Evaluation is MSOL compatible

Sylvain Salvati, Igor Walukiewicz

Evaluation is MSOL compatible

S. Salvati and I. Walukiewicz

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

Abstract

We consider simply-typed lambda calculus with fixpoint operators. Evaluation of a term gives as a result the Böhm tree of the term. We show that evaluation is compatible with monadic second-order logic (MSOL). This means that for a fixed finite vocabulary of terms, the MSOL properties of Böhm trees of terms are effectively MSOL properties of terms themselves. Theorems of this kind have been known for some graph operations: unfolding, and Muchnik iteration. Similarly to those results, our main theorem has diverse applications. It can be used to show decidability results, to construct classes of graphs with decidable MSOL theory, or to obtain MSOL formulas expressing properties of terms. Another application is decidability of a control-flow synthesis problem.

1 Introduction

Rice theorem tells us that no non-trivial property of the behaviour of a Turing machine can be decided by looking at the machine itself. In this paper we consider a much simpler abstract computing system: simply-typed lambda calculus with fixpoint operators. We denote it λY. A behaviour of a λY-term is its Böhm tree. Since not all λY-terms have normal forms, Böhm tree is a standard choice for the result of computation of a term. To express properties of results we use monadic second-order logic (MSOL) because it is a fundamental logic over trees. The *transfer theorem* we prove says that every MSOL property of Böhm trees is effectively an MSOL property of terms. In other words, we show that MSOL is compatible with evaluation.

The λY-calculus is a standard formalism in lambda-calculus community. It can be seen as a simplification of the programming language PCF introduced by Plotkin [Plo77]. It is much simpler than the full, untyped, lambda-calculus. For example, it is decidable if a λY-term has a normal form, while the same question for untyped lambda-calculus is undecidable. Still, λY-calculus has some nontrivial computational power. For example, the problem of deciding if two λY-terms are equivalent is undecidable [Sta04]. By

now, it is usual [Bar84] to approach the semantics of the lambda-calculus through infinite Böhm trees. Such a tree is just a normal form of the term, if the term has one. Otherwise it is a potentially infinite tree representing the visible part of the infinite computation of the term. In this paper we consider also infinite λY-terms. This is less standard but introduces relatively little complications while bringing real strengthening of the main theorem.

Under different syntax λY-calculus has also been intensively studied by language theoretic community. One can cite the PhD thesis of Fisher [Fis68] on macro languages, the work of Engelfriet and Schmidt on IO and OI [ES77, ES78], or the work of Damm on (safe) recursive schemes [Dam82]. More recently Knapik, Niwinski and Urzyczyn [KNU02] considered recursive schemes as generators of infinite trees, and studied the model-checking problem for such trees. After a series of intermediate results [KNUW05], Ong [Ong06] has shown that the model-checking problem of MSOL properties for such trees is decidable. It has been already clear to Engelfriet and Schmidt as well as to Damm that the grammars, or recursive schemes they study are a different representation of λY-terms (and their subclasses). Indeed trees generated by recursive schemes are just Böhm trees of corresponding terms of λY-calculus. So, for example, the theorem of Ong can be rephrased as saying that Böhm trees of finite λY-terms have decidable MSOL theory. The transfer theorem presented in this paper implies this decidability result.

Our transfer theorem says that for a fixed finite vocabulary of terms, an MSOL formula φ can be effectively tranformed into an MSOL formula $\widehat{\varphi}$ such that for every term M over the fixed vocabulary: M satisfies $\widehat{\varphi}$ iff the Böhm tree of M satisfies φ. The result is stronger than the Ong's theorem in at least two aspects. First, it holds also for infinite λY-terms. Second, and more importantly, the theorem gives an effective reduction of one theory to another. For example, since finiteness of a tree is definable in MSOL, we immediately obtain that if we restrict to λY-terms over a fixed finite vocabulary then the set of terms having a (finite) normal form is MSOL definable. In the last section of the paper we give several other applications of additional power provided by the transfer theorem.

In the terminology of Courcelle [Cou94] our result says that for λY-calculus MSOL is compatible with evaluation. A flagship example of a result of this kind is MSOL-compatibility of the unfolding operation. A finite automaton, i.e. a graph, can be unfolded into the tree of all its possible executions: this tree can be seen as the evaluation of the automaton. The MSOL-compatibility of the unfolding means that the MSOL theory of this tree can be effectively reduced to the MSOL theory of the automaton itself. The more powerful Muchnik iteration allows to get similar results for pushdown automata and higher-order pushdown automata. This way we obtain the pushdown hierarchy of trees with decidable MSOL theory. Here we consider λY-terms instead of automata as a computational model, and show the analogous result for evaluation instead of unfolding or Muchnik

iteration. The operation of evaluation cannot be directly compared to the other two since it works on different objects (trees with back edges instead of graphs). Yet in a well-known context where these operations can be compared, the evaluation operation is strictly stronger. Indeed, every tree in the pushdown hierarchy is a Böhm tree of some term, but not vice versa [Par12].

Related work: This work can be seen as generalization of Ong's theorem, in the same way as compatibility of MSOL with unfolding is a generalization of Rabin's theorem. Moreover the unfolding theorem is in some sense also a special case of our main theorem. Knapik and Courcelle [CK02a] have used ufolding theorem to prove a special case of our theorem for infinite terms of order 1. The proof presented here is based on our proof of Ong's theorem using Krivine machines [SW11].

MSOL properties of higher-order systems are an active area of research. It has been shown that many interesting properties of higher-order programs can be analyzed with recursive schemes and automata [Kob09b, Kob09a, Kob09c, KO11, OR11]. Since the decidability result of Ong has been revisited in a number of ways [HMOS08, KO09, SW11, BCHS12].

Plan of the paper In the next section we introduce infinitary λY-calculus: a simply typed λ-calculus of infinite terms with fixpoint operators. Extensions of untyped lambda-calculus to infinite terms are rather delicate. Here we have made some design choices that together with the use of typing allow to retain the fundamental properties in a relatively straightforward way. We prove that Böhm trees are still the canonical values for infinite λY-terms. We then introduce the notion of canonical form of a term. Finally, we also adapt Krivine machine to compute Böhm trees of infinite terms. More precisely, the machine computes Lévy-Longo trees of terms but for terms we are interested in the two notions coincide. We find it convenient to consider Y not as a constant but as a binder. So we write $Yx.N$ and not $Y(\lambda x.N)$. This allows to split variables into two categories λ-variables and Y-variables. The canonical form ensures that the dependencies between the types of variables are well-controlled. As it will turn out, the translation of a term to a canonical form is quite easy; it can be even realized by means of MSOL transduction.

Section 3 presents the main theorem. For this it describes how terms are represented as logical structures so that we can talk about their MSOL properties. Our representation requires that we have a fixed finite set of λ-variables. At the same time we do not need to restrict the number of Y-variables. We show that the theorem is not likely to hold if the number of λ-variables in not fixed.

The following section is devoted to the proof of the theorem. It starts with the overview of the proof. The major part of the proof works on terms in a canonical form. At the end we use the power of MSOL to get the result for all terms.

Section 5 gives three applications of transfer thoerem. We explain how to

obtain formulas expressing computational properties of terms. We show decidability of higher-order matching for terms over a fixed vocabulary [SS03]. We present a synthesis result that allows to construct λY-programs from λY-modules.

In the conlcusions section we give more relations between the transfer theorem and other results in the literature.

2 Infinitary λY-calculus

In this paper, we work with infinitary simply typed λY-calculus with fixpoints. We refer to it as *infinitary λY-calculus*, sometimes omitting the adjective "infinitary". In rare cases when we want terms to be finite, we say it explicitly. This section introduces infinitary simply typed λY-calculus as a rather straightforward generalization of finitary λY-calculus. We start with the notion of infinitary terms together with the operational semantics of the infinitary λY-calculus. We adopt a slightly modified syntax where Y is used as a variable binder rather than as a combinator. This allows us to distinguish between the variables that may be bound by Y from those that may be bound by λ. Such a distinction is of a little importance for the calculus itself, but it will allow us to get a more general theorem later when we will put restrictions on the number of variables. Next, we define the notion of *Böhm trees*. As it can be expected, Böhm trees are actual normal forms of infinitary terms, and every infinitary term has a normal form. Afterwards, we introduce a particular class of terms, that will be useful in the proof of the main theorem: *terms in canonical form*. Finally, we introduce a slight extension of the Krivine machine that is able to compute the normal forms of infinitary λY-terms, thus giving an actual computational content to infinitary λY-terms. The notion of Krivine machine we use is tailor-made so as to work on terms in canonical form.

2.1 Syntax and operational semantics

The *set of types* 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 $(\alpha \to \beta)$. As usual, so as to use less parentheses, we consider that $\to$ associates to the right. For example, $0 \to 0 \to 0$ stands for $(0 \to (0 \to 0))$. We will write $0^i \to 0$ as short notation for $0 \to 0 \to \cdots \to 0 \to 0$, where there are $i+1$ occurrences of 0. The order of a type is defined by: $order(0) = 1$, and $order(\alpha \to \beta) = max(1 + order(\alpha), order(\beta))$.

A *signature*, denoted Σ, is a set of typed constants, that is symbols with associated types. Of special interest to us will be *tree signatures* where all constants other than the special constant Ω have order at most 2. Observe that types of order 2 have the form $0^i \to 0$ for some i. For simplicity of

notation we will always assume that $i = 2$, but of course our results do not depend on this convention.

The terms will be built over two disjoint countable sets of typed variables: λ-variables and Y-variables. We shall write x^α for a λ-variable of type α, and $\mathbf{x}^\alpha$ for a Y-variable of type α. In this paper, we work with potentially infinite λY-terms. We assume that for every type α we have a constant Ω^α to denote the undefined term of type α. We will also have typed application symbols $@^\alpha$, and typed binders Y^α as well as $\lambda^{\alpha\to\beta}$. For all types α we define simultaneously the sets of infinite terms of type α as trees satisfying the following conditions.

- A node labelled by Ω^α, x^α, $\mathbf{x}^\alpha$, or c^α is a term of type α.
- A tree with the root labelled $@^\beta$ having as the left subtree a term of type $\alpha \to \beta$ and as a right subtree a term of type α, is a term of type β.
- A tree with the root labelled $\lambda^{\alpha\to\beta}x^\alpha$ with the unique immediate subtree being a term of type β, is a term of type $\alpha \to \beta$.
- A tree with a root labelled $Y^\alpha\mathbf{x}^\alpha$ with the unique immediate subtree being a term of type α, is a term of type α.

Some examples of infinitary λY-terms, as well as trees that are not terms, are presented in Figure 2.1.

Notice that all variables and constructors carry type labeling that makes typing of a term unique. We shall often omit those labels when they are unnecessary for the understanding or when they can be inferred from the context. We will also use standard conventions and write (MN) for $M\,@\,N$, and $N_0N_1\dots N_p$ for $(\dots(N_0N_1)\dots N_p)$.

We assume the usual notions of free and bound variables. Moreover, in general, we work up to the renaming of bound variables, that is up to α-conversion of terms. On infinitary terms, this convention is not really innocent since a term may contain infinitely many free variables. Nevertheless, a careful use of de Bruijn indices is, as in the finite case, a way to represent equivalence classes of terms modulo α-conversion. However, in the sequel, we are going to work sometimes on particular representatives of α-equivalence classes assuming certain properties on the naming of variables such as the Barendregt convention on Y-variables (every Y-binders binds a distinct Y-variable), and the use of finitely many λ-variables. To make things clear, when working up to α-conversion, we will speak about *terms*, and when working on particular instances of α-equivalence classes, we will speak about *concrete terms*.

This definition of infinitary terms comes with two main differences with respect to the usual one given for the untyped infinitary calculus. First, the

typing discipline rules out terms with infinite sequences of λ-abstractions (cf. Figure 2.1). The second difference is that we use the Y combinator as a binder and we distinguish between Y-variables and λ-variables. Notice that the definition of infinitary terms allows infinite sequences of Y-abstractions (cf. Figure 2.1).

The reason why we need to distinguish between λ-variables and Y-variables is that the main theorem we prove is about terms which use finitely many λ-variables but possibly infinitely many Y-variables. As a small remark, if the main theorem did not need to make this assumption, we could simply get rid of Y-binders. Indeed the term $Y\mathbf{x}.N$ has the same Böhm tree (see section 2.2) as the term $rec(\lambda x.N[x/\mathbf{x}])$ where $rec = \lambda f.f(f(f(f\ldots)))$. This shows that λ-abstraction, or parameter instantiation, is more powerful than recursive definition in the context of the infinitary λ-calculus.

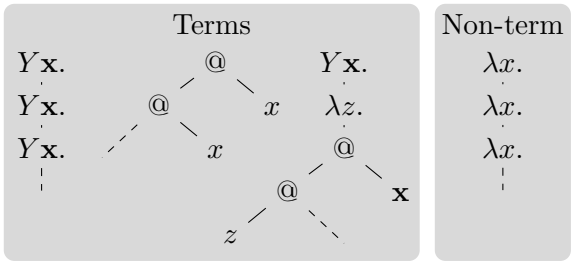


Figure 1: Examples of infinitary terms and non-terms

The notion of capture-avoiding substitution can be easily extended to infinitary λ-calculus. Given two terms M and N we shall write $M[x := N]$ (*resp.* $M[\mathbf{x} := N]$) for the result of the capture-avoiding substitution of N for the free occurrences of x (*resp.* $\mathbf{x}$) in M. It is important to notice that x or $\mathbf{x}$ may have infinitely many free occurrences in M, and that computing the result of $M[x := N]$ or of $M[\mathbf{x} := N]$ may require an infinite amount of work. This is why, following a suggestion made in Terese [], we will use the Krivine machine later on so as to compute substitutions explicitly and to avoid the infinite overhead of computation that each substitution may require if it were to be completely performed.

We may now define β-contraction on infinitary terms as the straightforward extension of β-contraction on finite terms. Concerning δ-contraction, we need to adapt its definition to our slightly modified syntax. The relation of δ-contraction $\to_\delta$ is the smallest relation that is compatible with the syntax of infinitary λY-calculus and so that $Y\mathbf{x}.M$ δ-contracts to $M[\mathbf{x} := Y\mathbf{x}.M]$. We let $\beta\delta$-contraction, $\to_{\beta\delta}$ to be the union of the relations $\to_\beta$ and $\to_\delta$. Of course, the subject reduction property for simple types transfers from finite terms to infinite ones so that the reduction preserves typing. As we are interested in infinitary terms as computational devices, we need to choose what we consider to be a value or the output of the computa-

tion performed by those terms. We thus introduce the notion of weak head normal form and of weak head reduction.

Definition 1 An infinitary term M is in *weak head normal* form, if it is of the form $\lambda x^{\alpha}.N$ for some term N, or if it is of the form $hN_1 \dots N_n$, with h being either a variable or a constant different from Ω.

When M is an infinitary term of the form $(\lambda x.\ P)P_1 \dots P_n$ or of the form $(Y\mathbf{x}.\ P)P_1 \dots P_n$, then M has a *head-redex*. Reducing M to $P[x := P_1]P_2 \dots P_n$ or to $P[\mathbf{x} := Y\mathbf{x}.P]P_1 \dots P_n$, respectively, is called *head-contracting* M. We write $M \to_h N$ when M head-contracts to N; we write $M \to_h^* N$ for head-reduction in some finite number of head-contraction steps.

The reason why, we wish to use weak-head normal forms for values instead of the usual notion of head-normal form is that we are going to use the Krivine machine to compute values. Krivine machine computes weak-head normal forms of terms and not head-normal forms.

2.2 Böhm trees

Now that we have settled a reduction strategy together with a notion of value, we may define a notion of *normal form* for infinitary terms, namely Böhm trees. We also show that in a strong sense Böhm trees are actual normal forms of infinitary terms. There is no particular difficulty to define the notions of this subsection for all infinite λY-terms. Yet, since we consider infinitary λY-terms as generators of infinite trees, we are really interested only in closed terms of type 0. We choose to present the definitions only for such terms. In order to further simplify the notation we will from the start consider only terms over a tree signature, and assume that all the constants are binary.

Definition 2 (Böhm trees) Given a closed term M of type 0 over tree signature, we define its Böhm tree, $BT(M)$, as follows:

1. if $M \to_h^* bN_1N_2$ where b is a constant different from Ω, then $BT(M)$ is a tree with root labelled b and with $BT(N_1)$ and $BT(N_2)$ as its subtrees.

2. otherwise $BT(M) = \Omega^{\alpha}$.

Observe that in our case Böhm trees are just labelled infinite binary trees. In a sense that can be made precise, Böhm trees are normal forms of terms. As such they are terms too, but due to their special shape we do not need to use application nodes to represent them. In Figure 2 we present a Böhm tree and its representation as a term.

The reader may be surprised that we talk about Böhm trees while in general Krivine machines compute Lévy-Longo trees. It turns out that the two notions coincide when working with tree signatures and terms of type 0. This is why we have preferred to use the better known notion.

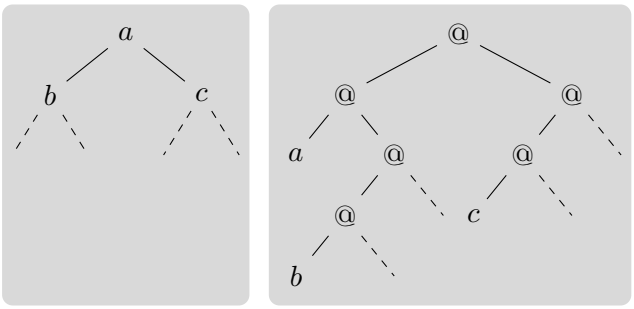
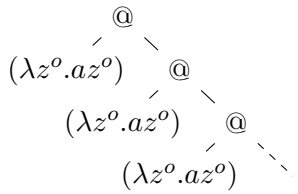


Figure 2: Böhm tree and the associated term

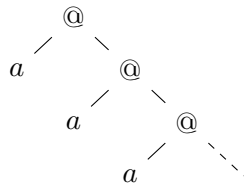
Even though we have defined Böhm trees (or Lévy-Longo trees) using a particular reduction strategy, they really are the unique normal forms of infinitary terms modulo $\beta\delta$-reductions of arbitrary ordinal length provided we add the reduction rule that allows to reduce terms without weak head normal form to Ω. We are using the end of this section to establish this fact.

The results we are about to show are known in the literature on untyped infinitary λ-calculus (see Kennaway *et al.* [KKSdV97] and Terese [KdV03]). We here adapt those results to the simply typed λY-calculus and we present them in way that suits better our needs. Notice that, even though we have defined Böhm trees only for closed terms of atomic types built on a tree signature, all the results we mention here hold without those restrictions.

As we are working with infinite terms, computing their possible normal forms may require an infinite number of $\beta\delta$-contraction steps. For example the term that is defined as the solution of the equation $u = (\lambda z^0.az^0)u$ that can be depicted as:



requires to reduce ω redices so a to obtain the term that is the solution of the equation $v = av$:



If we were to reduce the term $(\lambda x^0.bx^0)u$ we may first reduce u in ω steps to v and we would obtain a term $(\lambda x^0.bx^0)v$ that can be reduced to bv with one more step. With this example we have constructed a reduction of length $\omega + 1$. It thus appears natural to define reductions of arbitrary ordinal length. The natural way of defining such sequences of reductions is to define them as continuous functions from ordinals to λ-terms. This is one of the reason why we need the constants Ω^α as part of our language for defining infinitary terms. The constants Ω^α stands for the undefined term of type α and allows us to define a natural partial order on infinitary terms. This is the least order $\sqsubseteq$ that is compatible with the syntax and so that for every term M in $\Lambda^{\alpha,\infty}(\Omega)$, $\Omega^\alpha \sqsubseteq M$. Notice that, whenever $M \sqsubseteq N$, M and N need to be terms that have the same types.

Definition 3 Given a relation R on λ-terms and an ordinal γ, a γ, R-reduction sequence of type α is a function φ that maps ordinals $\delta \leq \gamma$ to infinitary terms of $\Lambda^{\alpha,\infty}(\Omega)$ so that:

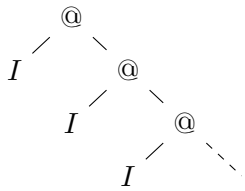
1. if $\delta < \gamma$, we have $\varphi(\delta)\ R\ \varphi(\delta + 1)$,
2. if $\delta \leq \gamma$ is a limit ordinal, then for every term $M \sqsubseteq \varphi(\delta)$ there is an ordinal $\theta < \delta$ so that $M \sqsubseteq \varphi(\theta)$.

When $R = \rightarrow_{\beta\delta}$, and there is a $\alpha, \rightarrow_{\beta\delta}$-sequence φ such that $\varphi(0) = M$ and $\varphi(\alpha) = N$ we write that $M \xrightarrow{\alpha}_\infty N$, or we write $M \longrightarrow_\infty N$ when there is α such that $M \xrightarrow{\alpha}_\infty N$. The notion of reduction sequences we use is called *weakly convergent* in the literature.

It is already known that in the untyped case infinitary β-reduction is not Church-Rosser. Typing does not fix this problem and it is easy to adapt the examples of the untyped case to the typed case. Indeed, if we take the terms $I = \lambda x^0.x^0$, $M = \lambda x^0.I(\mathbf{f}^{0\to 0}x^0)$ and $J = \lambda x^0.\mathbf{f}^{0\to 0}x^0$, then the term $(Y\mathbf{f}^{0\to 0}.M)x^0$ we can be reduced in the following ways:

$$\begin{array}{ll}(Y\mathbf{f}^{0\to 0}.M)x^0 & \longrightarrow_\infty \lambda x^0.I(Y\mathbf{f}^{0\to 0}.Mx^0)\\ & \longrightarrow_\infty (\lambda x^0.I((\lambda x^0.I(Y\mathbf{f}^{0\to 0}.Mx^0))x^0))x^0\\ & \longrightarrow_\infty I(I(Y\mathbf{f}^{0\to 0}.Mx^0)\\ & \longrightarrow_\infty I^\omega\end{array}$$

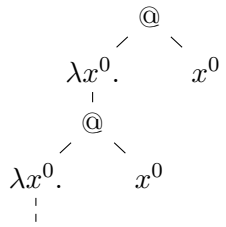
where I^ω is the term satisfying the syntactic identity $II^\omega = I^\omega$ and I^ω can be depicted by:



The term $(Y\mathbf{f}^{0\to 0}.M)x^0$ can also be reduced as follows:

$$(Y\mathbf{f}^{0\to 0}.M)x^0 \longrightarrow_\infty (\lambda x^o.(Y\mathbf{f}^{0\to 0}.J)x^0)x^0 \longrightarrow_\infty (\lambda x^0.(YJ)x^0)x^0 \xrightarrow{\omega}_\infty u$$

where u is the infinite term verifying the syntactic identity $u = (\lambda x^0.u)x^0$ and that is depicted by:



Since, for any redex that is contracted in I^ω the result of the contraction is I^ω again and similarly u can only be reduced to u, it is obvious that there is no term P so that $I^\omega \longrightarrow_\infty P$ and $u \longrightarrow_\infty P$. Nevertheless we do not assume that those terms are meaningful values, we rather assume that meaningful values are terms in weak-head normal forms.

With this notion of value we may enrich the our operational semantics with a reduction to Ω for terms that do not yield a value.

Definition 4 We introduce the relation $\to_\Omega$, the Ω-*contraction*, so that, given a term M of $\Lambda^{\alpha,\infty}(\Omega^0)$, if there is no term N in weak-head normal from so that $M \longrightarrow_\infty N$, then $M \to_\Omega \Omega^\alpha$. We let $\to_{\beta\delta\Omega}$ be the union of $\to_{\beta\delta}$ and $\to_\Omega$.

When there is a $\alpha, \to_{\beta\delta\Omega}$-sequence φ such that $\varphi(0) = M$ and $\varphi(\alpha) = N$ we write that $M \xrightarrow{\alpha}_{\Omega,\infty} N$, or we write $M \longrightarrow_{\Omega,\infty} N$ when there is α such that $M \xrightarrow{\alpha}_{\Omega,\infty} N$.

The operational semantics $\longrightarrow_{\Omega,\infty}$ on infinitary terms has nice properties, and in particular it has the Church-Rosser property (it is a consequence of Terese [] Theorem 12.9.6 p.699).

Theorem 5 *If $M \longrightarrow_{\Omega,\infty} N_1$ and $M \longrightarrow_{\Omega,\infty} N_2$, then, there is P so that $N_1 \longrightarrow_{\Omega,\infty} P$ and $N_2 \longrightarrow_{\Omega,\infty} P$.*

This confluence property is partially grounding the fact that Böhm trees of terms are their normal forms in proving that every term has a unique

normal form. It remains to see that these unique normal forms are the Böhm trees. The main problem is that there is a gap between the relation $\rightarrow_\Omega$ which may reduce a term M to Ω only when there is no *infinitary reduction* that turns M into a weak-head normal form while the Böhm tree of a term M is Ω when there is no *finite head-reduction* that turns M into a weak-head normal form. This is precisely this gap that Lemma 6 is filling. We now give its proof.

Lemma 6 Given M in $\Lambda^{\alpha,\infty}(\Omega)$, if there a term N in weak-head normal form so that $M \longrightarrow_\infty N$, then there is $k < \omega$ and P in weak-head normal form so that $M \xrightarrow{k}_{h,\infty} P$.

Proof
Let γ be an ordinal so that $M \xrightarrow{\gamma}_\infty N$ and let φ be the $\gamma, \rightarrow_{\beta\delta}$-reduction sequence reducing M to N. In case $\gamma < \omega$, it is easy to see that there is a finite term $M' \sqsubseteq M$ which can be put in finitely many steps of head-contraction in weak-head normal form P'. Therefore preforming those head-contraction steps on M allows to obtain a term P so that $P' \sqsubseteq P$, and therefore P is also in weak-head normal form.

In case $\gamma \geq \omega$, there is a finite ordinal k and a limit ordinal γ' so that $\gamma = \gamma' + k$. Let $P = \varphi(\gamma')$, we have that $P \xrightarrow{k}_\infty N$. As N is in weak-head normal form, there is a finite term P' so that $P' \sqsubseteq P$ so that P' head-reduces in finitely many steps to P'' which is in weak-head normal form. By definition of $\gamma, \rightarrow_{\beta\delta}$-reduction sequences, there is $\theta < \gamma'$ so that $P' \sqsubseteq \varphi(\theta)$. As a consequence, $\varphi(\theta)$ head-reduces in finitely many steps to a weak-head normal form. Thus, by transfinite induction, M head-reduces to a weak-head normal form in finitely many steps. □

We can now state that Böhm trees are normal forms of infinitary term and that they may be computed with at most ω steps of $\beta\delta\Omega$-contraction (this is also a consequence of Corollary 12.9.15 p.701 in Terese [KdV03]). This also gives Böhm trees an actual computational content since it shows that, provided an infinitary term is given in some effective manner, its Böhm tree can be computed. Hereafter, we use the Krivine machine as an actual device performing this computation.

Theorem 7 *Given M in $\Lambda^{\alpha,\infty}(\Omega)$, there an ordinal $\gamma \leq \omega$ so that $M \xrightarrow{\gamma}_{\Omega,\infty} BT(M)$.*

2.3 Canonical terms

Infinitary terms may contain infinitely many free λ-variables. But, as we already said, we are interested in closed terms and the subterms of closed terms contain only finitely many free λ-variables. Thus, from now on, we restrict our attention to terms with finitely many free λ-variables.

It will be convenient to work with terms in what we call *canonical form*. This form permits to separate λ-variables from Y-variables making recursion and parameter instantiation isolated in a way that will prove useful.

Definition 8 An infinitary term M is in a *canonical form* when none of its subterms of the form $Y\mathbf{x}.\ N$ contains a free λ-variable.

There is a simple process, that transforms every term M with a finite set of free λ-variables into a canonical form. For this it suffices to abstract away free λ-variables in every subterm $Y\mathbf{x}.\ N$ of M. More precisely, if $\{x_1, \ldots, x_n\}$ is the set of free λ-variables of N (which is necessarily finite) we perform the replacement

$$Y\mathbf{x}.\ N \qquad \mapsto \qquad (Y\mathbf{y}.\lambda x_1 \ldots \lambda x_n . N[\mathbf{x} := \mathbf{y}x_1 \ldots x_n])x_1 \ldots x_n$$

With standard techniques based on approximations, it is rather direct to show that the new term has the same Lévy-Longo tree as M.

Proposition 1 *For every infinite closed λY-term M of type 0 over tree signature the result of the above operation is a canonical term generating the same Böhm tree.*

2.4 Krivine machine

We are now going to introduce the notion of Krivine machine that will allow us to compute the normal forms of infinite terms.

Since we wish to treat δ-contraction in a particular manner, we use the Krivine machine to reduce a particular *concrete canonical closed term M of type* 0. This means that the names of bound variables do matter – actually they will matter only for Y-variables. We make this choice to avoid the introduction of environments for Y-variables that would induce some unnecessary notational burden. If we were to be completely rigorous, we should parametrize each notion related to the Krivine machine with this term M, and we should speak about M-Krivine machine, M-environments, M-closures, M-configurations... As M will be always clear from the context we will mention it explicitly only if necessary.

Let us fix for this section a concrete canonical term M of type 0. Working with concrete terms gives us some control over Y-variables. We will assume that in the term M every Y-binder binds a distinct Y-variable and that there is no occurrence of Ω. This assumption on the names of Y-variables allows us to stipulate the existence of a function *term* that maps a Y-variable $\mathbf{x}^\alpha$ to the subterm $Y\mathbf{x}^\alpha.N$ of M where it is bound. In later sections, we will represent terms as infinite graphs and the function *term* will be implemented directly with an edge between Y-binders and the positions they bind in the term.

The Krivine machine [Kri07], is an abstract machine that computes the weak head normal form of a 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:

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

As in this grammar, we will use $\emptyset$ for the empty environment. The notation $\rho[x \mapsto C]$ represents the environment which associates the same closure as ρ to variables except for the variable x that it maps to C. The terms N that can be used to form a closure must be subterms of the term M we are considering. This restriction is harmless since, when computing the normal form of M, the Krivine machine only needs closures made with subterms of M (see Theorem 11).

Since we work within a typed context, these two notions follow the typing discipline: in an environment the types of a variable and the closure it is assigned to must be the same. The type of a closure (N, ρ), is simply the type of the term N. We require that in a closure (N, ρ), the environment is defined for every free λ-variable of N, while the values of Y variables are given by *term* function.

Intuitively a closure C denotes a closed λ-term $E(C)$ as follows. A closure (N, ρ) denotes a λ-term obtained by substituting: (i) for every free λ-variable x of N the term denoted by the closure $\rho(x)$, and (ii) for every free Y-variable $\mathbf{x}$ the term $term(\mathbf{x})$. It is important to note that the definition of an environment is inductive, so every environment has finite depth. More precisely this means that every sequence of the form:

$$\rho_0(x_0) = (N_1, \rho_1), \ \rho_1(x_1) = (N_2, \rho_2), \ldots$$

is finite and ends in the empty environment.

A *stack* $S \equiv C_1 \ldots C_n$ is a possibly empty sequence of closures. We use $\perp$ to denote the empty stack.

A *configuration of a Krivine machine* is a triple (N, ρ, S) where:

1. N is a term (a subterm of M);
2. ρ is an environment defined for all free variables of N;
3. S is a stack $C_1 \ldots C_k$, where k and the types of the closures are determined by the type of N: the type of C_i is α_i where the type of N is $\alpha_1 \to \cdots \to \alpha_k \to 0$.

A configuration (N, ρ, S) represents an infinitary term:

$$E((N, \rho, S)) = E(N, \rho)E(C_1) \ldots E(C_n)$$

where $C_1 \dots C_n$ are the closures on the stack S. Notice that the third condition in the definition of configurations implies that $E(N, \rho, S)$ needs to be a term of type 0. Observe also that $E(N, \rho, S)$ may not be a subterm of M even though N as well as every term appearing in ρ, C_1,...,C_n is.

The transition rules of the Krivine machine are:

$$\begin{aligned}
(\lambda x.N, \rho, (K, \rho')S) &\to (N, \rho[x \mapsto (K, \rho')], S) \\
(Y\mathbf{x}.N, \rho, S) &\to (N, \rho, S) \\
(NK, \rho, S) &\to (N, \rho, (K, \rho)S) \\
(x, \rho, S) &\to (N, \rho', S) \quad \text{where } (N, \rho') = \rho(x) \\
(\mathbf{x}, \rho, S) &\to (term(\mathbf{x}), \emptyset, S)
\end{aligned}$$

It is rather straightforward to check that these computation rules transform configurations of the Krivine machine into configurations of the Krivine machine. This means that the typing properties of environments and the fact that only subterms of M are used in a closure or as the main term of configuration is preserved by those reduction rules. The first two rules do β-contraction and δ-contraction respectively. The application rule creates a closure and puts it on the stack. The λ-variable rule looks up the meaning of the variable in the environment. The Y-variable rule replaces the variable by its definition. Since we work with concrete canonical terms, there are no free λ-variables in $term(\mathbf{x})$ so the environment can be discarded.

Notice that the terms reduced by the Krivine machine, that are in its environment or on its stack are in general infinite terms which are inspected with respect to their syntax. A more concrete, but equivalent way of representing the same thing would be to build the Krivine machine with addresses in the infinite term and a call to an external mechanism so as to read its labels.

We use the Krivine machine to compute a tree that, as we will see, is the Böhm tree of the term. Recall that we consider only terms over tree signatures and we assume that all the constants are binary (cf. page 4).

Definition 9 Let $(N, \rho, \bot)$ be a configuration of the Krivine machine, with N a term of type 0. We define the tree $KT(N, \rho, \bot)$ as follows:

1. if started in $(N, \rho, \bot)$ the machine reaches a configuration (b, ρ, C_1C_2) where b is a constant and $C_i = (N_i, \rho_i)$ are closures of type 0, then $KT(N, \rho, \bot)$ is a tree with the root labelled by b and two subtrees: $KT(N_1, \rho_1, \bot)$ and $KT(N_2, \rho_2, \bot)$.
2. in the other case $KT(N, \rho, \bot) = \Omega^0$.

We write $KT(M)$ for $KT(M, \emptyset, \bot)$.

The next lemma says that Krivine machine computes the weak head normal form (that is the same as head normal form in our case).

Lemma 10 Let $(N, \rho, \bot)$ be a configuration of the Krivine machine. Term $E(N, \rho, \bot)$ has a head normal form iff Krivine machine reduces $(N, \rho, \bot)$ to a configuration (b, ρ, S) for some constant b.

Proof
Recall that Lemma 6 guarantees that head reduction reaches a weak head normal form if a term has one. We also know that Krivine machine performs the head reduction. It suffices to examine in what configurations the machine gets blocked. Looking at the rules we can see that there can be only two kinds of configurations when no rule is applicable. The first is $(\lambda x.N, \rho, \bot)$, but it is excluded by the third condition on configurations of Krivine machine. The second is (h, ρ, S) where h is either a constant or a variable that is not in the domain of definition of ρ. The case of a variable is also excluded by the second condition on the form of configurations of the Krivine machine. Hence the machine stops when it reaches a constant that is the head of the normal form of $E(N, \rho, \bot)$. □

Lemma 10 entails that the Krivine machine gives an effective way of computing the Böhm tree of an infinitary term. The second statement of the following theorem follows by a direct inspection.

Theorem 11 *For a fixed tree signature Σ. For every concrete canonical and closed λY-term M of type 0, we have $BT(M) = KT(M)$. All the terms appearing in configurations of the Krivine machine during the computation of $KT(M)$ are subterms of M.*

3 Transfer theorem for evaluation

In this section we will investigate logical theories of terms and Böhm trees. We will consider monadic-second order logic (MSOL) on such objects. For this, it will be essential to restrict to some finite set of λ-variables: both free and bound. On the other hand, we will be able to handle infinitely many Y-variables. Once we make it clear how to represent terms and Böhm trees as logical structures, we will state our main theorem. We will justify our representation of terms by showing two facts: (i) the set of all terms is definable in MSOL, (ii) unless the polynomial-time hierarchy collapses, the main theorem is false when we do not fix the number of bound λ-variables.

Terms will be represented as labelled, potentially infinite, graphs. For us here such a graph is a structure of the form $\mathcal{M} = \langle V, \{E_i\}_{i=1,2}, \{P_i\}_{i=1,\ldots,n} \rangle$, where V is the set of vertices, every E_i is a binary relation on vertices, and every P_i is a subset of vertices. We will have two edge relations representing left and right successor. In our case predicates P_i will be a partition of V.

So every vertex will have a unique label given by the predicate it belongs too.

Monadic second-order logic (MSOL) is an extension of first-order logic with quantification over sets of elements and the membership predicate $x \in Z$, where x is a variable ranging over elements, and Z is a variable ranging over sets of elements. The definition of satisfiability of a formula φ in a structure $\mathcal{M}$, denoted $\mathcal{M} \vDash \varphi$, is standard.

Let us fix a tree signature Σ with finitely many constants other than Ω. As postulated at the beginning of Section 2.1, for simplicity of notation all the constants are binary. We would like to consider terms as models of formulas of monadic second-order logic. We will work with terms over some arbitrary but finite vocabulary. We take a finite set of typed λ-variables $\mathcal{X} = \{x_1^{\alpha_1}, \ldots, x_k^{\alpha_k}\}$, and a finite set of types $\mathcal{T}$. We denote by $\mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ the set of *infinite closed concrete terms*[1] M over the signature Σ such that M uses only λ-variables from $\mathcal{X}$, and every subterm of M has a type in $\mathcal{T}$. We also write $\mathit{CTerms}(\Sigma, \mathcal{T}, \mathcal{X})$ for the terms M of $\mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ that are in canonical form. Observe that bound λ-variables in M should come from $\mathcal{X}$. In contrast we do not put restrictions on the use Y-variables. It will be convenient to assume that every Y-variable in M is bound at most once: for every Y-variable $\mathbf{x}$ there is at most one occurrence of $Y\mathbf{x}$ in M.

A term from $\mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ can be seen as a labelled tree where the labels come from finite alphabet, but for the Y-variables and Y binders. We will now eliminate the possible source of infiniteness of labels related to Y-variables and Y binders. Take a closed term M considered as a tree. For every node of this tree labelled by a Y-variable $\mathbf{x}^\alpha$ we put an E_1-edge from the node to the node labelled $Y\mathbf{x}^\alpha$. Since M is closed, such a node exists and is an ancestor of the node labelled by $\mathbf{x}$; such a node is also unique since we assume that every Y-variable is bound at most once. In the next step we introduce a new symbol $\Rsh^\alpha$ and for every node labelled with a Y-variable of type α, we change its label to $\Rsh^\alpha$. Finally, we replace all labels of the form $Y\mathbf{x}^\alpha$ by just Y^α. This way we have eliminated all occurrences of Y-variables from labels, but now a term is represented not as a labelled tree but as a labelled graph. Let us denote it by $\mathit{Graph}(M)$. Observe that the nodes of this graph have labels from a finite set

$$\begin{aligned} \mathit{Talph}(\Sigma, \mathcal{T}, \mathcal{X}) \quad = \quad & \Sigma \cup \{@^\alpha, Y^\alpha, \Rsh^\alpha \colon \alpha \in \mathcal{T}\} \cup \mathcal{X} \cup \\ & \{\lambda^{\alpha\to\beta} x^\alpha : \alpha \in \mathcal{T} \wedge \alpha \to \beta \in \mathcal{T} \wedge x^\alpha \in \mathcal{X}\} \ . \end{aligned}$$

There are two edge relations, E_1 and E_2, in $\mathit{Graph}(M)$ since nodes labelled by application symbol have both left and right successor. The nodes with other labels have no successor (nodes labeled with labels from $\Sigma \cup \mathcal{X}$) or just one successor, given by E_1. The example of $\mathit{Graph}(M)$ is presented in Figure 3.

[1] Recall that concrete terms are particular elements of α-equivalence classes.

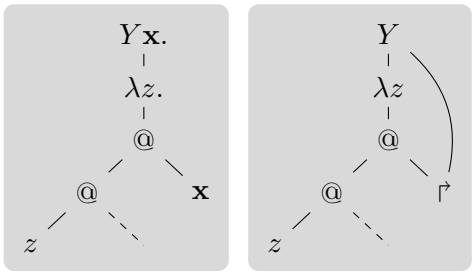


Figure 3: M and $\mathit{Graph}(M)$

Since $\mathit{Graph}(M)$ is a labelled graph over a finite alphabet it makes sense to talk about satisfiability of an MSOL formula in this graph. We will just write $M \models \varphi$ instead of $\mathit{Graph}(M) \models \varphi$. The first easy, but important, observation is that for fixed Σ, $\mathcal{T}$, $\mathcal{X}$, there is an MSOL formula determining if a graph is of the form $\mathit{Graph}(M)$ for some $M \in \mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$. Indeed, in this case we deal with models over the signature consisting of two binary relations E_1, E_2 and a unary relation P_b for every $b \in \mathit{Talph}(\Sigma, \mathcal{T}, \mathcal{X})$. The formula should say that P_b form the partition of the set of vertices and then express conditions from the definition of infinite λY-terms on page 5. These conditions are clearly expressible in MSOL as they talk about dependencies between labels of a node and its successors.

For a closed term $M \in \mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ of type 0, its Böhm tree is a tree with nodes labelled by symbols from Σ. Hence one can talk about satisfiability of MSOL formulas in $BT(M)$. The transfer theorem says that the MSOL-theory of $BT(M)$ is recursive in the MSOL-theory of M.

Theorem 12 (Transfer theorem) *Let Σ be a finite tree signature, $\mathcal{X}$ a finite set of typed variables, and $\mathcal{T}$ a finite set of types. For every MSOL formula φ one can effectively construct an MSOL formula $\widehat{\varphi}$ such that for every λY-term $M \in \mathit{Terms}(\Sigma, \mathcal{T}, X)$ of type 0:*

$$BT(M) \models \varphi \qquad \textit{iff} \qquad M \models \widehat{\varphi}.$$

Notice that the formula $\widehat{\varphi}$ is independent from M.

In principle, it is possible to represent terms with an unbounded number of λ-variables by using the same trick for λ-binder as the one we have used for the Y-binder. However, we conjecture that the transfer theorem does not hold when we allow infinitely many λ-variables. Below we give a simple argument under the hypothesis that the polynomial hierarchy is strict.

It is customary to represent booleans with the λ-terms of type $0 \to 0 \to 0$: *true* is represented by $\lambda xy.x$ and *false* by $\lambda xy.y$. This permits the definition of the boolean connectives as terms too: $\mathit{and} = \lambda b_1 b_2 xy.b_1(b_2 xy)y$, $\mathit{or} = \lambda b_1 b_2 xy.b_1 x(b_2 xy)$ and $\mathit{neg} = \lambda bxy.byx$. One can also define propositional quantifiers $\mathit{All} = \lambda f.\mathit{and}(f\ \mathit{true})(f\ \mathit{false})$ and $\mathit{Ex} = \lambda f.\mathit{or}(f\ \mathit{true})(f\ \mathit{false})$.

This allows us to represent in a direct manner every quantified boolean formula θ as a simply typed finite closed term M_θ such that M_θ reduces to *true* iff θ is true. Observe that M_θ has a linear size with respect to that of θ. So if we assume that we are given two distinct terms N_{true}, N_{false} of type 0 we get $M_\theta N_{true} N_{false}$ reduces to N_{true} iff θ is true. Take an MSOL formula φ that is true exactly in $Graph(N_{true})$. The transfer theorem without any restriction on the number of λ-variables would give an MSOL formula $\widehat{\varphi}$ such that, for every quantified boolean formula θ: θ is true iff $M_\theta \vDash \widehat{\varphi}$. The model-checking problem for the fixed formula $\widehat{\varphi}$ belongs to the polynomial hierarchy: the level of the hierarchy is bounded from the above by the alternation of quantifiers in $\widehat{\varphi}$. So the extension of the transfer theorem to infinite number of λ-variables would imply that QBF satisfiability problem is in the polynomial hierarchy.

Using finite terms to prove that the transfer theorem does not hold in case an unbounded number of λ-variables is allowed cannot give a better result than the example we have given with QBF since the evaluation in a fixed finite model of finite terms that use only types from a finite set $\mathcal{T}$ can easily be proved to be in PSPACE. Thus, so as to get rid of the hypothesis that the polynomial hierarchy is strict to prove that the transfer theorem does not hold when an unbounded number of λ-variables is allowed, it seems that we need to use an infinite term.

4 Proof of the transfer theorem

This section presents a proof of the transfer theorem. Before we give the proof we need to introduce parity automata which recognize infinite trees satisfying MSOL formulas. We then give an overview of the the proof. We first prove the transfer theorem for terms in canonical form. We then remove this assumption by observing that transformation of a term into a canonical form is MSOL definable.

4.1 Parity autamata and MSOL on infinite binary trees

Recall that Σ is a fixed set of constants of type $0 \to 0 \to 0$. For a closed term M of type 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 define sets of Böhm trees. Such an automaton has the form

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

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 a function $r : \{0,1\}^* \to Q$ such that $r(\varepsilon) = q^0$ and for every

sequence $w \in \{0,1\}^*$: $(r(w0), r(w1)) \in \delta(q, t(w))$. The run is accepting if for every infinite path in the tree, the sequence of states assigned to this path satisfies the parity condition determined by 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 rk^0. We will simply assume that every tree containing rk^0 is rejected by the automaton. This is frequently done in this context. Handling rk^0 would not be difficult but would require to add one more case in all the constructions. The other, more difficult, solution is to convert a term to a term not generating rk^0.

4.2 Structure of the proof

The proof of this theorem will use games. The core of the proof will work with concrete terms M in canonical form. Since we have assumed that all our constants are binary, $BT(M)$ is a binary tree. There is a non-deterministic parity automaton $\mathcal{A}$ on infinite binary trees that recognizes binary trees that are models of φ. For a concrete canonical term M we will define a game $\mathcal{K}(\mathcal{A}, M)$. This game will be presented in terms of configurations of the Krivine machine. Eve will win in $\mathcal{K}(\mathcal{A}, M)$ iff $BT(M)$ is accepted by $\mathcal{A}$. Then we will define a reduced game $G(\mathcal{A}, M)$ with the property that Eve wins in the later game iff she wins in $\mathcal{K}(\mathcal{A}, M)$. Finally, we will show that $G(\mathcal{A}, M)$ is MSOL definable inside M (considered as the graph $Graph(M)$). Since there is an MSOL formula γ_{win} describing games where Eve wins, this gives the desired formula $\hat{\varphi}$. The last step is to remove the assumption that M is in canonical forms. For this it is enough to observe that the translation of M into a canonical form is an MSOL transduction. The schema of the proof is presented in Figure 4.

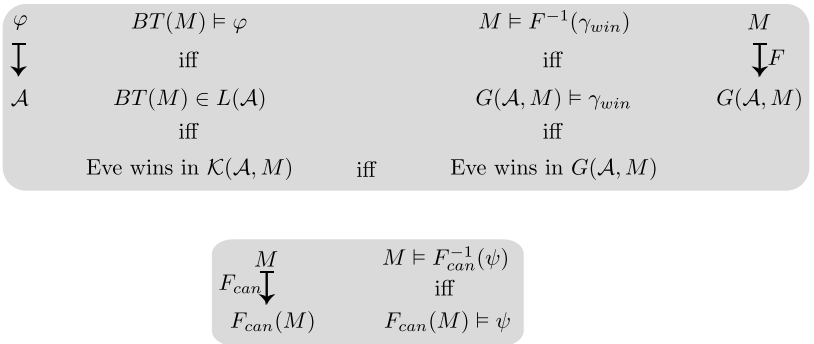


Figure 4: Schema of the proof

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

We now give the definition of $RT(\mathcal{A}, M)$, the runs of the automaton $\mathcal{A}$ on the graph of configurations of the Krivine Machine computing $BT(M)$. The actual runs of $\mathcal{A}$ on $BT(M)$ can easily be read off $RT(\mathcal{A}, M)$.

Definition 13 For a given $M \in CTerms(\Sigma, \mathcal{T}, X)$ of type 0 , and a parity automaton $\mathcal{A}$ we define the tree of runs $RT(\mathcal{A}, M)$ of $\mathcal{A}$ on the graph of configurations of the execution of the Krivine Machine on M:

1. The root of the tree is labeled with $q^0 : (M, \emptyset, \bot)$
2. A node labeled $q : (a, \rho, S)$ has a successor $(q_0, q_1) : (a, \rho, S)$ for every $(q_0, q_1) \in \delta(q, a)$.
3. A node labeled $(q_0, q_1) : (a, \rho, (v_0, N_0, \rho_0)(v_1, N_1, \rho_1))$ has two successors $q_0 : (N_0, \rho_0, \bot)$ and $q_1 : (N_1, \rho_1, \bot)$.
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 $q : (Y\mathbf{x}.N, \rho, S)$ has a unique successor $q : (N, \rho, S)$.
6. A node v labeled $q : (\mathbf{x}, \rho, S)$, for $\mathbf{x}$ a recursive variable, has a unique successor $q : (term(\mathbf{x}), \emptyset, S)$.
7. A node v labeled $q : (NK, \rho, S)$ has a unique successor labeled $q : (N, \rho, (v, K, \rho)S)$. We say that here a v-closure is *created*.
8. A node v labeled $q : (x, \rho, S)$, for x a λ-variable and $\rho(x) = (v', N, \rho')$, 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 or constant rule (rules 8 and 3), 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 2,3,4 rely on the typing properties of the Krivine machine ensured by the definition of its configurations (cf. page 13). Indeed, when the machine reaches a configuration of the form (a, ρ, S) then, since we are working with tree signature, a is of type $0 \to 0 \to 0$. In consequence, the stack S consists of two closures of type 0. The environment ρ plays no role in such a configuration as a is a constant. 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 14 We use the tree $RT(\mathcal{A}, M)$ to define a game between two players: Eve chooses a successor in nodes of the form $q : (a, \rho, S)$, and Adam in nodes $(q_0, q_1) : (a, \rho, S)$. We set the parity rank of nodes labeled $q : (a, \rho, S)$ to $rk(q)$, and the parity ranks of all the other nodes to 1. We can use max parity condition to decide who wins an infinite play. Let us call the resulting game $\mathcal{K}(\mathcal{A}, M)$.

The following is a direct consequence of the definitions and Theorem 11.

Proposition 2 *For every parity automaton $\mathcal{A}$ and concrete canonical term M. 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 rk 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.

Hence deciding whether $BT(M)$ is accepted by $\mathcal{A}$ is reduced to deciding who has a winning strategy from the root of $\mathcal{K}(\mathcal{A}, M)$. We will introduce a "smaller" game $G(\mathcal{A}, M)$, and show that the winner in the two games is the same. While $G(\mathcal{A}, M)$ will still be infinite, it will be definable by MSOL formula inside the term M itself.

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

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

The idea of the reduction is to eliminate stacks and environments using alternation. Consider situation in Figure 5. 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 (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. This means that 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 $G(\mathcal{A}, M)$.

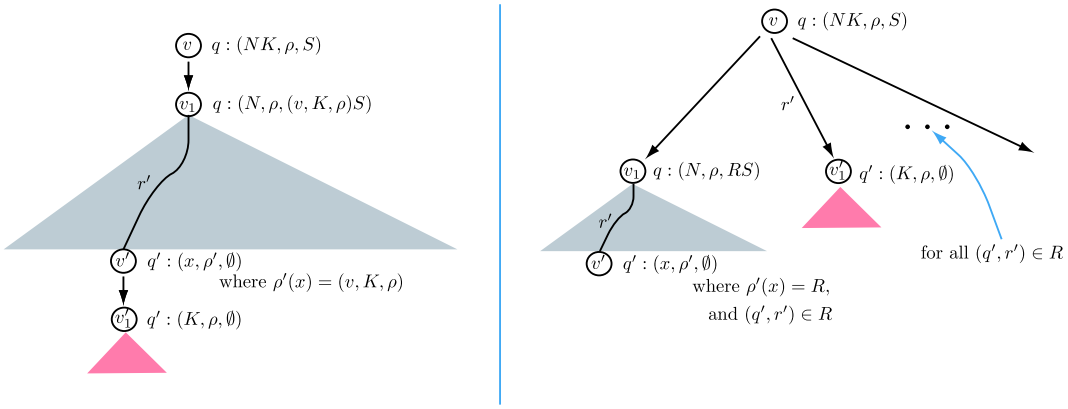


Figure 5: Game $\mathcal{K}(\mathcal{A}, M)$ on the left, and $G(\mathcal{A}, M)$ on the right.

The right part of Figure 5 represents the result of taking these shortcuts. In a set R, that we call residual of the closure (K, ρ), we have collected all states q' which appear when the closure (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 same problem for reachability games can be solved using much lighter methods. 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 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.

The final level of complication comes from the fact that till now we have assumed 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 type is determined by the type of x, that is the same as the type of K. Observe that the typing invariant of 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, 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.

After these 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 in an elegant way.

Definition 15 (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 will define the game.

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

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

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 λ-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 λ-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.

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 5 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 perform this verification. Of course this operation is defined for residuals of any orders.

Definition 16 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 $R\restriction_r$ of the same type as R satisfying for every sequence of arguments S:

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

Recall that $D_0 = \mathcal{P}(Q \times [d])$ so what $R\restriction_r$ 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

$$\begin{aligned} R\restriction_1 &= \{(q_1, 0); (q_1, 1); (q_2, 2); (q_3, 3)\}, \\ R\restriction_2 &= \{(q_2, 0); (q_2, 1); (q_2, 2); (q_3, 3)\}, \\ R\restriction_3 &= \{(q_3, 0); (q_3, 1); (q_3, 2); (q_3, 3)\}, \text{ and} \\ R\restriction_4 &= \emptyset. \end{aligned}$$

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 $R\restriction_2$ maps $\{(q_1, 1)\}$ to $\{(q_2, 0); (q_2, 1); (q_2, 2); (q_3, 3)\}$ and all other residuals to $\emptyset$.

Lemma 17 For every residual R and ranks r_1, r_2: $(R\restriction_{r_1})\restriction_{r_2} = R\restriction_{\max(r_1, r_2)}$.

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

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

$$\begin{aligned} q : (\lambda x.N, \rho, R \cdot S) &\to q : (N, \rho[x \mapsto R], S) \\ q : (a, \rho, R_0 R_1) &\to (q_0, q_1) : (a, \rho, R_0 R_1) \qquad \text{for } (q_0, q_1) \in \delta(q, a) \\ q : (Y\mathbf{x}.N, \rho, S) &\to q : (N, \rho, S) \\ q : (\mathbf{x}, \rho, S) &\to q : (term(\mathbf{x}), \rho, S) \qquad \mathbf{x} \text{ a recursion variable} \end{aligned}$$

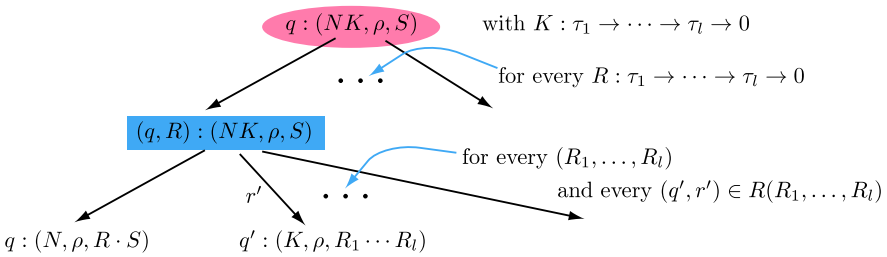


Figure 6: Dealing with application in $G(\mathcal{A}, M)$.

We now proceed to the rule for application (cf. Figure 6). 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

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

In the last line $R\restriction_{rk(q)}$ 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 in which vertices. Eve chooses a successor from vertices of the form $q : (NK, \rho, S)$, and $q : (a, \rho, R_0R_1)$. 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 residuals proposed by Eve.

Observe that we do not have a rule for nodes with a term being a λ-variable. Also positions of the form $(q_0, q_1) : (a, \rho, R_0R_1)$ are terminal. This means that we need to say who is the winner in nodes of these two forms.

For the case of a variable, Eve wins in a position

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

if $(q, rk(q)) \in R_x(R_1, \ldots, R_k)$.

For the case of a constant, Eve wins in a position

$$(q_0, q_1) : (a, \rho, R_0R_1)$$

if $(q_0, rk(q_0)) \in R_0\restriction_{rk(q_0)}$ and $(q_1, rk(q_1)) \in R_1\restriction_{rk(q_1)}$. Observe that in this case both R_0 and R_1 are necessarily residuals of type 0.

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 transitions of the form $(q, R) : (NK, \rho, S) \to q' : (K, \rho\restriction_{r'}, R_1 \cdots R_k)$ that have rank r'.

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.

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

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

Proposition 3 *For every parity automaton $\mathcal{A}$ and concrete canonical term M. Eve wins in $\mathcal{K}(\mathcal{A}, M)$ iff Eve wins in $G(\mathcal{A}, M)$.*

The proof of this lemma proceeds as follows. For the direction from left to right we take a winning strategy for Eve in $\mathcal{K}(\mathcal{A}, M)$ and define residuals for every closure with respect to this strategy. Then we show how Eve is winning in $\mathcal{G}(\mathcal{A}, M)$ using these residuals. The winning strategy in $\mathcal{G}(\mathcal{A}, M)$ 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 3.

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

We here introduce the key notion of the proof, the notion of *residuals of nodes*. Given a subtree $\mathcal{T}$ of $\mathcal{K}(\mathcal{A}, M)$, i.e. a tree obtained from $\mathcal{K}(\mathcal{A}, M)$ by pruning some of its subtrees, we calculate the residuals $R_{\mathcal{T}}(v)$ and $res_{\mathcal{T}}(v, v')$ for some nodes and pair 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 type. 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 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 $\mathcal{T}$. 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 $Q \times [d]$. We have two cases

- We put $(q', \max(v, v')) \in R(v)$ when there is v' in $\mathcal{T}$ labeled $q' : (x, \rho', \bot)$ such that $\rho'(x) = (v, K, \rho)$;
- We put $(q_i, \max(v, v'')) \in R(v)$ when there is v'' in $\mathcal{T}$ labelled $q_i : (K, \rho, \bot)$ having a parent labelled $(q_0, q_1) : (a, \rho', C_0C_1)$ with $C_i = (v, K, \rho)$; for $i = 0$ or $i = 1$.

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 $(v_1, N_1, \rho_1) \dots (v_k, N_k, \rho_k)$ with N_i of type τ_i. We put

$$(q', \max(v, v')) \in R(v)(R(v_1) \restriction_{\max(v_1,v')}, \dots, R(v_k) \restriction_{\max(v_k,v')}) \ .$$

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 18 ($R(v)$ 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 write $res(v, v_1)$ for $R(v) \restriction_{\max(v,v_1)}$. 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.

Let v be a node of $\mathcal{T}$ labelled 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 $res(S', v') = \vec{R}$.
- $(q_i, \max(v, v'))$, for every node v' labelled $q_i : (K, \rho, \bot)$ having a parent labelled $(q_0, q_1) : (a, \rho', C_0C_1)$ with $C_i = (v, K, \rho)$; for $i = 0$ or $i = 1$. Notice that this case applies only if K is of type 0.

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

Let's assume that Eve has a winning strategy σ on $\mathcal{K}(\mathcal{A}, M)$ defining the subtree $\mathcal{K}_\sigma$ of $\mathcal{K}(\mathcal{A}, M)$. Let's also assume that we have computed the residuals for $\mathcal{K}_\sigma$ as in Definition 18. We will use the residuals to define a winning strategy for Eve in $G(\mathcal{A}, M)$.

The invariant 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 $q : (N, \rho_1, S_1)$ and v_2 is labeled by $q : (N, \rho_2, S_2)$, where $\rho_1 = res(\rho_2, v_2)$ and $S_1 = res(S_2, v_2)$,
- v_1 is labeled by $(q_0, q_1) : (a, \rho_1, S_1)$ and v_2 is labeled by $(q_0, q_1) : (a, \rho_2, S_2)$ with $\rho_1 = res(\rho_2, v_2)$ and $S_1 = res(S_2, v_2)$.

The strategy The initial positions in both games have the same label $q^0 : (M, \emptyset, \bot)$, 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 chose are the nodes with a constant or with an application. A node with a constant is of the form $q : (a, \rho_1, R_0 R_1)$. Eve should then simply take the same transition of the automaton as taken from v_2. So it advances to a node labelled $(q_0, q_1) : (a, \rho_1, R_0 R_1)$. It is clear that the invariant is satisfied as the environment and the stack do not change. This implies moreover that no matter what Adam's next move is, the new position also satisfies the invariant.

The strategy and its analysis in the case of application node is more complicated. 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 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

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

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

Suppose Adam chooses v_1'' whose label is as in (2). By definition $R(v_2) \restriction_{rk(q)} = res(v_2, v_2)$. Hence the stack $(R(v_2) \restriction_{rk(q)}) \cdot S_1$ 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 7). Looking at the definition of $R(v_2)$, Definition 18, the first possible case is 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 $\rho_1 \restriction_{r'} = res(\rho_2, v_2) \restriction_{\max(v_2, v_2'')} = res(\rho_2, v_2'')$ by Lemma 17. Hence the pair (v_1'', v_2'') satisfies the invariant.

The second possibility for $(q', r') \in R(v_2) \restriction rk(q)$ is when K has type 0, and there is a node v_2' labelled $q_i : (K, \rho, \bot)$ having a parent labelled $(q_0, q_1) : (a, \rho', C_0 C_1)$ with $C_i = (v, K, \rho)$; for $i = 0$ or $i = 1$. By definition

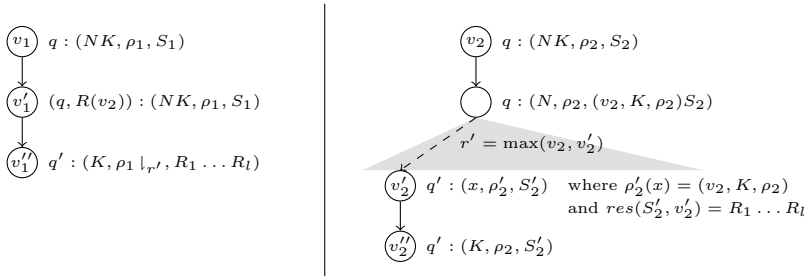


Figure 7: Adam chooses a node of the form (3)

of $R(v_2)$ we have that $r' = \max(v_2, v_2')$. We can take v_2' as a companion of v_1'' since $\rho_1 \lfloor_{r'} = res(\rho_2, v_2) \lfloor_{\max(v_2, v_2')} = res(\rho_2, v_2')$ by the invariant and Lemma 17. Hence the pair (v_1'', v_2') satisfies the invariant.

The strategy is winning We need to show that the strategy defined above is winning. Consider a sequence of nodes $(v_1^1, v_2^1), (v_1^2, v_2^2), \dots$ 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 or a constant node.

A variable 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 $(q, rk(q)) \in R(v)(res(S_2, v_2)) \lfloor_{\max(v,v_2)}$ since $rk(q) \leq \max(v, v_2)$. This is the required statement $(q, rk(q)) \in R_x(S_1)$.

A constant node is labeled by $(q_0, q_1) : (a, \rho_1, R_0 R_1)$. We need to show that $(q_i, rk(q_i)) \in R_i \lfloor_{rk(q_i)}$, for $i = 0, 1$. Let $i = 0$, the argument is the same for $i = 1$. By the invariant we have that the companion node v_2 is labeled by $(q_0, q_1) : (a, \rho_2, C_0 C_1)$. Suppose C_0 is (v, N, ρ). So v_2 has a successor v_2' labelled with $q_0 : (N, \rho, \bot)$. We have that $R_0 = R(v) \lfloor_{\max(v,v_2)}$, since $S_1 = res(S_2, v_2)$ by the invariant. By definition of $R(v)$ we get $(q_0, m) \in R(v)$; where $m = \max(v, v_2')$. From the definition $\lfloor_m$ operation: $(q_0, m) \in$

$R(v)\lfloor_m$. Which implies $(q_0, rk(q_0)) \in R(v)\lfloor_m$ as $rk(q_0) \leq m$. Since $m = \max(\max(v, v_2), rk(q_0))$ we get $R(v)\lfloor_m = (R(v)\lfloor_{\max(v,v_2)})\lfloor_{rk(q_0)} = R_0\lfloor_{rk(q_0)}$, and we are done.

4.5.3 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 $G(\mathcal{A}, M)$ form 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 $\mathcal{K}(\mathcal{A}, M)$, and 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 18. 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 complementarity predicate $Comp(R_1, R_2)$ between a pair of residuals:

- For $R_1, R_2 \in D_0$ we put $Comp(R_1, R_2)$ 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: *Comp* predicate is a logical relation (see [AC98]), but we have prefer 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 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 $\lfloor_r$ operation

Lemma 19 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 $R_1 \restriction_r (S_1)$. Therefore $R_1 \restriction_r (S_1) \cap R_2 \restriction_r (S_2) = \emptyset$. Since S_1, S_2 were arbitrary, we get $Comp(R_1 \restriction_r, R_2 \restriction_r)$. □

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:

> v_1 labeled by $q : (N, \rho_1, S_1)$ and v_2 labeled by $q : (N, \rho_2, S_2)$;
> where $Comp(\rho_1, res(\rho_2, v_2))$ and $Comp(S_1, res(S_2, v_2))$;

The strategy We define the strategy by considering one by one the rules for constructing the tree $\mathcal{K}(\mathcal{A}, M)$. The only case where Adam makes a choice is the application rule.

A node labeled $q : (NK, \rho_1, S_1)$ is a node of Eve and it has successors labeled $(q, R) : (NK, \rho_1, S_1)$ for every residual R of appropriate type. Suppose Eve chooses some R and in consequence such a node v_1'. Then Adam has a choice between the children of v_1' that have labels of the form:

$$q : (N, \rho_1, R \restriction_{rk(q)} \cdot S_1) \tag{4}$$

$$q' : (K, \rho_1 \restriction_{r'}, R_1 \cdots R_l) \qquad \text{for } (q', r') \in R \restriction_{rk(q)} (R_1, \ldots, R_l) \tag{5}$$

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 $Comp(R \restriction_{rk(q)}, R(v_2))$ 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 $Comp(R \restriction_{rk(q)}, R(v_2))$ 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 $Comp(R_{1,i}, R_{2,i})$ 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 18, there are two reasons why $(q', r') \in R(v_2)(R_{2,1}, \ldots, R_{2,l})$.

The first case (see figure 8) is when there is in $\mathcal{K}_\theta$ a node v_2' labeled by $q' : (x, \rho_2', S_2')$ such that $\rho_2'(x) = (v_2, K, \rho_2)$, $res(S_2', v_2') = (R_{2,1}, \ldots, R_{2,l})$ 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' labelled by $q' : (K, \rho_2, S_2')$. So the new position becomes (v_1'', v_2''). We need to show that $Comp(\rho_1 \restriction_{r'}, res(\rho_2, v_2''))$ holds. For this take an arbitrary variable y for which $\rho_1(y)$ is defined. Since $Comp(\rho_1, res(\rho_2, v_2))$ we have $Comp(\rho_1(y), res(\rho_2(y), v_2))$. As $r' = \max(v_2, v_2') = \max(v_2, v_2'')$ we have $res(\rho_2(y), v_2'') = res(\rho_2(y), v_2) \restriction_{r'}$, and then by Lemma 19 we get the required $Comp(\rho_1(y) \restriction_{r'}, res(\rho_2(y), v_2) \restriction_{r'})$. And of course, by hypothesis

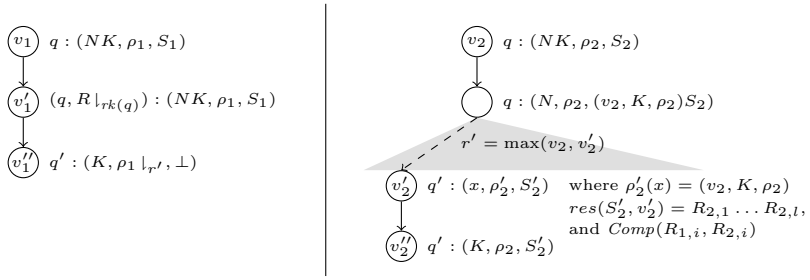


Figure 8: First case when $(q', r') \in R\restriction_{rk(q)} (R_{1,1}, \dots, R_{1,l}) \cap R(v_2)(R_{2,1}, \dots, R_{2,l})$

we have $Comp(R_{1,i}, R_{2,i})$ for all $i = 1, \dots, l$ so that the new configuration satisfies the invariant.

The second case (see Figure 9) is when K has type 0 and there is in $\mathcal{K}_\theta$ a node v_2' labelled $q_i : (K, \rho_2, \bot)$, with $q_i = q'$, having a parent labelled $(q_0, q_1) : (a, \rho', C_0C_1)$ with $C_i = (v, K, \rho_2)$; for $i = 0$ or $i = 1$ (Figure 9 represents the case where $i = 0$); and $r' = \max(v_2, v_2')$. For v_1'' take the node labelled $q' : (K, \rho_1\restriction_{r'}, \bot)$, and for its companion take the node v_2'. We need to show that $Comp(\rho_1\restriction_{r'}, res(\rho_2, v_2'))$ holds. For this take arbitrary variable y for which $\rho_1(y)$ is defined. Since $Comp(\rho_1, res(\rho_2, v_2))$ we have $Comp(\rho_1(y), res(\rho_2(y), v_2))$. As $r' = \max(v_2, v_2')$ we have $res(\rho_2(y), v_2') = res(\rho_2(y), v_2)\restriction_{r'}$, and then by Lemma 19 we get the required $Comp(\rho_1(y)\restriction_{r'}, res(\rho_2(y), v_2)\restriction_{r'})$.

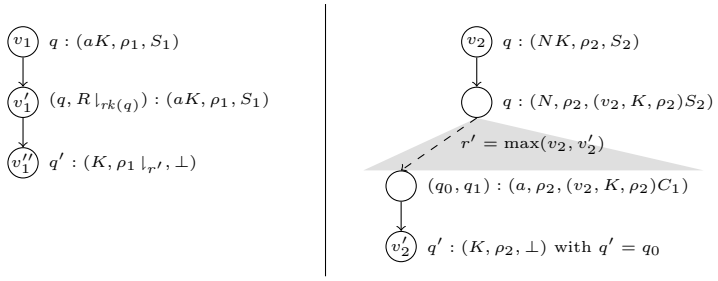


Figure 9: Second case when $(q', r') \in R\restriction_{r'} \cap R(v_2)$ (with $i = 0$)

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 or a constant node.

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

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

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 18). Hence we also have

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

and in consequence

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

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

A constant node is labelled by $(q_0, q_1) : (a, \rho_1, R_0R_1)$. To show that Adam wins here we need to prove that $(q_i, rk(q_i)) \notin R_i \restriction_{rk(q_i)}$ for $i = 0$ or $i = 1$. By the invariant, the companion node v_2 is labeled by $(q_0, q_1) : (a, \rho_2, C_0C_1)$ and satisfying $Comp(R_i, res(C_i, v_2))$ for $i = 0, 1$. The node v_2 must have a successor in $\mathcal{K}_\theta$. Let it be $q_0 : (N, \rho, \bot)$ where $C_0 = (v, N, \rho)$. We will show that $(q_0, rk(q_0)) \notin R_0 \restriction_{rk(q_0)}$. We have $Comp(R_0, res(C_0, v_2))$ so also $Comp(R_0 \restriction_{rk(q_0)}, res(C_0, v_2) \restriction_{rk(q_0)})$ by Lemma 19. But then we have $res(C_0, v_2) \restriction_{rk(q_0)}$ is $(R(v) \restriction_{\max(v,v_2)}) \restriction_{rk(q_0)}$. Let $m = \max((\max(v, v_2), rk(q_0))$. By definition of $R(v)$, Definition 18, $(q_0, m) \in R(v)$. Then $(q_0, m) \in R(v) \restriction_{\max(v,v_2)}$ and $(q_0, m) \in (R(v) \restriction_{\max(v,v_2)}) \restriction_{rk(q_0)}$ so that $(q_0, rk(q_0)) \in (R(v) \restriction_{\max(v,v_2)}) \restriction_{rk(q_0)}$. By the definition of $Comp$ predicate $(q_0, rk(q_0)) \notin R_0 \restriction_{rk(q_0)}$.

4.6 Transductions

In this subsection we will finish the proof of the theorem. We know that for every term M in concrete canonical form: $BT(M) \vDash \varphi$ iff Eve has a winning strategy in the game $\mathcal{K}(\mathcal{A}, M)$. We have also shown that the later condition is equivalent to Eve having a winning strategy in $G(\mathcal{A}, M)$. To conclude we will construct a formula $\widehat{\varphi}$ such that $M \vDash \widehat{\varphi}$ iff Eve has a winning strategy in $G(\mathcal{A}, M)$. We will also show how to get rid of the assumption that M is in a canonical form.

In order to construct the formula $\widehat{\varphi}$ we will examine the definition of the game and use MSOL transductions. Positions of $G(\mathcal{A}, M)$ (cf. Section 4.4) are subterms of M together with some information of a bounded size: a state of $\mathcal{A}$, an environment ρ assigning residuals to λ-variables, and a stack of residuals. We make it precise in the following lemma.

Lemma 20 There is a number, call it *maxpos*, depending only on Σ, $\mathcal{T}$ and $\mathcal{X}$ such that for every subterm N of a term $M \in \mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$, the number of positions in $G(\mathcal{A}, M)$ of the form $q : (N, \rho, S)$ is bounded by *maxpos*.

Proof
First observe that the number of residuals of a given type is finite. A residual of type 0 is a subset of $Q \times [d]$ where Q is the set of states of $\mathcal{A}$ and d is the maximal value of the rank function of $\mathcal{A}$. A residual of type $\alpha \to \beta$ is a function from residuals of type α to residuals of type β. Recall that ρ is a function assigning residuals to free λ-variables of N. So the number of possible ρ is bounded since λ-variables come from a fixed finite set $\mathcal{X}$, and the type of the residual assigned to a λ-variable is determined by the type of the λ-variable. For the stack S, we know that the type of N determines the type of S in the sense that it determines the length of the sequence S and the type of each element of the sequence. Since the type of N belongs to the finite set of types $\mathcal{T}$ we have that the number of possible stacks depends on $\mathcal{T}$ and not on N. □

We will argue that it is possible to define $G(\mathcal{A}, M)$ inside $\mathit{Graph}(M)$ by means of formulas of MSOL. In other words, that for a fixed $\mathcal{A}$, the mapping from M to $G(\mathcal{A}, M)$ is a monadic second-order transduction. Having done this, the desired formula $\widehat{\varphi}$ will come from the inverse image determined by this transduction of the formula defining games where Eve has a winning strategy.

First observe that the game $G(\mathcal{A}, M)$ can be represented with a structure over a signature depending only on $\mathcal{A}$. To represent $G(\mathcal{A}, M)$, we need a predicate determining the transitions in the game, a predicate distinguishing the positions for Eve, and a predicate for every rank in order to encode the parity condition. So the signature depends only on ranks and these come from $\mathcal{A}$. We will write $G(\mathcal{A}, M) \vDash \psi$ to mean that ψ holds in the structure representing $G(\mathcal{A}, M)$. Recall that for every fixed set of ranks, there is an MSOL formula defining the set of parity games where Eve has a winning strategy. Let γ_{win} be such a formula for the set of ranks determined by $\mathcal{A}$. We have:

$$G(\mathcal{A}, M) \vDash \gamma_{win} \quad \text{iff} \qquad \text{Eve wins in } G(\mathcal{A}, M) \tag{6}$$

In order to define $G(\mathcal{A}, M)$ inside $\mathit{Graph}(M)$, we will need a transduction that duplicates a given structure a certain number of times. Take a natural number k and let $[k]$ stand for $\{1, \ldots, k\}$. Consider a structure $\mathcal{M} = \langle V, E_1, E_2, P_1, \ldots, P_l \rangle$, in our case it will be a graph of a term. A k-fold duplication of $\mathcal{M}$ is a structure $\mathcal{M} \times [k] = \langle V \times [k], E_1, E_2, eq, P_1, \ldots, P_l, C_1, \ldots, C_k \rangle$ whose elements are pairs $(v, i) \in V \times [k]$; we think of (v, i) as of v in the i-th copy. The relations E_1, E_2 are as in $\mathcal{M}$, that is they hold between elements

of the same copy: $E_1((v,i),(v',j))$ iff $E_1(v,v')$ and $i = j$. The eq predicate says that two elements are copies of the same element: $eq((v,i),(v',j))$ iff $v = v'$. Predicates $P_1, \ldots, P_l$ are as in $\mathcal{M}$: $P_i(v,j)$ if $P_i(v)$. Finally, predicate C_i holds for all elements of copy i: $C_i(v,j)$ iff $i = j$.

It is well-known [Cou94] that for every k and MSOL formula ψ there is a formula $\psi/[k]$ such that for every structure $\mathcal{M}$:

$$\mathcal{M} \times [k] \models \psi \qquad \text{iff} \qquad \mathcal{M} \models \psi/[k] \tag{7}$$

Let M be a term and $maxpos$ the constant from the Lemma 20. We will show how to define $G(\mathcal{A}, M)$ in $Graph(M) \times [maxpos]$. A configuration of $G(\mathcal{A}, M)$ has a form $q : (N, \rho, S)$. We will think of it as consisting of a term N and a context $q : (\cdot, \rho, S)$. To every possible context $q : (\cdot, \rho, S)$ we can associate a number from $[maxpos]$. So a node (v,i) in $Graph(M) \times [maxpos]$ is a configuration $q : (N_v, \rho, S)$ where N_v is the term rooted in v and $q : (\cdot, \rho, S)$ is the context number i. We write $C_{q:(\cdot,\rho,S)}$ for C_i where i is the number of the context. With this convention one can directly express the transitions of the game $G(\mathcal{A}, M)$ with MSOL formulas over $\mathcal{M} \times [k]$. We give two examples. For all N the transitions :

$$q : (\lambda x.N, \rho, R \cdot S) \to q : (N, \rho[x \mapsto R], S)$$

are defined by

$$next_\lambda(z,z') \equiv C_{q:(\cdot,\rho,R\cdot S)}(z) \wedge P_{\lambda x}(z) \wedge \exists z''.\ E_1(z,z'') \wedge eq(z'',z') \wedge C_{q(\cdot,\rho[x\mapsto R],S)}(z')$$

The transitions on Y-variables are defined by

$$next_{\Uparrow}(z,z') \equiv C_{q:(\cdot,\rho,S)}(z) \wedge P_{\Uparrow}(z) \wedge \exists z''.E_1(z,z'') \wedge eq(z'',z') \wedge C_{q:(\cdot,\emptyset,S)}$$

Observe that the formulas depend on $\mathcal{A}$ but not on M. These formulas define $G(\mathcal{A}, M)$ inside $Graph(M) \times [maxpos]$. We obtain that for every formula ψ there is a formula ψ^{int} such that for all M:

$$G(\mathcal{A}, M) \models \psi \qquad \text{iff} \qquad Graph(M) \times [k] \models \psi^{int}. \tag{8}$$

We now have all ingredients to prove the following lemma.

Lemma 21 For a fixed $\mathcal{A}$, $\mathcal{T}$ and $\mathcal{X}$. There is a formula $\widehat{\varphi}_{\mathcal{A}}$ such that for every term $M \in CTerms(\Sigma, \mathcal{T}, \mathcal{X})$

$$\text{Even wins in } G(\mathcal{A}, M) \quad \text{iff} \quad Graph(M) \models \widehat{\varphi}_{\mathcal{A}}$$

Indeed it is enough to use the properties from (6), (7), (8). The formula $\varphi_{\mathcal{A}}$ from the lemma is $(\gamma^{int}_{win})/[maxpos]$.

Recall that the automaton $\mathcal{A}$ was supposed to be equivalent over trees to our initial formula φ. If M is a concrete canonical term then Proposition 2 tells us that $BT(M) \vDash \varphi$ iff Eve wins in $\mathcal{K}(\mathcal{A}, M)$. By Proposition 3, the later condition is equivalent to Eve winning in $G(\mathcal{A}, M)$. The lemma above says that this is equivalent to $\mathit{Graph}(M) \vDash \widehat{\varphi}_{\mathcal{A}}$. Thus we can take $\widehat{\varphi}_{\mathcal{A}}$ as our formula $\widehat{\varphi}$ in Theorem 12. Observe that this formula does not depend on M.

The last step in the proof is to remove the hypothesis of M being concrete canonical. The translation presented in Section 2.3 is actually an MSOL transduction. So we have that for every formula ψ there is a formula ψ^C such that for every term M:

$$\mathit{Can}(M) \vDash \psi \qquad \text{iff} \qquad M \vDash \psi^C$$

Putting this together we get that for if we start with M not in canonical form we should take $\widehat{\varphi}^C$.

5 Consequences of the transfer theorem for evaluation

The objective of this short section is to present several corollaries of the transfer theorem. The first consequence is the decidability of the MSOL theory of $BT(M)$ for every finite term M of λY-calculus. This gives another proof of the result of Ong [Ong06], since for every higher-order recursive scheme $\mathcal{F}$ there is a finite term M such that $BT(M)$ is the tree generated by $\mathcal{F}$ (see[SW12]).

Corollary 22 For every finite term M of λY-calculus, the MSOL theory of $BT(M)$ is decidable.

The part of the strength of the transfer theorem comes from the fact that the formula $\widehat{\varphi}$ does not depend on M but just on Σ, $\mathcal{T}$ and $\mathcal{X}$. This has an immediate consequence that once Σ, $\mathcal{T}$ and $\mathcal{X}$ are fixed, every MSOL property of Böhm trees is an MSOL property of terms. For example, since the set of finite trees is MSOL definable, the set of terms from $\mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ having a normal form, i.e. that reduce to a finite term in a normal form, is also MSOL definable.

Decidability of higher-order matching restricted to $\mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ without fixpoint operators follows also by the same type of reasoning. The problem is stated as follows. Given a finite term M, and a closed finite term K with no occurrence of Y: can one replace free variables of M with terms in such a way that the result is β-equal to K.

Corollary 23 Fix $\Sigma, \mathcal{T}, \mathcal{X}$. For every pair of terms $M, K \in \mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ such that K is closed and does not have fixpoint operators. It is decidable if there is a substitution σ such that $M\sigma =_\beta K$.

Let us briefly sketch the argument. For a term M we will write $\mathit{shape}(M)$ for an MSOL formula defining the set of terms that can be obtained from M by substitutions. This formula just states that the top of the tree is the term M, but in places where M has free variables the tree is arbitrary, but without Y binders. Since K is a term without Y binder, it has a normal form K' that is a finite term. Since K' does not have free variables, $\mathit{shape}(K)$ defines exactly one term. Suppose that $\mathit{shape}(K)$ is our formula φ and take $\widehat{\varphi}$ given by the theorem. Consider the formula

$$\mathit{shape}(M) \wedge \widehat{\varphi}$$

We have that a term satisfying this formula is obtained from M by a substitution, and that its normal form is K'. So the higher-order matching problem for M and K reduces to satisfiability of this formula. Observe that in principle the substitution can use infinite terms. Nevertheless, recall that if there is a tree satisfying an MSOL formula then there is a regular one. This means that if there is a solution to a matching problem then there is one that is a finite term. Here we allowed both M and σ to use fixpoint operators, we can disallow this by adding one more conjunct to the formula above. The decidability of the case without fixpoints has already been shown by Schmidt-Schauß [SS03] (see also[Sal09]).

Next, we would like to present a kind of synthesis result. The task is to construct a program satisfying a given specification from a given finite set of modules. The specification is given by an MSOL formula φ. Every module is a finite λY-term.

Corollary 24 Given a formula φ and finite set of closed λY-terms $M_1 \dots, M_l$. It is decidable if there is a closed term K constructed from $M_1 \dots, M_l$ by means of application, Y-variables and Y-binders, such that $BT(K) \vDash \varphi$. If there is a such a term then there exists a finite one.

The requirement that K does not use constants from the signature is essential, since otherwise we could get a trivial solution ignoring the modules. For the proof of the corollary consider terms of the form $(\lambda x_1 \dots x_l.N)M_1 \dots M_l$ where N is constructed only with applications, Y-variables and Y-binders. Indeed, after reducing this term l-times we get a term as required in the corollary. So the restriction on the shape of K can be expressed by a formula $\mathit{shape}((\lambda x_1 \dots x_l.z)M_1 \dots M_l) \wedge \gamma$, where γ is a formula saying that the only labels appearing in the subtree starting in the node corresponding to z are variables $x_1, \dots, x_l$, application, Y-variables, and fixpoint binders. Summing-up, the solution to the problem states in the corollary is to decide

the satisfiability of the formula

$$\widehat{\varphi} \wedge \mathit{shape}((\lambda x_1 \dots x_l.z)M_1 \dots M_l) \wedge \gamma \ .$$

This formula asks for a term of the particular shape such that when evaluated gives a Böhm tree satisfying $\widehat{\varphi}$. Once again, if there is a solution then there is a regular solution, that is a finite term.

6 Conclusion

We have shown that every MSOL property of Böhm trees is effectively an MSOL property of terms. The possibility that such a transfer theorem may hold has been indicated by a number of results in the literature: Ong's Theorem [Ong06] stating that the MSOL properties of Böhm trees of λY-terms is decidable; Courcelle and Knapik's [CK02b] result showing a similar theorem for a variant of evaluation of first-order terms; and finally, a result of Salvati [Sal09] demonstrating that in the context of λ-calculus, recognizability is preserved under inverse homomorphism.

The translation we propose for going form MSOL properties of Böhm trees to MSOL properties of λY-terms not only works for infinite terms but also depends on very few properties of terms themselves. We just need to fix a finite set of λ-variables that a term can use as well as a finite set of types that the subterms of a term may have. Apart from this, the translation does not depend on any other structural properties of terms. This is rather surprising as the set of terms that use a fixed number of λ-variables does not form a set that is closed under $\beta\delta$-reduction. For this reason the method of Courcelle and Knapik could not be extended to higher-order types, since due to α-conversion intermediate results of evaluation could need an unbounded number of λ-variables. The invariants on reduction that we express with the Krivine machine and residuals overcome this difficulty.

The obvious question is the relation of our transfer theorem concerning $\beta\delta$-reduction to two other transfer results concerning unfolding operation [CW98a] and for Muchnik iteration [Sem84b, Wal02a], respectively. Recall that while Muchnik iteration is strictly stronger than unfolding, the two operations can be used to define the classes of trees forming so called pushdown hierarchy [CW03].

We cannot compare directly the three results since the transfer theorems for unfolding [CW98b] and Muchnik iteration [Sem84a, Wal02b] work on graphs, while our theorem works with λY-terms that seen as graphs are trees with back edges. For this reason we cannot say that our result implies the other two. Yet, if restricted to trees with back edges, the result on unfolding is a special case of our result: the unfolding can be simulated by δ-reduction. We do not know if it is possible to simulate Muchnik iteration

directly using $\beta\delta$-reduction. Nevertheless, it is well-known that, up to simple MSOL-interpretations, every tree in the pushdown hierarchy is a result of a $\beta\delta$-reduction of a finite term. Hence, similarly to Muchnik's iteration, $\beta\delta$-reduction allows to obtain all trees in the pushdown hierarchy [Cau02]. Moreover, thanks to the recent result of Parys [Par12], we know that $\beta\delta$-reduction can give trees that do not belong to the pushdown hierarchy. This shows that our transfer theorem is not a consequence of the Muchnik's theorem. The study of relations between Muchnik iteration and $\beta\delta$-reduction, as well as constructing new hierarchies using the new transfer theorem, are interesting directions for further research.

References

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

[Bar84] H. Barendregt. *The Lambda Calculus, Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.

[BCHS12] Christopher H. Broadbent, Arnaud Carayol, Matthew Hague, and Olivier Serre. A saturation method for collapsible pushdown systems. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *ICALP (2)*, volume 7392 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 2012.

[Cau02] Didier Caucal. On infinite terms having a decidable monadic theory. In *MFCS'02*, volume 2420 of *LNCS*, pages 165–176, 2002.

[CK02a] Bruno Courcelle and Teodor Knapik. The evaluation of first-order substitution is monadic second-order compatible. *Theoretical Computer Science*, 281:177–206, 2002. Special issue offered to Maurice Nivat.

[CK02b] Bruno Courcelle and Teodor Knapik. The evaluation of first-order substitution is monadic second-order compatible. *Theor. Comput. Sci.*, 281(1-2):177–206, 2002.

[Cou94] Bruno Courcelle. Monadic second-order graph transductions: A survey. *Theoretical Computer Science*, 126:53–75, 1994.

[CW98a] Bruno Courcelle and Igor Walukiewicz. Monadic second-order logic, graph coverings and unfoldings of transition systems. *Ann. Pure Appl. Logic*, 92(1):35–62, 1998.

[CW98b] Bruno Courcelle and Igor Walukiewicz. Monadic second-order logic, graphs and unfoldings of transition systems. *Annals of Pure and Applied Logic*, 92:35–62, 1998.

[CW03] Arnaud Carayol and Stefan Wöhrle. The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2003.

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

[ES77] J. Engelfriet and E. M. Schmidt. IO and OI. I. *Journal of computer and system sciences*, 15:328–353, 1977.

[ES78] J. Engelfriet and E. M. Schnidt. IO and OI. II. *Journal of computer and system sciences*, 16:67–99, 1978.

[Fis68] M. J. Fischer. *Grammars with macro-like productions.* PhD thesis, Harvard University, 1968.

[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.

[KdV03] Richard Kennaway and Fer-Jan de Vries. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*, chapter 12, pages 668–711. Cambridge University Press, 2003.

[KKSdV97] Richard Kennaway, Jan Willem Klop, M. Ronan Sleep, and Fer-Jan de Vries. Infinitary lambda calculus. *Theor. Comput. Sci.*, 175(1):93–125, 1997.

[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. Urzycyzn, 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.

[KO11] Naoki Kobayashi and C.-H. Luke Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. *Logical Methods in Computer Science*, 7(4), 2011.

[Kob09a] N. Kobayashi. Higher-order program verification and language-based security. In *ASIAN*, volume 5913 of *LNCS*, pages 17–23. Springer, 2009.

[Kob09b] N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL*, pages 416–428. ACM, 2009.

[Kob09c] N. Kobayashi. Types and recursion schemes for higher-order program verification. In *APLAS*, volume 5904 of *LNCS*, pages 2–3, 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.

[OR11] C.-H. Luke Ong and Steven James Ramsay. Verifying higher-order functional programs with pattern-matching algebraic data types. In Thomas Ball and Mooly Sagiv, editors, *POPL*, pages 587–598. ACM, 2011.

[Par12] Pawel Parys. On the significance of the collapse operation. In *LICS*, pages 521–530. IEEE, 2012.

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

[Sal09] Sylvain Salvati. Recognizability in the simply typed lambda-calculus. In Hiroakira Ono, Makoto Kanazawa, and Ruy J. G. B. de Queiroz, editors, *WoLLIC*, volume 5514 of *Lecture Notes in Computer Science*, pages 48–60. Springer, 2009.

[Sem84a] A.L. Semenov. Decidability of monadic theories. In *MFCS'84*, volume 176 of *LNCS*, pages 162–175, 1984.

[Sem84b] Alexei L. Semenov. Decidability of monadic theories. In Michal Chytil and Václav Koubek, editors, *MFCS*, volume 176 of *Lecture Notes in Computer Science*, pages 162–175. Springer, 1984.

[SS03] Manfred Schmidt-Schauß. Decidability of arity-bounded higher-order matching. In Franz Baader, editor, *CADE*, volume 2741 of *Lecture Notes in Computer Science*, pages 488–502. Springer, 2003.

[Sta04] Richard Statman. On the λy calculus. *Annals of Pure and Applied Logic*, 130(1-3):325–337, 2004.

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

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

[Wal02a] Igor Walukiewicz. Monadic second-order logic on tree-like structures. *Theor. Comput. Sci.*, 275(1-2):311–346, 2002.

[Wal02b] Igor Walukiewicz. Monadic second order logic on tree-like structures. *Theoretical Computer Science*, 257(1–2):311–346, 2002.