

## Krivine machines and higher-order schemes

Sylvain Salvati, Igor Walukiewicz

► **To cite this version:**

Sylvain Salvati, Igor Walukiewicz. Krivine machines and higher-order schemes. [Research Report] 2011, pp.17. <inria-00589407>

**HAL Id: inria-00589407**

**<https://hal.inria.fr/inria-00589407>**

Submitted on 28 Apr 2011

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Krivine machines and higher-order schemes

S. Salvati<sup>1</sup> and I. Walukiewicz<sup>1</sup>

Université de Bordeaux, INRIA, CNRS, LaBRI UMR5800  
LaBRI Bât A30, 351 crs Libération, 33405 Talence, France

**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  $\lambda Y$ -calculus. Since they were investigated mainly in formal language community, the tools developed were by large inspired by 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 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.

In the last decade 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*. Subsequently, higher order pushdowns have been extended with *panic* operation to handle all level 2 schemes [KNUW05], 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<sup>+</sup>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 recursion schemes. First, instead of using Damm’s definition of higher-order schemes, we turn to the  $\lambda Y$ -calculus as the mean 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 [Plot77]. Second, instead of using higher-order collapsible automata as an abstract machine, we use Krivine abstract machine [Kri07]. This machine is much closer to  $\lambda$ -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  $\lambda Y$ -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, at least at the level of presentation. 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.

*Related work* We have already mentioned a body of related work, we will comment more on the proof of Kobayashi and Ong [KO09] in the concluding section. Concerning the global model-checking result, Carayol et al. [CHM<sup>+</sup>08] showed regularity of winning regions in parity games over higher-order pushdown automata without collapse. More recently, Broadbent and Ong [BO09] showed that winning positions of a parity game generated by an order  $n$  recursive scheme are recognizable by a non-deterministic collapsible pushdown automaton. The proof uses game semantics instead of automata. Finally, Broadbent et al. [BCOS10] show that the winning positions can be also recognized by a deterministic collapsible pushdown automaton. Here we show that in a different representation they are recognizable by a tree automaton. 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  $\lambda Y$  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 appendix.

## 2 Basic notions

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

A *signature*, denoted  $\Sigma$ , 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  $\omega^\alpha$  and  $Y^{(\alpha \rightarrow \alpha) \rightarrow \alpha}$  standing for the fixpoint operator and the undefined value. For simplicity of notation we assume that all other constants are of type  $0 \rightarrow 0 \rightarrow 0$ .

The set of *simply typed  $\lambda$ -terms* is defined inductively as follows. A constant of type  $\alpha$  is a term of type  $\alpha$ . For each type  $\alpha$  there is a countable set of variables  $x^\alpha, y^\alpha, \dots$  that are also terms of type  $\alpha$ . If  $M$  is a term of type  $\beta$  and  $x^\alpha$  a variable of type  $\alpha$  then  $\lambda x^\alpha.M$  is a term of type  $\alpha \rightarrow \beta$ . Finally, if  $M$  is of type  $\alpha \rightarrow \beta$  and  $N$  is a term of type  $\alpha$  then  $MN$  is a term of type  $\beta$ . Together with the usual operational semantics of  $\lambda$ -calculus, that is  $\beta$ -reduction, we use  $\delta$ -reduction ( $\rightarrow_\delta$ ) giving the semantics to the fixpoint operator:  $YM \rightarrow_\delta M(YM)$ . Thus, the operational semantics of the  $\lambda Y$ -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).

A *Böhm tree* is an unranked ordered, and potentially infinite tree with nodes labelled, by  $\omega^\alpha$  or terms of the form  $\lambda x_1 \dots x_n.N$ , where  $N$  is a variable or a constant; and the sequence of lambda abstractions is optional. So for example  $x^0$ ,  $\lambda x.\omega^0$  are labels, but  $\lambda y^0.x^{0 \rightarrow 0}.y^0$  is not. A Böhm tree of a term  $M$  is obtained as follows. If  $M \rightarrow_{\beta\delta}^* \lambda \mathbf{x}.N_0 N_1 \dots N_k$  with  $N_0$  a variable or a constant then the root of  $BT(M)$  is labelled by  $\lambda \mathbf{x}.N_0$  and has  $BT(N_1), \dots, BT(N_k)$  as a sequence of its children. If  $M$  is not solvable then  $BT(M) = \omega^\alpha$ , where  $\alpha$  is the type of  $M$ .

*From recursive schemes to  $\lambda Y$ -calculus* A recursive scheme is a set of equations defining a  $\lambda Y$ -term by mutual recursion. Formally, a recursion scheme is a function  $\mathcal{R}$  assigning to every variable  $F^\alpha$  from a finite set  $\mathcal{N}$ , a term  $M$  of type  $\alpha$  and with free variables only from  $\mathcal{N}$ . Fixing  $F^0$  in  $\mathcal{N}$  as *the starting symbol*, the semantics of a scheme is the infinite tree computed by unfolding the definitions of the variables starting from  $F^0$ . This tree, can also be seen as the Böhm tree generated from  $F^0$  by recursively applying the substitution defined by  $\mathcal{R}$ .

*Krivine machine* A Krivine machine [Kri07], is an abstract machine that computes the weak head normal form of a  $\lambda$ -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:

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

As in this grammar, we will use  $\emptyset$  for the empty environment. We require that in a closure  $(M, \rho)$ , the environment is defined for every free variable of  $M$ . Intuitively such a closure denotes closed  $\lambda$ -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  $(M, \rho, S)$ , where  $M$  is a term,  $\rho$  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:

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

Note that the machine is deterministic. We will be only interested in configurations accessible from  $(M_0, \emptyset, \varepsilon)$  for some term  $M_0$  of type 0. Every such configuration  $(M, \rho, S)$  enjoys very strong typing invariants. Environment  $\rho$  associates to a variable  $x^\alpha$  a closure  $(N, \rho')$  so that  $N$  has type  $\alpha$ ; we will say that the closure is of type  $\alpha$  too. If  $M$  has type  $\alpha_1 \rightarrow \dots \rightarrow \alpha_n \rightarrow 0$ , then  $S$  is a stack of  $n$  closures, with  $i$ -th closure from the top being of type  $\alpha_i$ .

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 with a term starting with a constant. Since all our constants are of type  $0 \rightarrow 0 \rightarrow 0$ , such a configuration has the form  $(a, \rho, (M_0, \rho)(M_1, \rho))$  (that we shall write  $(a(M_0, M_1), \rho, \varepsilon)$ ). At that point we create a node labelled  $a$  and start reducing both  $(M_0, \rho, \varepsilon)$  and  $(M_1, \rho, \varepsilon)$ . 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  $(N, \rho, S)$  is reachable from  $(M, \emptyset, \varepsilon)$  then  $N$ , and the terms that occur in  $\rho$  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 remains finite.

We present an execution of a Krivine machine on an example taken from [KO09]. We will use constants  $a : 0 \rightarrow 0 \rightarrow 0$ ,  $b : 0 \rightarrow 0$  and  $c : 0$ . The scheme is defined by  $S \mapsto Fc$  and  $F \mapsto \lambda x.ax(Fbx)$  which can be represented by the following term in the  $\lambda Y$ -calculus:

$$YM c \quad \text{where } M = \lambda f x.a x (f(bx)).$$

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

$$\begin{aligned} (YM c, \emptyset, \varepsilon) &\rightarrow (YM, \emptyset, (c, \emptyset)) \rightarrow (M(YM), \emptyset, (c, \emptyset)) \rightarrow (M, \emptyset, (YM, \emptyset)(c, \emptyset)) \rightarrow \\ &(\lambda x.a x (f(bx)), [f \mapsto (YM, \emptyset)], (c, \emptyset)) \rightarrow \\ &(a x (f(bx)), [f \mapsto (YM, \emptyset)][x \mapsto (c, \emptyset)], \varepsilon) \end{aligned}$$

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:

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

*Parity automata and the definition of the problem* Recall that  $\Sigma$  is a fixed set of constants of type  $0 \rightarrow 0 \rightarrow 0$ . 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

$$\mathcal{A} = \langle Q, \Sigma, q^0 \in Q, \delta : Q \times \Sigma \rightarrow \mathcal{P}(Q^2), \Omega : Q \rightarrow [d] \rangle \quad (1)$$

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

In general, an infinite binary tree is a function  $t : \{0, 1\}^* \rightarrow \Sigma$ . A run of  $\mathcal{A}$  on  $t$  is a function  $r : \{0, 1\}^* \rightarrow Q$  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  $\Omega$ ; this means that the maximal rank of a state seen infinitely often should be even.

**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 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  $v$  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  $v'$  closure.

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

**Definition 2.** We use 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 0. We can use max parity condition to decide who wins an infinite play. Let us call the resulting game  $\mathcal{K}(\mathcal{A}, M)$ .

The following is a direct consequence of the definitions.

**Proposition 1.** Eve has a strategy from the root position in  $\mathcal{K}(\mathcal{A}, M)$  iff  $\mathcal{A}$  accepts  $Tree(M)$ .

Hence the model checking problem reduces to deciding who has a winning strategy from the root of  $\mathcal{K}(\mathcal{A}, M)$ . To decide this we will define a finite game and then show that the winner in this game is the same as the winner in  $\mathcal{K}(\mathcal{A}, M)$ .

## 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 the 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 we will put an assumption as to how we will get to a position where the closure is used. At the same time we will check what happens when we use the closure under the assumptions we have made. Since the closure can be of higher type, the assumptions are a bit more complicated than in 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  $\mathcal{A}$ . For every type  $\tau = \tau_1 \rightarrow \dots \rightarrow \tau_k \rightarrow 0$  the set of residuals  $D_\tau$  is the set of functions  $D_{\tau_1} \rightarrow \dots \rightarrow D_{\tau_k} \rightarrow \mathcal{P}(Q \times [d])$ .

For example, we have that  $R_0$  is  $\mathcal{P}(Q \times [d])$  and  $R_{0 \rightarrow 0}$  is  $\mathcal{P}(Q \times [d]) \rightarrow \mathcal{P}(Q \times [d])$ . The meaning of residuals will become clearer when we will define the game.

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

$$q : (N, \rho, S), \quad (q_0, q_1) : (N, \rho, S) \quad (q, R) : (N, \rho, S)$$

where  $q, q_0, q_1$  are states of  $\mathcal{A}$ ,  $N$  is a term (more precisely a subterm of  $M$ ),  $\rho$  is an environment assigning a residual to every variable that has a free occurrence in  $N$  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  $R : D_{\tau_1} \rightarrow \dots \rightarrow D_{\tau_k} \rightarrow D_0$ . Recall that  $D_0 = \mathcal{P}(Q \times \mathbb{N})$ . We define  $R \downarrow_r$  to be the function such that for every sequence of arguments  $S$ :

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

The idea is that  $(q_1, r_1) \in R(S)$  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. 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. We state a simple but useful property of the operation.

**Lemma 1.** *For every residual  $R$  and ranks  $r_1, r_2$ :  $(R \downarrow_{r_1}) \downarrow_{r_2} = R \downarrow_{\max(r_1, r_2)}$ .*

If  $\rho$  is an environment then  $\rho \downarrow_r$  is an environment such that for every  $x$ :  $(\rho \downarrow_r)(x) = \rho(x) \downarrow_r$ .

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  $\mathcal{K}(\mathcal{A}, M)$ :

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

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

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

for every residual  $R : D_{\tau_1} \rightarrow \dots \rightarrow D_{\tau_k} \rightarrow D_0$ . From this position we have transitions

$$\begin{aligned} (q, R) &: (NK, \rho, S) \rightarrow q : (N, \rho, R \downarrow_{\Omega(q)} \cdot S) \\ (q, R) &: (NK, \rho, S) \rightarrow q' : (K, \rho \downarrow_{r'}, R_1 \dots R_l) \text{ for every } R_1 \in D_{\tau_1}, \dots, R_l \in D_{\tau_l} \\ &\quad \text{and } (q', r') \in R \downarrow_{\Omega(q)}(R_1, \dots, R_l). \end{aligned}$$

Here  $R \downarrow_{\Omega(q)}$  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  $(q, \Omega(q)) \in R_x(R_1, \dots, R_k)$ .

Finally, we need to define ranks. It will be much simpler to define ranks on transitions instead of nodes. All the transitions will have rank 0 but for two cases:

- a transition  $(q, R) : (NK, \rho, S) \rightarrow q' : (K, \rho \upharpoonright_{r'}, R_1 \cdots R_k)$  has rank  $r'$ ;
- $(q_0, q_1) : (a(N_0, N_1), \rho, S) \rightarrow q_i : (N_i, \rho \upharpoonright_{\Omega(q_i)}, S \upharpoonright_{\Omega(q_i)})$  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 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  $\mathcal{K}(\mathcal{A}, M)$ . In the appendix we show with very similar arguments the analogous fact for Adam.

Let us fix a winning strategy  $\sigma$  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  $\mathcal{K}(\mathcal{A}, M)$  that will use  $\sigma$  to guess residual 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 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 closure  $v$ . We also define a variation of this notion: a residual  $R(v)$  seen from a node  $v'$ , denoted  $res(v, v')$ . The two notions are the main technical tools used in the proof of the theorem. Before proceeding we will need one simple abbreviation. If  $v$  is an ancestor of  $v'$  in  $\mathcal{K}_\sigma$  then we write  $\max(v, v')$  for the maximal rank appearing on the path between  $v$  and  $v'$ , including both ends.

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  $(v, K, \rho)$  is created in  $v$ . We will look at all the places where this closure is used and summarize the information about them in  $R(v)$ .

First, suppose that the closure, or equivalently the term  $K$ , is of type 0. The residual  $R(v)$  should be also of type 0 which means that  $R(v) \subseteq Q \times [d]$ . We put

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

for every node  $v'$  in  $\mathcal{K}_\sigma$  labelled  $q' : (x, \rho', \varepsilon)$  such that  $\rho'(x) = (v, K, \rho)$ . Additionally, for every descendant  $v_1$  of  $v$  in  $\mathcal{K}_\sigma$  we define  $\text{res}(v, v_1) = R(v) \downarrow_{\max(v, v_1)}$ ; intuitively it is  $R(v)$  seen from  $v_1$ .

For the induction step, suppose that  $K$  is of type  $\tau_1 \rightarrow \dots \rightarrow \tau_k \rightarrow 0$  and that we have already calculated residuals for all closures of types  $\tau_1, \dots, \tau_k$ . Suppose that we have a closure  $(v, K, \rho)$  created at a node  $v$ . This time  $R(v) : D_{\tau_1} \rightarrow \dots D_{\tau_k} \rightarrow \mathcal{P}(Q \times [d])$ . Take a node  $v'$  using the closure. Its label has the form  $q' : (x, \rho', S')$  for some  $x, \rho'$  and  $S'$  such that  $\rho'(x) = (v, K, \rho)$ . The stack  $S'$  has the form  $(v_1, N_1, \rho_1) \dots (v_k, N_k, \rho_k)$  with  $N_i$  of type  $\tau_i$ . We put

$$(q', \max(v, v')) \in R(\text{res}(v_1, v'), \dots, \text{res}(v_k, v')) . \quad (2)$$

As above, for every descendant  $v_1$  of  $v$  we define  $\text{res}(v, v_1) = R(v) \downarrow_{\max(v, v_1)}$ .

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

*The strategy in  $G(\mathcal{A}, M)$*  Now we are ready to define the strategy for Eve in  $G(\mathcal{A}, M)$ . This strategy will use positions in the game  $\mathcal{K}(\mathcal{A}, M)$  and the strategy  $\sigma$  as hints. The strategy will take a pair of positions  $(v_1, v_2)$  with  $v_1$  in  $G(\mathcal{A}, M)$  and a  $v_2$  in  $\mathcal{K}(\mathcal{A}, M)$ . Then the strategy will 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  $\sigma$ . Moreover, all visited pairs  $(v_1, v_2)$  will satisfy the following invariant:

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

The initial positions in both games have the same labels 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 it is evident what the strategy should do. 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  $\rho_1 = \text{res}(\rho_2, v_2)$  and  $S_1 = \text{res}(S_2, v_2)$ . 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

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

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

Suppose Adam chooses (3). By definition  $R(v_2) \downarrow_{\Omega(q)} = \text{res}(v_2, v_2)$ . Hence the stack  $(R(v_2) \downarrow_{\Omega(q)} \cdot S_1)$  is just  $\text{res}((v_2, K, \rho_2)S_2, v_2)$ . 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 that 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  $\text{res}(S'_2, v'_2) = R_1 \dots R_k$ . 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  $\rho_1 \downarrow_{r'} = \text{res}(\rho_2, v_2) \downarrow_{\max(v_2, v'_2)} = \text{res}(\rho_2, v''_2)$  by Lemma 3. Hence the strategy is able to preserve the invariant.

In the appendix we prove that the strategy is winning. The main point is to check what happens in variable nodes.

## 6 Global model checking

In this section we will show how to compute a finite representation of the set of winning positions of Eve in the game  $\mathcal{K}(\mathcal{A}, M)$ . For this we will first define a, rather straightforward, representation of positions of the game as trees. We will then 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  $M$ ,  $\rho$  is an environment assigning a closure to every free variable of  $N$ , and  $S$  is a stack of closures. Recall that terms from all the closures of  $\rho$  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  $(N, \rho)$  is represented by a tree whose root is labelled by  $N$  and the subtree  $t_i$  rooted in  $i$ -th child representing  $\rho(x_i)$ ; for  $i = 1, \dots, l$ . For  $t$  of this form, we write  $\text{term}(t)$  to denote the lambda term obtained by substituting  $\text{term}(x_i)$  for  $x_i$  in  $N$ , for  $i = 1, \dots, l$ . Observe that  $\text{term}(t)$  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  $(N, \rho)$ , 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  $\tau_N$  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.*

In order to prove the theorem we define an alternating tree automaton, and show that it accepts the desired set of configurations.

Of course we would like to use our reduction from infinite to finite games. Let  $term(N, \rho, S)$  be the term denoted by the closure  $(N, \rho)$  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  $(N, \rho, S)$  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  $\mathcal{K}(\mathcal{A}, M)$  iff she wins from  $q : (term(N, \rho, S), \emptyset, \varepsilon)$  in  $\mathcal{K}(\mathcal{A}, term(N, \rho, S))$ . 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 is this last condition that our automaton will check.

At the core of the construction we will have an automaton recognizing closures. Its states will be pairs  $(q, R)$ , where  $q$  is a state and  $R$  a sequence of residuals. In a state  $(q, R)$  when reading a symbol  $N$  the automaton will do the following:

1. It will check if the type of  $N$  corresponds to the type of  $R$ , namely that  $R$  is a sequence of arguments of right types. The automaton will reject if it is not the case.
2. Next, it will guess an assignment  $\rho_N$  of residuals to free variables of  $N$ , such that the position  $q : (N, \rho_N, R)$  is winning for Eve in  $G(\mathcal{A}, M)$ .
3. For every free variable  $x_i$  of  $N$ , every stack  $R_i$  of residuals of appropriate type and every  $q_i$  such that  $(q_i, r_i) \in \rho_N(x_i)(R_i)$  the automaton will send a copy of itself with the state  $(q_i, R_i)$  to the  $i$ -th child of the root. If there are no copies to be send then the automaton just accepts.

Before we describe the language accepted by this automaton in the next Lemma, we need to define a slight generalization of the finite games introduced section 4. Given a closed term  $N$  of type  $\alpha$  ( $\alpha$  may be higher-order) we define  $G(\mathcal{A}, N)$  to be the game that has the same transitions and the same winning conditions as the one in section 4 and that contains all the positions  $q : (N, \emptyset, R)$  where  $q$  is a state of  $\mathcal{A}$  and  $R$  is a sequence of residuals with the correct types with respect to  $N$ .

**Lemma 2.** *For every tree  $t$  representing a closure  $(N, \rho)$  every sequence  $R$  of residuals of a type determined by the type of  $N$ , and every state  $q$ :*

*$\mathcal{A}$  accepts  $t$  from  $(q, R)$  iff  $q : (term(t), \emptyset, R)$  is winning for Eve in  $G(\mathcal{A}, term(t))$*

The lemma allows us to construct an automaton recognizing winning positions. Recall that a position  $q : (N, \rho, S)$  is encoded as with  $q, \tau_N$  in the root, one son for  $(N, \rho)$  and a son for every element of  $S$ . In the initial state when reading  $q : \tau_N$  the automaton guesses a sequence of residuals  $R$  of types determined by  $\tau_N$ . It then starts the above automaton in the state  $(q, R)$  on the son representing  $(N, \rho)$ . On the son representing the  $i$ -th element of the stack it

starts the above automaton in  $(q_i, R_i)$  for every  $R_i$  of appropriate type and  $q_i$  such that  $(q_i, r) \in R(R_i)$ . In short, the automaton simulates several instances of the application rule.

## 7 Conclusions

In this paper we have proposed to use Krivine machines to analyse higher-order recursive schemes. Using two prominent results in the area we have demonstrated that rich structure of this formalism allows to write compact and powerful invariants on computation. The proof of decidability of local model checking gives a good example of this. The proof for global model checking shows that the structure of configurations of the Krivine machine, although rich, is quite easy to work with. This said Krivine machine is a very sophisticated model despite its simple presentation. As it happens, its rich and relatively rigid structure appears to be a good frame that helps formulating strong invariants of the Böhm tree it computes with rather elementary definitions.

Let us give some more comments on the relations with the proof of Kobayashi and Ong [KO09]. The later has been 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 CPDA. Our residuals are very similar to the additional indices in types introduced in that paper. Also handling of ranks via  $\downarrow_{\Omega(q)}$  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 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 our opinion, the presented proof of decidability of global model checking is an important argument in favor of the use of Krivine machines. With CPDA, the only induction parameter available is the rank of the stack. The result in [BCOS10] is proved by reducing the stack level one by one. This is technically very difficult.

In the present paper we have kept models of  $\lambda Y$ -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. The main obstacle is to understand the meaning of the fixpoint operator.

## References

- BCOS10. Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong, and Olivier Serre. Recursion schemes and logical reflection. In *Proceedings of the 2010 25th Annual IEEE Symposium on Logic in Computer Science, LICS '10*, pages 120–129, Washington, DC, USA, 2010. IEEE Computer Society.

- BO09. C. Broadbent and C.-H. Luke Ong. On global model checking trees generated by higher-order recursion schemes. In *Proceedings of 12th International Conference on Foundations of Software Science and Computational Structures (FOSSACS 2009)*, volume 5504 of *LNCS*, pages 107–121. Springer, 2009.
- CHM<sup>+</sup>08. Arnaud Carayol, Matthew Hague, Antoine Meyer, Luke Ong, and Olivier Serre. Winning regions of higher-order pushdown games. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008 Twenty-Third Annual IEEE Symposium on Logic in Computer Science (LICS 2008)*, pages 193–204, Pittsburgh United States, 2008. IEEE Computer Society.
- CR58. H.B. Curry and R. Feys. *Combinatory Logic*, volume 1. North-Holland Publishing Co., Amsterdam, 1958.
- Dam82. Werner Damm. The IO- and OI-hierarchies. *Theoretical Computer Science*, 20:95–207, 1982.
- DF80. Werner Damm and Elfriede Fehr. A schematological approach to the analysis of the procedure concept in algol-languages. In *CLAAP*, pages 130–134. Université de Lille 1, 1980.
- HMOS08. Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, pages 452–461. IEEE Computer Society, 2008.
- Kar10. Alexander Kartzow. Collapsible pushdown graphs of level 2 are tree-automatic. In Jean-Yves Marion and Thomas Schwentick, editors, *STACS*, volume 5 of *LIPICs*, pages 501–512. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.
- KNU02. Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In Springer, editor, *Proc. FoSSaCS'02*, volume 2303, pages 205–222, 2002.
- KNUW05. Teodor Knapik, Damian Niwinski, Pawel Urzyczyn, and Igor Walukiewicz. Unsafe grammars and pannic automata. In Springer, editor, *ICALP 05*, number 3580 in *LNCS*, pages 1450–1461, 2005.
- KO09. Naoki Kobayashi and Luke Ong. A type system equivalent to modal mu-calculus model checking of recursion schemes. In *Proceedings of 24th Annual IEEE Symposium on Logic in Computer Science (LICS 2009), Los Angeles*, pages 179–188, 2009.
- Kri07. Jean-Louis Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, 2007.
- Ong06. C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90. IEEE Computer Society, 2006.
- Pl077. Gordon D. Plotkin. Lcf considered as a programming language. *Theor. Comput. Sci.*, 5(3):223–255, 1977.
- Wal01. Igor Walukiewicz. Pushdown processes: Games and model checking. *Information and Computation*, 164(2):234–263, 2001.
- Wan07. Mitchell Wand. On the correctness of the krivine machine. *Higher-Order and Symbolic Computation*, 20:231–235, 2007. 10.1007/s10990-007-9019-8.

## A Proofs from Section 4

**Lemma 3.** For every residual  $R$  and ranks  $r_1, r_2$ :  $(R \downarrow_{r_1}) \downarrow_{r_2} = R \downarrow_{\max(r_1, r_2)}$ .

*Proof.* Suppose  $r_1 \geq r_2$ . We show  $(R \downarrow_{r_1}) \downarrow_{r_2} = R \downarrow_{r_1}$ . Fix a state  $q$  and look at all the pairs with this state. The pairs  $(q, r)$  with  $r > r_1$  are the same in  $R$  and  $(R \downarrow_{r_1}) \downarrow_{r_2}$ . If  $(q, r_1) \notin R$  then there is no pair  $(q, r)$  in  $R \downarrow_{r_1}$  with  $r \leq r_1$ , and  $\downarrow_{r_2}$  does nothing for pairs with state  $q$ . If  $(q, r_1) \in R$  then all the pairs  $(q, r)$  for  $r \leq r_1$  are in  $R \downarrow_{r_1}$ . The operation  $\downarrow_{r_2}$  does not add or remove any pairs with the state  $q$ .

When  $r_1 < r_2$  we show that  $(R \downarrow_{r_1}) \downarrow_{r_2} = R \downarrow_{r_2}$ . The proof is similar.

## B From Eve's winning strategy in $\mathcal{K}(\mathcal{A}, M)$ to his winning strategy in $G(\mathcal{A}, M)$

We need to show that the strategy defined in Section 5 is winning. Consider a sequence of nodes  $(v_1^1, v_2^1), (v_1^2, v_2^2), \dots$  consistent with the strategy. Suppose that this sequence is infinite. By construction we have that  $v_2^1, v_2^2, \dots$  is a path in  $\mathcal{K}_\sigma$ , hence a play winning for Eve. We have defined the strategy in such a way that a rank of a transition from  $v_1^i$  to  $v_1^{i+1}$  is the same as the maximal rank of a node on the path between  $v_2^i$  and  $v_2^{i+1}$ . Hence  $v_1^1, v_1^2, \dots$  is winning for Eve too.

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 Eve wins here we need to prove that

$$(q, \Omega(q)) \in R_x(R_1, \dots, R_k) \text{ where } R_x = \rho_1(x) \text{ and } (R_1 \dots R_k) = S_1.$$

By the invariant we have that the companion node  $v_2$  is labelled by  $q : (x, \rho_2, S_2)$  and  $\rho_1 = \text{res}(\rho_2, v_2)$ ,  $S_1 = \text{res}(S_2, v_2)$ . Suppose that  $\rho_2(x) = (v, N, \rho)$ . We have  $R_x = R(v) \downarrow_{\max(v, v_2)}$ , since  $\rho_1 = \text{res}(\rho_2, v_2)$ . By definition of  $R(v)$  we get  $(q, \max(v, v_2)) \in R(v)(\text{res}(S_2, v_2))$ . Then from the definition of  $\downarrow_{\max(v, v_2)}$  operation:  $(q, \max(v, v_2)) \in R(v)(\text{res}(S_2, v_2)) \downarrow_{\max(v, v_2)}$ . Which implies  $(q, \Omega(q)) \in R(v)(\text{res}(S_2, v_2)) \downarrow_{\max(v, v_2)}$  since  $\Omega(q) \leq \max(v, v_2)$ . This is the required statement  $(q, \Omega(q)) \in R_x(R_1, \dots, R_k)$ .

## C From Adam's winning strategy in $\mathcal{K}(\mathcal{A}, M)$ to his winning strategy in $G(\mathcal{A}, M)$

Let us fix a winning strategy  $\theta$  of Adam in  $\mathcal{K}(\mathcal{A}, M)$ , and consider the tree  $\mathcal{K}_\theta$  of plays respecting this strategy. This is a subtree of  $\mathcal{K}(\mathcal{A}, M)$ . We assign a residual to every closure appearing in  $\mathcal{K}_\theta$  in exactly the same way as we have done in the previous section.

**The invariant** In order to formulate the invariant for the strategy we introduce complementarity predicate  $\text{Comp}(R_1, R_2)$  between a pair of residuals:

- For  $R_1, R_2 \in D_0$  we have  $\text{Comp}(R_1, R_2)$  if  $R_1 \cap R_2 = \emptyset$ .

- For  $R_1, R_2 \in D_\tau$  where  $\tau = \tau_1 \rightarrow \dots \rightarrow \tau_k \rightarrow 0$  we have  $\text{Comp}(R_1, R_2)$  if for all  $(R_{1,1}, \dots, R_{1,k}), (R_{2,1}, \dots, R_{2,k}) \in D_{\tau_1} \times \dots \times D_{\tau_k}$  such that  $\text{Comp}(R_{1,i}, R_{2,i})$  for all  $i = 1, \dots, k$  we get  $R_1(R_{1,1}, \dots, R_{1,k}) \cap R_2(R_{2,1}, \dots, R_{2,k}) = \emptyset$ .

For two closures  $(v, N, \rho)$  and  $(v', N, \rho')$  the predicate  $\text{Comp}((v, N, \rho), (v', N, \rho'))$  holds if  $\text{Comp}(R(v), R(v'))$  is true. For two environments  $\rho, \rho'$  we write  $\text{Comp}(\rho, \rho')$  if the two environments have the same domain and for every  $x$ , the predicate  $\text{Comp}(\rho(x), \rho'(x))$  holds. Finally  $\text{Comp}(S, S')$  if the two sequences are of the same length and the predicate holds for every coordinate.

It is important to observe that  $\text{Comp}$  behaves well with respect to  $\downarrow_r$  operation

**Lemma 4.** *If  $\text{Comp}(R_1, R_2)$  then also  $\text{Comp}(R_1 \downarrow_r, R_2 \downarrow_r)$  for every rank  $r$ .*

*Proof.* Take a pair  $(q_1, r_1)$ . In the case when  $r_1 > r$  by definition we have:  $(q_1, r_1) \in R \downarrow_r$  iff  $(q_1, r_1) \in R$ . Hence,  $(q_1, r_1)$  can be in at most one of  $R_1 \downarrow_r, R_2 \downarrow_r$ . When  $r_1 \leq r$  then  $(q_1, r_1) \in R \downarrow_r$  iff  $(q_1, r) \in R$ . Once gain,  $(q_1, r_1)$  can be in at most one of  $R_1 \downarrow_r, R_2 \downarrow_r$ .

Given two sequences  $S_1$  and  $S_2$  of the correct type with respect to  $R_1$  and  $R_2$  and such that  $\text{Comp}(S_1, S_2)$ , since  $\text{Comp}(R_1, R_2)$ , we have  $R_1(S_1) \cap R_2(S_2) = \emptyset$ . Let's suppose that  $(q_1, r_1)$  is in  $R_1 \downarrow_r (S_1)$ , then either  $r_1 > r$  and  $(q_1, r_1)$  is in  $R_1(S_1)$  so that  $(q_1, r_1)$  is neither in  $R_2(S_2)$  nor in  $R_2 \downarrow_r (S_2)$ ; or  $r_1 \leq r$  and  $(q_1, r)$  is in  $R_1(S_1)$  so that  $(q_1, r)$  is not in  $R_2(S_2)$  and  $(q_1, r_1)$  is not in  $R_2 \downarrow_r (S_2)$ . Similarly we get that whenever  $(q_2, r_2)$  is in  $R_2 \downarrow_r (S_2)$  it is not in  $R_1 \downarrow_r (S_1)$ . Therefore, we finally have that  $R_1 \downarrow_r (S_1) \cap R_2 \downarrow_r (S_2) = \emptyset$  and finally that  $\text{Comp}(R_1 \downarrow_r, R_2 \downarrow_r)$ .

As in the case for Eve, the strategy for Adam will take a pair of vertices  $(v_1, v_2)$  from  $G(\mathcal{A}, M)$  and  $\mathcal{K}(\mathcal{A}, M)$ , respectively. It will then consult the strategy  $\sigma$  for Adam in  $\mathcal{K}(\mathcal{A}, M)$  and calculate a new pair  $(v'_1, v'_2)$ . All the pairs will satisfy the invariant:

$$\begin{aligned} &v_1 \text{ labelled by } q : (M, \rho_1, S_1) \text{ and } v_2 \text{ labelled by } q : (M, \rho_2, S_2); \\ &\text{where } \text{Comp}(\rho_1, \text{res}(\rho_2, v_2)) \text{ and } \text{Comp}(S_1, \text{res}(S_2, v_2)); \end{aligned}$$

**The strategy** We define the strategy by considering one by one the rules for constructing the tree  $\mathcal{K}(\mathcal{A}, M)$ . Apart from the immediate cases we have the following.

*Transition rule* If a label of  $v_1$  is of the form  $q : (a(M_0, M_1), \rho_1, \varepsilon)$  then in  $\mathcal{K}_\theta$  this node has a son  $(q_0, q_1) : (a(M_0, M_1), \rho_1, \varepsilon)$  for every  $(q_0, q_1) \in \delta(q, a)$ . The same happens from  $v_2$ : it has successors  $(q_0, q_1) : (a(M_0, M_1), \rho_2, \varepsilon)$ . Now each such successor is a node of Adam so it has itself a unique successor in  $\mathcal{K}_\theta$ . Let us suppose that Adam chooses  $q_0 : (M_0, \rho_2, \varepsilon)$ . We make the strategy to choose from  $(q_0, q_1) : (a(M_0, M_1), \rho_1, \varepsilon)$  the node  $v'_2$  labelled  $q_0 : (M_0, \rho_1 \downarrow_{\Omega(q_0)}, \varepsilon)$ . By definition and Lemma 3 we have that  $\text{res}(\rho_2, v'_2) = \text{res}(\rho_2, v_2) \downarrow_{\Omega(q_0)}$ . It follows from Lemma 4 that  $\text{Comp}(\rho_1 \downarrow_{\Omega(q_0)}, \text{res}(\rho_2, v'_2))$  holds.



*Application rule* Eve can choose a transition

$$q : (MN, \rho_1, S_1) \rightarrow (q, R) : (MN, \rho_1, S_1)$$

for some residual  $R$  and then Adam has a choice between the transitions:

$$\begin{aligned} (q, R) : (MN, \rho_1, S_1) &\rightarrow q : (M, \rho_1, R \downarrow_{\Omega(q)} \cdot S_1) \\ (q, R) : (MN, \rho_1, S_1) &\rightarrow q' : (N, \rho_1 \downarrow_{r'}, R_1 \cdots R_k) \quad \text{for } (q', r') \in R \downarrow_{\Omega(q)} (R_1, \dots, R_k) \end{aligned}$$

At the same time in node  $v_2$  of  $K(\mathcal{A}, M)$  a new closure for  $N$  is created, hence we have a new residual  $R(v_2)$ . We have two cases

Suppose  $\text{Comp}(R \downarrow_{\Omega(q)}, R(v_2))$  holds. In this case Adam chooses for  $v'_1$  the node labelled  $q : (M, \rho_1, R \downarrow_{\Omega(q)} \cdot S_1)$ . For  $v'_2$  he can choose the successor of  $v_2$ . Since it is labeled by  $q : (M, \rho_2, (v_2, N, \rho)S_2)$ , the invariant holds.

The other case is when  $\text{Comp}(R \downarrow_{\Omega(q)}, R(v_2))$  does not hold. This means that there are  $(R_{1,1}, \dots, R_{1,k})$  and  $(R_{2,1}, \dots, R_{2,k})$  such that  $\text{Comp}(R_{1,i}, R_{2,i})$  for all  $i = 1, \dots, k$  and  $R \downarrow_{\Omega(q)} (R_{1,1}, \dots, R_{1,k}) \cap R(v_2)(R_{2,1}, \dots, R_{2,k}) \neq \emptyset$ . Let  $(q', r')$  be the element from the intersection. Since  $(q', r') \in R(v_2)(R_{2,1}, \dots, R_{2,k})$ , there is a node  $v'_2$  labelled by  $q' : (x, \rho'_2, S'_2)$  such that  $\rho'_2(x) = (v_2, N, \rho_2)$  and  $\text{res}(S'_2, v'_2) = (R_{2,1}, \dots, R_{2,k})$ . We choose for  $v'_1$  the node labelled  $q' : (N, \rho_1 \downarrow_{r'}, R_{1,1} \cdots R_{1,k})$ . We need to show that  $\text{Comp}(\rho_1 \downarrow_{r'}, \text{res}(\rho_2, v'_2))$  holds. Take a variable  $y$ , by hypothesis we have  $\text{Comp}(\rho_1(y), \text{res}(\rho_2(y), v_2))$ , since  $\max(v_2, v'_2) = r'$  we have by Lemma 4 that as required  $\text{Comp}(\rho_1(y) \downarrow_{r'}, \text{res}(r_2(y), v_2) \downarrow_{r'})$ .

*The strategy is winning* As in the case of the strategy for Eve, it is easy to show that every infinite play is winning. It remains to check what happens if  $v_1$  is a variable node. Such a node is labelled by  $q : (x, \rho_1, S_1)$ . To show that Adam wins here we need to prove that

$$(q, \Omega(q)) \notin R_x(R_1, \dots, R_k) \text{ where } R_x = \rho_1(x) \text{ and } (R_1 \dots R_k) = S_1.$$

By the invariant, the companion node  $v_2$  is labelled by  $q : (x, \rho_2, S_2)$  and  $\text{Comp}(R_x, \text{res}(\rho_2, v_2)(x))$ ,  $\text{Comp}(S_1, \text{res}(S_2, v_2))$  hold. Suppose  $\rho_2(v_2) = (v, N, \rho)$ . Then  $(q, \max(v, v_2)) \in R(v)(\text{res}(S_2, v_2))$  by the definition of  $R(v)$ . Hence also  $(q, \max(v, v_2)) \in R(v)(\text{res}(S_2, v_2)) \downarrow_{\max(v, v_2)}$ , and in consequence  $(q, \Omega(q)) \in R(v)(\text{res}(S_2, v_2)) \downarrow_{\max(v, v_2)}$ . Since  $R(v) \downarrow_{\max(v, v_2)} = \text{res}(\rho_2, v_2)(x)$  we get by the invariant that  $\text{Comp}(R_x, \text{res}(r_2, v_2)(x))$ . As  $\text{Comp}(S_1, \text{res}(S_2, v_2))$  we conclude by the definition of  $\text{Comp}$ .

## D Proofs from Section 6

y

**Lemma 5.** *For every tree  $t$  representing a closure  $(N, \rho)$  every sequence  $S$  of residuals of a type determined by the type of  $N$ , and every state  $q$ :*

*$A$  accepts  $t$  from  $(q, S)$  iff  $q : (\text{term}(t), \emptyset, S)$  is winning for Eve in  $G(\mathcal{A}, \text{term}(t))$*

*Proof.* The proof is by induction on the size of  $t$ .

When  $t$  has only the root, the statement is immediate.

For the induction step consider first the left to right direction. Suppose  $t$  has a root labelled by  $N$  and let  $t_1, \dots, t_k$  be its children. The existence of a run on  $t$  gives us an environment  $\rho_N$  such that the position  $q : (N, \rho_N, S)$  is winning for Eve in  $G(\mathcal{A}, M)$ . Moreover, this run gives an accepting run on  $term(t_i)$  from the state  $(q_i, S_i)$ ; for every  $S_i$  and  $q_i$  such that  $(q_i, r_i) \in \rho_N(x_i)(S_i)$ . By induction assumption Eve has a winning strategy from  $q_i : (term(t_i), \emptyset, S_i)$  in  $G(\mathcal{A}, term(t_i))$ . Observe that  $term(t_i)$  is a closed term so the environment is empty.

Our goal is to show how Eve can win from the position  $q : (term(t), \emptyset, S)$ . She should start by playing exactly the same way as from  $q : (N, \rho_N, S)$  as long as it is possible. It stops being possible when the play she reaches a leaf  $q_i : (x_i, \rho_i, S_i)$ . It means that Eve reaches at this point a position labelled  $q : (term(t_i), \emptyset, S_i)$ . Since  $q_i : (x_i, \rho_i, S_i)$  is winning we get that  $(q_i, r') \in \rho_i(x_i)(S_i)$  for some  $r'$ . Moreover, as  $x_i$  is a variable free in  $N$ , we either have  $\rho_i(x_i) = \rho_N(x_i)$  (when  $N$  is of the form  $x_i N_1 \dots N_p$ ) or  $\rho_i(x_i) = \rho_N(x_i) \downarrow_r$  for some  $r$  (using Lemma 3). So, there is  $r_i$  such that  $(q_i, r_i) \in \rho_N(x_i)(S_i)$ . But then, as we have noted above, Eve has a winning strategy from  $q : (term(t_i), \emptyset, S_i)$  is winning in  $G(\mathcal{A}, term(t_i))$ . The same strategy is winning from this position in  $G(\mathcal{A}, term(t))$ .

For the induction step for the right to left direction let us take a winning strategy for Eve from the position  $q : (term(t), \emptyset, R)$ . Let us write  $term(t)$  as  $N[\sigma]$  where  $N$  is the label of the root of  $t$ . Now rewrite the winning strategy from  $q : (term(t), \emptyset, S)$  using  $N[\sigma]$  notation. For every free variable  $x_i$  of  $N$  look at all the positions  $q_i : (x_i[\sigma], \rho_i, S_i)$ . Such a position corresponds to  $q_i : (term(t_i), \rho_i, S_i)$ . It is winning for Eve as it is reached by following a winning strategy. Since  $term(t_i)$  does not have free variables, the position  $q_i : (term(t_i), \emptyset, S_i)$  is also winning. By induction assumption, we have an accepting run on  $t_i$  from the state  $(q_i, S_i)$ . We add all such  $(q_i, r_i)$  to  $\rho_N(x_i)(S_i)$ . By definition we get that  $q : (N, \rho_N, S)$  is winning for Eve. Hence in order to accept  $t$  from the state  $(q, R)$  the automaton should choose  $\rho_N$ .