Krivine Machines and Higher-Order Schemes

S. Salvati and I. Walukiewicz*

Université de Bordeaux, INRIA, CNRS, LaBRI UMR5800

Abstract. We propose a new approach to analysing higher-order recursive schemes. Many results in the literature use automata models generalising pushdown automata, most notably higher-order pushdown automata with collapse (CPDA). Instead, we propose to use the Krivine machine model. Compared to CPDA, this model is closer to lambda-calculus, and incorporates nicely many invariants of computations, as for example the typing information. The usefulness of the proposed approach is demonstrated with new proofs of two central results in the field: the decidability of the local and global model checking problems for higher-order schemes with respect to the mu-calculus.

1 Introduction

Higher-order recursive schemes were introduced by Damm in [Dam82] as a respelling of λY-calculus. Since higher-order recursive schemes were investigated mainly in the formal language community, the tools developed were by and large inspired by the treatment of pushdown-automata and context-free grammars. Subsequent research has shown that it is very useful to have an automata model characterising schemes. For the class of all schemes, we know only one such model, that is higher-order pushdown automata with collapse [HMOS08]. In this paper we propose another model based on Krivine machines [Kri07, Wan07]. The notion of Krivine machine is actually a standard concept in the lambda-calculus community, and it needs almost no adaptation to treat higher-order schemes. We claim that the proposed model offers a fresh tool to analyse schemes. To substantiate we give new proofs of two central results in the field: decidability of local and global model-checking problems for higher-order schemes with respect to the mu-calculus.

The interest in higher-order schemes has been renewed by the discovery by Knapik et al.[KNU02] of the equivalence of higher-order pushdown automata of order n with schemes of order n satisfying a syntactic constraint called *safety*. The safety condition implicitly appeared in Damm's work. Indeed, Damm considers only the so-called *derived types* in the definition of higher-order schemes. This in itself does not restrict the expressive power of schemes if one permits explicit lambda abstractions [DF80]. But then Damm studies only *applicative* schemes which are equivalent to schemes satisfying the safety condition [dM06]. In recent years, higher-order pushdowns have been extended with *panic* operation to

* This work has been supported by ANR 2010 BLAN 0202 01 FREC.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 162–173, 2011.

handle all level 2 schemes [KNUW05, AdMO05], and with *collapse* operation for schemes of all levels [HMOS08]. Higher-order pushdown automata with collapse are at present the main tool to analyse schemes [HMOS08, BO09, BCOS10].

The model checking problem for schemes with respect to the mu-calculus is to decide if a given formula holds in the root of the tree generated by a given scheme. The problem has proved to be very stimulating, and generated many advances in our understanding of schemes. Its decidability has been shown by Ong [Ong06], but even afterwards the problem continued to drive interesting work. Several different proofs of Ong's result have been proposed [HMOS08, KO09]. In a series of recent papers [CHM$^+$08, BO09, BCOS10] the global version of the problem is considered. In the last citation it is shown that the set of nodes satisfying a given mu-calculus formula is definable in a finitary way.

In this paper, we go several steps back with respect to the usual ways of working with higher-order recursive schemes. First, instead of using Damm's definition of higher-order schemes, we turn to the λY-calculus as the means of generating infinite trees. The Y combinator, or the fixpoint combinator, has first been considered in [CR58] and is at the core of Plotkin's PCF [Plo77]. Second, instead of using higher-order collapsible automata as an abstract machine, we use Krivine abstract machine [Kri07]. This machine is much closer to λ-calculus, it performs standard reductions and comes with typing. These features are hard to overestimate as they allow to use standard techniques to express powerful invariants on the computation. For example, in the main proof presented here, we use standard models of the λY-calculus to express such invariants.

Using these tools, we reprove in a rather succinct way Ong's result. Similarly to a recent proof of Kobayashi and Ong [KO09], our proof gives a reduction to a finite parity game. It seems though that our game is simpler. For example, the paper [BCOS10] on global model checking continues to use collapsible pushdown automata and gives an involved proof by induction on the rank of the stack. On the other hand, we can reuse our game to give a short proof of this result. In particular unlike op cit. we use finite trees to represent positions, and standard automata on finite trees to represent sets of winning positions. In this context we would like to mention a result of Kartzow [Kar10] showing that order-2 collapsible stacks can be encoded as trees in such a way that the set of stacks reachable from the initial configuration is a regular set of trees.

Organization of the paper. In the next section we introduce λY-calculus and Krivine machines. We also define formally the local model checking problem. In the following section we reduce the problem to determining a winner in a game over configurations of the Krivine machine, $\mathcal{K}(\mathcal{A}, M)$. In the next section we define a finite game $\mathcal{G}(\mathcal{A}, M)$. We then show that the same player is winning in the two games. This gives decidability of the local model checking problem. In the following section we reuse this result to obtain the proof for the global model checking problem. All missing proofs can be found in the long version of the paper [SW11].

2 Basic Notions

The set of types $\mathcal{T}$ is constructed from a unique *basic type* 0 using a binary operation $\to$. Thus 0 is a type and if α, β are types, so is $(\alpha \to \beta)$. The order of a type is defined by: $order(0) = 1$, and $order(\alpha \to \beta) = max(1 + order(\alpha), order(\beta))$.

A *signature*, denoted Σ, is a set of typed constants, that is symbols with associated types from $\mathcal{T}$. We will assume that for every type $\alpha \in \mathcal{T}$ we have ω^α and $Y^{(\alpha\to\alpha)\to\alpha}$ standing for the undefined value and the fixpoint operator. For simplicity of notation we assume that all other constants are of type $0 \to 0 \to 0$. In general, in recursive schemes all constants of order 0 and 1 are allowed; it is straightforward to extend our arguments to all such constants.

The set of *simply typed λ-terms* is defined inductively as follows. A constant of type α is a term of type α. For each type α there is a countable set of variables $x^\alpha, y^\alpha, \ldots$ that are also terms of type α. If M is a term of type β and x^α a variable of type α then $\lambda x^\alpha.M$ is a term of type $\alpha \to \beta$. Finally, if M is of type $\alpha \to \beta$ and N is of type α then MN is a term of type β. Together with the usual operational semantics of λ-calculus, that is β-reduction, we use δ-reduction ($\to_\delta$) giving the semantics to the fixpoint operator: $YM \to_\delta M(YM)$. Thus, the operational semantics of the λY-calculus is the $\beta\delta$-reduction, it is well-known that this semantics is confluent and enjoys subject reduction (*i.e.* the type of terms is invariant under computation). In this paper, we only consider terms in η-long form [Hue76]. This makes the presentation easier as the structure of types is reflected syntactically in terms. Later we will point out the place where we use this assumption. We will also often omit type annotations.

A *Böhm tree* is an unranked ordered, and potentially infinite tree with nodes labelled by ω^α, or terms of the form $\lambda x_1 \ldots x_n.N$; where N is a variable or a constant, and the sequence of lambda abstractions is optional. So for example x^0, $\lambda x^0.\omega^0$ are labels, but $\lambda y^0.x^{0\to 0}y^0$ is not. A Böhm tree of a term M is obtained as follows. If $M \to^*_{\beta\delta} \lambda \boldsymbol{x}.N_0N_1 \ldots N_k$ with N_0 a variable or a constant then the root of $BT(M)$ is labelled by $\lambda \boldsymbol{x}.N_0$ and has $BT(N_1), \ldots, BT(N_k)$ as a sequence of its children. If M is not solvable then $BT(M) = \omega^\alpha$, where α is the type of M. If M is of type 0 then given our assumption on the type of constants we get that $BT(M)$ is a binary tree with finite branches ending in ω^0. It is known that λY-calculus allows to define the same trees as recursive schemes [DF80].

Krivine machine. A Krivine machine [Kri07], is an abstract machine that computes the weak head normal form of a λ-term, using explicit substitutions, called *environments*. Environments are functions assigning *closures* to variables, and closures themselves are pairs consisting of a term and an environment. This mutually recursive definition is schematically represented by the grammar:

$$C ::= (M, \rho) \qquad \rho ::= \emptyset \mid \rho[x \mapsto C]$$

As in this grammar, we will use $\emptyset$ for the empty environment. We require that in a closure (M, ρ), the environment is defined for every free variable of M. Intuitively such a closure denotes closed λ-term: it is obtained by substituting for every free variable x of M the lambda term denoted by the closure $\rho(x)$.

A configuration of the Krivine machine is a triple (M, ρ, S), where M is a term, ρ is an *environment*, and S is a *stack* (a sequence of closures with the topmost element on the left). The rules of the Krivine machine are as follows:

$$\begin{aligned}
(\lambda x.M, \rho, (N, \rho')S) &\to (M, \rho[x \mapsto (N, \rho')], S) \\
(YM, \rho, S) &\to (M(YM), \rho, S) \\
(MN, \rho, S) &\to (M, \rho, (N, \rho)S) \\
(x, \rho, S) &\to (M, \rho', S) \qquad \text{where } (M, \rho') = \rho(x)
\end{aligned}$$

Note that the machine is deterministic. We will be only interested in configurations accessible from $(M, \emptyset, \varepsilon)$ for some term M of in η-long form and of type 0. Every such configuration (N, ρ, S) enjoys very strong typing invariants. Environment ρ associates to a variable x^α a closure (K, ρ') so that K has type α; we will say that the closure is of type α too. If N has type $\alpha_1 \to \cdots \to \alpha_n \to 0$, then S is a stack of n closures, with i-th closure from the top being of type α_i.

For aesthetic reasons we prefer to stop the Krivine machine in configurations of the form $(bN_0N_1, \rho, \varepsilon)$, where b is a constant; since b is of type $0 \to 0 \to 0$, the stack must be empty. We shall write this configuration as $(b(N_0, N_1), \rho, \varepsilon)$ to make a link with the Böhm tree being constructed. (Notice that formally from such a configuration the machine should perform two more reductions to put the arguments on the stack.) Thus, if we start with a closed term M of type 0 we get a sequence of reductions from $(M, \emptyset, \varepsilon)$ that is either infinite or terminates in a configuration of a form $(b(N_0, N_1), \rho, \varepsilon)$, thanks to the fact that M is supposed to be in η-long form. At that point we create a node labelled b and start reducing both (N_0, ρ, ε) and (N_1, ρ, ε), that are both in η-long form as well. This process gives at the end a tree labelled with constants that is precisely $BT(M)$; that is the object of our study. Notice that if (N, ρ, S) is reachable from $(M, \emptyset, \varepsilon)$ then N, and the terms that occur in ρ and in S are all subterms of M. One should be careful with a definition of a subterm though. Since we have a fixpoint operator we consider that $N(YN)$ is a subterm of YN. Of course even with this twist, the number of subterms of a term remains finite.

We present an execution of a Krivine machine on an example taken from [KO09]. For clarity, in this example we suspend our convention on types of the constants and take constants $a : 0 \to 0 \to 0$, $b : 0 \to 0$ and $c : 0$. The scheme is defined by $S \mapsto F\,c$ and $F \mapsto \lambda x.a\,x\,(F(b\,x))$ which can be represented by the following term in the λY-calculus:

$$YM\,c \qquad \text{where } M = \lambda f x.a\,x\,(f(b\,x)).$$

Starting from a configuration $(YMc, \emptyset, \varepsilon)$, the Krivine machine produces the following sequence of reductions

$$\begin{aligned}
(YMc, \emptyset, \varepsilon) \to (YM, \emptyset, (c, \emptyset)) \to (M(YM), \emptyset, (c, \emptyset)) \to (M, \emptyset, (YM, \emptyset)(c, \emptyset)) \to \\
(\lambda x.a\,x\,(f(b\,x)), [f \mapsto (YM, \emptyset)], (c, \emptyset)) \to \\
(a\,x\,(f(b\,x)), [f \mapsto (YM, \emptyset)][x \mapsto (c, \emptyset)], \varepsilon)
\end{aligned}$$

At this point we have reached a final configuration and we get the constant a that is the symbol of the root of $BT(YMc)$. We can start reducing separately the two arguments of a, that is reducing the configurations:

$$(x, [f \mapsto (YM, \emptyset)][x \mapsto (c, \emptyset)], \varepsilon) \text{ and } (f(b\,x), [f \mapsto (YM, \emptyset)][x \mapsto (c, \emptyset)], \varepsilon).$$

Parity automata and the definition of the problem. Recall that Σ is a fixed set of constants of type $0 \to 0 \to 0$. These constants label nodes in $BT(M)$. Since $BT(M)$ is an infinite binary tree we can use standard non-deterministic parity automata to describe its properties. Such an automaton has the form

$$\mathcal{A} = \langle Q, \Sigma, q^0 \in Q, \delta : Q \times \Sigma \to \mathcal{P}(Q^2), \Omega : Q \to \{1, \ldots, d\}\rangle \tag{1}$$

where Q is a finite set of states, q^0 is the initial state, δ is the transition function, and Ω is a function assigning a rank (a number between 1 and d) to every state.

In general, an infinite binary tree is a function $t : \{0,1\}^* \to \Sigma$. A run of $\mathcal{A}$ on t is a function $r : \{0,1\}^* \to Q$ such that $r(\varepsilon) = q^0$ and for every sequence $w \in \{0,1\}^*$: $(r(w0), r(w1)) \in \delta(q, t(w))$. The run is accepting if for every infinite path in the tree, the sequence of states assigned to this path satisfies the parity condition determined by Ω; this means that the maximal rank of a state seen infinitely often should be even.

Formally, it may be the case that $BT(M)$ contains also nodes labelled with ω^0. We will simply assume that every tree containing ω^0 is rejected by the automaton. This is frequently done in this context. Handling ω^0 would not be difficult but would require to add one more case in all the constructions. The other, more difficult, solution is to convert a term to a term not generating ω^0.

Definition 1. The (local) model checking problem *is to decide if* $\mathcal{A}$ *accepts* $BT(M)$ *for given* $\mathcal{A}$ *and* M.

3 Game Over Configurations of the Krivine Machine

In this section we will reduce the model checking problem to the problem of determining a winner in a specially constructed parity game.

Given an automaton $\mathcal{A}$ as in (1) we construct the tree of all its possible runs on $BT(M)$. We define the tree of runs formally as we will make one twist to the rules of the Krivine machine. The twist is that in the environment the value of the variable will not be a closure, that is a pair (term, environment), but a triple containing additionally the node of the tree where the closure has been created. For a given M and $\mathcal{A}$ we define the tree of runs $RT(\mathcal{A}, M)$ of $\mathcal{A}$ on $BT(M)$:

- The root of the tree is labelled with $q^0 : (M, \emptyset, \varepsilon)$.
- A node labelled $q : (a(N_0, N_1), \rho, \varepsilon)$ has a successor $(q_0, q_1) : (a(N_0, N_1), \rho, \varepsilon)$ for every $(q_0, q_1) \in \delta(q, a)$.
- A node labelled $(q_0, q_1) : (a(N_0, N_1), \rho, \varepsilon)$ has two successors $q_0 : (N_0, \rho, \varepsilon)$ and $q_1 : (N_1, \rho, \varepsilon)$.
- A node labelled $q : (\lambda x.N, \rho, (v', N', \rho')S)$ has a unique successor labelled $q : (N, \rho[x \mapsto (v', N', \rho')], S)$.

- A node $q : (YN, \rho, S)$ has a unique successor $q : (N(YN), \rho, S)$.
- A node v labelled $q : (NK, \rho, S)$ has a unique successor $q : (N, \rho, (v, K, \rho)S)$. (Here the v closure is created.)
- A node v labelled $q : (x, \rho, S)$ with $\rho(x) = (v', N, \rho')$ has a unique successor labelled $q : (N, \rho', S)$. (We say that the node v uses the v' closure.)

The definition is as expected but for the fact that in the rule for the application we store the current node in the closure that is pushed on the stack. When we use the closure in the variable rule, the stored node does not influence the result, but allows us to detect the exact closure that we are using. This will be important in the proof.

Definition 2. *We use the tree $RT(\mathcal{A}, M)$ to define a game between two players: Eve chooses a successor in nodes of the form $q : (a(N_0, N_1), \rho, \varepsilon)$, and Adam in nodes $(q_0, q_1) : (a(N_0, N_1), \rho, \varepsilon)$. We set the rank of nodes labelled $q : (a(N_0, N_1), \rho, \varepsilon)$ to $\Omega(q)$ and the ranks of all the other nodes to 1. We can use max parity condition to decide who wins an infinite play. Let us call the resulting game $\mathcal{K}(\mathcal{A}, M)$.*

Directly from the definition it follows that Eve has a strategy from the root position in $\mathcal{K}(\mathcal{A}, M)$ iff $\mathcal{A}$ accepts $BT(M)$. The only interesting point to observe is that it is important to disallow rank 0 in the definition of the parity automaton since we assign rank 1 to all "intermediate" positions. This is linked to our handling of infinite sequences of reductions of the Krivine machine that do not reach a head normal form. Such a sequence results in a node labelled ω^0 in the Böhm tree, hence the tree should not be accepted by the automaton. Indeed, in the game $\mathcal{K}(\mathcal{A}, M)$ this will give an infinite sequence of states of rank 1.

Summarizing, the model checking problem is equivalent to deciding who has a winning strategy from the root of $\mathcal{K}(\mathcal{A}, M)$. We will show decidability of the latter problem by reducing the game to a finite game.

4 Finite Game $G(\mathcal{A}, M)$

The game $\mathcal{K}(\mathcal{A}, M)$ may have infinitely many positions as there may be infinitely many closures that are created. In order to obtain a finite game we abstract these closures to some finite set. Closures are created by the application rule, so this is where we will concentrate our efforts. As in the construction for a pushdown game [Wal01] we will use alternation to "disarm" the application rule. Instead of putting a closure on the stack, Eve will make an assumption on the context in which the closure will be used. Adam will be then given a chance to either contest this assumption or to check what happens with the closure when it is used under the assumptions Eve has made. Since the closure can be of higher type, the assumptions are a bit more complicated than in a pushdown game.

Definition 3 (Residuals). *Recall that Q is the set of states of $\mathcal{A}$ and d is the maximal value of the rank function of $\mathcal{A}$. Let $[d]$ stand for the set $\{1, \ldots, d\}$. For every type $\tau = \tau_1 \to \cdots \to \tau_k \to 0$ the set of residuals D_τ is the set of functions $D_{\tau_1} \to \cdots \to D_{\tau_k} \to \mathcal{P}(Q \times [d])$.*

For example, we have that D_0 is $\mathcal{P}(Q\times[d])$ and $D_{0\to 0}$ is $\mathcal{P}(Q\times[d]) \to \mathcal{P}(Q\times[d])$. The meaning of residuals will become clearer when we define the game.

A position of the game $G(\mathcal{A}, M)$ will be of one of the forms:

$$q : (N, \rho, S) \quad \text{or} \quad (q_0, q_1) : (N, \rho, S) \quad \text{or} \quad (q, R) : (N, \rho, S).$$

where q, q_0, q_1 are states of $\mathcal{A}$, N is a term (more precisely a subterm of M), ρ is an environment assigning a residual to every variable that has a free occurrence in N, R is a residual, and S is a stack of residuals. Of course the types of residuals will agree with the types of variables/arguments they are assigned too. As there are only finitely many residuals of each type, the game $G(\mathcal{A}, M)$ has finitely many positions.

We need one more operation before defining the game. Take a rank r and a residual $R : D_{\tau_1} \to \cdots \to D_{\tau_k} \to D_0$. Recall that $D_0 = \mathcal{P}(Q \times [d])$. We define $R\restriction_r$ to be the function such that for every sequence of arguments S:

$$R\restriction_r (S) = \{(q_1, r_1) \in R(S) : r_1 > r\} \cup \{(q_1, r_2) : (q_1, r_1) \in R(S), r_2 \leq r_1 = r\}$$

Intuitively, $(q_1, r_1) \in R(S)$ means that Eve is allowed to reach a leaf labelled with a state q_1 if r_1 is the maximal rank between the creation and the use of the closure. Now suppose that with this residual at hand we see rank r. If $(q_1, r_1) \in R(S)$ and $r_1 > r$ then we are still waiting for r_1 so we just keep the pair. If $r_1 < r$ then such a pair is impossible and is removed. If $r_1 = r$ then in the future we can see any rank not bigger than r. This explains the second component of the sum. If ρ is an environment then $\rho\restriction_r$ is an environment such that for every x: $(\rho\restriction_r)(x) = \rho(x)\restriction_r$.

We have all ingredients to define transitions of the game $G(\mathcal{A}, M)$. Most of the rules are just reformulation of the rules in $\mathcal{K}(\mathcal{A}, M)$:

$$\begin{aligned}
q : (\lambda x.N, \rho, R \cdot S) &\to q : (N, \rho[x \mapsto R], S) \\
q : (a(N_0, N_1), \rho, \varepsilon) &\to (q_0, q_1) : (a(N_0, N_1), \rho, \varepsilon) \qquad \text{for } (q_0, q_1) \in \delta(q, a) \\
(q_0, q_1) : (a(N_0, N_1), \rho, \varepsilon) &\to q_i : (N_i, \rho\restriction_{\Omega(q_i)}, \varepsilon) \qquad \text{for } i = 0, 1 \\
q : (YN, \rho, S) &\to q : (N(YN), \rho, S)
\end{aligned}$$

We now proceed to the rule for application. Consider $q : (NK, \rho, S)$ with K of type $\tau = \tau_1 \to \cdots \to \tau_l \to 0$. We have a transition

$$q : (NK, \rho, S) \to (q, R) : (NK, \rho, S)$$

for every residual $R : D_{\tau_1} \to \ldots D_{\tau_l} \to D_0$. From this position we have transitions

$$\begin{aligned}
&(q, R) : (NK, \rho, S) \to q : (N, \rho, R\restriction_{\Omega(q)} \cdot S) \\
&(q, R) : (NK, \rho, S) \to q' : (K, \rho\restriction_{r'}, R_1 \cdots R_l) \text{ for every } R_1 \in D_{\tau_1}, \ldots, R_l \in D_{\tau_l} \\
&\qquad\qquad \text{and } (q', r') \in R\restriction_{\Omega(q)} (R_1, \ldots, R_l).
\end{aligned}$$

Here $R\restriction_{\Omega(q)}$ is needed to "normalise" the residual, so that it satisfies the invariant described below.

Since we are defining a game we need to say who makes a choice in which vertices. Eve chooses a successor from vertices of the form $q : (NK, \rho, S)$ and $q : (a(N_0, N_1), \rho, S)$. It means that she can choose a residual, and a transition of the automaton. This leaves for Adam the choices in nodes of the form $(q_0, q_1) : (a(N_0, N_1), \rho, S)$ and $(q, R) : (NK, \rho, S)$. So he chooses a direction, or decides whether to accept (by choosing a transition of the first type) or contest the residual proposed by Eve.

Observe that we do not have a rule for nodes with a term being a variable. This means that such a node has no successors, so we need to say who is the winner when the node is reached. Consider a node

$$q : (x, \rho, S) \quad \text{with } \rho(x) = R_x \text{ and } S = R_1 \cdots R_k.$$

Eve wins in this position if $(q, \Omega(q)) \in R_x(R_1, \ldots, R_k)$.

Finally, we define ranks. It will be much simpler to define ranks on transitions instead of nodes. All the transitions will have rank 1 but for two cases: (i) a transition $(q, R) : (NK, \rho, S) \to q' : (K, \rho \restriction_{r'}, R_1 \cdots R_k)$ has rank r'; (ii) a transition $(q_0, q_1) : (a(N_0, N_1), \rho, S) \to q_i : (N_i, \rho \restriction_{\Omega(q_i)}, S \restriction_{\Omega(q_i)})$ has rank $\Omega(q_i)$.

A play is winning for Eve iff the sequence of ranks on transitions satisfies the parity condition: the maximal rank appearing infinitely often is even.

5 Equivalence of $\mathcal{K}(\mathcal{A}, M)$ and $G(\mathcal{A}, M)$

In this section we present the main technical result of the paper

Theorem 1. *Eve wins in $G(\mathcal{A}, M)$ iff Eve wins in $\mathcal{K}(\mathcal{A}, M)$.*

Since $G(\mathcal{A}, M)$ is finite, this gives the decidability of the winner in $\mathcal{K}(\mathcal{A}, M)$ and hence also of the model-checking problem. We will show how to construct the winning strategy for Eve in $G(\mathcal{A}, M)$ from her winning strategy in $\mathcal{K}(\mathcal{A}, M)$. Due to lack of space we omit a dual construction of a winning strategy for Adam in $G(\mathcal{A}, M)$ from his winning strategy in $\mathcal{K}(\mathcal{A}, M)$.

Let us fix a winning strategy σ of Eve in $\mathcal{K}(\mathcal{A}, M)$, and consider the tree $\mathcal{K}_\sigma$ of plays respecting this strategy. This is a subtree of $\mathcal{K}(\mathcal{A}, M)$. We will define the strategy for Eve in $G(\mathcal{A}, M)$ that will use σ to guess residuals in the application rule. The first step before constructing the strategy is to calculate residuals $R(v)$ and $res(v, v')$ for all nodes in the tree $\mathcal{K}_\sigma$.

Residuals $R(v)$ and $res(v, v')$. The crucial step in the proof is the assignment of residuals to positions of $\mathcal{K}(\mathcal{A}, M)$. Thanks to typing, this can be done by induction on the order of type. We will assign a residual $R(v)$ to every v closure. Before proceeding we will need two simple abbreviations. If v is an ancestor of v_1 in $\mathcal{K}_\sigma$ then we write $\max(v, v_1)$ for the maximal rank appearing on the path between v and v_1, including both ends. We write $res(v, v_1)$ for $R(v) \restriction_{\max(v,v_1)}$; intuitively it is residual $R(v)$ as seen from v_1.

Consider an application node v in $\mathcal{K}(\mathcal{A}, M)$. It means that v has a label of the form $q : (NK, \rho, S)$, and its unique successor has the label $q : (N, \rho, (v, K, \rho)S)$. That is the closure (v, K, ρ) is created in v. We will look at all the places where this closure is used and summarize the information about them in $R(v)$.

When the closure is of type 0 then the residual $R(v)$ is a subset of $Q \times [d]$. For every node v' in $\mathcal{K}_\sigma$ labelled $q' : (x, \rho', \varepsilon)$ such that $\rho'(x) = (v, K, \rho)$ we put

$$(q', \max(v, v')) \in R(v)$$

When the closure is of type $\tau_1 \to \cdots \to \tau_k \to 0$, then by induction we assume that we have residuals for all closures of types $\tau_1, \ldots, \tau_k$. This time $R(v) : D_{\tau_1} \to \ldots D_{\tau_k} \to \mathcal{P}(Q \times [d])$. Take a node v' using the closure. Its label has the form $q' : (x, \rho', S')$ for some x, ρ' and S' such that $\rho'(x) = (v, K, \rho)$. The stack S' has the form $(v_1, N_1, \rho_1) \ldots (v_k, N_k, \rho_k)$ with N_i of type τ_i. We put

$$(q', \max(v, v')) \in R(res(v_1, v'), \ldots, res(v_k, v')) \ . \tag{2}$$

We extend the definition of residuals to closures, environments and stacks. For a closure (v, K, ρ) we define $res((v, K, \rho), v') = res(v, v')$. For an environment ρ, the environment $\rho' = res(\rho, v')$ is obtained by setting $\rho'(x) = res(\rho(x), v')$ for every variable x. Similarly, $res(S, v')$ is S where $res(\cdot, v')$ is applied to every element of the stack. With this notation the condition (2) can be rewritten as $(q', \max(v, v')) \in R(res(S', v'))$.

The strategy in $G(\mathcal{A}, M)$ Now we are ready to define the strategy for Eve in $G(\mathcal{A}, M)$. It will use positions in the game $\mathcal{K}(\mathcal{A}, M)$ and the strategy σ as hints. The new strategy will take a pair of positions (v_1, v_2) with v_1 in $G(\mathcal{A}, M)$, and v_2 in $\mathcal{K}(\mathcal{A}, M)$. It will then give a new pair of positions (v_1', v_2') such that v_1' is a successor v_1, and v_2' is reachable from v_2 using the strategy σ. Moreover, all visited pairs (v_1, v_2) will satisfy the following invariant:

$$\begin{gathered} v_1 \text{ is labelled by } q : (N, \rho_1, S_1) \text{ and } v_2 \text{ is labelled by } q : (N, \rho_2, S_2); \\ \text{where } \rho_1 = res(\rho_2, v_2) \text{ and } S_1 = res(S_2, v_2). \end{gathered}$$

The initial positions in both games have the same label $q^0 : (M, \varepsilon, \emptyset)$, so the invariant is satisfied. In order to define the strategy we will consider one by one the rules defining the transitions in $G(\mathcal{A}, M)$.

In most of the cases the strategy in $G(\mathcal{A}, M)$ just copies the moves of the strategy in $\mathcal{K}(\mathcal{A}, M)$. The only complicated case is the application rule.

Suppose that the term in the label of v_1 is an application, say $q : (NK, \rho_1, S_1)$. By our invariant we have a position v_2 labelled by $q : (NK, \rho_2, S_2)$, where $\rho_1 = res(\rho_2, v_2)$ and $S_1 = res(S_2, v_2)$. The strategy in $G(\mathcal{A}, M)$ is to choose $R(v_2)$, that is to go from v_1 to the node v_1' labelled $(q, R(v_2)) : (NK, \rho_1, S_1)$. From this node Adam can choose either

$$q : (N, \rho_1, (R(v_2) \restriction_{\Omega(q)}) \cdot S_1), \quad \text{or} \tag{3}$$

$$q' : (K, \rho_1 \restriction_{r'}, R_1 \ldots R_l) \quad \text{where } (q', r') \in R(v_2) \restriction_{\Omega(q)} (R_1, \ldots, R_l). \tag{4}$$

Suppose Adam chooses (3). By definition $R(v_2) \restriction_{\Omega(q)} = res(v_2, v_2)$. Hence the stack $(R(v_2) \restriction_{\Omega(q)}) \cdot S_1$ is just $res((v_2, K, \rho_2)S_2, v_2)$. The unique successor v_2' of v_2 is labelled by $q : (N, \rho_2, (v_2, K, \rho_2)S_2)$. So the pair (v_1', v_2') satisfies the invariant.

Let us now examine the case when Adam chooses a node of the form (4). By definition of $R(v_2)$ this means that in $\mathcal{K}_\sigma$ there is a node v_2' labelled $q' : (x, \rho_2', S_2')$ with $\rho_2'(x) = (v_2, K, \rho_2)$ and $res(S_2', v_2') = R_1 \dots R_l$. Moreover $r' = \max(v_2, v_2')$. The successor v_2'' of v_2' is labelled by $q' : (K, \rho_2, S_2')$. We can take it as a companion for v_1' since by easy calculation $\rho_1 \restriction_{r'} = res(\rho_2, v_2) \restriction_{\max(v_2, v_2'')} = res(\rho_2, v_2'')$. Hence the strategy is able to preserve the invariant.

It is easy to check that the above strategy is winning. It remains to check what happens when a maximal play is finite. This means that the path ends in a pair (v_1, v_2) where v_1 is a variable node. Such a node is labelled by $q : (x, \rho_1, S_1)$. To show that this position is winning requires a small calculation proving that $(q, \Omega(q)) \in R_x(S_1)$ for $R_x = \rho_1(x)$.

6 Global Model Checking

The finite game $\mathcal{G}(\mathcal{A}, M)$ allows us also to compute a finite representation of the set of winning positions of Eve in the game $\mathcal{K}(\mathcal{A}, M)$. For this we represent positions in the later game as trees and show that the set of winning positions for Eve is regular: the tree representations of winning positions are recognizable by a finite tree automaton.

Recall that positions of $\mathcal{K}(\mathcal{A}, M)$ are of the form $q : (N, \rho, S)$ where N is a subterm of M, ρ is an environment assigning a closure to every free variable of N, and S is a stack of closures. Recall also that terms from all the closures of ρ and S are subterms of M.

We start by defining a representation of closures as trees. We take the set of all subterms of M as the alphabet: the arity of a letter N being the number of free variables in N. So, for example, if N does not have free variables then a node labelled by N is a leaf in a tree. When N has free variables $x_1, \dots, x_l$; a closure (N, ρ) is represented by a tree whose root is labelled by N and the subtree t_i rooted in i-th child represents $\rho(x_i)$; for $i = 1, \dots, l$. For t of this form, we write $term(t)$ to denote the lambda term obtained by substituting $term(x_i)$ for x_i in N, for $i = 1, \dots, l$. Observe that $term(t)$ is closed: it has no free variables.

A position $q : (N, \rho, S)$ of $\mathcal{K}(\mathcal{A}, M)$ is represented as a tree whose root labelled $q : \tau_N$ has the sequence of children: the tree rooted in the first child representing (N, ρ), and the others representing the closures from S in the same order as in S. Hence the number of children of the root depends on the size of S that in turn is determined by the type τ_N of N.

Since representations of configurations are finite trees over a finite ranked alphabet, we can use standard finite automata to recognize sets of such trees. This gives a notion of a regular set of positions of $\mathcal{K}(\mathcal{A}, M)$.

Theorem 2. *For every $\mathcal{A}$ and M: the set of representations of positions of $\mathcal{K}(\mathcal{A}, M)$ that are winning for Eve is regular.*

The proof uses reduction to finite games. Let $term(N, \rho, S)$ be the term denoted by the closure (N, ρ) applied to terms denoted by the closures in S. It is a closed term of type 0. Of course the behaviours of the Krivine machine from

(N, ρ, S) and $(term(N, \rho, S), \emptyset, \varepsilon)$ are the same, that is they give the same Böhm trees. This implies that Eve wins from $q : (N, \rho, S)$ in $K(\mathcal{A}, M)$ iff she wins from $q : (term(N, \rho, S), \emptyset, \varepsilon)$ in $K(\mathcal{A}, term(N, \rho, S))$. By the reduction theorem (Theorem 1) the later is equivalent to Eve winning from $q : (term(N, \rho, S), \emptyset, \varepsilon)$ in the finite game $G(\mathcal{A}, term(N, \rho, S))$. This condition can be checked by a finite alternating automaton that essentially applies the rules of the construction of the finite game.

7 Conclusions

We have proposed to use Krivine machines to analyse higher-order recursive schemes. The rich structure of this formalism allows to write compact and powerful invariants on computation giving a rather succinct proof of decidability of local model checking. The result on global model checking shows that the structure of configurations of the Krivine machine, although rich, is quite easy to work with.

The proof of Kobayashi and Ong [KO09] is a remarkable achievement showing that one can prove the result with the assumption method (in the spirit of [Wal01]) on the level of terms instead of higher-order pushdown automata with collapse (CPDA). Our residuals are very similar to the additional indices in types introduced in that paper. Also handling of ranks via $\lfloor_{\Omega(q)}$ operation is similar in both proofs. The typing rule for application gives naturally essentially the same rule as we use here. The finite game in that paper is rather different though, as the typing system of Kobayashi and Ong has not been designed to handle fixpoints or lambda-abstraction. The proof of the correctness of the reduction is just different since without configurations of the Krivine machine it is very difficult to state the correspondence between nodes in the tree generated by the scheme and nodes in the finite game.

In his original proof Ong [Ong06] constructed an infinite term, called computation tree, and then showed how to find the Böhm tree inside it. So naturally he concentrated on lambda-abstractions and variables. To find the paths of the Böhm tree he needed a fine description of computation offered by game semantics. Here, thanks to our invariants expressed in terms of closures, we manage to avoid the problem of finding paths of the Böhm tree inside the computation tree. This is probably the biggest technical simplification of our approach.

In the present paper we have kept models of λY-calculus in the background. Yet, the two proofs strongly suggest that there is a finitary model where we can calculate the behaviour of a fixed automaton on a given term. It would be very interesting to find a useful representation of this model.

References

[AdMO05] Aehlig, K., de Miranda, J.G., Ong, C.-H.L.: Safety is not a restriction at level 2 for string languages. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 490–504. Springer, Heidelberg (2005)

[BCOS10] Broadbent, C., Carayol, A., Ong, L., Serre, O.: Recursion schemes and logical reflection. In: LICS, pp. 120–129 (2010)

[BO09] Broadbent, C., Ong, C.-H.L.: On global model checking trees generated by higher-order recursion schemes. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 107–121. Springer, Heidelberg (2009)

[CHM+08] Carayol, A., Hague, M., Meyer, A., Ong, L., Serre, O.: Winning regions of higher-order pushdown games. In: LICS, Pittsburgh, United States, pp. 193–204 (2008)

[CR58] Curry, H.B., Feys, R.: Combinatory Logic, vol. 1. North-Holland Publishing Co., Amsterdam (1958)

[Dam82] Damm, W.: The IO- and OI-hierarchies. Theoretical Computer Science 20, 95–207 (1982)

[DF80] Damm, W., Fehr, E.: A schematalogical approach to the alalysis of the procedure concept in Algol-languages. In: CLAAP, vol. 1, pp. 130–134. Université de Lille (1980)

[dM06] de Miranda, J.: Structures generated by Higher-Order Grammars and the Safety Constraint. PhD thesis, Oxford University (2006)

[HMOS08] Hague, M., Murawski, A.S., Ong, C.-H.L., Serre, O.: Collapsible pushdown automata and recursion schemes. In: LICS, pp. 452–461 (2008)

[Hue76] Huet, G.: Résolution d'équations dans des langages d'ordre 1,2,.,ω. Thèse de doctorat en sciences mathématiques, Université Paris VII (1976)

[Kar10] Kartzow, A.: Collapsible pushdown graphs of level 2 are tree-automatic. In: STACS. LIPIcs, vol. 5, pp. 501–512 (2010)

[KNU02] Knapik, T., Niwinski, D., Urzyczyn, P.: Higher-order pushdown trees are easy. In: Nielsen, M., Engberg, U. (eds.) FOSSACS 2002. LNCS, vol. 2303, pp. 205–222. Springer, Heidelberg (2002)

[KNUW05] Knapik, T., Niwinski, D., Urzycyzn, P., Walukiewicz, I.: Unsafe grammars and pannic automata. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1450–1461. Springer, Heidelberg (2005)

[KO09] Kobayashi, N., Ong, L.: A type system equivalent to modal mu-calculus model checking of recursion schemes. In: LICS, pp. 179–188 (2009)

[Kri07] Krivine, J.-L.: A call-by-name lambda-calculus machine. Higher-Order and Symbolic Computation 20(3), 199–207 (2007)

[Ong06] Luke Ong, C.-H.: On model-checking trees generated by higher-order recursion schemes. In: LICS, pp. 81–90 (2006)

[Plo77] Plotkin, G.D.: LCF considered as a programming language. Theor. Comput. Sci. 5(3), 223–255 (1977)

[SW11] Salvati, S., Walukiewicz, I.: Krivine machines and higher-order schemes (2011), `http://www.labri.fr/perso/salvati/downloads/articles/hpda-dec.pdf`

[Wal01] Walukiewicz, I.: Pushdown processes: Games and model checking. Information and Computation 164(2), 234–263 (2001)

[Wan07] Wand, M.: On the correctness of the Krivine machine. Higher-Order and Symbolic Computation 20, 231–235 (2007), 10.1007/s10990-007-9019-8