Practical Alternating Parity Tree Automata
Model Checking of Higher-Order
Recursion Schemes

Koichi Fujima\textsuperscript{1}, Sohei Ito\textsuperscript{2}, and Naoki Kobayashi\textsuperscript{1}

\textsuperscript{1} The University of Tokyo
\textsuperscript{2} National Fisheries University

Abstract. Higher-order (HO) model checking is the problem of deciding whether the tree generated by a higher-order recursion scheme (HORS) is accepted by an alternating parity tree automaton (APT). HO model checking has been shown to be decidable by Ong and recently applied to automated program verification. Practical HO model checkers have been, however, developed only for subclasses of APT such as trivial tree automata and weak APT. In this paper, we develop a practical model checking algorithm for the full class of APT, and implement an APT model checker for HORS. To our knowledge, this is the first model checker for HORS that can deal with the full class of APT. We also discuss its applications to program verification.

1 Introduction

The model checking of higher-order recursion schemes (HORS) has recently been drawing attention and applied to fully-automated verification of functional programs \cite{8,10,12,17}. A HORS is a higher-order tree grammar for generating a single, possibly infinite tree, which can also be viewed as a term of the simply-typed call-by-name $\lambda$-calculus with recursion and tree constructors (but not destructors). Given a HORS $G$ and an alternating parity tree automaton (APT) $A$, the model checking of HORS \cite{16} asks whether the tree generated by $G$ is accepted by $A$. Although the model checking problem is $k$-EXPTIME complete (for order-$k$ HORS) \cite{16}, practical model checkers, which do not immediately suffer from the $k$-EXPTIME bottleneck, have been developed for subclasses of the APT model checking of HORS \cite{8,12,15}.

The previous studies on practical model checking algorithms for HORS and their applications have, however, not exploited the full power of APT model checking of HORS. Most of them \cite{8,15} restricted tree automata (for expressing tree properties) to Aehlig’s trivial tree automata \cite{1}, which can only express safety properties. Accordingly, applications have also been limited to verification of safety properties (that bad events will never happen). The only exception is the work of Lester et al.’s \cite{12}, who implemented a model checker for a larger subclass of APT called weak alternating tree automata \cite{14}, and applied it to verification of some liveness properties.
The goal of the present paper is to develop a model checker for HORS that can deal with the full class of APT, and apply it to automated verification of functional programs. To this end, we propose a new APT model checking algorithm, which combines Kobayashi and Ong’s reduction from APT model checking of HORS to a typability problem [9], and Kobayashi’s algorithm for trivial automata model checking [8]. We use an extension of Kobayashi’s algorithm to collect type candidates and decide whether a given HORS is typable under Kobayashi and Ong’s type system. As the naive implementation of the algorithm suffers from the explosion of the number of type candidates, we also introduce a novel subtyping relation on Kobayashi and Ong’s intersection types, and apply optimizations. To demonstrate the usefulness of the full APT model checking of HORS, we also discuss extensions of the two previous applications to program verification: resource usage verification and HMTT verification (verification of tree-processing programs) [8,11]. Thanks to the power of the full APT model checking, we can verify more elaborate properties of programs, like “does a program eventually close a file as long as the end of the file is eventually read?” and “does a tree-processing program generate a finite tree as long as the input tree is finite?”

Our contributions are: (i) the first practical, full APT model checking algorithm for HORS and its implementation. To our knowledge, ours is the first implementation of an APT model checker for HORS. (ii) Optimization based on a novel subtyping relation that respects priorities of APT. (iii) Applications to program verification, which take advantage of the full APT model checking.

In the rest of the paper, we first review basic definitions in Section 2. Sections 3 and 4 discuss our APT model checking algorithm and its optimizations. Section 5 discusses applications and Section 6 reports experiments. Section 7 discusses related work and Section 8 concludes the paper. A longer version of this paper is available from the first author’s web page.

2 Preliminaries

We write \( \text{dom}(f) \) and \( \text{codom}(f) \) for the domain and codomain of a map \( f \). A ranked alphabet \( \Sigma \) is a map from a finite set of symbols to non-negative integers (called arities). We write \( \text{ar} (\Sigma) \) for the largest arity of symbols in \( \Sigma \). Let \( \text{Pos} \) be the set of positive integers. An \( L \)-labeled tree is a partial map \( T \) from \( \text{Pos}^* \) to \( L \), such that \( \forall \pi \in \text{Pos}^*.\forall i \in \text{Pos}.(\pi i \in \text{dom}(T) \implies \{\pi\} \cup \{\pi j \mid 1 \leq j \leq i\} \subseteq \text{dom}(T)) \). For a ranked alphabet \( \Sigma \), a \( \Sigma \)-labeled ranked tree is a \( \text{dom} (\Sigma) \)-labeled tree \( T \) such that \( \forall \pi \in \text{Pos}^*.\{i \mid \pi i \in \text{dom}(T)\} = \{i \mid 1 \leq i \leq \Sigma(T(\pi))\} \).

HORS. The sets of sorts and terms are defined by:

\[
\kappa (\text{sorts}) ::= o \mid \kappa_1 \to \kappa_2 \\
t (\text{terms}) ::= a \mid x \mid t_1 t_2 \mid \lambda x : \kappa.t.
\]

Here, meta-variables \( a \) and \( x \) range over \( \text{dom}(\Sigma) \) and a set of variables respectively. We call a term \( t \) an applicative term if it does not contain \( \lambda \)-abstractions. We often omit the sort annotation and write \( \lambda x.t \) for \( \lambda x : \kappa.t \).