Higher-Order Model Checking Step by Step

Paweł Parys
Institute of Informatics, University of Warsaw, Poland

Abstract

We show a new simple algorithm that solves the model-checking problem for recursion schemes: check whether the tree generated by a given higher-order recursion scheme is accepted by a given alternating parity automaton. The algorithm amounts to a procedure that transforms a recursion scheme of order n to a recursion scheme of order $n-1$, preserving acceptance, and increasing the size only exponentially. After repeating the procedure n times, we obtain a recursion scheme of order 0, for which the problem boils down to solving a finite parity game. Since the size grows exponentially at each step, the overall complexity is n-EXPTIME, which is known to be optimal. More precisely, the transformation is linear in the size of the recursion scheme, assuming that the arity of employed nonterminals and the size of the automaton are bounded by a constant; this results in an FPT algorithm for the model-checking problem.

Our transformation is a generalization of a previous transformation of the author (2020), working for reachability automata in place of parity automata. The step-by-step approach can be opposed to previous algorithms solving the considered problem "in one step", being compulsorily more complicated.

2012 ACM Subject Classification Theory of computation → Rewrite systems

Keywords and phrases Higher-order recursion schemes, Parity automata, Model-checking, Transformation, Order reduction

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.140

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: http://arxiv.org/abs/2105.01861

Funding Work supported by the National Science Centre, Poland (grant no. 2016/22/E/ST6/00041).

Acknowledgements The author would like to thank the anonymous reviewers for their constructive comments.

1 Introduction

Recursion schemes are faithful and algorithmically manageable abstractions of the control flow of programs involving higher-order functions [19]. Such functions are nowadays widely used not only in functional programming languages such as Haskell and the OCAML family, but also in mainstream languages such as Java, JavaScript, Python, and C++. Simultaneously, the formalism of recursion schemes is equivalent via direct translations to simply-typed λY-calculus [28]. Collapsible pushdown systems [15] and ordered tree-pushdown systems [10] are other equivalent formalisms. Recursion schemes cover some other models such as indexed grammars [1] and ordered multi-pushdown automata [3].

The most celebrated algorithmic result in the analysis of recursion schemes is the decidability of the *model-checking problem* against regular properties of trees: given a recursion scheme $\mathcal{G}$ and a parity tree automaton $\mathcal{A}$, one can decide whether the tree generated by $\mathcal{G}$ is accepted by $\mathcal{A}$ [23]. This fundamental result has been reproved several times, that is, using collapsible higher-order pushdown automata [14], intersection types [20], Krivine machines [26], and it has been extended in diverse directions such as global model checking [7], logical reflection [5], effective selection [9], and a transfer theorem via models of lambda-

calculus [27]. The model-checking problem for recursion schemes of order n is complete for n-fold exponential time [23]. Despite this hardness result, the model-checking problem can be solved efficiently on multiple nontrivial examples, thanks to the development of several recursion-scheme model checkers [13, 21, 29] (including some model checkers that work only for automata models weaker than parity tree automata [17, 18, 6, 22, 25]).

In this paper, we give a new simple algorithm solving the model-checking problem for recursion schemes, mentioned above. The algorithm amounts to a procedure that transforms a recursion scheme of order n to a recursion scheme of order $n-1$, preserving acceptance, and increasing the size only exponentially. After repeating the procedure n times, we obtain a recursion scheme of order 0, for which acceptance boils down to winning a finite parity game. Since the size grows exponentially at each step, we reach the optimal overall complexity of n-fold exponential time. In a more detailed view, the complexity looks even better: the size growth is exponential only in the arity of types appearing in the recursion scheme, and in the size of the parity automaton; if these two parameters are bounded by a constant, the transformation is linear in the size of the recursion scheme. Since solving a finite parity game is FPT in the number of priorities [8], our algorithm for the the model-checking algorithm is FTP in the two parameters.[1]

The main difference between our algorithm and all the others is that we solve the problem step by step, repeatedly reducing the order by one, while most previous algorithms work "in one step", being compulsorily more complicated. The only algorithms that have been reducing the order by one, were algorithms using collapsible pushdown automata [14, 5, 9]. Notice, however, that these algorithms: first, are very technical; second, are contained only in unpublished appendices and in an arXiv paper [4]; third, if we want to use them for recursion schemes, it is necessary to employ a (nontrivial) translation from recursion schemes to collapsible pushdown automata [15, 28, 9]. A reduction of order was also possible for a subclass of recursion schemes, called *safe* recursion schemes [16], but it was not known how to extend it to all recursion schemes.

The transformation presented in this paper generalizes of a previous transformation of the author [24], working for reachability automata in place of parity automata. It has also a close relationship with a transformation given by Asada and Kobayashi [2].

2 Preliminaries

For a number $k \in \mathbb{N}$ we write $[k]$ for $\{1, \dots, k\}$. For any relation $\longrightarrow$ we write $\longrightarrow^*$ for the reflexive transitive closure of $\longrightarrow$.

For a function Z we write $Z[z \mapsto r]$ to denote the function that maps z to r while all other elements of the domain of Z are mapped as in Z. Likewise, we write $Z[z_i \mapsto r_i \mid i \in I]$ to denote the function that maps z_i to r_i for all $i \in I$, while all other elements of the domain of Z are mapped as in Z. We also use this notation without the "Z" part, for a function Z with empty domain.

Recursion schemes. The set of *(simple) types* is constructed from a unique ground type o using a binary operation $\to$; namely o is a type, and if α and β are types, so is $\alpha \to \beta$. By convention, $\to$ associates to the right, that is, $\alpha \to \beta \to \gamma$ is understood as $\alpha \to (\beta \to \gamma)$.

[1] This is not new. Actually, most previous algorithms reduce the model-checking problem to the problem of solving a parity game whose size is polynomial (for a polynomial of a fixed degree, for some algorithms just linear) in the size of the input, assuming that the arity of types appearing in the recursion scheme and the size of the parity automaton are fixed. Thus, only the method introduced by us is new, not the complexity results.

We often abbreviate $\underbrace{\alpha \to \cdots \to \alpha}_{\ell} \to \beta$ as $\alpha^{\ell} \to \beta$. The *order* of a type α, denoted $\mathsf{ord}(\alpha)$, is defined by induction: $\mathsf{ord}(\alpha_1 \to \cdots \to \alpha_k \to \mathsf{o}) = \max(\{0\} \cup \{\mathsf{ord}(\alpha_i) + 1 \mid i \in [k]\})$; for example $\mathsf{ord}(\mathsf{o}) = 0$, $\mathsf{ord}(\mathsf{o} \to \mathsf{o} \to \mathsf{o}) = 1$, and $\mathsf{ord}((\mathsf{o} \to \mathsf{o}) \to \mathsf{o}) = 2$.

Having a set of typed nonterminals $\mathcal{X}$, a set of typed variables $\mathcal{Y}$, and a set of symbols Σ, *terms* over $(\mathcal{X}, \mathcal{Y}, \Sigma)$ are defined by induction:

- nonterminal: every nonterminal $X \in \mathcal{X}$ of type α is a term of type α;
- variable: every variable $y \in \mathcal{Y}$ of type α is a term of type α;
- node constructor: if $K_1, \ldots, K_k$ are terms of type o and $a \in \Sigma$, then $\langle a, K_1, \ldots, K_k \rangle$ is a term of type o;
- application: if K is a term of type $\alpha \to \beta$, and L is a term of type α, then $K\,L$ is a term of type β.

The type of a term K is denoted $\mathsf{tp}(K)$. The order of a term K, written $\mathsf{ord}(K)$, is defined as the order of its type.

A *(higher-order) recursion scheme* is a tuple $\mathcal{G} = (\mathcal{X}, X_0, \Sigma, \mathcal{R})$, where $\mathcal{X}$ is a finite set of typed nonterminals, and $X_0 \in \mathcal{X}$ is a *starting nonterminal* of type o, and Σ is a finite set of symbols (called an *alphabet*), and $\mathcal{R}$ is a function assigning to every nonterminal $X \in \mathcal{X}$ a *rule* of the form $X\,y_1\,\ldots\,y_k \to R$, where $\mathsf{tp}(X) = (\mathsf{tp}(y_1) \to \cdots \to \mathsf{tp}(y_k) \to \mathsf{o})$, and R is a term of type o over $(\mathcal{X}, \{y_1, \ldots, y_k\}, \Sigma)$. The order of a recursion scheme, $\mathsf{ord}(\mathcal{G})$, is defined as the maximum of orders of its nonterminals.

Having a recursion scheme $\mathcal{G} = (\mathcal{X}, X_0, \Sigma, \mathcal{R})$, for every set of variables $\mathcal{Y}$ we define a *reduction relation* $\longrightarrow_{\mathcal{G}}$ between terms over $(\mathcal{X}, \mathcal{Y}, \Sigma)$ as the least relation such that

- $X\,K_1\,\ldots\,K_k \longrightarrow_{\mathcal{G}} R[K_1/y_1, \ldots, K_k/y_k]$ if the rule for X is $X\,y_1\,\ldots\,y_k \to R$, where $R[K_1/y_1, \ldots, K_k/y_k]$ denotes the term obtained from R by substituting K_i for y_i for all $i \in [k]$.

A (potentially infinite) *tree* over an alphabet Σ is defined by coinduction: every tree over Σ is of the form $\langle a, T_1, \ldots, T_k \rangle$, where $a \in \Sigma$ and $T_1, \ldots, T_k$ are again trees over Σ (for an introduction to coinductive definitions and proofs see, e.g., Czajka [12]). We employ the usual notions of nodes, children, branches, etc. Formally, we can define nodes as sequences of natural numbers; then for a tree $T = \langle a, T_1, \ldots, T_k \rangle$, the empty sequence $()$ is a node of T labeled by a, and any longer sequence $(i_1, i_2, \ldots, i_n)$ is a node of T labeled by b if $i_1 \in [k]$ and $(i_2, \ldots, i_n)$ is a node of T_{i_1} labeled by b. For a tree T and its node v, we write $T{\restriction_v}$ for the subtree of T starting at v.

Again by coinduction, we define the tree *generated* by a recursion scheme $\mathcal{G} = (\mathcal{X}, X_0, \Sigma, \mathcal{R})$ from a term M of type o (over $(\mathcal{X}, \varnothing, \Sigma)$), denoted $\mathsf{BT}_{\mathcal{G}}(M)$:

- if $M \longrightarrow_{\mathcal{G}}^{*} \langle a, K_1, \ldots, K_k \rangle$, then $\mathsf{BT}_{\mathcal{G}}(M) = \langle a, \mathsf{BT}_{\mathcal{G}}(K_1), \ldots, \mathsf{BT}_{\mathcal{G}}(K_k) \rangle$;
- otherwise, $\mathsf{BT}_{\mathcal{G}}(M) = \langle \omega \rangle$ for a special symbol $\omega \notin \Sigma$.

The tree generated by $\mathcal{G}$ (without mentioning a term), denoted $\mathsf{BT}(\mathcal{G})$, is defined as $\mathsf{BT}_{\mathcal{G}}(X_0)$.

Parity games. As already said, in the model-checking problem we are given a recursion scheme $\mathcal{G}$ and an alternating parity automaton $\mathcal{A}$, and we are asked whether the tree $T_{\mathcal{G}}$ generated by $\mathcal{G}$ is accepted by $\mathcal{A}$. One can, however, create a product of $\mathcal{G}$ and $\mathcal{A}$, which is a recursion scheme $\mathcal{G}_{\mathcal{A}}$ generating the tree of all possible runs of $\mathcal{A}$ on $T_{\mathcal{G}}$. This tree is a parity game; the game is won by Eve if and only if $\mathcal{A}$ accepts $T_{\mathcal{G}}$ (see Appendix A of the full version for more details). Due to this reduction, it is enough to work with recursion schemes generating parity games, and consider the problem of finding a winner in such games.

For every $d \in \mathbb{N}_+$ we consider the alphabet $\Sigma_d = \{\mathrm{Adam}, \mathrm{Eve}\} \times [d]$. A *parity tree* is a tree over Σ_d where every node has at least one child. A *parity recursion scheme* is a recursion scheme generating a parity tree (in particular the generated tree cannot have nodes

without children, including ω-labeled nodes). For a node labeled by $(\wp, p) \in \Sigma_d$, we say that it *belongs* to the player $\wp$, and that it has *priority* p. For trees and terms we write $\langle \wp, p, K_1, \ldots, K_k \rangle$ instead of $\langle (\wp, p), K_1, \ldots, K_k \rangle$, avoiding excessive brackets.

A branch ξ in a parity tree T is *won by Eve* (*Adam*) if the greatest priority appearing infinitely often on ξ is even (odd, respectively). A *strategy* ρ of a player $\wp \in \{\text{Adam}, \text{Eve}\}$ in a parity tree T is a function that assigns numbers to nodes of T belonging to the player $\wp$; if a node v has k children, we require that $\rho(v) \in [k]$. A branch ξ *agrees* with ρ if for every node v on ξ that belongs to $\wp$, the next node of ξ is the $\rho(v)$-th child of v. A strategy ρ of $\wp$ is *winning* if all branches that agree with ρ are winning for $\wp$. Finally, $\wp$ *wins* in T if $\wp$ has a winning strategy in T; otherwise $\wp$ *loses* in T. It is a standard result that in every parity tree T exactly one of the players wins.

It is useful to consider the following order $\preceq$ on positive natural numbers (priorities): $\cdots \preceq 5 \preceq 3 \preceq 1 \preceq 2 \preceq 4 \preceq 6 \preceq \ldots$ (first we have odd numbers in the reversed order, and then positive even numbers). We use the words *worse* and *better* to say that a priority is, respectively, earlier or later in this order. The intuition is that while playing a parity game, Eve always prefers to see better priorities.

3 Transformation

In this section we present a transformation, called *order-reducing transformation*, resulting in the main theorem of this paper:

▶ **Theorem 3.1.** *For any $n \geq 1$, there exists a transformation from order-n parity recursion schemes to order-$(n-1)$ parity recursion schemes, and a polynomial p_n such that, for any order-n parity recursion scheme $\mathcal{G}$, the winner in the tree generated by the resulting recursion scheme $\mathcal{G}^\dagger$ is the same as in the tree generated by $\mathcal{G}$, and $|\mathcal{G}^\dagger| \leq 2^{p_n(|\mathcal{G}|)}$.*

Intuitions. Let us first present intuitions behind our transformation. While reducing the order, we have to replace, in particular, order-1 functions by order-0 terms. Consider for example a tree T generated from a term $K\,L$ of type o, where K has type $\mathsf{o} \to \mathsf{o}$. Essentially, T consists of a context C_K, generated by K, where the tree T_L generated by L is inserted in some "holes". Instead of playing in T, we propose the following modification of the game. At the beginning, we ask Eve a question: how is she going to reach subtrees T_L while playing in T? She may declare that, according to her winning strategy,

- she is able to ensure that the greatest priority seen before reaching T_L will not be worse than r, for some number r of her choice, or
- she will not reach subtrees T_L at all, which amounts to choosing for r an even number greater than d, say $r = 2d$.

Then, we ask Adam if he believes in this declaration. If so, we simply read the declared worst-case priority r, and we continue playing in T_L (this possibility is unavailable for Adam, if Eve declared that she will not visit T_L). Otherwise, we check the declaration: we start playing in C_K; while reaching a place where T_L should be placed, Eve immediately wins (loses) if her declaration is fulfilled (not fulfilled, respectively).

We can see that such a modification of the game (even applied in infinitely many places of the considered tree) does not change the winner. A subtle point is that, in the modified game, Eve has to make a declaration on the priority r before actually starting the game in the tree generated from $K\,L$, and it is not completely obvious why the need for the declaration introduces no disadvantage for Eve. Nevertheless, for a fixed Eve's winning strategy, the worst greatest priority seen before reaching T_L is fixed, so that Eve can declare it as r.

In the transformation, we change the order-1 term K into several order-0 terms: K_r for $r \in \{1, \ldots, d, 2d\}$ (where d is a bound on priorities in the considered parity recursion scheme $\mathcal{G}$). These terms generate trees of the same shape as the context C_K generated by K but with some fixed trees substituted in place of the holes of C_K (where originally trees generated by the argument L were attached). The generated trees correspond to particular declarations made by Eve, as described above. Namely, we consider some fixed trees $\bot$ and $\top$ in which Eve loses and wins, respectively. Then, in the tree generated by K_r, the tree $\top$ is placed in holes such that the greatest priority on the path from the root to the hole is not worse than r, and the tree $\bot$ is placed in the remaining holes. In particular, the tree $\bot$ is placed in all holes of the tree generated by K_{2d}, because all priorities actually appearing in the tree are worse than $2d$. Finally, we replace $K\,L$ by $\langle \mathrm{Eve}, 1, K_1^L, K_2^L, \ldots, K_d^L, K_{2d}\rangle$, where $K_r^L = \langle \mathrm{Adam}, 1, K_r, \langle \mathrm{Eve}, r, L\rangle\rangle$. In this way we realize the modified game described above: first Eve chooses a declaration r and then Adam either proceed to K_r or to L after seeing priority r (the latter possibility is disabled for $r = 2d$). The priority 1 of the newly created tree nodes should be seen as a neutral priority; higher priorities visited later will be more important anyway.

When a term K of order 1 takes multiple arguments (instead of one argument L), we proceed in the same way, allowing Eve to make declarations for each of the arguments.

While applying the above-described transformation to recursion schemes, it is possible that the term K considered above contains some nonterminals or variables. Then, in order to realize the transformation, we need to create multiple copies of these nonterminals and variables, corresponding to particular declarations of Eve.

For example, say that in a recursion scheme we have (among others) the following two rules:

$$\mathsf{X} \to \mathsf{Y}\,\mathsf{Z},$$
$$\mathsf{Y}\,\mathsf{z} \to \langle \mathrm{Eve}, 1, \mathsf{z}, \langle \mathrm{Eve}, 2, \mathsf{z}\rangle\rangle.$$

Here X and Z are of type o, and Y is of type $\mathsf{o} \to \mathsf{o}$, so $\mathsf{Y}\,\mathsf{Z}$ is an application that should be replaced by the transformation. Assuming $d = 2$, we should obtain the following rules:

$$\mathsf{X}' \to \langle \mathrm{Eve}, 1, \langle \mathrm{Adam}, 1, \mathsf{Y}_1, \langle \mathrm{Eve}, 1, \mathsf{Z}'\rangle\rangle, \langle \mathrm{Adam}, 1, \mathsf{Y}_2, \langle \mathrm{Eve}, 2, \mathsf{Z}'\rangle\rangle, \mathsf{Y}_4\rangle,$$
$$\mathsf{Y}_1 \to \langle \mathrm{Eve}, 1, \overline{\top}, \langle \mathrm{Eve}, 2, \overline{\top}\rangle\rangle,$$
$$\mathsf{Y}_2 \to \langle \mathrm{Eve}, 1, \underline{\bot}, \langle \mathrm{Eve}, 2, \overline{\top}\rangle\rangle,$$
$$\mathsf{Y}_4 \to \langle \mathrm{Eve}, 1, \underline{\bot}, \langle \mathrm{Eve}, 2, \underline{\bot}\rangle\rangle,$$

where $\underline{\bot}$ and $\overline{\top}$ are nonterminals from which the trees $\bot$ and $\top$ (in which Eve loses and wins, respectively) are generated.

Another possibility is that in the original recursion scheme we have $\mathsf{y}\,\mathsf{Z}$ instead of $\mathsf{Y}\,\mathsf{Z}$:

$$\mathsf{S} \to \mathsf{T}\,\mathsf{Y},$$
$$\mathsf{T}\,\mathsf{y} \to \mathsf{y}\,\mathsf{Z}.$$

Then, the single parameter y gets transformed into three parameters:

$$\mathsf{S}' \to \mathsf{T}'\,\mathsf{Y}_1\,\mathsf{Y}_2\,\mathsf{Y}_4,$$
$$\mathsf{T}'\,\mathsf{y}_1\,\mathsf{y}_2\,\mathsf{y}_4 \to \langle \mathrm{Eve}, 1, \langle \mathrm{Adam}, 1, \mathsf{y}_1, \langle \mathrm{Eve}, 1, \mathsf{Z}'\rangle\rangle, \langle \mathrm{Adam}, 1, \mathsf{y}_2, \langle \mathrm{Eve}, 2, \mathsf{Z}'\rangle\rangle, \mathsf{y}_4\rangle.$$

Formal definition. We now formalize the above intuitions. Fix a parity recursion scheme $\mathcal{G} = (\mathcal{X}, X_0, \Sigma_d, \mathcal{R})$; in particular fix a bound d on priorities appearing in $\mathcal{G}$.

A set D_d of Eve's *declarations* is defined as $D_d = \{1, \ldots, d, 2d\}$. For a priority $p \in [d]$ and a declaration $r \in D_d$ we define a *shifted* declaration $r{\restriction_p}$ (obtained from r after seeing priority p):

$$r{\restriction_p} = \begin{cases} p+1 & \text{if } p \text{ is odd and } p > r, \\ p-1 & \text{if } p \text{ is even and } p \geq r, \\ r & \text{otherwise.} \end{cases}$$

We remark that the same definition appears in Tsukada and Ong [30] (where shifts are called left-residuals); a slightly different representation is present also in Salvati and Walukiewicz [26] (with declarations called residuals and shifts called liftings).

The *leader* ("most important priority") of a sequence of priorities π is the greatest priority appearing in π, or 1 if π is empty. A sequence of priorities π *fulfils* a declaration $r \in D_d$ if r is worse or equal than the leader of π (where "worse" refers to the $\preceq$ order defined in Section 2). For example, $1, 4, 2$, and $1, 1, 1$, both fulfil 3, but $1, 5, 4$ does not. The empty sequence fulfils r exactly when r is odd. No sequence of priorities from $[d]$ fulfils $2d$. The following lemma is obtained by a direct analysis (see Appendix B of the full version):

▶ **Lemma 3.2.** *A sequence of priorities $p_1, p_2, \ldots, p_k \in [d]$ fulfils a declaration $r \in D_d$ if and only if $p_2, \ldots, p_k$ fulfils $r{\restriction_{p_1}}$.*

Having a type, we are interested in cutting off its suffix of order 1. Thus, we use the notation $\alpha_1 \to \cdots \to \alpha_k \Rightarrow \mathsf{o}^\ell \to \mathsf{o}$ for a type $\alpha_1 \to \cdots \to \alpha_k \to \mathsf{o}^\ell \to \mathsf{o}$ such that either $k = 0$ or $\alpha_k \neq \mathsf{o}$. Notice that every type α can be uniquely represented in this form. We remark that some among the types $\alpha_1, \ldots, \alpha_{k-1}$ (but not α_k) may be o. For a type α we write $\mathsf{gar}(\alpha)$ ("ground arity") for the number ℓ for which we can write $\alpha = (\alpha_1 \to \cdots \to \alpha_k \Rightarrow \mathsf{o}^\ell \to \mathsf{o})$; we also extend this to terms: $\mathsf{gar}(M) = \mathsf{gar}(\mathsf{tp}(M))$.

We transform terms of type α to terms of type $\alpha^{\dagger_d}$, which is defined by induction:

$$(\alpha_1 \to \cdots \to \alpha_k \Rightarrow \mathsf{o}^\ell \to \mathsf{o})^{\dagger_d} = \left((\alpha_1^{\dagger_d})^{|D_d|^{\mathsf{gar}(\alpha_1)}} \to \cdots \to (\alpha_k^{\dagger_d})^{|D_d|^{\mathsf{gar}(\alpha_k)}} \to \mathsf{o} \right).$$

Thus, we remove all trailing order-0 arguments, and we multiplicate (and recursively transform) remaining arguments. The number of copies depends on the bound d on priorities appearing in the considered parity recursion scheme.

For a finite set S, we write D_d^S for the set of functions $A \colon S \to D_d$. Moreover, we assume some fixed order on functions in D_d^S, and we write $P\,(Q_A)_{A \in D_d^S}$ for an application $P\,Q_{A_1} \ldots Q_{A_{|D_d|^{|S|}}}$, where $A_1, \ldots, A_{|D_d|^{|S|}}$ are all the functions from D_d^S listed in the fixed order. The only function in $D_d^\varnothing$ is denoted $\varnothing$.

For every variable y and for every function $A \in D_d^{[\mathsf{gar}(y)]}$ we consider a variable $y_A^{\dagger_d}$ of type $(\mathsf{tp}(y))^{\dagger_d}$. Likewise, for every nonterminal X of $\mathcal{G}$ and for every function $A \in D_d^{[\mathsf{gar}(X)]}$ we consider a nonterminal $X_A^{\dagger_d}$ of type $(\mathsf{tp}(X))^{\dagger_d}$. As the new set of nonterminals we take $\mathcal{X}^{\dagger_d} = \{X_A^{\dagger_d} \mid X \in \mathcal{X}, A \in D_d^{[\mathsf{gar}(X)]}\} \cup \{\underline{\bot}, \overline{\top}\}$.

We now define a function tr_d transforming terms. Its value $\mathsf{tr}_d(A, Z, M)$ is defined when M is a term over $(\mathcal{X}, \mathcal{Y}, \Sigma_d)$ for some set of variables $\mathcal{Y}$, and $A \in D_d^{[\mathsf{gar}(M)]}$, and $Z \colon \mathcal{Y} \rightharpoonup D_d$ is a partial function such that $\mathrm{dom}(Z)$ contains only variables of type o. The intention is that A specifies Eve's declarations for trailing order-0 arguments, and Z specifies them for order-0 variables (among those in $\mathrm{dom}(Z)$). The transformation is defined by induction on the structure of M, as follows:

(1) $\mathsf{tr}_d(A, Z, X) = X_A^{\dagger_d}$ for $X \in \mathcal{X}$;
(2) $\mathsf{tr}_d(A, Z, y) = y_A^{\dagger_d}$ for $y \in \mathcal{Y} \setminus \mathrm{dom}(Z)$;
(3) $\mathsf{tr}_d(\varnothing, Z, z) = \overline{\top}$ if $Z(z)$ is odd;
(4) $\mathsf{tr}_d(\varnothing, Z, z) = \underline{\bot}$ if $Z(z)$ is even;
(5) $\mathsf{tr}_d(\varnothing, Z, \langle \wp, p, K_1, \dots, K_k\rangle) = \langle \wp, p, \mathsf{tr}_d(\varnothing, Z{\restriction_p}, K_1), \dots, \mathsf{tr}_d(\varnothing, Z{\restriction_p}, K_k)\rangle$, where $Z{\restriction_p}$ is the function defined by $Z{\restriction_p}(z) = (Z(z)){\restriction_p}$ for all $z \in \mathrm{dom}(Z)$;
(6) $\mathsf{tr}_d(A, Z, K\,L) = \langle \mathrm{Eve}, 1, K_1^L, K_2^L, \dots, K_d^L, K_{2d}\rangle$ if $\mathsf{tp}(K) = (\mathsf{o}^{\ell+1} \to \mathsf{o})$, where $K_r^L = \langle \mathrm{Adam}, 1, K_r, \langle \mathrm{Eve}, r, \mathsf{tr}_d(\varnothing, Z{\restriction_r}, L)\rangle\rangle$ for $r \in [d]$ and $K_r = \mathsf{tr}_d(A[\ell+1 \mapsto r], Z, K)$ for $r \in D_d$;
(7) $\mathsf{tr}_d(A, Z, K\,L) = (\mathsf{tr}_d(A, Z, K))\,(\mathsf{tr}_d(B, Z, L))_{B \in D_d^{[\mathsf{gar}(L)]}}$ if $\mathsf{tp}(K) = (\alpha_1 \to \cdots \to \alpha_k \Rightarrow \mathsf{o}^{\ell} \to \mathsf{o})$ with $k \geq 1$.

In Cases (3), (4), and (5) the term is of type o, so the "A" argument is necessarily $\varnothing$ (a function with an empty domain).

For every rule $X\,y_1 \dots y_k\,z_1 \dots z_\ell \to R$ in $\mathcal{R}$, where $\ell = \mathsf{gar}(X)$, and for every function $A \in D_d^{[\ell]}$, to $\mathcal{R}^{\dagger_d}$ we take the rule

$$X_A^{\dagger_d}\,(y_{1,B}^{\dagger_d})_{B \in D_d^{[\mathsf{gar}(y_1)]}} \dots (y_{k,B}^{\dagger_d})_{B \in D_d^{[\mathsf{gar}(y_k)]}} \to \mathsf{tr}_d(\varnothing, [z_i \mapsto A(\ell+1-i) \mid i \in [\ell]], R).$$

In the function A it is more convenient to count arguments from right to left (then we do not need to shift the domain in Case (6) above), but it is more natural to have variables $z_1, \dots, z_\ell$ numbered from left to right; this is why in the rule for $X_A^{\dagger_d}$ we assign to z_i the value $A(\ell+1-i)$, not $A(i)$. Additionally, in $\mathcal{R}^{\dagger_d}$ we have rules $\underline{\bot} \to \langle \mathrm{Eve}, 1, \underline{\bot}\rangle$ and $\overline{\top} \to \langle \mathrm{Eve}, 2, \overline{\top}\rangle$. Then Eve loses (wins) in the tree $\bot$ ($\top$) generated by $\mathcal{G}^\dagger$ from $\underline{\bot}$ ($\overline{\top}$, respectively).

Finally, the resulting recursion scheme $\mathcal{G}^\dagger$ is $(\mathcal{X}^{\dagger_d}, X_{0,\varnothing}^{\dagger_d}, \Sigma_d, \mathcal{R}^{\dagger_d})$. This finishes the definition of the transformation. In the next section we analyze its complexity, and in Section 5 we justify its correctness.

▶ Remark 3.3. Let us briefly compare our transformation with a transformation by Broadbent et al. [4] reducing the order of a collapsible pushdown automaton by one while preserving the winner of the generated parity game. Although their transformation seems technically more complicated, its overall idea is quite similar to what we do in this paper. Their transformation is split into three independent steps. First, they make the automaton "rank-aware", which means that it knows what was the highest priority visited between creation of a collapse link and its usage. This corresponds to adding the parameters A and Z to our transformation, so that we know whether a declaration is fulfilled when a variable z is used. Second, they eliminate collapse links of order n, which in our case corresponds to removing trailing arguments of order 0 and introducing the gadget asking Eve for a declaration. Third, they reduce the order of the automaton by one, which we also do for recursion schemes.

4 Complexity

In this section we analyze complexity of our transformation. First, we formally define the *size* of a recursion scheme. The size of a term is defined by induction on its structure:

$$|X| = |y| = 1, \qquad |K\,L| = 1 + |K| + |L|,$$
$$|\langle a, K_1, \dots, K_k\rangle| = 1 + |K_1| + \cdots + |K_k|.$$

Then $|\mathcal{G}|$, the size of $\mathcal{G}$, is defined as the sum of $|R| + k$ over all rules $X\,y_1 \dots y_k \to R$ of $\mathcal{G}$. In Asada and Kobayashi [2] such a size is called *Curry-style* size; it does not include sizes of types of employed variables.

We say that a type α *appears in the definition* of a type β if either $\alpha = \beta$, or $\beta = (\beta_1 \to \beta_2)$ and α appears in the definition of β_1 or of β_2. We write $A_{\mathcal{G}}$ for the largest arity of types appearing in the definition of types of nonterminals in a recursion scheme $\mathcal{G}$. Notice that types of other objects used in $\mathcal{G}$, namely variables and subterms of right-hand sides of rules, appear in the definition of types of nonterminals, hence their arity is also bounded by $A_{\mathcal{G}}$. It is reasonable to consider large recursion schemes, consisting of many rules, where simultaneously the maximal arity $A_{\mathcal{G}}$ is respectively small.

While the exponential bound mentioned in Theorem 3.1 is obtained by applying the order-reducing transformation to an arbitrary parity recursion scheme, the complexity becomes slightly better if we first apply a preprocessing step. This is in particular necessary, if we want to obtain linear dependence in the size of $\mathcal{G}$ (and exponential only in the maximal arity $A_{\mathcal{G}}$). The preprocessing, making sure that the recursion scheme is in a *simple form* (defined below), amounts to splitting large rules into multiple smaller rules. A similar preprocessing is present already in prior work [19, 2, 11, 24].

An *application depth* of a term R is defined as the maximal number of applications on a single branch in R, where a compound application $K\,L_1\,\ldots\,L_k$ counts only once. More formally, we define by induction:

$$\mathsf{ad}(\langle a, K_1, \ldots, K_k\rangle) = \max\{\mathsf{ad}(K_i) \mid i \in [k]\},$$
$$\mathsf{ad}(X\,K_1\,\ldots\,K_k) = \mathsf{ad}(y\,K_1\,\ldots\,K_k) = \max(\{0\} \cup \{\mathsf{ad}(K_i) + 1 \mid i \in [k]\}).$$

We say that a recursion scheme $\mathcal{G}$ is in a *simple form* if the right-hand side of each its rule has application depth at most 2. We have the following:

▶ **Lemma 4.1** ([24, Lemma 4.1])**.** *For every recursion scheme $\mathcal{G}$ there exists a recursion scheme $\mathcal{G}'$ being in a simple form, generating the same tree as $\mathcal{G}$, and such that $\mathsf{ord}(\mathcal{G}') = \mathsf{ord}(\mathcal{G})$, and $A_{\mathcal{G}'} \le 2A_{\mathcal{G}}$, and $|\mathcal{G}'| = \mathcal{O}(A_{\mathcal{G}} \cdot |\mathcal{G}|)$. The recursion scheme $\mathcal{G}'$ can be created in time linear in its size.*

We now state and prove the main lemma of this section:

▶ **Lemma 4.2.** *For every parity recursion scheme $\mathcal{G} = (\mathcal{X}, X_0, \Sigma_d, \mathcal{R})$ in a simple form, the recursion scheme $\mathcal{G}^\dagger$ (i.e., the result of the order-reducing transformation) is also in a simple form, and $\mathsf{ord}(\mathcal{G}^\dagger) = \max(0, \mathsf{ord}(\mathcal{G}) - 1)$, and $A_{\mathcal{G}^\dagger} \le A_{\mathcal{G}} \cdot (d+1)^{A_{\mathcal{G}}}$, and $|\mathcal{G}^\dagger| = \mathcal{O}(|\mathcal{G}| \cdot (d+1)^{5 \cdot A_{\mathcal{G}}})$. Moreover, $\mathcal{G}^\dagger$ can be created in time linear in its size.*

Proof. The part about the running time is obvious. It is also easy to see by induction that $\mathsf{ord}(\alpha^{\dagger_d}) = \max(0, \mathsf{ord}(\alpha) - 1)$. It follows that the order of the recursion scheme satisfies the same equality, because nonterminals of $\mathcal{G}^\dagger$ have type $\alpha^{\dagger_d}$ for α being the type of a corresponding nonterminal of $\mathcal{G}$.

Recall that in the type $\alpha^{\dagger_d}$ obtained from $\alpha = (\alpha_1 \to \cdots \to \alpha_k \to \mathsf{o})$, every α_i either disappears or becomes (transformed and) repeated $|D_d|^{\mathsf{gar}(\alpha_i)}$ times, that is, at most $(d+1)^{A_{\mathcal{G}}}$ times. This implies the inequality concerning $A_{\mathcal{G}^\dagger}$.

Every compound application can be written as $f\,K_1\,\ldots\,K_k\,L_1\,\ldots\,L_\ell$, where f is a nonterminal or a variable, and $\ell = \mathsf{gar}(f)$. In such a term, every K_i (after being transformed) gets repeated $|D_d|^{\mathsf{gar}(K_i)}$ times, that is, at most $(d+1)^{A_{\mathcal{G}}}$ times. Then, for every L_i we replicate the outcome $d+1$ times, and we append a small prefix; this replication happens ℓ times (and $\ell \le A_{\mathcal{G}}$). In consequence, we easily see by induction that while transforming a term of application depth c, its size gets multiplicated by at most $O((d+1)^{2c \cdot A_{\mathcal{G}}})$. Moreover, every nonterminal X is repeated $|D_d|^{\mathsf{gar}(X)}$ times, that is, at most $(d+1)^{A_{\mathcal{G}}}$ times. Because the application depth of right-hand sides of rules is at most 2, this bounds the size of the new recursion scheme by $\mathcal{O}(|\mathcal{G}| \cdot (d+1)^{5 \cdot A_{\mathcal{G}}})$.

Looking again at the above description of the transformation, we can notice that the application depth cannot grow; in consequence the property of being in a simple form is preserved. ◀

Thus, if we want to check whether Eve wins in the tree generated by a parity recursion scheme $\mathcal{G}$ of order n, we can first convert $\mathcal{G}$ to a simple form, and then apply the order-reducing transformation n times. This gives us a parity recursion scheme of order 0, which can be seen as a finite parity game with d priorities. Such a game can be solved in time $O(N^4 \cdot 2^d)$, where N is its size [8]. Thus, by Lemmata 4.1 and 4.2, the whole algorithm works in time n-fold exponential in $A_{\mathcal{G}}$ and d, and polynomial (quartic) in $|\mathcal{G}|$.

If $\mathcal{G}$ is created as a product of a recursion scheme $\mathcal{H}$ and an alternating parity automaton $\mathcal{A}$, the running time is n-fold exponential in $A_{\mathcal{H}}$ and $|\mathcal{A}|$, and quartic in $|\mathcal{H}|$ (cf. Appendix A of the full version).

5 Correctness

In this section we finish a proof of Theorem 3.1 by showing that the winner in the tree generated by the recursion scheme $\mathcal{G}^\dagger$ resulting from transforming a recursion scheme $\mathcal{G}$ is the same as in the tree generated by the original recursion scheme $\mathcal{G}$. Our proof consists of three parts. First, we show that reductions performed by $\mathcal{G}$ can be reordered, so that we can postpone substituting for (trailing) variables of order 0. To store such postponed substitutions, called *explicit substitutions*, we introduce *extended trees*. Second, we show that such reordered reductions in $\mathcal{G}$ are in a direct correspondence with reductions in $\mathcal{G}^\dagger$. Finally, we show how winning strategies of particular players from the tree generated by $\mathcal{G}^\dagger$ can be transferred to the tree generated by $\mathcal{G}$.

Extended trees and terms. In the sequel, trees and terms defined previously are sometimes called non-extended trees and non-extended terms, in order to distinguish them from extended trees and extended terms defined below. Having a set $\mathcal{Z}$ of variables of type o and a set of symbols Σ, (potentially infinite) *extended trees* over $(\mathcal{Z}, \Sigma)$ are defined by coinduction: every extended tree over $(\mathcal{Z}, \Sigma)$ is of the form either

- $\langle a, T_1, \dots, T_k \rangle$, where $a \in \Sigma$ and $T_1, \dots, T_k$ are again extended trees over Σ, or
- z for some variable $z \in \mathcal{Z}$, or
- $T(\!|U/z|\!)$, where $z \notin \mathcal{Z}$ is a variable of type o, and T is an extended tree over $(\mathcal{Z} \cup \{z\}, \Sigma)$, and U is an extended tree over $(\mathcal{Z}, \Sigma)$.

The construction of the form $T(\!|U/z|\!)$ is called an *explicit substitution*. Intuitively, it denotes the tree obtained by substituting U for z in T. Notice that the variable z being free in T becomes bound in $T(\!|U/z|\!)$.

Likewise, having a set of typed nonterminals $\mathcal{X}$, a set $\mathcal{Z}$ of variables of type o, and a set of symbols Σ, *extended terms* over $(\mathcal{X}, \mathcal{Z}, \Sigma)$ are defined by induction:

- if $z \notin \mathcal{Z}$ is a variable of type o, and E is an extended term over $(\mathcal{X}, \mathcal{Z} \cup \{z\}, \Sigma)$, and L is a non-extended term of type o over $(\mathcal{X}, \mathcal{Z}, \Sigma)$, then $E(\!|L/z|\!)$ is an extended term over $(\mathcal{X}, \mathcal{Z}, \Sigma)$;
- every non-extended term of type o over $(\mathcal{X}, \mathcal{Z}, \Sigma)$ is an extended term over $(\mathcal{X}, \mathcal{Z}, \Sigma)$.

Notice that explicit substitutions can be placed anywhere inside an extended tree, while in an extended term they are allowed only to surround a non-extended term.

Of course an extended tree over $(\mathcal{Z}, \Sigma)$ can be also seen as an extended tree over $(\mathcal{Z}', \Sigma)$, where $\mathcal{Z}' \supseteq \mathcal{Z}$; likewise for extended terms. In the sequel, such extending of the set of variables is often performed implicitly.

Having a recursion scheme $\mathcal{G} = (\mathcal{X}, X_0, \Sigma, \mathcal{R})$, for every set $\mathcal{Z}$ of variables of type o we define an *ext-reduction* relation $\rightsquigarrow_{\mathcal{G}}$ between extended terms over $(\mathcal{X}, \mathcal{Z}, \Sigma)$, as the least relation such that

- $X\,K_1\,\ldots\,K_k\,L_1\,\ldots\,L_\ell \;\rightsquigarrow_{\mathcal{G}}\; R[K_1/y_1, \ldots, K_k/y_k, z'_1/z_1, \ldots, z'_\ell/z_\ell](\!|L_1/z'_1|\!)\ldots(\!|L_\ell/z'_\ell|\!)$ if $\ell = \mathsf{gar}(X)$, and $\mathcal{R}(X) = (X\,y_1\,\ldots\,y_k\,z_1\,\ldots\,z_\ell \to R)$, and $z'_1, \ldots, z'_\ell$ are fresh variables of type o not appearing in $\mathcal{Z}$.

Then, we define by coinduction the extended tree (over $(\mathcal{Z}, \Sigma)$) *ext-generated* by $\mathcal{G}$ from an extended term E (over $(\mathcal{X}, \mathcal{Z}, \Sigma)$), denoted $\mathsf{BT}^{\mathsf{ext}}_{\mathcal{G}}(E)$:

- if $E \rightsquigarrow^*_{\mathcal{G}} \langle a, F_1, \ldots, F_k\rangle$, then $\mathsf{BT}^{\mathsf{ext}}_{\mathcal{G}}(E) = \langle a, \mathsf{BT}^{\mathsf{ext}}_{\mathcal{G}}(F_1), \ldots, \mathsf{BT}^{\mathsf{ext}}_{\mathcal{G}}(F_k)\rangle$;
- if $E \rightsquigarrow^*_{\mathcal{G}} F(\!|L/z|\!)$, then $\mathsf{BT}^{\mathsf{ext}}_{\mathcal{G}}(E) = \mathsf{BT}^{\mathsf{ext}}_{\mathcal{G}}(F)(\!|\mathsf{BT}^{\mathsf{ext}}_{\mathcal{G}}(L)/z|\!)$;
- otherwise, $\mathsf{BT}^{\mathsf{ext}}_{\mathcal{G}}(E) = \langle\omega\rangle$.

The extended tree ext-generated by $\mathcal{G}$ (without mentioning a term), denoted $\mathsf{BT}^{\mathsf{ext}}(\mathcal{G})$, is defined as $\mathsf{BT}^{\mathsf{ext}}_{\mathcal{G}}(X_0)$. Formally, the ext-generated extended tree is not unique, because arbitrary fresh names may be used for bound variables; we should thus identify extended trees differing only in names of bound variables.

Finally, we say how to convert extended trees to trees, by performing all postponed substitutions. To this end, having fixed a set Σ of symbols, we define a *simplification* relation $\mapsto$ between extended trees over $(\varnothing, \Sigma)$ as the least relation such that

- $\langle a, T_1, \ldots, T_k\rangle(\!|L_1/z_1|\!)\ldots(\!|L_\ell/z_\ell|\!) \mapsto \langle a, T_1(\!|L_1/z_1|\!)\ldots(\!|L_\ell/z_\ell|\!), \ldots, T_k(\!|L_1/z_1|\!)\ldots(\!|L_\ell/z_\ell|\!)\rangle$, and
- $z_i(\!|L_1/z_1|\!)\ldots(\!|L_\ell/z_\ell|\!) \mapsto L_i(\!|L_{i+1}/z_{i+1}|\!)\ldots(\!|L_\ell/z_\ell|\!)$.

Then, we define by coinduction the *expansion* of an extended tree T over $(\varnothing, \Sigma)$, being a tree over Σ, and denoted $\mathsf{BT}^{\mathsf{s}}(T)$:

- if $T \mapsto^* \langle a, T_1, \ldots, T_k\rangle$, then $\mathsf{BT}^{\mathsf{s}}(T) = \langle a, \mathsf{BT}^{\mathsf{s}}(T_1), \ldots, \mathsf{BT}^{\mathsf{s}}(T_k)\rangle$;
- otherwise, $\mathsf{BT}^{\mathsf{s}}(T) = \langle\omega\rangle$.

The following lemma says that instead of generating a tree, we can first ext-generate an extended tree, and then expand all the explicit substitutions:

▶ **Lemma 5.1.** *For every recursion scheme $\mathcal{G}$ it holds that* $\mathsf{BT}(\mathcal{G}) = \mathsf{BT}^{\mathsf{s}}(\mathsf{BT}^{\mathsf{ext}}(\mathcal{G}))$.

The lemma can be proved in a standard way; a proof is contained in Appendix C of the full version (similar lemmata appear in previous work [2, Lemma 18], [24, Lemma 5.1]).

Transforming extended parity trees. An *extended parity tree* is an extended tree whose expansion is a parity tree. We now show how the transformation, defined previously for terms, can be applied to extended parity trees. Namely, we define $\mathsf{tr}^{\mathsf{t}}_d(Z, T)$ when T is an extended tree over $(\mathcal{Z}, \Sigma_d)$ for some set $\mathcal{Z}$ of variables of type o, and $Z\colon \mathcal{Z} \to D_d$ (we do not need an "A" argument, used previously to store declarations for arguments, because extended trees have no arguments). The definition is by coinduction:

(3') $\mathsf{tr}^{\mathsf{t}}_d(Z, z) = \top$ if $Z(z)$ is odd;
(4') $\mathsf{tr}^{\mathsf{t}}_d(Z, z) = \bot$ if $Z(z)$ is even;
(5') $\mathsf{tr}^{\mathsf{t}}_d(Z, \langle \wp, p, K_1, \ldots, K_k\rangle) = \langle \wp, p, \mathsf{tr}^{\mathsf{t}}_d(Z\restriction_p, K_1), \ldots, \mathsf{tr}^{\mathsf{t}}_d(Z\restriction_p, K_k)\rangle$;
(8') $\mathsf{tr}^{\mathsf{t}}_d(Z, T(\!|U/z|\!)) = \langle \mathrm{Eve}, 1, T^U_1, T^U_2, \ldots, T^U_d, T_{2d}\rangle$, where we take $T^U_r = \langle \mathrm{Adam}, 1, T_r, \langle \mathrm{Eve}, r, \mathsf{tr}^{\mathsf{t}}_d(Z\restriction_r, U)\rangle\rangle$ for $r \in [d]$ and $T_r = \mathsf{tr}^{\mathsf{t}}_d(Z[z \mapsto r], T)$ for $r \in D_d$.

Notice that tr_d transforms a term z to nonterminals $\overline{\top}$ or $\underline{\bot}$, while $\mathsf{tr}^{\mathsf{t}}_d$ transforms an extended tree z to trees $\top$ or $\bot$, generated from those nonterminals.

In the next lemma we observe that the tree generated by the transformed recursion scheme $\mathcal{G}^\dagger$ can be obtained by transforming the extended tree ext-generated by the original recursion scheme $\mathcal{G}$:

▶ **Lemma 5.2.** *For every parity recursion scheme $\mathcal{G}$ it holds that* $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, \mathsf{BT}^{\mathsf{ext}}(\mathcal{G})) = \mathsf{BT}(\mathcal{G}^\dagger)$.

The proof is purely syntactical, and is contained in Appendix D of the full version.

Transforming strategies. We finish our correctness proof by showing the following lemma:

▶ **Lemma 5.3.** *Let T be an extended parity tree over $(\varnothing, \Sigma_d)$. If a player $\wp \in \{\text{Adam}, \text{Eve}\}$ wins in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$, then $\wp$ wins also in $\mathsf{BT}^{\mathsf{s}}(T)$.*

Recall that the goal of this section is to prove that the winner in $\mathsf{BT}(\mathcal{G}^\dagger)$ is the same as in $\mathsf{BT}(\mathcal{G})$, for every parity recursion scheme $\mathcal{G}$. This follows from the above lemma used for $T = \mathsf{BT}^{\mathsf{ext}}(\mathcal{G})$, because $\mathsf{BT}(\mathcal{G}^\dagger) = \mathsf{tr}^{\mathsf{t}}_d(\varnothing, \mathsf{BT}^{\mathsf{ext}}(\mathcal{G}))$ by Lemma 5.2 and $\mathsf{BT}(\mathcal{G}) = \mathsf{BT}^{\mathsf{s}}(\mathsf{BT}^{\mathsf{ext}}(\mathcal{G}))$ by Lemma 5.1.

We now come to a proof of Lemma 5.3. In the sequel we assume a fixed extended parity tree T over $(\varnothing, \Sigma_d)$. Suppose first that it is Eve who wins in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$; thus, we also fix her winning strategy ρ in this tree. Our goal is to construct Eve's winning strategy ρ' in $\mathsf{BT}^{\mathsf{s}}(T)$.

In the proof, we use two additional notions. First, we say that a sequence of priorities $r_1, \dots, r_k$ is a $\preceq$-*contraction* of a sequence of priorities $p_1, \dots, p_n$ if the latter can be split at some indices $i_0, i_1, \dots, i_k$, where $0 = i_0 \le i_1 \le \cdots \le i_k = n$, so that for every $j \in [k]$ the infix $p_{i_{j-1}+1}, p_{i_{j-1}+2}, \dots, p_{i_j}$ fulfils declaration r_j. Likewise we define $\preceq$-contractions for infinite sequences, only there are infinitely many splitting indices (which necessarily tend to infinity, meaning that the whole infinite sequence is split).

Notice that we allow empty infixes, so one can arbitrarily insert odd numbers r_j (i.e., numbers r_j fulfilled by the empty sequence) to the $\preceq$-contraction. For example, $3, 4, 2$ is a $\preceq$-contraction of $4, 3, 2, 3, 4$ because the empty sequence fulfils 3, and $4, 3$ fulfils 4, and $2, 3, 4$ fulfils 2. On the other hand, $3, 4, 2$ is not a $\preceq$-contraction of $4, 3, 2, 3$. The idea of $\preceq$-contractions is to describe what happens when we move from $\mathsf{BT}^{\mathsf{s}}(T)$ to $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$. Indeed, if T has a subtree of the form $U(\!|V/z|\!)$, then in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$ the play can continue to V after playing only an Eve's declaration r (skipping completely U), while in $\mathsf{BT}^{\mathsf{s}}(T)$ before reaching V we traverse through U, where visited priorities are intended to fulfil r.

It is easy to see that $\preceq$-contractions are transitive, and that they can make the situation only worse for Eve:

▶ **Lemma 5.4.** *If a sequence π_1 is a $\preceq$-contraction of a sequence π_2, which is in turn a $\preceq$-contraction of a sequence π_3, then π_1 is a $\preceq$-contraction of π_3.*

▶ **Lemma 5.5.** *If an infinite sequence π_1 is a $\preceq$-contraction of an infinite sequence π_2, and the greatest priority appearing infinitely often in π_1 is even, then the greatest priority appearing infinitely often in π_2 is even as well.*

We now introduce the second notion (it concerns only finite sequences, and is relative to the bound d on priorities): for a declaration $r \in D_d$ and two sequences π_1, π_2 of priorities from $[d]$ we say that π_1 is an r-*extension* of π_2 if for every sequence π_3 of priorities from $[d]$ that fulfils the declaration r, the sequence π_1 is a $\preceq$-contraction of the concatenation $\pi_2 \cdot \pi_3$.

For example, the sequence $3, 4, 4$ is a 5-extension of the sequence $4, 3, 6$ (independently from the value of $d \ge 6$), because the empty sequence fulfils 3, and $4, 3$ fulfils 4, and $6, p_1, \dots, p_k$ fulfils 4 whenever $p_1, \dots, p_k$ fulfils 5 (i.e., the maximum among $p_1, \dots, p_k$ is either even or at most 5). Notice, moreover, that every sequence is a $2d$-extension of every sequence, because no sequence of priorities from $[d]$ can fulfil the declaration $2d$.

The following lemma is a direct consequence of the definition and of Lemma 3.2:

▶ **Lemma 5.6.** *If a sequence π is an r-extension of a sequence $p_1, \dots, p_n$, then π is also an $r{\restriction_{p_{n+1}}}$-extension of $p_1, \dots, p_n, p_{n+1}$ for every priority $p_{n+1} \in [d]$.*

Additionally, for a node v (of some parity tree) we write $\pi(v)$ for the sequence of priorities in ancestors of v (not including the priority in v).

We now come back to the proof, showing how to construct the new strategy ρ', winning for Eve in $\mathsf{BT}^{\mathsf{s}}(T)$. In order to describe ρ', we play simultaneously in both trees, $\mathsf{BT}^{\mathsf{s}}(T)$ and $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$, and we use moves in one tree to choose moves in the other tree. Namely, at every moment of the play, we remember

- a current node v in $\mathsf{BT}^{\mathsf{s}}(T)$,
- nodes $w_0, w_1, \dots, w_\ell$ in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$, for some $\ell \in \mathbb{N}$,
- variables $z_1, \dots, z_\ell$ of type o,
- functions $Z_0, Z_1, \dots, Z_\ell$ storing Eve's declarations, where $Z_i \colon \{z_{i+1}, \dots, z_\ell\} \to D_d$ for every i, and
- extended trees $U_0, U_1, \dots, U_\ell$, where every U_i is over $(\{z_{i+1}, \dots, z_\ell\}, \Sigma_d)$.

They satisfy the following invariant:

(a) $\mathsf{BT}^{\mathsf{s}}(T){\restriction_v} = \mathsf{BT}^{\mathsf{s}}(U_0(\!|U_1/z_1|\!) \dots (\!|U_\ell/z_\ell|\!))$,
(b) $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T){\restriction_{w_i}} = \mathsf{tr}^{\mathsf{t}}_d(Z_i, U_i)$ for all $i \in \{0, 1, \dots, \ell\}$,
(c) $\pi(w_0)$ is a $\preceq$-contraction of $\pi(v)$, and
(d) $\pi(w_j)$ is a $Z_i(z_j)$-extension of $\pi(w_i)$, for all i, j such that $0 \le i < j \le \ell$.

We start with $\ell = 0$, with v and w_0 at the root of $\mathsf{BT}^{\mathsf{s}}(T)$ and $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$, respectively, with $Z_0 = \varnothing$, and with $U_0 = T$. The invariant is clearly satisfied.

Then, during the play, we have one of three cases, depending on the shape of U_0:

1. First, assume that $U_0 = \langle \wp, p, T_1, \dots, T_k \rangle$. Then

$$\mathsf{BT}^{\mathsf{s}}(T){\restriction_v} = \langle \wp, p, \mathsf{BT}^{\mathsf{s}}(T_1(\!|U_1/z_1|\!) \dots (\!|U_\ell/z_\ell|\!)), \dots, \mathsf{BT}^{\mathsf{s}}(T_k(\!|U_1/z_1|\!) \dots (\!|U_\ell/z_\ell|\!)) \rangle;$$
$$\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T){\restriction_{w_0}} = \langle \wp, p, \mathsf{tr}^{\mathsf{t}}_d(Z_0, T_1), \dots, \mathsf{tr}^{\mathsf{t}}_d(Z_0, T_k) \rangle.$$

If $\wp$ = Adam, Adam chooses some child of v in $\mathsf{BT}^{\mathsf{s}}(T)$, and we choose the same child of w_0 in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$. If $\wp$ = Eve, Eve chooses some child of w_0 in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$, according to her strategy ρ, and in ρ' we choose the same child of v. Thus, in both cases, we move both v and w_0 to their c-th child, for some $c \in [k]$. We also take $Z_0{\restriction_p}$ as the new Z_0 and T_c as the new U_0. Lemma 5.6 ensures that Item (d) of the invariant is preserved.

2. Another possibility is that U_0 is a variable, that is, $U_0 = z_c$ for some $c \in [\ell]$. Then $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T){\restriction_{w_0}}$ (i.e., $\mathsf{tr}^{\mathsf{t}}_d(Z_0, U_0)$) is either $\bot$ or $\top$, depending on the parity of $Z_0(z_c)$. But our play in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$ follows an Eve's winning strategy, so it will be won by Eve, thus the subtree cannot be $\bot$, in which Eve is losing. In consequence $Z_0(z_c)$ is odd, so the empty sequence fulfils $Z_0(z_c)$. This implies that $\pi(w_c)$, being an $Z_0(z_c)$-extension of $\pi(w_0)$, is its $\preceq$-contraction, and thus also an $\preceq$-contraction of $\pi(v)$ (by Lemma 5.4). We discard w_i, z_i, Z_i, U_i for $i < c$ (so that w_c becomes now w_0, etc.).

3. Finally, assume that $U_0 = V(\!|W/z|\!)$. Then $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T){\restriction_{w_0}} = \langle \mathrm{Eve}, 1, V_1^W, \dots, V_d^W, V_{2d} \rangle$, where $V_r^W = \langle \mathrm{Adam}, 1, V_r, \langle \mathrm{Eve}, r, \mathsf{tr}^{\mathsf{t}}_d(Z_0{\restriction_r}, W) \rangle \rangle$ for $r \in [d]$ and $V_r = \mathsf{tr}^{\mathsf{t}}_d(Z_0[z \mapsto r], V)$ for $r \in D_d$. In such a node w_0 Eve, according to her strategy ρ, chooses a declaration r by going to an appropriate subtree V_r^W (or V_r if $r = 2d$). We then update our memory as follows:

- We leave v and w_i, z_i, Z_i, U_i for $i \ge 1$ unchanged.
- We move w_0 to the root of V_r (this adds once or twice priority 1 to $\pi(w_0)$, hence Item (c) of the invariant is preserved).

- Let $r' = r$ if $r \in [d]$, and $r' = 1$ if $r = 2d$.
- We add an additional node $w_{0.5}$ between w_0 and w_1 (saying this differently, we shift w_i for $i \geq 1$ by one, and we insert the new node in place of w_1). For $w_{0.5}$ we choose the root of $\mathsf{tr}^{\mathsf{t}}_d(Z_0{\restriction}_{r'}, W)$. Notice that $\pi(w_{0.5})$ is an r-extension of $\pi(w_0)$ (for $r \in [d]$ because $\pi(w_{0.5})$ is obtained from $\pi(w_0)$ by appending the priority $r' = r$, and for $r = 2d$ because no sequence of priorities from $[d]$ fulfils $2d$), and that every $\pi(w_j)$ for $1 \leq j \leq \ell$ is a $Z_0{\restriction}_{r'}(z_j)$-extension of $\pi(w_{0.5})$ (by Lemma 5.6).
- As Z_0, U_0, $z_{0.5}$, $Z_{0.5}$, and $U_{0.5}$ we take $Z_0[z \mapsto r]$, V, z, $Z_0{\restriction}_{r'}$, and W, respectively.

Observe that after finitely many repetitions of Cases 2 and 3 necessarily Case 1 has to occur, where the play advances in $\mathsf{BT}^{\mathsf{s}}(T)$. Indeed, $U_0(\!|U_1/z_1|\!)\dots(\!|U_\ell/z_\ell|\!)$ has to generate the next node of $\mathsf{BT}^{\mathsf{s}}(T)$ in finitely many steps; in particular, the number of explicit substitution at the head of U_0 has to be finite.

We have to prove that the infinite branch ξ of $\mathsf{BT}^{\mathsf{s}}(T)$ obtained this way is won by Eve. To this end, consider the corresponding sequence of "w_0" nodes in the construction and observe that this sequence converges to some infinite branch ζ in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$. Indeed, whenever the sequence enters to a subtree of the form $\mathsf{tr}^{\mathsf{t}}_d(Z_0, V(\!|W/z|\!))$ and stays there forever, then either it enters to the subtree $V_r = \mathsf{tr}^{\mathsf{t}}_d(Z_0[z \mapsto r], V)$ for some r and stays there forever, or, after some time, it enters to the subtree $\mathsf{tr}^{\mathsf{t}}_d(Z_0{\restriction}_r, W)$ for some r and stays there forever. Moreover, the sequence of priorities on ζ is a $\preceq$-contraction of the sequence of priorities on ξ (the function from elements of the former sequence to infixes of the latter sequence, as needed for $\preceq$-contraction, is obtained as the limit of such functions witnessing that always $\pi(w_0)$ is a $\preceq$-contraction of $\pi(v)$). Since ζ agrees with the strategy ρ, it is won by Eve, hence by Lemma 5.5 also ξ is won by Eve, as required. This finishes the proof in the case of Eve winning in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$.

Suppose now that it is Adam who wins in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$. The proof in this case is similar, so we only list differences. First, $\succeq$-*contraction* is defined like $\preceq$-contraction, but for every infix $p_{i_{j-1}+1}, p_{i_{j-1}+2}, \dots, p_{i_j}$ in the split we require that r_j is $\succeq$ (instead of $\preceq$) than the leader of the infix. Second, we say that a sequence π_1 of priorities from $[d]$ is an *r-neg-extension* of a sequence π_2 of priorities from $[d]$ if for every sequence π_3 of priorities from $[d]$ that does NOT fulfil the declaration r, the sequence π_1 is a $\succeq$-contraction of the concatenation $\pi_2 \cdot \pi_3$. In Items (c) and (d) of the invariant we replace $\preceq$-contraction by $\succeq$-contraction, and r-extension by r-neg-extension. Then, in Case 1 of the construction we only swap the role of Eve and Adam. In Case 2 we now have that the play is won by Adam, so $Z_0(z_c)$ is even, that is, not fulfilled by the empty sequence; this implies that $\pi(w_c)$, being an $Z_0(z_c)$-neg-extension of $\pi(w_0)$, is also its $\succeq$-contraction. The main difference is in Case 3. For every $r \in [d]$ we know Adam's decision in the root of V_r^W, according to his winning strategy. Take the worst $r \in [d]$ such that in V_r^W Adam goes to the left subtree, or $r = 2d$ if he goes right everywhere; in both cases, Adam's strategy allows to enter V_r. Let also s be the best among priorities that are worse than r; in V_s^W Adam goes to the right subtree (if there are no priorities worse than r, we choose s arbitrarily, e.g., $s = 1$). Then as the new w_0 we take the root of V_r, and as $w_{0.5}$ we take the root of $\mathsf{tr}^{\mathsf{t}}_d(Z{\restriction}_s, W)$. Notice that $\pi(w_{0.5})$ is an r-neg-extension of $\pi(w_0)$: s is better or equal than the leader of every sequence not fulfilling r (also when r is the worst priority, because no such a sequence exists), which ensures that the invariant is preserved.

6 Final remarks

We have presented a new, simple model-checking algorithm for higher-order recursion schemes. One may ask whether this algorithm can be used in practice. Of course the complexity n-EXPTIME for recursion schemes of order n is unacceptably large (even if we take into

account the fact that we are n-fold exponential only in the arity of types and in the size of an automaton, not in the size of a recursion scheme), but one has to recall that there exist tools solving the considered problem in such a complexity. The reason why these tools work is that the time spent by them on "easy" inputs is much smaller than the worst-case complexity (and many "typical inputs" are indeed easy). Unfortunately, this is not the case for our algorithm: the size of the recursion scheme resulting from our transformation is always large. Moreover, it seems unlikely that any simple analysis of the resulting recursion scheme (like removing useless nonterminals or some control flow analysis) may help in reducing its size. Indeed, one can see that if no nonterminals nor arguments were useless in the original recursion scheme, then also no nonterminals nor arguments are useless in the resulting recursion scheme. Thus, our algorithm is mainly of a theoretical interest.

It seems feasible that a transformation similar to the one presented in this paper can be used to solve the simultaneous unboundedness problem (aka. diagonal problem) [11] for recursion schemes. Developing such a transformation is a possible direction for further work.

References

1 Alfred V. Aho. Indexed grammars – an extension of context-free grammars. *J. ACM*, 15(4):647–671, 1968. doi:10.1145/321479.321488.

2 Kazuyuki Asada and Naoki Kobayashi. Size-preserving translations from order-$(n+1)$ word grammars to order-n tree grammars. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPIcs*, pages 22:1–22:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSCD.2020.22.

3 Luca Breveglieri, Alessandra Cherubini, Claudio Citrini, and Stefano Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996. doi:10.1142/S0129054196000191.

4 Christopher H. Broadbent, Arnaud Carayol, Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown parity games. *CoRR*, abs/2010.06361, 2020. arXiv:2010.06361.

5 Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong, and Olivier Serre. Recursion schemes and logical reflection. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 120–129. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.40.

6 Christopher H. Broadbent and Naoki Kobayashi. Saturation-based model checking of higher-order recursion schemes. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013, CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPIcs*, pages 129–148. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPIcs.CSL.2013.129.

7 Christopher H. Broadbent and C.-H. Luke Ong. On global model checking trees generated by higher-order recursion schemes. In Luca de Alfaro, editor, *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5504 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2009. doi:10.1007/978-3-642-00596-1_9.

8 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 252–263. ACM, 2017. doi:10.1145/3055399.3055409.

9 Arnaud Carayol and Olivier Serre. Collapsible pushdown automata and labeled recursion schemes: Equivalence, safety and effective selection. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 165–174. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.73.

10 Lorenzo Clemente, Paweł Parys, Sylvain Salvati, and Igor Walukiewicz. Ordered tree-pushdown systems. In Prahladh Harsha and G. Ramalingam, editors, *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, volume 45 of *LIPIcs*, pages 163–177. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.163.

11 Lorenzo Clemente, Paweł Parys, Sylvain Salvati, and Igor Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. *CoRR*, abs/1605.00371, 2016. arXiv:1605.00371.

12 Łukasz Czajka. Coinductive techniques in infinitary lambda-calculus. *CoRR*, abs/1501.04354, 2015. arXiv:1501.04354.

13 Koichi Fujima, Sohei Ito, and Naoki Kobayashi. Practical alternating parity tree automata model checking of higher-order recursion schemes. In Chung-chieh Shan, editor, *Programming Languages and Systems - 11th Asian Symposium, APLAS 2013, Melbourne, VIC, Australia, December 9-11, 2013. Proceedings*, volume 8301 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2013. doi:10.1007/978-3-319-03542-0_2.

14 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 452–461. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.34.

15 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. *ACM Trans. Comput. Log.*, 18(3):25:1–25:42, 2017. doi:10.1145/3091122.

16 Teodor Knapik, Damian Niwiński, and Paweł Urzyczyn. Higher-order pushdown trees are easy. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002. doi:10.1007/3-540-45931-6_15.

17 Naoki Kobayashi. Model-checking higher-order functions. In António Porto and Francisco Javier López-Fraguas, editors, *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, September 7-9, 2009, Coimbra, Portugal*, pages 25–36. ACM, 2009. doi:10.1145/1599410.1599415.

18 Naoki Kobayashi. A practical linear time algorithm for trivial automata model checking of higher-order recursion schemes. In Martin Hofmann, editor, *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6604 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2011. doi:10.1007/978-3-642-19805-2_18.

19 Naoki Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3):20:1–20:62, 2013. doi:10.1145/2487241.2487246.

20 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 179–188. IEEE Computer Society, 2009. doi:10.1109/LICS.2009.29.

21 Robin P. Neatherway and C.-H. Luke Ong. TravMC2: Higher-order model checking for alternating parity tree automata. In Neha Rungta and Oksana Tkachuk, editors, *2014 International Symposium on Model Checking of Software, SPIN 2014, Proceedings, San Jose, CA, USA, July 21-23, 2014*, pages 129–132. ACM, 2014. doi:10.1145/2632362.2632381.

22 Robin P. Neatherway, Steven J. Ramsay, and C.-H. Luke Ong. A traversal-based algorithm for higher-order model checking. In Peter Thiemann and Robby Bruce Findler, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP'12, Copenhagen, Denmark, September 9-15, 2012*, pages 353–364. ACM, 2012. doi:10.1145/2364527.2364578.

23 C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *21th IEEE Symposium on Logic in Computer Science, LICS 2006, 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 81–90. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.38.

24 Paweł Parys. Higher-order nonemptiness step by step. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, December 14-18, 2020, BITS Pilani, K K Birla Goa Campus, Goa, India (Virtual Conference)*, volume 182 of *LIPIcs*, pages 53:1–53:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSTTCS.2020.53.

25 Steven J. Ramsay, Robin P. Neatherway, and C.-H. Luke Ong. A type-directed abstraction refinement approach to higher-order model checking. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 61–72. ACM, 2014. doi:10.1145/2535838.2535873.

26 Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. *Inf. Comput.*, 239:340–355, 2014. doi:10.1016/j.ic.2014.07.012.

27 Sylvain Salvati and Igor Walukiewicz. A model for behavioural properties of higher-order programs. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 229–243. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CSL.2015.229.

28 Sylvain Salvati and Igor Walukiewicz. Simply typed fixpoint calculus and collapsible pushdown automata. *Math. Struct. Comput. Sci.*, 26(7):1304–1350, 2016. doi:10.1017/S0960129514000590.

29 Ryota Suzuki, Koichi Fujima, Naoki Kobayashi, and Takeshi Tsukada. Streett automata model checking of higher-order recursion schemes. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017, September 3-9, 2017, Oxford, UK*, volume 84 of *LIPIcs*, pages 32:1–32:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.FSCD.2017.32.

30 Takeshi Tsukada and C.-H. Luke Ong. Compositional higher-order model checking via ω-regular games over Böhm trees. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 78:1–78:10. ACM, 2014. doi:10.1145/2603088.2603133.