for each \( i \). For proving/disproving fair termination of a higher-order functional program, a natural approach is to convert the program to a HORS that generates a tree representing all the possible event sequences, and then check whether the tree contains a fair but infinite event sequence. For example, consider the program:

\[
\text{let} \quad \text{rec} \quad f() = \text{if} \quad *\text{int} < 0 \quad \text{then} \quad (\text{event} \ B; ()) \quad \text{else} \quad (\text{event} \ A; f())
\]

where \(*\text{int}\) represents a non-deterministic integer. It can be converted to the following HORS, whose value tree represents all the possible event sequences.

\[
S \rightarrow F \quad F \rightarrow \text{br} \ (\text{ev}_B \ \text{end}) \ (\text{ev}_A \ F)
\]

Then, the problem of checking that the original program is not terminating with respect to the fairness constraint \( \{(A, B)\} \) is reduced to the problem of checking that the tree
generated by the HORS has a fair infinite path (i.e., an infinite path such that if \( e v_A \) occurs infinitely often, so does \( e v_B \)). In the case of the HORS above, there is no infinite path in which \( e v_B \) occurs infinitely often, from which we can conclude that the original program is fair terminating. Indeed, Watanabe et al. [41] took such an approach for disproving fair termination of functional programs.\(^2\) The resulting decision problem for HORS can naturally be expressed as a Streett model checking problem; in the above case, we can use the Streett automaton \( \mathcal{A} = (\{ br \mapsto 2, ev_A \mapsto 1, ev_B \mapsto 1, \text{end} \mapsto 0 \}, \{ q_0, q_A, q_B \}, \delta, q_0, \{ \{ q_A \}, \{ q_B \} \} ) \) where \( \delta(q, ev_A) = (1, q_A), \delta(q, ev_B) = (1, q_B), \delta(q, \text{br}) = (1, q_0) \land (2, q_0), \) and \( \delta(q, \text{end}) = \text{true} \) for every \( q \in \{ q_0, q_A, q_B \}. \)

As usual [25, 19], we assume that the value tree \( [G] \) of a HORS \( G \) does not contain \( \bot \) in the rest of the paper. Note that this is not a limitation, because any instance of the model checking problem for a HORS \( G \) and a Streett automaton \( \mathcal{A} \) can be reduced to an equivalent one for \( G' \) and \( \mathcal{A}' \) such that \( [G'] \) does not contain \( \bot \).

### 3 A Type System for Streett Automata Model Checking

This section presents an intersection type system (parameterized by a Streett automaton \( \mathcal{A} \)) for Streett automata model checking of HORS, such that a HORS \( G \) is typable in the type system if and only if \( [G] \) is accepted by \( \mathcal{A} \). The type system is obtained by modifying the Kobayashi-Ong type system [19] for parity automata model checking. We prove the correctness of our type system by showing that it is actually an instance of Tsukada and Ong’s type system for model checking Böhm trees [38].

Let \( \mathcal{A} = (\Sigma, Q, \delta, q_0, C) \) be a Streett automaton with \( C = \{ (E_1,F_1), \ldots, (E_k,F_k) \} \). We define the set of *effects* by \( E = 2^{(E_1,F_1),\ldots,(E_k,F_k)} \).\(^3\) Here, \( E_1, \ldots, E_k, F_1, \ldots, F_k \) in effects should be considered just symbols (so that they are different from each other, even if \( E_i \) and \( F_j \) in \( C \) happen to be the same set of states) although they intuitively represent the sets used in \( C \). The set of *prime types*, ranged over by \( \theta \), and the set of *intersection types*, ranged over by \( \tau \), are defined by:

\[
\theta \ (\text{prime types}) ::= q \mid \tau \rightarrow \theta' \quad \tau \ (\text{intersection types}) ::= \{ \{ \theta_i, e_i \} \}_{i \in I}
\]

where \( q \in Q, e_i \in \Sigma \) and \( I \) is a finite index set. Note that \( \{ \{ \theta_i, e_i \} \}_{i \in I} \) is a shorthand for \( \{ \{ \theta_i, e_i \} \mid i \in I \} \). We sometimes write \( (\theta_1, e_1) \land \cdots \land (\theta_k, e_k) \) for \( \{ \{ \theta_i, e_i \} \mid i \in I \} \), and \( \top \) for the empty intersection type \( \emptyset \).

Intuitively, \( q \) is the type of trees accepted from \( q \) by \( \mathcal{A} \) (i.e., by \( \mathcal{A}_q = (\Sigma, Q, \delta, q, C) \)), and \( \{ \{ \theta_i, e_i \} \}_{i \in I} \rightarrow \theta' \) is the type of functions which take an argument that has type \( \theta_i \) for every \( i \in I \), and return a value of type \( \theta' \). Here, \( e_i \), describes what states may/must be visited before the argument is used as a value of type \( \theta_i \). For example, the type \( (q_1, \{ E_1 \}) \land (q_2, \{ F_1 \}) \rightarrow q \) describes the type of functions that take a tree that can be accepted from both \( q_1 \) and \( q_2 \) as an argument, and return a tree of type \( q \). Furthermore, the effect parts \( \{ \{ E_1 \} \} \) and \( \{ F_1 \} \) describe that in an accepting run of \( \mathcal{A}_q \) over the returned tree, the argument can be used as a value of type \( q_1 \) (i.e., visited with state \( q_1 \)) only after visiting states in \( E_1 \), and also used as a value of type \( q_2 \) only after visiting states in \( F_1 \).

---

\(^2\) The actual method in [41] is more complicated due to a combination with predicate abstraction.

\(^3\) One can use \( E = 2^{(E_1,\ldots,E_k,F_1,\ldots,F_k)} \) instead, where \( \sim \) is an equivalence relation defined in the proof of Theorem 8. This improves the time complexity of our algorithm. We use \( E = 2^{(E_1,\ldots,E_k,F_1,\ldots,F_k)} \) here for understandability, however.
A type environment is a set of type bindings of the form \(x : (\theta, e)\) where \(x\) is a variable or a non-terminal, \(\theta\) is a prime type, and \(e\) is an effect. The part \(e\) represents \(when\) \(x\) may be used as a value of type \(\theta\). Note that a type environment may contain multiple bindings for each variable.

The type judgement relation \(\vdash_{\mathcal{A}}\) among type environments, terms and prime types are defined inductively by the typing rules in Figure 1. The operations used in the rules are defined by: \(e_{\text{id}} = 0\), \(\text{Eff}(q) = \{E \in \{E_1, \ldots, E_k, F_1, \ldots, F_k\} \mid q \in E\}\), and \(\Gamma \upharpoonright e = \{x : (\theta, e) \mid (x : (\theta, e)) \in \Gamma\}\) where \(e \circ e' = e \cup e'\).

In the rule (VAR), the effect \(e_{\text{id}}\) indicates that no state has been visited before the use of \(x\). The rule (CONST) is for terminals (i.e., tree constructors); the premise \(\{(i, q_{ij}) \mid i \in \{1, \ldots, n\}, j \in J_i\}\) satisfies \(\delta(q, a)\) implies that a tree of terminals is accepted from \(q\) if \(T_i\) is accepted from \(q_{ij}\) for each \(j \in J_i\). In the rule (APP), each type environment \(\Gamma_i\) is “lifted” by \(e_i\), to reflect the condition (as indicated by the argument type of \(t_i\)) that the argument \(t_i\) is used as a value of type \(\theta_i\) only after the effect \(e_i\) occurs.

\begin{figure}[h]
\centering
\begin{align*}
\{x : (\theta, e_{\text{id}})\} & \vdash_{\mathcal{A}} x : \emptyset & \text{(VAR)} \\
q \in Q & a \in \Sigma & n = \Sigma(a) & \{i, q_{ij}\} \mid i \in \{1, \ldots, n\}, j \in J_i\} \text{ satisfies } \delta(q, a) & e_{ij} = \text{Eff}(q_{ij}) \quad \vdash_{\mathcal{A}} a : \{\{q_{ij}, e_{ij}\}\}_{j \in J_i} \rightarrow \cdots \rightarrow \{(q_{nj}, e_{nj})\}_{j \in J_n} \rightarrow q \quad \text{(CONST)} \\
\emptyset & \vdash_{\mathcal{A}} t_0 : \{\theta_i, e_i\} \mid i \in I \rightarrow \emptyset & \Gamma_i & \vdash_{\mathcal{A}} t_i : \theta_i \text{ for each } i \in I \quad \Gamma_0 \cup \bigcup_{i \in I} (\Gamma_i \upharpoonright e_i) & \vdash_{\mathcal{A}} t_0 t_i : \theta \quad \text{(APP)} \\
\Gamma & \vdash_{\mathcal{A}} x : (\theta_i, e_i) \mid i \in I \rightarrow \theta & \Gamma & \text{ has no bindings for } x \mid I \subseteq J \quad \Gamma & \vdash_{\mathcal{A}} \lambda x.t : \{\theta_i, e_i\} \mid i \in J \rightarrow \emptyset & \text{(ABS)}
\end{align*}
\caption{Typing Rules.}
\end{figure}

\textbf{Example 6} (type judgement). Let \(A_1 = (\Sigma, Q, \delta, q_0, C)\) where \(\Sigma = \{a \rightarrow 2, b \rightarrow 1\}\), \(Q = \{q_0, q_a, q_b\}\), \(C = \{E_1, F_1\}\) with \(E_1 = \{q_a\}\) and \(F_1 = \{q_b\}\), and \(\delta(q, a) = (1, q_a) \land (2, q_a)\) for each \(q\) and \(\delta(q, b) = (1, q_b)\) for each \(q\). Types of the terminals can be determined by the (CONST) rule; for example, one can derive \(\emptyset \vdash_{\mathcal{A}} a : \{q_a, \{E_1\}\} \rightarrow \{q_a, \{E_1\}\} \rightarrow q_0\). By using the typing rules, one can derive a type judgement: \(\{x : \{q_a, \{E_1\}\}, x : \{q_b, \{F_1\}\}\} \vdash_{\mathcal{A}} a \ (b \ x) \ x : q_0\).

\textbf{Remark.} The difference from the Kobayashi-Ong type system [19, 20] is condensed into the definitions of \(E, e_{\text{id}}, \text{Eff}\) and \(\circ\). Indeed, a variant of the Kobayashi-Ong type system for a parity automaton \((\Sigma, Q, \delta, q_0, \Omega)\) is produced by the following definitions: \(E = \{0, 1, \ldots, M\}\) where \(M = \max\{\Omega(q) \mid q \in Q\}\), \(e_{\text{id}} = 0\), \(\text{Eff}(q) = \Omega(q)\), and \(e \circ e' = \max\{e, e'\}\). This variant actually deviates from Kobayashi and Ong’s type system [20] in the way the priorities of visited states are counted in the rules (VAR) and (CONST). The variant is an instance of
Tsukada and Ong’s type system [38], and is also close to the type system of Grellois and Melliès [10, 9].

The typability of a HORS is defined using Streett games.

**Definition 7 (Streett game).** A Streett game is a quadruple \( G = (V_2, V_\forall, E, C) \) where \( V_2 \) and \( V_\forall \) are disjoint sets of vertices (we write \( V = V_2 \cup V_\forall \)), \( E \subseteq V \times V \) is a set of edges and \( C \) is a Streett acceptance condition on vertices. A play on \( G \) from \( v_0 \in V_2 \) is a finite or infinite path from \( v_0 \) of the directed graph \((V, E)\). A play is maximal if it is infinite, or if it is a finite play \( v_0 \ldots v_n \) and there is no \( v_{n+1} \in V \) such that \((v_n, v_{n+1}) \in E\). A (maximal) play is winning either if it is finite and the last vertex of it is in \( V_\forall \), or if it is infinite and it satisfies the acceptance condition \( C \), i.e., for each \((E_i, F_i) \in C\), a vertex in \( F_i \) occurs infinitely often whenever a vertex in \( E_i \) occurs infinitely often. A strategy \( W \) is a partial map from \( V^* V_2 \) to \( V \) that is edge-respecting, i.e., for every \( \bar{v} = v_0 \ldots v_n \in \text{dom}(W) \), \((v_n, W(\bar{v})) \in E\). A play \( \bar{v} = v_0 v_1 \ldots \) follows a strategy \( W \) if for every \( i, v_i \in V_2 \) and \( |\bar{v}| > i \) imply \( W(v_0 \ldots v_i) = v_{i+1} \). A strategy \( W \) is a winning strategy from \( v_0 \in V_2 \) if every play that follows \( W \) does not get stuck (i.e., if \( \bar{v} \in V^* V_2 \) follows \( W \), then \( \bar{v} \in \text{dom}(W) \)), and every maximal play from \( v_0 \) that follows \( W \) is winning.

For a HORS \( \mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S) \), we define the typability game \( G_{\mathcal{G}, A} \) as the Streett game \((V_2, V_\forall, E_2, E_\forall, C^\uparrow)\) where:

- \( V_2 = \{(F, \theta, e) \mid F \in \text{dom}(\mathcal{N}), \theta :: \mathcal{N}(F), \text{ and } e \in E\} \)
- \( V_\forall = \{\Gamma \mid \forall (F : (\theta, e)) \in \Gamma, F \in \text{dom}(\mathcal{N}), \theta :: \mathcal{N}(F), \text{ and } e \in E\} \)
- \( E_2 = \{(F, \theta, e), \Gamma) \in V_2 \times V_\forall \mid \Gamma \vdash_A \mathcal{R}(F) : \theta\} \)
- \( E_\forall = \{(\Gamma, (F, \theta, e)) \in V_\forall \times V_2 \mid (F : (\theta, e)) \in \Gamma\} \)
- \( C^\uparrow = \{(E_1^\uparrow, F_1^\uparrow), \ldots, (E_k^\uparrow, F_k^\uparrow)\} \text{ where } E_i^\uparrow = \{(F, \theta, e) \in V_2 \mid E \in e\} \)

Intuitively, in a position \((F, \theta, e)\), Player tries to show why \( F \) has type \( \theta \) by giving a type environment \( \Gamma \) such that \( \Gamma \vdash_A \mathcal{R}(F) : \theta \); in a position \( \Gamma \), Opponent tries to challenge Player by picking a type binding from \( \Gamma \), and asking why that assumption is valid. A HORS \( \mathcal{G} \) is well-typed, denoted by \( \vdash_A \mathcal{G} \), when \( G_{\mathcal{G}, A} \) has a winning strategy from \((S, q_0, e_{\text{id}})\).

**Theorem 8 (Correctness).** Given a HORS \( \mathcal{G} \) and a Streett automaton \( A \), the value tree of \( \mathcal{G} \) is accepted by \( A \) if and only if \( \vdash_A \mathcal{G} \).

We sketch a proof below;\(^4\) See the longer version [36] for more details.

**Proof.** We are to define a winning condition that instantiates Tsukada and Ong’s type system for model checking Böhm trees [38] so that the resulting type system is equivalent to our type system and the correctness of our type system follows from the correctness of Tsukada and Ong’s type system. A winning condition is a structure \((E, \mathcal{F}, \Omega)\) where \( E \) and \( \mathcal{F} \) are partially ordered sets (we denote both the orders by \( \preceq \)) and \( \Omega \) is a downward-closed subset of \( \mathcal{F} \), equipped with four operations \( \circ : E \times E \to E, \oplus : E \times \mathcal{F} \to \mathcal{F}, \pi : \mathcal{F} \to \mathcal{F}, \text{ and } \setminus : E \times \mathcal{F} \to E \) that satisfy additional requirements.

Let \( \mathcal{E} \) be defined by \( \mathcal{E} = 2^{\{E_1, \ldots, E_k, F_1, \ldots, F_k\}} \). Let \( S_i : \mathcal{E} \to \{-1, 0, 1\} \) for each \( i \in \{1, \ldots, k\} \) be defined by: (i) \( S_i(e) = -1 \) if \( F_i \in e \), (ii) \( S_i(e) = 0 \) if \( E_i \not\in e \) and \( F_i \not\in e \), and (iii) \( S_i(e) = 1 \) otherwise. A preorder \( \preceq \) on \( \mathcal{E} \) is defined by \( e \preceq e' \iff S_i(e) \leq S_i(e') \) for every \( i \). It induces an equivalence relation \( \sim \) on \( \mathcal{E} \) and a partial order \( \preceq \) on \( \mathcal{E}/\sim \). We define a

\(^4\) Here we assume some familiarity with Tsukada and Ong’s type system [38].
Streett Automata Model Checking of Higher-Order Recursion Schemes

The overall structure of the algorithm HorSatS is shown in Figure 2. An **effectless type environment**, denoted by $\Theta$, is a set of type bindings $F : \theta$ where $F$ is a non-terminal and $\theta$ is a prime type. The algorithm starts with a certain initial effectless type environment $\Theta_0$

\[ f(x) = g(O(h(x))) \] to mean that $f(x)$ is bounded by $g(h'(x))$ for some $h'(x)$ with $h'(x) = O(h(x))$. 

---

5 We use the notation $f(x) = g(O(h(x)))$ to mean that $f(x)$ is bounded by $g(h'(x))$ for some $h'(x)$ with $h'(x) = O(h(x))$. 

---

This section proposes HorSatS and HorSatP, new practical algorithms for Streett and parity automata model checking of HORS, respectively.

The type-based characterization in the previous section actually yields a straightforward model checking algorithm, which first constructs the typability game $G_{\mathcal{A}}$ and solves it, as discussed at the end of the previous section. It is, however, impractical since the size of the typability game is huge: $N$-fold exponential for an order-$N$ HORS. This is because the number of intersection types for order-$N$ functions is $N$-fold exponential.
\[ \Theta := \Theta_0 \]

while \( (F(\Theta)) \neq \Theta) \{
    \Theta := F(\Theta)
\}

return whether ConstructGame(\(\Theta\)) has a winning strategy

\textbf{Figure 2} The proposed algorithm HorSatP.

(which will be defined below), and then expands it by repeatedly applying \(F\), until it reaches a fixpoint \(\Theta^\text{fix}\). The algorithm then constructs a subgame of \(G_{\mathcal{G},A}\) consisting of only types occurring in \(\Theta^\text{fix}\) and solves it. The algorithm HorSatP also has exactly the same structure; we just need to adapt \(F\) and ConstructGame for parity games.

We describe the construction of \(\Theta_0\) and the expansion function \(F\) below; it has been inspired by Broadbent and Kobayashi’s HorSat algorithm for trivial automata model checking [3]. Let an input HORS be \(G = (\Sigma, \mathcal{N}, \mathcal{R}, S)\) and an input Streett automaton be \(A = (\Sigma, Q, \delta, q_0, C)\). The initial effectless type environment \(\Theta_0\) is defined by:

\[ \Theta_0 = \{F : \top \rightarrow \cdots \rightarrow \top \rightarrow q \mid F \in \text{dom}(\mathcal{N}), \ m = \text{arity}(\mathcal{N}(F)), \ q \in Q \}. \]

The expansion function \(F\) is defined by:

\[
F(\Theta) = \Theta \cup \{ F : \tau_1 \rightarrow \cdots \rightarrow \tau_m \rightarrow q \mid \mathcal{R}(F) = \lambda x_1, \ldots, x_m. \ t, \\
(\tau_1 \rightarrow \cdots \rightarrow \tau_m \rightarrow q) :: \mathcal{N}(F), \\
\tau_i \subseteq \text{Types}_0(\text{Flow}(x_i)) \text{ for each } i \in \{1, \ldots, m\}, \\
\Gamma \cup \{x_1 : \tau_1, \ldots, x_m : \tau_m\} \vdash_A t : q \\
\text{for some } \Gamma \text{ such that } \forall (G : (\theta, e)) \in \Gamma. (G : \theta) \in \Theta \}.
\]

Here, \(\text{Flow}(x)\) is an overapproximation of the (possibly infinite) set of terms to which \(x\) may be bound in a reduction sequence from \(S\); it can be obtained by a flow analysis algorithm like 0CFA. For a set \(U\) of terms, \(\text{Types}_0(U)\) is defined as \(\{\theta \mid \Gamma \vdash_A u : \theta, u \in U \}, \forall (F : (\theta, e)) \in \Gamma. (F : \theta) \in \Theta\). The notation \(\{x_1 : \tau_1, \ldots, x_m : \tau_m\} \vdash_A t : q\) represents the type environment \(\{x_1 : (\theta_{i,j}, e_{i,j}) \mid i \in \{1, \ldots, m\}, (\theta_{i,j}, e_{i,j}) \in \tau_i\}\).

Finally, ConstructGame(\(\Theta\)) returns a subgame \(G_{\mathcal{G},A,\Theta} \) of \(G_{\mathcal{G},A}\), obtained by restricting \(E_3\) to the following subset:

\[ E'_3 = \{((F, \theta, e), \Gamma) \in V_3 \times V_\nu \mid \Gamma \vdash_A \mathcal{R}(F) : \theta \text{ and } \forall (G : (\theta, e)) \in \Gamma. (G : \theta) \in \Theta \}. \]

The algorithm HorSatP is obtained by (i) replacing the type judgment relation used in the expansion function with that of the Kobayashi-Ong type system, and (ii) modifying ConstructGame(\(\Theta\)) to produce a subgame of the typability game for the Kobayashi-Ong type system. See the longer version [36] for more details.

\textbf{Example 9} (a sample run of the algorithm). Consider a HORS \(G_2 = (\Sigma, \mathcal{N}, \mathcal{R}, S)\) where \(\Sigma = \{a \mapsto 2, b \mapsto 1, c \mapsto 0\}\), \(\mathcal{N} = \{F \mapsto (0 \rightarrow 0), S \mapsto 0\} \) and \(\mathcal{R} = \{F \mapsto (\lambda x.a x (F (b x))), S \mapsto (F c)\} \). and a Streett automaton \(A_2 = (\Sigma, Q, \delta, q_a, C)\) with \(Q = \{q_a, q_b\}\), \(C = \{(E_1,\emptyset)\}\) where \(E_1 = \{q_b\}\), and \(\delta\) is defined by \(\delta(q_a, a) = (1, q_a) \wedge (2, q_a)\) for
each q, δ(q, b) = (1, qb) for each q, and δ(q, c) = true for each q. The automaton A2 accepts trees in which b occurs only finitely often in every path. The initial effectless type environment is Θ0 = {F : ⊤ → qa, F : ⊤ → qb, S : qa, S : qb}. Let Flow(x) = {b^n c | n ≥ 0} (hence Typesθ(Flow(x)) = {qa, qb} for any Θ). The fixpoint calculation proceeds as follows.

Θ1 = F(Θ0) = Θ0 ∪ {F : (qa, ∅) → qa, F : (qb, {E1}) → qb}.
Θ2 = F(Θ1) = Θ1 ∪ {F : (qa, ∅) ∧ (qb, {E1}) → qa, F : (qa, ∅) ∧ (qb, {E1}) → qb}.
Θ3 = F(Θ2) = Θ2.

The game constructed by the algorithm is shown in Figure 3, where:

θF = (qa, ∅) ∧ (qb, {E1}) → qa. θ′F = (qa, ∅) → qa. θ′′F = ⊤ → qa.
ΓF = {F : (θF, ∅)}. Γ′F = {F : (θ′F, ∅)}. Γ′′F = {F : (θ′′F, ∅)}.

As the game has a winning strategy, the algorithm returns “Yes.”

The proposed algorithm is sound and complete, as stated in the following theorems. The soundness (Theorem 10) follows from the fact that ConstructGame(Θ) produces a subgame of \( \mathcal{G}_{\mathcal{A}} \) obtained by restricting only Player’s moves. The completeness is, however, non-trivial, as to why the fixpoint \( \Theta^{\text{fix}} \) is sufficiently large so that Player can win the subgame \( G_{\mathcal{A},\Theta^{\text{fix}}} \) if she can win the whole game \( G_{\mathcal{A},\mathcal{G}} \). Whilst the construction of \( \Theta_0 \) and \( F \) is essentially the same as that of HORSatT algorithm, the completeness proof for HORSatS is much more involved than that for HORSatT.

The proofs below apply to both HORSatS and HORSatP. We use the notations \( e_{id}, \text{Eff} \) and \( \circ \) for this generalization.

\[ \textbf{Theorem 10 (Soundness). If HORSatS (resp. HORSatP) returns “Yes” for an input HORS } \mathcal{G} \text{ and a Streett (resp. parity) automaton } \mathcal{A}, \text{ then } \vdash_{\mathcal{A}} \mathcal{G}. \]

\[ \textbf{Proof.} \text{ Suppose that the algorithm returns “Yes.” Then, the Streett game } G_{\mathcal{A},\Theta^{\text{fix}}} \text{ has a winning strategy } W \text{ from } (S, q_0, e_{id}). \text{ As } G_{\mathcal{A},\Theta^{\text{fix}}} \text{ is a subgame of } G_{\mathcal{A},\mathcal{G}} \text{ obtained by only restricting edges in } E_2, W \text{ is also a winning strategy for } G_{\mathcal{A},\mathcal{G}}. \text{ Thus, we have } \vdash_{\mathcal{A}} \mathcal{G}. \]

\[ \textbf{Theorem 11 (Completeness). Given an input HORS } \mathcal{G} \text{ and a Streett (resp. parity) automaton } \mathcal{A}, \text{ HORSatS (resp. HORSatP) returns “Yes” if } \vdash_{\mathcal{A}} \mathcal{G}. \]

Here we give only a proof sketch. See the longer version [36] for more details.
Proof Sketch. Suppose $\vdash_A G$. By the definition of $\vdash_A G$, there exists a finite memory winning strategy $W$ of the typability game $G_{G,A}$. By adding a rule for unfolding non-terminals (i.e., a rule for deriving $F : (\theta, c) \vdash_A F : \theta$ from $\Gamma \vdash_A R(F)$), the typability of HORS can alternatively be described by the existence of a (possibly infinite) type derivation tree $\Pi_0$ for $S : \rho_0$ such that every infinite path in it satisfies the Streett/parity condition derived from that of $A$.\footnote{This alternative view of the typability follows easily from the definition of the typability game. Grellois and Melliès [9, 10] have indeed chosen such a formalization for parity tree automata model checking.} Such a derivation tree $\Pi_0$ can be constructed based on $W$.

We show that we can transform $\Pi_0$ to another derivation tree $\Pi''_0$ which only uses the types occurring in $\Theta^\text{fix}$ (where $\Theta^\text{fix}$ is the effectless type environment produced by the fixpoint calculation in our algorithm). We first “cut” $\Pi_0$ by stopping unfoldings of non-terminals with a certain threshold for the depth of unfoldings, and replace the type of the non-terminal at each “cut” node with $\top \rightarrow \ldots \rightarrow \top \rightarrow q$, treating $\Gamma \vdash F : \underbrace{\top \rightarrow \ldots \rightarrow \top}_{\text{arity}(F)} \rightarrow q$ as an “axiom”. This axiom corresponds to the initial type environment $\Theta_0$ used in the algorithm. By accordingly reassigning the types in the tree, we have a finite type derivation tree $\Pi'_0$ that uses only the types in $\Theta^\text{fix}$ computed by the algorithm.

Unfortunately, $\Pi'_0$ itself does not represent a winning strategy of $G_{G,A,\Theta^\text{fix}}$ as it uses the types of non-terminals in $\Theta_0$ as axioms. If we choose a sufficiently large number as the threshold for the depth of unfoldings, however, by matching each node of $\Pi'_0$ with a corresponding node in $\Pi_0$, we can reconstruct a valid (in the sense that every infinite path satisfies the Streett/parity condition) infinite derivation tree $\Pi''_0$, by replacing some edges in $\Pi'_0$ with “back edges”. Since $\Pi''_0$ has been obtained by only rearranging edges in $\Pi'_0$, $\Pi''_0$ also contains only types in $\Theta^\text{fix}$. By the correspondence between a valid infinite derivation tree and a winning strategy of the typability game, we obtain a winning strategy $W'$ for the subgame $G_{G,A,\Theta^\text{fix}}$ constructed by $\text{ConstructGame}(\Theta^\text{fix})$. Thus, the algorithm should return “Yes”.

Appendix 8 shows an example of the construction of $\Pi''_0$.

The algorithm runs in time polynomial in the size of HORS if the other parameters (the largest order and arity of terminals/non-terminals in HORS and the automaton) are fixed. 

Here, we assume that, as in [37], the linear-time sub-transitive control flow analysis [11] is used for computing the part $\text{Types}_P(\text{Flow}(x_1))$ in $F$. We also assume that an input HORS is normalized as in [19] so that for each $F \in \text{dom}(N)$, $R(F)$ is of the form $\lambda \vec{x}. c(F_1 \vec{x}_1) \ldots (F_j \vec{x}_j)$ where $0 \leq j$, $F_1, \ldots, F_j$ are non-terminals, $\vec{x}_1, \ldots, \vec{x}_j$ are variables and $c$ is a terminal, a non-terminal or a variable. As an increase of the size of HORS caused by this normalization is linear, it does not affect the time complexity result. Upon those assumptions, (i) The size of a type environment is bounded by $O(P)$ where $P$ is the size of an input HORS, and thus the number of the iteration is bounded by $O(P)$. (ii) Calculation of $F(\Gamma)$ is done in $O(P)$ time. Thus, the fixpoint $\Theta^\text{fix}$ can be calculated in time quadratic in $P$.\footnote{Actually, using the technique of [32], $\Theta^\text{fix}$ can be calculated in linear time.} Since the size of $\Theta^\text{fix}(F)$ is bounded above by a constant (under the fixed-parameter assumption) for each $F$, the size of (the relevant part of) the typability game is linear in $P$. Because we have assumed that the automaton is fixed (which also implies that the index $k$ of a Streett automaton or the largest parity of a parity tree automaton is also fixed), the game can also be solved in time polynomial in $P$. Thus, the whole algorithm runs in polynomial time under the fixed-parameter assumption.
5 Experiments

We have implemented HorSatS and HorSatP, Streett and parity automata model checkers for HORS, respectively, based on the algorithm in Section 4. The implementations use a parity game solver PGSolver [6] as a backend for solving the typability game. Although the typability games solved by HorSatS are Streett games, they are actually converted to parity games and passed to PGSolver; this is because we could not find a practical implementation of a direct algorithm for Streett game solving.

We have conducted two kinds of experiments, as reported below. The first experiment aims to confirm the effectiveness of the proposed fixed-parameter polynomial time algorithm. To this end, we have compared HorSatP with the previous parity automata model checkers APTRecS [7] and TravMC2 [28]. The second experiment aims to evaluate the effectiveness of the direct approach to Streett automata model checking. To this end, we have compared HorSatS with a combination of HorSatP and a conversion from Streett to parity tree automata.

HorSatP vs APTRecS/TravMC2

We have used a benchmark consisting of 97 inputs of parity automata model checking problems, which include all the inputs used in the evaluation of APTRecS and TravMC2 [7, 28], and also new inputs derived from verification of tree processing programs [39, 22]. The experiment was conducted on a laptop computer with an Intel Core i5-6200U CPU and 8GB of RAM. To achieve the best performance of each model checker, we have used Ubuntu 16.04 LTS for APTRecS and HorSatP, and Windows 10 for TravMC2. Figure 4 shows the results. The horizontal axis is the size (the number of symbols in HORS) of an input, and the vertical axis is the elapsed time of each model checker. The points at the upper edge are timed-out runs (runs that took more than 50 seconds).

For 27 tiny inputs (of size less than 20) APTRecS tends to be the fastest. For the remaining 70 inputs, APTRecS, TravMC2, and HorSatP won 6, 3, and 61 of them respectively. The results indicate that HorSatP usually outperforms the existing model checkers except for tiny inputs. In particular, HorSatP can handle a number of large inputs for which APTRecS and TravMC2 timed-out.
Direct vs Indirect Approaches to Streett Automata Model Checking of HORS

We have compared HorSatS with an indirect approach, which first converts an input Streett automaton to a parity automaton by means of the IAR construction [4, 8] and then performs parity automata model checking using HorSatP. The experiment was conducted on a desktop computer with an Intel Core i7-2600 CPU and 8GB of RAM. The OS was Ubuntu 16.04 LTS. We have used two benchmark sets. The first one has been prepared by the authors, hand-made or program-generated. The second one has been taken from Watanabe et al.’s fair non-termination verification tool for functional programs [41], with a slight modification to increase \(k\) (the number of pairs in Streett conditions); Watanabe et al.’s original tool supported only the case for \(k = 1\). To evaluate the dependency on \(k\), we have tested some of the inputs for different values of \(k\). The results are shown in Figure 5. The left figure shows the elapsed time of both approaches for several inputs. The horizontal axis is that of the direct approach, and the vertical axis is that of the indirect approach. (Thus, plots above the line \(y = x\) indicate instances for which the direct approach outperformed the indirect one.) The right figure shows the elapsed time for two series of inputs with the same structure but different values of \(k\). The results suggest that the direct approach often outperforms the indirect approach. In particular, the direct approach seems to be noticeably more scalable with respect to \(k\).

6 Related Work

As already mentioned, our type system for Streett automata model checking of HORS presented in Section 3 is a variant of the Kobayashi-Ong type system [19] for parity tree automata model checking, and may also be viewed as an instance of Tsukada and Ong’s type system [38], an extension/generalization of the Kobayashi-Ong type system. Our main contribution in this respect is the specific design of “effects” suitable for Streett automata model checking. A naive approach would have been to use a set of states (rather than a set consisting of \(E_i, F_i\)) as an effect; that would suffer from the \((N + 1)\)-fold exponential time
complexity in the size of the automaton, as opposed to $N$-fold exponential time complexity obtained in the last paragraph of Section 3.

Our algorithms HorSatP and HorSatS are the first practical algorithms for Streett or parity tree automata model checking of HORS which run in time polynomial in the size of HORS (under the fixed-parameter assumption). The previous algorithms APTRecS [7] and TravMC2 [28] for parity tree automata model checking did not satisfy that property. The advantage of the new algorithms has been confirmed also through experiments. For trivial automata model checking, several fixed-parameter polynomial time algorithms have been known, including GTRecS [16], HorSat/HorSatT [3], and Preface [31]. Our algorithms are closest and similar to HorSatT algorithm, although the correctness proof for our new algorithms is much more involved.

Model checking of HORS has been applied to automated verification of higher-order programs [15, 21, 17, 26, 24, 23, 41]. In particular, parity/Streett automata model checking has been applied to liveness verification [24, 7, 41]. Among others, Watanabe et al. [41] reduced the problem of disproving fair termination of functional programs (which is obtained from general liveness verification problems through Vardi’s reduction [40]) to Streett automata model checking of HORS, and used an indirect approach to solving the latter by a further reduction to parity automata model checking of HORS. As confirmed through experiments, our direct approach to Streett automata model checking often outperforms their indirect approach.

7 Conclusion

We have proposed a type system and an algorithm for Streett automata model checking of HORS. The main contributions are twofold. First, ours is the first type system and algorithm that can directly be applied to Streett automata model checking of HORS; we have confirmed the advantage of the direct approach through experiments. Secondly, our algorithm HorSatS and its variant HorSatP for parity automata model checking are the first practical algorithms for Streett/parity automata model checking of HORS that run in time polynomial in the size of HORS, under the fixed-parameter assumption. We have also confirmed through experiments that HorSatP often outperforms the previous parity automata model checkers for HORS. Future work includes further optimizations of Streett/parity automata model checkers.

Acknowledgments. We would like to thank anonymous referees for useful comments.

References


Oliver Friedmann and Martin Lange. PGSolver. Available at https://github.com/tcsprojects/pgsolver.


Let $G_2$ and $A_2$ be the HORS and the Streett automaton in Example 9 respectively. Let $W$ be the memoryless winning strategy of $G_2, A_2$ defined by $W((S, q_a, \emptyset)) = \Gamma_1$ and $W((F, \hat{\theta}_F, \emptyset)) = \Gamma_1$ where $\theta_F = (q_a, \emptyset) \land (q_a, \{E_1\}) \land (q_a, \{E_1\}) \rightarrow q_a$ and $\Gamma_1 = \{ F : (\hat{\theta}_F, \emptyset) \}$. We write $F : \theta$ instead of $F : (\theta, \emptyset)$ in type environments. A derivation tree $\Pi_0$ that corresponds to $W$ is as follows:
We construct $\Pi'_0$ by “cutting” the derivation tree at a certain threshold of depth, and then reassigning the types in it. Here, we choose to “cut” at the uppermost unfolding shown in the above tree. The resulting tree $\Pi'_0$ looks like:

\[
\begin{align*}
\vdots
\end{align*}
\]

Here, $\theta_F = (q_a, \emptyset) \land (q_b, \{E_1\}) \rightarrow q_a$ and $\theta_{a1} = (q_a, \emptyset) \rightarrow q_a$.

The uppermost unfolding has been replaced by the axiom. At the next unfolding below it, $F$ is given the type $(q_a, \emptyset) \rightarrow q_a$. In the body of this $F$, $x$ should have type $q_a$ because it is used as an argument of $a$, which has type $(q_a, \emptyset) \rightarrow (q_a, \emptyset) \rightarrow q_a$. On the other hand, $x$ in $b \ x$ need not have type $q_b$, since $b \ x$ is not typed in the derivation. Thus, the type
(qa, ∅) \rightarrow qa is assigned to \( F \). By repeating this kind of argument for other unfoldings of \( F \), we update the type of the other occurrences of \( F \) to \( \theta_F \). Note that \( \Pi'_0 \) assigns only the types in \( \Theta^{\text{fix}} \) to non-terminals. In fact, the types \( \top \rightarrow qa \), \( (qa, \emptyset) \rightarrow qa \), and \( \theta_F \) assigned to \( F \) at the axiom, the first unfolding (counted from the top), and the second unfolding respectively belong to \( \Theta_0 \), \( F(\Theta_0) \), and \( F^2(\Theta_0) \).

Notice that, in \( \Pi'_0 \), there are two unfolding nodes (the second and third unfolding nodes) labeled by the same judgment \( \{ F : \theta_F \} \vdash F : \theta_F \). (Note that this is not a coincidence; since there are only finitely many different type judgments, if the threshold for the number of unfoldings is sufficiently large, then there always exist such two nodes in each (sufficiently long) path of \( \Pi'_0 \).) By introducing a “back edge” between them, the following derivation tree \( \Pi''_0 \) is obtained.

\[
\begin{array}{c}
\{ x : qa \} \vdash a \ x : (qa, \emptyset) \rightarrow qa \\
(\text{UNFOLD}) \\
\{ F : \theta_F \} \vdash F : \theta_F \\
\{ x : (qa, \emptyset) \} \vdash b \ x : qa \\
\{ F : \theta_F, x : (qa, \emptyset) \} \vdash F \ (b x) : qa \\
\{ F : \theta_F, x : (qa, \emptyset) \} \vdash a \ x \ (F \ (b x)) : qa \\
\{ F : \theta_F \} \vdash \lambda x. a \ x \ (F \ (b x)) : \theta_F \\
\{ F : \theta_F \} \vdash F \ c : qa \\
\{ S : qa \} \vdash S : qa
\end{array}
\]

(UNFOLD)

Note that \( \Pi''_0 \) uses only types that occur in \( \Theta^{\text{fix}} \). Furthermore, this tree is a valid infinite derivation tree; indeed, it corresponds to the winning strategy of the subgame \( G_{G_2, A_2, \Theta^{\text{fix}}} \) shown in Figure 3.