A Type System Equivalent to the Modal Mu-Calculus Model Checking of Higher-Order Recursion Schemes

Naoki Kobayashi
Tohoku University

C.-H. Luke Ong
University of Oxford

***Abstract*—The model checking of higher-order recursion schemes has important applications in the verification of higher-order programs. Ong has previously shown that the modal mu-calculus model checking of trees generated by order-n recursion scheme is n-EXPTIME complete, but his algorithm and its correctness proof were rather complex. We give an alternative, type-based verification method: Given a modal mu-calculus formula, we can construct a type system in which a recursion scheme is typable if, and only if, the (possibly infinite, ranked) tree generated by the scheme satisfies the formula. The model checking problem is thus reduced to a type checking problem. Our type-based approach yields a simple verification algorithm, and its correctness proof (constructed without recourse to game semantics) is comparatively easy to understand. Furthermore, the algorithm is polynomial-time in the size of the recursion scheme, assuming that the formula and the largest order and arity of non-terminals of the recursion scheme are fixed.**

I. Introduction

The model checking of infinite structures generated by higher-order recursion schemes has drawn growing attention from both theoretical and practical communities. From a theoretical perspective, the recent interest was sparked by the discovery of Knapik et al. [11] that higher-order recursion schemes satisfying a syntactic constraint called *safety* generate the same class of (possibly infinite, ranked) trees as higher-order pushdown automata. Remarkably they also showed that these trees have decidable monadic second-order (MSO) theories [12], subsuming earlier well-known MSO decidability results for regular (or order-0) trees [19] and algebraic (or order-1) trees [4]. (MSO logic is a kind of gold standard of expressivity for logics that describe computational properties: all the standard temporal logics can be embedded into it, and it is hard to extend it meaningfully without sacrificing decidability where it holds.) Ong [18] has subsequently shown that the modal mu-calculus model checking problem for trees generated by arbitrary order-n recursion schemes is n-EXPTIME complete (and hence these trees have decidable MSO theories); further [7] these schemes are equi-expressive with a new class of automata, called *collapsible* pushdown automata. On the practical side, Kobayashi [14] has recently shown that the verification of higher-order programs can be reduced to that of higher-order recursion schemes. He constructed a transformation of a higher-order program into a recursion scheme that generates a (possibly infinite) tree representing all the possible event sequences of the program; thus, temporal properties of the program can be verified by model-checking the recursion scheme.

Ong's algorithm for verifying higher-order recursion schemes is rather complex and probably hard to understand: The algorithm reduces the model-checking problem to a parity game over *variable profiles*, and its correctness proof relies on game semantics [9]. Hague et al. [7] gave an alternative proof via a reduction of the model checking of recursion schemes to that of collapsible pushdown automata; their reduction is also based on game semantics. Kobayashi [14] showed that given a Büchi tree automaton with a trivial acceptance condition (a class which Aehlig [1] has called *trivial automata*), one can construct an intersection type system in which a recursion scheme is typable if, and only if, the tree generated by the scheme is accepted by the automaton. (Prior to Kobayashi's work [14], Aehlig [1] has also proposed a verification method for the same class of trivial automata. Kobayashi's type system is closely related to Aehlig's, which was not presented in the form of a type system.) The advantages of the type system are that the correctness of the algorithm is much simpler, and it is easier to optimize the algorithm in a number of special cases, by standard methods for type inference. Specifically, Kobayashi [14] has shown that, assuming that the automaton and the largest order and arity of non-terminals of the recursion scheme are fixed, the verification algorithm runs in time linear in the size of the recursion scheme.

This paper builds on Kobayashi's type system [14] and extends it to a type system capable of the modal mu-calculus model checking of trees generated by higher-order recursion schemes. Equivalently (thanks to Emerson and Jutla [5]), given an alternating parity tree automaton $\mathcal{A}$, one can construct a type system $\mathcal{T}_{\mathcal{A}}$ in which a recursion scheme $\mathcal{G}$ is well-typed if, and only if, the tree generated by $\mathcal{G}$ is accepted by $\mathcal{A}$. Thus, the modal mu-calculus model checking problem is reduced to a type inference problem.

Our type-based verification algorithm has a number of advantages:

- The algorithm is simple: the type system, to which the model checking problem is reduced, is defined by induction over four rules. The correctness proof is, arguably, considerably easier to understand than that of Ong's original approach [18]. The correctness of the algorithm has two parts: the correctness of the type system, and that of the type inference algorithm. For both parts, standard methods (such as proving type soundness via type preservation) remain applicable, although the reasoning about parity conditions is novel and non-trivial. It is also worth noting that this is the first proof of Ong's result without recourse to game semantics.

- It is much easier to discuss the parameterized complexity and possible optimization of the verification algorithm. In fact, our type-based verification algorithm runs in time polynomial in the size of the recursion scheme, assuming that the automaton and the largest order and arity of non-terminals of the recursion scheme are fixed. In contrast, Ong's algorithm [18] runs in time n-EXPTIME in the size of the scheme, under the same assumption.
- Framed as a type system, we believe that it is easy to modify the verification algorithm to deal with various extensions of higher-order recursion schemes. For example, one can extend higher-order recursion schemes with a limited form of polymorphism that admits (say) a non-terminal of kind $(\mathsf{o} \to \mathsf{o}) \wedge ((\mathsf{o} \to \mathsf{o}) \to (\mathsf{o} \to \mathsf{o}))$ where o describes trees, and also with finite data domains such as booleans: see Section VII.

From a type-theoretic point of view, the type system has a number of novel features which we think are interesting: (i) variable bindings in a type environment have flags and priorities to express *when* the variables can be used, and (ii) the well-typedness of recursive definitions is defined via the winning condition of a parity game. The latter is a non-trivial generalization of the usual treatment of recursion in type systems for programming languages.

The rest of this paper is organized as follows. Section II gives preliminary definitions. Section III defines the type system equivalent to the model checking of recursion schemes, and Section IV proves its correctness. Section V discusses the type inference algorithm (which serves as a model-checking algorithm for recursion schemes) and its complexity. Section VI discusses related work and Section VII discusses future directions. A longer version of this paper is available from the authors' web page.

II. Preliminaries

This section reviews definitions of higher-order recursion schemes, alternating parity tree automata, and parity games [6]. Alternating parity tree automata are used for expressing properties of infinite trees and are equi-expressive with logics such as MSO and modal μ-calculus. Parity games will be used for defining our type system (more specifically, for the purpose of typing recursive definitions).

Higher-Order Recursion Schemes: A higher-order recursion scheme is a grammar for describing an infinite tree. A *kind*[1] is either o, describing a tree, or $\kappa_1 \to \kappa_2$, describing a function that takes an entity of kind κ_1 and returns an entity of kind κ_2. The *order* and *arity* of κ, written $ord(\kappa)$ and $arity(\kappa)$ respectively, are defined by:

$$\begin{array}{ll} ord(\mathsf{o}) := 0 & ord(\kappa_1 \to \kappa_2) := max(ord(\kappa_1)+1, ord(\kappa_2)) \\ arity(\mathsf{o}) := 0 & arity(\kappa_1 \to \kappa_2) := arity(\kappa_2)+1 \end{array}$$

A (deterministic) *higher-order recursion scheme* (or *recursion scheme*, for short) $\mathcal{G}$ is a quadruple $(\Sigma, \mathcal{N}, \mathcal{R}, S)$, where

- Σ is a *ranked alphabet* i.e. a map from a finite set of symbols called *terminals* to kinds of order 0 or 1.
- $\mathcal{N}$ is a map from a finite set of symbols called *non-terminals* to kinds.
- $\mathcal{R}$ is a map from the set of non-terminals (i.e. $dom(\mathcal{N})$) to terms of the form $\lambda\widetilde{x}.t$. Here, $\widetilde{x}$ abbreviates a sequence of variables, and t is a term constructed from non-terminals, terminals, and variables (see below).
- S is a special non-terminal called the *start symbol*.

We require that $\mathcal{N}(S) = \mathsf{o}$. The set of (typed) terms is defined in the standard manner: A symbol (i.e., a terminal, non-terminal, or variable) of kind κ is a term of kind κ. If terms t_1 and t_2 have kinds $\kappa_1 \to \kappa_2$ and κ_1 respectively, then $t_1\ t_2$ is a term of kind κ_2. For each $\mathcal{R}(F) = \lambda\widetilde{x}.t$, $F\ \widetilde{x}$ and t must be terms of kind o,[2] and the variables that occur in t are contained in $\widetilde{x}$. The *order* of a recursion scheme is the highest order of its non-terminals.

By abuse of notation, we often write $a \in \Sigma$ and $F \in \mathcal{N}$ for $a \in dom(\Sigma)$ and $F \in dom(\mathcal{N})$.

The rewriting relation $\longrightarrow_{\mathcal{G}}$ is defined inductively by:

- $F\ \widetilde{s} \longrightarrow_{\mathcal{G}} [\widetilde{s}/\widetilde{x}]t$ if $\mathcal{R}(F) = \lambda\widetilde{x}.t$.
- If $t \longrightarrow_{\mathcal{G}} t'$, then $ts \longrightarrow_{\mathcal{G}} t's$ and $st \longrightarrow_{\mathcal{G}} st'$.

We omit the subscript $\mathcal{G}$ whenever it is clear from the context.

Let Δ be a set of symbols. A Δ-*labelled tree* is just a partial function t from $\{1,\ldots,n\}^*$ (for some fixed $n \geq 1$) to Δ such that $dom(t)$ is prefix-closed. Note that t is *unranked* i.e. nodes in t that have the same label are not required to have the same number of children. When considering the possibly infinite term-trees that are generated by recursion schemes, we assume a given *ranked* alphabet Σ (say). Let n be the largest arity of symbols in Σ; a Σ-*labelled tree* is thus a partial function t from $\{1,\ldots,n\}^*$ to $dom(\Sigma)$ such that $dom(t)$ is prefix-closed. Further, t is said to be *ranked* just if whenever $t(w) = a$ and $arity(\Sigma(a)) = m$, then $\{i \mid wi \in dom(t)\} = \{1,\ldots,m\}$. A (possibly infinite) sequence π over $\{1,\ldots,n\}$ is a *path* of t if every finite prefix of π is in $dom(t)$.

[1]They are usually called *types* [18]. We use the term "kinds" to avoid confusion with the intersection types introduced later.

[2]By the definition of terms, t does not contain λ-abstractions. We think however that the type system presented in Section III is correct even if λ-abstractions are allowed in t.

Given a term t, we define a (finite) tree $t^{\perp}$ by:

$$t^{\perp} = \begin{cases} f & \text{if } t \text{ is a terminal } f \\ {t_1}^{\perp}{t_2}^{\perp} & \text{if } t \text{ is of the form } t_1 t_2 \text{ and } {t_1}^{\perp} \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

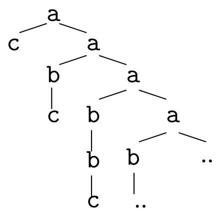
For example, $(f\ (F\ a)\ b)^{\perp} = f\ \perp\ b$. Let $\sqsubseteq$ be the partial order on $dom(\Sigma) \cup \{\perp\}$ defined by $\forall a \in dom(\Sigma).\perp \sqsubseteq a$. It is extended to a partial order on trees by: $t \sqsubseteq s$ iff $\forall w \in dom(t).(w \in dom(s) \land t(w) \sqsubseteq s(w))$. 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 $\sqsubseteq$.

The tree generated by $\mathcal{G}$, or the *value tree* of $\mathcal{G}$, written $[\![\mathcal{G}]\!]$, is $\bigsqcup\{t^{\perp} \mid S \longrightarrow^{*}_{\mathcal{G}} t\}$. By construction, $[\![\mathcal{G}]\!]$ is a possibly infinite, ranked $(\Sigma \cup \{\perp\})$-labelled tree (but see Remark 2.1).

Example 2.1: Consider the recursion scheme $\mathcal{G}_0 = (\Sigma, \mathcal{N}, \mathcal{R}, S)$, where: $\Sigma = \{\mathtt{a} : \mathtt{o} \to \mathtt{o} \to \mathtt{o}, \mathtt{b} : \mathtt{o} \to \mathtt{o}, \mathtt{c} : \mathtt{o}\}$, $\mathcal{N} = \{S : \mathtt{o}, F : \mathtt{o} \to \mathtt{o}\}$, and $\mathcal{R} = \{S \mapsto F\ \mathtt{c}, F \mapsto \lambda x.\mathtt{a}\ x\ (F(\mathtt{b}\,x))\}$. S is reduced as follows.

$$\begin{aligned} &S \longrightarrow F\ \mathtt{c} \longrightarrow \mathtt{a}\ \mathtt{c}\ (F(\mathtt{b}\,\mathtt{c})) \\ &\longrightarrow \mathtt{a}\ \mathtt{c}\ (\mathtt{a}\ (\mathtt{b}\,\mathtt{c})\ (F(\mathtt{b}(\mathtt{b}\,\mathtt{c})))) \longrightarrow \ldots \end{aligned}$$

The value tree $[\![\mathcal{G}_0]\!]$ is depicted as follows.



Alternating Parity Tree Automata: Given a finite set X, the set $\mathsf{B}^{+}(X)$ of *positive Boolean formulas* over X is defined as follows:

$$\mathsf{B}^{+}(X) \ni \theta \ ::= \ \mathsf{t} \mid \mathsf{f} \mid x \mid \theta \land \theta \mid \theta \lor \theta$$

where x ranges over X. We say that a subset Y of X *satisfies* θ just if assigning true to elements in Y and false to elements in $X \setminus Y$ makes θ true.

An *alternating parity tree automaton* (or APT for short) over Σ-labelled trees is a tuple $\mathcal{A} = (\Sigma, Q, \delta, q_I, \Omega)$ where

- Σ is a ranked alphabet; let m be the largest arity of the terminal symbols.
- Q is a finite set of states, and $q_I \in Q$ is the initial state.
- $\delta : Q \times \Sigma \longrightarrow \mathsf{B}^{+}(\{1, \ldots, m\} \times Q)$ is the transition function where, for each $f \in \Sigma$ and $q \in Q$, we have $\delta(q, f) \in \mathsf{B}^{+}(\{1, \ldots, \mathit{arity}(f)\} \times Q)$.
- $\Omega : Q \longrightarrow \{0, \cdots, M-1\}$ is the priority function.

A *run-tree* of an alternating parity tree automaton $\mathcal{A}$ over a Σ-labelled ranked tree t is a $(dom(t) \times Q)$-labelled unranked tree r satisfying:

- $\epsilon \in dom(r)$ and $r(\epsilon) = (\epsilon, q_I)$; and
- for every $\beta \in dom(r)$ with $r(\beta) = (\alpha, q)$, there is a set S that satisfies $\delta(q, t(\alpha))$; and for each $(i, q') \in S$, there is some j such that $\beta\, j \in dom(r)$ and $r(\beta\, j) = (\alpha\, i, q')$.

Let $\pi = \pi_1\, \pi_2 \cdots$ be an infinite path in r; for each $i \geq 0$, let the state label of the node $\pi_1 \cdots \pi_i$ be q_{n_i} where q_{n_0}, the state label of ϵ, is q_I. We say that π satisfies the *parity* condition just if the largest priority that occurs infinitely often in $\Omega(q_{n_0})\, \Omega(q_{n_1})\, \Omega(q_{n_2}) \cdots$ is even. A run-tree r is *accepting* if every infinite path in it satisfies the parity condition.

Ong [18] showed that there is a procedure that, given a recursion scheme $\mathcal{G}$ and an alternating parity tree automaton $\mathcal{A}$, decides whether $\mathcal{A}$ accepts the value tree of $\mathcal{G}$.

Theorem 2.1 (Ong [18]): Let $\mathcal{G}$ be a recursion scheme of order n, and $\mathcal{A}$ be an alternating parity tree automaton. The problem of checking whether $\mathcal{A}$ accepts $[\![\mathcal{G}]\!]$ is n-EXPTIME-complete.

Remark 2.1: In this paper, we only consider recursion schemes whose value trees do not contain $\perp$. Given a recursion scheme $\mathcal{G}$ and an alternating parity tree automaton $\mathcal{A}$, one can construct $\mathcal{G}'$ and $\mathcal{A}'$ such that (i) the value tree of $\mathcal{G}'$ does not contain $\perp$, and (ii) $\mathcal{A}'$ accepts $\mathcal{G}'$ if, and only if, $\mathcal{A}$ accepts $\mathcal{G}$.

Example 2.2: Let Σ be the alphabet used in Example 2.1. Let $\mathcal{A}_1$ be the alternating parity tree automaton $(\Sigma, \{q_0, q_1\}, \delta_1, q_0, \{q_0 \mapsto 2, q_1 \mapsto 1\})$, where, for each $q \in \{q_0, q_1\}$, $\delta_1(q, \mathtt{a}) = (1, q) \land (2, q)$, $\delta_1(q, \mathtt{b}) = (1, q_1)$, and $\delta_1(q, \mathtt{c}) = \mathtt{true}$. Then, $\mathcal{A}_1$ accepts a Σ-labelled tree t if, and only if, in every path of t, $\mathtt{c}$ occurs eventually after $\mathtt{b}$ occurs.

Parity Games: A *parity game* is a tuple $(V_\forall, V_\exists, v_0, E, \Omega)$ such that $E \subseteq V \times V$ is the edge relation of a directed graph whose node-set V is the disjoint union of $V_\forall$ and $V_\exists$; $v_0 \in V$ is the start node; and $\Omega : V \longrightarrow \{0, \cdots, M-1\}$ assigns a priority to each node. A play consists in the players, $\forall$ and $\exists$, taking turns to move a token along the edges of the graph. At a given stage of the play, suppose the token is on node $v \in V_\forall$ (respectively $v \in V_\exists$), then $\forall$ (respectively $\exists$) chooses an edge (v, v') and moves the token onto v'. At the start of a play, the token is placed on v_0. Thus we define a *play* to be a finite or infinite path $\pi = v_0\, v_{n_1}\, v_{n_2} \cdots$ in the graph that starts from v_0. Suppose π is a maximal play. The winner of π is determined as follows:

- If π is finite, and it ends in a $V_\exists$-node (respectively $V_\forall$-node), then $\forall$ (respectively $\exists$) wins.
- If π is infinite, then $\exists$ wins if π satisfies the *parity condition* i.e. the largest number that occurs infinitely often in the sequence $\Omega(v_0)\, \Omega(v_{n_1})\, \Omega(v_{n_2}) \cdots$ is even; otherwise $\forall$ wins.

A $\exists$-*strategy* (or *strategy*, for short) $\mathcal{W}$ is a map from plays that end in a $V_\exists$-node to a node that extends the play. We say that a strategy $\mathcal{W}$ is *winning* just if $\exists$ wins every (maximal) play π that *conforms* with the strategy (i.e. for every prefix π_0 of π that ends in a $V_\exists$-node, $\pi_0\ \mathcal{W}(\pi_0)$ is a prefix of π). Finally a strategy $\mathcal{W}$ is *memoryless* just if $\mathcal{W}$'s action is determined by the last node of the play;

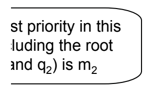


Figure 1. A tree function described by $(q_1, m_1) \wedge (q_2, m_2) \rightarrow q$

formally, for all plays π_1 and π_2 that are consistent with $\mathcal{W}$, if their respective last nodes are the same $V_\exists$-node, then $\mathcal{W}(\pi_1) = \mathcal{W}(\pi_2)$. It is known that if there is a winning strategy for a parity game, then there is also a memoryless winning strategy for the game.

III. Type system

Given an alternating parity tree automaton $\mathcal{A} = (Q, \Sigma, \delta, q_I, \Omega)$, we construct a type system $\mathcal{T}_\mathcal{A}$ in which a recursion scheme is well-typed if, and only if, the tree generated by the recursion scheme is accepted by $\mathcal{A}$. Let q and m respectively range over the states and priorities of $\mathcal{A}$. We define:

$$\begin{array}{llcl} \textit{Atomic types} & \theta & ::= & q \mid \tau \rightarrow \theta \\ \textit{Types} & \tau & ::= & \bigwedge\{(\theta_1, m_1), \ldots, (\theta_k, m_k)\} \end{array}$$

Notations: We write $(\theta_1, m_1) \wedge \cdots \wedge (\theta_k, m_k)$, or simply $\bigwedge_{i=1}^{k} (\theta_i, m_i)$, for types $\bigwedge\{(\theta_1, m_1), \ldots, (\theta_k, m_k)\}$. We write $\top$ for the type $\bigwedge \emptyset$. Given a priority $\Omega(q)$ for each element q of Q, we extend it to all atomic types by $\Omega(\tau \rightarrow \theta) := \Omega(\theta)$.

Intuitively, the type $(q_1, m_1) \wedge \cdots \wedge (q_k, m_k) \rightarrow q$ describes a function that takes a tree (say, x) that can be accepted from each of the states $q_1, \ldots, q_k$, and returns a tree that is accepted from state q. (See Figure 1 for an illustration.) The priority m_i describes the maximal priority in the path from the root of the output tree (of type q) to the input tree of type q_i. In other words, the input tree can be used as a tree of type q_i only after visiting a state of priority m_i, and before visiting a state of priority greater than m_i.

The set of "well-formed" types is defined by the relations $\tau :: \kappa$ and $\theta ::_a \kappa$, which should be read "τ is a type of kind κ" and "θ is an atomic type of kind κ" respectively. We also impose a condition on priorities.

Definition 3.1 (Well-formed types): The relations $\tau :: \kappa$ and $\theta ::_a \kappa$ are the least relations closed under the following rules:

$$\frac{}{q_i ::_a \mathsf{o}} \qquad \frac{\tau :: \kappa_1 \qquad \theta ::_a \kappa_2}{\tau \rightarrow \theta \ ::_a \ \kappa_1 \rightarrow \kappa_2}$$

$$\frac{\theta_i ::_a \kappa \qquad \text{for each } i \in \{1, \ldots, n\}}{\bigwedge\{(\theta_1, m_1), \ldots, (\theta_n, m_n)\} \ :: \ \kappa}$$

A type τ (respectively, atomic type θ) is *well-formed* just if (i) $\tau :: \kappa$ (respectively, $\theta ::_a \kappa$) for some κ, and (ii) for each subexpression of the form $\bigwedge_{i=1}^{k} (\theta_i, m_i) \rightarrow \theta'$, we have $m_i \geq \mathit{max}(\Omega(\theta'), \Omega(\theta_i))$ for each $1 \leq i \leq k$.

For example, $q_1 \wedge ((q_2, 1) \rightarrow q_3)$ is not well-formed, as it combines types of different kinds. $(q_1, m_1) \wedge (q_2, m_2) \rightarrow q$ is well-formed if $m_1 \geq \Omega(q), \Omega(q_1)$ and $m_2 \geq \Omega(q), \Omega(q_2)$; this reflects the intuition that m_1 and m_2 are the largest priorities in the paths shown in Figure 1, *including the root and leaf nodes*. Henceforth we consider only well-formed types.

Type Environment and Judgement: A *type judgement* has the form $\Gamma \vdash t : \theta$, where t is a λ-term (where non-terminals are treated as variables), and Γ, called a *type environment*, is a set of bindings of the form $x : (\theta, m)^b$. Expressions of the form, $(\theta, m)^b$ where $b \in \{\mathtt{t}, \mathtt{f}\}$, are called *flagged types*, which are ranged over by meta-variables σ.

Note that Γ may contain multiple occurrences of the same variable. In the type environment Γ, each (atomic) type of a variable is annotated with a flag b, indicating *when* the variable can be used as a value of that type. For example, $x : (q, m)^\mathtt{t} \in \Gamma$ means that x can be used only before visiting a state with priority larger than m. If the flag is $\mathtt{f}$ (i.e. $x : (q, m)^\mathtt{f} \in \Gamma$), then it is additionally required that x can be used only after visiting a state with priority m. Thus, if $x : (q, m)^\mathtt{f} \in \Gamma$, then the largest priority seen in the path (of the value tree) from the current tree node to the node where x is used must be exactly m.

Example 3.1: Suppose the priority of q, $\Omega(q)$, is 0.

(i) The judgement $\{x : (q, 1)^\mathtt{f}\} \vdash x : q$ is invalid. The type environment says that x can be used only after visiting a state of priority 1, but the current state q has only priority 0, so x cannot be used.

(ii) The judgement $\{x : (q, 1)^\mathtt{t}\} \vdash x : q$ is however valid: since the flag is $\mathtt{t}$, x can be used any time before a priority larger than 1 is seen.

(iii) The judgement $\{x : (q, 1)^\mathtt{f}, y : ((q, 1) \rightarrow q, 0)^\mathtt{f}\} \vdash y\ x : q$ is also valid, because y uses the argument x only after visiting a state of priority 1.

Notations: We shall often drop the set braces to save writing. We write $\Gamma, x : \bigwedge_{i=1}^{k} (\theta_i, m_i)^{b_i}$ as a shorthand for $\Gamma \cup \{x : (\theta_1, m_1)^{b_1}, \ldots, x : (\theta_k, m_k)^{b_k}\}$ where x is assumed *not* to occur in Γ. We write $\mathit{dom}(\Gamma)$ for the set $\{x \mid \exists \theta, m, b \,.\, x : (\theta, m)^b \in \Gamma\}$. For technical convenience, we assume type environments Γ satisfy an *injectivity* condition: If $x : (\theta, m)^b, x : (\theta, m)^{b'} \in \Gamma$ then $b = b'$.

The type judgement $\Gamma \vdash t : \theta$ is defined by induction over the following rules.

$$\frac{(\theta,m)^b \uparrow \Omega(\theta) = (\theta,m)^{\mathtt{t}}}{x:(\theta,m)^b \vdash x:\theta} \quad \text{(T-VAR)}$$

$$\frac{\{(i,q_{ij}) \mid 1\le i\le n, 1\le j\le k_i\} \text{ satisfies } \delta_{\mathcal{A}}(q,a)}{\begin{array}{l}\emptyset \vdash \\ a:\bigwedge_{j=1}^{k_1}(q_{1j},m_{1j}) \to \cdots \to \bigwedge_{j=1}^{k_n}(q_{nj},m_{nj}) \to q \\ \text{where } m_{ij} = \mathit{max}(\Omega(q_{ij}),\Omega(q))\end{array}} \quad \text{(T-CONST)}$$

$$\frac{\begin{array}{c}\Gamma_0 \vdash t_0 : (\theta_1,m_1)\wedge\cdots\wedge(\theta_k,m_k)\to\theta \\ \Gamma_i \uparrow m_i \vdash t_1:\theta_i \text{ for each } i\in\{1,\dots,k\}\end{array}}{\Gamma_0\cup\Gamma_1\cup\cdots\cup\Gamma_k \vdash t_0\ t_1:\theta} \quad \text{(T-APP)}$$

$$\frac{\Gamma, x:\bigwedge_{i\in I}(\theta_i,m_i)^{\mathtt{f}} \vdash t:\theta \qquad I\subseteq J}{\Gamma\vdash \lambda x.t:\bigwedge_{i\in J}(\theta_i,m_i)\to\theta} \quad \text{(T-ABS)}$$

The operation $(\cdot)\uparrow m$ used in the rules T-VAR and T-APP above are defined as follows.

$$(\theta,m)^b \uparrow m' := \begin{cases} (\theta,m)^b & \text{if } m' < m \\ (\theta,m)^{\mathtt{t}} & \text{if } m' = m \\ \text{undefined} & \text{if } m' > m \end{cases}$$
$$\{x_1:\sigma_1,\dots,x_n:\sigma_n\}\uparrow m := \{x_1:\sigma_1\uparrow m,\dots,x_n:\sigma_n\uparrow m\}.$$

In T-VAR, x can be used either if $b=\mathtt{t}$ and the current priority is less than or equal to m, or if $b=\mathtt{f}$ and the current priority is m. The rule T-CONST is for input symbols. The premise means that when reading a, the automaton $\mathcal{A}$ in state q can spawn new states q_{ij}, and read the i-th subtree with state q_{ij}. Thus, in order for a tree $a\,t_1\cdots t_n$ to have type q (i.e. to be accepted from state q), it is sufficient that t_i has type q_{ij} for every $j\in\{1,\dots,k_i\}$. For example, for the automaton $\mathcal{A}_1$ in Example 2.2, a has type $(q_0,2)\to(q_0,2)\to q_0$ and $(q_1,1)\to(q_1,1)\to q_1$.

In T-APP, the first premise requires that the argument of t_0 should have types $\theta_1,\dots,\theta_k$. Thus, the second premise requires that t_1 has these types. Furthermore, the first premise means that the argument is used as a value of type θ_i only in a context where the largest priority that has been seen (since the function t_0 is called) is m_i. The operation $\Gamma_i\uparrow m_i$ takes that into account.

The rule T-ABS for abstraction is standard, except that weakening on x is allowed,[3] and that the bindings on x are annotated with flag $\mathtt{f}$, indicating that x can be used only after the expected priority is seen.

Remark 3.1: In rule T-APP, k can be 0. Thus, for example, $x:(\top\to\theta,\Omega(q))^{\mathtt{f}}\vdash x\ t:q$ is derivable for any t, even if t is ill-typed or contains variables other than x.

Example 3.2: Recall the automaton $\mathcal{A}_1$ in Example 2.2. Let $\theta=(q_0,2)\wedge(q_1,2)\to q_0$, $\theta_{\mathtt{a}}=(q_0,2)\to(q_0,2)\to q_0$, and $\Gamma_1 = F:(\theta,2)^{\mathtt{t}}, x:(q_1,2)^{\mathtt{t}}$. The term $\lambda x.\mathtt{a}\ x\ (F(\mathtt{b}\,x))$ is typed as follows.

$$\frac{\dfrac{\emptyset\vdash\mathtt{a}:\theta_{\mathtt{a}} \quad x:(q_0,2)^{\mathtt{t}}\vdash x:q_0 \quad \Gamma_1\vdash F(\mathtt{b}\,x):q_0}{F:(\theta,2)^{\mathtt{f}}, x:(q_0,2)^{\mathtt{f}}\wedge(q_1,2)^{\mathtt{f}}\vdash \mathtt{a}\ x\ (F(\mathtt{b}\,x)):q_0}}{F:(\theta,2)^{\mathtt{f}}\vdash\lambda x.\mathtt{a}\ x\ (F(\mathtt{b}\,x)):\theta}$$

Here, $\Gamma_1\vdash F(\mathtt{b}\,x):q_0$ is derived by:

$$\frac{F:(\theta,2)^{\mathtt{t}}\vdash F:\theta \quad \Gamma_2\vdash\mathtt{b}\,x:q_0 \quad \Gamma_2\vdash\mathtt{b}\,x:q_1}{\Gamma_1\vdash F(\mathtt{b}\,x):q_0}$$

where $\Gamma_2 = x:(q_1,2)^{\mathtt{t}}$, and $\Gamma_2\vdash\mathtt{b}\,x:q_i$ is derived from $\emptyset\vdash b:(q_1,\Omega_1(q_i))\to q_i$ and $\Gamma_2\vdash x:q_1$.

Typing for recursion schemes: We now define the typing relation $\vdash_{\mathcal{A}}\mathcal{G}$ for recursion schemes in terms of parity games.

Definition 3.2: Given an alternating parity tree automaton $\mathcal{A}=(\Sigma,Q,\delta,q_I,\Omega)$ and a recursion scheme $\mathcal{G}=(\Sigma,\mathcal{N},\mathcal{R},S)$, we define a parity game $(V_\forall,V_\exists,(S,q_I,\Omega(q_I)),E,\Omega')$ as follows.

$$\begin{array}{rcl} V_\exists &=& \{(F,\theta,m)\mid F\in\mathit{dom}(\mathcal{N}),\theta::\mathcal{N}(F)\} \\ V_\forall &=& \{\Gamma\mid\mathit{dom}(\Gamma)\subseteq\mathit{dom}(\mathcal{N}), \text{all flags in } \Gamma \text{ are } \mathtt{f}\} \\ E &=& \{((F,\theta,m),\Gamma)\mid\Gamma\vdash\mathcal{R}(F):\theta\} \quad\cup \\ && \{(\Gamma,(F,\theta,m))\mid F:(\theta,m)^{\mathtt{f}}\in\Gamma\} \end{array}$$

and the priority function Ω' maps (F,θ,m) to m and Γ to 0. $\mathcal{G}$ is *well-typed*, written $\vdash_{\mathcal{A}}\mathcal{G}$, if player $\exists$ has a winning strategy for the game.

The above definition may be understood intuitively as follows. The player $\exists$ tries to prove that the recursion scheme is well-typed, and the other player $\forall$ tries to disprove it. At a node (F,θ,m), the player $\exists$ has to pick a type environment Γ under which $\mathcal{R}(F)$ has type θ. The player $\forall$ then picks a binding $F':(\theta',m')^{\mathtt{f}}$ from Γ, and asks $\exists$ to show why F' has type θ', and then it is again the player $\exists$'s turn to choose a type environment Γ' under which $\mathcal{R}(F')$ has type θ'. The play continues indefinitely, or ends when one of the players is unable to move. The player $\exists$ wins a play if at some point, it chooses the empty type environment (so that $\forall$ cannot pick a binding), or if the play is infinite, and the largest priority occurring infinitely often is even. The recursion scheme is well-typed if the player $\exists$ has a strategy that wins every play, whatever choice is made by the player $\forall$.

Example 3.3: Recall the recursion scheme $\mathcal{G}$ in Example 2.1 and the automaton $\mathcal{A}_1$ in Example 2.2. Let θ be $(q_0,2)\wedge(q_1,2)\to q_0$. Then, $F:(\theta,2)^{\mathtt{f}}\vdash F\ \mathtt{c}:q_0$ and $F:(\theta,2)^{\mathtt{f}}\vdash\lambda x.\mathtt{a}\ x\ (F(\mathtt{b}\,x)):\theta$ are valid judgments (recall Example 3.2 for the derivation of the second judgement). A memoryless winning strategy $\mathcal{W}$ for the parity game is given by: $\mathcal{W}(S,q_0,2)=F:(\theta,2)^{\mathtt{f}}$ and $\mathcal{W}(F,\theta,2)=F:(\theta,2)^{\mathtt{f}}$.

Remark 3.2: Note that it is unsound to use the usual rule for recursion:

$$\frac{\Gamma,F:(\theta,m)^{\mathtt{f}}\vdash\mathcal{R}(F):\theta}{\Gamma\vdash\mathcal{R}(F):\theta}$$

[3]For technical convenience, this is the only place where weakening is allowed.

and define $\vdash_{\mathcal{A}} \mathcal{G}$ by $\emptyset \vdash S : q_I$. For example, let $\mathcal{A}'_1$ be the alternating parity tree automaton obtained from $\mathcal{A}_1$ of Example 2.2 by replacing the inital state replaced with q_1, and let $\mathcal{G}$ be the recursion scheme $\mathcal{G} = (\Sigma, \{S\}, \{S \mapsto b(S)\}, S)$. Then, $\emptyset \vdash S : q_1$ is derivarable, but the value tree of $\mathcal{G}$ is not accepted by $\mathcal{A}'_1$.

The standard rule for recursion can be considered a degenerate case of our definition (using parity games), where all the priorities are 0. In fact, Kobayashi's type system [14] is obtained as a special case of our type system $\mathcal{T}_{\mathcal{A}}$ where the priorities are restricted to 0.

IV. Correctness of the Type System

A. Soundness

Suppose that we are given a recursion scheme $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ and an alternating parity tree automaton $\mathcal{A}$ such that $\vdash_{\mathcal{A}} \mathcal{G}$. The goal is to show that there exists an *accepting run-tree* of $\mathcal{A}$ over $[\![\mathcal{G}]\!]$.

We first note the following type preservation property, which can be proved in a standard manner.

Lemma 4.1 (Type preservation by β-reduction): If $\Gamma \vdash (\lambda x.t_0)t_1 : \theta$, then there exists Γ' such that $\Gamma' \vdash [t_1/x]t_0 : \theta$ and $\Gamma' \subseteq \Gamma$.

Now we shall define a rewrite system for generating an *accepting run-tree* of $\mathcal{A}$ over the value tree of $\mathcal{G}$. The rewrite relation is a binary relation on (finite, unranked) $\mathbf{RLab}$-labelled trees, where an element of $\mathbf{RLab}$ is either of the form $\langle \alpha, q \rangle$ or $\langle \alpha, l, \Gamma \vdash t : q \rangle$ where $\Gamma \vdash t : q$ holds. Here l is a natural number, and α is an element of $\{1, \dots, A\}^*$, where A is the largest arity of the terminal symbols of $\mathcal{G}$. By the assumption $\vdash_{\mathcal{A}} \mathcal{G}$, there exists a (memoryless) winning strategy $\mathcal{W}$ for the parity game associated with $\vdash_{\mathcal{A}} \mathcal{G}$. $\mathcal{W}$ can be considered as a map from tuples of the form (F, θ, m) to type environments. We write $\Gamma_{(F,\theta,m)}$ for $\mathcal{W}(F, \theta, m)$ below.

In a type judgment $\Gamma \vdash F\widetilde{t} : q$, we often annotate the head symbol F with its type and priority, as $\Gamma \vdash F^{(\theta,m)}\widetilde{t} : q$. It means that $\Gamma \vdash F\widetilde{t} : q$ is derived from the typing $F : (\theta, m)^b \vdash F : \theta$ for the occurence of F as the head symbol, followed by applications of T-App.

The initial tree of the rewrite system is $\langle \epsilon, 1, S^0 : (q_I, \Omega(q_I))^{\mathtt{f}} \vdash S^0 : q_I \rangle$. Here, each non-terminal symbol is annotated with a natural number, to indicate when the symbol was introduced. The rewrite relation $t \rhd t'$ is defined by induction over the following rules:

(i) If $\Gamma \vdash F_i^{l',(\theta,m)}\widetilde{t} : q$ holds, then

$$\langle \alpha, l, \Gamma \vdash F_i^{l'}\widetilde{t} : q \rangle \ \rhd\ \langle \alpha, l+1, \Gamma' \vdash [\widetilde{t}/\widetilde{x}]\rho(t') : q \rangle$$

writing $\rho(-) := [F_1^l/F_1, \dots, F_n^l/F_n](-)$ and $\mathcal{R}(F_i) = \lambda\widetilde{x}.t'$.

Here, Γ' is determined as follows: Take the derivation of $\Gamma \vdash F_i^{l',(\theta,m)}\widetilde{t} : q$, and replace the T-Var instance $F : (\theta, m)^b \vdash F : \theta$ by $\rho(\Gamma_{(F_i,\theta,m)}) \vdash \rho(\mathcal{R}(F_i)) : \theta$, yielding (a derivation for) $\Gamma_1 \cup \rho(\Gamma_{(F_i,\theta,m)}) \vdash \rho(\mathcal{R}(F_i))\widetilde{t}$. Note that $\Gamma_1 \cup \{F : (\theta, m)^b\} = \Gamma$ holds but *not* necessarily $\Gamma_1 = \Gamma \backslash \{F : (\theta, m)^b\}$. By the type preservation property (Lemma 4.1), there exists Γ' such that $\Gamma' \subseteq \Gamma_1 \cup \rho(\Gamma_{(F_i,\theta,m)})$ and $\Gamma' \vdash [\widetilde{t}/\widetilde{x}]\rho(t') : q$. Thus, we choose one such Γ' above.
Note that it is necessary to rename non-terminals F_i to $F_{l,i}$ in order to state Lemma 4.3.

(ii) If $\{(i, q_{i,j}) \mid 1 \le i \le n, 1 \le j \le k_i\}$ satisfies $\delta_{\mathcal{A}}(q, a)$, and $\Gamma \vdash at_1 \cdots t_n : q$ is derived from $\Gamma_{i,j} \vdash t_i : q_{i,j}$, then

$$\begin{array}{l} \langle \alpha, l, \Gamma \vdash at_1 \cdots t_n : q \rangle \rhd \\ \langle \alpha, q \rangle (\langle \alpha 1, l, \Gamma_{1,1} \vdash t_1 : q_{1,1} \rangle, \dots, \langle \alpha 1, l, \Gamma_{1,k_1} \vdash t_1 : q_{1,k_1} \rangle \\ \quad \dots \langle \alpha n, l, \Gamma_{n,1} \vdash t_n : q_{n,1} \rangle, \dots, \langle \alpha n, l, \Gamma_{n,k_n} \vdash t_n : q_{n,k_n} \rangle) \end{array}$$

(iii) If $T \rhd T'$ then $C[T] \rhd C[T']$ for every tree context C.

The following lemma follows from the definition of $\rhd$.

Lemma 4.2: If $\langle \epsilon, 1, S^0 : (q_I, \Omega(q_I))^{\mathtt{f}} \vdash S^0 : q_I \rangle \rhd^* C[\langle \alpha, l, \Gamma \vdash t : q \rangle]$, then $\Gamma \vdash t : q$ holds.

By the *priority* of a tree context $C[\,]_q$ (wherein the hole $[\,]$ is assumed to have the state q), written $\Omega(C[\,]_q)$, we mean the largest priority occurring in the path from the root of $C[\,]_q$ to its hole $[\,]_q$. The following lemma confirms that variables in the type environment are used correctly, according to the intuition on type environments explained in Section III.

Lemma 4.3: Suppose $\langle \alpha_0, l_0, \Gamma_0 \vdash s_0 : q_0 \rangle \rhd^* C[\langle \alpha, l, \Gamma \vdash F^{(\theta,m)}\widetilde{t} : q \rangle]$, and F is not introduced by renaming (i.e. via $\rho(-)$) in any of the intermediate reduction steps. Then, either (i) $F : (\theta, m)^{\mathtt{f}} \in \Gamma_0$ and $m = \Omega(C[\,]_q)$; or (ii) $F : (\theta, m)^{\mathtt{t}} \in \Gamma_0$ and $m \ge \Omega(C[\,]_q)$ hold.

Theorem 4.4 (Soundness): Let $\mathcal{A}$ be an alternating parity tree automaton, and $\mathcal{G}$ be a recursion scheme. If $\vdash_{\mathcal{A}} \mathcal{G}$, then the tree generated by $\mathcal{G}$ is accepted by $\mathcal{A}$.

Proof: We write $T^\sharp$ for the (unranked) tree obtained by replacing each label of the form $\langle \alpha, l, \Gamma \vdash t : q \rangle$ with $\langle \alpha, q \rangle$. Let $T_0 \rhd T_1 \rhd T_2 \rhd T_3 \rhd \cdots$ be a maximal[4] fair (possibly infinite) reduction sequence, where $T_0 := \langle \epsilon, 1, S^0 : (q_I, \Omega(q_I))^{\mathtt{f}} \vdash S^0 : q_I \rangle$. By the definition of $\rhd$, every $T_i{}^\sharp$ is a prefix[5] of a run-tree of $\mathcal{A}$. By Lemma 4.2, reductions of $\langle \epsilon, 1, S^0 : (q_I, \Omega(q_I))^{\mathtt{f}} \vdash S^0 : q_I \rangle$ never get stuck: It either ends up with a finite tree all of whose labels are of the form $\langle \alpha, q \rangle$, or continues indefinitely. Thus, every leaf of the form $\langle \alpha, l, \Gamma \vdash t : q \rangle$ occuring in the sequence is eventually reduced. Thus, together with the assumption that the value tree of $\mathcal{G}$ does not contain $\perp$ (Remark 2.1), it follows that $T := \bigcup_{i \in \omega} T_i{}^\sharp$ is a run-tree (i.e. a tree that satisfies the conditions on accepting run-trees except the parity condition) of $\mathcal{A}$ over the value tree of $[\![\mathcal{G}]\!]$.

[4]A reduction sequence is *maximal* if it is either infinite or finite and the last tree is irreducible.

[5]A tree T_1 is a *prefix* of T_2 if $dom(T_1) \subseteq dom(T_2)$ and $T_1(\alpha) = T_2(\alpha)$ for every $\alpha \in dom(T_1)$.

It remains to show that T satisfies the parity condition. Now, for any infinite path π of T, there must exist an infinite reduction sequence:

$$\begin{aligned}&\langle \epsilon, 1, S^0 : (q_I, \Omega(q_I))^{\mathtt{f}} \vdash S^0 : q_I\rangle\\ \rhd^*\ & C_1[\langle \alpha_1, l_1, \Gamma_1 \vdash F^1_{i_1}\ \widetilde{t}_1 : q_1\rangle]\\ \rhd^*\ & C_1[C_2[\langle \alpha_2, l_2, \Gamma_2 \vdash F^{l_1}_{i_2}\ \widetilde{t}_2 : q_2\rangle]]\\ \rhd^*\ & C_1[C_2[C_3[\langle \alpha_3, l_3, \Gamma_3 \vdash F^{l_2}_{i_3}\ \widetilde{t}_3 : q_3\rangle]]]\ \rhd^* \cdots\end{aligned}$$

such that the holes of $C_1, C_1[C_2], C_1[C_2[C_3]], \ldots$ occur in the path. For each $k \geq 0$, the reduction $\langle \alpha_k, l_k, \Gamma_k \vdash F^{l_{k-1}}_{i_k}\widetilde{t}_k : q_k\rangle \rhd^* C_{k+1}[\langle \alpha_{k+1}, l_{k+1}, \Gamma_{k+1} \vdash F^{l_k}_{i_{k+1}}\widetilde{t}_{k+1} : q_{k+1}\rangle]$ must be of the form

$$\begin{aligned}&\langle \alpha_k, l_k, \Gamma_k \vdash F^{l_{k-1}}_{i_k}\widetilde{t}_k : q_k\rangle\\ \rhd\ & \langle \alpha_k, l_k+1, \Gamma'_k \vdash [\widetilde{t}_k/\widetilde{x}]\rho(t') : q_k\rangle\\ \rhd^*\ & C_{k+1}[\langle \alpha_{k+1}, l_{k+1}, \Gamma_{k+1} \vdash F^{l_k,(\theta_{k+1},m_{k+1})}_{i_{k+1}}\widetilde{t}_{k+1} : q_{k+1}\rangle]\end{aligned}$$

where $\rho := [F^{l_k}_1/F_1, \ldots, F^{l_k}_n/F_n]$ and $\mathcal{R}(F_{i_k}) = \lambda\widetilde{x}.t'$, with $\Gamma'_k \subseteq \Gamma_1 \cup \rho(\Gamma_{(F_{i_k},\theta_k,m_k)})$. Note that all the bindings on $F^{l_k}_{i_{k+1}}$ in $\rho(\Gamma_{(F_{i_k},\theta_k,m_k)})$ have the flag $\mathtt{f}$. Thus, by Lemma 4.3, $\Omega(C_{k+1}[\,]_{q_{k+1}}) = m_{k+1}$ and $F^{l_k}_{i_{k+1}} : (\theta_{k+1}, m_{k+1})^{\mathtt{f}} \in \Gamma'_k$, which implies $F_{i_{k+1}} : (\theta_{k+1}, m_{k+1})^{\mathtt{f}} \in \Gamma_{(F_{i_k},\theta_k,m_k)}$.

Now from the preceding infinite $\rhd$-reduction sequence, we can extract an infinite sequence

$$\begin{aligned}&(F_1, q_I, \Omega(q_I))\ \Gamma_{(F_1,q_I,0)}\ (F_{i_1}, \theta_1, m_1)\ \Gamma_{(F_{i_1},\theta_1,m_1)}\\ &\quad (F_{i_2}, \theta_2, m_2)\ \Gamma_{(F_{i_2},\theta_2,m_2)} \cdots\end{aligned}$$

which is a winning play. It follows that the largest priority that occurs infinitely often in $m_1, m_2, \ldots$ is even. Therefore, the largest priority that occurs in the infinite path π of t must also be even. ■

B. Completeness

Let $\mathcal{A}$ be an alternating parity tree automaton. Assume an accepting run-tree of $\mathcal{A}$ over the value tree of a recursion scheme $\mathcal{G}$. The goal is to show $\vdash_{\mathcal{A}} \mathcal{G}$.

We define a reduction relation $\gtrdot$ on (finite, unranked) $\mathbf{RLab}'$-labelled trees as follows, where an element of $\mathbf{RLab}'$ is either of the form $\langle \alpha, q\rangle$ or $\langle \beta, l, t, q\rangle$. Here l is a natural number, β is a sequence of pairs of natural numbers, and α is an element of $\{1, \ldots, A\}^*$, where A is the largest arity of the terminal symbols of $\mathcal{G}$. We use β and l to uniquely identify each leaf introduced by reductions. The initial tree is $\langle \epsilon, 0, S, q_I\rangle$. The reduction relation $\gtrdot$ is defined by induction over the following rules:

(i) If $\mathcal{R}(F) = \lambda\widetilde{x}.t'$, then:

$$\langle \beta, l, F\,\widetilde{t}, q\rangle \gtrdot \langle \beta, l+1, [\widetilde{t}/\widetilde{x}]t', q\rangle$$

(ii) If $\mathit{fst}(\beta) = \alpha$ and the children of the node $\langle \alpha, q\rangle$ of the run-tree are $\{\langle \alpha i, q_{i,j}\rangle \mid 1 \leq i \leq n, 1 \leq j \leq k_i\}$, then:

$$\begin{aligned}&\langle \beta, l, at_1 \cdots t_n, q\rangle \gtrdot\\ &\quad \langle \mathit{fst}(\beta), q\rangle(\langle \beta(1,1), l, t_1, q_{1,1}\rangle, \ldots, \langle \beta(1,k_1), l, t_1, q_{1,k_1}\rangle,\\ &\quad \ldots \langle \beta(n,1), l, t_n, q_{n,1}\rangle, \ldots, \langle \beta(n,k_n), l, t_n, q_{n,k_n}\rangle)\end{aligned}$$

Here $\mathit{fst}((m_1,n_1)(m_2,n_2)(m_3,n_3)\cdots) = m_1m_2m_3\cdots$.

(iii) If $t \gtrdot t'$, then $C[t] \gtrdot C[t']$ for any tree context C.

There is a (fair) infinite reduction sequence

$$\langle \epsilon, 0, S, q_I\rangle \gtrdot T_1 \gtrdot T_2 \gtrdot \cdots$$

such that $\bigsqcup {T_i}^{\perp}$ coincides with the accepting run-tree of $\mathcal{A}$ over the value tree of $\mathcal{G}$. We pick one such infinite reduction sequence, and extract type information from it, as shown below.

We assume below that each subterm is implicitly labelled, so that different occurrences of the same term are distinguished. For example, when we write $\langle \beta, l, t_0t_1, q\rangle \gtrdot^* C[\langle \beta', l', t_1t_2, q'\rangle]$, we assume that t_1 in t_1t_2 originates from t_1 in the argument position of t_0t_1 (i.e. the former t_1 is a *residual* of the latter t_1 w.r.t. the reduction sequence). As before, we write $\Omega(C[\,]_q)$ for the largest priority in the path from the root of the $\mathbf{RLab}'$-tree context C to the hole $[\,]_q$ which is assumed to have state q.

Type $\theta_{(t_0,\beta,l)}$ of a prefix t_0: A term t_0 is called a *prefix* of t if t is of the form $t_0t_1\cdots t_k$. For each leaf $\langle \beta, l, t, q\rangle$ and a prefix t_0 of t, we can determine the type $\theta_{(t_0,\beta,l)}$ by induction on the kind of t_0 as follows.

(i) If the kind of t_0 is o, then $\theta_{(t_0,\beta,l)} := q$ (note that the leaf is $\langle \beta, l, t_0, q\rangle$).

(ii) If the kind of t_0 is $\kappa_1 \to \cdots \to \kappa_n \to \mathsf{o}$, then the leaf is of the form $\langle \beta, l, t_0t_1\cdots t_n, q\rangle$. Let S_i be the set of pairs $(\theta_{(t_i,\beta',l')}, \Omega(C[\,]_{q'}))$ such that $\langle \beta, l, t_0t_1\cdots t_n, q\rangle \gtrdot^* C[\langle \beta', l', t_i\widetilde{t'}, q'\rangle]$. Note that since the kind of κ_i is less than that of t_0, by the induction hypothesis, we can determine $\theta_{(t_i,\beta',l')}$. Note also that although the set of trees $C[\langle \beta', l', t_i\widetilde{t'}, q'\rangle]$ such that $\langle \beta, l, t_0t_1\cdots t_n, q\rangle \gtrdot^* C[\langle \beta', l', t_i\widetilde{t'}, q'\rangle]$ may be infinite, S_i is finite. Thus we can define

$$\theta_{(t_0,\beta,l)} := \bigwedge S_1 \to \cdots \to \bigwedge S_n \to q.$$

Type environment $\Gamma_{(t_0,\beta,l)}$ of a prefix t_0: Next, we determine a type environment $\Gamma_{(t_0,\beta,l)}$ for each prefix term t_0 of the leaf $\langle \beta, l, t_0t_1\cdots t_n, q\rangle$, with a view to proving $\Gamma_{(t_0,\beta,l)} \vdash t_0 : \theta_{(t_0,\beta,l)}$, by induction on the structure of the term.

- If $t_0 = a\ (\in \Sigma)$, then $\Gamma_{(t_0,\beta,l)} := \emptyset$.
- If $t_0 = F\ (\in \mathcal{N})$, then $\Gamma_{(F,\beta,l)} := F : (\theta_{(F,\beta,l)}, \Omega(q))^{\mathtt{f}}$.
- If $t_0 = t_{0,1}t_{0,2}$, then let S be the set of triples

$$(\beta', l', \Omega(C[\,]_{q'}))$$

such that $\langle \beta, l, t_0t_1\cdots t_n, q\rangle \gtrdot^* C[\langle \beta', l', t_{0,2}\widetilde{t'}, q'\rangle]$. Let S' be a subset of S such that for every $(\beta'', l'', m) \in S$, there exists exactly one $(\beta', l', m) \in S'$ such that $\theta_{(t_{0,2},\beta',l')} = \theta_{(t_{0,2},\beta'',l'')}$. We then define $\Gamma_{(t_0,\beta,l)}$ as

$$\Gamma_{(t_{0,1},\beta,l)} \cup (\bigcup\{\Gamma_{(t_{0,2},\beta',l')} \Uparrow m \mid (\beta', l', m) \in S'\})$$

where $\Gamma \Uparrow m := \{x : (\theta, \mathit{max}(m, m'))^b \mid x : (\theta, m')^b \in \Gamma\}$.

Remark 4.1: The typing rule T-App requires that there is exactly one type environment for each (θ_i, m_i). Accordingly, by construction S' contains exactly one element for each (θ, m) of type $t_{0,2}$.

The following lemma intuitively states that for each binding of a type environment $\Gamma_{(t,\beta,l)}$, there exists at least one corresponding use of the variable.

Lemma 4.5: If $\langle \epsilon, 0, S, q_I\rangle \gtrdot^* C[\langle \beta, l, t, q\rangle]$ and $F : (\theta, m)^{\mathtt{f}} \in \Gamma_{(t,\beta,l)}$, then there exist $C', \beta', l', \widetilde{t}', q'$ such that $\langle \beta, l, t, q\rangle \gtrdot^* C'[\langle \beta', l', F\,\widetilde{t}', q'\rangle]$ and $m = \Omega(C'[\,]_{q'})$ with $\theta = \theta_{(F,\beta',l')}$.

The following lemma guarantees the consistency of typing: the conclusion says that the body of F, $\mathcal{R}(F) = \lambda\widetilde{x}.t$, can be given the same type (i.e. $\theta_{(F,\beta,l)}$) as F.

Lemma 4.6: If $\langle \epsilon, 0, S, q_I\rangle \gtrdot^* C[\langle \beta, l, F\widetilde{s}, q\rangle] \gtrdot C[\langle \beta, l+1, [\widetilde{s}/\widetilde{x}]t, q\rangle]$, then there exists Γ such that $\Gamma \vdash \lambda\widetilde{x}.t : \theta_{(F,\beta,l)}$ and $\Gamma \subseteq \Gamma_{([\widetilde{s}/\widetilde{x}]t,\beta,l+1)}$.

Theorem 4.7 (Completeness): Let $\mathcal{A}$ be an alternating parity tree automaton, and $\mathcal{G}$ be a recursion scheme. If the tree generated by $\mathcal{G}$ is accepted by $\mathcal{A}$, then $\vdash_{\mathcal{A}} \mathcal{G}$.

Proof: From an accepting run-tree of $\mathcal{A}$ over the value tree of $\mathcal{G}$, we can construct an infinite reduction sequence $\langle \epsilon, 0, S, q_I\rangle \gtrdot T_1 \gtrdot T_2 \gtrdot \cdots$ that converges to the run-tree. We shall construct a winning strategy $\mathcal{W}$ for the parity game $(V_\forall, V_\exists, v_0, E, \Omega)$ associated with $\vdash_{\mathcal{A}} \mathcal{G} : q_I$ below. We annotate each state Γ of $V_\forall$ occurring in $\mathcal{W}$ with a label of the form $[\beta, l, t]$ to indicate the corresponding node in the reduction sequence $\langle \epsilon, 0, S, q_I\rangle \gtrdot T_1 \gtrdot T_2 \gtrdot \cdots$. Note that by the construction of $\mathcal{W}$ below, $\Gamma^{[\beta,l,t]} \subseteq \Gamma_{(t,\beta,l)}$ holds. The winning strategy $\mathcal{W}$ is defined as follows. Consider a play $\pi\,(F, \theta, m) \in (V_\exists V_\forall)^* V_\exists$ that conforms to $\mathcal{W}$. Let $\Gamma^{[\beta,l,t]}$ be $(S : (q_I, \Omega(q_I))^{\mathtt{f}})^{[\epsilon,0,S]}$ if $\pi = \epsilon$; otherwise, let it be the last state of π (in $V_\forall$). It must be the case that $F : (\theta, m)^{\mathtt{f}} \in \Gamma^{[\beta,l,t]} \subseteq \Gamma_{(t,\beta,l)}$. By Lemma 4.5, there must exist C, β', l' such that

$$\begin{aligned} &\langle \beta, l, t, q_t\rangle \\ \gtrdot^* \;& C[\langle \beta', l', F\widetilde{s}, q'\rangle] \gtrdot C[\langle \beta', l'+1, [\widetilde{s}/\widetilde{x}]t_F, q'\rangle] \end{aligned}$$

with $\Omega(C[\,]_{q'}) = m$ and $\theta = \theta_{(F,\beta',l')}$ where $\mathcal{R}(F) = \lambda\widetilde{x}.t_F$.

By Lemma 4.6, there exists Γ' such that $\Gamma' \vdash \lambda\widetilde{x}.t_F : \theta_{(F,\beta',l')}$ and $\Gamma' \subseteq \Gamma_{([\widetilde{s}/\widetilde{x}]\mathcal{R}(F),\beta',l'+1)}$. We pick one such Γ', and define $\mathcal{W}(\pi\,(F, \theta, m))$ as $\Gamma'^{[\beta',l'+1,[\widetilde{s}/\widetilde{x}]t_F]}$.

To check that $\mathcal{W}$ is indeed winning, consider an infinite play:
$(F_0, q_0, m_0)\,\Gamma_0^{[\beta_0,l_0,t_0]}\,(F_1, \theta_1, m_1)\,\Gamma_1^{[\beta_1,l_1,t_1]}(F_2, \theta_2, m_2)\cdots$ that conforms to $\mathcal{W}$ where $(F_0, q_0, m_0) = (S, q_I, \Omega(q_I))$. Then the reduction sequence $\langle \epsilon, 0, S, q_I\rangle \gtrdot T_1 \gtrdot T_2 \gtrdot \cdots$ must be of the form:

$$\begin{aligned} &\langle \epsilon, 0, S, q_I\rangle \gtrdot \langle \beta_0, l_0, \mathcal{R}(S), q_0\rangle \\ &\gtrdot^* C_1[\langle \beta_1, l_1 - 1, F_1\widetilde{s}_1, q_1\rangle] \gtrdot C_1[\langle \beta_1, l_1, t_1, q_1\rangle] \\ &\gtrdot^* C_1[C_2[\langle \beta_2, l_2 - 1, F_2\widetilde{s}_2, q_2\rangle]] \gtrdot C_1[C_2[\langle \beta_2, l_2, t_2, q_2\rangle]] \\ &\gtrdot^* \cdots \end{aligned}$$

where $\Omega(C_i[\,]_{q_i}) = m_i (i \geq 1)$. Since the reduction sequence converges to the accepting run-tree of $\mathcal{A}$ over the value tree of $\mathcal{G}$, the largest priority that occurs infinitely often in $m_0, m_1, m_2, \ldots$ must be even. Thus, we have $\vdash_{\mathcal{A}} \mathcal{G}$. ■

V. Type Inference Algorithm

Thanks to the development of the previous sections, the model checking of higher-order recursion schemes is reduced to a type inference problem. The reduction allows us to analyze the parameterized complexity of model checking higher-order recursion schemes. The main result is that, assuming that the size of kinds, the largest priority, and the number of states of the alternating parity tree automaton are bounded by a constant, the time complexity of the type checking problem (hence also the recursion scheme model checking problem) is polynomial in the size of the grammar.

The type-checking algorithm consists of the following two phases:

- Step 1: Construct the parity game $(V_\forall, V_\exists, v_0, E, \Omega)$ associated with the type system.
- Step 2: Decide whether there is a winning strategy for the parity game.

We assume below that each rule of the recursion scheme is of the form $F \mapsto \lambda\widetilde{x}.c\ (F_1\ \widetilde{x}_1)\ \cdots (F_J\ \widetilde{x}_J)$, where c is a terminal, a non-terminal, or a variable, and J may be 0. Note that any recursion scheme $\mathcal{G}$ can be transformed into $\mathcal{G}'$ such that $\mathcal{G}'$ satisfies the assumption above and the size of $\mathcal{G}'$ is linear in that of $\mathcal{G}$.

We write A for the maximum arity, N for the order of the recursion scheme, P for the number of rewrite rules, Q for the number of states of the automaton, and $M-1$ for the largest priority of the states. For a kind κ of order n, an upper-bound of the number of types of kind κ, written K_n, is given by:

$$K_0 = Q \qquad K_{n+1} = Q2^{AMK_n}.$$

Note that K_n is bounded by $\mathbf{exp}_n((AQM)^{1+\epsilon})$ for any $\epsilon > 0$, where $\mathbf{exp}_n(x)$ is defined by: $\mathbf{exp}_0(x) = x$ and $\mathbf{exp}_{i+1}(x) = 2^{\mathbf{exp}_i(x)}$.

For step 1, we first compute the set

$$S_i := \{(\Gamma, \theta) \mid \Gamma \vdash \mathcal{R}(F_i) : \theta \text{ and all flags in } \Gamma \text{ are } \mathtt{f}.\}$$

for each non-terminal F_i. Assume that $\mathcal{R}(F_i)$ is of the form $\lambda\widetilde{x}.c(F'_1\widetilde{x}_1)\cdots(F'_J\widetilde{x}_J)$. We first compute:

$$S_{i,0} := \{(\Gamma_0, \theta_0) \mid \Gamma_0 \vdash c : \theta_0, \text{ and } \theta_0 ::_a \kappa_c\}$$

where κ_c is the kind of c and all flags in Γ_0 must be $\mathtt{f}$. Γ_0 is a singleton set or empty, so that $|S_{i,0}|$ is at most MK_N. Next, for each $(\Gamma_0, \tau_1 \to \cdots \to \tau_J \to \theta'_0) \in S_{i0}$ with $\tau_j = \bigwedge_{k\in I_j}(\theta_{j,k}, m_{j,k})$, we compute

$$\begin{aligned} S_{j,k} := \{\Gamma_{j,k} \mid \Gamma_{j,k} \uparrow m_{j,k} \vdash F'_j\widetilde{x}_j : \theta_{j,k} \\ \text{and all flags in } \Gamma_{j,k} \text{ are } \mathtt{f}\}. \end{aligned}$$

The number of candidates for the type of F'_j is at most K_N, so that $|S_{j,k}|$ is at most MK_n for each j,k. Note also that since the order of the kind of $\theta_{j,k}$ is at most $N-1$, $|I_j|$ is bounded by MK_{N-1}. By choosing one element $\Gamma_{j,k}$ from each of the sets $S_{j,k}$, we can derive a judgement $\Gamma_0 \cup (\bigcup_{j,k} \Gamma_{j,k}) \vdash c(F'_1\widetilde{x}_1)\cdots(F'_J\widetilde{x}_J) : \theta'_0$. S_i is the set of all pairs (Γ,θ) such that $\Gamma \vdash \lambda\widetilde{x}.c(F'_1\widetilde{x}_1)\cdots(F'_J\widetilde{x}_J) : \theta$ is obtained by applying T-ABS to $\Gamma_0 \cup (\bigcup_{j,k} \Gamma_{j,k}) \vdash c(F'_1\widetilde{x}_1)\cdots(F'_J\widetilde{x}_J) : \theta'_0$. The number of elements of S_i generated from each element of $S_{i,0}$ is at most $K_N \times \Pi_{j,k}|S_{j,k}|$, which is bounded by $K_N(MK_N)^{AMK_{N-1}}$. Thus, the size of S_i is bounded by

$$(MK_N)\times(K_N(MK_N)^{AMK_{N-1}}) = \mathbf{exp}_N(O((AQM)^{1+\epsilon}))$$

for $N \geq 2$.

Since the size of each type environment in S_i is at most $1 + |I_1| + \cdots + |I_J| \leq 1 + AMK_{N-1}$, both the set $V_\forall \cup V_\exists$ of vertices and the set E of edges have size $P \times \mathbf{exp}_N(O((AQM)^{1+\epsilon}))$.

In Step 2, we can use Jurdziński's algorithm [10] for solving parity games. The time complexity for Step 2 is

$$O(|V_\forall \cup V_\exists||E|^{\lfloor M/2\rfloor}) = O(P^{1+\lfloor M/2\rfloor}\mathbf{exp}_N((AQM)^{1+\epsilon})).$$

Thus, the time complexity of our algorithm is

$$O(P^{1+\lfloor M/2\rfloor}\mathbf{exp}_N((AQM)^{1+\epsilon})).$$

for $N \geq 2$. If N, A, Q, and M are bounded by constants, then the algorithm runs in time $O(P^{1+\lfloor M/2\rfloor})$. Since P is bounded by the size of the recursion scheme, the time complexity is polynomial in the size of the recursion scheme, under the assumption that the largest arity A is bounded above by a constant.

VI. Related Work

As summarized in Section I, studies of model checking recursion schemes were initiated by Knapik et al. [11], [12], who showed the decidability of the MSO theory for *safe* recursion schemes. Their verification algorithm is based on a reduction of the model-checking of an order-n recursion scheme to that of a recursion scheme of order $n-1$. They [12] also showed the equi-expressivity of safe recursion schemes and higher-order pushdown automata. Cachat and Walukiewicz [2], [3] showed n-EXPTIME completeness of the modal μ-calculus model checking problem over the configuration graph of higher-order pushdown automata. For the full higher-order recursion schemes (without the safety restriction), there are two previous proofs of the decidability of the modal μ-calculus model checking. One is Ong's original proof [18], and the other is due to Hague et al. [7]. The former reduces the model checking problem to parity games over *variable profiles*, while the latter reduces it to a parity game over the configuration graph of a collapsible pushdown automaton. Both proofs use game semantics, and are probably rather hard to understand (at least for readers unfamiliar with game semantics).

Our type-based approach is a generalization of Kobayashi's type system [14]; when priorities are restricted to 0, our type system coincides with his system. Our type system is also inspired by Ong's *variable profiles* [18]. In fact, variable bindings (in type environments) in our type system are similar to Ong's variable profiles: both are assertions for variables about the state being simulated and the largest priority encountered for a relevant part of the computation, and both are defined by recursion over the kind in question. Nevertheless, the details of their constructions are dissimilar, and they give rise to radically different correctness arguments.

In addition to the advantages discussed in Section I, a general advantage of the type-based approach is that, when the verification succeeds, it is easy to understand why the recursion scheme satisfies the property, by looking at the type of each non-terminal (and the winning strategy).

Naik and Palsberg [17], [16] constructed an intersection type system that is equivalent to model checking of an imperative language and an interrupt calculus. They consider only the reachability problem, and do not treat higher-order languages. Kobayashi [14] showed that the model checking of temporal properties of higher-order programs can be (rather straightforwardly) reduced to that of higher-order recursion schemes. Thus, combined with Kobayashi's reduction, our type system can be regarded as an extension of Naik and Palsberg's scenario to the full modal μ-calculus and higher-order programs. Type systems for tree-manipulating programs have been studied in the context of programming languages for XML processing [8]. Programming languages for XML processing are concerned with *finite* trees, while our type system deals with infinite trees; that is why we need the notion of priorities and parity games for typing recursion.

VII. Conclusion

We have presented a novel type system that is equivalent to the modal μ-calculus model checking of higher-order recursion schemes. Compared to existing approaches [18], [7], our type-based method gives a simpler algorithm, and its correctness proof seems easier to understand. Furthermore, our approach yields a polynomial-time algorithm, assuming that the automaton and the largest order and arity of non-terminals of the recursion scheme are fixed. From a type-theoretic point of view, our type system introduces a novel approach to typing recursion, via parity games.

Future work includes: (i) implementation of a model checker, (ii) studies of the complexity of the model-checking problem for various restricted fragments of the modal μ-calculus, and (iii) extensions of the type system for various extensions of recursion schemes. Our type-based approach seems indeed convenient for these tasks. For (i), we have already implemented a prototype type-based model checker for a subclass of the modal μ-calculus [13]. For (ii), the reader is referred to [15]. For (iii), for instance, one can easily extend rewriting rules of recursion schemes with boolean parameters, and conditionals on them. For example, F defined by the rewrite rule $F\ b\ x\ y \mapsto \mathbf{if}\ b\ \mathbf{then}\ x\ \mathbf{else}\ y$ would be given an intersection type $(\mathsf{true} \to (q_0, \Omega(q_0)) \to \top \to q_0) \wedge (\mathsf{false} \to \top \to (q_1, \Omega(q_1)) \to q_1)$.

Acknowledgment

This work was partially supported by Kakenhi 20240001 and EPSRC EP/F036361.

References

[1] K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(3), 2007.

[2] T. Cachat. Higher order pushdown automata, the caucal hierarchy of graphs and parity games. In *Proceedings of ICALP 2003*, volume 2719 of *LNCS*, pages 556–569. Springer-Verlag, 2003.

[3] T. Cachat and I. Walukiewicz. The complexity of games on higher order pushdown automata. *CoRR*, abs/0705.0262, 2007.

[4] B. Courcelle. The monadic second-order logic of graphs IX: machines and their behaviours. *Theoretical Computer Science*, 151:125–162, 1995.

[5] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FOCS 1991*, pages 368–377, 1991.

[6] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*. Springer-Verlag, 2002.

[7] M. Hague, A. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *Proceedings of 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 452–461. IEEE Computer Society, 2008.

[8] H. Hosoya, J. Vouillon, and B. C. Pierce. Regular expression types for XML. *ACM Trans. Program. Lang. Syst.*, 27(1):46–90, 2005.

[9] 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*, 163:285–408, 2000.

[10] M. Jurdziński. Small progress measures for solving parity games. In *Proc. STACS*, volume 1770 of *LNCS*, pages 290–301. Springer-Verlag, 2000.

[11] T. Knapik, D. Niwiński, and P. Urzyczyn. Deciding monadic theories of hyperalgebraic trees. In *TLCA 2001*, volume 2044 of *LNCS*, pages 253–267. Springer-Verlag, 2001.

[12] T. Knapik, D. Niwiński, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS 2002*, volume 2303 of *LNCS*, pages 205–222. Springer-Verlag, 2002.

[13] N. Kobayashi. Model-checking higher-order functions. Unpublished, 2009.

[14] N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proc. of POPL*, 2009.

[15] N. Kobayashi and C.-H. L. Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. In *Proceedings of ICALP 2009*, LNCS. Springer-Verlag, 2009.

[16] M. Naik. A type system equivalent to a model checker. Master Thesis, Purdue University.

[17] M. Naik and J. Palsberg. A type system equivalent to a model checker. In *ESOP 2005*, volume 3444 of *LNCS*, pages 374–388. Springer-Verlag, 2005.

[18] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS 2006*, pages 81–90. IEEE Computer Society Press, 2006.

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