A TYPE SYSTEM EQUIVALENT TO THE MODAL MU-CALCULUS MODEL CHECKING OF HIGHER-ORDER RECURSION SCHEMES

NAOKI KOBAYASHI AND C.-H. LUKE ONG

Graduate School of Information Science and Technology, The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-0033 Japan
e-mail address: koba@is.s.u-tokyo.ac.jp

Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford OX1 3QD, United Kingdom
e-mail address: Luke.Ong@cs.ox.ac.uk

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 μ-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.

1. 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. [14] 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 [15], subsuming earlier well-known MSO decidability results for regular (or order-0) trees [30] and algebraic (or order-1) trees [7]. (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 [27] has subsequently shown that the modal μ-calculus model checking problem for trees generated by arbitrary order-n recursion schemes is

Received by the editors March 14, 2013.
A preliminary summary of this article appeared in Proceedings of 24th Annual IEEE Symposium on Logic in Computer Science (LICS 2009), pp.179-188, 2009.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.2168/LMCS-???

n-EXPTIME complete (and hence these trees have decidable MSO theories); further these schemes are equi-expressive with a new class of automata, called *collapsible* pushdown automata [10]. On the practical side, Kobayashi [17] 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. Following his work, a number of program verification methods based on the model checking of recursion schemes have been proposed, and automated verification tools for functional programs have been implemented [21, 22, 23, 28, 36].

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 [12]. Hague et al. [10] 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 [17] showed that given a Büchi tree automaton with a trivial acceptance condition (the 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 [17], 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: See Section 6.) 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 [17] 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. Based on the type system, Kobayashi [16, 18] has also constructed practical model checking algorithms, which work reasonably well for typical inputs despite the extremely high worst-case complexity.

This paper builds on Kobayashi's type system [17] and extends it to a type system capable of the modal μ-calculus model checking of trees generated by higher-order recursion schemes. Equivalently (thanks to Emerson and Jutla [8]), 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 μ-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 [27]. 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.[1]

- It is much easier to discuss the parametrized complexity and possible optimization of the model-checking 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 [27] runs in time n-fold exponential in the size of the scheme, under the same assumption. Furthermore, almost all

[1]Since the first publication of our proof [19], Salvati and Walukiewicz [32] provided another proof that does not rely on game semantics.

the known practical model-checking algorithms [16, 18, 23, 26] (albeit for subclasses of the modal μ-calculus) are based on the type-based approach, although some of them [18, 26] also use game-semantics. The only exception is Broadbent et al.'s saturation-based algorithm for model checking of collapsible pushdown systems [2].

• 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. In fact, Kobayashi's type system for trivial automaton model checking of higher-order recursion schemes has been extended to deal with finite data domains (such as booleans) [22, 26] and untyped higher-order recursion schemes [37].

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 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 2 gives preliminary definitions. Section 3 defines the type system equivalent to the model checking of recursion schemes, and Section 4 proves its correctness. Section 5 discusses the type inference algorithm (which serves as a model-checking algorithm for recursion schemes) and its complexity. Section 6 discusses related work and Section 7 concludes.

A preliminary summary of this article appeared in Proceedings of LICS 2009 [19]. The main new contributions compared with the preliminary version are:

- Simplification of the type system: We have removed the *flags* used in the earlier type system [19], and simplified the type system accordingly.
- More detailed proofs.
- More extensive discussion of related work, including those since 2009.

2. Preliminaries

This section reviews basic definitions used throughout the paper. We first review the definition of higher-order recursion schemes [15, 27] in Section 2.1. We then review the definition of alternating parity tree automata [9] in Section 2.2. 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. Finally, we review the definition of parity games [9] in Section 2.3. Parity games are often used in the context of modal μ-calculus model checking; in fact, Ong's algorithm [27] reduces the model checking of higher-order recursion schemes to the solvability of a parity game. We shall use it for defining the type system (more specifically, for typing recursion schemes).

We write $\{x_1 \mapsto v_1, \dots, x_n \mapsto v_n\}$ or $\{x_1 \!:\! v_1, \dots, x_n \!:\! v_n\}$[2] for the map f such that $dom(f) = \{x_1, \dots, x_n\}$ and $f(x_i) = v_i$ for $i \in \{1, \dots, n\}$. For a map f, we write $dom(f)$ and $codom(f)$ for the domain and co-domain of f respectively. We write $f\{x \mapsto v\}$ for the map f' such that $dom(f') = dom(f) \cup \{x\}$, $f'(x) = v$, and $f'(y) = f(y)$ for $y \in dom(f) \setminus \{x\}$.

2.1. **Higher-Order Recursion Schemes.**

A higher-order recursion scheme is a grammar for describing an infinite tree.

[2] We prefer the latter notation when v_i contains a symbol similar to $\mapsto$, like $\to$.

The set of *sorts*[3] is defined by:

$$\kappa ::= \mathsf{o} \mid \kappa_1 \to \kappa_2$$

Intuitively, o describes trees, while $\kappa_1 \to \kappa_2$ describes a function that takes an entity of sort κ_1 and returns an entity of sort κ_2. The *order* and *arity* of κ, written $ord(\kappa)$ and $arity(\kappa)$ respectively, are defined by:

$$ord(\mathsf{o}) := 0 \quad ord(\kappa_1 \to \kappa_2) := max(ord(\kappa_1) + 1, ord(\kappa_2))$$
$$arity(\mathsf{o}) := 0 \qquad arity(\kappa_1 \to \kappa_2) := arity(\kappa_2) + 1$$

A *ranked alphabet* Σ is a map from a finite set of symbols to sorts of order 0 or 1. The *sort judgment* is of the form $\mathcal{K}; \Sigma \vdash t : \kappa$, where $\mathcal{K}$ is a map from a finite set of variables to sorts, t is a λ-term (that may contain elements of $dom(\Sigma$ as constants), and κ is a sort. The relation is inductively defined by the following rules.

$$\frac{\mathcal{K}(x) = \kappa}{\mathcal{K}; \Sigma \vdash x : \kappa}$$

$$\frac{\Sigma(a) = \kappa}{\mathcal{K}; \Sigma \vdash a : \kappa}$$

$$\frac{\mathcal{K}; \Sigma \vdash t_1 : \kappa_2 \to \kappa \qquad \mathcal{K}; \Sigma \vdash t_2 : \kappa_2}{\mathcal{K}; \Sigma \vdash t_1 t_2 : \kappa}$$

$$\frac{\mathcal{K}\{x \mapsto \kappa_1\}; \Sigma \vdash t : \kappa_2}{\mathcal{K}; \Sigma \vdash \lambda x.t : \kappa_1 \to \kappa_2}$$

When $\mathcal{K}$ and Σ are fixed, there is at most one κ such that $\mathcal{K}; \Sigma \vdash t : \kappa$. We often call κ "the sort of t" and say "t has sort κ". A λ-term t that does not contain λ-abstractions is called an *applicative term*. We often write $\widetilde{\cdot}$ to indicate a (possibly empty) sequence. For example, $\lambda\widetilde{x}.t$ means $\lambda x_1. \cdots \lambda x_m.t$ (where m can be 0). We write $|\widetilde{s}|$ for the length of the sequence $\widetilde{s}$.

Definition 2.1. 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*. The elements of Σ are called *terminals*.
- $\mathcal{N}$ is a map from a finite set of symbols called *non-terminals* to sorts.
- $\mathcal{R}$ is a map[4] from the set of non-terminals (i.e. $dom(\mathcal{N})$) to λ-terms of the form $\lambda\widetilde{x}.t$, where t is an applicative term. We require that (i) $\mathcal{N}; \Sigma \vdash \mathcal{R}(F) : \mathcal{N}(F)$, and (ii) if $\mathcal{R}(F) = \lambda\widetilde{x}.t$ and t is an applicative term, then t must have sort o.
- S is a special non-terminal called the *start symbol*, such that $\mathcal{N}(S) = \mathsf{o}$.

The *order* of a non-terminal F is $ord(\mathcal{N}(F))$. 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 $F \in dom(\mathcal{N})$.

Next, we define the tree generated by a recursion scheme. The rewriting relation $\longrightarrow_{\mathcal{G}}$ is defined inductively by:

- $F\,\widetilde{s} \longrightarrow_{\mathcal{G}} [\widetilde{s}/\widetilde{x}]t$ if $\mathcal{R}(F) = \lambda\widetilde{x}.t$.

[3]It is usually called a *type* [27]. We use the term "sorts" to avoid confusion with the intersection types introduced later.

[4]Thus we assume that there is exactly one rewriting rule for each non-terminal symbol, i.e., that recursion schemes are *deterministic* in this paper.

- If $t \longrightarrow_{\mathcal{G}} t'$, then $ts \longrightarrow_{\mathcal{G}} t's$ and $st \longrightarrow_{\mathcal{G}} st'$.

Here, $[\widetilde{s}/\widetilde{x}]t$ denotes the term obtained from t by replacing variables in $\widetilde{x}$ with the corresponding terms in $\widetilde{t}$ (with the assumption that $|\widetilde{s}| = |\widetilde{x}|$). We omit the subscript $\mathcal{G}$ whenever it is clear from the context.

Let $\mathcal{L}$ be a set of symbols. A *$\mathcal{L}$-labelled tree* is just a partial function T from $\{1,\ldots,n\}^*$ (for some fixed $n \geq 1$) to $\mathcal{L}$ such that $\pi i \in dom(t)$ implies $\{\pi\} \cup \{\pi j \mid j \in 1 \leq j \leq i\} \subseteq dom(t)$. 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 ranked tree* is thus a $dom(\Sigma)$-labelled tree such that 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 $dom(T)$.

We often use the usual term representation for trees. For example, we write $\mathtt{a}\ \mathtt{c}\ (\mathtt{b}\ \mathtt{c})$ for the tree:

$$\{\epsilon \mapsto \mathtt{a}, 1 \mapsto \mathtt{c}, 2 \mapsto \mathtt{b}, 2\,1 \mapsto \mathtt{c}\}.$$

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 $\forall w \in dom(t).(w \in dom(s) \wedge t(w) \sqsubseteq s(w))$. For example, $\perp \sqsubseteq f\ \perp\ \perp \sqsubseteq f\ \perp\ b \sqsubseteq f\ a\ b$. For a directed set $\mathcal{T}$ of trees, we write $\bigsqcup T$ for the least upper bound of elements of $\mathcal{T}$ with respect to $\sqsubseteq$.

We are now ready to define the tree generated by a recursion scheme.

Definition 2.2 (value trees). Let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ be a higher-order recursion scheme. The tree generated by $\mathcal{G}$, or the *value tree* of $\mathcal{G}$, written $[\![\mathcal{G}]\!]$, is defined by:

$$[\![\mathcal{G}]\!] \ := \ \bigsqcup\{t^\perp \mid S \longrightarrow^*_{\mathcal{G}} t\}.$$

Note that $[\![\mathcal{G}]\!]$ is well-defined because $\longrightarrow_{\mathcal{G}}$ is confluent and $t \longrightarrow^*_{\mathcal{G}} u$ implies $t^\perp \sqsubseteq u^\perp$. By construction, $[\![\mathcal{G}]\!]$ is a possibly infinite, ranked $(\Sigma\{\perp \mapsto \mathtt{o}\})$-labelled tree (but see Remark 2.1).

Example 2.1. Consider the order-1 recursion scheme $\mathcal{G}_0 = (\Sigma, \mathcal{N}, \mathcal{R}, S)$, where:

$$\begin{aligned} &\Sigma = \{\mathtt{a} : \mathtt{o} \to \mathtt{o} \to \mathtt{o}, \mathtt{b} : \mathtt{o} \to \mathtt{o}, \mathtt{c} : \mathtt{o}\} \\ &\mathcal{N} = \{S : \mathtt{o}, F : \mathtt{o} \to \mathtt{o}\} \\ &\mathcal{R} = \{S \mapsto F\ \mathtt{c}, \quad F \mapsto \lambda x.\mathtt{a}\ x\ (F(\mathtt{b}\ x))\} \end{aligned}$$

S is reduced as follows.

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

The value tree $[\![\mathcal{G}_0]\!]$ is shown on the left-hand side of Figure 1.

Example 2.2. Consider the order-2 recursion scheme $\mathcal{G}_1 = (\Sigma, \mathcal{N}_1, \mathcal{R}_1, S)$, where:

$$\begin{aligned} &\Sigma = \{\mathtt{a} : \mathtt{o} \to \mathtt{o} \to \mathtt{o}, \mathtt{b} : \mathtt{o} \to \mathtt{o}, \mathtt{c} : \mathtt{o}\} \\ &\mathcal{N}_1 = \{S : \mathtt{o}, F : (\mathtt{o} \to \mathtt{o}) \to \mathtt{o}, D : (\mathtt{o} \to \mathtt{o}) \to \mathtt{o} \to \mathtt{o}\} \\ &\mathcal{R}_1 = \{S \mapsto F\ \mathtt{b}, \quad F \mapsto \lambda f.\mathtt{a}\ (f\ \mathtt{c})\ (F(D\ f)), \quad D \mapsto \lambda f.\lambda x.f(f\ x)\} \end{aligned}$$

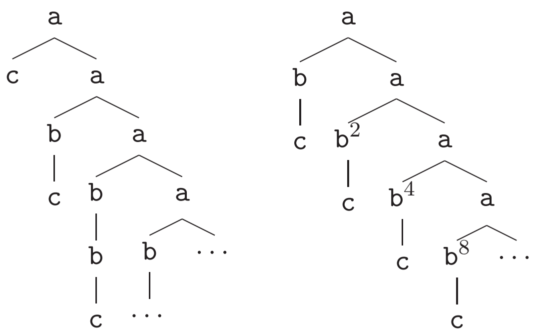

Figure 1: The value trees $[\![\mathcal{G}_0]\!]$ (on the left-hand side) and $[\![\mathcal{G}_1]\!]$ (on the right-hand side).

The value tree $[\![\mathcal{G}_1]\!]$ is shown on the right-hand side of Figure 1,x where $\mathtt{b}^k$ denotes k repeated occurrences of $\mathtt{b}$.

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

$$\mathsf{B}^+(X) \ni \psi \ ::= \ \mathsf{true} \mid \mathsf{false} \mid x \mid \psi \wedge \psi \mid \psi \vee \psi$$

where x ranges over X. We say that a subset Y of X *satisfies* ψ just if assigning true to elements in Y and false to elements in $X \setminus Y$ makes ψ true. For example, $\{(1, q_0), (2, q_1)\}$ satisfies the formula $((1, q_0) \vee (2, q_0)) \wedge (2, q_1)$, since $(\mathsf{true} \vee \mathsf{false}) \wedge \mathsf{true}$ is equivalent to true.

Definition 2.3 (alternating parity tree automata)**.** 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 $(\mathit{dom}(T) \times Q)$-labelled unranked tree R satisfying:

- $\epsilon \in \mathit{dom}(R)$ and $R(\epsilon) = (\epsilon, q_I)$; and
- for every $\beta \in \mathit{dom}(R)$ with $R(\beta) = (\alpha, q)$, the set $\{(i, q') \mid \exists j.R(\beta j) = (\alpha i, q')\}$ satisfies $\delta(q, T(\alpha))$.

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. Finally, an alternating parity tree automaton $\mathcal{A}$ *accepts* a Σ-labeled ranked tree T if there is an accepting run-tree of $\mathcal{A}$ over T.

We use alternating parity tree automata for describing properties of (the value tree of) recursion schemes, instead of modal μ-calculus formulas.

Ong [27] 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 $\mathcal{G}$.

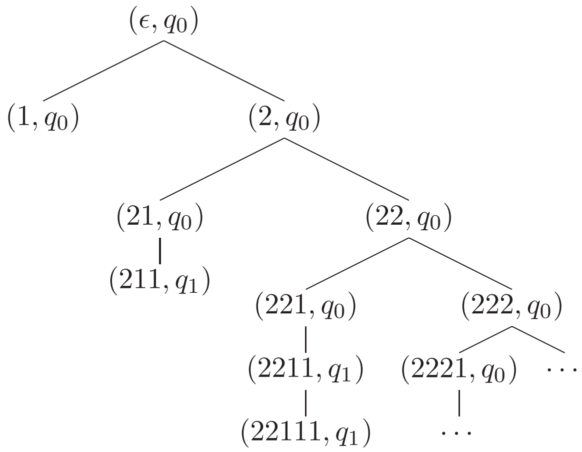

Figure 2: An accepting run-tree of $\mathcal{A}_1$ over $[\![\mathcal{G}_0]\!]$.

Theorem 2.1 (Ong [27])**.** 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 $\mathcal{G}$. Let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ be an arbitrary recursion scheme and $\mathcal{A} = (\Sigma\{\perp \mapsto 0\}, Q, \delta, q_I, \Omega)$ be an alternating parity tree automaton. Define $\mathcal{G}' = (\Sigma', \mathcal{N}, \mathcal{R}', S)$ and $\mathcal{A}' = (\Sigma', Q', \delta', q_I, \Omega')$ by:

$$
\begin{aligned}
&\Sigma' = \Sigma\{\mathtt{loop} \mapsto 1\}\\
&\mathcal{R}' = \{F \mapsto \lambda\widetilde{x}.\mathtt{loop}(t) \mid \mathcal{R}(F) = \lambda\widetilde{x}.t\}\\
&Q' = Q \cup \{q_i' \mid q_i \in Q\}\\
&\delta'(q, a) = \begin{cases} \delta(q, a) & \text{if } q \in Q \text{ and } a \in \mathit{dom}(\Sigma)\\ \delta(q_i, a) & \text{if } q = q_i' \text{ with } q_i \in Q \text{ and } a \in \mathit{dom}(\Sigma)\\ (1, q') & \text{if } q \in Q \text{ and } a = \mathtt{loop}\\ (1, q) & \text{if } q = q_i' \text{ with } q_i \in Q \text{ and } a = \mathtt{loop} \end{cases}\\
&\Omega'(q) = \begin{cases} 2 + \Omega(q) & \text{if } q \in Q\\ 0 & \text{if } q = q_i' \text{ with } q_i \in Q \text{ and } \delta(q, \perp) = \mathsf{true}\\ 1 & \text{if } q = q_i' \text{ with } q_i \in Q \text{ and } \delta(q, \perp) = \mathsf{false} \end{cases}
\end{aligned}
$$

Then $\mathcal{G}'$ and $\mathcal{A}'$ satisfy the conditions (i) and (ii) above.

Example 2.3. 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 $q \in \{q_0, q_1\}$,

$$
\begin{aligned}
&\delta_1(q, \mathtt{a}) = (1, q) \wedge (2, q) \quad \delta_1(q, \mathtt{b}) = (1, q_1)\\
&\delta_1(q, \mathtt{c}) = \mathtt{true}
\end{aligned}
$$

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. Figure 2 shows an accepting run-tree of $\mathcal{A}_1$ over the tree $[\![\mathcal{G}_0]\!]$ in Figure 1. Note that it has the only one infinite path labelled by $(\epsilon, q_0)(2, q_0)(22, q_0)(222, q_0)\cdots$, which satisfies the parity condition.

$$\begin{array}{c} (\epsilon, q_0) \\ | \\ (2, q_0) \\ | \\ (22, q_0) \\ | \\ (222, q_0) \\ | \\ \cdots \end{array}$$

Figure 3: An accepting run-tree of $\mathcal{A}_3$ over $[\![\mathcal{G}_0]\!]$.

Example 2.4. Let Σ be the same alphabet as above. Let $\mathcal{A}_2$ be the alternating parity tree automaton $(\Sigma, \{q_0, q_1\}, \delta_2, q_0, \Omega_2)$, where

$$\begin{array}{ll} \delta_2(q, \mathtt{a}) = (1, q_1) \wedge (2, q) \text{ for each } q \in \{q_0, q_1\} & \\ \delta_2(q, \mathtt{b}) = (1, q) \text{ for each } q \in \{q_0, q_1\} & \delta_2(q, \mathtt{c}) = \mathsf{true} \\ \Omega_2(q_0) = 2 \qquad \Omega_2(q_1) = 1 & \end{array}$$

$\mathcal{A}_2$ accepts a Σ-tree t if, and only if, for every path of t, if the path takes the left branch of a node labeled by $\mathtt{a}$, then the path is finite, ending with $\mathtt{c}$.

Example 2.5. The shape of a run-tree can be different from that of a Σ-labelled tree. Let Σ be the same alphabet as above, and let $\mathcal{A}_3$ be the alternating parity tree automaton $(\Sigma, \{q_0\}, \delta_3, q_0, \Omega_3)$, where

$$\delta_3(q_0, \mathtt{a}) = (2, q_0) \qquad \delta_3(q, \mathtt{b}) = \delta_3(q, \mathtt{c}) = \mathsf{false} \qquad \Omega_3(q_0) = 0.$$

$\mathcal{A}_3$ accepts a Σ-labelled tree T if and only if its rightmost path is labelled by $\mathtt{a}^\omega$. Figure 3 shows an accepting run-tree of $\mathcal{A}_3$ over the tree $[\![\mathcal{G}_0]\!]$ in Figure 1.

2.3. **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, i.e. either π is infinite, or ends in a node v such that $\neg\exists v'.(v, v') \in E$. 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; formally, for all plays π_1 and π_2 that conform with $\mathcal{W}$, if their respective last nodes are the same $V_\exists$-node, then $\mathcal{W}(\pi_1) = \mathcal{W}(\pi_2)$. We say that a parity game is *solvable* just if there is a winning strategy (for player $\exists$). 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 [8].

3. Type system

Following Kobayashi's type system [17] for trivial-automaton model checking of recursion schemes, for a given APT $\mathcal{A}$, 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 $\mathcal{A}$. Henceforth we fix an APT $\mathcal{A} = (Q, \Sigma, \delta, q_I, \Omega)$.

3.1. **Types.** We first introduce intersection types to capture the shape of trees and tree contexts represented by terms. Following Bakel [39], we use the normalized form of intersection types, where intersection type constructors occur only in the lefthand side of function types.

Definition 3.1. Let q and m respectively range over the states and priorities of $\mathcal{A}$. The sets of *atomic types* and *types* are given by:

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

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 $\bigwedge\{(\theta_1, m_1), \dots, (\theta_k, m_k)\}$. 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 $\Omega(\tau \to \theta) := \Omega(\theta)$.

Intuitively, the type q describes a tree that can be accepted from q by $\mathcal{A}$, i.e., accepted by $\mathcal{A}' = (Q, \Sigma, \delta, q, \Omega)$. In other words, q describes a tree that has a run-tree whose root is labeled by q.

The type $(q_1, m_1) \wedge \cdots \wedge (q_k, m_k) \to q$ describes a function (or more accurately a tree context, as a tree function that can be expressed by higher-order recursion schemes cannot inspect its arguments) that takes a tree that can be accepted from each of the states $q_1, \dots, q_k$, and returns a tree that is accepted from state q. To understand the meaning of the priorities m_i in the type, we need to associate (higher-order) tree contexts with corresponding contexts on run-trees. For example, if the arity of $\mathtt{a}$ is 1 and $\delta(q_0, \mathtt{a}) = (1, q_0) \wedge (1, q_1)$, then the tree context on the left-hand side of Figure 4 (which takes a tree T and returns the tree $\mathtt{a}\,T$ by filling the hole H with T) has the context on the right-hand side as an associated context on run-trees. In fact, if a tree T (to fill the hole H) has two run-trees R_0 and R_1 whose roots are labelled by q_0 and q_1 respectively, then a run-tree for $\mathtt{a}\,T$ is obtained by filling the holes H_0 and H_1 with R_0 and R_1. In this manner, one can associate a (higher-order) context on trees with a (higher-order) context on run-trees. The type of a term then describes the shape of a context on *run-trees* associated with the tree context represented by the term. Henceforth, in illustrations of run-tree contexts, we omit the first component of a label and specify only the second component (which is a state of the automaton). In the type $(q_1, m_1) \wedge \cdots \wedge (q_k, m_k) \to q$, the priority m_i describes the largest priority in the path from the root of the run-tree to the hole of type q_i. Figure 5 illustrates a tree context of type $(q_1, m_1) \wedge (q_2, m_2) \to q$. For example, in Figure 4, if $\Omega(q_0) = 0$ and $\Omega(q_1) = 1$, the run-tree (and the corresponding tree context on the left-hand side) has type $(q_0, 0) \wedge (q_1, 1) \to q_0$.

More generally, $(\theta_1, m_1) \wedge \cdots \wedge (\theta_k, m_k) \to \theta$ describes a higher-order context on trees that has an associated run-tree context with holes of shapes $\theta_1, \dots, \theta_k$ where the largest priority from the root to each hole of type θ_i is m_i. When discussing the intuitions behind types, we do not explicitly

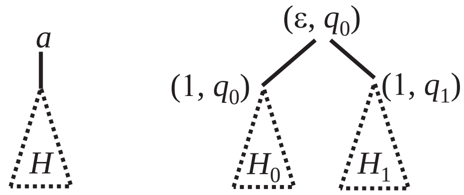

Figure 4: A context on trees and an associated context on run-trees (where only states are shown as labels). Dashed triangles represent holes to be filled with trees. H_0 and H_1 are holes on run-trees that correspond to the hole H on trees.

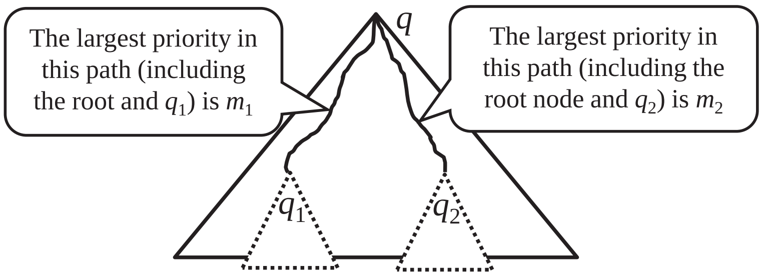

Figure 5: A context on run-trees described by $(q_1, m_1) \wedge (q_2, m_2) \to q$

distinguish between a tree and an associated run-tree, but note that when we speak of "the largest priority in a path", we mean a path in a run-tree.

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 sort κ" and "θ is an atomic type of sort κ" respectively. We also impose a condition on priorities.

Definition 3.2 (Well-formed types)**.** The relations $\tau :: \kappa$ and $\theta ::_a \kappa$ are the least relations closed under the following rules:

$$\frac{}{q_i ::_a \mathsf{o}} \qquad \frac{\tau :: \kappa_1 \qquad \theta ::_a \kappa_2}{\tau \to \theta \quad ::_a \quad \kappa_1 \to \kappa_2} \qquad \frac{\theta_i ::_a \kappa \quad \text{ for each } i \in \{1, \dots, n\}}{\bigwedge\{(\theta_1, m_1), \dots, (\theta_n, m_n)\} \quad :: \quad \kappa}$$

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 $\bigwedge_{i=1}^{k} (\theta_i, m_i) \to \theta'$, we have $m_i \geq \max(\Omega(\theta'), \Omega(\theta_i))$ for each $1 \leq i \leq k$.

For example, $q_1 \wedge ((q_2, 1) \to q_3)$ is not well-formed, as it combines types of different sorts. $(q_1, m_1) \wedge (q_2, m_2) \to q$ is well-formed if $m_1 \geq \max(\Omega(q), \Omega(q_1))$ and $m_2 \geq \max(\Omega(q), \Omega(q_2))$; this reflects the intuition that m_1 and m_2 are the largest priorities in the paths shown in Figure 5, *including the start and end nodes*. Henceforth we consider only well-formed types.

3.2. **Typing for Terms.** Henceforth we treat non-terminals as variables. A *type judgement* has the form $\Gamma \vdash_{\mathcal{A}} t : \theta$ where t is a λ-term, and Γ, called a *type environment*, is a set of bindings of the form $x : (\theta, m)$. The intuition is that if $\Gamma \vdash_{\mathcal{A}} t : q$ and $x : (\theta, m) \in \Gamma$, then x can be used as the head term in a position where m is the largest priority seen in the path (of the run-tree) from the root of the tree generated by t. Note that Γ may contain different bindings of the same variable.

In the following we shall omit the subscript $\mathcal{A}$ from $\vdash_{\mathcal{A}}$ whenever it is clear from the context.

Example 3.1. Suppose the priority $\Omega(q_i)$ of q_i is i for $i \in \{0,1\}$, and the transition rule for $\mathtt{a}$ is: $\delta(q_0, \mathtt{a}) = (1, q_1)$.

(i) The judgement $\{x : (q_1, 1)\} \vdash a\,x : q_0$ is valid. The type environment stipulates that x can only be used at a node such that the largest priority in the path from the root is 1; the path from the root of $a\,x$ to x is labelled by $q_0 q_1$, which has largest priority 1.

(ii) The judgement $\{x : (q_0, 1)\} \vdash x : q_0$ is invalid, because the path from the root to x is q_0, which has largest priority 0.

(iii) The judgement $\{x : (q_0, 1), y : ((q_0, 1) \to q_0, 0)\} \vdash y\ x : q_0$ is valid, because y uses the argument x only in a node such that the path from the root has largest priority 1.

Notations. We shall often drop the set braces to save writing. We write $\Gamma, x : \bigwedge_{i=1}^{k} (\theta_i, m_i)$ as a shorthand for

$$\Gamma \cup \{x : (\theta_1, m_1), \ldots, x : (\theta_k, m_k)\}$$

where x is assumed *not* to occur in Γ. We write $dom(\Gamma)$ for the set $\{x \mid \exists \theta, m \,.\, x : (\theta, m) \in \Gamma\}$.

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

$$\frac{}{x : (\theta, \Omega(\theta)) \vdash_{\mathcal{A}} x : \theta} \quad \text{(T-VAR)}$$

$$\frac{\{(i, q_{ij}) \mid i \in \{1, \ldots, n\}, j \in J_i\} \text{ satisfies } \delta_{\mathcal{A}}(q, a) \qquad m_{ij} = \max(\Omega(q_{ij}), \Omega(q)) \text{ for each } i \in \{1, \ldots, n\}, j \in J_i}{\emptyset \vdash_{\mathcal{A}} a : \bigwedge_{j \in J_1} (q_{1j}, m_{1j}) \to \cdots \to \bigwedge_{j \in J_n} (q_{nj}, m_{nj}) \to q} \quad \text{(T-CONST)}$$

$$\frac{\Gamma_0 \vdash_{\mathcal{A}} t_0 : \bigwedge_{i \in I} (\theta_i, m_i) \to \theta \qquad \Gamma_i \vdash_{\mathcal{A}} t_1 : \theta_i \text{ for each } i \in I \qquad \forall i, j \in I.((\theta_i, m_i) = (\theta_j, m_j) \Rightarrow i = j)}{\Gamma_0 \cup \bigcup_{i \in I} (\Gamma_i \Uparrow m_i) \vdash_{\mathcal{A}} t_0\, t_1 : \theta} \quad \text{(T-APP)}$$

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

Here, $\Gamma \Uparrow m$ is defined by:

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

In (T-VAR), x is used at the root, so that the largest priority of the path from the root to x is $\Omega(\theta)$. The rule (T-CONST) is for terminal symbols. The premise means that the automaton $\mathcal{A}$ in state q, upon reading a, spawns a new automaton that reads the i-th subtree in state q_{ij}, for each $i \in \{1, \cdots, n\}$ and $j \in J_i$. The tree $a\,T_1 \cdots T_n$ and an associated run-tree are shown in Figure 6. The figure suggests we can view the constant a as a function that takes trees $T_1, \ldots, T_n$ as input such that each T_i has types $q_{i1}, \ldots, q_{ik_i}$, and returns a tree of type q. Furthermore, it uses a run-tree of type q_{ij} in the position where the path from the root is labelled by qq_{ij}. Thus, a can be considered a function of type

$$\bigwedge_{j=1}^{k_1} (q_{1j}, m_{1j}) \to \cdots \to \bigwedge_{j=1}^{k_n} (q_{nj}, m_{nj}) \to q,$$

where $m_{ij} = \max(\Omega(q_{ij}), \Omega(q))$. For example, for the automaton $\mathcal{A}_1$ in Example 2.3, a has types $(q_0, 2) \to (q_0, 2) \to q_0$ and $(q_1, 1) \to (q_1, 1) \to q_1$.

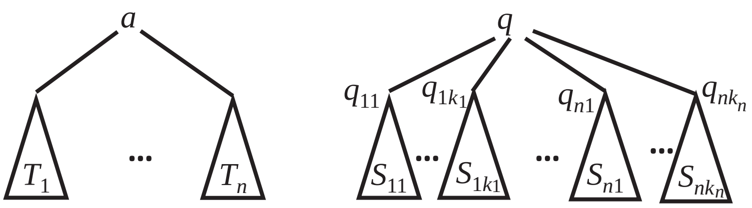

Figure 6: The tree $a\,T_1 \cdots T_n$ (on the left-hand side) and a run-tree for it. S_{ij} is a run-tree for T_i, where the root is given state q_{ij}.

In (T-APP), the first premise requires that the argument of t_0 should have types θ_i for every $i \in I$. Correspondingly the second premise requires that t_1 has these types. We explain the operations on priorities using Figure 7, which shows the case for $\Gamma_i = x : (\theta'_i, m'_i)$ (for each $i \in \{0, 1, \ldots, n\}$) and $I = \{1, \ldots, k\}$. The upper-half corresponds to the premises of the rule. The premise $\Gamma_0 \vdash t_0 : \bigwedge_{i\in I}(\theta_i, m_i) \to \theta$ means that the tree context generated by t_0 has a run-tree of the form S_0 shown on the left-hand side. $\Gamma_i \vdash t_1 : \theta_i$ means that the tree context generated by t_1 has a run-tree of the form S_i (for each $i \in I = \{1, \ldots, k\}$) shown on the right-hand side. Note that S_i has a hole of type θ'_i in a position where the largest priority in the path from the root of S_i is m'_i. By filling the hole of type θ_i in S_0 with S_i (for each $i \in \{1, \ldots, k\}$), we obtain a run-tree context for the tree context generated by $t_1 t_2$. Now, the hole x of type θ'_i occurs in a position where the largest priority in the path from the root to the hole is $max(m_i, m'_i)$. Thus, $t_1 t_2$ should be typed under $x : (\theta'_0, m'_0), x : (\theta'_1, max(m_1, m'_1)), \ldots, x : (\theta'_1, max(m_k, m'_k))$, i.e., $\Gamma_0 \cup \bigcup_{i\in I}(\Gamma_i \Uparrow m_i)$. The last premise of T-APP forbids, for example, the following derivation:

$$\frac{\Gamma_0 \vdash t_0 : (\theta_1, m_1) \to \theta \qquad \Gamma_1 \vdash t_1 : \theta_1 \qquad \Gamma_2 \vdash t_1 : \theta_1}{\Gamma_0 \cup (\Gamma_1 \Uparrow m_1) \cup (\Gamma_2 \Uparrow m_1) \vdash t_0 t_1 : \theta}$$

which combines different type environments for the same type (θ_1, m_1) of t_1. Although the restriction does not affect the soundness and completeness of the type system, it is necessary for the discussion of the complexity in Section 5.

The rule (T-ABS) for abstraction is standard, except that weakening on x is allowed. For technical convenience, this is the (only) place where weakening is introduced.

Remark 3.1. In rule (T-APP), k can be 0. Thus, for example, $x : (\top \to q, \Omega(q)) \vdash x\ t : q$ 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 2.3. By using rule (T-CONST), we obtain the following types for input symbols.

$$\begin{array}{l}
\mathtt{a} : (q, \Omega_1(q)) \to (q, \Omega_1(q)) \to q \text{ for each } q \in \{q_0, q_1\} \\
\mathtt{b} : (q_1, \Omega_1(q)) \to q \text{ for each } q \in \{q_0, q_1\} \\
\mathtt{c} : q \text{ for each } q \in \{q_0, q_1\}
\end{array}$$

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 $\Gamma_1 = F : (\theta, 2), x : (q_1, 2)$. The term $\lambda x.\mathtt{a}\, x\ (F(\mathtt{b}\, x))$ is typed as follows.

$$\dfrac{\dfrac{\emptyset \vdash \mathtt{a} : \theta_{\mathtt{a}} \quad x : (q_0, 2) \vdash x : q_0 \quad \dfrac{F : (\theta, 2) \vdash F : \theta \quad x : (q_1, 2) \vdash \mathtt{b}\, x : q_0 \quad x : (q_1, 1) \vdash \mathtt{b}\, x : q_1}{F : (\theta, 2), x : (q_1, 2) \vdash F(\mathtt{b}\, x) : q_0}}{F : (\theta, 2), x : (q_0, 2), x : (q_1, 2) \vdash \mathtt{a}\, x\ (F(\mathtt{b}\, x)) : q_0}}{F : (\theta, 2) \vdash \lambda x.\mathtt{a}\, x\ (F(\mathtt{b}\, x)) : \theta}$$

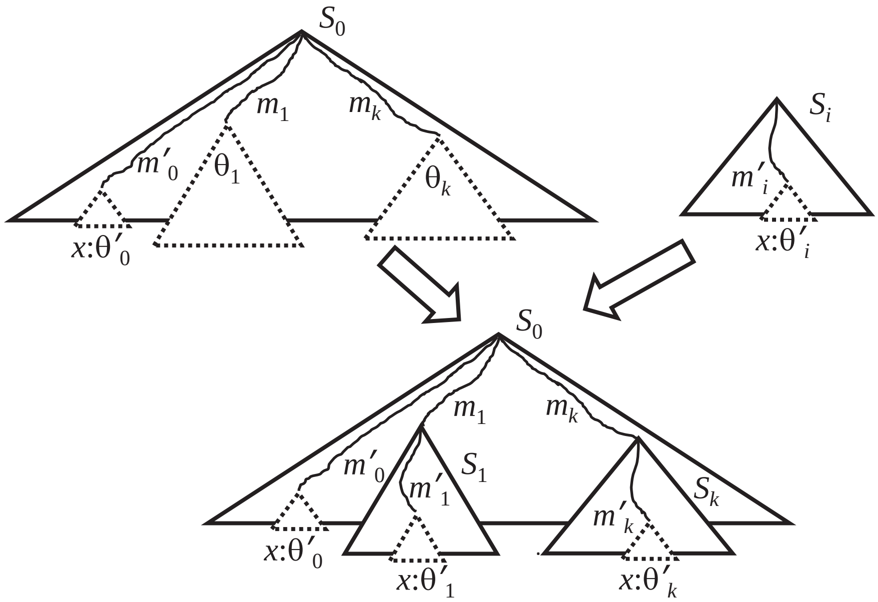

Figure 7: A run-tree context for the tree context generated by $t_0 t_1$ (shown on the lower-half). On the upper-half, S_0 is a run-tree context for the tree context generated by t_0, and $S_1, \ldots, S_n$ are run-tree contexts for the tree context generated by t_1.

Here, $x : (q_1, 2) \vdash \mathtt{b}\, x : q_0$ and $x : (q_1, 1) \vdash \mathtt{b}\, x : q_1$ are derived by:

$$\frac{\emptyset \vdash \mathtt{b} : (q_1, 2) \to q_0 \quad x : (q_1, 1) \vdash x : q_1}{(x : (q_1, 1)) \Uparrow 2 \vdash \mathtt{b}\, x : q_0}$$

and

$$\frac{\emptyset \vdash \mathtt{b} : (q_1, 1) \to q_1 \quad x : (q_1, 1) \vdash x : q_1}{(x : (q_1, 1)) \Uparrow 1 \vdash \mathtt{b}\, x : q_1}$$

Note that $(x : (q_1, 1)) \Uparrow 2 = x : (q_1, 2)$ and $(x : (q_1, 1)) \Uparrow 1 = x : (q_1, 1)$.

3.3. **Typing for recursion schemes.** We now define the typing relation $\vdash_{\mathcal{A}} \mathcal{G}$ for recursion schemes. In type systems for programming languages, a standard rule for recursion $F = t$ is:

$$\frac{\Gamma, F : \tau \vdash t : \tau}{\Gamma \vdash F : \tau} \qquad \text{(UNSOUND-REC)}$$

Kobayashi [17] used essentially the same rule for the restricted class of automata (Büchi automata with a trivial acceptance condition).

The standard rule for recursion is however insufficient for dealing with the properties described by alternating parity tree automata (or equivalently, MSO or modal μ-calculus formula). For example, let $\mathcal{A}'_1$ be the alternating parity tree automaton obtained from $\mathcal{A}_1$ of Example 2.3 by replacing the initial state replaced with q_1, and let $\mathcal{G}$ be the recursion scheme $\mathcal{G} = (\Sigma, \{S : \mathtt{o}\}, \{S \mapsto b(S)\}, S)$.

Then, $\emptyset \vdash S : q_1$ would be derivable by:

$$\dfrac{\dfrac{\emptyset \vdash b : (q_1, 1) \to q_1 \quad S : (q_1, 1) \vdash S : q_1}{S : (q_1, 1) \vdash b(S) : q_1}\ \textsc{T-App}}{\emptyset \vdash S : q_1}\ \textsc{Unsound-Rec}$$

The value tree of $\mathcal{G}$ is however not accepted by $\mathcal{A}'_1$.

We shall therefore define the typing relation $\vdash_{\mathcal{A}} \mathcal{G} : q$ in terms of parity games.

Definition 3.3. 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.

$$\begin{array}{rcl} V_\exists & = & \{(F, \theta, m) \mid F \in dom(\mathcal{N}), \theta ::_a \mathcal{N}(F), m \in dom(\Omega)\} \\ V_\forall & = & \{\Gamma \mid dom(\Gamma) \subseteq dom(\mathcal{N}) \text{ and } \forall F : (\theta, m) \in \Gamma. \theta ::_a \mathcal{N}(F)\} \\ E & = & \{((F, \theta, m), \Gamma) \mid \Gamma \vdash_{\mathcal{A}} \mathcal{R}(F) : \theta\} \cup \{(\Gamma, (F, \theta, m)) \mid F : (\theta, m) \in \Gamma\} \end{array}$$

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 $F' : (\theta', m')$ 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 $\forall$.

The standard typing for recursion (using UNSOUND-REC above) can be considered a degenerate case of our definition (using parity games), where all the priorities are 0. In fact, Kobayashi's type system [17] is obtained as a special case of our type system $\mathcal{T}_{\mathcal{A}}$ where the priorities are restricted to 0.

Example 3.3. Recall the recursion scheme $\mathcal{G}_0$ in Example 2.1 and the automaton $\mathcal{A}_1$ in Example 2.3. Let θ be $(q_0, 2) \wedge (q_1, 2) \to q_0$. Then, valid judgements include (recall Example 3.2 for the derivation of the second judgement):

$$\begin{array}{l} F : (\theta, 2) \vdash F\ \mathtt{c} : q_0 \\ F : (\theta, 2) \vdash \lambda x.\mathtt{a}\ x\ (F(\mathtt{b}\, x)) : \theta \end{array}$$

A memoryless winning strategy $\mathcal{W}$ for the parity game is given by:

$$\begin{array}{l} \mathcal{W}(S, q_0, 2) = F : (\theta, 2) \\ \mathcal{W}(F, \theta, 2) = F : (\theta, 2) \end{array}$$

Example 3.4. Recall the recursion scheme $\mathcal{G}_1$ in Example 2.2 and the automaton $\mathcal{A}_1$ in Example 2.3. Let $\theta_1 = (q_1, 1) \to q_1$ and $\theta_2 = (q_1, 2) \to q_0$. Then, the following type judgments hold:

$$\begin{array}{l} F : ((\theta_1, 2) \wedge (\theta_2, 2) \to q_0, 2) \vdash \mathcal{R}_1(S) : q_0 \\ \Gamma_F \vdash \mathcal{R}_1(F) : (\theta_1, 2) \wedge (\theta_2, 2) \to q_0 \\ \emptyset \vdash \mathcal{R}_1(D) : (\theta_1, 1) \to \theta_1 \\ \emptyset \vdash \mathcal{R}_1(D) : (\theta_1, 2) \wedge (\theta_2, 2) \to \theta_2 \end{array}$$

where Γ_F is:

$$D : ((\theta_1, 1) \to \theta_1, 2), D : ((\theta_1, 2) \wedge (\theta_2, 2) \to \theta_2, 2), F : ((\theta_1, 2) \wedge (\theta_2, 2) \to q_0, 2).$$

A memoryless winning strategy $\mathcal{W}$ for the parity game is given by:

$$\begin{array}{l} \mathcal{W}(S, q_0, 2) = F : ((\theta_1, 2) \wedge (\theta_2, 2) \to q_0, 2) \\ \mathcal{W}(F, (\theta_1, 2) \wedge (\theta_2, 2) \to q_0, 2) = \Gamma_F \\ \mathcal{W}(D, (\theta_1, 1) \to \theta_1, 2) = \emptyset \\ \mathcal{W}(D, (\theta_1, 2) \wedge (\theta_2, 2) \to \theta_2, 2) = \emptyset \end{array}$$

4. Correctness of the Type System

This section shows that the type system is sound and complete: a higher-order recursion scheme $\mathcal{G}$ is well-typed if, and only if, the tree generated by $\mathcal{G}$ is accepted by the alternating parity tree automaton.

4.1. Soundness.

Suppose that we are given a recursion scheme $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ with $dom(\mathcal{N}) = \{F_1, \ldots, F_n\}$ and $S = F_1$, 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 $\Gamma \vdash_{\mathcal{A}} (\lambda x.t_0)t_1 : \theta$, then there exists Γ' such that $\Gamma' \vdash_{\mathcal{A}} [t_1/x]t_0 : \theta$ and $\Gamma' \subseteq \Gamma$.

Proof. See Appendix A. □

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 $\langle \alpha, \ell, \Lambda, \Gamma \vdash t : q \rangle$ where $\Gamma \vdash t : q$ holds. Here ℓ is a natural number, Λ is a partial map from natural numbers to priorities, and α is an element of $\{1, \ldots, A\}^*$, where A is the largest arity of the terminal symbols of $\mathcal{G}$. The label ℓ counts the number of rewriting steps along each path, and $\Lambda(\ell)$ denotes the largest priority in the path between the current node and the node where a non-terminal labelled by ℓ has been introduced. 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 (partial) 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, as $\Gamma \vdash F^{\theta}\widetilde{t} : q$. It means that $\Gamma \vdash F\,\widetilde{t} : q$ is derived from the typing $F : (\theta, \Omega(\theta)) \vdash F : \theta$ for the occurrence of F as the head symbol, followed by applications of T-App.

The initial tree of the rewrite system is $\langle \epsilon, 1, \{0 \mapsto \Omega(q_I)\}, S^0 : (q_I, \Omega(q_I)) \vdash S^0 : q_I \rangle$. Here, each non-terminal symbol is annotated with a natural number, to indicate when the symbol was introduced. The rewrite relation $T \;\triangleright\; T'$ on $\mathbf{RLab}$-labelled trees is defined by induction over the following rules:

(i) If $\Gamma \vdash F_i^{\ell',\theta}\widetilde{t} : q$ holds and $\Gamma_{(F_i,\theta,\Lambda(\ell'))}$ is defined, then

$$\langle \alpha, \ell, \Lambda, \Gamma \vdash F_i^{\ell'}\widetilde{t} : q \rangle \;\triangleright\; \langle \alpha, \ell+1, \Lambda\{\ell \mapsto \Omega(q)\}, \Gamma' \vdash [\widetilde{t}/\widetilde{x}]\rho(t') : q \rangle$$

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

Here, Γ' is determined as follows: Take the derivation of $\Gamma \vdash F_i^{\ell',\theta}\widetilde{t} : q$, and replace the T-VAR instance $F : (\theta, \Omega(\theta)) \vdash F : \theta$ by $\rho(\Gamma_{(F_i,\theta,\Lambda(\ell'))}) \vdash \rho(\mathcal{R}(F_i)) : \theta$, yielding (a derivation for) $\Gamma_1 \cup \rho(\Gamma_{(F_i,\theta,\Lambda(\ell'))}) \vdash \rho(\mathcal{R}(F_i))\widetilde{t} : q$ for Γ_1 such that $\Gamma_1 \cup \{F : (\theta, \Omega(\theta))\} = \Gamma$. By the type preservation property (Lemma 4.1), there exists Γ' such that $\Gamma' \subseteq \Gamma_1 \cup \rho(\Gamma_{(F_i,\theta,\Lambda(\ell'))})$ and $\Gamma' \vdash [\widetilde{t}/\widetilde{x}]\rho(t') : q$. Thus, we choose one such Γ' above.

The renaming of F_i to F_i^{ℓ} is required to state Lemma 4.4.

(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

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

Here, $\Lambda \Uparrow m$ is defined by:

$$(\Lambda \Uparrow m)(\ell) = \max(\Lambda(\ell), m).$$

(iii) If $T \rhd T'$ then $C[T] \rhd C[T']$ for every tree context C. Here, ($\mathbf{RLab}$-labelled) tree contexts are defined by:

$$C ::= [\,] \mid \langle \alpha, q\rangle\, T_1 \cdots T_{i-1}\, C\, T_{i+1}, \cdots T_k,$$

where $T_1, \ldots, T_k$ are $\mathbf{RLab}$-labelled trees, and $C[T]$ denotes the tree obtained by replacing $[\,]$ in C with T.

Example 4.1. Recall the recursion scheme $\mathcal{G}_0$ in Example 2.1 and the automaton $\mathcal{A}_1$ in Example 2.3. By using the winning strategy $\mathcal{W}$ in Example 3.3, we obtain the following rewrite sequence:

$$\begin{array}{l}\langle \epsilon, 1, \Lambda_0, S^0 : (q_0, 2) \vdash S^0 : q_0\rangle \\ \rhd \langle \epsilon, 2, \Lambda_1, F^1 : (\theta, 2) \vdash F^1\,\mathtt{c} : q_0\rangle \\ \rhd \langle \epsilon, 3, \Lambda_2, F^2 : (\theta, 2) \vdash \mathtt{a}\,\mathtt{c}\,(F^2(\mathtt{b}\,\mathtt{c})) : q_0\rangle \\ \rhd \langle \epsilon, q_0\rangle\langle 1, 4, \Lambda_2, \emptyset \vdash \mathtt{c} : q_0\rangle\langle 2, 4, \Lambda_2, F^2 : (\theta, 2) \vdash F^2(\mathtt{b}\,\mathtt{c}) : q_0\rangle \\ \rhd^* \langle \epsilon, q_0\rangle\langle 1, q_0\rangle\langle 2, 5, \Lambda_3, F^4 : (\theta, 2) \vdash \mathtt{a}\,(\mathtt{b}\,\mathtt{c})\,(F^4(\mathtt{b}\,(\mathtt{b}\,\mathtt{c}))) : q_0\rangle \\ \rhd^* \langle \epsilon, q_0\rangle\langle 1, q_0\rangle(\langle 2, q_0\rangle(\langle 21, q_0\rangle\langle 211, q_1\rangle)\langle 2, 6, \Lambda_3, F^4 : (\theta, 2) \vdash F^4(\mathtt{b}\,(\mathtt{b}\,\mathtt{c})) : q_0\rangle \\ \rhd \cdots\end{array}$$

Here, $\Lambda_0 = \{0 \mapsto 2\}$, $\Lambda_1 = \Lambda_0\{1 \mapsto 2\}$, $\Lambda_2 = \Lambda_1\{2 \mapsto 2\}$, $\Lambda_3 = \Lambda_2\{4 \mapsto 2\}$, and $\theta = (q_0, 2) \wedge (q_1, 2) \to q_0$. The rewrite sequence generates the accepting run-tree in Figure 2 in Section 2. □

To show the soundness of the type system (Theorem 4.9 below), we shall prove that there exists a (possibly infinite) rewrite sequence of $\langle \epsilon, 1, \{0 \mapsto \Omega(q_I)\}, S^0 : (q_I, \Omega(q_I)) \vdash S^0 : q_I\rangle$ that generates an accepting run-tree of $\mathcal{A}$ over $[\![\mathcal{G}]\!]$. The proof consists of the following three main lemmas:

(I) Rewrite sequences from $\langle \epsilon, 1, \{0 \mapsto \Omega(q_I)\}, S^0 : (q_I, \Omega(q_I)) \vdash S^0 : q_I\rangle$ never get stuck (Lemma 4.2).

(II) Any maximal[5] fair rewrite sequence

$$\langle \epsilon, 1, \{0 \mapsto \Omega(q_I)\}, S^0 : (q_I, \Omega(q_I)) \vdash S^0 : q_I\rangle \rhd T_1 \rhd T_2 \rhd \cdots$$

generates a run-tree of $\mathcal{A}$ over $[\![\mathcal{G}]\!]$ (Lemma 4.6).

(III) Any infinite path of the run-tree mentioned in (II) satisfies parity conditions; the maximal fair rewrite sequence of (II), therefore, generates an *accepting* run-tree (Lemma 4.7).

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

We first prove the first key property (I), stated more formally as follows:

Lemma 4.2. If $\langle \epsilon, 1, \{0 \mapsto \Omega(q_I)\}, S^0 : (q_I, \Omega(q_I)) \vdash S^0 : q_I\rangle \ \triangleright^* \ C[\langle \alpha, \ell, \Lambda, \Gamma \vdash t : q\rangle]$, then $\langle \alpha, \ell, \Lambda, \Gamma \vdash t : q\rangle \ \triangleright \ T$ holds for some T.

We prepare a few lemmas to prove the above lemma. The following lemma states that typing is preserved by the rewrite relation $\triangleright$.

Lemma 4.3. If $\langle \epsilon, 1, \{0 \mapsto \Omega(q_I)\}, S^0 : (q_I, \Omega(q_I)) \vdash S^0 : q_I\rangle \ \triangleright^* \ C[\langle \alpha, \ell, \Lambda, \Gamma \vdash t : q\rangle]$, then $\Gamma \vdash t : q$ holds.

Proof. This follows by straightforward induction on the length of the rewrite sequence. The induction step follows immediately from the definition of $\triangleright$. $\square$

Thanks to Lemma 4.3, the only possibility for a rewrite sequence to get stuck is that $\Gamma_{(F,\theta,\Lambda(\ell'))}$ is undefined in clause (i) of the definition of $\triangleright$. To deny that possibility, we shall match a rewrite sequence with a play of the parity game associated with the type system.

By the *priority* of a ($\mathbf{RLab}$-labelled) 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 $[\,]_q$. Formally, it is defined by:

$$\begin{array}{l}\Omega([\,]_q) = \Omega(q)\\ \Omega(\langle \alpha, q'\rangle\, T_1\, \cdots\, T_{i-1}\, C[\,]_q\, T_{i+1}, \cdots\, T_k) = \max(\Omega(q'), \Omega(C[\,]_q))\end{array}$$

The following lemma confirms that variables in the type environment are used correctly, according to the intuition on type environments explained in Section 3.

Lemma 4.4. Suppose $\langle \alpha_0, \ell_0, \Lambda_0, \Gamma_0 \vdash s_0 : q_0\rangle \ \triangleright^* \ C[\langle \alpha, \ell, \Lambda, \Gamma \vdash F^\theta \widetilde{t} : q\rangle]$, and F is not introduced by renaming (i.e. via $\rho(-)$) in any of the intermediate rewriting steps. Then, $F{:}(\theta, \Omega(C[\,]_q)) \in \Gamma_0$.

Proof. The proof proceeds by induction on the length r of the reduction sequence

$$\langle \alpha_0, \ell_0, \Lambda_0, \Gamma_0 \vdash s_0 : q_0\rangle \ \triangleright^* \ C[\langle \alpha, \ell, \Lambda, \Gamma \vdash F^\theta \widetilde{t} : q\rangle].$$

For the base case of $r = 0$, we have $q = q_0$, $\Gamma_0 = \Gamma$ and the context $C[\,]_q$ is $[\,]_q$. By the definition of the annotation F^θ, $\Gamma \vdash F^\theta \widetilde{t} : q$ must have been derived from $F : (\theta, \Omega(\theta)) \vdash F : \theta$, which implies that $F : (\theta, \Omega(\theta)) \in \Gamma$.

We show the inductive case by case analysis on the first reduction step.

- Suppose the first reduction step is of the form

$$\langle \alpha_0, \ell_0, \Lambda_0, \Gamma_0 \vdash F_k^{\ell'} \widetilde{t}_0 : q_0\rangle \ \triangleright \ \langle \alpha, \ell_0 + 1, \Lambda', \Gamma' \vdash [\widetilde{t}/\widetilde{x}]\rho(t') : q_0\rangle$$

where $s_0 = F_k^{\ell'} \widetilde{t}_0$ with $\rho(-) := [F_1^{\ell_0}/F_1, \ldots, F_n^{\ell_0}/F_n](-)$ and $\mathcal{R}(F_k) = \lambda\widetilde{x}.t'$. Here, by the assumption that F is not introduced by the intermediate reduction steps, $F \notin \{F_1^{\ell_0}, \ldots, F_n^{\ell_0}\}$. By the induction hypothesis, $F : (\theta, \Omega(C[\,]_q)) \in \Gamma' \setminus \{F_1^{\ell_0}, \ldots, F_n^{\ell_0}\}$ holds. (Here, we write $\Gamma \setminus S$ for the type environment obtained from Γ by removing all the bindings on variables in S.) By the definition of $\triangleright$, we have $\Gamma' \setminus \{F_1^{\ell_0}, \ldots, F_n^{\ell_0}\} \subseteq \Gamma_0$. Thus, the required result follows.
- Suppose the first reduction step is of the form

$$\begin{array}{l}\langle \alpha_0, \ell_0, \Lambda_0, \Gamma_0 \vdash a t_1 \cdots t_n : q_0\rangle \triangleright\\ \langle \alpha_0, q_0\rangle(\langle \alpha_0 1, \ell_0 + 1, \Lambda_{1,1}, \Gamma_{1,1} \vdash t_1 : q_{1,1}\rangle, \ldots, \langle \alpha_0 1, \ell_0 + 1, , \Lambda_{1,k_1} \Gamma_{1,k_1} \vdash t_1 : q_{1,k_1}\rangle\\ \qquad \ldots \langle \alpha_0 n, \ell_0 + 1, \Lambda_{n,1}, \Gamma_{n,1} \vdash t_n : q_{n,1}\rangle, \ldots, \langle \alpha_0 n, \ell_0 + 1, \Lambda_{n,k_n}, \Gamma_{n,k_n} \vdash t_n : q_{n,k_n}\rangle)\end{array}$$

where $s_0 = a t_1 \cdots t_n$ and $\Lambda_{i,j} = \Lambda_0 \Uparrow \Omega(q_{i,j})$. Then, we have $T_{1,1}, \ldots, T_{n,k_n}$ such that:

(i) $C[\langle \alpha, \ell, \Lambda, \Gamma \vdash F^{\theta}\widetilde{t} : q\rangle] = \langle \alpha_0, q_0\rangle T_{1,1} \cdots T_{n,k_n}$; and

(ii) there exist $i, j (1 \leq i \leq n, 1 \leq j \leq k_n)$ such that $T_{i,j} = C'[\langle \alpha, \ell, \Lambda, \Gamma \vdash F^{\theta}\widetilde{t} : q\rangle]$ and
$$\langle \alpha_0 i, \ell_0 + 1, \Lambda_{i,j}, \Gamma_{i,j} \vdash t_i : q_{i,j}\rangle \ \rhd^* \ C'[\langle \alpha, \ell, \Lambda, \Gamma \vdash F^{\theta}\widetilde{t} : q\rangle].$$

Note that $\Omega(C[\,]_q) = \max(\Omega(q_0), \Omega(C'[\,]_q)) = \max(\Omega(q_0), \Omega(q_{i,j}), \Omega(C'[\,]_q))$. By the induction hypothesis, we have $F : (\theta, \Omega(C'[\,]_q)) \in \Gamma_{i,j}$. Since $\Gamma_0 \vdash at_1 \cdots t_n : q_0$ is derived from $\Gamma_{i,j} \vdash t_i : q_{i,j}$, it must be the case that $\Gamma_0 = \bigcup_{i,j}(\Gamma_{i,j} \Uparrow \max(\Omega(q_{i,j}), \Omega(q_0)))$. Thus, we have $F : (\theta, \max(\Omega(q_{i,j}), \Omega(q_0), \Omega(C'[\,]_q)) \in \Gamma_0$. The required result follows from $\Omega(C[\,]_q) = \max(\Omega(q_{i,j}), \Omega(q_0), \Omega(C'[\,]_q))$.
□

The following lemma confirms that the Λ-component keeps the largest priority in the path between the current node and the node a non-terminal has been introduced.

Lemma 4.5. If $\langle \alpha_0, \ell_0, \Lambda_0, \Gamma_0 \vdash F\,\widetilde{t} : q_0\rangle \ \rhd^+ \ C[\langle \alpha, \ell, \Lambda, \Gamma \vdash s : q\rangle]$, then $\Lambda(\ell_0) = \Omega(C[\,]_q)$.

Proof. This follows by straightforward induction on the length of the reduction sequence. For the base case, by the definition of $\rhd$, $C = [\,]$ and $\Lambda(\ell_0) = \Omega(q)$, so that the result follows immediately. For the induction step, suppose $\langle \alpha_0, \ell_0, \Lambda_0, \Gamma_0 \vdash F\,\widetilde{t} : q_0\rangle \ \rhd^+ \ C'[\langle \alpha', \ell', \Lambda', \Gamma' \vdash s' : q'\rangle] \ \rhd \ C[\langle \alpha, \ell, \Lambda, \Gamma \vdash s : q\rangle]$. If the last step comes from (i) (with (iii)), $C = C'$ and $q = q'$ with $\Lambda = \Lambda'\{\ell \mapsto \Omega(q)\}$. By the induction hypothesis, we have $\Lambda(\ell_0) = \Lambda'(\ell_0) = \Omega(C'[\,]_q) = \Omega(C[\,]_q)$ as required. If the last step comes from (ii), $C = C'[\langle \alpha', q'\rangle \cdots [\,] \cdots]$ with $\Lambda = \Lambda' \Uparrow \Omega(q)$. By the induction hypothesis, we have $\Lambda(\ell_0) = \max(\Lambda'(\ell_0), \Omega(q)) = \max(\Omega(C'[\,]_{q'}), \Omega(q)) = \Omega(C[\,]_q)$ as required. □

We are now ready to prove Lemma 4.2.

Proof of Lemma 4.2. Suppose $\langle \epsilon, 1, \{0 \mapsto \Omega(q_I)\}, S^0 : (q_I, \Omega(q_I)) \vdash S^0 : q_I\rangle \rhd^* C[\langle \alpha, \ell, \Lambda, \Gamma \vdash t : q\rangle]$. By Lemma 4.3, we have $\Gamma \vdash t : q$. If t is of the form $at_1 \cdots t_n$, then the result follows immediately from $\Gamma \vdash t : q$ and clause (ii) for $\rhd$. If t is of the form $F_i^{\ell',\theta}\,\widetilde{s}$, then it suffices to show that $\Gamma_{(F_i, \theta, \Lambda(\ell'))}$ is well defined. By the assumption $\langle \epsilon, 1, \{0 \mapsto \Omega(q_I)\}, S^0 : (q_I, \Omega(q_I)) \vdash S^0 : q_I\rangle \ \rhd^* \ C[\langle \alpha, \ell, \Lambda, \Gamma \vdash t : q\rangle]$, we have the following rewrite sequence (obtained by possible permutations of the rewrite sequence above):
$$\begin{array}{ll}
 & \langle \epsilon, 1, \{0 \mapsto \Omega(q_I)\}, S^0 : (q_I, \Omega(q_I)) \vdash S^0 : q_I\rangle \\
\rhd^* & C_1[\langle \alpha_1, \ell_1, \Lambda_1, \Gamma_1 \vdash F_{i_1}^{\ell_0,\theta_1}\,\widetilde{t}_1 : q_1\rangle] \\
\rhd^* & C_1[C_2[\langle \alpha_2, \ell_2, \Lambda_2, \Gamma_2 \vdash F_{i_2}^{\ell_1,\theta_2}\,\widetilde{t}_2 : q_2\rangle]] \\
\rhd^* & C_1[C_2[C_3[\langle \alpha_3, \ell_3, \Lambda_3, \Gamma_3 \vdash F_{i_3}^{\ell_2,\theta_3}\,\widetilde{t}_3 : q_3\rangle]]] \\
\rhd^* & C_1[C_2[C_3[\cdots C_n[\langle \alpha_n, \ell_n, \Lambda_n, \Gamma_n \vdash F_{i_n}^{\ell_{n-1},\theta_n}\,\widetilde{t}_n : q_n\rangle] \cdots]]] \\
= & C[\langle \alpha, \ell, \Lambda, \Gamma \vdash F_i^{\ell',\theta}\,\widetilde{s} : q\rangle]
\end{array}$$
where $C_1[C_2[C_3[\cdots C_n \cdots]]] = C$ and $\ell_0 = 1$. For each $k \geq 0$, the reduction
$$\langle \alpha_k, \ell_k, \Lambda_k, \Gamma_k \vdash F_{i_k}^{\ell_{k-1},\theta_k}\widetilde{t}_k : q_k\rangle \ \rhd^* \ C_{k+1}[\langle \alpha_{k+1}, \ell_{k+1}, \Lambda_{k+1}, \Gamma_{k+1} \vdash F_{i_{k+1}}^{\ell_k,\theta_{k+1}}\widetilde{t}_{k+1} : q_{k+1}\rangle]$$
must be of the form
$$\begin{array}{ll}
 & \langle \alpha_k, \ell_k, \Lambda_k, \Gamma_k \vdash F_{i_k}^{\ell_{k-1},\theta_k}\widetilde{t}_k : q_k\rangle \\
\rhd & \langle \alpha_k, \ell_k + 1, \Lambda_k\{\ell_k \mapsto \Omega(q_k)\}, \Gamma_k' \vdash [\widetilde{t}_k/\widetilde{x}]\rho(t') : q_k\rangle \\
\rhd^* & C_{k+1}[\langle \alpha_{k+1}, \ell_{k+1}, \Lambda_{k+1}, \Gamma_{k+1} \vdash F_{i_{k+1}}^{\ell_k,\theta_{k+1}}\widetilde{t}_{k+1} : q_{k+1}\rangle]
\end{array}$$

where $\rho := [F_1^{\ell_k}/F_1, \dots, F_n^{\ell_k}/F_n]$ and $\mathcal{R}(F_{i_k}) = \lambda\widetilde{x}.t'$, with $\Gamma'_k \subseteq \Gamma_k \cup \rho(\Gamma_{(F_{i_k},\theta_k,\Lambda_k(\ell'_{k-1}))})$. By Lemma 4.4, $F_{i_{k+1}}^{\ell_k} : (\theta_{k+1}, \Omega(C_{k+1}[\,]_{q_{k+1}})) \in \Gamma'_k$, which implies $F_{i_{k+1}} : (\theta_{k+1}, \Omega(C_{k+1}[\,]_{q_{k+1}})) \in \Gamma_{(F_{i_k},\theta_k,\Lambda_k(\ell_{k-1}))}$. By Lemma 4.5, $\Lambda_k(\ell_{k-1}) = \Omega(C_k[\,]_{q_k})$.

Now from the preceding infinite $\triangleright$-rewrite sequence, we can extract a sequence

$$(S, q_I, \Omega(q_I))\ \Gamma_{(S,q_I,\Omega(q_I))}\ (F_{i_1}, \theta_1, \Omega(C_1[\,]_{q_1}))\ \Gamma_{(F_{i_1},\theta_1,\Omega(C_1[\,]_{q_1}))}\ (F_{i_2}, \theta_2, \Omega(C_2[\,]_{q_2}))$$
$$\Gamma_{(F_{i_2},\theta_2,\Omega(C_2[\,]_{q_2}))} (F_{i_3}, \theta_3, \Omega(C_3[\,]_{q_3}))\ \Gamma_{(F_{i_3},\theta_3,\Omega(C_3[\,]_{q_3}))} \cdots (F_{i_n}, \theta_n, \Omega(C_n[\,]_{q_n})),$$

which is a (partial) play for the parity game that conforms to the winning strategy $\mathcal{W}$. Thus, $\mathcal{W}(F_{i_n}, \theta_n, \Omega(C_n[\,]_{q_n}))(= \Gamma_{F_i,\theta,\Lambda(\ell')})$ must be well defined (since the player would lose otherwise). This completes the proof of the lemma. □

We now turn to prove the second key lemma (II), formally stated as Lemma 4.6 below. We write $T^\sharp$ for the (unranked) tree obtained by replacing each label of the form $\langle \alpha, \ell, \Lambda, \Gamma \vdash t : q\rangle$ with $\langle \alpha, q\rangle$.

Lemma 4.6. Let T_0 be $\langle \epsilon, 1, \{0 \mapsto \Omega(q_I)\}, S^0 : (q_I, \Omega(q_I)) \vdash S^0 : q_I\rangle$. If $T_0 \triangleright T_1 \triangleright T_2 \triangleright \cdots$ is a maximal fair rewrite sequence, then $T := \bigcup_{i\in\omega} {T_i}^\sharp$ is a run-tree of $\mathcal{A}$ over the value tree of $[\![\mathcal{G}]\!]$.

Proof. We first note that $\{{T_i}^\sharp\}_{i\in\omega}$ is a monotonically increasing sequence (with respect to the subset relation) and that each ${T_i}^\sharp$ is a $(dom([\![\mathcal{G}]\!]) \times Q)$-labelled (unranked) tree, so that T is also a $(dom([\![\mathcal{G}]\!]) \times Q)$-labelled tree. We check the two conditions for T being a run-tree. First, as ${T_0}^\sharp = \langle \epsilon, q_I\rangle$, $T(\epsilon) = \langle \epsilon, q_I\rangle$ holds. To check the second property, suppose $T(\beta) = \langle \alpha, q\rangle$. By the definition of T, there exists i such that $T_i(\beta) = \langle \alpha, q\rangle$ or $T_i(\beta) = \langle \alpha, \ell, \Lambda, \Gamma \vdash t : q\rangle$. If $T_i(\beta) = \langle \alpha, q\rangle$, then the node β must have been generated by using clause (ii) of $\triangleright$. Thus, the set $\{(j, q') \mid \exists j.{T_i}^\sharp(\beta j) = \langle \alpha i', q'\rangle\}$ satisfies $\delta_{\mathcal{A}}(q, a)$. Since ${T_i}^\sharp \subseteq T$, the set $\{(j, q') \mid \exists j.T(\beta j) = \langle \alpha i', q'\rangle\}$ also satisfies $\delta_{\mathcal{A}}(q, a)$ as required.

If $T_i(\beta) = \langle \alpha, \ell, \Lambda, \Gamma \vdash t : q\rangle$, then by Lemma 4.2 and the assumptions that the rewrite sequence is fair and that $[\![\mathcal{G}]\!]$ does not contain $\perp$, there exists $j(> i)$ such that $T_j(\beta) = \langle \alpha, q\rangle$. Thus, the required condition follows from the first case above. □

The last key property is stated as follows.

Lemma 4.7. Let T_0 be $\langle \epsilon, 1, \{0 \mapsto \Omega(q_I)\}, S^0 : (q_I, \Omega(q_I)) \vdash S^0 : q_I\rangle$. If $T_0 \triangleright T_1 \triangleright T_2 \triangleright \cdots$ is a maximal fair rewrite sequence, then for every infinite path π of $T := \bigcup_{i\in\omega} {T_i}^\sharp$, the largest priority that occurs infinitely often in π is even.

To prove the above lemma, we need the following property.

Lemma 4.8. For any infinite rewrite sequence:

$$\begin{aligned}
&\langle \epsilon, 1, \{0 \mapsto \Omega(q_I)\}, S^0 : (q_I, \Omega(q_I)) \vdash S^0 : q_I\rangle \\
&= \langle \alpha_1, 1, \Lambda_1, \Gamma_1 \vdash t_1 : q_1\rangle] \\
&\triangleright C_1[\langle \alpha_2, 2, \Lambda_2, \Gamma_2 \vdash t_2 : q_2\rangle] \\
&\triangleright C_1[C_2[\langle \alpha_3, 3, \Lambda_3, \Gamma_3 \vdash t_3 : q_3\rangle]] \\
&\triangleright C_1[C_2[C_3[\langle \alpha_4, 4, \Lambda_4, \Gamma_4 \vdash t_4 : q_4\rangle]]] \triangleright \cdots,
\end{aligned}$$

there exists an infinite sequence of indices $i_0(= 0), i_1, i_2, \dots$ such that $t_{i_j} = F_{k_{i_j}}^{i_{j-1}} \widetilde{s_{i_j}}$ for each $j \geq 1$. (In other words, there must be an infinite sequence of non-terminals: $S^0, F_{k_{i_1}}^{i_0},, F_{k_{i_2}}^{i_1}, \cdots$ such that $F_{k_{i_{j+1}}}^{i_j}$ has been obtained by unfolding $F_{k_{i_j}}^{i_{j-1}}$ at the i_j-th rewriting step.)

Proof. See Appendix B. □

Proof of Lemma 4.7. By Lemma 4.8, for any infinite path π of T, there must exist an infiniterewrite sequence:

$$\begin{array}{ll} & \langle \epsilon, 1, \{0 \mapsto \Omega(q_I)\}, S^0 : (q_I, \Omega(q_I)) \vdash S^0 : q_I \rangle \\ \rhd^* & C_1[\langle \alpha_1, \ell_1, \Lambda_1, \Gamma_1 \vdash F^1_{i_1}\, \widetilde{t}_1 : q_1 \rangle] \\ \rhd^* & C_1[C_2[\langle \alpha_2, \ell_2, \Lambda_2, \Gamma_2 \vdash F^{\ell_1}_{i_2}\, \widetilde{t}_2 : q_2 \rangle]] \\ \rhd^* & C_1[C_2[C_3[\langle \alpha_3, \ell_3, \Lambda_3, \Gamma_3 \vdash F^{\ell_2}_{i_3}\, \widetilde{t}_3 : q_3 \rangle]]] \\ \rhd^* & \cdots \end{array}$$

such that the holes of $C_1, C_1[C_2], C_1[C_2[C_3]], \ldots$ occur in the path. By using the same argument in the proof of Lemma 4.2, we can extract an infinite sequence

$$\begin{array}{l} (S, q_I, \Omega(q_I))\ \Gamma_{(S,q_I,\Omega(q_I))}\ (F_{i_1}, \theta_1, \Omega(C_1[\,]_{q_1}))\ \Gamma_{(F_{i_1},\theta_1,\Omega(C_1[\,]_{q_1}))} \\ \quad (F_{i_2}, \theta_2, \Omega(C_2[\,]_{q_2}))\ \Gamma_{(F_{i_2},\theta_2,\Omega(C_2[\,]_{q_2}))} (F_{i_3}, \theta_3, \Omega(C_3[\,]_{q_3}))\ \Gamma_{(F_{i_3},\theta_3,\Omega(C_3[\,]_{q_3}))} \cdots, \end{array}$$

which is a winning play. It follows that the largest priority that occurs infinitely often in $\Omega(C_1[\,]_{q_1}), \Omega(C_2[\,]_{q_2}), \Omega(C_3[\,]_{q_3}), \ldots$ is even. Therefore, the largest priority that occurs in the infinite path π of t must also be even. □

We are now ready to prove the soundness of the type system.

Theorem 4.9 (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. Suppose $\vdash_{\mathcal{A}} \mathcal{G}$. By Lemma 4.2, we can construct a maximal, fair rewrite sequence

$$\langle \epsilon, 1, \{0 \mapsto \Omega(q_I)\}, S^0 : (q_I, \Omega(q_I)) \vdash S^0 : q_I \rangle\ \rhd\ T_1\ \rhd\ T_2\ \rhd\ \cdots.$$

By Lemmas 4.6 and 4.7, $T := \bigcup_{i \in \omega} T_i^\sharp$ is an accepting run-tree of $\mathcal{A}$ over $[\![\mathcal{G}]\!]$. Thus, $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$. □

4.2. **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 $\vdash_{\mathcal{A}} \mathcal{G}$. To this end, we first define a rewrite relation $\gtrdot$, similar to $\rhd$, so that a fair rewrite sequence based on $\gtrdot$ generates the accepting run-tree. From the rewrite sequence, we extract a winning strategy for $\vdash_{\mathcal{A}} \mathcal{G}$.

We define the rewrite 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 $\langle \beta, \ell, t, q \rangle$. 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 rewrite relation $\gtrdot$ is defined by induction over the following rules:

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

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

(ii) If $\mathit{fst}(\beta) = \alpha$ and the children of the node $\langle \alpha, q \rangle$ of the run-tree are

$$\langle \alpha 1, q_{1,1} \rangle, \ldots, \langle \alpha 1, q_{1,k_1} \rangle, \ldots, \langle \alpha n, q_{n,1} \rangle, \ldots, \langle \alpha n, q_{n,k_n} \rangle$$

then:

$$\begin{array}{l} \langle \beta, \ell, a t_1 \cdots t_n, q \rangle \gtrdot \\ \quad \langle \mathit{fst}(\beta), q \rangle (\langle \beta(1,1), \ell + 1, t_1, q_{1,1} \rangle, \ldots, \langle \beta(1,k_1), \ell + 1, t_1, q_{1,k_1} \rangle, \\ \qquad \ldots \langle \beta(n,1), \ell + 1, t_n, q_{n,1} \rangle, \ldots, \langle \beta(n,k_n), \ell + 1, t_n, q_{n,k_n} \rangle) \end{array}$$

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

(iii) If $t \gtrdot t'$, then $C[t] \gtrdot C[t']$ for any $\mathbf{RLab}'$-labelled tree context C. Here, $\mathbf{RLab}'$-labelled tree contexts are defined by:

$$C ::= [\,] \mid \langle \alpha, q \rangle\, T_1 \,\cdots\, T_{i-1}\, C\, T_{i+1}, \cdots\, T_k,$$

where $T_1, \ldots, T_k$ are $\mathbf{RLab}'$-labelled trees.

Example 4.2. Recall the recursion scheme $\mathcal{G}_0$ in Example 2.1 and the automaton $\mathcal{A}_1$ in Example 2.3. Using the accepting run-tree in Figure 2 in Section 2, we obtain the following rewrite sequence:

$$
\begin{array}{l}
\langle \epsilon, 0, S, q_0 \rangle \\
\quad \gtrdot \langle \epsilon, 1, F\,\mathtt{c}, q_0 \rangle \\
\quad \gtrdot \langle \epsilon, 2, \mathtt{a}\,\mathtt{c}\,(F(\mathtt{b}\,\mathtt{c})) : q_0 \rangle \\
\quad \gtrdot \langle \epsilon, q_0 \rangle\, \langle (1,1), 3, \mathtt{c}, q_0 \rangle\, \langle (2,2), 3, F(\mathtt{b}\,\mathtt{c}), q_0 \rangle \\
\quad \gtrdot^* \langle \epsilon, q_0 \rangle\, \langle 1, q_0 \rangle\, \langle (2,2), 4, \mathtt{a}\,(\mathtt{b}\,\mathtt{c})\,(F(\mathtt{b}\,(\mathtt{b}\,\mathtt{c}))), q_0 \rangle \\
\quad \gtrdot \langle \epsilon, q_0 \rangle\, \langle 1, q_0 \rangle\, (\langle 2, q_0 \rangle\, \langle (2,2)(1,1), 5, (\mathtt{b}\,\mathtt{c}), q_0 \rangle\, \langle (2,2)(2,2), 5, (F(\mathtt{b}\,(\mathtt{b}\,\mathtt{c}))), q_0 \rangle) \\
\quad \gtrdot \langle \epsilon, q_0 \rangle\, \langle 1, q_0 \rangle\, (\langle 2, q_0 \rangle\, (\langle 21, q_0 \rangle \langle (2,2)(1,1)(1,1), 6, \mathtt{c}, q_1 \rangle)\, \langle (2,2)(2,2), 5, (F(\mathtt{b}\,(\mathtt{b}\,\mathtt{c}))), q_0 \rangle) \\
\quad \gtrdot \langle \epsilon, q_0 \rangle\, \langle 1, q_0 \rangle\, (\langle 2, q_0 \rangle\, (\langle 21, q_0 \rangle \langle 211, q_1 \rangle)\, \langle (2,2)(2,2), 5, (F(\mathtt{b}\,(\mathtt{b}\,\mathtt{c}))), q_0 \rangle) \\
\quad \gtrdot \cdots
\end{array}
$$

It generates the accepting run-tree in Figure 2. □

We write $T^\sharp$ for the (unranked) tree obtained by replacing each label of the form $\langle \beta, \ell, t, q \rangle$ with $\langle \mathit{fst}(\beta), q \rangle$. By the definition of the rewrite relation, there is a fair, possibly infinite rewrite sequence

$$\langle \epsilon, 0, S, q_I \rangle (= T_0) \gtrdot T_1 \gtrdot T_2 \gtrdot \cdots$$

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

We consider below only the reductions occurring in the sequence $T_0 \gtrdot T_1 \gtrdot T_2 \gtrdot \cdots$ (fixed above) and assume that each subterm is implicitly labelled, so that different occurrences of the same term are distinguished. For example, when we write $\langle \beta, \ell, t_0 t_1, q \rangle \gtrdot^* C[\langle \beta', \ell', t_1 t_2, q' \rangle]$, we assume that $T_i = C_0[\langle \beta, \ell, t_0 t_1, q \rangle]$ and $T_j = C_0[C[\langle \beta', \ell', t_1 t_2, q' \rangle]]$ for some i and j (where $i \leq j$), and t_1 in $t_1 t_2$ originates from t_1 in the argument position of $t_0 t_1$ (i.e. the former t_1 is a *residual* of the latter t_1 w.r.t. the rewrite 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 $\theta_{(t_0,\beta,\ell)}$ of a prefix t_0. A term t_0 is called a *prefix* of t if t is of the form $t_0 t_1 \cdots t_k$. For each leaf $\langle \beta, \ell, t, q \rangle$ of T_i and a prefix t_0 of t, we define the type $\theta_{(t_0,\beta,\ell)}$ by induction on the sort κ of t_0 as follows, so that $\theta_{(t_0,\beta,\ell)} ::_a \kappa$ holds.

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

(ii) If the sort of t_0 is $\kappa_1 \to \cdots \to \kappa_n \to \mathtt{o}$, then the leaf is of the form $\langle \beta, \ell, t_0 t_1 \cdots t_n, q \rangle$. Let S_i be the set of pairs $(\theta_{(t_i,\beta',\ell')}, \Omega(C[\,]_{q'}))$ such that $\langle \beta, \ell, t_0 t_1 \cdots t_n, q \rangle \gtrdot^* C[\langle \beta', \ell', t_i \widetilde{t'}, q' \rangle]$. Note that since the sort of κ_i is less than that of t_0, by the induction hypothesis, we can determine $\theta_{(t_i,\beta',\ell')}$. Note also that although the set of trees $C[\langle \beta', \ell', t_i \widetilde{t'}, q' \rangle]$ such that

$$\langle \beta, \ell, t_0 t_1 \cdots t_n, q \rangle \gtrdot^* C[\langle \beta', \ell', t_i \widetilde{t'}, q' \rangle]$$

may be infinite, S_i is finite. Thus we can define

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

Type environment $\Gamma_{(t_0,\beta,\ell)}$ *of a prefix* t_0. Next, we determine a type environment $\Gamma_{(t_0,\beta,\ell)}$ for each prefix term t_0 of the leaf $\langle \beta, \ell, t_0 t_1 \cdots t_n, q\rangle$ so that $\Gamma_{(t_0,\beta,\ell)} \vdash t_0 : \theta_{(t_0,\beta,\ell)}$ holds, by induction on the structure of the term.

- If $t_0 = a\ (\in \Sigma)$, then $\Gamma_{(t_0,\beta,\ell)} := \emptyset$.
- If $t_0 = F\ (\in \mathcal{N})$, then $\Gamma_{(F,\beta,\ell)} := F : (\theta_{(F,\beta,\ell)}, \Omega(\theta_{(F,\beta,\ell)}))$.
- If $t_0 = t_{0,1} t_{0,2}$, then let S be the set of triples
$$(\beta', \ell', \Omega(C[\,]_{q'}))$$
such that $\langle \beta, \ell, t_0 t_1 \cdots t_n, q\rangle \ \geqslant^* \ C[\langle \beta', \ell', t_{0,2}\widetilde{t'}, q'\rangle]$. Let S' be a subset of S such that for every $(\beta'', \ell'', m) \in S$, there exists exactly one $(\beta', \ell', m) \in S'$ such that $\theta_{(t_{0,2},\beta',\ell')} = \theta_{(t_{0,2},\beta'',\ell'')}$. We then define $\Gamma_{(t_0,\beta,\ell)}$ as
$$\Gamma_{(t_{0,1},\beta,\ell)} \cup (\bigcup \{\Gamma_{(t_{0,2},\beta',\ell')} \Uparrow m \mid (\beta', \ell', m) \in S'\}).$$

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 type (with priority) (θ, m) of $t_{0,2}$.

Example 4.3. Recall the rewrite sequence in Example 4.2. Then we have:

$$\begin{aligned}
&\theta_{(S,\epsilon,0)} = q_0\\
&\theta_{(F,\epsilon,1)} = (\theta_{(\mathtt{c},(1,1),3)}, 2) \wedge (\theta_{(\mathtt{c},(2,2)(1,1)(1,1),6)}, 2) \to \theta_{(F\,\mathtt{c},\epsilon,1)} = (q_0, 2) \wedge (q_1, 2) \to q_0\\
&\Gamma_{(S,\epsilon,0)} = S : \theta_{(S,\epsilon,0)} = S : q_0\\
&\Gamma_{(F\,\mathtt{c},\epsilon,1)} = \Gamma_{(F,\epsilon,1)} \cup \Gamma_{(\mathtt{c},(1,1),3)} \Uparrow 2 \cup \Gamma_{(\mathtt{c},(2,2)(1,1)(1,1),6)} \Uparrow 2 = F : (q_0, 2) \wedge (q_1, 2) \to q_0.
\end{aligned}$$

□

The rest of the proof proceeds by showing the following properties.

(I) The extracted types for non-terminals are correct, in the sense that there exists a type environment Γ such that $\Gamma \vdash \mathcal{R}(F) : \theta_{(F,\beta,\ell)}$ and $\Gamma \subseteq \Gamma_{(t,\beta,\ell)}$ for some t, β, ℓ, for each $\theta_{(F,\beta,\ell)}$.

(II) The strategy to choose Γ above in the position $(F, \theta_{(F,\beta,\ell)}, m)$ is a (memoryless) winning strategy for the parity game associated with the type system.

The following is the key lemma for showing (I).

Lemma 4.10. If $\langle \epsilon, 0, S, q_I\rangle \ \geqslant^* \ C[\langle \beta, \ell, t_0 t_1 \cdots t_n, q\rangle]$ where $t_0 = [s_1/x_1, \ldots, s_k/x_k]u$ then there exist Γ_0, $J_1, \ldots, J_k$, and $\theta_{i,j}, m_{i,j}$ $(1 \le i \le k, j \in J_i)$ that satisfy:

$$\begin{aligned}
&\Gamma_0, x_1 : \textstyle\bigwedge_{j \in J_1} (\theta_{1,j}, m_{1,j}), \ldots, x_k : \bigwedge_{j \in J_k} (\theta_{k,j}, m_{k,j}) \vdash u : \theta_{(t_0,\beta,\ell)}\\
&\{(\theta_{i,j}, m_{i,j}) \mid j \in J_i\} \subseteq \{(\theta_{(s_i,\beta',\ell')}, \Omega(C'[\,]_{q'})) \mid \langle \beta, \ell, t_0 t_1 \cdots t_n, q\rangle \ \geqslant^* \ C'[\langle \beta', \ell', s_i\,\widetilde{t'}, q'\rangle]\}\\
&\Gamma_0 \subseteq \Gamma_{(t_0,\beta,\ell)}
\end{aligned}$$

Proof. The proof proceeds by induction on the structure of u.

- Case where u is $a\ (\in \Sigma)$ or $F\ (\in \mathcal{N})$:
The required conditions hold for $\Gamma_0 = \Gamma_{(t_0,\beta,\ell)}$ and $J_i = \emptyset$ $(1 \le i \le k)$.
- Case where u is x_i:
In this case, $t_0 = s_i$. The required conditions hold for: $\Gamma_0 = \emptyset$, $J_i = \{1\}$ and $J_{i'} = \emptyset$ for $i' \ne i$, with $\theta_{i,1} = \theta_{(t_0,\beta,\ell)}, m_{i,1} = \Omega(q)$.

- Case where u is $u_0 u_1$:
  In this case, $t_0 = t_{0,0}t_{0,1}$ where $t_{0,0} = [\widetilde{s}/\widetilde{x}]u_0$ and $t_{0,1} = [\widetilde{s}/\widetilde{x}]u_1$. By the definition of $\Gamma_{(t_0,\beta,\ell)}$, we have:

$$\begin{array}{l}\Gamma_{(t_0,\beta,\ell)} = \Gamma_{(t_{0,0},\beta,\ell)} \cup (\bigcup_{h\in H} \Gamma_{(t_{0,1},\beta_h,\ell_h)} \Uparrow m_h) \\ \langle \beta, \ell, t_0 t_1 \cdots t_n, q\rangle \ \gtrdot^* \ C_h[\langle \beta_h, \ell_h, t_{0,1}\widetilde{t}_h, q_h\rangle], \qquad m_h = \Omega(C_h[\,]_{q_h}) \quad \text{(for each } h \in H) \\ \theta_{(t_{0,0},\beta,\ell)} = \bigwedge_{h\in H}(\theta_{(t_{0,1},\beta_h,\ell_h)}, m_h) \ \to \theta_{(t_0,\beta,\ell)}\end{array}$$

  By the induction hypothesis, we have:

$$\begin{array}{l}\Gamma_{0,0}, x_1 : \bigwedge_{j\in J_{0,1}} (\theta_{0,1,j}, m_{0,1,j}), \ldots, x_k : \bigwedge_{j\in J_{0,k}} (\theta_{0,k,j}, m_{0,k,j}) \vdash u_0 : \theta_{(t_{0,0},\beta,\ell)} \\ \{(\theta_{0,i,j}, m_{0,i,j}) \mid j \in J_{0,i}\} \subseteq \\ \qquad \{(\theta_{(s_i,\beta',\ell')}, \Omega(C'[\,]_{q'})) \mid \langle \beta, \ell, ([\widetilde{s}/\widetilde{x}]u_0)t_{0,1}t_1 \cdots t_n, q\rangle \ \gtrdot^* \ C'[\langle \beta', \ell', s_i\widetilde{t'}, q'\rangle]\} \\ \Gamma_{0,0} \subseteq \Gamma_{(t_{0,0},\beta,\ell)}\end{array}$$

  and, for each $h \in H$,

$$\begin{array}{l}\Gamma_{0,h}, x_1 : \bigwedge_{j\in J_{h,1}} (\theta_{h,1,1}, m_{h,1,1}), \ldots, x_k : \bigwedge_{j\in J_{h,k}} (\theta_{h,k,1}, m_{h,k,1}) \vdash u_1 : \theta_{(t_{0,1},\beta_h,\ell_h)} \\ \{(\theta_{h,i,j}, m_{h,i,j}) \mid j \in J_{h,i}\} \subseteq \\ \qquad \{(\theta_{(s_i,\beta',\ell')}, \Omega(C'[\,]_{q'})) \mid \langle \beta_h, \ell_h, ([\widetilde{s}/\widetilde{x}]u_1)\widetilde{t}_h, q_h\rangle \ \gtrdot^* \ C'[\langle \beta', \ell', s_i\widetilde{t'}, q'\rangle]\} \\ \Gamma_{0,h} \subseteq \Gamma_{(t_{0,1},\beta_h,\ell_h)}.\end{array}$$

  Let $m'_{h,i,j} := \max(m_{h,i,j}, m_h)$ (for $h \in H, i \in \{1, \ldots, k\}, j \in J_{h,i}$) and $m'_{0,i,j}$ be $m_{0,i,j}$. Let Γ_0 be $\Gamma_{0,0} \cup (\bigcup_{h\in H} \Gamma_{0,h} \Uparrow m_h)$. By applying T-APP, we get:

$$\Gamma_0, x_1 : \bigwedge_{h\in\{0\}\cup H, j\in J_{h,1}} (\theta_{h,1,j}, m'_{h,1,j}), \ldots, x_k : \bigwedge_{h\in\{0\}\cup H, j\in J_{h,k}} (\theta_{h,k,j}, m'_{h,k,j}) \vdash u : \theta_{(t_0,\beta,\ell)}.$$

  Furthermore, we have:

$$\begin{array}{rcl}\Gamma_0 & \subseteq & \Gamma_{(t_{0,0},\beta,\ell)} \cup (\bigcup_{h\in H} \Gamma_{(t_{0,1},\beta_h,\ell_h)} \Uparrow m_h) \\ & = & \Gamma_{(t_0,\beta,\ell)}\end{array}$$

  and $\{(\theta_{h,i,j}, m'_{h,i,1}) \mid h \in \{0\} \cup H, j \in J_{h,i}\}$ consists of pairs $(\theta_{(s_i,\beta',\ell')}, \Omega(C'[\,]_{q'}))$ satisfying $\langle \beta, \ell, t_0 t_1 \cdots t_n, q\rangle \ \gtrdot^* \ C'[\langle \beta', \ell', s_i\widetilde{t'}, q'\rangle]$ as required.

□

The first property (I) follows as an easy corollary of Lemma 4.10 above.

Lemma 4.11. If $\langle \epsilon, 0, S, q_I\rangle \ \gtrdot^* \ C[\langle \beta, \ell, F\widetilde{s}, q\rangle] \ \gtrdot \ C[\langle \beta, \ell + 1, [\widetilde{s}/\widetilde{x}]t, q\rangle]$, then there exists Γ such that $\Gamma \vdash \lambda\widetilde{x}.t : \theta_{(F,\beta,\ell)}$ and $\Gamma \subseteq \Gamma_{([\widetilde{s}/\widetilde{x}]t,\beta,\ell+1)}$.

Proof. By Lemma 4.10, there exists Γ such that:

$$\begin{array}{l}\Gamma, x_1 : \bigwedge_{j\in J_1} (\theta_{1,j}, m_{1,j}), \ldots, x_k : \bigwedge_{j\in J_k} (\theta_{k,j}, m_{k,j}) \vdash t : q \\ \{(\theta_{i,j}, m_{i,j}) \mid j \in J_i\} \subseteq \{(\theta_{(s_i,\beta',\ell')}, \Omega(C'[\,]_{q'})) \mid \langle \beta, \ell, [\widetilde{s}/\widetilde{x}]t, q\rangle \ \gtrdot^* \ C'[\langle \beta', \ell', s_i\widetilde{t'}, q'\rangle]\} \\ \Gamma \subseteq \Gamma_{([\widetilde{s}/\widetilde{x}]t,\beta,\ell+1)}\end{array}$$

By the second condition and the construction of $\theta_{(F,\beta,\ell)}$, it must be the case that

$$\begin{array}{l}\theta_{(F,\beta,\ell)} = \bigwedge_{j\in J'_1} (\theta_{1,j}, m_{1,j}) \to \cdots \to \bigwedge_{j\in J'_k} (\theta_{k,j}, m_{k,j}) \to q \\ J_1 \subseteq J'_1, \ldots, J_k \subseteq J'_k\end{array}$$

for some $J'_1, \ldots, J'_k$ Thus, $\Gamma \vdash \lambda\widetilde{x}.t : \theta_{(F,\beta,\ell)}$ is obtained by applying T-ABS. □

To show the second property (II), we need to match each priority occurring in a type environment with the largest priority occurring in a (partial) path in the accepting run-tree. That is carried out by the following lemma.

Lemma 4.12. If $\langle \epsilon, 0, S, q_I\rangle \gtrdot^* C[\langle \beta, \ell, t, q\rangle]$ and $F{:}(\theta, m) \in \Gamma_{(t,\beta,\ell)}$, then there exist $C', \beta', \ell', \widetilde{t'}, q'$ such that $\langle \beta, \ell, t, q\rangle \gtrdot^* C'[\langle \beta', \ell', F\,\widetilde{t'}, q'\rangle]$ and $m = \Omega(C'[\,]_{q'})$ with $\theta = \theta_{(F,\beta',\ell')}$.

Proof. We show the following, slightly strengthened property by induction on the structure of t_0.

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

- Case t is a terminal a or a non-terminal $F' \neq F$: This cannot happen by the construction of $\Gamma_{(t,\beta,\ell)}$.
- Case $t = F$: The required properties holds for $C' = [\,]$ $\beta' = \beta, \ell' = \ell$, and $q' = q$.
- Case $t = t_0t_1$: By the definition of $\Gamma_{(t,\beta,\ell)}$, we have:

$$\Gamma_{(t,\beta,\ell)} \;=\; \Gamma_{(t_0,\beta,\ell)} \cup \Gamma_{(t_1,\beta_1,\ell_1)} \Uparrow m_1 \cup \cdots \cup \Gamma_{(t_1,\beta_k,l_k)} \Uparrow m_k$$

where $\langle \beta, \ell, t_0t_1\widetilde{u}, q\rangle \gtrdot^* C_i[\langle \beta_i, \ell_i, t_1\widetilde{s}_i, q_i\rangle]$ and $m_i = \Omega(C_i[\,]_{q_i})$. If $F : (\theta, m) \in \Gamma_{(t_0,\beta,\ell)}$, then the result follows immediately from the induction hypothesis. Otherwise, we have $F : (\theta, m) \in \Gamma_{(t_1,\beta_i,\ell_i)} \Uparrow m_i$ for some i. By the definition of $\cdot \Uparrow m$, we have $F : (\theta, m') \in \Gamma_{(t_1,\beta_i,\ell_i)}$ for some m' such that $m = \max(m', m_i)$. By $\langle \beta, \ell, t_0t_1\widetilde{u}, q\rangle \gtrdot^* C_i[\langle \beta_i, \ell_i, t_1\widetilde{s}_i, q_i\rangle]$ and the induction hypothesis, we have:

$$\langle \beta_i, \ell_i, t_1\widetilde{s}_i, q_i\rangle \;\gtrdot^*\; C'_i[\langle \beta'_i, \ell'_i, F\widetilde{t'}, q'\rangle]$$

with $m' = \Omega(C'_i[\,]_{q'})$ and $\theta = \theta_{(F,\beta'_i,\ell'_i)}$. Thus, the required properties hold for $C = C_i[C'_i]$, $\beta' = \beta'_i$, and $\ell' = \ell'_i$.

□

We are now ready to prove the completeness.

Theorem 4.13 (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 rewrite 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 $[\beta, \ell, t]$ to indicate the corresponding node in the rewrite 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, $\Gamma^{[\beta,\ell,t]} \subseteq \Gamma_{(t,\beta,\ell)}$ 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 $\Gamma^{[\beta,\ell,t]}$ be $(S : (q_I, \Omega(q_I)))^{[\epsilon,0,S]}$ if $\pi = \epsilon$; otherwise, let it be the last state of π (in $V_\forall$). It must be the case that $F : (\theta, m) \in \Gamma^{[\beta,\ell,t]} \subseteq \Gamma_{(t,\beta,\ell)}$. By Lemma 4.12, there must exist C, β', ℓ' such that

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

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

By Lemma 4.11, there exists Γ' such that $\Gamma' \vdash \lambda\widetilde{x}.t_F : \theta_{(F,\beta',\ell')}$ and $\Gamma' \subseteq \Gamma_{([\widetilde{s}/\widetilde{x}]t_F,\beta',\ell'+1)}$. We pick one such Γ', and define $\mathcal{W}(\pi\,(F, \theta, m))$ as $\Gamma'^{[\beta',\ell'+1,[\widetilde{s}/\widetilde{x}]t_F]}$.

To check that $\mathcal{W}$ is indeed winning, consider an infinite play:

$$(F_0, q_0, m_0)\; \Gamma_0^{[\beta_0,\ell_0,t_0]}\; (F_1, \theta_1, m_1)\; \Gamma_1^{[\beta_1,\ell_1,t_1]}\; (F_2, \theta_2, m_2) \cdots$$

that conforms to $\mathcal{W}$ where $(F_0, q_0, m_0) = (S, q_I, \Omega(q_I))$. Then the rewrite sequence $\langle \epsilon, 0, S, q_I \rangle \gtrdot T_1 \gtrdot T_2 \gtrdot \cdots$ must be of the form:

$$\begin{array}{l}\langle \epsilon, 0, S, q_I \rangle \gtrdot \langle \beta_0, \ell_0, \mathcal{R}(S), q_0 \rangle \\ \gtrdot^* C_1[\langle \beta_1, \ell_1 - 1, F_1 \widetilde{s}_1, q_1 \rangle] \gtrdot C_1[\langle \beta_1, \ell_1, t_1, q_1 \rangle] \\ \gtrdot^* C_1[C_2[\langle \beta_2, \ell_2 - 1, F_2 \widetilde{s}_2, q_2 \rangle]] \gtrdot C_1[C_2[\langle \beta_2, \ell_2, t_2, q_2 \rangle]] \\ \gtrdot^* \cdots \end{array}$$

where $\Omega(C_i[\,]_{q_i}) = m_i (i \geq 1)$. Since the rewrite 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, $\mathcal{W}$ is winning, hence $\vdash_{\mathcal{A}} \mathcal{G}$. $\square$

5. 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 parametrized complexity of model checking higher-order recursion schemes. The main result is that, assuming that the alternating parity tree automaton and the largest arity and order of non-terminals are fixed, the time complexity of the type inference problem (hence also the recursion scheme model checking problem) is polynomial in the size of the recursion scheme.

The type inference 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 recursion schemes are normalized, so 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. To get such a normalized recursion scheme, it suffices to replace each rule of the form $F \mapsto \lambda\widetilde{x}.c\ t_1\ \cdots\ t_i\ \cdots\ t_J$ (where t_i is not of the form $F_i\ \widetilde{x}_i$) with $F \mapsto \lambda\widetilde{x}.c\ t_1\ \cdots\ (H\,\widetilde{x}')\ \cdots\ t_J$ and add the new rule: $H \mapsto \lambda\widetilde{x}'.\lambda\widetilde{y}.t_i\ \widetilde{y}$, where H is a fresh non-terminal, $\{\widetilde{x}'\}$ is the subset of variables $\{\widetilde{x}\}$ that occur in t_i, and $\widetilde{y}$ is a sequence of variables added to ensure that $t_i\ \widetilde{y}$ has sort o. Let $|\mathcal{G}_0|$ and A_0 be the size and the largest arity of the original recursion scheme. Since at most $|\mathcal{G}_0|$ replacements are required to normalize the recursion scheme, the size and the largest arity of the normalized recursion scheme is $O(|\mathcal{G}_0|A_0)$ and $2A_0$ respectively (the increase of the arity by A_0 is due to the extra parameters $\widetilde{y}$ above). Thus, the complexity result obtained below is not affected by the normalization.

Below we write A for the largest arity of non-terminal or terminal symbols, $N (\geq 1)$ 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 for the number of priorities, i.e., $|codom(\Omega)|$. In the discussion of the complexity below, we fix N and discuss the asymptotic complexity with respect to P, A, $|Q|$, and M. For a sort κ of order n, an upper-bound of the number of types of sort κ, written K_n, is given by:

$$K_0 = |Q| \qquad K_{n+1} = |Q| 2^{AMK_n}.$$

We note that $K_n = \mathbf{exp}_n(O(A|Q|M))$ for $n \geq 1$,[6] where $\mathbf{exp}_n(x)$ is defined by:

$$\mathbf{exp}_0(x) = x \qquad\qquad \mathbf{exp}_{i+1}(x) = 2^{\mathbf{exp}_i(x)}.$$

[6]We write $f(x) = g(O(h(x)))$ when $f(x)$ is bounded above by $g(h_0(x))$ for some function $h_0(x)$ such that $h_0(x) = O(h(x))$.

For $n = 1$, $K_n = |Q|2^{AM|Q|} = 2^{\log|Q|+AM|Q|} = 2^{O(A|Q|M)}$. For $n > 1$, we have:

$$\begin{aligned}K_n &= |Q|2^{AM\mathbf{exp}_{n-1}(O(AM|Q|))} = 2^{\log|Q|+\mathbf{exp}_{n-1}(\log(AM)+O(AM|Q|))} = 2^{\mathbf{exp}_{n-1}(O(AM|Q|))}\\ &= \mathbf{exp}_n(O(AM|Q|))\end{aligned}$$

.

For step 1, we first compute the set

$$S_F := \{(\Gamma,\theta) \mid \Gamma \vdash \mathcal{R}(F) : \theta,\quad \theta ::_a \mathcal{N}(F), \text{ and } \forall (G : (\theta', m)) \in \Gamma.\theta' ::_a \mathcal{N}(G)\}$$

for each non-terminal F.

Assume that $\mathcal{R}(F)$ is of the form $\lambda\widetilde{x}.c(F_1'\widetilde{x}_1)\cdots(F_J'\widetilde{x}_J)$. We first compute:

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

where κ_c is the sort of c. Since Γ_0 is a singleton set $\{c : (\theta_0, \Omega(\theta_0))\}$ (if c is a non-terminal or a variable) or empty (if c is a terminal), $|S_{F,0}|$ is at most K_N. Next, for each $(\Gamma_0, \tau_1 \to \cdots \to \tau_J \to q) \in S_{F,0}$ with $\tau_j = \bigwedge_{k\in I_j}(\theta_{j,k}, m_{j,k})$, we compute

$$S_{F,j,k} := \{\Gamma_{j,k} \mid \Gamma_{j,k} \vdash F_j'\widetilde{x}_j : \theta_{j,k}\}.$$

for each $j \in \{1,\ldots,J\}, k \in I_j$. Since the number of candidates for the type of F_j' is at most K_N and the types for the variables $\widetilde{x}_j$ are uniquely determined from the type of F_j', $|S_{F,j,k}|$ is at most K_N for each j,k. Note also that since the order of the sort 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 $S_{F,j,k}$, we can derive a judgment $\Gamma_0 \cup (\bigcup_{j,k}\Gamma_{j,k} \Uparrow m_{j,k}) \vdash c(F_1'\widetilde{x}_1)\cdots(F_J'\widetilde{x}_J) : q$. S_F 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 $\Gamma_0 \cup (\bigcup_{j,k}\Gamma_{j,k} \Uparrow m_{j,k}) \vdash c(F_1'\widetilde{x}_1)\cdots(F_J'\widetilde{x}_J) : \theta_0'$. Thus, the size $|S_F|$ of S_F is bounded by:

$$\begin{array}{ll} \;\;K_N & \text{(the number of choices of an element from } S_{F,0})\\ \times\Pi_{j\in\{1,\ldots,J\},k\in I_j}|S_{F,j,k}| & \text{(the number of choices of elements from } S_{F,j,k})\\ \times K_N & \text{(the number of choices for } \theta)\\ \leq K_N^{2+AMK_{N-1}} & \end{array}$$

The size of each type environment in S_F is at most $1 + |I_1| + \cdots + |I_J| \leq 1 + AMK_{N-1}$. Thus, the number of edges of the parity game (hence also $|V_\forall \cup V_\exists|$ to be considered) is bounded by:

$$\begin{array}{ll} \;\;PM|S_F| & \text{(a bound for } |\{((F,\theta,m),\Gamma) \mid \Gamma \vdash_{\mathcal{A}} \mathcal{R}(F) : \theta\}|)\\ +PM|S_F|(1+AMK_{N-1}) & \text{(a bound for } |\{(\Gamma,(F,\theta,m)) \mid F : (\theta,m) \in \Gamma\}|)\\ = PM(2+AMK_{N-1})K_N^{2+AMK_{N-1}} & \end{array}$$

Here, we have:

$$\begin{array}{rcl} K_1^{2+AMK_0} &=& (2^{\log|Q|+AM|Q|})^{(2+AM|Q|)} = 2^{O((AM|Q|)^2)}\\ K_N^{2+AMK_{N-1}} &=& (2^{\mathbf{exp}_{N-1}(O(AM|Q|))})^{2+AM\mathbf{exp}_{N-1}(O(AM|Q|))}\\ &=& 2^{\mathbf{exp}_{N-1}(O(AM|Q|))\times AM\mathbf{exp}_{N-1}(O(AM|Q|))}\\ &=& 2^{\mathbf{exp}_{N-1}(O(AM|Q|))}\\ &=& \mathbf{exp}_N(O(AM|Q|)) \qquad \text{(for } N \geq 2) \end{array}$$

Therefore, the size of the arena for the parity game is $P \times 2^{O((AM|Q|)^2)}$ for $N = 1$, and $P \times \mathbf{exp}_N(O(AM|Q|))$ for $N = 2$. The time complexity for constructing the arena is also in the same order.

In Step 2, we can use Schewe's algorithm [34] for solving parity games in time $O(|V_\forall \cup V_\exists||E|^{cM})$ for $c \approx \frac{1}{3}$.

Thus, the time complexity for the whole algorithm is

$$O(P^{1+cM}\mathbf{exp}_N(O(AM|Q|)))$$

for $N \geq 2$, and $O(P^{1+cM}2^{O(p(AM|Q|))})$ for $N = 1$ where $p(x)$ is a polynomial on x.

If N, A, $|Q|$, and M are fixed, then the algorithm runs in time $O(P^{1+cM})$. 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 other parameters (N, A, M, and $|Q|$) are fixed.

For a restricted class of APT called *disjuctive APT* [20], the time complexity is $(N-1)$-EXPTIME complete instead of N-EXPTIME compelte. Appendix C and [20] give proofs of the upper-bound and the lower-bound respectively.

6. Related Work

6.1. **Model checking recursion schemes.** As summarized in Section 1, studies of model checking recursion schemes were initiated by Knapik et al. [14, 15], 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 [15] also showed the equi-expressivity of safe recursion schemes and higher-order pushdown automata. Cachat and Walukiewicz [4, 5] 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 three other proofs of the decidability of the modal μ-calculus model checking. One is Ong's original proof [27], and the other two are due to Hague et al. [10] and Salvati and Walukiewicz [32] respectively. Ong [27] reduces the model checking problem to parity games over *variable profiles*, while Hague et al. [10] reduce 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). Salvati and Walukiewicz [32] reduce the model checking problem to a parity game over configurations of Krivine machines, and further reduce the latter to a parity game on a finite game. The notion of residuals [32] used in the second step is similar to our intersection types.

For a restricted class of properties called *trivial automata* (but for the full recursion schemes), Aehlig [1] gave a simpler proof. His approach is based on a novel finite semantics for simply-typed lambda term-trees: the meaning of an infinite tree is the set of states starting from which the given automaton has an infinite run. Kobayashi [17] recently showed a simple type-based proof based on a similar idea. Lester et al. [23] developed a type system for the class of alternating weak tree automata, as a degenerate case of our type system in the present article. Kobayashi and Ong [20] studied the complexity of model checking recursion schemes for various fragments of the modal μ-calculus. They use the type-based technique to show upper-bounds of the complexity.

Our type-based approach is a generalization of Kobayashi's type system [17]; when priorities are restricted to 0, our type system coincides with his system. Our type system is also inspired by Ong's *variable profiles* [27]. 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 sort 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 1, 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).

Broadbent et al. [3] considered an extension of the model checking of recursion schemes called *logical reflection*, whose goal is, given a recursion scheme $\mathcal{G}$ and a property ψ, to construct another recursion scheme that generates the same tree as $[\![\mathcal{G}]\!]$ except that each node is labelled by whether the node satisfies ψ. Carayol and Serre [6] considered a further extension called *effective MSO selection*. The proofs of the decidability of those extended problems take a detour to collapsible higher-order pushdown automata. It is left as future work to see whether our type-based approach can be extended to solve those problems directly without using collapsible higher-order pushdown automata.

6.2. **Implementations and applications of model checking of higher-order recursion schemes.** Since the development of type systems for model checking recursion schemes [17, 19], significant progress has been made on implementations of model checkers for higher-order recursion schemes and their applications to higher-order program verification. Kobayashi [16] implemented the first model checker for higher-order recursion schemes, for the class of deterministic trivial automata. Lester et al. [23] extended Kobayashi's algorithm to deal with weak alternating tree automata and implemented a model checker for that class. Kobayashi [18] and Neatherway et al. [26] implemented radically different algorithms by combining ideas from types and game semantics. Various program verification tools have been implemented on top of those model checkers. Unno et al. [22, 38] implemented a verification tool for tree-processing higher-order functional programs. Ong and Ramsay [28, 26] implemented a verification tool for higher-order functional programs with pattern matching. Kobayashi et al. [21, 33] also implemented a software model checker for a small subset of ML. So far those model checkers and program verification tools are mainly for safety properties (except [23]) and there are no implementations of full modal μ-calculus model checkers. Many of the algorithms mentioned above can, however, be easily (at least in theory) extended based on the type system in the present article, so that only engineering problems are left to implement a full modal μ-calculus model checker.

6.3. **Other related type systems.** Naik and Palsberg [25, 24] 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 [17] 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 [11]. There are substantial differences between those type systems and our type system. On one hand, programming languages for XML processing are concerned about *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. On the other hand, programming languages for XML have pattern match constructs on trees and one of the main issues in designing type systems for XML processing is how to type patterns, while recursion schemes do not have such constructs.

Intersection types have been recently applied to study problems similar to model checking of recursion schemes [31, 35]. Among others, Terui [35] used intersection types to study complexity of deciding whether a given simply-typed λ-term normalizes to a member of a fixed regular language.

7. 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 [27, 10], 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.

Implementation of a full modal μ-calculus model checker is left for future work. It is also interesting future work to see whether the type-based method can be used to solve other problems on higher-order recursion schemes, such as logical reflection [3] and pumping lemmas [29, 13].

Acknowledgment

We would like to thank Takeshi Tsukada for spotting a gap (formulated as Lemma 4.8) in the soundness theorem in an earlier version of this paper. This work was partially supported by Kakenhi (23220001 and 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] C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. A saturation method for collapsible pushdown systems. In *Proceedings of ICALP 2012*, volume 7392 of *Lecture Notes in Computer Science*, pages 165–176. Springer-Verlag, 2012.
[3] C. H. Broadbent, A. Carayol, C.-H. L. Ong, and O. Serre. Recursion schemes and logical reflection. In *Proceedings of LICS 2010*, pages 120–129. IEEE Computer Society Press, 2010.
[4] T. Cachat. Higher order pushdown automata, the caucal hierarchy of graphs and parity games. In *Proceedings of ICALP 2003*, volume 2719 of *Lecture Notes in Computer Science*, pages 556–569. Springer-Verlag, 2003.
[5] T. Cachat and I. Walukiewicz. The complexity of games on higher order pushdown automata. *CoRR*, abs/0705.0262, 2007.
[6] A. Carayol and O. Serre. Collapsible pushdown automata and labeled recursion schemes: Equivalence, safety and effective selection. In *Proceedings of LICS 2012*. IEEE Computer Society Press, 2012.
[7] B. Courcelle. The monadic second-order logic of graphs IX: machines and their behaviours. *Theoretical Computer Science*, 151:125–162, 1995.
[8] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FOCS 1991*, pages 368–377, 1991.
[9] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.
[10] 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.
[11] H. Hosoya, J. Vouillon, and B. C. Pierce. Regular expression types for XML. *ACM Trans. Program. Lang. Syst.*, 27(1):46–90, 2005.
[12] 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.
[13] A. Kartzow and P. Parys. Strictness of the collapsible pushdown hierarchy. In *Proceedings of MFCS 2012*, volume 7464 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2012.
[14] T. Knapik, D. Niwiński, and P. Urzyczyn. Deciding monadic theories of hyperalgebraic trees. In *TLCA 2001*, volume 2044 of *Lecture Notes in Computer Science*, pages 253–267. Springer-Verlag, 2001.
[15] T. Knapik, D. Niwiński, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer-Verlag, 2002.

[16] N. Kobayashi. Model-checking higher-order functions. In *Proceedings of PPDP 2009*, pages 25–36. ACM Press, 2009.

[17] N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 331–342, 2009.

[18] N. Kobayashi. A practical linear time algorithm for trivial automata model checking of higher-order recursion schemes. In *Proceedings of FoSSaCS 2011*, volume 6604 of *Lecture Notes in Computer Science*, pages 260–274. Springer-Verlag, 2011.

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

[20] N. Kobayashi and C.-H. L. Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. *Logical Methods in Computer Science*, 7(4), 2011.

[21] N. Kobayashi, R. Sato, and H. Unno. Predicate abstraction and cegar for higher-order model checking. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 222–233, 2011.

[22] N. Kobayashi, N. Tabuchi, and H. Unno. Higher-order multi-parameter tree transducers and recursion schemes for program verification. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 495–508, 2010.

[23] M. M. Lester, R. P. Neatherway, C.-H. L. Ong, and S. J. Ramsay. Model checking liveness properties of higher-order functional programs. In *Proceedings of ML Workshop 2011*, 2011.

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

[25] M. Naik and J. Palsberg. A type system equivalent to a model checker. In *ESOP 2005*, volume 3444 of *Lecture Notes in Computer Science*, pages 374–388. Springer-Verlag, 2005.

[26] R. P. Neatherway, C.-H. L. Ong, and S. J. Ramsay. A traversal-based algorithm for higher-order model checking. In *Proceedings of International Conference on Functional Programming*. ACM, 2012.

[27] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *Proceedings of IEEE Symposium on Logic in Computer Science*, pages 81–90. IEEE Computer Society Press, 2006.

[28] C.-H. L. Ong and S. Ramsay. Verifying higher-order programs with pattern-matching algebraic data types. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 587–598, 2011.

[29] P. Parys. A pumping lemma for pushdown graphs of any level. In *Proceedings of STACS 2012*, volume 14 of *LIPIcs*, pages 54–65. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

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

[31] S. Salvati. Recognizability in the simply typed lambda-calculus. In *Proceedings of WoLLIC 2009*, volume 5514 of *Lecture Notes in Computer Science*, pages 48–60. Springer, 2009.

[32] S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In *Proceedings of ICALP 2011*, volume 6756 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 2011.

[33] R. Sato, H. Unno, and N. Kobayashi. Towards a scalable software model checker for higher-order programs. In *Proceedings of PEPM 2013*, pages 53–62. ACM Press, 2013.

[34] S. Schewe. Solving parity games in big steps. In *Proceedings of FSTTCS 2007*, volume 4855 of *Lecture Notes in Computer Science*, pages 449–460. Springer-Verlag, 2007.

[35] K. Terui. Semantic evaluation, intersection types and complexity of simply typed lambda calculus. In *23rd International Conference on Rewriting Techniques and Applications (RTA'12)*, volume 15 of *LIPIcs*, pages 323–338. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[36] Y. Tobita, T. Tsukada, and N. Kobayashi. Exact flow analysis by higher-order model checking. In *Proceedings of FLOPS 2012*, volume 7294 of *Lecture Notes in Computer Science*, pages 275–289. Springer, 2012.

[37] T. Tsukada and N. Kobayashi. Untyped recursion schemes and infinite intersection types. In *Proceedings of FOSSACS 2010*, volume 6014 of *Lecture Notes in Computer Science*, pages 343–357. Springer-Verlag, 2010.

[38] H. Unno, N. Tabuchi, and N. Kobayashi. Verification of tree-processing programs via higher-order model checking. In *Proceedings of APLAS 2010*, volume 6461 of *Lecture Notes in Computer Science*, pages 312–327. Springer-Verlag, 2010.

[39] S. van Bakel. Complete restrictions of the intersection type discipline. *Theor. Comput. Sci.*, 102(1):135–163, 1992.

APPENDIX

APPENDIX A. TYPE PRESERVATION BY β-REDUCTION (LEMMA 4.1)

Lemma A.1. If $\Gamma \vdash_{\mathcal{A}} t : \theta$, then $(\Gamma \Uparrow \Omega(\theta)) = \Gamma$.

Proof. The proof proceeds by induction on the derivation of $\Gamma \vdash_{\mathcal{A}} t : \theta$, with case analysis on the last rule used.

- Case T-VAR: In this case, we have $t = x$ and $\Gamma = x : (\theta, \Omega(\theta))$. Thus, $\Gamma \Uparrow \Omega(\theta) = \Gamma$ follows immediately.
- Case T-CONST: Trivial, as $\Gamma = \emptyset$.
- Case T-APP: In this case, we have $t = t_0 t_1$, with:

  $$\Gamma_0 \vdash t_0 : \bigwedge_{i\in I}(\theta_i, m_i) \to \theta \qquad \Gamma_i \vdash t_1 : \theta_i \text{ for each } i \in I \qquad \Gamma = \Gamma_0 \cup \bigcup_{i\in I}(\Gamma_i \Uparrow m_i)$$

  By the induction hypothesis, $\Gamma_0 \Uparrow \Omega(\theta) = \Gamma_0$. By the well-formedness of $\bigwedge_{i\in I}(\theta_i, m_i) \to \theta$, it must be the case that $m_i \geq \Omega(\theta)$. Thus, $(\Gamma_i \Uparrow m_i) \Uparrow \Omega(\theta) = \Gamma_i \Uparrow m_i$. We have therefore $\Gamma \Uparrow \Omega(\theta) = \Gamma$ as required.
- Case T-ABS:
  In this case, we have $t = \lambda x.t_0$, with:

  $$\theta = \bigwedge_{i\in J}(\theta_i, m_i) \to \theta_0 \qquad I \subseteq J \qquad \Gamma, x : \bigwedge_{i\in I}(\theta_i, m_i) \vdash t_0 : \theta_0$$

  By the induction hypothesis, we have $(\Gamma, x : \bigwedge_{i\in I}(\theta_i, m_i)) \Uparrow \Omega(\theta_0) = \Gamma, x : \bigwedge_{i\in I}(\theta_i, m_i)$. Since $\Omega(\theta_0) = \Omega(\theta)$ and $x \notin dom(\Gamma)$, we have $\Gamma \Uparrow \Omega(\theta) = \Gamma$ as required.

$\square$

Lemma A.2 (Substitution)**.** If $\Gamma_0, x : \bigwedge_{i\in I}(\theta_i, m_i) \vdash t_0 : \theta$ and $\Gamma_i \vdash t : \theta_i$ for each $i \in I$, then $\Gamma_0 \cup \bigcup_{i\in I}(\Gamma_i \Uparrow m_i) \vdash [t/x]t_0 : \theta$ holds.

Proof. The proof proceeds by induction on derivation of $\Gamma_0, x : \bigwedge_{i\in I}(\theta_i, m_i) \vdash t_0 : \theta$, with case analysis on the last rule used.

- Cases for T-CONST:
  The result follows immediately, as x does not occur in t_0 and $\{(\theta_i, m_i) \mid i \in I\}$ is empty.
- Case for T-VAR:
  The case where $t_0 \neq x$ is trivial. If $t_0 = x$, we have:

  $$[t/x]t_0 = t \qquad \Gamma_0 = \emptyset \qquad I = \{1\} \qquad \theta = \theta_1 \qquad m_1 = \Omega(\theta)$$

  By applying Lemma A.1 to $\Gamma_1 \vdash t : \theta$, we obtain $\Gamma_1 \Uparrow m_1 = \Gamma_1$. Thus, we have $\Gamma_0 \cup (\Gamma_1 \Uparrow m_1) \vdash [t/x]t_0 : \theta$ as required.
- Case for T-APP:
  In this case, we have $t_0 = t_1 t_2$, with:

  $$\Gamma_{0,0}, x : \bigwedge_{i\in I, m\in S_{0,i}}(\theta_i, m) \vdash t_1 : \bigwedge_{j\in J}(\theta'_j, n_j) \to \theta$$
  $$\Gamma_{0,j}, x : \bigwedge_{i\in I, m\in S_{j,i}}(\theta_i, m) \vdash t_2 : \theta'_j \text{ for each } j \in J$$
  $$\Gamma_0 = \Gamma_{0,0} \cup \bigcup_{j\in J}(\Gamma_{0,j} \Uparrow n_j) \qquad S_{0,i} \cup \bigcup_{j\in J}\{max(m, n_j) \mid m \in S_{j,i}\} = \{m_i\} \text{ for each } i \in I.$$

  By the induction hypothesis, we have:

  $$\Gamma_{0,0} \cup \bigcup_{i\in I, m\in S_{0,i}}(\Gamma_i \Uparrow m) \vdash [t/x]t_1 : \bigwedge_{j\in J}(\theta'_j, n_j) \to \theta$$
  $$\Gamma_{0,j} \cup \bigcup_{i\in I, m\in S_{j,i}}(\Gamma_i \Uparrow m) \vdash [t/x]t_2 : \theta'_j.$$

By using T-APP, we obtain:

$$\Gamma_{0,0} \cup \bigcup_{i\in I, m\in S_{0,i}} (\Gamma_i \Uparrow m) \cup (\bigcup_{j\in J} (\Gamma_{0,j} \cup \bigcup_{i\in I, m\in S_{j,i}} (\Gamma_i \Uparrow m)) \Uparrow n_j) \vdash [t/x]t_0 : \theta$$

Here, we have:

$$\begin{aligned}
&\Gamma_{0,0} \cup \bigcup_{i\in I, m\in S_{0,i}} (\Gamma_i \Uparrow m) \cup (\bigcup_{j\in J} (\Gamma_{0,j} \cup \bigcup_{i\in I, m\in S_{j,i}} (\Gamma_i \Uparrow m)) \Uparrow n_j) \\
&= \Gamma_{0,0} \cup \bigcup_{j\in J} (\Gamma_{0,j} \Uparrow n_j) \cup \bigcup_{i\in I, m\in S_{0,i}} (\Gamma_i \Uparrow m) \cup \bigcup_{j\in J, i\in I, m\in S_{j,i}} (\Gamma_i \Uparrow \mathit{max}(m, n_j)) \\
&\qquad \text{(by } (\Gamma_1 \cup \Gamma_2) \Uparrow m = (\Gamma_1 \Uparrow m) \cup (\Gamma_2 \Uparrow m) \text{ and } (\Gamma \Uparrow m_1) \Uparrow m_2 = \Gamma \Uparrow \mathit{max}(m_1, m_2)\text{)} \\
&= \Gamma_0 \cup \bigcup_{i\in I} (\bigcup_{m\in S_{0,i}} (\Gamma_i \Uparrow m) \cup \bigcup_{j\in J, m\in S_{j,i}} \Gamma_i \Uparrow \mathit{max}(m, n_j)) \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{(by } \Gamma_0 = \Gamma_{0,0} \cup \bigcup_{j\in J} (\Gamma_{0,j} \Uparrow n_j)\text{)} \\
&= \Gamma_0 \cup \bigcup_{i\in I} (\Gamma_i \Uparrow m_i) \qquad \text{(by } S_{0,i} \cup \bigcup_{j\in J} \{\mathit{max}(m, n_j) \mid m \in S_{j,i}\} = \{m_i\}\text{)}
\end{aligned}$$

Thus, we have $\Gamma_0 \cup \bigcup_{i\in I} (\Gamma_i \Uparrow m_i) \vdash [t/x]t_0 : \theta$ as required.

- Case for T-ABS:
  In this case, $t_0 = \lambda y.t_1$. We can assume without loss of generality that $y \neq x$ and y does not occur in t. Thus, we have:

  $$\theta = \bigwedge_{j\in J} (\theta'_j, n_j) \to \theta' \qquad J' \subseteq J \qquad \Gamma_0, y : \bigwedge_{j\in J'} (\theta'_j, n_j), x : \bigwedge_{i\in I} (\theta_i, m_i) \vdash t_1 : \theta'$$

  By the induction hypothesis, we have:

  $$\Gamma_0 \cup \bigcup_{i\in I} (\Gamma_i \Uparrow m_i), y : \bigwedge_{j\in J'} (\theta'_j, n_j) \vdash [t_0/x]t_1 : \theta'.$$

  The required result is obtained by using T-ABS.

□

We are now ready to show that typing is preserved by β-reduction.

Proof of Lemma 4.1. By the assumption, we have:

$$\begin{aligned}
&\Gamma_0, x : \bigwedge_{i\in I} (\theta_i, m_i) \vdash t_0 : \theta \qquad \Gamma_i \vdash t_1 : \theta_i \text{ for each } i \in J \qquad I \subseteq J \\
&\Gamma = \Gamma_0 \cup (\bigcup_{i\in J} \Gamma_i \Uparrow m)
\end{aligned}$$

By Lemma A.2, we have:

$$\Gamma_0 \cup \bigcup_{i\in I} (\Gamma_i \Uparrow m_i) \vdash [t_1/x]t_0 : \theta.$$

Thus, the required result holds for $\Gamma' = \Gamma_0 \cup \bigcup_{i\in I} (\Gamma_i \Uparrow m_i)$. □

APPENDIX B. PROOF OF LEMMA 4.8

This section provides a proof of Lemma 4.8, which states that every infinite sequence of rewriting along a path must contain an infinite chain of unfoldings. In Lemma 4.8, only the components ℓ_i and t_i of each label $\langle \alpha_i, \ell_i, \Lambda_i, \Gamma_i \vdash t_i : q_i \rangle$ are actually important. Thus, we redefine the rewriting relation $\rhd$ by:

$$\langle \ell, F^{\ell'}\,\widetilde{t} \rangle \;\rhd\; \langle \ell+1, [\widetilde{t}/\widetilde{x}][F_1^{\ell}/F_1, \ldots, F_n^{\ell}/F_n](t') \rangle \text{ if } \mathcal{R}(F) = \lambda\widetilde{x}.t'$$

$$\langle \ell, a\,t_1, \ldots, t_n \rangle \;\rhd\; \langle \ell+1, t_i \rangle$$

Lemma 4.8 is restated (or strengthened, since we no longer have conditions on typing) as follows.

Lemma B.1. For every infinite rewriting sequence:

$$\langle 1, S^0\rangle = \langle 1, t_1\rangle \triangleright \langle 2, t_2\rangle \triangleright \langle 3, t_3\rangle \triangleright \langle 4, t_3\rangle \triangleright \cdots$$

there exists an infinite sequence of indices $i_0(=0), i_1, i_2, \dots$ such that $t_{i_j} = F^{i_{j-1}}_{k_{i_j}} \widetilde{s_{i_j}}$ for each $j \geq 1$.

The proof requires a subtle argument. Let us consider the tree U labelled by indices, whose root is lablled by 0, and j is a child of i if and only if t_j is of the form $F^i_{k_j}\, \widetilde{s_j}$. The existence of an infinite sequence of indices above means that the tree has an infinite path. By the strong normalization of the simply-typed λ-calculus, it follows that the height of the tree U is unbounded. (If the height were bounded, then the rewriting sequence would be simulated by a simply-typed λ-term obtained by unfolding non-terminal symbols a finite number of terms, but that is impossible as the rewriting sequence is infinite.) That does not, however, immediately imply that U has an infinite path; it may be the case that U has infinite width but each path is finite.

We use a type-based argument to show Lemma B.1. We first prepare a type system for characterizing (non-deterministic) recursion schemes that have an infinite reduction sequence in Section B.1, and then use it to prove the required property in Section B.2.

B.1. **A Type System for Recursion Schemes Generating Infinite Trees.** A *non-deterministic* recursion scheme is a quadruple $(\Sigma, \mathcal{N}, \mathcal{R}, S)$, which is the same as a deterministic recursion scheme (Definition 2.1), except that the set of non-terminals may be infinite, and that $\mathcal{R}$ maps each non-terminal to a (possibly empty) *set* of terms of the form $\lambda\widetilde{x}.t$ such that $\mathcal{N}; \Sigma \vdash \lambda\widetilde{x}.t : \mathcal{N}(F)$. As in the case for deterministic recursion schemes, we require that t is an applicative term of sort $\mathtt{o}$. We sometimes write $F \to \lambda\widetilde{x}.t$ if $\lambda\widetilde{x}.t \in \mathcal{R}(F)$.

We define the reduction relation $\overset{\bullet}{\triangleright}$ by

$$F\,\widetilde{t} \overset{\bullet}{\triangleright} [\widetilde{t}/\widetilde{x}]t' \text{ if } \lambda\widetilde{x}.t' \in \mathcal{R}(F)$$

$$a\, t_1, \dots, t_n \overset{\bullet}{\triangleright} t_i$$

It is the same as $\triangleright$ defined at the beginning of this section, except that labels have been dropped.

The syntax of types is given by:

$$\delta ::= \mathtt{i} \mid \bigwedge\{\delta_1, \dots, \delta_n\} \to \delta$$

We often write $\bigwedge_{i\in I} \delta_i \to \delta$ for $\bigwedge\{\delta_i \mid i \in I\} \to \delta$. Intuitively, $\mathtt{i}$ describes terms of sort $\mathtt{o}$ that have an infinite reduction sequence. As in the intersection types used in Section 3, we restrict types by the relation $\delta ::_a \kappa$, defined inductively by:

$$\frac{}{\mathtt{i} ::_a \mathtt{o}}$$

$$\frac{\delta ::_a \kappa_2 \qquad \delta_i ::_a \kappa_1 \text{ for every } i \in \{1, \dots, n\}}{(\bigwedge\{\delta_1, \dots, \delta_n\} \to \delta) ::_a (\kappa_1 \to \kappa_2)}$$

The typing rules for recursion schemes are given by:

$$\Delta, x : \delta \vdash x : \delta$$

$$\frac{\Sigma(a) = k \qquad S_1 \cup \cdots \cup S_k = \{\mathtt{i}\}}{\Delta \vdash a : \bigwedge S_1 \to \cdots \to \bigwedge S_k \to \mathtt{i}}$$

$$\frac{\Delta \vdash t_1 : \bigwedge_{i\in I} \delta_i \to \delta \qquad \Delta \vdash t_2 : \delta_i \text{ (for each } i \in I)}{\Delta \vdash t_1 t_2 : \delta}$$

$$\frac{\Delta, x : \bigwedge_{i\in I} \delta_i \vdash t : \delta}{\Delta \vdash \lambda x.t : \bigwedge_{i\in I} \delta_i \to \delta}$$

$$\frac{\forall F : \delta \in \Delta.(\delta ::_a \mathcal{N}(F) \wedge \exists t \in \mathcal{R}_{\mathcal{G}}(F).\Delta \vdash t : \delta)}{\vdash \mathcal{G} : \Delta}$$

$$\frac{\vdash \mathcal{G} : \Delta \qquad \Delta \vdash t : \delta}{\vdash (\mathcal{G}, t) : \delta}$$

Note that a terminal symbol of arity 0 cannot have type $\mathtt{i}$; if $k = 0$ in the rule for constants, the assumption $S_1 \cup \cdots \cup S_k = \{\mathtt{i}\}$ cannot hold.

We show the following theorem.

Theorem B.2. Let S be the start symbol of a non-deterministic recursion scheme $\mathcal{G}$. Then, $\vdash (\mathcal{G}, S) : Tinf$ if, and only if, S has an infinite reduction sequence (by $\overset{\bullet}{\rhd}$).

The "only if" part of the above theorem follows from Lemmas B.3 and B.4 below.

Lemma B.3. If $\vdash (\mathcal{G}, t) : \mathtt{i}$, then there exists t' such that $t \overset{\bullet}{\rhd} t'$.

Proof. The proof is by contradiction. Suppose that $\vdash (\mathcal{G}, t) : \mathtt{i}$ but t is irreducible. By $\vdash (\mathcal{G}, t) : \mathtt{i}$, there must exist Δ such that $\mathcal{G} : \Delta$ and $\Delta \vdash t : \mathtt{i}$. Since $\mathcal{N}; \emptyset \vdash t : \mathtt{o}$, t must be either a terminal symbol a of arity 0, or of the form $F\,\widetilde{s}$ with $\mathcal{R}(F) = \emptyset$. Both cases, however, contradict with the assumption $\vdash (\mathcal{G}, t) : \mathtt{i}$. If $t = a$, then a cannot have type $\mathtt{i}$. If $t = F\,\widetilde{s}$, then by the condition $\Delta \vdash t : \mathtt{i}$, there must exist δ such that $F : \delta \in \Delta$, but it is impossible as $\mathcal{R}(F) = \emptyset$ and $\vdash \mathcal{G} : \Delta$. □

Lemma B.4. If $\vdash (\mathcal{G}, t) : \mathtt{i}$ and $t \overset{\bullet}{\rhd} t'$, then there exists t'' such that $t \overset{\bullet}{\rhd} t''$ and $\vdash (\mathcal{G}, t'') : \mathtt{i}$.

Proof. We use the following substitution lemma, proved by straightforward induction on the structure of t:

> If $\Delta, x : \delta_1, \ldots, x : \delta_k \vdash t : \delta$ and $\Delta \vdash s : \delta_i$ for each $i \in \{1, \ldots, k\}$, then $\Delta \vdash [s/x]t : \delta$ holds.

The proof of the present lemma proceeds by case analysis on the rule used for deriving $t \overset{\bullet}{\rhd} t'$. If $t \overset{\bullet}{\rhd} t'$ has been derived from the first rule, $t = F\,s_1\,\ldots\,s_k$ and $t' = [s_1/x_1, \ldots, s_k/x_k]u$ with $\lambda x_1. \cdots \lambda x_k.u \in \mathcal{R}(F)$. By the assumption $\vdash (\mathcal{G}, t) : \mathtt{i}$, we have:

$$\begin{array}{l}\vdash \mathcal{G} : \Delta \\ F : \bigwedge_{j\in S_1} \delta_{1,j} \to \cdots \to \bigwedge_{j\in S_k} \delta_{k,j} \to \mathtt{i} \in \Delta \\ \Delta \vdash s_i : \delta_{i,j} \text{ for each } i \in \{1, \ldots, k\}, j \in S_i\end{array}$$

By the first two conditions, there must exist a term $\lambda x_1. \cdots \lambda x_k.u' \in \mathcal{R}(F)$ such that

$$\Delta \vdash \lambda x_1. \cdots \lambda x_k.u' : \bigwedge_{j\in S_1} \delta_{1,j} \to \cdots \to \bigwedge_{j\in S_k} \delta_{k,j} \to \mathtt{i},$$

which implies $\Delta, x_1 : \bigwedge_{j \in S_1} \delta_{1,j}, \ldots, x_k : \bigwedge_{j \in S_k} \delta_{k,j} \vdash u' : \mathtt{i}$. By the substitution lemma, we obtain $\Delta \vdash [s_1/x_1, \ldots, s_k/x_k]u' : \mathtt{i}$. Thus, the required result holds for $t'' = [s_1/x_1, \ldots, s_k/x_k]u'$.

If $t \overset{\bullet}{\triangleright} t'$ has been derived from the second rule, $t = a\, t_1 \cdots t_n$ and $t' = t_j$ for some $j \in \{1, \ldots, n\}$. By the assumption $\vdash (\mathcal{G}, t) : \mathtt{i}$, there exists Δ such that $\vdash \mathcal{G} : \Delta$ and $\Delta \vdash a\, t_1 \cdots t_n : \mathtt{i}$. By the latter condition and the typing rules, there must exist k such that $\Delta \vdash t_k : \mathtt{i}$. The required condition therefore holds for $t'' = t_k$. □

To show the "if" direction of Theorem B.2, we extract a derivation of $\vdash (\mathcal{G}, t)$ from an infinite reduction sequence. The idea of the extraction is similar to (but simpler than) the technique we used in the completeness theorem (Theorem 4.13), and also similar to the type inference algorithm in [16]. Suppose that we are given an infinite reduction sequence:

$$S (= t_1) \overset{\bullet}{\triangleright} t_2 \overset{\bullet}{\triangleright} t_3 \overset{\bullet}{\triangleright} \cdots$$

For each prefix s of t_i (i.e., $t_i = s\, \widetilde{u}$ for some $\widetilde{u}$), we shall define the type $\delta_{s,i}$ by induction on the sort of s:

- If s has sort $\mathtt{o}$, we let $\delta_{s,i} := \mathtt{i}$.
- If s has sort $\kappa_1 \to \kappa_2$, t_i must be of the form $s\, u'\, \widetilde{u}$, where u' has sort κ_1 and $s\, u'$ has sort κ_2 Let S be the set of indices j such that u' is a prefix of t_j. Define $\delta_{s,i} := \bigwedge_{j \in S} \delta_{u',j} \to \delta_{s\, u',i}$.

We also define the type environment $\Delta_{s,i}$ by induction on the structure of s:

- If $s = a$, then $\Delta_{s,i} := \emptyset$.
- If $s = F$, then $\Delta_{s,i} := F : \delta_{s,i}$.
- If $s = s_1 s_2$, then $\Delta_{s,i} := \Delta_{s_1,i} \cup \bigcup_{j \in S} \Delta_{s_2,j}$, where S is the set of indices j such that s_2 is a prefix of t_j.

We show the following lemma.

Lemma B.5. Suppose we have an infinite sequence:

$$S (= t_1) \overset{\bullet}{\triangleright} t_2 \overset{\bullet}{\triangleright} t_3 \overset{\bullet}{\triangleright} \cdots$$

Then, $\vdash (\mathcal{G}, t_i) : \mathtt{i}$ holds for every i.

Proof. We first show that for every prefix s of t_i, $\Delta_{s,i} \vdash s : \delta_{s,i}$ is derivable by using only judgments of the form $\Delta_{s,i} \vdash u : \delta_{u,j}$, by induction on the structure of s. The base cases follow immediately from the definition of $\Delta_{s,i}$. Suppose $s = s_1 s_2$. By the induction hypothesis, we have

$$\Delta_{s_1,i} \vdash s_1 : \bigwedge_{j \in S} \delta_{s_2,j} \to \delta_{s,i} \qquad \Delta_{s_2,j} \vdash s_2 : \delta_{s_2,j} \text{ (for each } j \in S\text{)},$$

and their derivations satisfy the required property. By weakening on type environments (which is admissible by the typing rules) and the rule for application, we have $\Delta_{s,i} \vdash s_1 s_2 : \delta_{s,i}$ as required.

Let $\Delta = \bigcup_{i \in \omega} \Delta_{t_i,i}$. We show that for every $F : \delta \in \Delta$, there exists $u \in \mathcal{R}(F)$ such that $\Delta \vdash u : \delta$. This completes the proof, as $\vdash \mathcal{G} : \Delta$ and $\Delta \vdash t_i : \mathtt{i}$ for every t_i. Suppose $F : \delta \in \Delta$. By the definition of Δ, we have $\delta = \delta_{F,i} = \bigwedge_{j \in S_1} \delta_{s_1,j} \to \cdots \to \bigwedge_{j \in S_k} \delta_{s_k,j} \to \mathtt{i}$, and $t_i = F\, s_1 \cdots s_k$ for some i. Let $F \to \lambda x_1. \cdots \lambda x_k.u'$ be the rule used for the reduction step $t_i \longrightarrow t_{i+1}$. Then we have $t_{i+1} = [s_1/x_1, \ldots, s_k/x_k]u'$ and $\Delta_{t_{i+1},i+1} \vdash [s_1/x_1, \ldots, s_k/x_k]u' : \mathtt{i}$, and in the type derivation for the latter, all the judgments for s_j must be of the form $\Delta_{t_{i+1},i+1} \vdash s_j : \delta_{s_j,\ell}$ where $\ell \in S_j$. Thus, by replacing those judgments with $\Delta_{t_{i+1},i+1}, x_j : \delta_{s_j,\ell} \vdash x_j : \delta_{s_j,\ell}$ and by weakening type environments, we obtain a derivation for $\Delta_{t_{i+1},i+1}, x_1 : \bigwedge_{j \in S_1} \delta_{s_1,j}, \ldots, x_k : \bigwedge_{j \in S_k} \delta_{s_k,j} \vdash u' : \mathtt{i}$. By using the rule for abstraction, we get $\Delta_{t_{i+1},i+1} \vdash \lambda x_1. \cdots \lambda x_k.u' : \delta$. Thus, the required condition holds for $u = \lambda x_1. \cdots \lambda x_k.u'$. □

Theorem B.2 is an immediate corollary of the lemmas above.

Proof of Theorem B.2. The "only if" direction follows immediately from Lemmas B.3 and B.4. The "if" direction follows from Lemma B.5. □

For the proof of Lemma B.1 in the next subsection, we introduce some notations. We write $\Delta_{\mathcal{G}}$ for $\bigcup\{\Delta \Vdash \mathcal{G} : \Delta\}$, which is actually the largest Δ such that $\vdash \mathcal{G} : \Delta$. By abuse of notations, we often write $\Delta(F)$ for the set $\{\delta \mid F : \delta \in \Delta\}$, and write $F_1 : S_1, \ldots, F_k : S_k$ for the type environment $\{F_i : \delta \mid i \in \{1, \ldots, k\}, \delta \in S_i\}$. By Theorem B.2, $\mathcal{G}$ has an infinite reduction sequence if, and only if, $\mathtt{i} \in \Delta_{\mathcal{G}}(S)$.

B.2. **Proof of Lemma B.1.** Below, we fix a deterministic recursion scheme $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R}, S(= F_1))$, where

$$\mathcal{N} = \{F_1 : \kappa_1, \ldots, F_m : \kappa_m\}$$
$$\mathcal{R} = \{F_1 \mapsto t_1, \ldots, F_m \mapsto t_m\}$$

Suppose that there is an infinite reduction sequence:

$$\langle 1, S^0 \rangle = \langle 1, t_1 \rangle \rhd \langle 2, t_2 \rangle \rhd \langle 3, t_3 \rangle \rhd \langle 4, t_3 \rangle \rhd \cdots$$

We write $j \prec i$ if $t_j = F^i_{k_j} \widetilde{s_j}$ for some F_{k_j} and $\widetilde{s_j}$, i.e., if a non-terminal introduced at the i-th step is unfolded at the j-th step. We need to show that there is an infinite decreasing sequence:

$$\cdots \prec i_2 \prec i_1 \prec i_0 = 0.$$

The proof is by contradiction. Suppose that the relation $\prec$ is well-founded. We construct the following non-deterministic recursion scheme $\mathcal{G}' = (\mathcal{N}', \Sigma', \mathcal{R}', S^0)$:

$$\mathcal{N}' = \{F^i : \kappa \mid F : \kappa \in \mathcal{N}, i \in \omega\}$$
$$\Sigma' = \Sigma$$
$$\mathcal{R}' = \{F^i \to [F^j_1/F_1, \ldots, F^j_n/F_n]t \mid \mathcal{R}(F) = t \wedge j \prec i\}$$

By the construction of $\mathcal{G}'$, there must be an infinite reduction sequence:

$$S^0(= t_1) \stackrel{\bullet}{\rhd} t_2 \stackrel{\bullet}{\rhd} t_3 \stackrel{\bullet}{\rhd} t_4 \stackrel{\bullet}{\rhd} \cdots$$

We show that cannot be the case: every reduction sequence of S^0 in $\mathcal{G}'$ must terminate. To this end, we define another recursion scheme $\mathcal{G}'' = (\mathcal{N}'', \Sigma'', \mathcal{R}'', S^{(N)})$ by:

$$\mathcal{N}'' = \{F^{(i)} : \kappa \mid F : \kappa \in \mathcal{N}, i \in \{0, \ldots, N\}\}$$
$$\Sigma'' = \Sigma \cup \{\mathtt{e} \mapsto 0\}$$
$$\mathcal{R}'' = \{F^{(i)} \to [F^{(i-1)}_1/F_1, \ldots, F^{(i-1)}_n/F_n]t \mid \mathcal{R}(F) = t \wedge i \in \{1, \ldots, N\}\}$$
$$\cup\{F^{(0)} \to \lambda\widetilde{x}.\mathtt{e} \mid k \in \{1, \ldots, n\}\}$$

Here, $N = \Sigma_{k \in \{1,\ldots,n\}} |\{\delta \mid \delta ::_a \mathcal{N}(F_k)\}|$, i.e., the maximal size of type environments for $F_1, \ldots, F_k$ in the type system of Section B.1. Since the reduction of $S^{(N)}$ can be simulated by the simply-typed λ-term:

$$[\lambda\widetilde{x}.\mathtt{e}/F_1, \ldots, \lambda\widetilde{x}.\mathtt{e}/F_n] \underbrace{[\mathcal{R}(F_1)/F_1, \ldots, \mathcal{R}(F_n)/F_n] \cdots [\mathcal{R}(F_1)/F_1, \ldots, \mathcal{R}(F_n)/F_n]}_{N} S,$$

it follows from the strong normalization of the simply-typed λ-calculus that $\mathcal{G}''$ cannot have an infinite reduction sequence:

$$S^{(N)} \stackrel{\bullet}{\rhd} t'_2 \stackrel{\bullet}{\rhd} t'_3 \stackrel{\bullet}{\rhd} t'_4 \stackrel{\bullet}{\rhd} \cdots .$$

By Theorem B.2, we have $\mathtt{i} \notin \Delta_{\mathcal{G}''}(S^{(N)})$.

Now, let us define a function $\mathcal{F}$ on type environments by:

$$\mathcal{F}(\Delta) = \{F : \delta \mid \delta ::_a \mathcal{N}(F) \wedge \Delta \vdash \mathcal{R}(F) : \delta\}.$$

Then, by the monotonicity of $\mathcal{F}$, we have a monotonically-increasing sequence:

$$\emptyset \subseteq \mathcal{F}(\emptyset) \subseteq \mathcal{F}^2(\emptyset) \subseteq \cdots \subseteq \mathcal{F}^N(\emptyset)$$

Since the size of $\mathcal{F}^i(\emptyset)$ is at most N, we have $\mathcal{F}^{N+1}(\emptyset) = \mathcal{F}^N(\emptyset)$. By the construction of $\mathcal{G}''$, we have $\mathcal{F}^N(\emptyset) = \{F_i : \delta \mid \delta \in \Delta_{\mathcal{G}''}(F_i^{(N)})\}$.

Now, we shall show that $\Delta_{\mathcal{G}'}(F_i^j) \subseteq \mathcal{F}^N(\emptyset)(F_i)(= \Delta_{\mathcal{G}''}(F_i^{(N)}))$ for every $j \in \omega$ and $i \in \{1,\dots,k\}$. That would finish the proof, since it implies $\mathtt{i} \notin \Delta_{\mathcal{G}'}(S^0)$, and by Theorem B.2, $\mathcal{G}'$ cannot have an infinite reduction sequence.

The proof proceeds by well-founded induction on j (with respect to the well-founded relation $\prec$).

Let S be the set $\{j' \mid j' \prec j\}$. Recall that the rules for F^j are:

$$\{F^j \to [F_1^{j'}/F_1,\dots,F_n^{j'}/F_n]\mathcal{R}(F) \mid j' \in S\}.$$

We have:

$$\begin{array}{ll}
\Delta_{\mathcal{G}'}(F_i^j) & \\
= \bigcup_{j'\in S}\{\delta \mid \delta ::_a \mathcal{N}(F_i) \wedge F_1^{j'} : \Delta_{\mathcal{G}'}(F_1^{j'}),\dots,F_k^{j'} : \Delta_{\mathcal{G}'}(F_k^{j'}) \vdash [F_1^{j'}/F_1,\dots,F_n^{j'}/F_n]\mathcal{R}(F_i) : \delta\} & \\
= \bigcup_{j'\in S}\{\delta \mid \delta ::_a \mathcal{N}(F_i) \wedge F_1 : \Delta_{\mathcal{G}'}(F_1^{j'}),\dots,F_k : \Delta_{\mathcal{G}'}(F_k^{j'}) \vdash \mathcal{R}(F_i) : \delta\} & \\
 & \text{(by renaming non-terminals)} \\
\subseteq \bigcup_{j'\in S}\{\delta \mid \delta ::_a \mathcal{N}(F_i) \wedge \mathcal{F}^N(\emptyset) \vdash \mathcal{R}(F_i) : \delta\} & \text{(by induction hypothesis)} \\
= \{\delta \mid \delta ::_a \mathcal{N}(F_i) \wedge \mathcal{F}^N(\emptyset) \vdash \mathcal{R}(F_i) : \delta\} & \\
= \mathcal{F}^{N+1}(\emptyset)(F_i) & \text{(by the definition of } \mathcal{F}\text{)} \\
= \mathcal{F}^N(\emptyset)(F_i) & \text{(by } \mathcal{F}^N(\emptyset) = \mathcal{F}^{N+1}(\emptyset)\text{)}
\end{array}$$

as required. This completes the proof.

APPENDIX C. $(N-1)$-EXPTIME UPPER BOUND OF DISJUNCTIVE APT MODEL CHECKING

In [20], we considered a subclass of alternating parity tree automata called *disjunctive APT*, and showed that disjunctive APT model checking of order-N recursion schemes is $(N-1)$-EXPTIME complete. As the proof of the upper-bound relies on the development of the present article, we only sketched the proof in [20] (as Theorem 4.2), and promised to provide a more elaborate proof here.

A disjunctive APT is an alternating tree automaton whose transition function δ is disjunctive, i.e., the co-domain of δ is a subset of the positive Boolean formulas without conjunctions, given by:

$$\psi \ ::= \ \mathtt{t} \mid \mathtt{f} \mid (i,q) \mid \psi \vee \psi$$

We claim:

Theorem C.1 ([20], Theorem 4.2)**.** Let $\mathcal{G}$ be an order-N recursion scheme ($N \geq 1$) and $\mathcal{A}$ a disjunctive APT. It is decidable in $(N-1)$-EXPTIME whether $\mathcal{A}$ accepts the value tree $[\![\mathcal{G}]\!]$.

The proof is easily obtained by modifying the proof of the completeness theorem (Theorem 4.13) and the complexity argument in Section 5.

Given a disjunctive APT $\mathcal{A} = (\Sigma, Q, \delta, q_I, \Omega)$, we define the relations $\theta ::_a^d \kappa$ and $\tau ::^d \kappa$ between types and sorts by:

$$\frac{q \in Q \qquad |S_1 \cup \cdots \cup S_k| \leq 1 \qquad S_i \subseteq Q \times \mathit{codom}(\Omega) \text{ for each } i \in \{1,\dots,k\}}{(\bigwedge S_1 \to \cdots \to \bigwedge S_k \to q) ::_a^d (\underbrace{\mathtt{o} \to \cdots \to \mathtt{o}}_{k} \to \mathtt{o})}$$

$$\frac{\mathit{ord}(\kappa_1 \to \cdots \to \kappa_\ell \to \mathtt{o}) \geq 2 \qquad q \in Q \qquad \tau_i ::^d \kappa_i \text{ for each } i \in \{1,\dots,\ell\}}{(\tau_1 \to \cdots \to \tau_\ell \to q) ::_a^d (\kappa_1 \to \cdots \to \kappa_\ell \to \mathtt{o})}$$

$$\frac{\theta_i ::_a^d \kappa \text{ for each } i \in I}{\bigwedge_{i \in I} \theta_i ::^d \kappa}$$

The relation $\theta ::_a^d \kappa$ is a restriction of $\theta ::_a \kappa$ (i.e., $::_a^d$ is a strict subset of $::_a$), where in order-1 types, each argument type is empty or singleton, and only one argument type can have an element. Thus, we have $((q_0, 1) \to \top \to q_1) ::_a^d (\mathtt{o} \to \mathtt{o} \to \mathtt{o})$, but neither $((q_0, 1) \wedge (q_1, 2) \to \top \to q_1) ::_a^d (\mathtt{o} \to \mathtt{o} \to \mathtt{o})$ nor $((q_0, 1) \to (q_1, 2) \to q_1) ::_a^d (\mathtt{o} \to \mathtt{o} \to \mathtt{o})$.

We write $\Gamma ::^d \mathcal{N}$ if $\theta ::_a^d \mathcal{N}(F)$ holds for every $F : \theta \in \Gamma$. The following is the key lemma, which allows us to restrict the search space for a winning strategy for the type system.

Lemma C.2. Let $\mathcal{A}$ be a disjunctive APT, and $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ be a recursion scheme. If the tree generated by $\mathcal{G}$ is accepted by $\mathcal{A}$, then there is a winning strategy $\mathcal{W}$ for the parity game associated with $\vdash_{\mathcal{A}} \mathcal{G}$ such that $\mathit{codom}(\mathcal{W}) \subseteq \{\Gamma \mid \Gamma ::^d \mathcal{N}\}$.

Proof. By the definition of disjunctive APT, if $[\![\mathcal{G}]\!]$ is accepted by a disjunctive APT $\mathcal{A}$, then there is an accepting run-tree that is *unary* (i.e., there is at most one child for each node of the accepting run-tree). For such a unary accepting run-tree, we fix a fair rewriting sequence

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

and obtain a winning strategy $\mathcal{W}$ by using the construction in Theorem 4.13. We show that $\mathcal{W}$ satisfies the required property. As $\mathcal{W}$ returns a subset of $\Gamma_{(t,\beta,\ell)} ::^d \mathcal{N}$, it suffices to show that $T_j = C[\langle \beta, \ell, t\widetilde{s}, q \rangle]$ implies $\Gamma_{(t,\beta,\ell)} ::^d \mathcal{N}$. To this end, we show that if $T_j = C[\langle \beta, \ell, t\widetilde{s}, q \rangle]$ and t has sort κ, then $\theta_{(t,\beta,\ell)} ::_a^d \kappa$ holds, by induction on κ. Since the other cases are trivial, we discuss only the case where $\kappa = \underbrace{\mathtt{o} \to \cdots \to \mathtt{o}}_{n} \to \mathtt{o}$. In this case, $t\,\widetilde{s} = t\, s_1 \cdots s_n$ and s_i has sort $\mathtt{o}$ for each $i \in \{1, \dots, n\}$. Since the accepting run-tree is unary, there is at most one i, β', ℓ' such that $\langle \beta, \ell, t\widetilde{s}, q \rangle \gtrdot C'[\langle \beta', \ell', s_i, q' \rangle]$. Thus, $\theta_{(t,\beta,\ell)}$ is either

$$\underbrace{\top \to \cdots \to \top}_{i-1} \to (q', \Omega(C'[\,]'_q)) \to \underbrace{\top \to \cdots \to \top}_{n-i} \to q$$

(if there is such i, β', ℓ'), or $\underbrace{\top \to \cdots \to \top}_{n} \to q$ (if there is no such i, β', ℓ'). In both cases, we have $\theta_{(t,\beta,\ell)} ::_a^d \kappa$ as required.

Now, by the definition of $\Gamma_{(t,\beta,\ell)}$, $T_j = C[\langle \beta, \ell, t\widetilde{s}, q \rangle]$ implies $\Gamma_{(t,\beta,\ell)} ::^d \mathcal{N}$. This completes the proof. □

We are now ready to prove Theorem C.1.

Proof of Theorem C.1. Thanks to Lemma C.2, we can restrict type environments Θ to those that satisfy $\Theta{::}^d\mathcal{N}$ in the type inference algorithm of Section 5. Thus, in the discussion of the complexity, the upper-bound K_j of the number of types of a given order-j sort can be replaced by K'_j, where:

$$\begin{aligned}
K'_0 &= K_0 = |Q| \\
K'_1 &= |\{\theta \mid \theta ::^d_a (\underbrace{\mathsf{o} \to \cdots \to \mathsf{o}}_{A} \to \mathsf{o})\}| \\
&= |\{\underbrace{\top \to \cdots \to \top}_{A} \to q | q \in Q\}| \\
&\quad +|\{\top \to \cdots \to \top \to (q,m) \to \top \to \cdots \to \top | q \in Q, m \in \mathit{codom}(\Omega)\}| \\
&= |Q| + AM|Q| \\
K'_{j+1} &= |Q|2^{AMK'_j} \text{ (for } j \geq 1)
\end{aligned}$$

K'_N is bounded by $\mathbf{exp}_{N-1}(O(A^2M^2|Q|))$ for $N \geq 1$. For $N \geq 2$, the rest of the discussion remains the same, and the time complexity of the whole algorithm is $O(P^{1+cM}\mathbf{exp}_{N-1}(p(AM|Q|)))$ for a polynomial $p(x)$ and $c \approx \frac{1}{3}$. For $N = 1$, $|I_1| + \cdots + |I_J| \leq 1$ holds in the construction of S_F, so that the size of S_F is bounded by $K_1 \times (K_1)^1 \times K_1 = (K_1)^3$. The size of the arena for the parity game is therefore polynomial in P, A, $|Q|$, and M. Thus, if M is fixed, the complexity of the whole algorithm is also polynomial in P, A, and $|Q|$. $\square$