Krivine machines and higher-order schemes[☆]

Sylvain Salvati

Bordeaux University/INRIA

Igor Walukiewicz

Bordeaux University/CNRS

Abstract

We propose a new approach to analyzing higher-order recursive schemes. Many results in the literature use automata models generalizing 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.

Keywords: Higher-order model checking, Simply typed lambda-calculus, Monadic second order logic, Krivine machine, Parity games

1. Introduction

Higher-order recursive schemes were introduced by Damm [Dam82] as a respelling of λY-calculus. Since they were investigated mainly in formal language community, the tools developed were largely inspired by the treatment of pushdown-automata and context-free grammars. Subsequent research has shown that it is very useful to have an automaton model characterizing 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]. Actually, even though Krivine's paper [Kri07] has been published in 2007, Krivine defined his machine in the 80's about the time when Damm introduced higher-order schemes and proved with Goerdt that higher-order OI languages were equivalent to the languages recognized by higher-order pushdown automata [DG86]. The notion of Krivine

[☆]Supported by ANR project FREC: ANR 2010 BLAN 0202 01 FREC

Email addresses: salvati@labri.fr (Sylvain Salvati), igw@labri.fr (Igor Walukiewicz)

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 analyze schemes. To substantiate this claim 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 between order n higher-order pushdown automata with schemes of order n satisfying a syntactic constraint called *safety*. Subsequently, higher order pushdowns have been extended with the *panic* operation to handle all order 2 schemes [KNUW05, AdMO05], and with the *collapse* operation for schemes of all orders [HMOS08]. In recent years, most theoretical advances on the subject have used pushdown automata with collapse model [HMOS08, BO09, BCOS10, CS12].

The model checking problem for schemes with respect to the mu-calculus is to decide if a given formula holds at 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, BCHS12] 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 λY-calculus as the means of generating infinite trees. The Y combinator, or the fixpoint combinator, has first been considered in [CF58] and is at the core of Plotkin's PCF [Plo77]. Second, instead of using higher-order collapsible automata as an abstract machine, we use the Krivine abstract machine [Kri07]. This machine is much closer to the λ-calculus, it performs standard reductions and comes with typing. These features are hard to overestimate as they allow one 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, 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$^+$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. Broadbent [Bro12] shows that such an encoding is not possible for collapsible stacks of higher-order. This paper is a long version of the paper [SW11], that provides the detailed proofs that were sketched there.

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 prove the decidability of the local model checking problem. For this, we reduce the problem to determining a winner in a game over configurations of the Krivine machine, $\mathcal{K}(\mathcal{A}, M)$. Next, 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 Section 4 we reuse this result to obtain the proof for the global model checking problem.

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 following Huet [Hue76]: $order(0) = 1$, and $order(\alpha \to \beta) = max(1 + order(\alpha), order(\beta))$. Notice that this convention is different from the usual convention adopted in the literature on recursive schemes for which ground types have order 0.

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. As usual in recursion schemes, it is important for us that all other constants have types of order smaller or equal to 2. So we assume this for the rest of the paper. For simplicity of notation we will consider only constants of type $0 \to 0 \to 0$. It is straightforward to extend our arguments to constants of arbitrary arity; that is of types 0 or $0 \to \cdots \to 0 \to 0$.

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. We take usual conventions for dropping parentheses in λ-terms: we write $\lambda x_1 \dots x_n.M$ for $(\lambda x_1.(\dots(\lambda x_n.M)\dots))$, we write $MN_1 \dots N_n$ for $(\dots(MN_1)\dots N_n)$ and we assume that application takes precedence over λ-abstraction, that is $\lambda x_1 \dots x_n.MN_1 \dots N_p$ denotes the term

$$(\lambda x_1.(\dots(\lambda x_n.(\dots(MN_1)\dots N_p))\dots)).$$

We will also often omit type annotations. We take for granted the notion of free variables and of α-conversion and we shall write $FV(M)$ for the set of free variables occurring in a λ-term M.

We shall write $M[N_1/x_1, \dots, N_n/x_n]$ for the simultaneous capture avoiding substitution of N_1, ..., N_n respectively for the free occurrences of x_1, ..., x_n in M. When we are given a function σ that maps variables to λ-terms, we shall write $M[\sigma]$ for the result of $M[\sigma(x_1)/x_1, \dots, \sigma(x_n)/x_n]$ where $FV(M) = \{x_1, \dots, x_n\}$.

Together with the usual operational semantics of λ-calculus, that is β-contraction, we use *δ-contraction* ($\to_\delta$) giving the semantics to the fixpoint operator: $YM \to_\delta M(YM)$. Thus, the operational semantics of the λY-calculus is the $\beta\delta$-contraction whose reflexive transitive closure $\stackrel{*}{\to}_{\beta\delta}$ is called *$\beta\delta$-reduction* and whose least equivalence it generates, $=_{\beta\delta}$, is called *$\beta\delta$-conversion*. It is well-known that this semantics is confluent and enjoys subject reduction (*i.e.* the type of terms is invariant under computation).

It is usual to consider also η-contraction rule saying that: $\lambda x.Mx \to_\eta M$ when $x \notin FV(M)$. This rule is a bit complex to implement because its application requires one to check whether a variable is free in a term. A convenient way of forgetting about η-contraction, is to work with terms in *η-long form* [Hue76]. A term M is in *η-long form* when every subterm of a functional type (*i.e.* of a type of the form $\alpha \to \beta$) is either a λ-abstraction or is applied to some other term in M. It is well-known (see [Hue76]) that: (i) for every term M there is a term M' in η-long form such that M' can be η-reduced to M; (ii) the set of terms in η-long form is closed under $\beta\delta$-reduction; and (iii) two terms M_1 and M_2 are $\beta\delta\eta$-convertible iff their long forms are $\beta\delta$-convertible. Working with terms in η-long form 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 recall that a term is in *head normal form* when its form is $\lambda\vec{x}.N_0N_1 \dots N_k$ with N_0 a variable or a constant. A term M is said to be *solvable* when there is N in head normal form such that $M \stackrel{*}{\to}_{\beta\delta} N$; otherwise it is said *unsolvable*. Another related notion is that of terms in *weak head normal form*, that is terms which are either of the form $\lambda x.N$ or of the form $N_0N_1 \dots N_k$ with N_0 a variable or a constant.

In the presence of the Y combinator, simply typed terms do not in general have a normal form. But, as for the untyped lambda-calculus, a notion of infinitary normal form, called a *Böhm tree* is defined for each term. A Böhm

tree is an unranked ordered, and potentially infinite tree with nodes labeled 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, $\lambda x.\omega^0$ are labels, but $\lambda y^0.x^{0\to 0}y^0$ is not. Formally a Böhm tree of a term M is obtained as follows.

Definition 1 If $M \to^*_{\beta\delta} N = \lambda\vec{x}.N_0 N_1 \ldots N_k$ with N in head normal form then $BT(M)$ is the tree with the root labeled $\lambda\vec{x}.N_0$ and with $BT(N_1), \ldots, BT(N_k)$ as the sequence of trees starting from successors of the root. If M is unsolvable then $BT(M) = \omega^\alpha$ where α is the type of M.

Remark: If M is a closed term 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. Indeed we have two cases. If M is unsolvable then $BT(M)$ consists just of a root labeled ω^0. Otherwise M can be reduced to a head normal form N. Because M is of type 0, N cannot start with λ-abstraction. As M is closed, N must be of the form aM_1M_2 where a is a constant. So M_1, M_2 are closed terms of type 0. Repeating the argument on M_1 and M_2 shows that $BT(M)$ is indeed a binary tree with finite branches ending in ω^0.

Recursive schemes and λY-calculus. In many works on model checking higher-order systems, these systems are presented as recursive schemes. We here choose an equivalent presentation in terms of λY-calculus that we find more convenient to work with. Nevertheless, we here give a slight introduction to the notion of recursive schemes.

A recursive scheme is a set of equations defining a λY-term by mutual recursion. Formally, a recursive scheme is a function $\mathcal{R}$ assigning to every variable F^α from a finite set $\mathcal{N}$ a term of type α 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 $\mathcal{R}$.

To transform λY-term into a recursive scheme, it is enough to name every subterm with a new variable, and then write a straightforward assignment function $\mathcal{R}$. To transform a recursive scheme into a λY-term one can solve the system of equations given by $\mathcal{R}$ using the fixpoint combinator Y. With some care these transformations are inverse to each other and do not increase the order of a term unnecessary. The details of these translations can be found in [SW12].

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 a closed λ-term: it is obtained by substituting for every free variable x of M the lambda term denoted by the closure $\rho(x)$. For a closure C we write $term(C)$ for the term that it denotes; when $C = (N, \rho)$, we have $term(C) = N[term(\rho(x_1))/x_1, \ldots, term(\rho(x_n))/x_n]$.

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. A configuration (M, ρ, S) represents the term M in the environment ρ determining the values of its free variables, applied to terms represented by the closures in S. The first rule simulates β-contraction, it binds in the environment the variable introduced by the λ-abstraction to the top-most element of the stack, thus the λ-abstraction is evaluated by assigning the value of the argument to the correct variable. The second rule, simulates δ-contraction in the obvious way. The third rule decomposes the evaluation of an application into the evaluation of the function in the current environment applied to the closure made out of its argument and the current environment. Finally, the last rule finds the value of a variable in the environment and evaluates it, restoring the environment in which it was created, with the current stack for its arguments.

We will be only interested in configurations accessible from $(M_0, \emptyset, \varepsilon)$ for some closed term M_0 of type 0 in η-long form. Recall that $\emptyset$ stands for the empty environment and ε for the empty stack. Every such configuration (M, ρ, S) is called a *configuration of type 0* and enjoys very strong typing invariants. The environment ρ associates to a variable x^α a closure (N, ρ') such that N has type α; we will say that the closure is of type α too. If M 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. Another property of the configurations that are reachable from $(M_0, \emptyset, \varepsilon)$ is that the terms involved in the configuration (M, ρ, S), that is M, the terms that are used to build the environment ρ and the closures in S, are all *subterms* of M_0. One should be careful with a definition of a subterm though. Since we have a fixpoint operator we consider that $N(YN)$ (but not $N(N(YN))$) is a subterm of YN . Of course even with this twist, the number of subterms of a term remains finite.

For aesthetic reasons we prefer to stop the Krivine machine in configurations of the form $(bM_0M_1, \rho, \varepsilon)$, 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 $(b(M_0, M_1), \rho, \varepsilon)$ 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.) Now the Krivine machine may be used to compute the Böhm tree of a term M. Indeed, 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 $(b(M_0, M_1), \rho, \varepsilon)$. Because we work with terms in η-long form, and the constants occurring in M must, by definition of η-long forms, be applied to two arguments, as the Krivine machine decomposes the terms in a top-down manner, the first time a term starting with a constant b appears as the main term of the configuration, it must be applied to its two arguments. Thus the use of η-long forms justifies the shape of the configurations in which the Krivine machine stops. At that point we create a node labeled b and start reducing both (M_0, ρ, ε) and (M_1, ρ, ε). This process gives at the end a tree labeled with constants.

Definition 2 Given a configuration (M, ρ, S) of type 0, if this configuration reduces to $(b(M_0, M_1), \rho, \varepsilon)$ then we let $KT(M, \rho, S)$ be the tree whose root is labeled with b and has as left child the tree $KT(M_1, \rho, \varepsilon)$ and as right child the tree $KT(M_2, \rho, \varepsilon)$. Otherwise $KT(M, \rho, S) = \omega^0$.

For a closed term of type 0, M, we write $KT(M)$ for $KT(M, \emptyset, \varepsilon)$.

The Krivine machine implements a particular reduction strategy of the λ-calculus called weak-head reduction. This strategy amounts to systematically reducing the top-most redex of a term. If there is a way of reducing a term M to a term N, then the *standardization Theorem* [CF58] shows that the reduction from M to N can be done by reducing top-most redices first and then other redices below. Therefore, when a term has a head normal form, it can be put in head-normal form by reducing the top-most redices only. This implies that the reduction strategy of the Krivine machine always computes, when it has one, the head-normal form of a closed term of type 0. Another consequence is that $KT(M)$ is no other than $BT(M)$. Here we state this result for the closed terms of type 0, because, for simplifying the exposition, we have defined $BT(M)$ and $KT(M)$ only for such terms. As a digression let us note that this equality holds for all closed terms in η-long form. For terms not in η-long form the Krivine machine would actually compute a slight variant of Böhm trees called Lévy-Longo trees which are to weak head normal forms what Böhm trees are to head normal forms. A consequence of the correctness of the Krivine machine [Kri07] is the following theorem.

Theorem 3 *Given a closed term M of type 0, we have $BT(M) = KT(M)$.*

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:

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

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

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

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:

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

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 a possibly infinite binary tree (cf. Remark on page 5) we can use standard non-deterministic parity automata to describe its properties. Such an automaton has the form

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

where Q is a finite set of states, q^0 is the initial state, δ is the transition function, and rk 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 another 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 rk*; 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 assumption is frequently made in this context. Handling ω^0 would not be difficult but would require to add one more case in all the constructions. Another solution is to convert a term to a term not generating ω^0. Haddad [Had12] proposes such a transformation at the level of schemes. Another such transformation would consist in transforming each rule $Fx_1 \ldots x_n \to N$ of a scheme into a rule $Fx_1 \ldots x_n \to e\, N$ where e is a fresh unary constant. Then the scheme becomes always productive and the MSOL theory of its generated tree can be translated into the MSOL theory of the tree generated by the new scheme. Those transformations can also be performed on λY-terms.

Definition 4 *The (local) model-checking problem for λY-calculus* is to decide if for a given parity automaton $\mathcal{A}$ and a λY-term M, the automaton accepts the tree $BT(M)$.

3. Decidability of the model checking problem

This section presents the proof of the decidability of the model checking problem for λY-calculus.

Theorem 5 (Ong [Ong06]) *Given M a closed λY-term of type 0, the modal mu-calculus model checking problem of $BT(M)$ is decidable, thus $BT(M)$ has a decidable MSOL theory.*

As over infinite trees, modal mu-calculus is equivalent to MSOL, for the proof we will work with parity automata instead of formulas. For every MSOL formula φ, there is a parity automaton $\mathcal{A}$ accepting precisely the trees satisfying φ.

The first step is to construct, given a parity automaton $\mathcal{A}$ and a closed λY-term M of type 0, an infinite parity game $\mathcal{K}(\mathcal{A}, M)$ such that Eve has a winning strategy in $\mathcal{K}(\mathcal{A}, M)$ iff $\mathcal{A}$ accepts $BT(M)$. The game $\mathcal{K}(\mathcal{A}, M)$ is constructed from the computation tree of the Krivine machine constructing $KT(M)$, and from Theorem 3 we know that $KT(M)$ is isomorphic to $BT(M)$. So $\mathcal{K}(\mathcal{A}, M)$ is a potentially infinite game, and we want to decide who has a winning strategy in it. The core of the proof consists in the effective construction of a *finite* parity game $\mathcal{G}(\mathcal{A}, M)$ such that Eve has a winning strategy in that game iff she has a winning strategy in the game $\mathcal{K}(\mathcal{A}, M)$. This yields the decidability of the model checking problem.

Krivine machine plays a key role in the construction of $\mathcal{G}(\mathcal{A}, M)$. The labels of positions in $\mathcal{K}(\mathcal{A}, M)$ contain configurations of the Krivine machine. The positions of $\mathcal{G}(\mathcal{A}, M)$ are approximations of the labels in $\mathcal{K}(\mathcal{A}, M)$ with respect to property being checked by the automaton $\mathcal{A}$. These bounded approximations are made thanks to a notion of a residual. The proof of the equivalence of the two games amounts to translating winning strategies for Eve or Adam in $\mathcal{K}(\mathcal{A}, M)$ into respective winning strategies in $\mathcal{G}(\mathcal{A}, M)$. In this translation the invariants are naturally expressed in terms of configurations of the Krivine machine.

3.1. Game $\mathcal{K}(\mathcal{A}, M)$

In this subsection we define the game $\mathcal{K}(\mathcal{A}, M)$. The game will be based on the tree $RT(\mathcal{A}, M)$ of the runs of the automaton $\mathcal{A}$ on the tree of configurations of the Krivine Machine computing $BT(M)$. The actual runs of $\mathcal{A}$ on $BT(M)$ can easily be read from $RT(\mathcal{A}, M)$.

Definition 6 For a given closed term M of type 0 , and a parity automaton $\mathcal{A}$ we define the tree of all runs $RT(\mathcal{A}, M)$ of $\mathcal{A}$ on $BT(M)$:

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

5. A node $q : (YN, \rho, S)$ has a unique successor $q : (N(YN), \rho, S)$.
6. A node v labeled $q : (NK, \rho, S)$ has a unique successor that is labeled $q : (N, \rho, (v, K, \rho)S)$. We say that here a v-closure is *created*.
7. A node v labeled $q : (x, \rho, S)$, with $\rho(x) = (v', N, \rho')$, has a unique successor labeled $q : (N, \rho', S)$. We say that the node v *uses* a 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 (rule 7), the stored node does not influence the result. The stored node allows us to detect what is exactly the closure that we are using. This will be important in the proof.

Notice also that the rules 2, 3, and 4 rely on the typing properties of the configurations of the Krivine machine we discussed earlier (cf. pages 5–8). Indeed, when the machine reaches a configuration of the form $(a(N_1, N_2), \rho, \varepsilon)$ then, by our convention on the types of constants, a is of type $0 \to 0 \to 0$. In consequence, so as to have a term of type 0, it must be applied to two arguments of type 0, while the stack is empty. Also from typing invariant we get that, when the machine is in a configuration like $(\lambda x.N, \rho, S)$, S cannot be the empty stack.

Definition 7 We use the tree $RT(\mathcal{A}, M)$ to define a game between two players: Eve chooses a successor in nodes of the form $q : (a(N_0, N_1), \rho, S)$, and Adam in nodes $(q_0, q_1) : (a(N_0, N_1), \rho, S)$. We set the parity rank of nodes labeled $q : (a(N_0, N_1), \rho, S)$ to $rk(q)$, and the parity ranks of all the other nodes to 1, and the minimal parity rank that we use is 1. We use the 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 and of Theorem 3.

Proposition 8 For every parity automaton $\mathcal{A}$ and closed term M of type 0. 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 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 without reaching a head normal form. Such a sequence results in a node labeled ω in a 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.

By Proposition 8 deciding whether $BT(M)$ is accepted by $\mathcal{A}$ is reduced to deciding who has a winning strategy from the root of $\mathcal{K}(\mathcal{A}, M)$. Using $\mathcal{K}(\mathcal{A}, M)$ we will construct a finite game $G(\mathcal{A}, M)$, and show that the winner in the two games is the same.

3.2. Game $G(\mathcal{A}, M)$

The game $\mathcal{K}(\mathcal{A}, M)$ may have infinitely many positions because there may be infinitely many closures that are created. We reduce this game to $G(\mathcal{A}, M)$ where we remove this source of infiniteness.

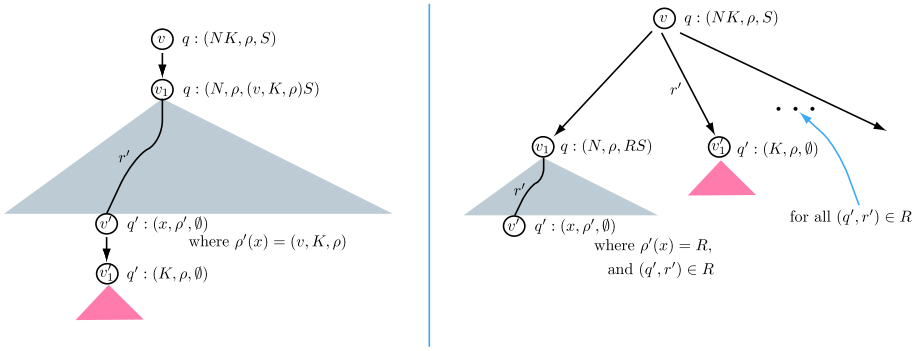

Figure 1: Game $\mathcal{K}(\mathcal{A}, M)$ on the left, and $G(\mathcal{A}, M)$ on the right.

The idea of the reduction is to eliminate stacks and environments using alternation. Consider the situation in Figure 1. On the top left we have a position v in the game $\mathcal{K}(\mathcal{A}, M)$ where the application rule is used. This means that the new closure (v, K, ρ) is put on the stack of the Krivine machine (node v_1). In some descendant v' of v_1 the closure may be used. In other words, the machine gets to the variable x whose value is the closure in question. Let us consider the simplest case when K is of type 0. Due to the typing invariants on configurations of the Krivine machine, we know that the stack is empty in v'. So the configuration in the successor v'_1 of v' is constructed just from the closure. This observation allows to shortcut the path from v to v'_1. This is what we do in the game $G(\mathcal{A}, M)$.

The right part of Figure 1 represents the result of taking these shortcuts. In a set R, that we call residual of the closure (v, K, ρ), we have collected all states q' which appear when the closure (v, K, ρ) is used: as in the node v'_1. For every such state we add directly a successor of v labeled with the corresponding configuration. So the edge from v to v'_1 in the right picture simulates the path from v to v'_1 in the left picture. Now the question is where we get R from. We actually just guess it and check if it is big enough. This is the task of the leftmost transition in the right picture. The gray triangle is the same as in the original game. But this time instead of a closure we have put R on the stack. When we get to v' we just check that the state in v' is in R. This check guarantees that we have put all uses of the closure into R.

The successive level of complication comes from the fact that $\mathcal{K}(\mathcal{A}, M)$ is a parity game and not a reachability game. This complication is not just cosmetic: the model-checking problem for reachability games can be solved using much lighter methods [Aeh07, Kob09, SW13]. In order to deal with parity conditions we need not only to remember the state in which the closure is used, but also the biggest rank on the path from the creation of the closure to its use. This is symbolized by r' in the left part of the figure. We use the same r' as the rank of the edge in the reduced game.

Till now we have assumed in our discussion that K is of type 0, but in

general we need to deal with terms K of types of any order. The difference is that if K is not of type 0 then the configuration in v' on the left will be of the form $q' : (x, \rho', S')$ for some stack S' whose contents is not related with the contents of S. The important thing though is that the type of elements in S' is determined by the type of x, that is the same as the type of K. Observe that the typing invariant of the Krivine machine tells us that the orders of types of closures on the stack are always strictly smaller than that of K. So by induction on types we can assume that S is composed of residuals and not of closures. Since there are finitely many residuals of a given type, the residual for K will be now a function from sequences of residuals representing possible stacks S to a set of states with ranks as in the case when K had type 0.

Residuals. After these intuitive explanations we will proceed to define residuals, the lifting operation on residuals, and finally the game $G(\mathcal{A}, M)$. The lifting operation on residuals will permit us to deal with all the book-keeping required by the parity condition.

Definition 9 (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, 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.

We need one more operation before defining the game. Indeed, when a set of residuals is guessed in $G(\mathcal{A}, M)$, it is, as mentioned above, the role of the left transition on the right of Figure 1 to check that the guess covers all the possibilities. In the particular case where the term K is of type 0, checking that the residual is correct makes it necessary to verify that the biggest ranks guessed on the paths from the node where the closure is created to the nodes where it is used are correct. The role of the lifting operation we introduce here is to implement this verification. Of course this operation is defined for residuals of any orders.

Definition 10 A lifting of a residual $R : D_{\tau_1} \to \cdots \to D_{\tau_k} \to D_0$ by a rank r is a residual $R\lfloor_r$ of the same type as R satisfying for every sequence of arguments S:

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

If ρ is an environment assigning residuals to variables then $\rho\lfloor_r$ is an environment such that for every x: $(\rho\lfloor_r)(x) = \rho(x)\lfloor_r$.

Recall that $D_0 = \mathcal{P}(Q \times [d])$ so what $R\lfloor_r$ does is to modify the set of pairs $R(S)$ which is the value of R on the sequence of residuals S of appropriate type. The operation leaves unchanged all pairs (q_1, r_1) with $r_1 > r$. For every pair

(q_1, r_1) with $r_1 = r$ it adds pairs (q_1, r_2) for all $r_2 \leq r$. All pairs (q_1, r_1) of $R(S)$ with $r_1 < r$ do not contribute to the result.

Example Let's take the residual $R = \{(q_1, 1); (q_2, 2); (q_3, 3)\}$ of type 0. We have that

$$\begin{aligned} R\lfloor_1 =& \{(q_1, 0); (q_1, 1); (q_2, 2); (q_3, 3)\}, \\ R\lfloor_2 =& \{(q_2, 0); (q_2, 1); (q_2, 2); (q_3, 3)\}, \\ R\lfloor_3 =& \{(q_3, 0); (q_3, 1); (q_3, 2); (q_3, 3)\}, \text{ and} \\ R\lfloor_4 =& \emptyset. \end{aligned}$$

If we take a residual R of type $0 \to 0$ that maps $\{(q_1, 1)\}$ to $\{(q_2, 2); (q_3, 3)\}$ and $\{(q_2, 1)\}$ to $\{(q_1, 1); (q_3, 1)\}$, and all other residuals to $\emptyset$ then $R\lfloor_2$ maps $\{(q_1, 1)\}$ to $\{(q_2, 0); (q_2, 1); (q_2, 2); (q_3, 3)\}$ and all other residuals to $\emptyset$.

As the lifting operation will be performed all along a path in $BT(M)$, we need the following technical lemma to handle multiple applications of this operation.

Lemma 11 For every residual R and ranks r_1, r_2: $(R\lfloor_{r_1})\lfloor_{r_2} = R\lfloor_{\max(r_1, r_2)}$.

Proof
Let's assume that R is a residual of type $\alpha = \alpha_1 \to \cdots \to \alpha_n \to 0$ and take $R_1, \ldots, R_n$ residuals of $D_{\alpha_1}, \ldots, D_{\alpha_n}$. If (q, r) is in $(R\lfloor_{r_1})\lfloor_{r_2}(R_1, \ldots, R_n)$, then either $r \geq r_2$ and (q, r) is in $R\lfloor_{r_1}(R_1, \ldots, R_n)$, or $r < r_2$ and (q, r_2) is in $R\lfloor_{r_1}(R_1, \ldots, R_n)$. Each case can then be split in two.

In the first case, (q, r) is in $R\lfloor_{r_1}(R_1, \ldots, R_n)$ either since $r \geq r_1$ and (q, r) is in $R(R_1, \ldots, R_n)$ or since $r < r_1$ and (q, r_1) is in $R(R_1, \ldots, R_n)$. If $r \geq r_1$ and (q, r) is in $R(R_1, \ldots, R_n)$, then, because $r \geq r_2$, we have $r \geq \max(r_1, r_2)$ and thus, (q, r) is in $R\lfloor_{\max(r_1, r_2)}(R_1, \ldots, R_n)$. If $r < r_1$, then because $r \geq r_2$, we have $\max(r_1, r_2) = r_1$, then because (q, r) is in $R\lfloor_{r_1}(R_1, \ldots, R_n)$, we also have that (q, r) in $R\lfloor_{\max(r_1, r_2)}(R_1, \ldots, R_n)$.

In the second case, (q, r_2) is in $R\lfloor_{r_1}(R_1, \ldots, R_n)$ either since $r_2 \geq r_1$ and (q, r_2) is in $R(R_1, \ldots, R_n)$ or since $r_2 < r_1$ and (q, r_1) is in $R(R_1, \ldots, R_n)$. If $r_2 \geq r_1$, then $r_2 = \max(r_1, r_2)$, and since (q, r_2) is in is in $R(R_1, \ldots, R_n)$, because $r < r_2$, we obtain that (q, r) is in $R\lfloor_{\max(r_1, r_2)}(R_1, \ldots, R_n)$. If $r_2 < r_1$, then $r_1 = \max(r_1, r_2)$ and as, (q, r_1) is in $R(R_1, \ldots, R_n)$, we obtain that (q, r) is in $R\lfloor_{\max(r_1, r_2)}(R_1, \ldots, R_n)$.

We have thus showed that

$$(R\lfloor_{r_1})\lfloor_{r_2}(R_1, \ldots, R_n) \subseteq R\lfloor_{\max(r_1, r_2)}(R_1, \ldots, R_n),$$

we now turn to the converse inclusion. If (q, r) is in $R\lfloor_{\max(r_1, r_2)}(R_1, \ldots, R_n)$, then it is either because $r \geq \max(r_1, r_2)$ and (q, r) is in $R(R_1, \ldots, R_n)$ or because $r < \max(r_1, r_2)$ and $(q, \max(r_1, r_2))$ is in $R(R_1, \ldots, R_n)$. In both cases, the definitions immediately lead to the fact that (q, r) is also in $(R\lfloor_{r_1})\lfloor_{r_2}(R_1, \ldots, R_n)$. In the first case, since $r \geq \max(r_1, r_2)$ and (q, r) is in $R(R_1, \ldots, R_n)$, we have that (q, r) is in $R\lfloor_{r_1}(R_1, \ldots, R_n)$ which finally entails that (q, r) is in $(R\lfloor_{r_1})\lfloor_{r_2}(R_1, \ldots, R_n)$. In the second case, we have that $(q, \max(r_1, r_2))$ is

in $R(R_1, \ldots, R_n)$ which implies that for every $r' \leq \max(r_1, r_2)$, (q, r') is in $(R\lfloor_{r_1})\lfloor_{r_2}(R_1, \ldots, R_n)$ and, thus, (q, r) is in $(R\lfloor_{r_1})\lfloor_{r_2}(R_1, \ldots, R_n)$. □

Definition of $G(\mathcal{A}, M)$. We have all the necessary ingredients to define the game $G(\mathcal{A}, M)$. A position of the game 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 subterm of M; ρ is a function 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 have to agree with the types of variables/arguments they are assigned to. Notice that we use the same letter ρ to denote an environment as well as an assignment of residuals. Similarly for S. It will be always clear from the context what object is denoted by these letters.

Most of the rules of $G(\mathcal{A}, M)$ are just reformulations of the rules in $\mathcal{K}(\mathcal{A}, M)$:

- $q : (\lambda x.N, \rho, R \cdot S) \to q : (N, \rho[x \mapsto R], S)$
- $q : (a(N_0, N_1), \rho, \varepsilon) \to (q_0, q_1) : (a(N_0, N_1), \rho, \varepsilon)$ for $(q_0, q_1) \in \delta(q, a)$
- $(q_0, q_1) : (a(N_0, N_1), \rho, \varepsilon) \to q_i : (N_i, \rho\lfloor_{rk(q_i)}, \varepsilon)$ for $i = 0, 1$
- $q : (YN, \rho, S) \to q : (N(YN), \rho, S)$.

Observe the use of $\lfloor_{rk(q_i)}$ in the third rule. Residuals express the constraints on contexts where we perform variable lookup. The lifting operation $\lfloor_{rk(q_i)}$ is used to modify these constraints taking into account that we have seen a state of rank $rk(q_i)$.

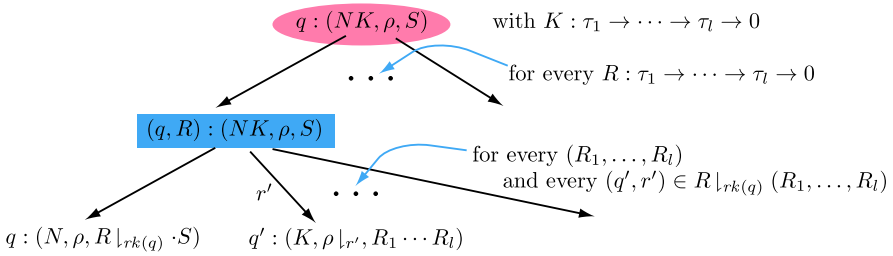

Figure 2: Dealing with application in $G(\mathcal{A}, M)$.

We now proceed to the rule for application (cf. Figure 2). 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 $R : D_{\tau_1} \to \cdots \to D_{\tau_l} \to D_0$.

From this position we have transitions

- $(q, R) : (NK, \rho, S) \to \ q : (N, \rho, R\lfloor_{rk(q)} \cdot S)$

- $(q, R) : (NK, \rho, S) \to\ q' : (K, \rho\lfloor_{r'}, R_1 \cdots R_l)$
  for every $R_1 \in D_{\tau_1}, \ldots, R_l \in D_{\tau_l}$ and $(q', r') \in R\lfloor_{rk(q)}(R_1, \ldots, R_l)$.

The operation $R\lfloor_{rk(q)}$ is needed to "normalize" the residual, so that it satisfies the invariant described below.

Since we are defining a game, we need to say who makes a choice at which vertices. Eve chooses a successor from vertices of the form $q : (NK, \rho, S)$, and $q : (a(N_0, N_1), \rho, \varepsilon)$. 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 $(q, R) : (NK, \rho, S)$. So he decides whether to accept (by choosing a transition of the first type) or to contest the residual proposed by Eve; cf. Figure 2.

Observe that we do not have a rule for nodes with a term being a variable. This means that we need to say who is the winner in variable nodes. Eve wins in a position:

- $q : (x, \rho, S)$ when $(q, rk(q)) \in \rho(x)(S)$.

Recall that $\rho(x)$ is a residual, call it R_x, and $S = R_1 \cdots R_k$ is a sequence of residuals; so $\rho(x)(S)$ is $R_x(R_1 \cdots R_k)$ which is an element of $\mathcal{P}(Q \times [d])$.

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 1 but for two cases

- transitions of the form $(q, R) : (NK, \rho, S) \to\ q' : (K, \rho\lfloor_{r'}, R_1 \cdots R_k)$ where $(q', r') \in R\lfloor_{rk(q)}(R_1, \ldots, R_k)$ have rank r';
- transitions of the form $(q_0, q_1) : (a(N_0, N_1), \rho, \varepsilon) \to q_i : (N_i, \rho\lfloor_{rk(q_i)}, \varepsilon)$ have rank $rk(q_i)$.

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.

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

We now prove the central property relating $G(\mathcal{A}, M)$ and $\mathcal{K}(\mathcal{A}, M)$.

Proposition 12 For every parity automaton $\mathcal{A}$ and every closed term M of type 0, Eve wins in $\mathcal{K}(\mathcal{A}, M)$ iff Eve wins in $G(\mathcal{A}, M)$.

The proof of this proposition proceeds as follows. For the direction from left to right we take a winning strategy for Eve in $\mathcal{K}(\mathcal{A}, M)$, and with respect to this strategy we define residuals for every closure. Then we show how Eve can win in $G(\mathcal{A}, M)$ using these residuals: her winning strategy in $G(\mathcal{A}, M)$ will simulate the one in $\mathcal{K}(\mathcal{A}, M)$. For the other direction we will calculate residuals with respect to Adam's winning strategy in $\mathcal{K}(\mathcal{A}, M)$ and use them to define Adam's winning strategy in $G(\mathcal{A}, M)$. As parity games are determined, we obtain Proposition 12.

3.3.1. Residuals in $\mathcal{K}(\mathcal{A}, M)$

We here introduce the key notion of the proof, the notion of a *residual of a node*. Given a subtree $\mathcal{T}$ of $\mathcal{K}(\mathcal{A}, M)$, i.e. a tree obtained by pruning $\mathcal{K}(\mathcal{A}, M)$, we calculate the residuals $R_{\mathcal{T}}(v)$ and $res_{\mathcal{T}}(v, v')$ for some nodes and pairs of nodes of $\mathcal{T}$. In particular, $\mathcal{T}$ may be taken as being a strategy of Eve or a strategy of Adam. When $\mathcal{T}$ is clear from the context we will simply write $R(v)$ and $res(v, v')$.

Recall that a node v in $\mathcal{K}(\mathcal{A}, M)$ is an application node when its label is of the form $q : (NK, \rho, S)$. We will assign a residual $R(v)$ to every application node v. Thanks to typing, this can be done by induction on the order of types. We also define a variation of this notion: a residual $R(v)$ seen from a node v', denoted $res(v, v')$. The role of residuals in the proof has been intuitively explained on pages 10-12.

Before giving a formal definition we will describe the assignment of residuals to nodes in concrete terms. We will need one simple abbreviation. If v is an ancestor of v' in $\mathcal{T}$ 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 $\mathcal{T}$ (cf. Figure 1). 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 $R(v)$. We will do this by induction on the type of K.

First, suppose that the closure, or equivalently the term K, is of type 0. The residual $R(v)$ is a subset of $Q \times [d]$ obtained as follows:

> We put $(q', \max(v, v')) \in R(v)$ when there is v' in $\mathcal{T}$ labeled with $q' : (x, \rho', \varepsilon)$ such that $\rho'(x) = (v, K, \rho)$.

For the induction step, suppose that K is of type $\tau_1 \to \cdots \to \tau_k \to 0$ and that we have already calculated residuals for all closures of types $\tau_1, \ldots, \tau_k$. Suppose that we have a closure (v, K, ρ) created at a node v. This time $R(v) : D_{\tau_1} \to \cdots \to D_{\tau_k} \to \mathcal{P}(Q \times [d])$. Consider 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

$$(q', \max(v, v')) \in R(v)(R(v_1)|_{\max(v_1, v')}, \ldots, R(v_k)|_{\max(v_k, v')}) \ .$$

We now give a formal definition of $R(v)$. By structural induction on types it is easy to see that such an assignment of residuals exists and is unique for $\mathcal{T}$.

Definition 13 ($R(v)$ and $res(v, v_1)$) Given $\mathcal{T}$ a subtree of $\mathcal{K}(\mathcal{A}, M)$, we define a residual $R(v)$ for every application node v of $\mathcal{T}$.

For more clarity we will use $res(v, v_1)$ for $R(v)|_{\max(v, v_1)}$. For a closure (v, K, ρ) we define $res((v, K, \rho), v') = res(v, v')$. We then extend this operation to stacks: $res(S, v')$ is S where $res(\cdot, v')$ is applied to every element of the stack; and to environments: $res(\rho, v')(x) = res(\rho(x), v')$.

Let v be a node of $\mathcal{T}$ labeled by $q : (NK, \rho, S)$ with K of type $\tau_1 \to \cdots \to \tau_k \to 0$. The residual $R(v)$ is a function $D_{\tau_1} \to \cdots \to D_{\tau_k} \to D_0$ such that for every sequence of residuals $\vec{R}$ of appropriate types the set $R(v)(\vec{R})$ contains:

> $(q', \max(v, v'))$ for every node v' of $\mathcal{T}$ with the label of the form $q' : (x, \rho', S')$ for some x, ρ', S' such that $\rho'(x) = (v, K, \rho)$, and $res(S', v') = \vec{R}$.

3.3.2. Transferring Eve's strategy in $\mathcal{K}(\mathcal{A}, M)$ to $G(\mathcal{A}, M)$

Let us assume that Eve has a winning strategy σ on $\mathcal{K}(\mathcal{A}, M)$. This strategy defines the subtree $\mathcal{K}_\sigma$ of $\mathcal{K}(\mathcal{A}, M)$ of all the plays respecting the strategy σ. Let us also assume that we have computed the residuals for $\mathcal{K}_\sigma$ as in Definition 13. We will use those residuals to define a winning strategy for Eve in $G(\mathcal{A}, M)$.

The invariant. We will use positions in the game $\mathcal{K}(\mathcal{A}, M)$ and the strategy σ as hints. The strategy in $G(\mathcal{A}, M)$ will take a pair of positions (v_1, v_2) with v_1 in $G(\mathcal{A}, M)$ and a v_2 in $\mathcal{K}(\mathcal{A}, M)$. 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:

- v_1 is labeled by $q : (N, \rho_1, S_1)$, v_2 is labeled by $q : (N, \rho_2, S_2)$, $\rho_1 = res(\rho_2, v_2)$ and $S_1 = res(S_2, v_2)$.

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

The two cases where Eve needs to decide which successor to choose are the nodes with a constant or with an application. For all other transitions, the invariant is trivially preserved.

A node with a constant is of the form $q : (a(N_0, N_1), \rho_1, \varepsilon)$. Eve should then simply take from v_1 the same transition of the automaton as taken from v_2. So she advances to a node labeled $(q_0, q_1) : (a(N_0, N_1), \rho_1, \varepsilon)$, and at the same time Eve goes to $(q_0, q_1) : (a(N_0, N_1), \rho_2, \varepsilon)$ in $\mathcal{K}(\mathcal{A}, M)$. Next Adam can choose a successor, so he can go either to $q_0 : (N_0, \rho_1, \varepsilon)$ or $q_1 : (N_1, \rho_1, \varepsilon)$. This move can be matched by going to $q_0 : (N_0, \rho_2, \varepsilon)$ or $q_1 : (N_1, \rho_2, \varepsilon)$ in $\mathcal{K}(\mathcal{A}, M)$. This shows that no matter what Adam's next move is, the new pair of positions in the two games will satisfy the invariant.

The strategy and its analysis in the case of application nodes is more complicated. Figure 2 represents the part of the game $G(\mathcal{A}, M)$ from such a node. 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 in $\mathcal{K}(\mathcal{A}, M)$ a position v_2 labeled by $q : (NK, \rho_2, S_2)$, where $\rho_1 = res(\rho_2, v_2)$ and $S_1 = res(S_2, v_2)$. To satisfy the invariant, the strategy in $G(\mathcal{A}, M)$ needs to choose $R(v_2)$, that is the residual assigned to v_2. This means that from v_1 the play proceeds to the node v_1' labeled $(q, R(v_2)) : (NK, \rho_1, S_1)$.

From this node Adam can choose either

$$q : (N, \rho_1, (R(v_2)\lfloor_{rk(q)}) \cdot S_1), \quad \text{or} \tag{2}$$

$$q' : (K, \rho_1\lfloor_{r'}, R_1 \dots R_l) \quad \text{where } (q', r') \in R(v_2)\lfloor_{rk(q)}(R_1, \dots, R_l). \tag{3}$$

Suppose Adam chooses v_1'' whose label is as in (2). By definition $R(v_2)\lfloor_{rk(q)} = res(v_2, v_2)$. Hence the stack $(R(v_2)\lfloor_{rk(q)}) \cdot S_1$ is just $res((v_2, K, \rho_2)S_2, v_2)$. The unique successor v_2' of v_2 is labeled 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 where Adam chooses for v_1'' a node of the form (3) for some q', r' and $R_1 \dots R_l$ (see Figure 3). Looking at the definition of $R(v_2)$, Definition 13, we have that, in $\mathcal{K}_\sigma$, the node v_2 has a descendant v_2' labeled $q' : (x, \rho_2', S_2')$ with $\rho_2'(x) = (v_2, K, \rho_2)$, $res(S_2', v_2') = R_1 \dots R_l$ and moreover $r' = \max(v_2, v_2')$. The successor v_2'' of v_2' is labeled by $q' : (K, \rho_2, S_2')$. We can take it as a companion for v_1'' since $\rho_1\lfloor_{r'} = res(\rho_2, v_2)\lfloor_{\max(v_2, v_2'')} = res(\rho_2, v_2'')$ by Lemma 11. Hence the pair (v_1'', v_2'') satisfies the invariant.

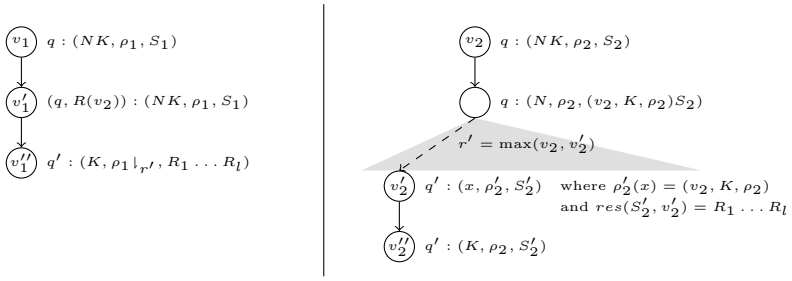

Figure 3: Adam chooses a node of the form (3)

The strategy is winning. We need to show that the strategy defined above is winning. Consider a sequence of nodes $(v_1^1, v_2^1), (v_1^2, v_2^2), \dots$ obtained when playing according to 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 play ends in a pair (v_1, v_2) where v_1 is a variable node. Such a node is labeled by $q : (x, \rho_1, S_1)$. To show that Eve wins here we need to prove that

$$(q, rk(q)) \in R_x(S_1) \text{ where } R_x = \rho_1(x).$$

By the invariant we have that the companion node v_2 is labeled by $q : (x, \rho_2, S_2)$ and $\rho_1 = res(\rho_2, v_2)$, $S_1 = res(S_2, v_2)$. Suppose that $\rho_2(x) = (v, N, \rho)$. We have $R_x = R(v)\lfloor_{\max(v, v_2)}$, since $\rho_1 = res(\rho_2, v_2)$. By definition of $R(v)$ we get

$$(q, \max(v, v_2)) \in R(v)(res(S_2, v_2)).$$

Then from the definition of the $\lfloor_{\max(v,v_2)}$ operation:

$$(q, \max(v, v_2)) \in R(v)(res(S_2, v_2))\lfloor_{\max(v,v_2)}.$$

Which implies that $(q, rk(q)) \in R(v)(res(S_2, v_2))\lfloor_{\max(v,v_2)}$ because $rk(q) \leq \max(v, v_2)$. But then $R(v)(res(S_2, v_2))\lfloor_{\max(v,v_2)} = R_x(S_1)$, and we are done.

This completes the proof of left-to-right implication of Proposition 12.

3.3.3. Transferring Adam's strategy from $\mathcal{K}(\mathcal{A}, M)$ to $G(\mathcal{A}, M)$

We will show how to get a winning strategy for Adam in $G(\mathcal{A}, M)$ from his winning strategy in $\mathcal{K}(\mathcal{A}, M)$. Once again we will use residuals. Let us fix a winning strategy θ of Adam in $\mathcal{K}(\mathcal{A}, M)$. Consider the tree $\mathcal{K}_\theta$ of plays respecting this strategy. This is a subtree of $\mathcal{K}(\mathcal{A}, M)$. Consider the assignment of residuals to application nodes in $\mathcal{K}_\theta$ as in Definition 13. We will define a strategy in $G(\mathcal{A}, M)$ that will preserve the invariant described below.

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

- For $R_1, R_2 \in D_0$ we define $Comp(R_1, R_2)$ to be true if $R_1 \cap R_2 = \emptyset$.
- For $R_1, R_2 \in D_\tau$ where $\tau = \tau_1 \to \cdots \to \tau_k \to 0$ we put $Comp(R_1, R_2)$ if for all sequences $(R_{1,1}, \ldots, R_{1,k}), (R_{2,1}, \ldots, R_{2,k}) \in D_{\tau_1} \times \cdots \times D_{\tau_k}$ such that $Comp(R_{1,i}, R_{2,i})$ for all $i = 1, \ldots, k$ we get $R_1(R_{1,1}, \ldots, R_{1,k}) \cap R_2(R_{2,1}, \ldots, R_{2,k}) = \emptyset$.

Remark: the *Comp* predicate is a logical relation (see [AC98]), but we have preferred to formulate the definition in a form that will be more useful for proofs.

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

It is important to observe that *Comp* behaves well with respect to the $\lfloor_r$ operation.

Lemma 14 If $Comp(R_1, R_2)$ then also $Comp(R_1\lfloor_r, R_2\lfloor_r)$ for every rank r.

Proof

Take two sequences S_1 and S_2 of the correct type with respect to R_1 and R_2 and such that $Comp(S_1, S_2)$. Since $Comp(R_1, R_2)$, we have $R_1(S_1) \cap R_2(S_2) = \emptyset$. Let's suppose that (q_1, r_1) is in $R_1\lfloor_r(S_1)$, 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 $R_2\lfloor_r(S_2)$; 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 $R_2\lfloor_r(S_2)$. Similarly we get that whenever (q_2, r_2) is in $R_2\lfloor_r(S_2)$ it is not in $R_1\lfloor_r(S_1)$. Therefore $R_1\lfloor_r(S_1) \cap R_2\lfloor_r(S_2) = \emptyset$. Since S_1, S_2 were arbitrary, we get $Comp(R_1\lfloor_r, R_2\lfloor_r)$. □

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 θ for Adam in $\mathcal{K}(\mathcal{A}, M)$ and calculate a new pair (v_1', v_2'). All the pairs will satisfy the invariant:

- $\mathit{Comp}(\rho_1, \mathit{res}(\rho_2, v_2))$ and $\mathit{Comp}(S_1, \mathit{res}(S_2, v_2))$, where v_1 labeled by $q : (N, \rho_1, S_1)$ and v_2 labeled by $q : (N, \rho_2, S_2)$.

The strategy. We define the strategy by considering one by one the rules for constructing the tree $\mathcal{K}(\mathcal{A}, M)$. As in the case for Eve's strategy, apart from the constant and the application rules, it is easy to see that the invariant is preserved. In the constant rule, Adam needs to make the same choices as in $\mathcal{K}(\mathcal{A}, M)$. The only complicated case is the application rule.

In the case of application, we are in a node labeled $q : (NK, \rho_1, S_1)$ that is a node of Eve and it has successors labeled $(q, R) : (NK, \rho_1, S_1)$ for every residual R of appropriate type (cf. Figure 2). Suppose Eve chooses some R and in consequence a node v_1'. Then Adam has a choice between the children of v_1' that have labels of one of the two forms:

$$\begin{aligned} q &: (N, \rho_1, R\restriction_{rk(q)} \cdot S_1) \\ q' &: (K, \rho_1\restriction_{r'}, R_1 \cdots R_l) \qquad \text{for } (q', r') \in R\restriction_{rk(q)}(R_1, \ldots, R_l). \end{aligned}$$

At the same time the node v_2 of $K(\mathcal{A}, M)$ is an application node so it has assigned residual $R(v_2)$. We have two cases.

Suppose $\mathit{Comp}(R\restriction_{rk(q)}, R(v_2))$ holds. In this case Adam chooses for v_1'' the node labeled $q : (N, \rho_1, R\restriction_{rk(q)} \cdot S_1)$. This works since the successor v_2' of v_2 is labeled by $q : (N, \rho_2, (v_2, K, \rho)S_2)$; hence the pair (v_1'', v_2') satisfies the invariant.

The other case is when $\mathit{Comp}(R\restriction_{rk(q)}, R(v_2))$ does not hold. This means that there are $(R_{1,1}, \ldots, R_{1,l})$ and $(R_{2,1}, \ldots, R_{2,l})$ such that $\mathit{Comp}(R_{1,i}, R_{2,i})$ for all $i = 1, \ldots, l$ and $R\restriction_{rk(q)}(R_{1,1}, \ldots, R_{1,l}) \cap R(v_2)(R_{2,1}, \ldots, R_{2,l}) \neq \emptyset$. Let (q', r') be the element from the intersection. Examining the definition of $R(v_2)$, Definition 13, the reason why $(q', r') \in R(v_2)(R_{2,1}, \ldots, R_{2,l})$ (see Figure 4) is that there is in $\mathcal{K}_\theta$ a node v_2' labeled by $q' : (x, \rho_2', S_2')$ such that $\rho_2'(x) = (v_2, K, \rho_2)$, $\mathit{res}(S_2', v_2') = (R_{2,1}, \ldots, R_{2,l})$ and $r' = \max(v_2, v_2')$. In that case, we choose for v_1'' the node labeled $q' : (K, \rho_1\restriction_{r'}, R_{1,1} \cdots R_{1,l})$. As its companion node we choose the successor v_2'' of v_2' labeled by $q' : (K, \rho_2, S_2')$. So the new position becomes (v_1'', v_2''). We need to show that $\mathit{Comp}(\rho_1\restriction_{r'}, \mathit{res}(\rho_2, v_2''))$ holds. For this take an arbitrary variable y for which $\rho_1(y)$ is defined. From $\mathit{Comp}(\rho_1, \mathit{res}(\rho_2, v_2))$ we derive by definition $\mathit{Comp}(\rho_1(y), \mathit{res}(\rho_2(y), v_2))$. As $r' = \max(v_2, v_2') = \max(v_2, v_2'')$ we have $\mathit{res}(\rho_2(y), v_2'') = \mathit{res}(\rho_2(y), v_2)\restriction_{r'}$, and then by Lemma 14 we get the required $\mathit{Comp}(\rho_1(y)\restriction_{r'}, \mathit{res}(\rho_2(y), v_2)\restriction_{r'})$. Since, by hypothesis, we have $\mathit{Comp}(R_{1,i}, R_{2,i})$ for all $i = 1, \ldots, l$, the new configuration satisfies the invariant.

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 a play reaches a node v_1 that is a variable node.

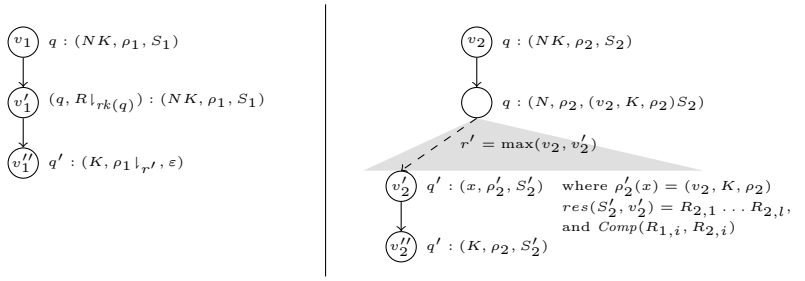

Figure 4: Case where $(q', r') \in R\downarrow_{rk(q)}(R_{1,1}, \dots, R_{1,l}) \cap R(v_2)(R_{2,1}, \dots, R_{2,l})$.

A variable node is labeled by $q : (x, \rho_1, S_1)$. To show that Adam wins here we need to prove that

$$(q, rk(q)) \notin \rho_1(x)(S_1) \ .$$

By the invariant, the companion node v_2 is labeled by $q : (x, \rho_2, S_2)$ and satisfying

$$Comp(R_x, res(\rho_2, v_2)(x)), \qquad Comp(S_1, res(S_2, v_2)) \ .$$

Suppose $\rho_2(x) = (v, N, \rho)$. Then $(q, \max(v, v_2)) \in R(v)(res(S_2, v_2))$ by the definition of $R(v)$ (Definition 13). Hence we also have

$$(q, \max(v, v_2)) \in R(v)(res(S_2, v_2))\downarrow_{\max(v,v_2)} \ ,$$

and in consequence

$$(q, rk(q)) \in R(v)(res(S_2, v_2))\downarrow_{\max(v,v_2)} \ .$$

As $R(v)\downarrow_{\max(v,v_2)} = res(\rho_2, v_2)(x)$ we get $Comp(\rho_1(x), R(v)\downarrow_{\max(v,v_2)}(x))$, from the invariant. As $Comp(S_1, res(S_2, v_2))$ we can obtain $(q, rk(q)) \notin \rho_1(x)(S_1)$ by the definition of *Comp*.

This completes the proof of Proposition 12.

3.4. Model checking

We can now conclude the proof of the decidability of the model-checking problem (Theorem 5). Our objective is to decide if $BT(M)$ is accepted by $\mathcal{A}$ for a given λY-term M and a parity automaton $\mathcal{A}$. By Proposition 8 this is equivalent to deciding if Eve has a winning strategy in the game $\mathcal{K}(\mathcal{A}, M)$. Proposition 12 tells us that Eve wins in $\mathcal{K}(\mathcal{A}, M)$ if and only if she wins in $G(\mathcal{A}, M)$. The latter is a finite parity game so it is decidable to determine who is the winner in $G(\mathcal{A}, M)$. Hence the algorithm is to construct the game $G(\mathcal{A}, M)$ and check if Eve has a winning strategy from the initial position in this game.

Let us briefly estimate the complexity of the procedure obtained from our proof. We write $|M|$ for the size of M which also gives an upper bound on the number of subterms of M. Let κ be the maximal order of a type of subterm

of M. Recall that an arity of a type is the number of arguments a term of this type can be applied to. An *extended arity of a term* is the sum of the arity of its type and the number of free variables in the term. Let ϑ denote the maximal extended arity of a subterm of M. Finally, we use $|\mathcal{A}|$ to denote the size of $\mathcal{A}$, and use d to stand for the biggest rank in the range of the rk function. We will write $Tower_\kappa(x)$ for the tower of exponentials function: $Tower_0(x) = x$ and $Tower_{i+1}(x) = 2^{Tower_i(x)}$.

First we calculate the number of possible residuals of a type of a given order (cf. Definition 9). For type 0, that has order 1, there are $2^{d|Q|}$ residuals. For a type of order 2 and arity bounded by ϑ, there are at most $(2^{d|Q|})^{2^{d|Q|\vartheta}}$ residuals. One can express this quantity as $Tower_2(d|Q|\vartheta)^{1+\varepsilon}$ for a suitable ε. In general for type of order r we have at most $Tower_r(d|Q|\vartheta)^{1+\varepsilon}$ residuals. Given this, the number of configurations of $G(\mathcal{A}, M)$ of the form $q : (N, \rho, S)$ is bounded by $|Q| \cdot |M| \cdot Tower_\kappa(d|Q|\vartheta)^{\vartheta+\varepsilon}$ thanks to the fact that ϑ gives a bound on the sum of free variables in N and length of the stack S. The number of configurations of the form $(q, R) : (N, \rho, S)$ is bounded by the same quantity since there is a bijection between such configurations and configurations $q : (N, \rho, R \cdot S)$. Finally, there are $|Q|^2 \cdot |M| \cdot Tower_\kappa(d|Q|\vartheta)^{\vartheta+\varepsilon}$ configurations of the form $(q_1, q_2) : (N, \rho, S)$. Adapting ε we obtain that the game $G(\mathcal{A}, M)$ has at most $|M| \cdot Tower_\kappa(d|Q|\vartheta)^{\vartheta+\varepsilon}$ states. Since it has d priorities it can be solved in time $(|M| \cdot Tower_\kappa(d|Q|\vartheta)^{\vartheta+\varepsilon})^{\lfloor d/2 \rfloor+1}$. To compare this complexity to that for schemes [KO09] let us note that a straightforward translation from schemes to λY-terms increases the order of types by 1 due to the fact that a recursive definition $F = \lambda\vec{x}.N_F$ is translated as $Y(\lambda F.\lambda\vec{x}.N_F)$. Then we have an additional exponent due to the fact that we need to consider also residuals for $Y(\lambda F.\lambda\vec{x}.N_F)$ while in op. cit. the fixpoint variables are treated externally. These two exponentials can be avoided by introducing a notion of a term in a canonical form and adapting the translation to give terms in such form. The Krivine machine needs to be slightly adjusted to treat the fixpoints in a special way, and not put YM on the stack as it is the case now. This construction is presented in [SW12].

4. 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 there is a finite automaton accepting the representation of a position of $\mathcal{K}(\mathcal{A}, M)$ iff this position is winning for Eve.

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 appearing in all the closures of ρ and S are subterms of M.

We start by defining tree representations of closures containing only subterms of M. The alphabet Σ_M of these trees is defined as follows. For every

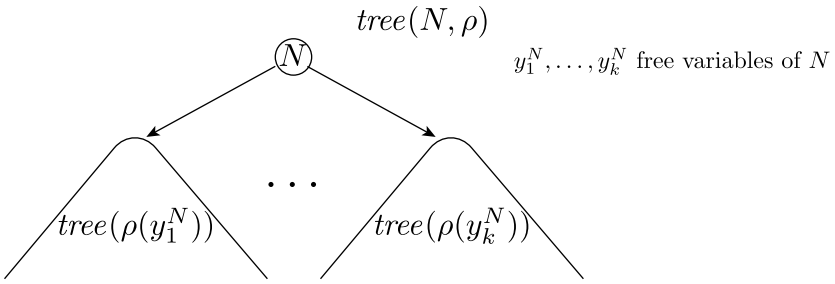

Figure 5: Tree representation of a closure (N, ρ).

subterm $N : \tau_1 \to \cdots \to \tau_l \to 0$ of M the alphabet Σ_M contains N as well as $Nz_1^N \ldots z_l^N$ for a fixed sequence of variables $z_1^N \ldots, z_l^N$ not free in N of types $\tau_1, \ldots, \tau_l$ respectively. The arity of a letter N or $Nz_1^N \ldots z_l^N$ from Σ_M is the number of free variables in the term. In particular, if N does not have free variables then a node labeled by N is a leaf in a tree. We denote by y_i^N the i-th free variable of N in some fixed ordering on variables. We have the following links between closure and tree notations (cf. Figure 5):

- A closure (N, ρ) is represented by a tree, $tree(N, \rho)$, whose root is labeled by N, and the subtree t_i rooted in the i-th child represents $\rho(y_i^N)$, that is $t_i = tree(\rho(y_i^N))$.
- Conversely, every tree t over Σ_M represents a closure: $closure(t) = (N, \rho)$ where N is the label of the root of t and $\rho(y_i^N) = closure(t_i)$ for subtrees $t_1 \ldots t_l$ of the root.
- Of course every t over Σ_M can be seen also as the term represented by $closure(t)$ (*i.e.* the term $term(closure(t))$), that we will denote simply $term(t)$.

A configuration $(N, \rho, C_1 \ldots C_k)$ of a Krivine machine is represented by a closure $(Nz_1^N \ldots z_k^N, \rho')$, where ρ' agrees with ρ on free variables of N, and moreover $\rho'(z_i^N) = C_i$; for $i = 1, \ldots, k$. So configurations of $\mathcal{K}(\mathcal{A}, M)$ are also represented as trees over Σ_M. Recall that a position $q : (N, \rho, S)$ of the game $\mathcal{K}(\mathcal{A}, M)$ is winning for Eve iff the Böhm tree of (N, ρ, S) is accepted by $\mathcal{A}$ from state q (Proposition 8). The following theorem implies that there is a finite automaton that can decide if a given configuration is winning.

Theorem 15 *For every term M of type 0 and every MSOL formula ψ the set:*

$$\{t \in Trees(\Sigma_M) : term(t) \text{ has type } 0 \text{ and } BT(term(t)) \vDash \psi\}$$

is a regular set of finite trees, and an automaton recognizing it can be effectively computed.

Let $\mathcal{A}$ be a parity automaton recognizing trees having the property ψ. We want to understand when for a given t, the automaton $\mathcal{A}$ accepts $BT(term(t))$. Let $Q_{\mathcal{A}}$ be the set of states of $\mathcal{A}$.

Of course we would like to use our reduction from infinite to finite games. By Proposition 8, $\mathcal{A}$ accepts $BT(term(t))$ iff Eve has a winning strategy in the game $\mathcal{K}(\mathcal{A}, term(t))$. By Proposition 12 the later is equivalent to Eve winning in $G(\mathcal{A}, term(t))$. This is this last characterization that we will use.

At the core of the proof we will have an alternating finite state automaton $\mathcal{B}$ recognizing closures. Its states are pairs (q, S), where q is a state, and S is a sequence of residuals. We will say that S is *type compatible* with a term N if N has a type $\tau_1 \to \cdots \to \tau_l \to 0$ and S is a sequence of l residuals of types $\tau_1, \ldots, \tau_l$ respectively. In a state (q, S) an automaton can read a symbol N that is type compatible with S, and then it has to do the following actions:

B1 Guess an assignment ρ_N of residuals to the free variables of N, such that the position $q : (N, \rho_N, S)$ is winning for Eve in $G(\mathcal{A}, M)$. If there is no such assignment then the automaton rejects.

B2 For every free variable y_i^N of N, every sequence S' of residuals type consistent with y_i^N, and every q' such that for some r' we have $(q', r') \in \rho_N(y_i^N)(S')$, the automaton will send a copy of itself with the state (q', S') to the i-th child of the root. If N has no free variables then the automaton accepts.

We describe in the next lemma the language accepted by the so constructed automaton $\mathcal{B}$, but before we need to define a slight generalization of the finite games $G(\mathcal{A}, M)$ introduced in Section 3.2. These games were defined only for terms M of type 0. Here we will consider $G(\mathcal{A}, N)$ for N of arbitrary type. The definition from Section 3.2 still applies. The only difference is that in this game there is no initial position $q : (N, \emptyset, \varepsilon)$, but rather $q : (N, \emptyset, S)$ for every S type compatible with N.

The next lemma describes the language of $\mathcal{B}$ and at the same time concludes the proof of the theorem.

Lemma 16 For every tree t over Σ_M, state q of $\mathcal{A}$, and sequence of residuals S type compatible with $term(t)$:

$\mathcal{B}$ accepts t from (q, S) iff

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

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

When t has only the root, the statement is immediate, by B1.

For the induction step suppose t has a root labeled by N and let $t_1, \ldots, t_k$ be the subtrees rooted at its children. Recall that by definition $term(t) = N[\sigma]$ where $\sigma(y_i^N) = term(t_i)$ for y_i^N the i-th free variable in N, for $i = 1, \ldots, k$.

We first consider the left to right direction. Assume that there is an accepting run of $\mathcal{B}$ on t from (q, S). Our goal is to show how Eve can win from position $q : (term(t), \emptyset, S)$ of $G(\mathcal{A}, term(t))$. Condition B1 of the definition of $\mathcal{B}$ gives us an environment ρ_N such that the position $q : (N, \rho_N, S)$ is winning for Eve in $G(\mathcal{A}, M)$. It will be convenient to see $term(t)$ as $N[\sigma]$. From the position $q : (N[\sigma], \emptyset, S)$ in $G(\mathcal{A}, term(t))$ Eve should play in the exactly same way as from $q : (N, \rho_N, S)$ in $G(\mathcal{A}, M)$ as long as it is possible. It stops being possible when the play reaches a position $q' : (term(t_i), \rho'', S')$ in $G(\mathcal{A}, term(t))$ and at the same time a leaf $q' : (y_i^N, \rho', S')$ in $G(\mathcal{A}, M)$. To complete the description of the winning strategy in $G(\mathcal{A}, term(t))$ we need to show how Eve can win from $q' : (term(t_i), \rho'', S')$. Since $term(t_i)$ is closed, it is enough to consider $q' : (term(t_i), \emptyset, S')$ instead.

We need now to find a winning strategy from $q' : (term(t_i), \emptyset, S')$. Since $q' : (y_i^N, \rho', S')$ is winning for Eve we get that $(q', \Omega(q')) \in \rho'(y_i^N)(S')$. Moreover, as y_i^N is a variable free in N, we have $\rho'(y_i^N) = \rho_N(y_i^N)\downarrow_r$ for some r. So, there is r'' such that $(q', r'') \in \rho_N(y_i^N)(S')$. Now, by condition B2, $term(t_i)$ is accepted from the state (q', S') of $\mathcal{B}$. By induction hypothesis Eve has a winning strategy from position $q' : (term(t_i), \emptyset, S')$ in $G(\mathcal{A}, term(t_i))$. This position also exists in the game $G(\mathcal{A}, term(t))$. Moreover in the two games the parts reachable from this position are identical since the moves from a position depend only on the form the position. So $q' : (term(t_i), \emptyset, S')$ is also winning 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 a position $q : (term(t), \emptyset, S)$ in $G(\mathcal{A}, term(t))$. As before, let us write $term(t)$ as $N[\sigma]$. Now follow the winning strategy from $q : (term(t), \emptyset, S)$ using $N[\sigma]$ notation. For every free variable y_i^N of N look at all the positions of the form $q' : (y_i^N[\sigma], \rho', S')$ reachable by a play consistent with the winning strategy and let r be the maximal rank seen between the initial position of the game and that position. Observe that $y_i^N[\sigma] = term(t_i)$. Since $term(t_i)$ does not have free variables, the position $q_i : (term(t_i), \emptyset, S')$ is winning in $\mathcal{G}(\mathcal{A}, term(t_i))$. By induction hypothesis, we have an accepting run on t_i from the state (q', S') of $\mathcal{B}$. Let R_i be the set of all such triples (q', r, S'). We define $\rho_N(y_i^N)(S') = \{(q', r) : (q', r, S') \in R_i\}$. This will make ρ_N satisfy condition B2. By definition of ρ_N we also get that $q : (N, \rho_N, S)$ is winning for Eve in $G(\mathcal{A}, M)$. Indeed, the winning strategy of Eve on $G(\mathcal{A}, term(t))$ can be transposed on $G(\mathcal{A}, M)$: it suffices to reproduce the moves as long it is possible. Thus the strategy in $G(\mathcal{A}, M)$ maintains a pair of positions respectively in $G(\mathcal{A}, M)$ and in $G(\mathcal{A}, term(t))$ $q' : (P, \rho_1, S')$ and $q' : (Q, \rho_2, S')$ so that $P[\sigma] = Q$ and ρ_1 is equal to ρ_2 for every variable but for the variables y_i^N for which $\rho_1(y_i^N) = \rho_N(y_i^N)\downarrow_r$ with r being the maximal rank seen since the beginning of the play. The strategy consists in replying to Adam in $G(\mathcal{A}, M)$ the same thing as Eve would reply to the corresponding move in $G(\mathcal{A}, term(t))$; a simple case analysis shows that this maintains the invariant. Then if the play is infinite or ends with a variable that is different from the y_i^N, the invariant together with the fact that we play with a winning strategy in $G(\mathcal{A}, term(t))$ implies that Eve wins. Now if the play ends in a position of the form $q' : (y_i^N, \rho', S')$, then Eve wins because, if r is the maximal rank seen on that play we have that

$\rho'(y_i^N) = \rho_N(y_i^N)\downarrow_r$, and thus by definition of ρ_N, (q', r) is in $\rho_N(S')$ and thus we have $(q', rk(q'))$ in $\rho_N(y_i^N)\downarrow_r(S')$. Hence condition B1 is satisfied too, and in consequence t is accepted from the state (q, S) of $\mathcal{B}$. □

Theorem 15 follows from Lemma 16. It is enough to consider the statement of the lemma when q is the initial state q_{init} of $\mathcal{A}$, and S is the empty stack. By Propositions 8 and 12, Eve winning from $q_{init} : (term(t), \emptyset, \varepsilon)$ in $G(\mathcal{A}, term(t), \varepsilon)$ is equivalent to $BT(term(t))$ being accepted by $\mathcal{A}$, and this by definition of $\mathcal{A}$ means that $BT(term(t)) \vDash \varphi$. So the automaton required in Theorem 15 is automaton $\mathcal{B}$ with (q_{init}, ε) as the initial state.

5. Conclusions

In this paper we have proposed to use the Krivine machine to analyze higher-order recursive schemes. Considering two prominent results in the area we have demonstrated that the rich structure of this formalism allows one 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, the Krivine machine is a very sophisticated model despite its simple presentation. As it happens, its relatively rigid structure appears to be a good framework that helps one to formulate strong invariants of the Böhm tree it computes with rather elementary definitions.

Let us comment further the relationship 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 the handling of ranks via $\downarrow_{\Omega(q)}$ operation is similar in both proofs. The typing rule for application gives 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 the 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 quite difficult.

In the present paper we have kept models of λY-calculus in the background. Yet, the two proofs strongly suggest that there may exist 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. In the meantime we have pursued this line of research [SW13].

References

[AC98] R. M. Amadio and P-L. Curien. *Domains and Lambda-Calculi*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1998.

[AdMO05] Klaus Aehlig, Jolie G. de Miranda, and C.-H. Luke Ong. The monadic second order theory of trees given by arbitrary level-two recursion schemes is decidable. In *TLCA'05*, volume 3461 of *LNCS*, pages 39–54, 2005.

[Aeh07] Klaus Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(1):1–23, 2007.

[BCHS12] Christopher H. Broadbent, Arnaud Carayol, Matthew Hague, and Olivier Serre. A saturation method for collapsible pushdown systems. In *ICALP (2)*, volume 7392 of *LNCS*, pages 165–176, 2012.

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

[BO09] C. Broadbent and C.-H. L. Ong. On global model checking trees generated by higher-order recursion schemes. In *FOSSACS*, volume 5504 of *LNCS*, pages 107–121, 2009.

[Bro12] C. H. Broadbent. The Limits of Decidability for First Order Logic on CPDA Graphs. In *STACS 2012*, volume 14 of *LIPIcs*, pages 589–600, 2012.

[CF58] H.B. Curry and R. Feys. *Combinatory Logic*, volume 1. North-Holland Publishing Co., Amsterdam, 1958.

[CHM+08] Arnaud Carayol, Matthew Hague, Antoine Meyer, Luke Ong, and Olivier Serre. Winning regions of higher-order pushdown games. In *LICS*, pages 193–204, Pittsburgh United States, 2008.

[CS12] Arnaud Carayol and Olivier Serre. Collapsible pushdown automata and labeled recursion schemes equivalence, safety and effective selection. In *LICS*, pages 165–174, 2012.

[Dam82] W. Damm. The IO- and OI-hierarchies. *Theor. Comp. Sci.*, 20:95–207, 1982.

[DG86] W. Damm and A. Goerdt. An automata-theoretical characterization of the OI-hierarchy. *Information and Control*, 71(1-2):1–32, 1986.

[Had12] A. Haddad. IO vs OI in higher-order recursion schemes. In *FICS*, volume 77 of *EPTCS*, pages 23–30, 2012.

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

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

[Kar10] A. Kartzow. Collapsible pushdown graphs of level 2 are tree-automatic. In *STACS*, volume 5 of *LIPIcs*, pages 501–512, 2010.

[KNU02] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS*, volume 2303 of *LNCS*, pages 205–222, 2002.

[KNUW05] T. Knapik, D. Niwinski, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and pannic automata. In *ICALP*, volume 3580 of *LNCS*, pages 1450–1461, 2005.

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

[Kob09] N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL*, pages 416–428. ACM, 2009.

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

[Ong06] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90, 2006.

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

[SW11] S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In *ICALP (2)*, volume 6756 of *LNCS*, pages 162–173, 2011.

[SW12] Sylvain Salvati and Igor Walukiewicz. Recursive schemes, Krivine machines, and collapsible pushdown automata. In *RP*, volume 7550 of *LNCS*, pages 6–20, 2012.

[SW13] S. Salvati and I. Walukiewicz. Using models to model-check recursive schemes. In *TLCA*, 2013. Full version submitted to a journal, available from the authors webpage.

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

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