Streett Automata Model Checking of Higher-Order Recursion Schemes*

Ryota Suzuki[1], Koichi Fujima[2], Naoki Kobayashi[3], and Takeshi Tsukada[4]

1 **The University of Tokyo, Tokyo, Japan**
rsuzuki@kb.is.s.u-tokyo.ac.jp
2 **The University of Tokyo, Tokyo, Japan**
kfujima@kb.is.s.u-tokyo.ac.jp
3 **The University of Tokyo, Tokyo, Japan**
koba@kb.is.s.u-tokyo.ac.jp
4 **The University of Tokyo, Tokyo, Japan**
tsukada@kb.is.s.u-tokyo.ac.jp

Abstract

We propose a practical algorithm for Streett automata model checking of higher-order recursion schemes (HORS), 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

* This work was supported by JSPS Kakenhi 15H05706.

is equi-expressive to the modal μ-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 $\mathcal{G}$ is accepted by a Streett automaton $\mathcal{A}$ if and only if $\mathcal{G}$ is typable in the type system parameterized by $\mathcal{A}$. 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 $\widetilde{x}$ for a sequence $x_1, x_2, \ldots, x_m$ (for some m). Given a set Σ of symbols (which we call *terminals*), a Σ-*labeled (unranked) tree* is a partial map T from $\{1, \ldots, N\}^*$ (for some fixed $N \in \mathbb{N}$) to Σ such that if $\pi i \in dom(T)$, then $\pi \in dom(T)$ and $\pi j \in dom(T)$ for all $1 \leq j \leq i$. If Σ is a *ranked alphabet* (i.e., a map from terminals to natural numbers), a Σ-*labeled ranked tree* T is a $dom(\Sigma)$-labeled tree such that for each $\pi \in dom(T)$, $\{i \mid \pi i \in dom(T)\} = \{1, \ldots, \Sigma(T(\pi))\}$.

2.1 Higher-Order Recursion Schemes

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

▶ **Definition 1** (higher-order recursion scheme)**.** A *higher-order recursion scheme* (HORS) is a quadruple $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ where: (i) Σ is a ranked alphabet. (ii) $\mathcal{N}$ 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 $\lambda\widetilde{x}.t$, where t does not include λ-abstractions and has the sort $\circ$. It is required that for each $F \in dom(\mathcal{N})$, $\lambda\widetilde{x}.t \in \mathbf{Terms}_{\mathcal{N} \cup \{a : \circ^{\Sigma(a)} \to \circ \mid a \in dom(\Sigma)\}, \mathcal{N}(F)}$.
$S \in dom(\mathcal{N})$ is a special non-terminal such that $\mathcal{N}(S) = \circ$.

The *rewriting relation* $\longrightarrow_{\mathcal{G}}$ on terms of $\mathcal{G}$ is defined inductively by: (i) $F\ s_1 \ldots s_m \longrightarrow_{\mathcal{G}} [s_1/x_1, \ldots, s_m/x_m]t$ if $\mathcal{R}(F) = \lambda x_1 \ldots x_m.t$, (ii) $s\ t \longrightarrow_{\mathcal{G}} s'\ t$ if $s \longrightarrow_{\mathcal{G}} s'$, and (iii) $s\ t \longrightarrow_{\mathcal{G}} s\ t'$ if $t \longrightarrow_{\mathcal{G}} t'$. Here, $[s_1/x_1, \ldots, s_m/x_m]t$ is the term obtained from t by replacing each x_i with s_i. The tree $[\![\mathcal{G}]\!]$ generated by $\mathcal{G}$, called the *value tree* of $\mathcal{G}$, is defined as the least upper bound of $\{t^\perp \mid S \longrightarrow_{\mathcal{G}}^* t\}$ 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 $[\![\mathcal{G}]\!]$ is a $(\Sigma \cup \{\perp \mapsto 0\})$-labeled ranked tree.

▶ **Example 2** (HORS)**.** Let $\mathcal{G}_1 = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ where $\Sigma = \{\mathtt{a} \mapsto 2,\ \mathtt{b} \mapsto 1,\ \mathtt{c} \mapsto 0\}$, $\mathcal{N} = \{F \mapsto ((\circ \to \circ) \to \circ),\ B \mapsto ((\circ \to \circ) \to \circ \to \circ),\ I \mapsto (\circ \to \circ),\ S \mapsto \circ\}$, and $\mathcal{R} = \{F \mapsto (\lambda f.\ f\ (\mathtt{a}\ (f\ \mathtt{c})\ (F\ (B\ f)))),\ B \mapsto (\lambda f x.\ \mathtt{b}\ (f\ x)),\ I \mapsto (\lambda x.\ x),\ S \mapsto (F\ I)\}$. From the start non-terminal S, the reduction proceeds as follows.

$$S \longrightarrow_{\mathcal{G}_1} F\ I \longrightarrow_{\mathcal{G}_1} I\ (\mathtt{a}\ (I\ \mathtt{c})\ (F\ (B\ I))) \longrightarrow_{\mathcal{G}_1}^* \mathtt{a}\ \mathtt{c}\ (F\ (B\ I))$$
$$\longrightarrow_{\mathcal{G}_1}^* \mathtt{a}\ \mathtt{c}\ (\mathtt{b}\ (\mathtt{a}\ (\mathtt{b}\ \mathtt{c})\ (F\ (B\ (B\ I))))) \longrightarrow_{\mathcal{G}_1} \ldots$$

The value tree $[\![\mathcal{G}_1]\!]$ has a finite path $\mathtt{abab}^2 \ldots \mathtt{ab}^n\mathtt{ab}^n\mathtt{c}$ for each $n \in \mathbb{N}$ and also has an infinite path $\mathtt{abab}^2\mathtt{ab}^3 \ldots$.

2.2 Streett Tree Automata

Given a set X, the set $\mathcal{B}^+(X)$ of *positive boolean formulas* over X, ranged over by φ, is defined by $\varphi ::= \mathbf{true} \mid \mathbf{false} \mid x \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2$ where x ranges over X. Given a subset

Y of X, one can calculate the boolean value $[\![\varphi]\!]_Y$ of φ by assigning **true** to the elements of Y and **false** to those of $X \setminus Y$. We say Y *satisfies* φ if $[\![\varphi]\!]_Y = \mathbf{true}$.

▶ **Definition 3** (Streett tree automaton [35, 8])**.** A *Streett tree automaton* is a tuple $\mathcal{A} = (\Sigma, Q, \delta, q_0, \mathcal{C})$ where Σ is a ranked alphabet, Q is a finite set of *states*, δ is a map from $Q \times dom(\Sigma)$ 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 $\mathcal{C}$ is a *Streett acceptance condition* of the form $\mathcal{C} = \{(E_1, F_1), \ldots, (E_k, F_k)\}$ where $E_i, F_i \subseteq Q$ for each i. Given a Σ-labeled ranked tree T, a *run-tree* of $\mathcal{A}$ over T is a $(dom(T) \times Q)$-labeled (unranked) tree R such that (i) $\varepsilon \in dom(R)$, (ii) $R(\varepsilon) = (\varepsilon, q_0)$, (iii) for every $\pi \in dom(R)$ with $R(\pi) = (\xi, q)$ and $j \in \{j \in \mathbb{N} \mid \pi j \in dom(R)\}$, there exist $i \in \mathbb{N}$ and $q' \in Q$ such that $R(\pi j) = (\xi i, q')$, and (iv) for every $\pi \in dom(R)$ with $R(\pi) = (\xi, q)$, $\{(i, q') \mid {}^{\exists} j.\ R(\pi j) = (\xi i, q')\}$ satisfies $\delta(q, T(\xi))$. Let $Paths(R)$ be defined by $Paths(R) = \{\pi \in \mathbb{N}^\omega \mid \text{every (finite) prefix of } \pi \text{ is in } dom(R)\}$ and $Inf_R : Paths(R) \to 2^Q$ be defined by $Inf_R(\pi) = \{q \in Q \mid \text{the state label of } R(\pi_i) \text{ is } q \text{ for infinitely many } i\}$ where π_i is the prefix of π of length i. A run-tree R is *accepting* if for every $\pi \in Paths(R)$ and every $(E_i, F_i) \in \mathcal{C}$, $(Inf_R(\pi) \cap E_i \neq \emptyset \Rightarrow Inf_R(\pi) \cap F_i \neq \emptyset)$ holds. A Streett automaton $\mathcal{A}$ *accepts* a Σ-labeled ranked tree T if there exists an accepting run-tree of $\mathcal{A}$ over T.

▶ **Example 4** (Streett tree automaton)**.** Let $\mathcal{A}_1 = (\Sigma, Q, \delta, q_0, \mathcal{C})$ be a Streett tree automaton where $\Sigma = \{\mathtt{a} \mapsto 2,\ \mathtt{b} \mapsto 1,\ \mathtt{c} \mapsto 0\}$, $Q = \{q_0, q_\mathtt{a}, q_\mathtt{b}\}$, $\delta(q, \mathtt{a}) = (1, q_\mathtt{a}) \wedge (2, q_\mathtt{a})$ for each q, $\delta(q, \mathtt{b}) = (1, q_\mathtt{b})$ for each q, and $\delta(q, \mathtt{c}) = \mathbf{true}$ for each q, and $\mathcal{C} = \{(E_1, F_1)\}$ where $E_1 = \{q_\mathtt{a}\}$ and $F_1 = \{q_\mathtt{b}\}$. This automaton accepts Σ-labeled ranked trees such that for each infinite path, if $\mathtt{a}$ appears infinitely often in it, then $\mathtt{b}$ 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 tree 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_i occurs infinitely often, so does B_i, 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:

$$\textbf{let rec } f() = \textbf{if } *_{\mathrm{int}} < 0 \textbf{ then } (\textbf{event } B;\ ()) \textbf{ else } (\textbf{event } A;\ f())$$

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

$$S \to F \qquad F \to \mathtt{br}\ (\mathtt{ev}_B\ \mathtt{end})\ (\mathtt{ev}_A\ F).$$

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 $\mathtt{ev}_A$ occurs infinitely often, so does $\mathtt{ev}_B$). In the case of the HORS above, there is no infinite path in which $\mathtt{ev}_B$ 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 $\mathcal{A} = (\{\mathtt{br} \mapsto 2, \mathtt{ev}_A \mapsto 1, \mathtt{ev}_B \mapsto 1, \mathtt{end} \mapsto 0\}, \{q_0, q_A, q_B\}, \delta, q_0, \{(\{q_A\}, \{q_B\})\})$ where $\delta(q, \mathtt{ev}_A) = (1, q_A)$, $\delta(q, \mathtt{ev}_B) = (1, q_B)$, $\delta(q, \mathtt{br}) = (1, q_0) \wedge (2, q_0)$, and $\delta(q, \mathtt{end}) = \mathbf{true}$ for every $q \in \{q_0, q_A, q_B\}$.

As usual [25, 19], we assume that the value tree $[\![\mathcal{G}]\!]$ of a HORS $\mathcal{G}$ 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 $\mathcal{G}$ and a Streett automaton $\mathcal{A}$ can be reduced to an equivalent one for $\mathcal{G}'$ and $\mathcal{A}'$ such that $[\![\mathcal{G}']\!]$ 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 $\mathcal{G}$ is typable in the type system if and only if $[\![\mathcal{G}]\!]$ 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 $\mathcal{A} = (\Sigma, Q, \delta, q_0, \mathcal{C})$ be a Streett automaton with $\mathcal{C} = \{(E_1, F_1), \ldots, (E_k, F_k)\}$. We define the set of *effects* by $\mathbb{E} = 2^{\{E_1, \ldots, E_k, F_1, \ldots, F_k\}}$.[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 $\mathcal{C}$ happen to be the same set of states) although they intuitively represent the sets used in $\mathcal{C}$. The set of *prime types*, ranged over by θ, and the set of *intersection types*, ranged over by τ, are defined by:

$$\theta \text{ (prime types)} ::= q \mid \tau \to \theta' \qquad \tau \text{ (intersection types)} ::= \{(\theta_i, e_i)\}_{i \in I}$$

where $q \in Q$, $e_i \in \mathbb{E}$ 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 $(\theta_1, e_1) \wedge \cdots \wedge (\theta_k, e_k)$ for $\{(\theta_1, e_1), \ldots, (\theta_k, e_k)\}$, and $\top$ for the empty intersection type $\emptyset$.

Intuitively, q is the type of trees accepted from q by $\mathcal{A}$ (i.e., by $\mathcal{A}_q = (\Sigma, Q, \delta, q, \mathcal{C})$), and $\{(\theta_i, e_i)\}_{i \in I} \to \theta'$ is the type of functions which take an argument that has type θ_i for every $i \in I$, and return a value of type θ'. Here, e_i describes what states may/must be visited before the argument is used as a value of type θ_i. For example, the type $(q_1, \{E_1\}) \wedge (q_2, \{F_1\}) \to q$ 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 $\mathbb{E} = 2^{\{E_1, \ldots, E_k, F_1, \ldots, F_k\}}/\sim$ instead, where $\sim$ is an equivalence relation defined in the proof of Theorem 8. This improves the time complexity of our algorithm. We use $\mathbb{E} = 2^{\{E_1, \ldots, E_k, F_1, \ldots, F_k\}}$ here for understandability, however.

$$\frac{}{\{x : (\theta, e_{\mathbf{id}})\} \vdash_{\mathcal{A}} x : \theta} \quad \text{(Var)}$$

$$\frac{q \in Q \qquad a \in \Sigma \qquad n = \Sigma(a) \qquad \{(i, q_{ij}) \mid i \in \{1, \ldots, n\}, j \in J_i\} \text{ satisfies } \delta(q, a) \qquad e_{ij} = \mathit{Eff}(q_{ij})}{\emptyset \vdash_{\mathcal{A}} a : \{(q_{1j}, e_{1j})\}_{j \in J_1} \to \cdots \to \{(q_{nj}, e_{nj})\}_{j \in J_n} \to q} \quad \text{(Const)}$$

$$\frac{\Gamma_0 \vdash_{\mathcal{A}} t_0 : \{(\theta_i, e_i)\}_{i \in I} \to \theta \qquad \Gamma_i \vdash_{\mathcal{A}} t_1 : \theta_i \text{ for each } i \in I}{\Gamma_0 \cup \bigcup_{i \in I} (\Gamma_i \Uparrow e_i) \vdash_{\mathcal{A}} t_0\ t_1 : \theta} \quad \text{(App)}$$

$$\frac{\Gamma \cup \{x : (\theta_i, e_i) \mid i \in I\} \vdash_{\mathcal{A}} t : \theta \qquad \Gamma \text{ has no bindings for } x \qquad I \subseteq J}{\Gamma \vdash_{\mathcal{A}} \lambda x.t : \{(\theta_i, e_i)\}_{i \in J} \to \theta} \quad \text{(Abs)}$$

Figure 1 Typing Rules.

We say that a prime type θ is a *refinement* of a sort κ, written $\theta :: \kappa$, when it is derivable from the following rules: (i) For each $q \in Q$, $q :: \circ$, and (ii) $(\{(\theta_i, e_i)\}_{i \in I} \to \theta) :: (\kappa_0 \to \kappa_1)$ if $\theta_i :: \kappa_0$ for all $i \in I$ and $\theta :: \kappa_1$. We say a prime type θ is *well-formed* if $\theta :: \kappa$ for some κ. We consider only well-formed prime types below.

A *type environment* is a set of type bindings of the form $x : (\theta, e)$ where x is a variable or a non-terminal, θ is a prime type, and e is an effect. The part e represents *when* x may be used as a value of type θ. 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: $e_{\mathbf{id}} = \emptyset$, $\mathit{Eff}(q) = \{E \in \{E_1, \ldots, E_k, F_1, \ldots, F_k\} \mid q \in E\}$, and $\Gamma \Uparrow e = \{x : (\theta, e \circ e') \mid (x : (\theta, e')) \in \Gamma\}$ where $e \circ e' = e \cup e'$.

In the rule (Var), the effect $e_{\mathbf{id}}$ 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 $a\,T_1 \cdots T_n$ is accepted from q if T_i is accepted from q_{ij} for each $j \in J_i$, hence the type of a in the conclusion. In addition, the effect $e_{ij} = \mathit{Eff}(q_{ij})$ reflects the fact that the state q_{ij} has been visited. In the rule (App), each type environment Γ_i is "lifted" by e_i, to reflect the condition (as indicated by the argument type of t_0) that the argument t_1 is used as a value of type θ_i only after the effect e_i occurs. In the rule (Abs) for λ-abstractions, we allow weakening on the types of x.

▶ **Example 6** (type judgement). Let $\mathcal{A}_1 = (\Sigma, Q, \delta, q_0, \mathcal{C})$ where $\Sigma = \{\mathtt{a} \mapsto 2,\ \mathtt{b} \mapsto 1\}$, $Q = \{q_0, q_\mathtt{a}, q_\mathtt{b}\}$, $\mathcal{C} = \{(E_1, F_1)\}$ with $E_1 = \{q_\mathtt{a}\}$ and $F_1 = \{q_\mathtt{b}\}$, and $\delta(q, \mathtt{a}) = (1, q_\mathtt{a}) \wedge (2, q_\mathtt{a})$ for each q and $\delta(q, \mathtt{b}) = (1, q_\mathtt{b})$ for each q. Types of the terminals can be determined by the (Const) rule; for example, one can derive $\emptyset \vdash_{\mathcal{A}} \mathtt{a} : (q_\mathtt{a}, \{E_1\}) \to (q_\mathtt{a}, \{E_1\}) \to q_0$. By using the typing rules, one can derive a type judgement: $\{x : (q_\mathtt{a}, \{E_1\}),\ x : (q_\mathtt{b}, \{F_1\})\} \vdash_{\mathcal{A}} \mathtt{a}\ (\mathtt{b}\ x)\ x : q_0$.

▶ Remark. The difference from the Kobayashi-Ong type system [19, 20] is condensed into the definitions of $\mathbb{E}$, $e_{\mathbf{id}}$, *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: $\mathbb{E} = \{0, 1, \ldots, M\}$ where $M = \max\{\Omega(q) \mid q \in Q\}$, $e_{\mathbf{id}} = 0$, $\mathit{Eff}(q) = \Omega(q)$, 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 $G = (V_\exists, V_\forall, E, \mathcal{C})$ where $V_\exists$ and $V_\forall$ are disjoint sets of vertices (we write $V = V_\exists \cup V_\forall$), $E \subseteq V \times V$ is a set of edges and $\mathcal{C}$ is a Streett acceptance condition on vertices. A *play* on G from $v_0 \in V_\exists$ 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 $v_0 \dots v_n$ 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 $\mathcal{C}$, i.e., for each $(E_i, F_i) \in \mathcal{C}$, a vertex in F_i occurs infinitely often whenever a vertex in E_i occurs infinitely often. A *strategy* $\mathcal{W}$ is a partial map from $V^*V_\exists$ to V that is edge-respecting, i.e., for every $\widetilde{v} = v_0 \dots v_n \in dom(\mathcal{W})$, $(v_n, \mathcal{W}(\widetilde{v})) \in E$. A play $\widetilde{v} = v_0 v_1 \dots$ *follows* a strategy $\mathcal{W}$ if for every i, $v_i \in V_\exists$ and $|\widetilde{v}| > i$ imply $\mathcal{W}(v_0 \dots v_i) = v_{i+1}$. A strategy $\mathcal{W}$ is a *winning strategy* from $v_0 \in V_\exists$ if every play that follows $\mathcal{W}$ does not get stuck (i.e., if $\widetilde{v} \in V^*V_\exists$ follows $\mathcal{W}$, then $\widetilde{v} \in dom(\mathcal{W})$), and every maximal play from v_0 that follows $\mathcal{W}$ is winning.

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

$$
\begin{array}{lcl}
V_\exists & = & \{(F, \theta, e) \mid F \in dom(\mathcal{N}),\ \theta :: \mathcal{N}(F),\ \text{and } e \in \mathbb{E}\} \\
V_\forall & = & \{\Gamma \mid {}^\forall(F : (\theta, e)) \in \Gamma.\ F \in dom(\mathcal{N}),\ \theta :: \mathcal{N}(F),\ \text{and } e \in \mathbb{E}\} \\
E_\exists & = & \{((F, \theta, e), \Gamma) \in V_\exists \times V_\forall \mid \Gamma \vdash_{\mathcal{A}} \mathcal{R}(F) : \theta\} \\
E_\forall & = & \{(\Gamma, (F, \theta, e)) \in V_\forall \times V_\exists \mid (F : (\theta, e)) \in \Gamma\} \\
\mathcal{C}^\uparrow & = & \{(E_1^\uparrow, F_1^\uparrow), \dots, (E_k^\uparrow, F_k^\uparrow)\} \text{ where } E^\uparrow = \{(F, \theta, e) \in V_\exists \mid E \in e\}
\end{array}
$$

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

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

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 $(\mathbb{E}, \mathbb{F}, \Omega)$ where $\mathbb{E}$ and $\mathbb{F}$ are partially ordered sets (we denote both the orders by $\preceq$) and Ω is a downward-closed subset of $\mathbb{F}$, equipped with four operations $\circ : \mathbb{E} \times \mathbb{E} \to \mathbb{E}$, $\circledast : \mathbb{E} \times \mathbb{F} \to \mathbb{F}$, $\pi : \mathbb{E}^\omega \to \mathbb{F}$, and $\setminus : \mathbb{E} \times \mathbb{E} \to \mathbb{E}$ that satisfy additional requirements.

Let $\mathcal{E}$ be defined by $\mathcal{E} = 2^{\{E_1, \dots, E_k, F_1, \dots, F_k\}}$. Let $\mathcal{S}_i : \mathcal{E} \to \{-1, 0, 1\}$ for each $i \in \{1, \dots, k\}$ be defined by: (i) $\mathcal{S}_i(e) = -1$ if $F_i \in e$, (ii) $\mathcal{S}_i(e) = 0$ if $E_i \notin e$ and $F_i \notin e$, and (iii) $\mathcal{S}_i(e) = 1$ otherwise. A preorder $\preceq$ on $\mathcal{E}$ is defined by $e \preceq e' \overset{\text{def}}{\Leftrightarrow} \mathcal{S}_i(e) \leq \mathcal{S}_i(e')$ 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].

winning condition $(\mathbb{E}, \mathbb{F}, \Omega)$ that represents a Streett condition $\mathcal{C} = \{(E_1, F_1), \ldots, (E_k, F_k)\}$ as follows: $\mathbb{E} = \mathcal{E}/\sim$, $\mathbb{F} = \{\mathbf{e}, \mathbf{o}\}$ with $\mathbf{e} \preceq \mathbf{o}$, and $\Omega = \{\mathbf{e}\}$. The required operations are defined by $e \circ e' = e \cup e'$, $e \circledast f = f$, $\pi(\widetilde{e}) = \mathbf{e}$ if and only if $\widetilde{e}$ satisfies the Streett acceptance condition, and $e \backslash e' = \bigvee\{d' \mid e \circ d' \preceq e'\}$ where $\bigvee\{d_1, \ldots, d_n\} = d_1 \vee \cdots \vee d_n$ with $e \vee e' = ((nf(e))^{(E)} \cup (nf(e'))^{(E)}) \cup (e^{(F)} \cap e'^{(F)})$. Here, $nf(e)$ is a minimal set such that $e \sim nf(e)$, $e^{(E)} = e \cap \{E_1, \ldots, E_k\}$, and $e^{(F)} = e \cap \{F_1, \ldots, F_k\}$.

It is known that for a HORS $\mathcal{G}$, there is a $\lambda\mathbf{Y}$-calculus term $T_{\mathcal{G}}$ whose Böhm tree is identical to the value tree $[\![\mathcal{G}]\!]$ of $\mathcal{G}$ [33]. By comparing the two type systems, it can be checked that $\vdash_{\mathcal{A}} \mathcal{G} \Leftrightarrow \Gamma_{\mathcal{A}} \vdash_{\mathbf{TO}} T_{\mathcal{G}} : q_0$, where $\vdash_{\mathbf{TO}}$ is the type judgement of $\lambda\mathbf{Y}$-term by Tsukada and Ong and $\Gamma_{\mathcal{A}}$ is the set of all type bindings (for terminals) that can be obtained by our (Const) rule. By the transfer theorem (Theorem 18 in [38]) by Tsukada and Ong, this judgement is equivalent to a type-checking game over a Böhm tree (written $\Gamma_{\mathcal{A}} \models \mathrm{BM}(T_{\mathcal{G}}) : q_0$) in that $\Gamma_{\mathcal{A}} \vdash_{\mathbf{TO}} T_{\mathcal{G}} : q_0$ if and only if this game is winning. Moreover, this game is winning if and only if $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$, which concludes the theorem. ◀

By a discussion similar to [19], the number of the edges of the typability game $G_{\mathcal{G},\mathcal{A}}$ is bounded by $|\mathcal{N}| \cdot \mathbf{exp}_N(O(A|\mathbb{E}||Q|))$ for $N \geq 2$ and $|\mathcal{N}| \cdot 2^{O((A|\mathbb{E}||Q|)^2)}$ for $N = 1$,[5] where $N = \max\{ord(\mathcal{N}(F)) \mid F \in dom(\mathcal{N})\}$ and $A = \max\{arity(\mathcal{N}(F)) \mid F \in dom(\mathcal{N})\}$, and $\mathbf{exp}_n$ is defined by $\mathbf{exp}_0(x) = x$ and $\mathbf{exp}_{n+1}(x) = 2^{\mathbf{exp}_n(x)}$. By means of Piterman and Pneuli's algorithm [30] for Streett game solving, the game can be solved in time $O(|\mathcal{N}|^{k+2}\mathbf{exp}_N(O(A|\mathbb{E}||Q|))kk!)$ where $k = |\mathcal{C}|$. Therefore, the typability game can be solved in time N-fold exponential in A and $|Q|$, and $(N+1)$-fold exponential in k, as $|\mathbb{E}| = 4^k$. If we use $\mathbb{E} = 2^{\{E_1,\ldots,E_k,F_1,\ldots,F_k\}}/\sim$ instead of $\mathbb{E} = 2^{\{E_1,\ldots,E_k,F_1,\ldots,F_k\}}$, where $\sim$ is the equivalence relation defined in the proof of Theorem 8, $|\mathbb{E}|$ is reduced to 3^k.

4 Practical Algorithms for Streett and Parity Automata Model Checking

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 $G_{\mathcal{G},\mathcal{A}}$ 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.

Following previous algorithms for trivial automata model checking [14, 16, 3] and parity automata model checking [7, 28], our algorithm for Streett automata model checking first computes a set of types relevant for deciding the model checking problem, constructs a subgame (i.e., a subgraph when viewed as a graph) of $G_{\mathcal{G},\mathcal{A}}$ constructed from those types, and then solves it. If the set of relevant types is sufficiently small for typical instances, the algorithm can be expected to terminate quickly, although in the worst case it still suffers from the N-fold exponential time complexity.

The overall structure of the algorithm HorSatS is shown in Figure 2. An *effectless type environment*, denoted by Θ, is a set of type bindings $F : \theta$ where F is a non-terminal and θ is a prime type. The algorithm starts with a certain initial effectless type environment Θ_0

[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 $h'(x) = O(h(x))$.

$\Theta := \Theta_0$
while $(\mathcal{F}(\Theta) \neq \Theta)\{$
  $\Theta := \mathcal{F}(\Theta)$
$\}$
return whether *ConstructGame*(Θ) has a winning strategy

Figure 2 The proposed algorithm HorSatS.

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

We describe the construction of Θ_0 and the expansion function $\mathcal{F}$ below; it has been inspired by Broadbent and Kobayashi's HorSatT algorithm for trivial automata model checking [3]. Let an input HORS be $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ and an input Streett automaton be $\mathcal{A} = (\Sigma, Q, \delta, q_0, \mathcal{C})$. The initial effectless type environment Θ_0 is defined by:

$$\Theta_0 = \{F : \overbrace{\top \to \cdots \to \top}^{m} \to q \mid F \in \mathit{dom}(\mathcal{N}),\ m = \mathit{arity}(\mathcal{N}(F)),\ q \in Q\}.$$

The expansion function $\mathcal{F}$ is defined by:

$$\begin{aligned}\mathcal{F}(\Theta) =& \Theta \cup \{F : \tau_1 \to \cdots \to \tau_m \to q \mid \mathcal{R}(F) = \lambda x_1 \ldots x_m.\ t,\\ & (\tau_1 \to \cdots \to \tau_m \to q) :: \mathcal{N}(F),\\ & \tau_i \subseteq \mathbf{Types}_\Theta(\mathbf{Flow}(x_i)) \text{ for each } i \in \{1, \ldots, m\},\\ & \Gamma \cup \{x_1 : \tau_1, \ldots, x_m : \tau_m\} \vdash_{\mathcal{A}} t : q\\ & \text{ for some } \Gamma \text{ such that } {}^{\forall}(G : (\theta, e)) \in \Gamma.\ (G : \theta) \in \Theta\}.\end{aligned}$$

Here, $\mathbf{Flow}(x)$ 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, $\mathbf{Types}_\Theta(U)$ is defined as $\{\theta \mid \Gamma \vdash_{\mathcal{A}} u : \theta, u \in U, {}^{\forall}(F : (\theta, e)) \in \Gamma.\ (F : \theta) \in \Theta\}$. The notation $\{x_1 : \tau_1, \ldots, x_m : \tau_m\}$ represents the type environment $\{x_i : (\theta_{i,j}, e_{i,j}) \mid i \in \{1, \ldots, m\},\ (\theta_{i,j}, e_{i,j}) \in \tau_i\}$.

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

$$E'_\exists = \{((F, \theta, e), \Gamma) \in V_\exists \times V_\forall \mid \Gamma \vdash_{\mathcal{A}} \mathcal{R}(F) : \theta \text{ and } {}^{\forall}(G : (\theta, e)) \in \Gamma.\ (G : \theta) \in \Theta\}.$$

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*(Θ) to produce a subgame of the typability game for the Kobayashi-Ong type system. See the longer version [36] for more details.

▶ **Example 9** (a sample run of the algorithm)**.** Consider a HORS $\mathcal{G}_2 = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ where $\Sigma = \{\mathtt{a} \mapsto 2,\ \mathtt{b} \mapsto 1,\ \mathtt{c} \mapsto 0\}$, $\mathcal{N} = \{F \mapsto (\circ \to \circ),\ S \mapsto \circ\}$ and $\mathcal{R} = \{F \mapsto (\lambda x.\mathtt{a}\ x\ (F\ (\mathtt{b}\ x))),\ S \mapsto (F\ \mathtt{c})\}$, and a Streett automaton $\mathcal{A}_2 = (\Sigma, Q, \delta, q_\mathtt{a}, \mathcal{C})$ with Q $= \{q_\mathtt{a}, q_\mathtt{b}\}$, $\mathcal{C} = \{(E_1, \emptyset)\}$ where $E_1 = \{q_\mathtt{a}\}$, and δ is defined by $\delta(q, \mathtt{a}) = (1, q_\mathtt{a}) \wedge (2, q_\mathtt{a})$ for

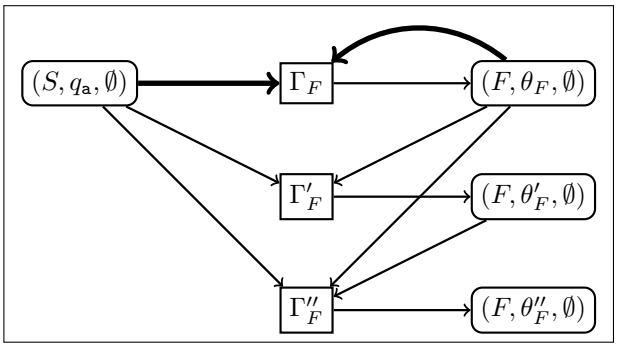

Figure 3 A Streett game generated by HorSatS for inputs $\mathcal{G}_2$ and $\mathcal{A}_2$. Only the part reachable from $(S, q_{\mathtt{a}}, \emptyset)$ is shown. A memoryless winning strategy is indicated by the bold arrows.

each q, $\delta(q, \mathtt{b}) = (1, q_{\mathtt{b}})$ for each q, and $\delta(q, \mathtt{c}) = \mathbf{true}$ for each q. The automaton $\mathcal{A}_2$ accepts trees in which $\mathtt{b}$ occurs only finitely often in every path. The initial effectless type environment is $\Theta_0 = \{F : \top \to q_{\mathtt{a}}, F : \top \to q_{\mathtt{b}}, S : q_{\mathtt{a}}, S : q_{\mathtt{b}}\}$. Let $\mathbf{Flow}(x) = \{\mathtt{b}^n\mathtt{c} \mid n \geq 0\}$ (hence $\mathbf{Types}_\Theta(\mathbf{Flow}(x_i)) = \{q_{\mathtt{a}}, q_{\mathtt{b}}\}$ for any Θ). The fixpoint calculation proceeds as follows.

$$\Theta_1 = \mathcal{F}(\Theta_0) = \Theta_0 \cup \{F : (q_{\mathtt{a}}, \emptyset) \to q_{\mathtt{a}},\ F : (q_{\mathtt{b}}, \{E_1\}) \to q_{\mathtt{b}}\}.$$
$$\Theta_2 = \mathcal{F}(\Theta_1) = \Theta_1 \cup \{F : (q_{\mathtt{a}}, \emptyset) \wedge (q_{\mathtt{b}}, \{E_1\}) \to q_{\mathtt{a}},\ F : (q_{\mathtt{a}}, \emptyset) \wedge (q_{\mathtt{b}}, \{E_1\}) \to q_{\mathtt{b}}\}.$$
$$\Theta_3 = \mathcal{F}(\Theta_2) = \Theta_2.$$

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

$$\begin{array}{llllll} \theta_F & = (q_{\mathtt{a}}, \emptyset) \wedge (q_{\mathtt{b}}, \{E_1\}) \to q_{\mathtt{a}}. & \theta'_F & = (q_{\mathtt{a}}, \emptyset) \to q_{\mathtt{a}}. & \theta''_F & = \top \to q_{\mathtt{a}}. \\ \Gamma_F & = \{F : (\theta_F, \emptyset)\}. & \Gamma'_F & = \{F : (\theta'_F, \emptyset)\}. & \Gamma''_F & = \{F : (\theta''_F, \emptyset)\}. \end{array}$$

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 $G_{\mathcal{G},\mathcal{A}}$ obtained by restricting only Player's moves. The completeness is, however, non-trivial, as to why the fixpoint Θ^{fix} is sufficiently large so that Player can win the subgame $G_{\mathcal{G},\mathcal{A},\Theta^{\mathrm{fix}}}$ if she can win the whole game $G_{\mathcal{G},\mathcal{A}}$. Whilst the construction of Θ_0 and $\mathcal{F}$ 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 $e_{\mathbf{id}}$, *Eff* and $\circ$ for this generalization.

▶ **Theorem 10** (Soundness). *If* HorSatS *(resp.* HorSatP*) returns "Yes" for an input HORS $\mathcal{G}$ and a Streett (resp. parity) automaton $\mathcal{A}$, then $\vdash_{\mathcal{A}} \mathcal{G}$.*

Proof. Suppose that the algorithm returns "Yes." Then, the Streett game $G_{\mathcal{G},\mathcal{A},\Theta^{\mathrm{fix}}}$ has a winning strategy $\mathcal{W}$ from $(S, q_0, e_{\mathbf{id}})$. As $G_{\mathcal{G},\mathcal{A},\Theta^{\mathrm{fix}}}$ is a subgame of $G_{\mathcal{G},\mathcal{A}}$ obtained by only restricting edges in $E_\exists$, $\mathcal{W}$ is also a winning strategy for $G_{\mathcal{G},\mathcal{A}}$. Thus, we have $\vdash_{\mathcal{A}} \mathcal{G}$. ◀

▶ **Theorem 11** (Completeness). *Given an input HORS $\mathcal{G}$ and a Streett (resp. parity) automaton $\mathcal{A}$,* 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 $\vdash_{\mathcal{A}} \mathcal{G}$. By the definition of $\vdash_{\mathcal{A}} \mathcal{G}$, there exists a finite memory winning strategy $\mathcal{W}$ of the typability game $G_{\mathcal{G},\mathcal{A}}$. By adding a rule for unfolding non-terminals (i.e., a rule for deriving $F : (\theta, e) \vdash_{\mathcal{A}} F : \theta$ from $\Gamma \vdash_{\mathcal{A}} \mathcal{R}(F)$), the typability of HORS can alternatively be described by the existence of a (possibly infinite) type derivation tree Π_0 for $S : q_0$ such that every infinite path in it satisfies the Streett/parity condition derived from that of $\mathcal{A}$.[6] Such a derivation tree Π_0 can be constructed based on $\mathcal{W}$.

We show that we can transform Π_0 to another derivation tree Π_0'' which only uses the types occurring in Θ^{fix} (where Θ^{fix} is the effectless type environment produced by the fixpoint calculation in our algorithm). We first "cut" Π_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 $\top \to \cdots \to \top \to q$, treating $\Gamma \vdash F : \underbrace{\top \to \cdots \to \top}_{arity(F)} \to q$ as an "axiom". This axiom corresponds to the initial type environment Θ_0 used in the algorithm. By accordingly reassigning the types in the tree, we have a finite type derivation tree Π_0' that uses only the types in Θ^{fix} computed by the algorithm.

Unfortunately, Π_0' itself does not represent a winning strategy of $G_{\mathcal{G},\mathcal{A},\Theta^{\mathrm{fix}}}$ as it uses the types of non-terminals in Θ_0 as axioms. If we choose a sufficiently large number as the threshold for the depth of unfoldings, however, by matching each node of Π_0' with a corresponding node in Π_0, we can reconstruct a valid (in the sense that every infinite path satisfies the Streett/parity condition) infinite derivation tree Π_0'', by replacing some edges in Π_0' with "back edges". Since Π_0'' has been obtained by only rearranging edges in Π_0', Π_0'' also contains only types in Θ^{fix}. By the correspondence between a valid infinite derivation tree and a winning strategy of the typability game, we obtain a winning strategy $\mathcal{W}'$ for the subgame $G_{\mathcal{G},\mathcal{A},\Theta^{\mathrm{fix}}}$ constructed by $\mathit{Constructgame}(\Theta^{\mathrm{fix}})$. Thus, the algorithm should return "Yes".

Appendix 8 shows an example of the construction of Π_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 $\mathbf{Types}_{\Gamma}(\mathbf{Flow}(x_i))$ in $\mathcal{F}$. We also assume that an input HORS is normalized as in [19] so that for each $F \in dom(\mathcal{N})$, $\mathcal{R}(F)$ is of the form $\lambda\widetilde{x}.\, c\,(F_1\,\widetilde{x}_1)\ldots(F_j\,\widetilde{x}_j)$ where $0 \leq j$, $F_1, \ldots, F_j$ are non-terminals, $\widetilde{x}_1, \ldots, \widetilde{x}_j$ 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 $\mathcal{F}(\Gamma)$ is done in $O(P)$ time. Thus, the fixpoint Θ^{fix} can be calculated in time quadratic in P.[7] Since the size of $\Theta^{\mathrm{fix}}(F)$ 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.

[6] 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.

[7] Actually, using the technique of [32], Θ^{fix} can be calculated in linear time.

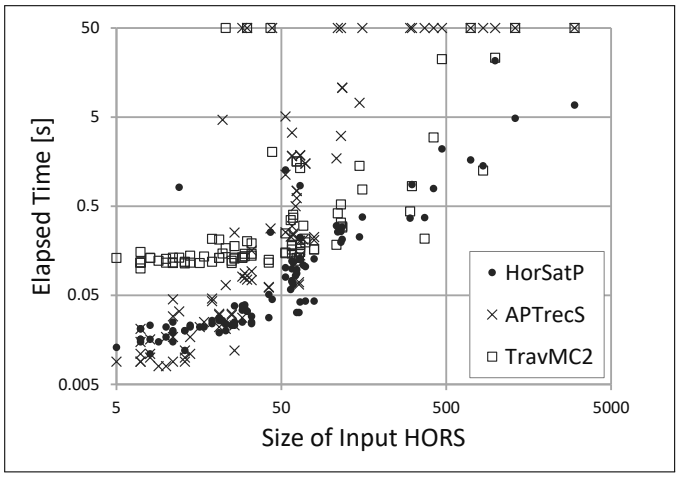

Figure 4 Elapsed time (in seconds) of the model checkers.

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.

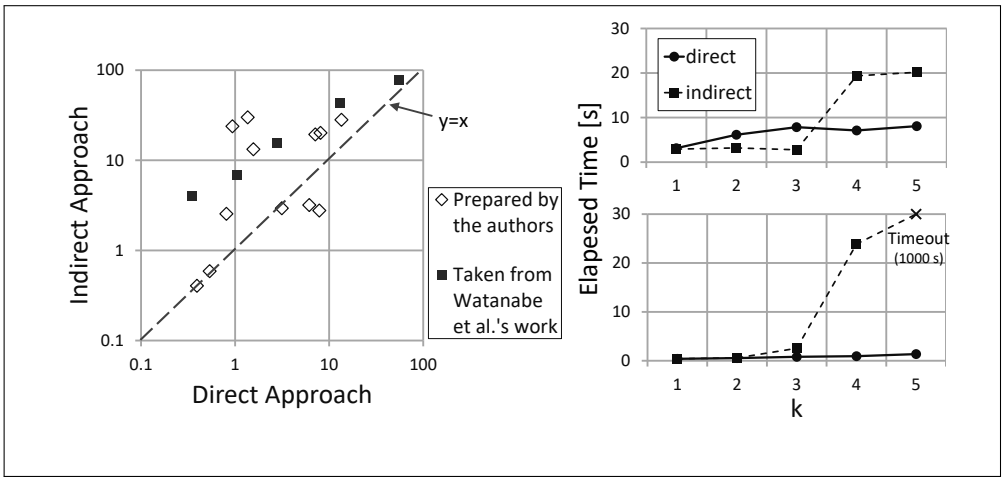

Figure 5 (left) Elapsed time (in seconds) of direct and indirect approaches for several inputs. (right) Elapsed time (in seconds) of direct and indirect approaches for series of inputs with the same structure but different values of k.

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 horizontal axis is the value of k, and the vertical axis is the elapsed time of each approach. 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

1 Klaus Aehlig, Jolie G. de Miranda, and C.-H. Luke Ong. The monadic second order theory of trees given by arbitrary level-two recursion schemes is decidable. In *TLCA 2005*, volume 3461 of *LNCS*, pages 39–54. Springer, 2005. doi:10.1007/11417170_5.

2 Mohamed Faouzi Atig, Ahmed Bouajjani, Michael Emmi, and Akash Lal. Detecting fair non-termination in multithreaded programs. In *Proc. of CAV 2012*, volume 7358 of *LNCS*, pages 210–226. Springer, 2012. doi:10.1007/978-3-642-31424-7_19.

3 Christopher H. Broadbent and Naoki Kobayashi. Saturation-based model checking of higher-order recursion schemes. In *Proc. of CSL 2013*, volume 23 of *LIPIcs*, pages 129–148, 2013. doi:10.4230/LIPIcs.CSL.2013.129.

4 Nils Buhrke, Helmut Lescow, and Jens Vöge. Strategy construction in infinite ganes with streett and rabin chain winning conditions. In *TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 207–224. Springer, 1996.

5 Byron Cook, Alexey Gotsman, Andreas Podelski, Andrey Rybalchenko, and Moshe Y. Vardi. Proving that programs eventually do something good. In *Proc. of POPL*, pages 265–276. ACM Press, 2007.

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

7 Koichi Fujima, Sohei Ito, and Naoki Kobayashi. Practical alternating parity tree automata model checking of higher-order recursion schemes. In *Proc. of APLAS 2013*, volume 8301 of *LNCS*, pages 17–32, 2013.

8 E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*. LNCS. Springer, 2002.

9 Charles Grellois. *Semantics of linear logic and higher-order model-checking*. PhD thesis, Universit'e Paris Diderot, 2016.

10 Charles Grellois and Paul-André Melliès. Relational semantics of linear logic and higher-order model checking. In *CSL*, volume 41 of *LIPIcs*, pages 260–276. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.

11 Nevin Heintze and David McAllester. On the cubic bottleneck in subtyping and flow analysis. In *Proc. of LICS*, pages 342–351, 1997.

12 Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Deciding monadic theories of hyperalgebraic trees. In *TLCA*, volume 2044 of *LNCS*, pages 253–267. Springer, 2001.

13 Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS 2002*, volume 2303 of *LNCS*, pages 205–222. Springer, 2002.

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

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

16 Naoki Kobayashi. A practical linear time algorithm for trivial automata model checking of higher-order recursion schemes. In *Proc. of FoSSaCS 2011*, volume 6604 of *LNCS*, pages 260–274. Springer, 2011.

17 Naoki Kobayashi. Model checking higher-order programs. *Journal of the ACM*, 60(3), 2013.

18 Naoki Kobayashi. HorSat2: A saturation-based model checker for hors. A tool available from `http://www-kb.is.s.u-tokyo.ac.jp/~koba/horsat2/`, 2016.

19 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proc. of LICS 2009*, pages 179–188. IEEE Computer Society Press, 2009.

20 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. A draft of the longer version of [20], available from `http://www-kb.is.s.u-tokyo.ac.jp/~koba/papers/lics09-full.pdf`, 2012.

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

22 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. ACM Press, 2010.

23 Takuya Kuwahara, Ryosuke Sato, Hiroshi Unno, and Naoki Kobayashi. Predicate abstraction and CEGAR for disproving termination of higher-order functional programs. In *Proc. of CAV 2015*, volume 9207 of *LNCS*, pages 287–303. Springer, 2015.

24 M. M. Lester, R. P. Neatherway, C.-H. Luke Ong, and S. J. Ramsay. Model checking liveness properties of higher-order functional programs. In *Proc. of ML Workshop*, 2011.

25 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.

26 C.-H. Luke Ong and Steven Ramsay. Verifying higher-order programs with pattern-matching algebraic data types. In *Proc. of POPL*, pages 587–598. ACM Press, 2011.

27 Akihiro Murase, Tachio Terauchi, Naoki Kobayashi, Ryosuke Sato, and Hiroshi Unno. Temporal verification of higher-order functional programs. In *Proc. of POPL 2016*, pages 57–68. ACM, 2016.

28 Robin P. Neatherway and C.-H. Luke Ong. TravMC2: higher-order model checking for alternating parity tree automata. In *Proc. of SPIN 2014*, pages 129–132. ACM, 2014.

29 Robin P. Neatherway, Steven James Ramsay, and C.-H. Luke Ong. A traversal-based algorithm for higher-order model checking. In *Proc. of ICFP 2012*, pages 353–364, 2012.

30 Nir Piterman and Amir Pnueli. Faster solutions of Rabin and Streett games. In *LICS 2006*, pages 275–284. IEEE Computer Society Press, 2006.

31 Steven Ramsay, Robin Neatherway, and C.-H. Luke Ong. An abstraction refinement approach to higher-order model checking. In *Proc. of POPL*, pages 61–72. ACM, 2014.

32 Jakob Rehof and Torben Mogensen. Tractable constraints in finite semilattices. *Science of Computer Programming*, 35(2):191–221, 1999.

33 Sylvain Salvati and Igor Walukiewicz. Recursive schemes, krivine machines, and collapsible pushdown automata. In *Proc. of RP*, volume 7550 of *LNCS*, pages 6–20. Springer, 2012.

34 Ryosuke Sato, Hiroshi Unno, and Naoki Kobayashi. Towards a scalable software model checker for higher-order programs. In *Proc. of PEPM*, pages 53–62. ACM Press, 2013.

35 Robert S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(1/2):121–141, 1982.

36 Ryota Suzuki, Koichi Fujima, Naoki Kobayashi, and Takeshi Tsukada. Streett automata model checking of higher-order recursion schemes. A longer version, available from `http://www-kb.is.s.u-tokyo.ac.jp/~koba/papers/fscd17-long.pdf`, 2017.

37 Taku Terao and Naoki Kobayashi. A ZDD-based efficient higher-order model checking algorithm. In *Proc. of APLAS 2014*, volume 8858 of *LNCS*, pages 354–371. Springer, 2014.

38 Takeshi Tsukada and C.-H. Luke Ong. Compositional higher-order model checking via ω-regular games over Böhm trees. In *CSL-LICS 2014*, pages 78:1–78:10. ACM, 2014.

39 Hiroshi Unno, Naoshi Tabuchi, and Naoki Kobayashi. Verification of tree-processing programs via higher-order model checking. In *Proc. of APLAS 2010*, volume 6461 of *LNCS*, pages 312–327. Springer, 2010.

40 Moshe Y. Vardi. Verification of concurrent programs: The automata-theoretic framework. *Ann. Pure Appl. Logic*, 51(1-2):79–98, 1991.

41 Keiichi Watanabe, Ryosuke Sato, Takeshi Tsukada, and Naoki Kobayashi. Automatically disproving fair termination of higher-order functional programs. In *Proc. of ICFP 2016*, pages 243–255. ACM, 2016.

8 Example of the Construction of Π_0'' in the Proof of Theorem 11

We show an example of the construction of the derivation tree Π_0'' explained in the proof sketch of Theorem 11 in Section 4. When showing derivation trees, we omit irrelevant or repeated parts to save space. An application of the new rule for unfolding non-terminals is indicated by (Unfold).

Let $\mathcal{G}_2$ and $\mathcal{A}_2$ be the HORS and the Streett automaton in Example 9 respectively. Let $\mathcal{W}$ be the memoryless winning strategy of $G_{\mathcal{G}_2,\mathcal{A}_2}$ defined by $\mathcal{W}((S, q_{\mathtt{a}}, \emptyset)) = \Gamma_1$ and $\mathcal{W}((F, \hat{\theta}_F, \emptyset)) = \Gamma_1$ where $\hat{\theta}_F = (q_{\mathtt{a}}, \emptyset) \wedge (q_{\mathtt{a}}, \{E_1\}) \wedge (q_{\mathtt{b}}, \{E_1\}) \to q_{\mathtt{a}}$ and $\Gamma_1 = \{F : (\hat{\theta}_F, \emptyset)\}$. We write $F : \theta$ instead of $F : (\theta, \emptyset)$ in type environments. A derivation tree Π_0 that corresponds to $\mathcal{W}$ is as follows:

$$
\dfrac{\dfrac{\emptyset\vdash\mathtt{c}:q_{\mathtt{a}} \quad \emptyset\vdash\mathtt{c}:q_{\mathtt{b}} \quad \dfrac{\dfrac{\dfrac{\dots\vdash\mathtt{a}\;x:(q_{\mathtt{a}},\emptyset)\to q_{\mathtt{a}} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\begin{matrix}\vdots\\ \{F:\hat{\theta}_F\}\vdash\lambda x.\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):\hat{\theta}_F\end{matrix}}{\{F:\hat{\theta}_F\}\vdash F:\hat{\theta}_F}\,(\textsc{Unfold})}{\begin{matrix}\vdots\\ \{F:\hat{\theta}_F\}\vdash\lambda x.\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):\hat{\theta}_F\end{matrix}}}{\{F:\hat{\theta}_F\}\vdash F:\hat{\theta}_F}\,(\textsc{Unfold})}{\begin{matrix}\vdots\\ \{F:\hat{\theta}_F\}\vdash\lambda x.\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):\hat{\theta}_F\end{matrix}}}{\{F:\hat{\theta}_F\}\vdash F:\hat{\theta}_F}\,(\textsc{Unfold}) \quad \dots\vdash\mathtt{b}\;x:q_{\mathtt{a}} \quad \dots\vdash\mathtt{b}\;x:q_{\mathtt{b}}}{\{F:\hat{\theta}_F, x:(q_{\mathtt{b}},\{E_1\})\}\vdash F\;(\mathtt{b}\;x):q_{\mathtt{a}}}}{\{F:\hat{\theta}_F, x:q_{\mathtt{a}}, x:(q_{\mathtt{b}},\{E_1\})\}\vdash\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):q_{\mathtt{a}}}}{\{F:\hat{\theta}_F\}\vdash\lambda x.\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):\hat{\theta}_F}}{\{F:\hat{\theta}_F\}\vdash F:\hat{\theta}_F}\,(\textsc{Unfold})}{\{F:\hat{\theta}_F\}\vdash F\;\mathtt{c}:q_{\mathtt{a}}}}{\{S:q_{\mathtt{a}}\}\vdash S:q_{\mathtt{a}}}\,(\textsc{Unfold})
$$

We construct Π_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 Π_0' looks like:

$$
\dfrac{\dfrac{\emptyset\vdash\mathtt{c}:q_{\mathtt{a}} \quad \emptyset\vdash\mathtt{c}:q_{\mathtt{b}} \quad \dfrac{\dfrac{\dfrac{\{x:q_{\mathtt{a}}\}\vdash\mathtt{a}\;x:\theta_{\mathtt{a}1} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\{x:(q_{\mathtt{b}},\{E_1\})\}\vdash\mathtt{b}\;x:q_{\mathtt{a}} \quad \dfrac{\dfrac{\dfrac{\{x:q_{\mathtt{a}}\}\vdash\mathtt{a}\;x:\theta_{\mathtt{a}1} \quad \dfrac{\dfrac{}{\{F:\top\to q_{\mathtt{a}}\}\vdash F:\top\to q_{\mathtt{a}}}\,(\textsc{Axiom})}{\{F:\top\to q_{\mathtt{a}}\}\vdash F\;(\mathtt{b}\;x):q_{\mathtt{a}}}}{\{F:\top\to q_{\mathtt{a}}, x:q_{\mathtt{a}}\}\vdash\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):q_{\mathtt{a}}}}{\{F:\top\to q_{\mathtt{a}}\}\vdash\lambda x.\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):(q_{\mathtt{a}},\emptyset)\to q_{\mathtt{a}}}}{\{F:(q_{\mathtt{a}},\emptyset)\to q_{\mathtt{a}}\}\vdash F:(q_{\mathtt{a}},\emptyset)\to q_{\mathtt{a}}}\,(\textsc{Unfold})}{\{F:(q_{\mathtt{a}},\emptyset)\to q_{\mathtt{a}}, x:(q_{\mathtt{b}},\{E_1\})\}\vdash F\;(\mathtt{b}\;x):q_{\mathtt{a}}} \quad \{x:q_{\mathtt{a}}\}\vdash\mathtt{a}\;x:\theta_{\mathtt{a}1}}{\{F:\theta_F, x:q_{\mathtt{a}}, x:(q_{\mathtt{b}},\{E_1\})\}\vdash\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):q_{\mathtt{a}}}}{\{F:\theta_F\}\vdash\lambda x.\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):\theta_F}}{\{F:\theta_F\}\vdash F:\theta_F}\,(\textsc{Unfold}) \quad \{x:(q_{\mathtt{b}},\{E_1\})\}\vdash\mathtt{b}\;x:q_{\mathtt{a}}}{\{F:\theta_F, x:(q_{\mathtt{b}},\{E_1\})\}\vdash F\;(\mathtt{b}\;x):q_{\mathtt{a}}}}{\{F:\theta_F, x:q_{\mathtt{a}}, x:(q_{\mathtt{b}},\{E_1\})\}\vdash\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):q_{\mathtt{a}}}}{\{F:\theta_F\}\vdash\lambda x.\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):\theta_F}}{\{F:\theta_F\}\vdash F:\theta_F}\,(\textsc{Unfold})}{\{F:\theta_F\}\vdash F\;\mathtt{c}:q_{\mathtt{a}}}}{\{S:q_{\mathtt{a}}\}\vdash S:q_{\mathtt{a}}}\,(\textsc{Unfold})
$$

Here, $\theta_F = (q_{\mathtt{a}},\emptyset)\wedge(q_{\mathtt{b}},\{E_1\})\to q_{\mathtt{a}}$ and $\theta_{\mathtt{a}1} = (q_{\mathtt{a}},\emptyset)\to q_{\mathtt{a}}$.

The uppermost unfolding has been replaced by the axiom. At the next unfolding below it, F is given the type $(q_{\mathtt{a}},\emptyset)\to q_{\mathtt{a}}$. In the body of this F, x should have type $q_{\mathtt{a}}$ because it is used as an argument of $\mathtt{a}$, which has type $(q_{\mathtt{a}},\emptyset)\to(q_{\mathtt{a}},\emptyset)\to q_{\mathtt{a}}$. On the other hand, x in $\mathtt{b}\;x$ need not have type $q_{\mathtt{b}}$, since $\mathtt{b}\;x$ is not typed in the derivation. Thus, the type

$(q_{\mathtt{a}}, \emptyset) \to q_{\mathtt{a}}$ 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 θ_F. Note that Π_0' assigns only the types in Θ^{fix} to non-terminals. In fact, the types $\top \to q_{\mathtt{a}}$, $(q_{\mathtt{a}}, \emptyset) \to q_{\mathtt{a}}$, and θ_F assigned to F at the axiom, the first unfolding (counted from the top), and the second unfolding respectively belong to Θ_0, $\mathcal{F}(\Theta_0)$, and $\mathcal{F}^2(\Theta_0)$.

Notice that, in Π_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 Π_0'.) By introducing a "back edge" between them, the following derivation tree Π_0'' is obtained.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{F : \theta_F\} \vdash F : \theta_F}\,(\textsc{Unfold}) \quad \{x : (q_{\mathtt{b}}, \{E_1\})\} \vdash \mathtt{b}\ x : q_{\mathtt{a}}}{\{x : q_{\mathtt{a}}\} \vdash \mathtt{a}\ x : (q_{\mathtt{a}}, \emptyset) \to q_{\mathtt{a}} \quad \{F : \theta_F, x : (q_{\mathtt{b}}, \{E_1\})\} \vdash F\ (\mathtt{b}\ x) : q_{\mathtt{a}}}}{\{F : \theta_F, x : q_{\mathtt{a}}, x : (q_{\mathtt{b}}, \{E_1\})\} \vdash \mathtt{a}\ x\ (F\ (\mathtt{b}\ x)) : q_{\mathtt{a}}}}{\{F : \theta_F\} \vdash \lambda x.\mathtt{a}\ x\ (F\ (\mathtt{b}\ x)) : \theta_F}}{\{F : \theta_F\} \vdash F : \theta_F}\,(\textsc{Unfold})}{\{F : \theta_F\} \vdash F\ \mathtt{c} : q_{\mathtt{a}}}}{\{S : q_{\mathtt{a}}\} \vdash S : q_{\mathtt{a}}}\,(\textsc{Unfold})$$

Note that Π_0'' uses only types that occur in Θ^{fix}. Furthermore, this tree is a valid infinite derivation tree; indeed, it corresponds to the winning strategy of the subgame $G_{\mathcal{G}_2, \mathcal{A}_2, \Theta^{\mathrm{fix}}}$ shown in Figure 3.