Higher-Order Model Checking: An Overview

Luke Ong
University of Oxford

Abstract—**Higher-order model checking is about the model checking of trees generated by recursion schemes. The past fifteen years or so have seen considerable progress in both theory and practice. Advances have been made in determining the expressive power of recursion schemes and other higher-order families of generators, automata-theoretic characterisations of these generator families, and the algorithmics and semantics of higher-order model checking and allied methods of formal analysis. Because the trees generated by recursion schemes are computation trees of higher-order functional programs, higher-order model checking provides a foundation for model checkers of such programming languages as Haskell, F# and Erlang. This paper aims to give an overview of recent developments in higher-order model checking.**

I. Introduction

Recursion schemes are a kind of simply-typed grammar for generating possibly infinite ranked trees. A recursion scheme is given by a finite system of equations, defining a finite set of higher-order functions by mutual recursion. The order of a recursion scheme is given by the highest type-theoretic order of the functions defined by it. From a programming language perspective, recursion schemes may be viewed as programs (i.e. closed, ground-type terms) of the simply-typed lambda calculus with recursion, constructed from a set of uninterpreted function symbols. Higher-order model checking is the model checking of trees generated by recursion schemes. The higher-order model checking problem asks, given a recursion scheme $\mathcal{G}$ and a correctness property φ, whether the tree generated by $\mathcal{G}$ satisfies φ.

The early work in the 1970s on (tree-generating) higher-order recursion schemes was mostly about questions of expressivity. The decision problem was first posed in the present form by Knapik et al. [56], [54]. They considered recursion schemes subject to a syntactic constraint called *safety*. Their main result is that, when restricted to trees generated by safe recursion schemes, the higher-order model checking problem with respect to monadic second-order (MSO) properties is decidable. However the general MSO higher-order model checking problem was left open. In [73], Ong proved that the modal mu-calculus model checking problem of trees generated by arbitrary order-n recursion schemes is n-EXPTIME complete. Since MSO logic and the modal mu-calculus are equi-expressive over trees, it follows that the MSO higher-order model checking problem is decidable.

The last fifteen years or so have seen a growth of interest in higher-order model checking from both the theory and practice communities. Since the MSO decidability result [73], which was proved using insights from game semantics [49], a variety of semantic and algorithmic techniques and models of computation have been employed to design higher-order model checking algorithms, notably, intersection types [59], [64], collapsible pushdown automata [45] and Krivine machines [84]. Algorithmic properties that refine and extend the decidability of MSO model checking have also been introduced, such as logical reflection [12], effective selection [15] and transfer theorem [85].

Recursive schemes are a very expressive family of generators of trees: the trees that are generated at orders 0, 1 and 2 are respectively the regular trees, algebraic trees (i.e. those generated by context-free tree grammars) and hyperalgebraic trees. There have been advances in the understanding of the relationship between higher-order families of generators. Hague et al. [45] showed that order-n recursion schemes generate the same class of trees as order-n collapsible pushdown automata. More recently, Parys [79] proved the Safety Conjecture, thus confirming the intuition that the safety constraint restricts the expressive power of recursion schemes, or equivalently that order-n collapsibile pushdown automata are more expressive than order-n pushdown automata.

Recursion schemes are thus an appealing abstract model for model checking higher-order programs: not only are they highly expressive, the trees they generate also enjoy a rich and decidable logical theory. In [59], Kobayashi introduced an approach to the verification of safety properties of functional programs by reduction to higher-order model checking. His verification method is sound, complete and automatic for a range of verification problems such as reachability, flow analysis and resource usage for simply-typed recursive functional programs generated from finite base types. Although the worst-case complexity of higher-order model checking is hyper-exponential, there have been remarkable advances in the design of model checking algorithms. The time complexity of state-of-the-art model checking algorithms, PREFACE [83] and HORSATZDD [94], are fixed-parameter polynomial in the size of the recursion schemes, and scale readily to many thousands of rules.

This paper aims to provide an overview of higher-order model checking, covering mostly theoretical developments. Our treatment is far from a complete survey of the field: in many places, we give no more than selected references where the reader can find further details. We stress that the bibliography is non-exhaustive. Nevertheless our hope is that people will find the paper a useful introduction to this exciting and challenging topic.

Outline: In Section II, we introduce two families of generators of infinite structures: recursion schemes and higher-order pushdown automata. We discuss results about expressivity and the relationships between the families, and the safety constraint. Section III is about the model checking of recursion schemes with respect to monadic second-order and related correctness properties. We survey a number of topics about the higher-order model checking problem: decidability proofs; collapsible pushdown automata as generators of trees and graphs, and their algorithmics; and the Safety Conjecture In Section IV, we outline recent advances in the compositional model checking of (higher-type) Böhm trees, and effective denotational semantics of higher-order model checking. In Section V, we give a quick overview of applications of higher-order model checking to the verification of functional programs. We conclude in Section V with a brief discussion of open problems and further directions.

II. Families of Generators of Infinite Structures

A. Recursion Schemes

In higher-order model checking, recursion schemes are typically used as simply-typed grammars for constructing possibly infinite term-trees or ranked trees. *Types* are defined by the grammar $A ::= o \mid A \to B$. By convention, arrows associate to the right; thus every type can be written uniquely as $A_1 \to \cdots \to A_n \to o$, for some $n \geq 0$. We define the *order* of a type: $ord(o) := o$ and $ord(A \to B) := \max(ord(A) + 1, ord(B))$. Intuitively the order of a type measures how deeply nested it is on the left of the arrow.

Assume a countably infinite set Var of typed variables. Let Θ be a set of typed symbols such as Var. Writing $s : A$ to mean s has type A, the set of *applicative terms* generated from Θ is the least set containing Θ closed under the *application rule*: if $s : A \to B$ and $t : A$ then $s\,t : B$. By convention, application associates to the left; thus $s_1 \cdots s_n$ means $((s_1\,s_2)s_3 \cdots)s_n$. Given a term $s : A$, we define $ord(s) := ord(A)$.

Definition 1. A *higher-order recursion scheme* (or simply *recursion scheme*) is a tuple $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ where

- Σ is a *ranked alphabet of terminals* viewed as tree constructors i.e. each $f \in \Sigma$ has an arity $\mathsf{ar}(f) \geq 0$, written $f : \mathsf{ar}(f)$ by abuse of notation, and we assume $f : \underbrace{o \to \cdots o \to}_{\mathsf{ar}(f)} o$.
- $\mathcal{N}$ is a set of typed *non-terminals*; and $S : o \in \mathcal{N}$ is a distinguished *start symbol* of type o.
- $\mathcal{R}$ is a finite set of rewrite rules of the form $F\,x_1 \cdots x_n \to e$ where $F : A_1 \to \cdots \to A_n \to o$, each $x_i : A_i$, and $e : o$ is an applicative term generated from $\Sigma \cup \mathcal{N} \cup \{x_1, \cdots, x_n\}$.

The *order* of a recursion scheme is defined to be the highest order of its non-terminals. In the following we shall use uppercase letters F, G, H, etc., to range over non-terminals, lowercase letters f, g, h, etc., to range over terminals, and φ, ψ, x, y, z, etc., to range over variables.

As a simply-typed formalism, recursion schemes may be viewed as a subsystem of the $\lambda\mathbf{Y}$-calculus [91], which is the (pure) simply-typed lambda calculus augmented with fixpoint combinators $\mathbf{Y}$. Precisely there is a correspondence between recursion schemes and ground-type $\lambda\mathbf{Y}$-terms with free variables (corresponding to the terminals) of order at most one: they define the same set of trees [84].

In general, recursion schemes define tree languages. In this paper, we are mainly concerned with recursion schemes that define trees. Hence, unless otherwise specified, recursion schemes are assumed to be *deterministic* i.e. for each non-terminal, there is at most one rewrite rule (whose left-hand side is) headed by that non-terminal.

Informally the tree generated by a recursion scheme $\mathcal{G}$, denoted $[\![\mathcal{G}]\!]$, is the abstract syntax tree underlying the possibly-infinite applicative term which is obtained from the start symbol S by unfolding the rewrite rules *ad infinitum*.

Example 2 (Order-1 recursion scheme)**.** Take the ranked alphabet $\Sigma = \{f : 2, g : 1, a : 0\}$. Consider the order-1 recursion scheme $\mathcal{G}_1$ with rewrite rules:

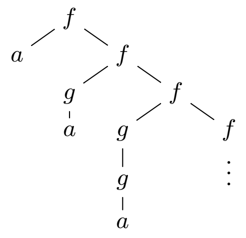
$$S \to F\,a \qquad F\,x \to f\,x\,(F\,(g\,x))$$

where $F : o \to o$. Unfolding from the start symbol S, we have

$$S \to F\,a \to f\,a\,(F\,(g\,a)) \to f\,a\,(f\,(g\,a)\,(F\,(g\,(g\,a)))) \to \cdots$$

thus generating the infinite applicative term

$$f\,a\,(f\,(g\,a)\,(f\,(g\,(g\,a))(\cdots))).$$



Because the rewrite system is Church-Rosser, it does not matter which reduction strategy is used provided it is *fair* in the sense of not neglecting any branch. The tree generated by $\mathcal{G}_1$ (as shown on the right), $[\![\mathcal{G}_1]\!]$, is the abstract syntax tree of the infinite term.

Formally, given a recursion scheme $\mathcal{G}$, the reduction relation $\to_{\mathcal{G}}$ is defined by induction over the rules:

$$\frac{F x_1 \cdots x_n \to e \text{ is a rule in } \mathcal{R}}{F t_1 \cdots t_n \to_{\mathcal{G}} e[t_1/x_1, \cdots, t_n/x_n]}$$

$$\frac{t \to_{\mathcal{G}} t'}{s\,t \to_{\mathcal{G}} s\,t'} \qquad \frac{t \to_{\mathcal{G}} t'}{t\,s \to_{\mathcal{G}} t'\,s}$$

We write $\to_{\mathcal{G}}^*$ for the reflexive, transitive closure of $\to_{\mathcal{G}}$. When it is clear from the context, we omit the subscript from $\to_{\mathcal{G}}$.

A Σ-*labelled tree* is a partial function t from $\{1, \cdots, M\}^*$ to Σ, where $M := \max\{\mathsf{ar}(f) \mid f \in \Sigma\}$, such that $dom(t)$ is prefix-closed; we assume that t is *ranked*: if $t(w) = a$ and $\mathsf{ar}(a) = r$, then $\{i \mid w\,i \in dom(t)\} = \{1, \cdots, r\}$. A (possibly infinite) sequence π over $\{1, \cdots, M\}$ is a *path* of t if every prefix of π is in $dom(t)$. Given a term t, we define a (finite) $(\Sigma \cup \{\bot\})$-labelled tree $t^\bot$ by:

$$t^\bot := \begin{cases} f & \text{if } t = f \text{, a terminal} \\ t_1^\bot t_2^\bot & \text{if } t = t_1\,t_2 \text{ and } t_1^\bot \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

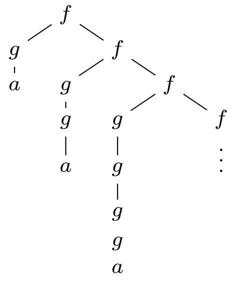
For example $(f\,(F\,a)\,b)^{\perp} = f \perp b$. Let $\sqsubseteq$ be the partial order on $\Sigma \cup \{\perp\}$ defined by $\forall a \in \Sigma. \perp \sqsubseteq a$. We extend $\sqsubseteq$ to a partial order on trees by:

$$s \sqsubseteq t := \forall w \in dom(s).(w \in dom(t) \wedge s(w) \sqsubseteq t(w)).$$

For example, $\perp \sqsubseteq f \perp \perp \sqsubseteq f \perp b \sqsubseteq f\,a\,b$. For a directed set T of trees, we write $\bigsqcup T$ for the least upper bound of elements of T with respect to $\sqsubseteq$. We define the *tree generated by* $\mathcal{G}$, or the *value tree* of $\mathcal{G}$, by $[\![\mathcal{G}]\!] := \bigsqcup\{t^{\perp} \mid S \rightarrow^*_{\mathcal{G}} t\}$. By construction, $[\![\mathcal{G}]\!]$ is a possibly infinite $(\Sigma \cup \{\perp\})$-labelled tree. For $n \geq 0$, we write $\mathit{RecSchTree}^{\Sigma}_n$ for the class of Σ-labelled trees generated by order-n recursion schemes.

Example 3 (Order-2 recursion scheme)**.** Take $\Sigma = \{f : 2, g : 1, a : 0\}$, with rewrite rules:

$$\begin{array}{rcl} S & \rightarrow & F\,g \\ B\,\varphi\,\psi\,x & \rightarrow & \varphi\,(\psi\,x) \\ F\,\varphi & \rightarrow & f\,(\varphi\,a)\,(F\,(B\,\varphi\,\varphi)) \end{array}$$



where $B : (o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow o \rightarrow o$ and $F : (o \rightarrow o) \rightarrow o$. The value tree, $[\![\mathcal{G}_2]\!] : \{1,2\}^* \longrightarrow \Sigma$, is shown on the right.

Historical remark: The precursor of higher-order recursion schemes is program schemes. Much studied in the 1970s, program schemes are a program calculus that separates control structure and operations on data, thus providing a framework for investigating the descriptive power of control structures and program transformation. The idea of program schemes can be traced back to the pioneering 1960 paper of Ianov [50], which inspired early work in the cateogry-theoretic and algebraic semantics of program schemes and related formalisms [35], [89], fixpoint theory and induction principles [78], and the question of decidability of equivalence of program schemes [80], [67]. There is a large literature (for example [32], [72], [24], [23], [44]) throughout the 1970s on the semantics of program schemes, using a variety of approaches including operational, denotational, algebraic and automata-theoretic. In the late 1970s, Indermark, Damm and others [27], [29], [28], [38], [39] studied program schemes that are constrained by a family of simple types, as generators of hierarchies of word and tree languages. For a survey of the early work, see Courcelle's handbook article [21].

B. The Hierarchy of Higher-Order Pushdown Automata

Higher-order pushdown automata were introduced by Maslov [68], [69] as a generalisation of pushdown automata and nested pushdown automata. Let Γ be a stack alphabet that contains a distinguished *bottom-of-stack symbol* $\perp$. An *order-0 stack* is just a stack symbol. An *order-$(n+1)$ stack* is a non-null sequence (written $[\,s_1 \cdots s_l\,]$) of order-n stacks. We often abbreviate order-n stack to n-stack, and write $n\text{-}\mathit{Stack}_{\Gamma}$ for the set of n-stacks over Γ. As usual, $\perp$ cannot be popped from or pushed onto a stack. (Thus an *order-1 stack* is a non-null sequence $[\,a_1 \cdots a_l\,]$ of Γ-symbols such that for all $1 \leq i \leq l$, $a_i = \perp$ if and only if $i = 1$.) We define $\perp_k$, the *empty k-stack*: $\perp_0 := \perp$ and $\perp_{k+1} := [\,\perp_k\,]$. When displaying examples of n-stacks, we omit $\perp$ to avoid clutter.

The operations on 1-stacks, namely, $push^Z_1$ with $Z \in \Gamma \setminus \{\perp\}$, pop_1 and top_1, are standard. For $n \geq 2$, the following operations are defined on n-stacks. Let $1 \leq k < n$, $2 \leq k' < n$ and $Z \in \Gamma \setminus \{\perp\}$.

$$\begin{array}{rcl} push_n\,[\,s_1 \cdots s_{i-1}\,s_i\,] & = & [\,s_1 \cdots s_{i-1}\,s_i\,s_i\,] \\ pop_n\,[\,s_1 \cdots s_{i-1}\,s_i\,] & = & [\,s_1 \cdots s_{i-1}\,] \\ push_{k'}\,[\,s_1 \cdots s_{i-1}\,s_i\,] & = & [\,s_1 \cdots s_{i-1}\,push_{k'}\,s_i\,] \\ push^Z_1\,[\,s_1 \cdots s_{i-1}\,s_i\,] & = & [\,s_1 \cdots s_{i-1}\,push^Z_1\,s_i\,] \\ pop_k\,[\,s_1 \cdots s_{i-1}\,s_i\,] & = & [\,s_1 \cdots s_{i-1}\,pop_k\,s_i\,] \\ top_n\,[\,s_1 \cdots s_{i-1}\,s_i\,] & = & s_i \\ top_k\,[\,s_1 \cdots s_{i-1}\,s_i\,] & = & top_k\,s_i \end{array}$$

For $1 \leq k \leq n$, the operation pop_k is undefined on any n-stack such that its top k-stack (or the n-stack itself, in case $k = n$) has only one element. For example $pop_2\,[\,[\,\perp\alpha\,\beta\,]\,]$ and $pop_1\,[\,[\,\perp\alpha\,\beta\,]\,[\,\perp\,]\,]$ are both undefined. For $n \geq 0$, we define the set Op_n of *order-n stack operations*:

$$\begin{array}{rl} & Op_1 := \{push^Z_1 \mid Z \in \Gamma \setminus \{\perp\}\} \cup \{pop_1\} \\ n \geq 2, & Op_n := \{push_k, pop_k \mid 2 \leq k \leq n\} \cup Op_1. \end{array}$$

Our aim in this subsection is to define higher-order pushdown automata as generators of possibly infinite ranked trees. We first introduce a general notion of store system.

Definition 4. A *store system* is a tuple $\langle \Gamma, \mathit{Store}_{\Gamma}, \mathit{Op}, \mathit{top}, \perp \rangle$ where Γ is a (finite) store alphabet, Store_{Γ} is a set of *stores* notionally generated from Γ, Op is a set of *store operations* which are partial functions $\mathit{Store}_{\Gamma} \rightharpoonup \mathit{Store}_{\Gamma}$, $\mathit{top} : \mathit{Store}_{\Gamma} \rightarrow \Gamma$ is a *read* function, and $\perp \in \mathit{Store}_{\Gamma}$ is the initial store.

For $n \geq 0$, the tuple $\langle \Gamma, n\text{-}\mathit{Stack}_{\Gamma}, Op_n, top_1, \perp_n \rangle$, which we shall call the *system of order-n stacks over* Γ, is a store system. Another example is the *system of order-n collapsible stacks*, which we will introduce in Section III. The semi-infinite tape (of a Turing machine) and the FIFO queue (of a Minsky machine) are also examples of store system.

Fix a ranked alphabet Σ with $m := \max\{\mathsf{ar}(f) \mid f \in \Sigma\}$. The branch language [20] of a Σ-labelled ranked tree t is a set of finite and infinite words that represent its maximal paths (or branches). Formally the *branch language* of $t : dom(t) \rightarrow \Sigma$ consists of:

- infinite words $(f_1, d_1)(f_2, d_2) \cdots$ such that there exists $d_1\,d_2 \cdots \in \{1, \cdots, m\}^{\omega}$ with $t(d_1 \cdots d_i) = f_{i+1}$ and $d_{i+1} \leq \mathsf{ar}(f_{i+1})$ for every $i \geq 0$, and
- finite words $(f_1, d_1) \cdots (f_n, d_n)\,f_{n+1}$ such that there exists $d_1 \cdots d_n \in \{1, \cdots, m\}^*$ with $t(d_1 \cdots d_i) = f_{i+1}$ for every $0 \leq i \leq n$, $d_i \leq \mathsf{ar}(f_i)$ for every $1 \leq i \leq n$, and $\mathsf{ar}(f_{n+1}) = 0$.

For example, the branch language of the tree generated by the recursion scheme $\mathcal{G}_1$ of Example 2 is $\{(f,2)^{\omega}\} \cup \{(f,2)^n\,(f,1)\,(g,1)^n\,a \mid n \geq 0\}$. It follows from the definition that two ranked trees are equal if, and only if, they have the same branch language.

Definition 5. (i) Let $\mathcal{S} = \langle \Gamma, \mathit{Store}_\Gamma, \mathit{Op}, \mathit{top}, \bot \rangle$ be a store system. A *tree-generating $\mathcal{S}$-automaton* is a tuple $\mathcal{A} = \langle \mathcal{S}, Q, \Sigma, \delta, q_I \rangle$ where Q is a finite set of states, $q_I \in Q$ is the initial state, Σ is a ranked alphabet, and

$$\delta \,:\, Q \times \Gamma \longrightarrow (Q \times \mathit{Op} \cup \{(f, q_1 \cdots q_{\mathsf{ar}(f)}) \mid f \in \Sigma, q_i \in Q\})$$

is a transition function. A *configuration* is either a pair (q, s) where $q \in Q$ and $s \in \mathit{Store}_\Gamma$, or a triple of the form $(f, q_1 \cdots q_{\mathsf{ar}(f)}, s)$ where $f \in \Sigma$ and $q_1 \cdots q_{\mathsf{ar}(f)} \in Q^*$. Let $\overline{\Sigma}$ be the label set $\{(f, i) \mid f \in \Sigma, 1 \leq i \leq \mathsf{ar}(f)\} \cup \{a \in \Sigma \mid \mathsf{ar}(a) = 0\}$. We define the labelled transition relation between configurations induced by δ:

$$\begin{array}{ll} (q,s) \xrightarrow{\epsilon} (q', \theta(s)) & \text{if } \delta(q, \mathit{top}\, s) = (q', \theta) \\ (q,s) \xrightarrow{\epsilon} (f, \overline{q}, s) & \text{if } \delta(q, \mathit{top}\, s) = (f, \overline{q}) \text{ and } \mathsf{ar}(f) \geq 1 \\ (q,s) \xrightarrow{a} (a, \epsilon, s) & \text{if } \delta(q, \mathit{top}\, s) = (a, \epsilon) \text{ and } \mathsf{ar}(a) = 0 \\ (f, \overline{q}, s) \xrightarrow{(f,i)} (q_i, s) & 1 \leq i \leq \mathsf{ar}(f) \end{array}$$

Let w be a finite or infinite word over the alphabet $\overline{\Sigma}$. We say that w is a *trace* of $\mathcal{A}$ if there is a possibly-infinite sequence of transitions $(q_I, \bot) \xrightarrow{\ell_1} \gamma_1 \cdots \xrightarrow{\ell_m} \gamma_m \xrightarrow{\ell_{m+1}} \cdots$ such that $w = \ell_1 \ell_2 \cdots$. We say that $\mathcal{A}$ *generates* the Σ-labelled tree t just if the branch language of t coincides with the set of traces of $\mathcal{A}$.

(ii) In case $\mathcal{S} = \langle \Gamma, n\text{-}\mathit{Stack}_\Gamma, \mathit{Op}_n, \mathit{top}_1, \bot_n \rangle$, we refer to a tree-generating $\mathcal{S}$-automaton as an *order-n pushdown tree-generating automaton* (PDA) (or simply order-n pushdown tree automaton) and specify it by the tuple $\langle \Gamma, Q, \Sigma, \delta, q_I \rangle$. For $n \geq 0$, we write $\mathit{PushdownTree}^\Sigma_n$ for the class of Σ-labelled ranked trees generated by order-n tree-generating pushdown automata.

Example 6. The tree of Example 2 is generated by the order-1 pushdown automaton $\langle \{Z, \bot\}, \{q_I, q_1, q_2\}, \Sigma, \delta, q_I \rangle$ where δ is defined as follows:

$$\begin{array}{ll} (q_I, \bot) \mapsto (f, q_1\, q_2) & (q_2, \bot) \mapsto \mathit{push}^Z_1\,;(f, q_1\, q_2) \\ (q_1, \bot) \mapsto (a, \epsilon) & (q_1, Z) \mapsto \mathit{pop}_1\,;(g, q_1). \\ (q_2, Z) \mapsto \mathit{push}^Z_1\,;(f, q_1\, q_2) & \end{array}$$

Expressivity: We conclude this subsection with a number of equi-expressivity results. It turns out that the class of trees generated by higher-order pushdown automata may be defined equivalently using two other methods, thereby underlying its robustness as a collection of trees.

In [54], Knapik, Niwiński and Urzyczyn introduced the class $\mathit{SafeRecTree}^\Sigma_n$ of Σ-labelled trees that are generated by order-n *homogeneously typed* recursion schemes satisfying a syntactic constraint called *safety*. (Homogeneous types and safety are defined in Section II-B.) They proved that for every $n \geq 0$, $\mathit{SafeRecTree}^\Sigma_n = \mathit{PushdownTree}^\Sigma_n$. Thus $\mathit{SafeRecTree}^\Sigma_0$ are the regular trees; and $\mathit{SafeRecTree}^\Sigma_1$ are those that are generated by order-1 pushdown automata.

At about the same time, Caucal [17] introduced a hierarchy of trees and a hierarchy of graphs, which are defined by mutual recursion via a pair of powerful functions: the function from graphs to trees is the standard unravelling operation, and the function from trees to graphs is given by inverse rational mappings. The construction starts from level 0 of the graph hierarchy, which consists of finite graphs. Since the functions preserve the decidability of monadic second-order (MSO) theories, all structures in each of the two hierarchies have decidable MSO theories. In addition Caucal [17] showed that $\mathit{SafeRecTree}^\Sigma_n$ coincides with $\mathit{CaucalTree}^\Sigma_n$, which consists of Σ-labelled trees that are obtained from the regular Σ-labelled-trees (i.e. trees from $\mathit{PushdownTree}^\Sigma_0$) by iterating n-times the operation of inverse deterministic rational mapping followed by unravelling. To summarise:

Theorem 7. *For $n \geq 0$, $\mathit{SafeRecTree}^\Sigma_n = \mathit{PushdownTree}^\Sigma_n = \mathit{CaucalTree}^\Sigma_n$.*

C. Homogeneous Types and the Safety Constraint

Safety [54] is a syntactic constraint on the rewrite rules of a recursion scheme. Roughly speaking, it specifies whether a formal parameter of a rewrite rule may occur in a subterm of the RHS of the rule, depending on the position of the subterm, and the respective orders of the parameter and the subterm.

Definition 8. (i) A type $A_1 \to \cdots \to A_n \to o$ is *homogeneous* if each A_i is homogeneous, and $\mathit{ord}(A_1) \geq \mathit{ord}(A_2) \geq \cdots \geq \mathit{ord}(A_n)$. It follows that the base type o is homogeneous. A term (or a rewrite rule or a recursion scheme) is *homogeneously typed* if all types that occur in it are homogeneous.

(ii) A rewrite rule $F\, y_1 \cdots y_k \to t$ is *safe* if for each subterm s of t that occurs in the operand position (i.e. as the second argument) of an application, for every $1 \leq j \leq k$, if the parameter y_j occurs in s then $\mathit{ord}(s) \leq \mathit{ord}(y_j)$.

(iii) A recursion scheme is *safe* if it is homogeneously typed and all its rewrite rules are safe.

For example, the order-2 rule

$$F\, \varphi\, x\, y \;\to\; f\, (F\, \underline{(F\, \varphi\, y)}\, y\, (\varphi\, x))\, a$$

where $F : (o \to o) \to o \to o \to o$ and $f : o \to o \to o$, is unsafe because the order-0 parameter y occurs in the underlined order-1 subterm, which is in an operand position. Note that it follows from the definition that order-0 and order-1 recursion schemes are always safe. Safety may be regarded as a reformulation of Damm's *derived types* [28]; see de Miranda's thesis [33] for a proof of equivalence.

Remark 9. The definition of safe recursion schemes by Knapik et al. [54] assumes that all types are homogeneous. In an unpublished note [8], Blum and Broadbent have shown that (i) homogeneity is a redundant condition for safety: for generating ranked trees, homogeneously-typed safe recursion schemes are equi-expressive as safe recursion schemes, (ii) homogeneity is redundant in general i.e. homogeneously-typed unsafe recursion schemes are equi-expressive as unsafe recursion schemes.

What is the point of safety? Though somewhat unwieldy as a syntactic constraint, safety does have a clear algorithmic value. It is a well-known fact of symbolic logic and formal systems such as the lambda calculus that *capture-permitting substitution is unsound*. Therefore, when performing substitution, one must take care to avoid variable capture, for example, by renaming bound variables. Remarkably, in safe recursion schemes, it is a relatively straightforward consequence of the definition that capture-permitting substitution *is* sound; in other words, it is safe *not* to rename bound variables. It follows that no fresh names are needed when computing the value tree of a safe recursion scheme. Knapik et al. [54] used this fact in their proof of the decidability of the MSO theories of the value trees of safe recursion schemes.

In view of their algorithmic advantage (namely, space efficiency), it is pertinent to ask if safe recursion schemes are less expressive. The safe lambda calculus [9], [7] is a formalisation of safety as a subsystem of the simply-typed lambda calculus. Blum and Ong [9] showed that, using Church numerals, the numeric functions representable by the simply-typed safe lambda-terms are exactly the multivariate polynomials. This should be contrasted with the classical result of Schwichtenberg [88]: the numeric functions representable by simply-typed lambda-terms are the multivariate polynomials augmented with conditionals. For a systematic study of the safe lambda calculus, including its expressivity, complexity, (game) semantics and categorical characterisation, see Blum's doctoral thesis [7].

D. Maslov's Pushdown Hierarchy of Word Languages

In the original 1974 paper [68], Maslov mentioned in brief several formalisms that are equivalent to higher-order pushdown automata. In a subsequent paper in 1976 [69], he set out the details of one such formalism, called *higher-order indexed grammars*. Using a key operation of "raising a language to a power given by another language", higher-order indexed grammars generalise Aho's indexed grammars [5] (which are the order-2 indexed grammars) to all finite orders; further, infinite-order indexed grammars define exactly the recursively enumerable languages.

By viewing a word as a linear tree i.e. a tree with branching factor at most one, we can use (nondeterministic) recursion schemes as generators of finite-word languages. In [27], [28], Damm studied a system of recursion schemes that are constrained by *derived types*. Damm and Goerdt [28], [30] showed that, as generators of word languages, order-n safe recursion schemes are equivalent to order-n pushdown automata, which are in turn equivalent to order-n indexed grammars.

Theorem 10 (Maslov 1976, Damm 1982, and Damm & Goerdt 1986)**.** *For each $n \geq 0$, the following formalisms define the same class of word languages:*

(i) order-n pushdown automata
(ii) order-n indexed grammars
(iii) order-n safe recursion schemes.

The low-order languages of Maslov's pushdown hierarchy are well-known. The order-2 languages are indexed languages; much studied in the 1970s and early 1980s ([47], [40], [77], [34]), they remain of interest to computational linguistics.

Several basic properties of the pushdown hierarchy of word languages were proved by Maslov [68], [69].

Theorem 11 (Maslov 1974)**.** *Let $n \geq 0$.*
(i) The emptiness problem for order-n PDA is decidable.
(ii) The order-n languages form an abstract family of languages[1].
(iii) Higher-order PDA define an infinite hierarchy of word languages.

Much of what is known about the complexity of languages recognisable by automata with higher-order stacks (and other auxiliary memory) is due to Engelfriet. His seminal paper [37] contains a wealth of results. We pick out a few that underpin the algorithmics of infinite structures generated by higher-order pushdown automata (PDA).

Theorem 12 (Engelfriet 1991)**.** *Let $s(n) \geq \log(n)$.*

(i) For $k \geq 0$, the word acceptance problem of nondeterministic order-k PDA with a two-way work-tape with $s(n)$ space is k-EXPTIME complete.
(ii) For $k \geq 1$, the word acceptance problem of alternating order-k PDA with a two-way work-tape with $s(n)$ space is $(k-1)$-EXPTIME complete.
(iii) For $k \geq 0$, the word acceptance problem of alternating order-k PDA is k-EXPTIME complete.
(iv) For $k \geq 1$, the emptiness problem of nondeterministic order-k PDA is $(k-1)$-EXPTIME complete.

Recent advances: Kartzow and Parys [53] established a pumping lemma for the ϵ-closure of collapsible pushdown graphs at each order. As a corollary, they constructed the first examples that separate the orders of the collapsible pushdown tree and graph hierarchies. Using a pumping lemma based on an intersection type system for reasoning about the reduction of λ-terms, Kobayashi [61] obtained an alternative proof of the strictness of the hierarchy of trees generated by higher-order recursion schemes (equivalently by collapsible pushdown tree automata, thanks to Theorem 23).

Another significant development was the following [51].

Theorem 13 (Inaba and Maneth 2008)**.** *Every word language of the Maslov Hierarchy is context sensitive.*

By considering word languages in the image of iterated composites of macro tree transducers, Inaba and Maneth [51] show that languages of the Hierarchy are in nondeterministic linear space (and so context sensitive) and NP-complete.

Are word languages generated by unsafe recursion schemes also context sensitive? Thanks to [3], we know that for

[1]An *abstract family of languages* is a collection of languages closed under union, concatenation, Kleene star, intersection with regular languages, homomorphism and inverse homomorphism.

generating word languages, order-2 nondeterministic safe recursion schemes are equi-expressive as order-2 unsafe recursion schemes. The case of order 3 was recently settled by Kobayashi et al. [63].

Theorem 14 (Kobayashi, Inaba and Tsukada 2014)**.** *The tree languages generated by order-2 unsafe recursion schemes are context sensitive. Hence order-3 unsafe word languages are context sensitive.*

Their proof uses intersection types to construct transformations of recursion schemes.

III. Model Checking Recursion Schemes

What classes of infinite structures have decidable monadic second-order (MSO) theories? One of the best known (and first) examples of such a class are the *regular trees* as studied by Rabin in a landmark paper in 1969 [82]. Muller and Shupp [70] subsequently proved that the *configuration graphs of pushdown systems* also have decidable MSO theories. In the 1990s, as finite-state technologies matured, researchers turned their attention to software verification and hence infinite-state model checking. A highlight was Caucal's result [16] that *prefix-recognisable graphs* (equivalently the ϵ-closure of configuration graphs of pushdown systems) have decidable MSO theories. Another concerned *algebraic trees*, which are trees generated by context-free tree grammars. Courcelle [22] showed that these trees have decidable MSO theories. In 2002, work by Knapik et al. [54] and Caucal [17] significantly extended and unified earlier developments.

Theorem 15 (Knapik et al. 2002 & Caucal 2002)**.** *For each $n \geq 0$, all trees in $SafeRecTree_n^\Sigma = PushdownTree_n\Sigma = CaucalTree_n^\Sigma$ have decidable MSO theories.*

We give an outline of the proof [54] which is by induction on n. Given an order-$(n{+}1)$ safe recursion scheme G, consider an associated tree, call it $\daleth_G$, which is obtained by contracting all the order-1 β-redexes in the rewrite rules of G. The tree $\daleth_G$ coincides with the tree generated by an order-n recursion scheme G^α i.e. $\daleth_G = [\![G^\alpha]\!]$; further the MSO theory of the original order-$(n + 1)$ tree $[\![G]\!]$ is reducible to that of the order-n tree $[\![G^\alpha]\!]$ i.e. there is a computable transformation of MSO sentences $\varphi \mapsto \varphi'$ such that $[\![G]\!] \vDash \varphi$ iff $[\![G^\alpha]\!] \vDash \varphi'$ [54]. Thanks to the safety assumption, it is sound to contract β-redexes using *capture-permitting* substitution i.e. without renaming bound variables. It follows that one can construct the tree $\daleth_G$ using only the original variables of the recursion scheme G. The same construction on an arbitrary recursion scheme would require an infinite set of fresh variable names.

Some Key Questions: Assuming safety, recursion schemes have decidable MSO theories, and their expressivity is characterised by higher-order pushdown automata. Yet safety seems an unnatural condition, both syntactically and semantically. Is safety really essential for these desirable properties? We consider a number of key questions.

Q1. *MSO Decidability*: Do trees generated by recursion schemes have decidable MSO theories?

Q2. *Automata-Theoretic Characterisation*: Find a class of automata that characterise the expressive power of recursion schemes. Specifically, can higher-order pushdown automata be so extended that they define the same class of ranked trees as recursion schemes?

Q3. *Graph Families*: Is there a good definition of graphs generated by recursion schemes? We expect the unravelling of these graphs to coincide with the value trees of recursion schemes. What theories of these graphs are decidable?

Q4. *Expressivity*: Does safety really constrain expressivity? Are there *inherently unsafe* word languages / trees / graphs?

The rest of the section is devoted to these questions.

A. MSO Decidability

A first partial answer to Q1 was obtained by Aehlig et al. [4]; they showed that all trees up to order 2, whether safe or not, and whether homogeneously typed or not, have decidable MSO theories. Independently, Knapik et al. obtained a slightly stronger result [55]. The question was answered positively by Ong [73].

Theorem 16 (Ong 2006)**.** *For each $n \geq 0$ the modal mu-calculus model checking problem for trees generated by order-n recursion schemes (i.e. given an order-n recursion scheme $\mathcal{G}$ and a mu-calculus formula φ, does $[\![\mathcal{G}]\!]$ satisfy φ at the root?) is n-*EXPTIME *complete.*

Since MSOL and the modal mu-calculus are equi-expressive over trees, it follows that these trees have decidable MSO theories. In the following we sketch a proof of the theorem.

Thanks to Emerson and Jutla [36], we can reduce the mu-calculus model checking problem to an alternating parity tree automaton (APT) acceptance problem: *Given an order-n recursion scheme $\mathcal{G}$ and an APT $\mathcal{B}$, does $\mathcal{B}$ have an accepting run-tree over the generated tree $[\![\mathcal{G}]\!]$?* The proof has two main ingredients.

I. The first is a *transference principle* from the value tree to an auxiliary *computation tree*, which is regular. Using game semantics [49], we establish a strong correspondence between *paths* in the value tree and *traversals* in the computation tree. This allows us to prove that the APT $\mathcal{B}$ has an accepting run-tree over the value tree if, and only if, it has an accepting *traversal-tree* over the computation tree.

II. The second ingredient is the simulation of an accepting traversal-tree by a certain set of annotated paths over the computation tree: we construct a *traversal-simulating* APT, $\widehat{\mathcal{B}}$, as a recognising device for this set of paths.

Thus the APT $\mathcal{B}$ has an accepting run-tree over $[\![\mathcal{G}]\!]$ if, and only if (thanks to I), $\mathcal{B}$ has an accepting *traversal-tree* over the *computation tree* $\lambda(\mathcal{G})$ if, and only if (thanks to II), the traversal-simulating APT $\widehat{\mathcal{B}}$ has an accepting run-tree over $\lambda(\mathcal{G})$, which is decidable, because the tree $\lambda(\mathcal{G})$ is regular.

Model Checking via Intersection Types

In [57], Kobayashi and Ong gave a different proof of Theorem 16 that uses idempotent intersection types. They exhibit a transformation that, given an alternating parity tree automaton (APT) $\mathcal{A}$, effectively constructs a type system $\mathcal{K}_{\mathcal{A}}$ such that a recursion scheme is typable in $\mathcal{K}_{\mathcal{A}}$ if, and only if, the tree it generates is accepted by the APT $\mathcal{A}$. The model checking problem is thus reduced to a type checking problem. We present the type system $\mathcal{K}_{\mathcal{A}}$ as follows.

Definition 17. Fix an APT $\mathcal{A} = \langle \Sigma, Q, \delta, q_I, \Omega\rangle$ where Q is a set of states, $q_I \in Q$ is the initial state, $\delta : \Sigma \times Q \to \mathsf{B}^+(\{1, \cdots, M\} \times Q)$ is the transition map (where $M = \max\{\mathsf{ar}(f) : f \in \Sigma\}$, and $\mathsf{B}^+(X)$ is the set of positive Boolean formulas over X), and $\Omega : Q \to \{0, 1, \cdots, p\}$ is the priority map. The (raw) *intersection types* are defined by:

$$\theta ::= q \mid \tau \to \theta$$
$$\tau ::= \bigwedge\{(\theta_1, m_1), \cdots, (\theta_k, m_k)\} \text{ (written } \textstyle\bigwedge_{i=1}^{k}(\theta_i, m_i))$$

where $q \in Q$ and $m \in \{0, \cdots, p\}$ (the priorities of $\mathcal{A}$). We extend the priority map Ω of $\mathcal{A}$ to intersection types by $\Omega(\tau \to \theta) := \Omega(\theta)$. Henceforth we restrict ourselves to intersection types that refine simple types. To capture these types, we say that an intersection type τ (respectively, *prime* intersection type θ) is *well-formed* if (i) $\tau :: \kappa$ (respectively, $\theta ::_{\mathsf{p}} \kappa$) for some κ, and (ii) for each subexpression of the form $(\bigwedge_{i=1}^{k}(\theta_i, m_i)) \to \theta'$, we have $m_i \geq \max(\Omega(\theta'), \Omega(\theta_i)$ for each $1 \leq i \leq k$; where the *refinement relations*, $\tau :: \kappa$ and $\theta ::_{\mathsf{p}} \kappa$, are defined by induction over the rules:

$$\frac{q \in Q}{q ::_{\mathsf{p}} o} \qquad \frac{\tau :: \kappa_1 \quad \theta ::_{\mathsf{p}} \kappa_2}{\tau \to \theta \; ::_{\mathsf{p}} \; \kappa_1 \to \kappa_2} \qquad \frac{\theta_i ::_{\mathsf{p}} \kappa \quad (\forall 1 \leq i \leq n)}{\bigwedge_{i=1}^{n}(\theta_i, m_i) \; :: \; \kappa}$$

Typing judgements of the system $\mathcal{K}_{\mathcal{A}}$ have the form $\Gamma \vdash_{\mathcal{A}} s : \theta$ where the environment Γ is a set of bindings $x : (\theta, m)$ with x ranging over Var (we assume $\mathcal{N} \subset \mathit{Var}$). Valid judgements are defined by induction over the following rules. We write $\Gamma, x : \bigwedge_{i=1}^{k}(\theta_i, m_i)$ as a shorthand for $\Gamma \cup \{x : (\theta_1, m_1), \cdots, x : (\theta_k, m_k)\}$ where x is assumed *not* to occur in Γ.

$$\frac{}{x : (\theta, \Omega(\theta)) \vdash_{\mathcal{A}} x : \theta}$$

$$\frac{\begin{array}{c}\{(i, q_{ij}) \mid 1 \leq i \leq n, j \in J_i\} \text{ satisfies } \delta(q, a) \\ m_{ij} = \max(\Omega(q_{ij}), \Omega(q)) \; (1 \leq i \leq n, j \in J_i)\end{array}}{\emptyset \vdash_{\mathcal{A}} a : \bigwedge_{j \in J_1}(q_{1j}, m_{1j}) \to \cdots \to \bigwedge_{j \in J_n}(q_{nj}, m_{nj}) \to q}$$

$$\frac{\Gamma_0 \vdash_{\mathcal{A}} s : \bigwedge_{i \in I}(\theta_i, m_i) \to \theta \qquad \Gamma_i \vdash_{\mathcal{A}} t : \theta_i \; (\forall i \in I)}{\Gamma_0 \cup \bigcup_{i \in I}(\Gamma_i \Uparrow m_i) \vdash_{\mathcal{A}} s\, t : \theta}$$

$$\frac{\Gamma, x : \bigwedge_{i \in I}(\theta_i, m_i) \vdash_{\mathcal{A}} t : \theta \qquad I \subseteq J}{\Gamma \vdash_{\mathcal{A}} \lambda x.t : \bigwedge_{i \in J}(\theta_i, m_i) \to \theta}$$

where $\Gamma \Uparrow m := \{F : (\theta, \max(m, m')) \mid F : (\theta, m') \in \Gamma\}$.

Now given a recursion scheme $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S\rangle$ (for each $F\, x_1 \cdots x_n \to e$ in $\mathcal{R}$, we write $\mathcal{R}(F) := \lambda x_1 \cdots x_n.e.$), we define a parity game

$$\mathbb{G}(\mathcal{A}, \mathcal{G}) = \langle V_{\mathbf{V}}, V_{\mathbf{R}}, (S, q_I, \Omega(q_I)), E, \Omega'\rangle$$

as follows.

$$\begin{aligned} V_{\mathbf{V}} &= \{(F, \theta, m) \mid F : \kappa \in \mathcal{N}, \theta ::_{\mathsf{p}} \kappa\} \\ V_{\mathbf{R}} &= \{\Gamma \mid \forall F : (\theta, m) \in \Gamma. F : \kappa \in \mathcal{N} \land \theta ::_{\mathsf{p}} \kappa\} \\ E &= \{((F, \theta, m), \Gamma) \mid \Gamma \vdash_{\mathcal{A}} \mathcal{R}(F) : \theta\} \\ &\quad \cup \{(\Gamma, (F, \theta, m)) \mid F : (\theta, m) \in \Gamma\} \end{aligned}$$

and the priority function Ω' maps (F, θ, m) to m and Γ to 0. We say that $\mathcal{G}$ is *typable in* $\mathcal{K}_{\mathcal{A}}$ if Player $\mathbf{V}$ has a winning strategy for $\mathbb{G}(\mathcal{A}, \mathcal{G})$.

The parity game $\mathbb{G}(\mathcal{A}, \mathcal{G})$ may be understood intuitively as follows. Player $\mathbf{V}$ (Verifier) tries to prove that the recursion scheme is typable, and Player $\mathbf{R}$ (Refuter) tries to disprove it. At a node (F, θ, m), Player $\mathbf{V}$ has to choose an environment Γ under which $\mathcal{R}(F)$ has type θ. Player $\mathbf{R}$ then chooses a binding $F' : (\theta', m')$ from Γ, and challenges $\mathbf{V}$ to show why F' has type θ', and then it is again $\mathbf{V}$'s turn to choose an environment Γ' under which $\mathcal{R}(F')$ has type θ'. The play either continues indefinitely or ends when one of the players is unable to move. Player $\mathbf{V}$ wins a play if at some point, she chooses the empty environment (so that $\mathbf{R}$ cannot choose a binding), or if the play is infinite and satisfies the parity condition: the largest priority occurring infinitely often is even. The recursion scheme is typable if Player $\mathbf{V}$ has a strategy that wins every play, regardless of Player $\mathbf{R}$'s choice.

The main result in [57] is the following.

Theorem 18 (Kobayashi & Ong 2009)**.** *Given a ranked alphabet Σ, there is an algorithm that, given an APT $\mathcal{A}$, constructs a type system $\mathcal{K}_{\mathcal{A}}$ such that for every recursion scheme $\mathcal{G}$, $\mathcal{A}$ accepts $[\![\mathcal{G}]\!]$ if, and only if, $\mathcal{G}$ is typable in $\mathcal{K}_{\mathcal{A}}$.*

Example 19. Take Σ and $\mathcal{G}_1$ of Example 2, and take APT $\mathcal{A}_1 = \langle \Sigma, \{q_0, q_1\}, \delta_1, q_0, \{q_0 \mapsto 2, q_1 \mapsto 1\}\rangle$ where, for each $q \in \{q_0, q_1\}$, $\delta_1 : (q, a) \mapsto \mathsf{true}, (q, f) \mapsto (1, q) \land (2, q), (q, g) \mapsto (1, q_1)$.

(i) Then $\mathcal{A}_1$ accepts a Σ-labelled tree t if, and only if, in every path of t, a occurs eventually after g occurs. Note that it has a unique infinite path labelled by $(\epsilon, q_0)(2, q_0)(22, q_0)(222, q_0)\cdots$, which satisfies the parity condition.

(ii) Set $\theta = (q_0, 2) \land (q_1, 2) \to q_0$. Then the following judgements are valid:

$$\begin{aligned} F : (\theta, 2) &\vdash_{\mathcal{A}_1} F\, a : q_0 \\ F : (\theta, 2) &\vdash_{\mathcal{A}_1} \lambda x. f\, x\; (F(g\, x)) : \theta \end{aligned}$$

A (positional) winning strategy for the parity game $\mathbb{G}(\mathcal{A}_1, \mathcal{G}_1)$ is given by: $(S, q_0, 2) \mapsto \{F : (\theta, 2)\}, (F, \theta, 2) \mapsto \{F : (\theta, 2)\}$.

The standard typing for recursion can be considered a degenerate case of our definition (using parity games), where all the priorities are 0. In fact, our type system $\mathcal{K}_{\mathcal{A}}$ specialises to Kobayashi's system [59] when 0, or any even number, is the only priority. An advantage of this model checking algorithm is that it has an improved parameterised complexity: the time complexity is polynomial in the size of the recursion scheme, assuming that the types and the APT are fixed.

Other Proofs of the Decidability Theorem: Several other proofs of Theorem 16 have been published.

In [45], Hague et al. give a proof by reducing the modal mu-calculus model checking of recursion schemes to the problem of solving parity games over the configuration graphs of collapsible pushdown automata.

Salvati and Walukiewicz [84] give yet another proof of Theorem 16. They consider trees generated by ground-type $\lambda\mathbf{Y}$-terms, and use Krivine machine as a model of computation. Their proof is by reducing the problem of solving a parity game over the configurations of a Krivine machine to that of solving a finite parity game.

Complexity of Higher-Order Model Checking: As already mentioned, the worst-case time complexity of the modal mu-calculus model checking of order-n recursion schemes is n-EXPTIME complete [73]. This is so even for *safe* recursion schemes, and also for model checking with respect to *alternating trivial automata* (i.e. APT that have an even number as their only priority). However the complexity is $(n-1)$-EXPTIME complete with respect to *deterministic* trivial automata. If the largest arity of the terminal symbols and the size of the formula are both fixed, then the time complexity is polynomial in the size of the recursion scheme [57]. Under the same assumption, for the class of trivial automata, the time complexity is linear in the size of the recursion scheme [58]. For other results on the complexity of higher-order model checking, we refer the reader to [65], [46].

B. Automata-Theoretic Characterisation

We aim to answer Q2 in this subsection. Collapsible pushdown automata (CPDA) are a variant of higher-order pushdown automata in which every symbol in the stack has a link to a prefix of the stack. In addition to the higher-order stack operations $push_i$ and pop_i, CPDA have an important operation called $collapse$, whose effect is to "collapse" the stack s to the prefix indicated by the link from the $top_1(s)$. The main result is that for every $n \geq 0$, order-n recursion schemes and order-n CPDA are equi-expressive as generators of ranked trees.

Let Γ be a stack alphabet and $n \geq 1$. An *order-n collapsible stack* s is an order-n stack such that every non-$\perp$ symbol that occurs in it has a link to a collapsible stack (of order k) situated below it in s; we call the link a $(k+1)$*-link*. Note that k is necessarily less than n. We shall abbreviate order-n collapsible stack to n-stack, whenever it is clear form the context. The *empty k-stack* is defined as before. When displaying examples of n-stacks, we shall omit $\perp$ and 1-links (i.e. links to stack symbols) to avoid clutter; thus we write $[\,[\,]\,[\,a\,b\,]\,]$ instead of $\perp\, a\, b$

For $n \geq 2$ the set $Op_n^{\dagger}$ of *order-n operations on collapsible stacks* consists of the following four types of operations:

(i) pop_k for each $1 \leq k \leq n$
(ii) $collapse$
(iii) $push_1^{a,k}$ for each $1 \leq k \leq n$ and each $a \in (\Gamma \setminus \{\perp\})$
(iv) $push_j$ for each $2 \leq j \leq n$.

The operation pop_i is defined as before in Section II-B. Let s be an n-stack and $2 \leq i \leq n$. To construct $push_1^{a,i}\, s$, we first attach a link from a fresh copy of a to the $(i-1)$-stack that is immediately below the top $(i-1)$-stack of s, and then push the symbol-with-link onto the top 1-stack of s. As for $collapse$, suppose the top_1-symbol of s has a link to a (particular occurrence of) k-stack u in s. Then $collapse\ s$ causes s to "collapse" to the prefix s_0 of s such that $top_{k+1}\, s_0 = u$. Finally, for $j \geq 2$, the *order-j push* operation, $push_j$, simply takes a stack s and duplicates the top $(j-1)$-stack of s, preserving its link structure.

Example 20. Take the 3-stack $s = [\,[\,[\,a\,]\,]\;[\,[\,]\,[\,a\,]\,]\,]$. We have (to avoid clutter, we only display the important links)

$$push_1^{b,2}\, s = [\,[\,[a]\,]\;[\,[\,]\;[a\;b\,]\,]\,]$$
$$collapse\,(push_1^{b,2}\, s) = [\,[\,[a]\,]\;[\,[\,]\,]\,]$$
$$\underbrace{push_1^{c,3}(push_1^{b,2}\, s)}_{\theta} = [\,[\,[a]\,]\;[\,[\,]\;[a\;b\;c\,]\,]\,]$$

Then $push_2\,\theta$ and $push_3\theta$ are respectively

$$[\,[\,[a]\,]\;[\,[\,]\;[a\;b\;c\,]\,[a\;b\;c\,]\,]\,]$$

$$[\,[\,[a]\,]\;[\,[\,]\;[a\;b\;c\,]\,]\,[\,[\,]\;[a\;b\;c\,]\,]\,]\,.$$

We have $collapse\,(push_2\,\theta) = collapse\,(push_3\,\theta) = collapse\,\theta = [\,[\,[\,a\,]\,]\,]$.

As in the case of order-n stacks, the tuple $\langle \Gamma, n\text{-}Stack_\Gamma^{\dagger}, Op_n^{\dagger}, top_1, \perp_n\rangle$ is called the *system of order-n collapsible stacks*, which is an abstract store system. We shall use automata equipped with order-n collapsible stacks to define word languages and trees, and (in Section III-C) infinite graphs.

Definition 21. Let $\mathcal{S} = \langle \Gamma, n\text{-}Stack_\Gamma^{\dagger}, Op_n^{\dagger}, top_1, \perp_n\rangle$ be the system of order-n collapsible stacks over Γ. We refer to a word-language $\mathcal{S}$-automaton $\langle \mathcal{S}, Q, \Sigma, \Delta, q_I, F\rangle$ as an *order-n collapsible pushdown word-language automaton*, and specify it as $\langle \Gamma, Q, \Sigma, \Delta, q_I, F\rangle$. Similarly we refer to a tree-generating $\mathcal{S}$-automaton $\langle \mathcal{S}, Q, \Sigma, \delta, q_I\rangle$ as an *order-n collapsible pushdown tree-generating automaton*, and specify it as $\langle \Gamma, Q, \Sigma, \delta, q_I\rangle$.

Example 22 (Aehlig, de Miranda and Ong 2005)**.** We define the language U over the alphabet $\{\mathbf{(}, \mathbf{)}, *\}$ as follows. A *U-word* is composed of 3 segments:

$$\underbrace{\mathbf{(}\cdots\mathbf{(}\cdots\mathbf{(}}_{A}\;\underbrace{\mathbf{(}\cdots\mathbf{)}\cdots\mathbf{(}\cdots\mathbf{)}}_{B}\;\underbrace{*\cdots*}_{C}$$

- Segment A is a prefix of a well-bracketed word that ends in $\mathbf{(}$, and the opening $\mathbf{(}$ is not matched in the (whole) word.
- Segment B is a well-bracketed word.
- Segment C has length equal to the number of $\mathbf{(}$ in A.

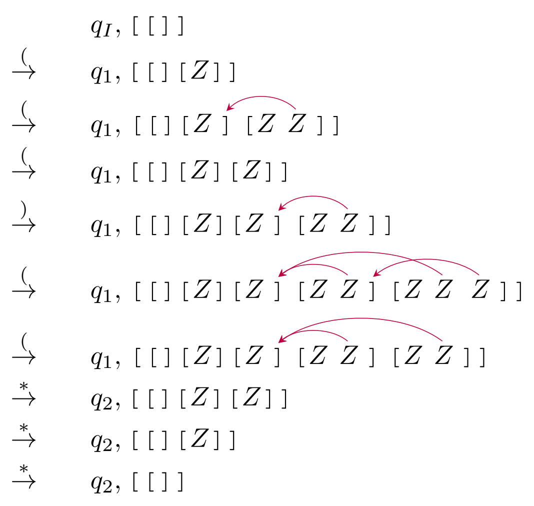

q_I, [[ ]]
q_1, [[ ] [Z]]
q_1, [[ ] [Z] [Z Z]]
q_1, [[ ] [Z] [Z]]
q_1, [[ ] [Z] [Z] [Z Z]]
q_1, [[ ] [Z] [Z] [Z Z] [Z Z Z]]
q_1, [[ ] [Z] [Z] [Z Z] [Z Z]]
q_2, [[ ] [Z] [Z]]
q_2, [[ ] [Z]]
q_2, [[ ]]

Fig. 1. The computation of the U-word **(()(()** $***$.

It is a consequence of the definition that every U-word has a unique decomposition. For example, **(()**$\underline{\textbf{(()(())}}$$***$ is in U; its B-segment is underlined. For each $n \geq 0$, the word **((**^{n} **)**^{n} **(** $*^n **$ is in U, the respective B-segments are all empty.

The language U is recognisable by a *deterministic* order-2 CPDA $\langle \{q_I, q_1, q_2\}, \{\textbf{(}, \textbf{)}, *\}, \{\bot, Z\}, \delta, q_I, \{q_2\}\rangle$, where $\delta : Q \times \Sigma \times \Gamma \to Op_n^* \times Q$ is as follows:

$$\begin{array}{ll} (q_I, \textbf{(}, \bot) \mapsto (push_2\,; push_1^Z, q_1) & (q_1, \textbf{)}, Z) \mapsto (pop_1, q_1) \\ (q_1, \textbf{(}, Z) \mapsto (push_2\,; push_1^Z, q_1) & (q_2, *, Z) \mapsto (pop_2, q_2) \\ (q_1, *, Z) \mapsto (collapse, q_2) & \end{array}$$

The idea is that the pair $(q_1, Z) \in Q \times \Gamma$ indicates that the number of "**(**" read, minus the number of "**)**" read, is at least one. Note that $(q_1, \bot)$ indicates a "stuck configuration" which is reachable upon reading e.g. **()**. To illustrate, we present the computation of the U-word **(()(()** $***$ in Figure 1. (In the figure, we only display certain links.)

It follows from [3] that U is recognisable by a *non-deterministic* order-2 pushdown automaton. This illustrates the power of collapse.

The main result of this subsection is the following equi-expressivity result [45].

Theorem 23 (Hague, Murawski, Ong and Serre 2008)**.** *For every $n \geq 0$, order-n recursion schemes and order-n collapsible pushdown tree automata define the same class of Σ-labelled trees.*

Since the construction of the CPDA-transform in the scheme-to-automata translation is based on game semantics [49], Theorem 23 also gives an automata-theoretic characterisation of innocent strategies. There are several machine-theoretic representations of innocent game semantics in the literature: PAM, the Pointer Abstract Machine of Danos et al. [31], may be viewed as an implementation of linear head reduction of lambda-terms; Curien and Herbelin [25], [26] used abstract Böhm trees as the basis of a family of abstract machines. Compared to these machines, the characterisation by CPDA seems clearly syntax-independent: contrast, for example, the type-theoretic notion of order with the order of collapsible stacks.

Blum and Broadbent [8] recently proved that if the recursion scheme G is safe, then the CPDA-transform, CPDA(G), does not use *collapse* in its computation. In [15], Carayol and Serre gave a syntactic proof of Theorem 23. Their scheme-to-CPDA translation does not use game semantics.

C. *Parity Games over CPDA Configuration Graphs*

Definition 24. (i) Let $\mathcal{S} = \langle \Gamma, \mathit{Store}_\Gamma, \mathit{Op}, \mathit{top}, \bot\rangle$ be a store system. An *$\mathcal{S}$-transition system* is a tuple $\langle \mathcal{S}, Q, \Delta, q_I\rangle$ where Q is a finite set of control-states, $q_I \in Q$ is the initial state, and $\Delta \subseteq Q \times \Gamma \times Q \times \mathit{Op}$ is the transition relation. A *configuration* is a pair (q, s) where $q \in Q$ and $s \in \mathit{Store}_\Gamma$; and $(q_I, \bot)$ is the initial configuration. The transition relation Δ induces a (labelled) transition relation between configurations according to the rule: $(q, s) \xrightarrow{(q', \theta)} (q', \theta(s))$ provided $(q, \mathit{top}(s), q', \theta) \in \Delta$. The *configuration graph* of an *$\mathcal{S}$-transition system* is a directed graph whose vertices are the configurations, and edge-set is the induced transition relation.

(ii) In case $\mathcal{S} = \langle \Gamma, n\text{-}\mathit{Stack}_\Gamma, \mathit{Op}_n, \mathit{top}_1, \bot_n\rangle$, the system of order-$n$ stacks over Γ, we refer to the configuration graphy of the $\mathcal{S}$-transition system $\langle \mathcal{S}, Q, \Delta, q_I\rangle$ as an *order-n pushdown automaton graph* (order-n PDA graph).

(iii) In case $\mathcal{S} = \langle \Gamma, n\text{-}\mathit{Stack}_\Gamma^\dagger, \mathit{Op}_n^\dagger, \mathit{top}_1, \bot_n\rangle$, the system of order-$n$ collapsible stacks over Γ, we refer to the configuration graph of a $\mathcal{S}$-transition system $\langle \mathcal{S}, Q, \Delta, q_I\rangle$ as an *order-n collapsible pushdown automaton graph* (order-n CPDA graph).

Example 25 (An undecidable CPDA graph)**.** Take the order-2 CPDA graph with state-set $\{0, 1, 2\}$, stack alphabet $\{a, b, \bot\}$ and transition relation given by

$$\{(0, -, 1, t),\ (1, -, 0, a),\ (1, -, 2, b),\ (2, \dagger, 2, 1),\ (2, \dagger, 0, 0)\}$$

where $-$ means any symbol, $\dagger$ means any non-$\bot$ symbol, and $t, a, b, 0$ and 1 are shorthand for the stack operations $push_2$, $push_1^{a,2}$, $push_1^{b,2}$, *collapse* and pop_1 respectively. We present its configuration graph (with edges labelled by stack operations only) in Figure 2.

Let $G = \langle V, E\rangle$ be the configuration graph of an $\mathcal{S}$-transition system $\mathcal{A}$, and $Q_{\mathbf{E}} \cup Q_{\mathbf{A}}$ be a partition of Q, and let $\Omega : Q \to \{0, \cdots M-1\}$ be a priority function. Together they define a partition $V_{\mathbf{E}} \cup V_{\mathbf{A}}$ of $\mathbf{V}$ whereby a vertex belongs to $V_{\mathbf{E}}$ if and only if its control state belongs to $Q_{\mathbf{E}}$, and a priority function $\Omega : V \to \{0, \cdots, M-1\}$ where a vertex is assigned the priority of its control state. We call the structure $\mathcal{G} = \langle G, V_{\mathbf{E}}, V_{\mathbf{A}}\rangle$ an *order-n CPDA game graph* and the pair $\mathbb{G} = \langle \mathcal{G}, \Omega\rangle$ an *order-n CPDA parity game*.

Another consequence of Theorem 23 is that it gives new techniques for model checking or solving games played on infinite structures generated by automata. In particular it leads to new proofs or optimal algorithms for the special cases that have been considered previously [97], [93], [55]. Actually, the

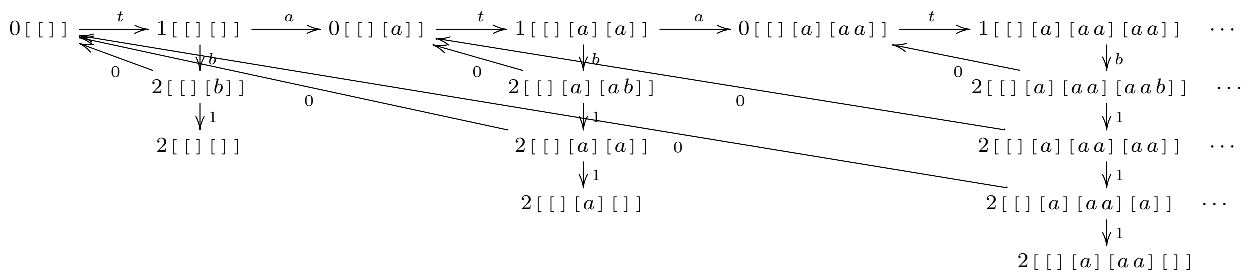


Fig. 2. A order-2 CPDA graph with an undecidable MSO theory

techniques of Walukiewicz [97] and Knapik et al. [55] can be generalised to solve order-n CPDA parity games without reference to Ong's work [73]. Further they give effective winning strategies for the winning player (which was not the case in [55] where the special case $n = 2$ was considered).

Theorem 26 (Hague, Murawski, Ong and Serre 2008)**.** *The problem of solving an order-n CPDA parity game is n-EXPTIME complete. Furthermore one can build an order-n collapsible pushdown transducer (i.e. automaton with output) that realises a winning strategy for the winning player.*

Remark 27. This result can be generalised to the case where the game has an arbitrary ω-regular winning condition, and is played on the ϵ-closure of the configuration graph of an order-n CPDA graph. Consequently parity games on Caucal graphs [17], [93] are a special case of this problem.

The Caucal graphs have decidable MSO theories [17]. Do the configuration graphs of CPDA also have decidable MSO theories?

Proposition 28 (Hague, Murawski, Ong and Serre 2008)**.** *There is an order-2 CPDA whose configuration graph has an undecidable MSO theory. Hence the class of ϵ-closure of configuration graphs of CPDA strictly contains the Caucal graphs.*

For a proof, recall that MSO interpretation preserves MSO decidability. Now consider the following MSO interpretation I of the configuration graph of the order-2 CPDA in Example 25:

$$I_A(x, y) = x \xrightarrow{C} y \wedge x \xrightarrow{R} y \qquad I_B(x, y) = x \xrightarrow{1} y$$

with $C = \overline{1}^* \, \overline{b} \, a \, t \, b \, 1^*$ and $R = 0 \, t \, a \, \overline{0} \, \vee \, \overline{1} \, 0 \, t \, a \, \overline{0} \, 1$.

With reference to Figure 3, note that for the A-edges, the constraint C requires that the target vertex should be in the next column to the right, while R specifies the correct row. Observe that I's image is the "infinite half-grid" which has an undecidable MSO theory.

Decidability of first-order theories of CPDA graphs: While order-2 CPDA graphs already have undecidable MSO theories, Kartzow [52] has shown that first-order logic is decidable at order 2. Surprisingly first-order logic ceases to be decidable at order 3 and above [11].

Logical Reflection and Effective Selection: In [12] Broadbent et al. give the first characterisation of the winning regions

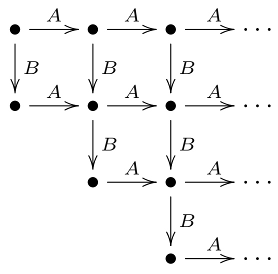


Fig. 3. An infinite half-grid.

of order-n CPDA parity games: they are regular sets defined by a class of automata. As a corollary, it is shown that recursion schemes are *reflective* with respect to MSOL and modal mu-calculus in the following sense.

Theorem 29 (Broadbent, Carayol, Serre and Ong 2010)**.** *There is an algorithm that transforms a given property φ and a recursion scheme $\mathcal{G}$ to a new recursion scheme $\mathcal{G}_\varphi$ that* reflects *φ, meaning that (i) $[\![\mathcal{G}_\varphi]\!]$ and $[\![\mathcal{G}]\!]$ have the same underlying tree, and (ii) the nodes that satisfy φ (and only those) have a special label in $[\![\mathcal{G}_\varphi]\!]$.*

Thus we may view $\mathcal{G}_\varphi$ as a transform of $\mathcal{G}$ that can "internally observe" its behaviour against a specification φ.

Carayol and Serre [15] have recently shown that the trees generated by recursion schemes enjoy *effective MSO selection*, a stronger property than logical reflection.

Theorem 30 (Carayol and Serre 2012)**.** *There is an algorithm that transforms a given MSO formula $\varphi(X)$ where X is a second-order variable, and a recursion scheme $\mathcal{G}$ satisfying $[\![\mathcal{G}]\!] \models \exists X.\varphi(X)$, to a new recursion scheme $\mathcal{G}^\varphi$ and a set U of nodes such that (i) $[\![\mathcal{G}]\!]$ and $[\![\mathcal{G}^\varphi]\!]$ have the same underlying tree, and (ii) $[\![\mathcal{G}]\!] \models \varphi(U)$, and (iii) the nodes from U (and only those) have a special label in $[\![\mathcal{G}^\varphi]\!]$.*

Thus the algorithm is a selector of a witness U of $[\![\mathcal{G}]\!] \models \exists X.\varphi(X)$.

D. Does Safety Constrain Expressivity?

Plainly the hierarchy of trees generated by safe recursion schemes is contained in the hierarchy of trees generated by arbitrary recursion schemes. But is the containment strict? This question was first raised by Knapik et al. [54] in 2002, and their conjecture came to be known as the *Safety Conjecture*: there are inherently unsafe trees. I.e. there is a tree, generated by an unsafe recursion scheme, which is not generatable by any safe recursion scheme. Central to the Conjecture is the question of whether the syntactic constraint of safety is also semantical; equivalently it concerns the expressive power of the stack operation of collapse on higher-order pushdown automata as tree generators. Though the Conjecture was widely known (it was mentioned in [3], [55], [73] and [45], among others) and considered important, virtually no progress was made for nearly a decade. The Conjecture was eventually proved by Parys [79] in 2012. He showed that there is a word language (such as U in Example 22) recognised by an order-2 deterministic collapsible pushdown automaton which is not recognisable by any deterministic higher-order pushdown automaton of any order. The following is then obtained as a corollary.

Theorem 31 (Parys 2012)**.** *There exists a tree generated by an order-2 RS (equivalently, by an order-2 collapsible pushdown automaton), which is not generatable by any safe RS of any order (equivalently, by any higher-order pushdown automaton of any order).*

IV. Model Checking Higher-type Böhm Trees & Compositional Higher-Order Model Checking

This section is about recent developments in the model checking of higher-type Böhm trees, compositional approaches to higher-order model checking, and the related topic of model checking by evaluation.

Like ordinary model checking, higher-order model checking has mainly been a whole-program analysis, in both theory and practice. This can seem surprising, since it contrasts with the rôle of higher order as enabling modular structuring of programs. Hitherto, higher-order model checking algorithms can analyse trees generated by recursion schemes (or equivalently the Böhm trees of ground-type $\lambda\mathbf{Y}$-terms with free variables of order at most one), which are the computation trees of ground-type functional programs that may contain higher-order subterms. It is important to extend these algorithms to model check the computation trees of *higher-type* functional programs, which are *trees with λ-binders* i.e. higher-type Böhm trees.

Compositional methods are typically guided and justified by a denotational semantics. Since Böhm trees are themselves models of the λ-calculus, we seek models that organise themselves into a cartesian closed category. Further, these models should be *strategy aware*, in the sense that they can interpret not just Böhm trees, but also witnesses of correctness properties of Böhm trees, namely, the accepting run trees of alternating parity automata. Thus, in order to support compositional higher-order model checking (and, in particular, the model checking of higher-type Böhm trees) with respect to ω-regular correctness properties, we need, in essence, a cartesian closed category of parity games.

However the algorithmic analysis of Böhm trees seems fraught with difficulties. The elegant theorems of "Rabin's Heaven" no longer hold: Example 32 presents a $\lambda\mathbf{Y}$-definable Böhm tree which has an undecidable MSO theory. Furthermore, Stirling's alternating dependency automata [92], a model of computation designed for the algorithmic analysis of Böhm trees, have an undecidable emptiness problem [76].

Example 32. Let $\Delta \vdash M :: (o \to o) \to o$ where $\Delta = a :: o, b :: o \to ((o \to o) \to o) \to o$ and

$$M = \mathbf{Y}\,(\lambda f^{o\to(o\to o)\to o}.\lambda y^{o}.\lambda x^{o\to o}.b\,(x\,y)\,(f\,(x\,y)))\,a.$$

The Böhm tree of M, $\mathrm{BT}(M)$, is displayed in Figure 4. Notice that in $\mathrm{BT}(M)$, infinitely many distinct names are required to represent all variable bindings; furthermore each name occurs infinitely often. When viewed as a graph, with back edges representing variable binding, $\mathrm{BT}(M)$ has an undecidable MSO theory [19].

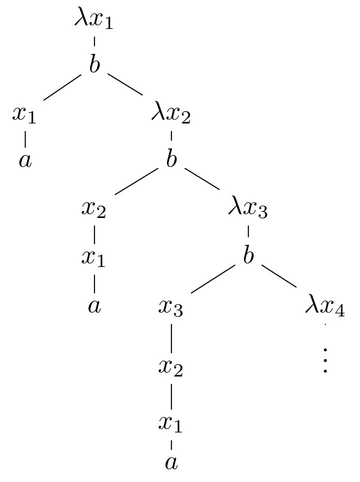


Fig. 4. Figure of the Böhm tree, $\mathrm{BT}(M)$, of Example 32

A. Compositional Model Checking of Higher-type Böhm Trees

Tsukada and Ong [96] have recently introduced a compositional approach to the model checking of higher-type Böhm trees with respect to a system of intersection types that refine simple types. Parameterised by a finite set Q of base types, and a *winning condition* $\langle \mathbb{E}, \mathbb{F}, \Omega \rangle$ – an algebraic formulation of ω-regular winning conditions, the intersection types are defined by:

$$\begin{array}{rcl} \theta & ::= & q \mid \tau \to \theta \\ \tau & ::= & \bigwedge_{i=1}^{n}(\theta_i, e_i) \end{array}$$

where $n \geq 0, q \in Q$ and $e_i \in \mathbb{E}$, the effect set. Note that the intersection types are the same as those of Kobayashi and Ong in Definition 17 [57], except that here they are generated from a winning condition, as opposed to an alternating parity automaton. A *winning condition* $\langle \mathbb{E}, \mathbb{F}, \Omega \rangle$ is given by an ordered ω-monoid [81],

$$\langle \mathbb{E}, \circ, \epsilon, \preceq_{\mathbb{E}},\ \mathbb{F}, \preceq_{\mathbb{F}},\ \circledast : \mathbb{E} \times \mathbb{F} \to \mathbb{F},\ \pi : \mathbb{E}^{\omega} \to \mathbb{F} \rangle,$$

such that the monoid $\langle \mathbb{E}, \circ, \epsilon \rangle$ has *left residuals* (i.e. there is a binary operation $e \backslash e'$ such that $e \circ d \preceq_{\mathbb{E}} e' \iff d \preceq_{\mathbb{E}} e \backslash e'$), and $\Omega \subseteq \mathbb{F}$ is a lower-closed subset of $\mathbb{F}$; see [96] for the definition. Every ω-regular winning condition (for example, parity) corresponds to a winning condition $\langle \mathbb{E}, \mathbb{F}, \Omega \rangle$, and vice versa.

In [96] a notion of two-player game over Böhm trees, called *type-checking game*, is introduced. Let U be a Böhm tree, θ be an intersection type, and Γ be a finite partial map from variables to intersection types such that $dom(\Gamma)$ contains all free variables of U; we write $\Gamma \models U : \theta$ to mean "Verifier has a winning strategy for the game that checks if U has type θ under assumption Γ". Type-checking games generalise the standard property-checking games (e.g. parity games) which are played over trees, to games which are played over *trees with binders*.

Properties of the Type-Checking Game: We summarise the key properties [96] as follows.

(1) The relation $\models$ *conservatively extends* the MSO properties of Σ-labelled trees generated by recursion schemes [73], or equivalently by ground-type $\lambda\mathbf{Y}$-terms with free variables of order at most one.

(2) *Two-Level Compositionality* [75]: If Böhm trees U and V are composable, then the set of properties (i.e. intersection types) of $U \circ V$ is completely determined by those of U and of V. Furthermore if $\Gamma \models U : \theta$ and $\Gamma \models V : \theta'$ imply $\Gamma \models U \circ V : \theta''$, then the winning strategies $\mathfrak{s}_U^{\theta}$ of $\Gamma \models U : \theta$ and $\mathfrak{s}_V^{\theta'}$ of $\Gamma \models V : \theta'$ are composable, and yield a winning strategy $\mathfrak{s}_U^{\theta} \circ \mathfrak{s}_V^{\theta'}$ of $\Gamma \models U \circ V : \theta''$.

(3) *Effective Selection*: If $\Gamma \models \mathrm{BT}(M) : \theta$ then there exists, constructively, a $\lambda\mathbf{Y}$-definable winning strategy of $\Gamma \models \mathrm{BT}(M) : \theta$.

(4) *Transfer Theorem*: A proof system for typing judgements of the form, $\Gamma \vdash M : \theta$, is introduced with proof rules given in Figure 5, satisfying:

$$\Gamma \vdash M : \theta \iff \Gamma \models \mathrm{BT}(M) : \theta.$$

Notation. In Figure 5, $\Gamma' \preceq \Gamma$ is defined by pointwise extension of a (decidable) subtyping relation $\theta \preceq \theta'$, and $e \circledast \Gamma$ is defined in terms of the semigroup action $\circledast : \mathbb{E} \times \mathbb{F} \to \mathbb{F}$ of the ω-monoid $\langle \mathbb{E}, \mathbb{F} \rangle$.

(5) *Decidability of Type Checking $\lambda\mathbf{Y}$-definable Böhm Trees*: It is decidable, given an arbitrary $\lambda\mathbf{Y}$-term M, intersection type θ, and assumption Γ such that $dom(\Gamma)$ contains the free variables of M, whether $\Gamma \models \mathrm{BT}(M) : \theta$ holds. Thanks to the Transfer Theorem, the decidability of $\Gamma \models M : \theta$ follows from the decidability of $\Gamma \vdash M : \theta$.

Example 33. With reference to the Böhm tree, $\mathrm{BT}(M)$, of Example 32, consider the tree property φ = "*in every branch, there are only finitely many occurrences of bound variables*". Let U be a Böhm tree of type $(o \to o) \to o$ (for example, $\mathrm{BT}(M)$). Take the parity winning condition [96] consisting of effects ordered as $2 \prec 0 \prec 1$, and a single base type q. Then U satisfies φ if, and only if, $\Gamma \vDash U : (((q,1) \to q, 1) \to q, 0)$ where $\Gamma = a : (q,1), b : ((q,0) \to (((q,1) \to q,1) \to q,0) \to q,1)$. It is decidable whether a given U satisfies φ.

$$\frac{\theta = \theta_i \;\;\&\;\; \epsilon \preceq_{\mathbb{E}} e_i \quad \text{for some } i \text{ where } \Gamma(x) = \bigwedge_{i \in I}(\theta_i, e_i)}{\Gamma \vdash x : \theta}$$

$$\frac{\Gamma \vdash M : \tau \to \theta \quad \Gamma \vdash N : \tau}{\Gamma \vdash M\,N : \theta} \qquad \frac{\Gamma, x : \tau \vdash M : \theta}{\Gamma \vdash \lambda x.M : \tau \to \theta}$$

$$\frac{\Gamma' \preceq \Gamma \quad \Gamma \vdash M : \theta \quad \theta \preceq \theta'}{\Gamma' \vdash M : \theta'} \qquad \frac{\vDash \mathrm{BT}(\mathbf{Y}) : \theta}{\Gamma \vdash \mathbf{Y} : \theta}$$

$$\frac{\forall i \in I.\ \Gamma \vdash M : (\theta_i, e_i)}{\Gamma \vdash M : \bigwedge_{i \in I}(\theta_i, e_i)} \qquad \frac{\Gamma \vdash M : \theta}{e \circledast \Gamma \vdash M : (\theta, e)}$$

Fig. 5. Typing Rules for Type Checking Game

Cartesian Closed Category of Effect Arena Games: Given a winning condition $\langle \mathbb{E}, \mathbb{F}, \Omega \rangle$ and a finite set Q of ground types, there is a cartesian closed category $\mathscr{G}_{(\mathbb{E},\mathbb{F},\Omega)}$ of *effect arena games*. (In the proof that the category $\mathscr{G}_{(\mathbb{E},\mathbb{F},\Omega)}$ is well-defined, the main technical lemmas are concerned with the preservation of the winning condition by strategy composition.) A strategy-aware *two-level game model* that can interpret intersection types with effect annotations is then constructed from $\mathscr{G}_{(\mathbb{E},\mathbb{F},\Omega)}$. The constructions are a straightforward adaptation of the two-level constructions in [75].

The above list of properties of the type checking game are proved semantically, by appealing to the two-level game model [96].

On Transfer Theorem: In higher-order model checking, a *transfer theorem* is a theorem stating that given a ranked alphabet Σ, there is an effective transformation of $\textsc{Form}(\Sigma)$ (a set of formulas over the vocabulary Σ) to $\textsc{Form}(\widehat{\Sigma})$, $\varphi \mapsto \widehat{\varphi}$, such that for every closed ground-type $\lambda\mathbf{Y}$-term M over Σ, $\mathrm{BT}(M)$ satisfies φ if, and only if, M satisfies $\widehat{\varphi}$.

Salvati and Walukiewicz's transfer theorem in [85] is actually stronger than the preceding statement, as the formula $\widehat{\varphi}$ is constructed for a certain infinite family of terms M. The crux of their work lies in the MSO definability of the set of (closed) $\lambda\mathbf{Y}$-terms, $\{M \in \mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X}) \mid \mathrm{BT}(M) \vDash \varphi\}$, for a given MSO formula φ, ranked alphabet Σ, a finite set $\mathcal{T}$ of simple types, and a finite set $\mathcal{X}$ of variables, where $\mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ is the set of closed $\lambda\mathbf{Y}$-terms M over the signature Σ such that all (bound) variables in M are from $\mathcal{X}$, and every subterm of M has a type in $\mathcal{T}$. There are earlier versions of transfer theorem in [73] (Theorem 16, as encapsulated in the Transference Principle), and in [57] (see Theorem 18).

B. Model Checking by Evaluating Effective Semantics

In recent work, Salvati and Walukiewicz have advocated an approach to higher-order model checking via *effective* denotational semantics of the $\lambda\mathbf{Y}$-calculus. The first example of such a (finite) semantics was given by Aehlig [2]; he used the semantics to prove the decidability of higher-order model checking with respect to *trivial tree automata*, which are parity tree automata that have an even number as the only priority (thus the acceptance condition is trivial). In [86], Salvati and Walukiewicz studied the model checking problem with respect to *insightful* trivial automata i.e. trivial automata endowed with the ability to detect if a term has a head normal form. (Aehlig's [2] and Kobayashi's [58] model checking algorithms are for non-insightful trivial automata.) They show that extremal (i.e. least or, dually, greatest) fixpoints in finitary models of $\lambda\mathbf{Y}$-calculus, which are built up from finite lattices using monotone function spaces, capture precisely boolean combinations of properties expressible in non-insightful trivial automata. Thus to construct models for insightful trivial automata, it is necessary to consider non-extremal fixpoints.

Building on [86], Salvati and Walukiewicz [87] have given a type system for model checking the Böhm trees of ground-type $\lambda\mathbf{Y}$-terms with respect to formulas of weak MSOL (i.e. MSOL with quantifications restricted to range over finite sets) or, equivalently, weak alternating tree automata. The denotational semantics underpinning the type system is obtained from the finitary semantics of [86] by stratification according to the ranks of the input weak alternating automaton, so that the denotation of a term in stratum k captures the behaviour of the term with respect to the automaton restricted to states of rank at most k. The denotation of a fixpoint is then defined by induction on the stratum levels: greatest fixpoint computation on even levels, and least fixpoint computation on odd levels. The (intersection) type system is related to the model in a manner reminiscent of Abramsky's domain theory in logical form [1].

C. Linear Logic Models of Kobayashi-Ong Type System

Linear logic offers a powerful organisational principle for semantics of (higher-order) computation. In recent work, Grellois and Melliès have developed models of the type system of Kobayashi and Ong [57] by adapting models of linear logic [41]. In [42], they construct an infinitary variant of the relational model of linear logic, based on a novel interpretation of the exponential modality as the set of countable multisets. The relational semantics is then extended with a notion of priority given by an alternating parity tree automaton (APT). In this extended model, a fixpoint operator is constructed using a mixture of inductive and coinductive interpretations as regulated by the priorities. The fixpoint operator satisfies some of the fundamental equational properties of the fixpoint operators in domain theory, such as (parameterised) dinaturality and diagonality. In [43] they use results from [57] to show that the denotation of a recursion scheme in this model contains the initial state of a given APT if, and only if, the tree it generates is accepted by the APT.

V. Applications

In this section, we briefly discuss the application of higher-order model checking to the verification of higher-order functional programs.

Higher-order recursion schemes, or equivalently the $\lambda\mathbf{Y}$-calculus, are an appealing abstract model for model checking higher-order programs. As we have seen in Section II, not only do they have rich and decidable logical theories, they accurately model higher-order control flow [45] and are highly expressive. Indeed, in a precise sense, recursion schemes are the higher-order analogue of Boolean programs, which have played a successful rôle in the model checking of first-order, imperative programs [6]. In an influential paper [59], Kobayashi introduced a method for the verification of safety properties of functional programs by reduction to the model checking of recursion schemes with respect to trivial automata. Techniques such as predicate abstraction [66] and CEGAR [74] have been incorporated into the higher-order model checking approach, enabling the safety verification of higher-order programs that use infinite data domains and pattern-matching algebraic data types.

In [58], Kobayashi presented a "practical" method for model checking recursion schemes with respect to trivial automata using an efficient type inference algorithm. A tool implementation of the algorithm, called TRecS, performs remarkably well on a range of small but tricky examples, despite the hyper-exponential worst-case complexity. There has been active research in the development of higher-order model checking algorithms in recent years. Kobayashi [60], and Neatherway, Ramsay and Ong [71] have introduced algorithms, called GTRecS and HORSC respectively, which are inspired by or based on game semantics [73], [49]; Broadbent et al. [13] have developed an algorithm C-SHORe that extends the saturation method on pushdown graphs to a backward reachability analysis of CPDA graphs. In [14], a hybrid algorithm HorSat is proposed that (like TRecS) produces intersection type certificates by a direct analyses of recursion schemes, but (like C-SHORe) propagates information backwards by computing pre-images via saturation, starting from target configurations. However none of these algorithms can scale beyond recursion schemes of several hundred rules. A breakthrough [83] was achieved by a type inference method that employs a type-directed form of abstraction refinement, and reasons simultaneously about acceptance by the property automaton and acceptance by its dual. A prototype implementation of the algorithm, called Preface, scales readily to recursion schemes of several thousand rules. Recently two modifications were made to HorSat in [94] which significantly improved its performance: linear-time collection of flow information using a sub-transitive flow graph, and the use of zero-suppressed binary decision diagrams for representing type information.

The framework of higher-order recursion schemes works well for the model checking of simply-typed functional programs. In order to extend the analysis to programs of more advanced type systems, Tsukada and Kobayashi [95] have introduced an untyped version of recursion schemes and an intersection type system that is equivalent to the model checking of untyped recursion scheme (which is undecidable in general). In a similar vein, motivated by the model checking of a broader range of programs including object-oriented programs and multithreaded programs, Kobayashi and Igarashi [62] have introduced recursively-typed recursion schemes and a model checking algorithm that is relatively complete with respect to a recursive intersection type system.

VI. Further Directions

We conclude by discussing a number of open problems and further directions.

a) Equivalence of Recursion Schemes: The Equivalence of Recursion Schemes problem asks whether two given recursion schemes generate the same tree. Perhaps the best known, and probably the most challenging, open problem in higher-order model checking is whether this problem is decidable. This problem is recursively equivalent to the *Böhm Tree Equivalence of $\lambda\mathbf{Y}$-Calculus* problem, which asks whether the Böhm trees of two given $\lambda\mathbf{Y}$-terms are equal [19].

b) The Nondeterministic Safety Conjecture: There is a nondeterministic version of the Safety Conjecture, which states that there is a word language recognised by a nondeterministic order-n collapsible pushdown automaton, which is not recognised by any nondeterministic higher-order pushdown automaton. Note that this problem is independent of Parys' result (Theorem 31). For $n = 2$, the conjecture is actually known to be false; however it is unlikely that the simulation argument in [3] can be adapted to higher orders. The problem is open for $n \geq 3$.

c) Context Sensitivity of Unsafe Word Languages: Are word languages generated by order-n collapsible pushdown automata (equivalently, recursion schemes) context-sensitive? As mentioned in the preceding, the answer is yes for $n \leq 3$ [63].

d) Computing Downward Closures of the Maslov Languages: A famous result of Higman [48] states that the set of finite words over a finite alphabet, partially ordered by the subword ordering, is a well-quasi ordering. A corollary is that the downward closure (with respect to the subword ordering) of an arbitrary word language is regular. However it is in general impossible to compute (representations of) these closures. Zetzsche [98] has recently shown that indexed languages have computable downward closures. It is natural to ask whether his result extends to all orders of the Maslov Hierarchy, and to those of the Unsafe Maslov Hierarchy.

e) Extensions of Higher-Order Model Checking: Another challenge is to find useful extensions of the decidability of higher-order model checking, in the sense of Theorem 16. Several directions are possible. The first is to extend the models. Is there a larger finitely-presentable class of Σ-labelled trees, or of graphs, that have decidable MSO theories? Higher-type Böhm trees ([96], as discussed in Section IV-A) are an example of such a class. Larger tree classes (for example, [95] and [62]) than those generated by higher-order recursion schemes have been studied in the literature; though well-motivated from a program verification perspective, they do not have a decidable MSO theory. The second direction is to extend the correctness properties. An intriguing question is whether Theorem 16 extends to *non-ω-regular* properties. For pushdown games, there are decidable winning conditions which induce non-ω-regular sets of winning positions [10], even those of arbitrary Borel complexity [90]. Can these results be extended to PDA or CPDA graphs of higher orders? In a related direction, solving *pushdown ωB-games* and pushdown games with *finitary parity* and *stack boundedness conditions* is decidable (and EXPTIME-complete) [18]. It would be interesting to extend these results to games of higher orders.

Acknowledgements: The author is grateful to Takeshi Tsukada for helpful comments on the paper.

References

[1] S. Abramsky, "Domain theory in logical form," *Annals of Pure and Applied Logic*, vol. 51, no. 1-2, pp. 1–77, Mar. 1991.

[2] K. Aehlig, "A finite semantics of simply-typed lambda terms for infinite runs of automata," *LMCS*, vol. 3, pp. 1–23, 2007.

[3] K. Aehlig, J. G. de Miranda, and C.-H. L. Ong, "Safety Is Not a Restriction at Level 2 for String Languages," in *FoSSaCS*, 2005, pp. 490–504.

[4] ——, "The monadic second order theory of trees given by arbitrary level-two recursion schemes is decidable," in *TLCA*, 2005, pp. 39–54.

[5] A. Aho, "Indexed grammars - an extension of context-free grammars," *J. ACM*, vol. 15, pp. 647–671, 1968.

[6] T. Ball and S. Rajamani, "Bebop: A Symbolic Model Checker for Boolean Programs," in *SPIN*, 2000, pp. 113 – 130.

[7] W. Blum, "The safe lambda calculus," Ph.D. dissertation, University of Oxford, 2008.

[8] W. Blum and C. Broadbent, "The CPDA-transform of a safe recursion scheme does not collapse," 2009, in preparation.

[9] W. Blum and C.-H. L. Ong, "The safe lambda calculus," *LMCS*, vol. 5, no. 1, 2009.

[10] A.-J. Bouquet, O. Serre, and I. Walukiewicz, "Pushdown Games with Unboundedness and Regular Conditions," in *FSTTCS*, 2003, pp. 88–99.

[11] C. H. Broadbent, "On First-Order Logic and CPDA Graphs," *Theory of Computing Systems*, vol. 55, pp. 771–832, 2014.

[12] C. H. Broadbent, A. Carayol, C.-H. L. Ong, and O. Serre, "Recursion schemes and logical reflection," in *LICS*, 2010, pp. 120–129.

[13] C. H. Broadbent, M. Hague, O. Serre, A. Carayol, M. Hague, and O. Serre, "C-SHORe: A Collapsible Approach to Verifying Higher-Order Programs," in *ICFP*, 2013, pp. 13–24.

[14] C. H. Broadbent and N. Kobayashi, "Saturation-Based Model Checking of Higher-Order Recursion Schemes," in *CSL*, 2013, pp. 129–148.

[15] A. Carayol and O. Serre, "Collapsible Pushdown Automata and Labeled Recursion Schemes: Equivalence, Safety and Effective Selection," in *LICS*, 2012, pp. 165–174.

[16] D. Caucal, "On infinite transition graphs having a decidable monadic theory," in *ICALP*, 1996, pp. 194–205.

[17] ——, "On infinite terms having a decidable monadic theory," in *MFCS*, 2002, pp. 165–176.

[18] K. Chatterjee and N. Fijalkow, "Infinite-state games with finitary conditions," in *CSL*, 2013, pp. 1–26.

[19] P. Clairambault and A. S. Murawski, "Böhm Trees as Higher-Order Recursive Schemes," in *FSTTCS*, 2013, pp. 91–102.

[20] B. Courcelle, "Fundamental properties of infinite trees," *TCS*, vol. 25, pp. 95–169, 1983.

[21] ——, "Recursive applicative program schemes," in *Handbook of Theoretical Computer Science, Volume B*. MIT Press, 1990, pp. 459–492.

[22] ——, "The monadic second-order logic of graphs IX: machines and their behaviours," *TCS*, vol. 151, pp. 125–162, 1995.

[23] B. Courcelle and M. Nivat, "The algebraic semantics of recursive program schemes," in *MFCS*, 1978, pp. 16–30.

[24] B. Courcelle and J. Vuillemin, "Completeness results for the equivalence of recursive schemes," *Journal of Computing and System Science*, vol. 12, pp. 179–197, 1976.

[25] P.-L. Curien and H. Herbelin, "Computing with abstract Böhm trees," in *Fuji International Symposium on Functional and Logic Programming*. World Scientific, 1998, pp. 20–39.

[26] ——, "Abstract machines for dialogue games," 2007, coRR abs/0706.2544.

[27] W. Damm, "Higher type program schemes and their tree languages," *Theoretical Computer Science*, pp. 51–72, 1977.

[28] ——, "The IO- and OI-hierarchy," *TCS*, vol. 20, pp. 95–207, 1982.

[29] W. Damm, E. Fehr, and K. Indermark, "Higher type recursion and self-application as control structures," in *Formal Descriptions of Programming Concepts*, E. Neuhold, Ed. North-Holland, Amsterdam, 1978, pp. 461–187.

[30] W. Damm and A. Goerdt, “An automata-theoretical characterization of the OI-hierarchy,” *Information and Control*, vol. 71, pp. 1–32, 1986.

[31] V. Danos, H. Herbelin, and L. Regnier, “Game semantics and abstract machines,” in *LICS*, 1996, pp. 394–405.

[32] J. W. de Bakker and W. P. de Roever, “A calculus for recursive program schemes,” in *ICALP*, 1972, pp. 167–196.

[33] J. de Miranda, “Structures generated by higher-order grammars and the safety constraint,” Ph.D. dissertation, University of Oxford, 2006.

[34] J. Duske and R. Parchmann, “Linear indexed languages,” *TCS*, vol. 32, pp. 47–60, 1984.

[35] C. C. Elgot, “Algebraic theories and program schemes,” in *Symposium on Semantics of Algorithmic Languages*, 1971, pp. 71–88.

[36] E. A. Emerson and C. S. Jutla, “Tree automata, mu-calculus and determinacy,” in *FOCS*, 1991, pp. 368–377.

[37] J. Engelfriet, “Interated stack automata and complexity classes,” *Information and Computation*, vol. 95, pp. 21–75, 1991.

[38] J. Engelfriet and E. M. Schmidt, “IO and OI. 1,” *Journal of Computer and System Sciences*, vol. 3, pp. 328–353, 1977.

[39] ——, “IO and 01. II,” *Journal of Computer and System Sciences*, vol. 16, pp. 67–99, 1978.

[40] R. H. Gilman, “A shrinking lemma for indexed languages,” *TCS*, vol. 163, pp. 277–281, 1996.

[41] J.-Y. Girard, “Linear Logic,” *Theoretical Computer Science*, vol. 50, pp. 1–102, 1987.

[42] C. Grellois and P.-A. Melliès, “An infinitary model of linear logic,” in *FoSSaCS*, vol. 9034, 2015, pp. 41–55.

[43] ——, “Tensorial logic with colours and higher-order model checking,” Tech. Rep., 2015.

[44] I. Guessarian, *Algebraic Semantics*. Springer, 1981.

[45] M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre, “Collapsible Pushdown Automata and Recursion Schemes,” in *LICS*, 2008, pp. 452–461.

[46] M. Hague and A. W. To, “The Complexity of Model Checking (Collapsible) Higher-Order Pushdown Systems,” in *FSTTCS*, 2010, pp. 228–239.

[47] T. Hayashi, “On derivation trees of indexed grammars: An extension of the uvwxy-theorem,” *Publ. RIMS Kyoto Univ.*, vol. 9, pp. 61–92, 1983.

[48] G. Higman, “Ordering by Divisibility in Abstract Algebras,” *Proceedings of the London Mathematical Society*, vol. 2, pp. 326–336, 1952.

[49] J. M. E. Hyland and C.-H. L. Ong, “On Full Abstraction for PCF: I. Models, observables and the full abstraction problem, II. Dialogue games and innocent strategies, III. A fully abstract and universal game model,” *Information and Computation*, vol. 163, pp. 285–408, 2000.

[50] Y. I. Ianov, “The logical schemes of algorithms,” *Problems of Cybernetics*, vol. 1, pp. 82–140, 1960.

[51] K. Inaba and S. Maneth, “The complexity of tree transducer output languages,” in *FSTTCS*, 2008, pp. 244–255.

[52] A. Kartzow, “First-order logic on higher-order nested pushdown trees,” *ACM Trans. Comput. Logic*, vol. 14, no. 2, pp. 1–42, 2013.

[53] A. Kartzow and P. Parys, “Strictness of the collapsible pushdown hierarchy,” in *MFCS*, 2012, pp. 566–577.

[54] T. Knapik, D. Niwinski, and P. Urzyczyn, “Higher-order pushdown trees are easy,” in *FoSSaCS*, 2002, pp. 205–222.

[55] T. Knapik, D. Niwinski, P. Urzyczyn, and I. Walukiewicz, “Unsafe grammars, panic automata, and decidability,” in *ICALP*, 2005, pp. 1–26.

[56] T. Knapik, D. Niwiski, and P. Urzyczyn, “Deciding monadic theories of hyperalgebraic trees,” *TLCA*, pp. 253–267, 2001.

[57] N. Kobayashi and C.-H. L. Ong, “A type theory equivalent to the modal mu-calculus model checking of higher-order recursion schemes,” in *LICS*, 2009, pp. 179–188.

[58] N. Kobayashi, “Model-checking higher-order functions,” in *PPDP*, 2009, pp. 25–36.

[59] ——, “Types and higher-order recursion schemes for verification of higher-order programs,” in *POPL*, Jan. 2009, p. 416.

[60] ——, “A Practical Linear Time Algorithm for Trivial Automata Model Checking of Higher-Order,” in *FoSSaCS*, 2011, p. 25.

[61] ——, “Pumping by typing,” in *LICS*, 2013, pp. 398–407.

[62] N. Kobayashi and A. Igarashi, “Model-Checking Higher-Order Programs with Recursive Types,” in *ESOP*, 2013, pp. 431–450.

[63] N. Kobayashi, K. Inaba, and T. Tsukada, “Unsafe order-2 tree languages are context-sensitive,” in *FoSSaCS*, 2014, pp. 149–163.

[64] N. Kobayashi and C.-H. L. Ong, “A Type System Equivalent to the Modal Mu-Calculus Model Checking of Higher-Order Recursion Schemes,” in *LICS*, 2009, pp. 179–188.

[65] ——, “Complexity of Model Checking Recursion Schemes for Fragments of the Modal Mu-Calculus,” *LMCS*, vol. 7, 2011.

[66] N. Kobayashi, R. Sato, and H. Unno, “Predicate abstraction and CEGAR for higher-order model checking,” *ACM SIGPLAN Notices*, vol. 46, no. 6, pp. 222–233, Jun. 2011.

[67] D. C. Luckham, D. M. R. Park, and M. S. Paterson, “On Formalised Computer Programs,” *Journal of Computer and System Sciences*, vol. 249, pp. 220–249, 1970.

[68] A. N. Maslov, “The hierarchy of indexed languages of an arbitrary level,” *Soviet Math. Dokl.*, vol. 15, pp. 1170–1174, 1974.

[69] ——, “Multilevel stack automata,” *Problems of Information Transmission*, vol. 12, pp. 38–43, 1976.

[70] D. E. Muller and P. E. Schupp, “The theory of ends, pushdown automata, and second-order logic,” *Theoretical Computer Science*, vol. 37, pp. 51–75, 1985.

[71] R. P. Neatherway, S. J. Ramsay, and C.-H. L. Ong, “A traversal-based algorithm for higher-order model checking,” in *ICFP*, 2012, pp. 353–364.

[72] M. Nivat, “On the interpretation of recursive program schemes,” *Symposia Mathematica*, vol. 15, pp. 255–281, 1975.

[73] C.-H. L. Ong, “On Model Checking Trees Generated by Higher-Order Recursion Schemes,” in *LICS*, 2006, pp. 81–90.

[74] C.-H. L. Ong and S. J. Ramsay, “Verifying higher-order functional programs with pattern-matching algebraic data types,” Tech. Rep., 2011.

[75] C.-H. L. Ong and T. Tsukada, “Two-Level Game Semantics, Intersection Types, and Recursion Schemes,” in *ICALP*, 2012, pp. 325–336.

[76] C.-H. L. Ong and N. Tzevelekos, “Functional Reachability,” in *LICS*, 2009, pp. 286–295.

[77] R. Parchmann, J. Duske, and J. Specht, “On deterministic indexed languages,” *Information and Control*, vol. 45, pp. 48–67, 1980.

[78] D. M. R. Park, “Fixpoint induction and proofs of program properties,” in *Machine Intelligence*, D. Michie and B. Meltzer, Eds., vol. 5, 1970.

[79] P. Parys, “On the Significance of the Collapse Operation,” in *LICS*. IEEE, Jun. 2012, pp. 521–530.

[80] M. Patterson, “Equivalence problems in a model of computation,” Ph.D. dissertation, University of Cambridge, 1967.

[81] D. Perrin and J.-E. Pin, “Semigroups and automata on infinite words,” 1995, pp. 1–28.

[82] M. O. Rabin, “Decidability of second-order theories and automata on infinite trees,” *Trans. Amer. Maths. Soc*, vol. 141, pp. 1–35, 1969.

[83] S. J. Ramsay, R. P. Neatherway, and C.-H. L. Ong, “An Abstraction Refinement Approach to Higher-Order Model Checking,” in *POPL*. New York, USA: ACM Press, 2014, pp. 61–72.

[84] S. Salvati and I. Walukiewicz, “Krivine machines and higher-order schemes,” in *ICALP*, 2011, pp. 162–173.

[85] ——, “Evaluation is MSOL-compatible,” in *FSTTCS*, 2013, pp. 103–114.

[86] ——, “Using Models to Model-Check Recursive Schemes,” in *TLCA*, 2013, pp. 189–204.

[87] ——, “Typing Weak MSOL Properties,” in *FoSSaCS*, vol. 7794, 2015, pp. 129–144.

[88] H. Schwichtenberg, “Definierbare funktionen im lambda-kalkul mit typen,” *Archiv Logik Grundlagen-forsch*, vol. 17, 1976.

[89] D. S. Scott, “Outline of a Mathematical Theory Of Computation,” PRG University of Oxford, Tech. Rep., 1970.

[90] O. Serre, “Parity Games Played on Transition Graphs of One-Counter Processes,” in *FoSSaCS*, 2006, pp. 337–351.

[91] R. Statman, “On the Lambda-Y calculus,” *Annals of Pure and Applied Logic*, vol. 130, no. 1-3 SPEC. ISS., pp. 325–337, 2004.

[92] C. Stirling, “Dependency Tree Automata,” in *FoSSaCS*, 2009, pp. 92–106.

[93] T. Cachat, “Games on pushdown graphs and extensions,” Ph.D. dissertation, RWTH Aachen, 2003.

[94] T. Terao and N. Kobayashi, “A ZDD-Based Efficient Higher-Order Model,” in *APLAS*, 2014, pp. 354–371.

[95] T. Tsukada and N. Kobayashi, “Untyped Recursion Schemes and Infinite Intersection Types,” in *FoSSaCS*, 2010, pp. 343–357.

[96] T. Tsukada and C.-H. L. Ong, “Compositional Higher-order Model Checking via omega-Regular Games over Boehm trees,” in *CSL-LICS*, 2014.

[97] I. Walukiewicz, “Pushdown processes: games and model-checking,” *Information and Computation*, vol. 157, pp. 234–263, 2001.

[98] G. Zetzsche, “An approach to computing downward closures,” in *ICALP*, 2015.