Evaluation is MSOL-compatible*

Sylvain Salvati[1] **and Igor Walukiewicz**[2]

1 **LaBRI, INRIA / Université Bordeaux, France**
2 **LaBRI, CNRS / Université Bordeaux, 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 behavioral properties of terms. Another application is decidability of a control-flow synthesis problem.

1998 ACM Subject Classification F.3 Logics and Meanings of Programs

Keywords and phrases Simply typed λY-calculus, monadic second order logic, Transfer Theorem, infinitary systems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.103

1 Introduction

Rice's theorem tells us that no non-trivial property of the behavior 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 behavior of a λY-term is its Böhm tree. Since not all λY-terms have normal forms, Böhm trees are a standard choice for the result of a 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 if we fix a finite set of term variables then 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 the functional language community. It can be seen as a simplification of the programming language PCF introduced by Plotkin [37]. By now, it is usual [3] to approach the semantics of the lambda-calculus through infinite Böhm trees. Such a tree is just the 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. We assume that all constants in the vocabulary have a type of order at most 2. This assumption is made in all the works considering language theoretic properties of Böhm trees, as it guarantees that a Böhm tree is just a labeled ranked tree. In this paper we consider also infinite λY-terms. This is less standard but introduces relatively few complications while substantially strengthening of the main theorem.

* Supported by ANR 2010 BLAN 0202 01 FREC

Our Transfer Theorem says that for a fixed finite vocabulary of terms, an MSOL formula φ can be effectively transformed 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. In terminology of Courcelle [12], our result says that evaluation of λY-terms is MSOL-compatible. A flagship example of a result of this kind is MSOL-compatibility of the unfolding operation [43, 14]. A finite automaton, i.e. a graph, can be unfolded into a 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 [43, 45] allows to get a similar result for pushdown automata and higher-order pushdown automata. This way we obtain the pushdown hierarchy of trees with a 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 [11] is a Böhm tree of some term, but not vice versa [36].

Related work: Under a different syntax λY-calculus has been intensively studied by the language theoretic community. One can cite the PhD thesis of Fischer [18] on macro languages, the work of Engelfriet and Schmidt on IO and OI [16, 17], or the work of Damm on (safe) recursive schemes [15]. More recently Knapik, Niwinski and Urzyczyn [21] considered recursive schemes as generators of infinite trees, and studied the model-checking problem for such trees. After a series of intermediate results [2, 22, 1]; Ong [31] 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 the corresponding terms of λY-calculus.

The Transfer Theorem for evaluation presented here is stronger than Ong's theorem in at least two aspects. First, it holds also for (possibly irregular) infinite λY-terms. Second, and more importantly, the theorem gives an effective way of expressing properties of results of evaluation (Böhm trees) in terms of properties of terms: the construction of the formula $\widehat{\varphi}$ depends on φ and a fixed vocabulary, but does not depend on a term M. 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. We show that removing the restriction to a fixed finite vocabulary would imply the collapse of the polynomial hierarchy. In the last section of the paper we give several other applications of additional power provided by the Transfer Theorem.

This work sheds some light on unfolding and Muchnik's theorems. It shows that they are in some sense special cases of the Transfer Theorem for evaluation. Knapik and Courcelle [13] have used the unfolding theorem to prove a special case of our theorem for infinite terms with variables only of type 0. They also mention that the method can be extended to all safe terms thus avoiding the use of Muchnik's theorem. Due to a result of Parys [36] we know that the approach based on unfoldings or Muchnik's theorem cannot work for all (unsafe) terms. Our proof proposes a different approach to substitution offered by the Krivine machine and that overcomes this difficulty (cf. Section 4).

MSOL properties of higher-order systems is an active area of research. After the seminal paper of Knapik, Niwinski and Urzyczyn[21]; Caucal has introduced the pushdown hierarchy [11] that since has been an object of intensive study from the logical point of view [10]. Many interesting properties of higher-order programs can be analyzed with recursive schemes and automata [24, 23, 25, 27, 33]. The decidability result of Ong has been revisited in a number of ways [19, 26, 39, 8].

The study of MSOL-compatible operations has a long history. Directly relevant here is an iteration operator proposed by Stupp; he showed that it is MSOL-compatible [44]. Rabin's theorem [38] on decidability of MSOL on infinite binary trees is an immediate corollary of this result. Much later Muchnik has proposed a strengthening of Stupp's iteration, and sketched the proof of MSOL-compatibility of this operation [43]. Courcelle and Walukiewicz have shown MSOL-compatibility of the unfolding operation [14] not using Muchnik iteration. Later Walukiewicz has given a detailed proof of MSOL-compatibility of Muchnik iteration [45]. This result has then been extended and adapted to other logics [6, 30, 29]. For a survey of compatible operations and their applications we refer the reader to [5].

Plan of the paper: In the next section we introduce infinitary λY-calculus: a simply typed λ-calculus of infinite terms with fixpoint operators. We define Böhm trees and show that, as for finite terms, they still define a standard notion of values.

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 is not fixed. We also compare our theorem to that of Ong [31]. An overview of proof methods is presented in Section 4.

Section 5 gives three applications of the Transfer Theorem. In the conclusion section we give more relations between the Transfer Theorem and other results in the literature. The extended version of the paper [40], presents all the proofs as well as some more comments and discussions.

2 Infinitary λY-calculus

In this paper, we work with the infinitary simply typed λ-calculus with fixpoints. We start with the notions of tree signatures, and of infinitary terms together with the operational semantics of the infinitary λY-calculus. We adopt a slightly modified syntax where Y, the fixpoint operator, 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 little importance for the calculus itself, but it will allow us to get a more general theorem later when we put restrictions on the number of λ-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. Moreover, a Böhm tree of a term of type 0 over a tree signature is just a ranked tree.

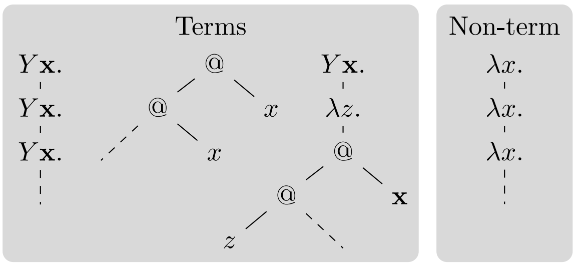
Syntax and operational semantics. The *set of types* is constructed from a unique *basic type* 0 using a binary operation $\to$. Thus 0 is a type and if α, β are types, so is $(\alpha \to \beta)$. As usual, so as to use fewer parentheses, we consider that $\to$ associates to the right. For example, $0 \to 0 \to 0$ 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 (in general noted c^α, $\dots$), 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 our results do not depend on this convention.

Terms will be built over two disjoint countable sets of typed variables: λ-variables and Y-variables: written x^α and $\mathbf{x}^\alpha$ respectively. 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 labeled by Ω^α, x^α, $\mathbf{x}^\alpha$, or c^α is a term of type α.
- A tree with the root labeled $@^\beta$ 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 labeled $\lambda^{\alpha\to\beta}x^\alpha$ with the unique immediate subtree being a term of type β, is a term of type $\alpha \to \beta$.
- A tree with a root labeled $Y^\alpha\mathbf{x}^\alpha$ 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 the Figure to the right. Notice that all variables and constructors carry type labelings that make typings of terms 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 $N_0N_1\ldots N_p$ for $(\ldots(N_0N_1)\ldots N_p)$.



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) as the term $\mathit{rec}(\lambda x.N[x/\mathbf{x}])$ where $\mathit{rec} = \lambda f.f(f(f(f\ldots)))$. This shows that λ-abstraction, or parameter instantiation, is more powerful than recursive definition in the context of the infinitary λ-calculus.

The notion of capture-avoiding substitution can be easily extended to infinitary λ-calculus. Given two terms M and N we shall write $M[N/x]$ (*resp.* $M[N/\mathbf{x}]$) 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 $M[N/x]$ or of $M[N/\mathbf{x}]$ may require an infinite amount of work. This is why, following a suggestion made in Terese [20], our technique is based on the Krivine machine which computes with explicit substitutions and avoids this infinite overhead.

We may now define β-contraction on infinitary terms as the direct 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 $M[Y\mathbf{x}.M/\mathbf{x}]$. 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.

We recall the notion of weak head normal form and of weak head reduction: the reduction strategy of the Krivine machine. 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 $(Y\mathbf{x}.\ P)P_1 \dots P_n$, then M has a *head-redex*. The operation of reducing M to $P[P_1/x]P_2 \dots P_n$ or to $P[Y\mathbf{x}.P/\mathbf{x}]P_1 \dots P_n$, 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.

Böhm trees. We now define a notion of *normal form* for infinitary terms. There is no particular difficulty to do it 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. 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 1** (Böhm trees). Given a closed term M of type 0 over a 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 labeled 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 labeled 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.

The reader may be surprised that we talk about Böhm trees while in general weak head reduction computes 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.

3 Transfer Theorem for evaluation

In this section we present the main theorem of the paper. The theorem relates 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 also 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.

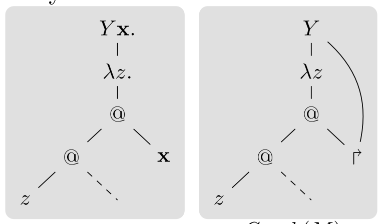
Terms will be represented as labeled, potentially infinite, graphs. For us here a labeled graph is a structure of the form $\mathcal{M} = \langle V, E_1, E_2, \{P_i\}_{i=1,\dots,n}\rangle$, where V is the set of vertices, E_1 and E_2 are binary relations on vertices, and every P_i is a subset of vertices. We will have two edge relations representing left and right successors. In our case predicates P_i will form 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, for simplicity of the 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 $\mathcal{X} = \{x_1^{\alpha_1}, \dots, x_k^{\alpha_k}\}$, and a finite set of types $\mathcal{T}$. We denote by *Terms*$(\Sigma, \mathcal{T}, \mathcal{X})$

the set of *infinite closed concrete terms* 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 call *concrete terms*, terms that are not considered up to α-conversion, so it makes sense to say that bound λ-variables in M should come from $\mathcal{X}$. Observe that 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 $\mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ is a labeled tree where the labels come from a 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 labeled by a Y-variable $\mathbf{x}^\alpha$ we put an E_1-edge from the node to the node labeled $Y^\alpha\mathbf{x}^\alpha$. Since M is closed, such a node exists and is an ancestor of the node labeled 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 labeled with a Y-variable of type α, we change its label to $\Rsh^\alpha$. Finally, we replace all labels of the form $Y^\alpha\mathbf{x}^\alpha$ 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 labeled tree but as a labeled graph. Let us denote it by $\mathit{Graph}(M)$. Observe that the nodes of this graph have labels from a finite set, call it $\mathit{Talph}(\Sigma, \mathcal{T}, \mathcal{X})$. There are two edge relations, E_1 and E_2, in $\mathit{Graph}(M)$ since nodes labeled by application symbol have both left and right successors. The nodes with other labels have either one successor, given by E_1, or no successor (nodes labeled with labels from $\Sigma \cup \mathcal{X}$).



M $\quad$ $\mathit{Graph}(M)$

Since $\mathit{Graph}(M)$ is a labeled graph over a finite alphabet, it makes sense to talk about satisfiability of an MSOL formula in this graph. We will just write $M \vDash \varphi$ instead of $\mathit{Graph}(M) \vDash \varphi$. 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 predicates P_b form the partition of the set of vertices and then express conditions from the definition of infinite λY-terms on page 106. 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 labeled by symbols from Σ. Hence one can talk about satisfiability of MSOL formulas in $BT(M)$. The Transfer Theorem says that evaluation, that is the function assigning to a term its Böhm tree, is MSOL compatible.

▶ **Theorem 2** (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:*

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

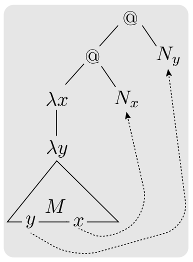
Notice that the formula $\widehat{\varphi}$ is independent from M; if it were dependent on M then we would simply obtain Ong's theorem since we could take $\widehat{\varphi}$ to be either *true* or *false*. At first sight the proof presented in [32] resembles our Transfer Theorem: for a recursive scheme G it constructs a tree $\lambda(G)$ and a property automaton that works on trees of the form $\lambda(G)$. Looking closer (definition on page 11 of [32]) the alphabet of $\lambda(G)$ depends on the size of G, since terms are η-expanded, and all bound variables are supposed to be different. So when fixing the alphabet of the property automaton, one fixes the number of rules in the recursive scheme (except maybe for rules for terminals of type 0). In our case the formula $\widehat{\varphi}$ is constructed for an infinite family of terms provided they use only the lambda-variables from $\mathcal{X}$. For completeness of this comparison we should add that Ong states that his approach can be straightforwardly adapted to give the proof of the Transfer Theorem [35]. Ong also observes that this adaptation would lower the complexity of his algorithm to match the one from [26]. In Section 5 we show that, a variant of global model-checking [9, 7] which was so far considered as a genuine extension of Ong's Theorem follows directly from the Transfer Theorem.

A natural question to ask is whether our restrictions on terms can be further weakened. Observe that the restriction on the fixed finite set of types is essential to be able to define in MSOL the set of representations of well typed terms. Concerning the restriction on the λ-variables, in principle it is possible to represent terms with an unbounded number of λ-variables by using the same trick for λ-binders as the one we have used for the Y-binders. 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.

Using a fixed set of types, it is possible to represent a Quantified Boolean Formula (QBF) θ by a λ-term M_θ whose Böhm tree is just a constant *true* iff θ is true. Moreover M_θ can be obtained in linear time from θ. Suppose that the Transfer Theorem would hold without restricting the number of λ-variables. There would then be a $\widehat{\varphi}$ so that for every QBF θ, $M_\theta \vDash \widehat{\varphi}$ iff θ is true. But verifying a fixed MSOL formula against some finite structure is in the polynomial hierarchy, while the validity of QBF formulas is PSPACE-complete. Thus, this more general formulation of the Transfer Theorem would imply the unlikely conclusion that the polynomial hierarchy collapses.

4 Overview of the proof

Our objective is to reduce the monadic theory of $BT(M)$ to the monadic theory of M, and moreover to do this in a smooth fashion depending only on the vocabulary of M. An instructive sub-case of terms of order ≤ 2 has been considered by Courcelle and Knapik [13]. They consider infinite closed terms of type 0 over a finite set of variables of type 0. Moreover, although it is somehow implicit in their formulation, they assume that every subterm has a type from some fixed finite set of types. In this case they are able to show the Transfer Theorem using the compatibility of MSOL with respect to the unfolding operation.



To explain the method proposed by Courcelle and Knapik let us look at the term $(\lambda x.\lambda y.M)N_xN_y$. This term is represented as a tree the same way as described in the previous section. In the figure to the right we have singled out occurrences of x and y in M. It turns out that it is possible in MSOL to define links between bound variables and the terms that would be substituted for these variables if β-reduction were performed (dashed arrows in the picture). For this the two assumptions on the number of variables and the number of types stated above are essential. Then, intuitively, the normal form of the term can be obtained by following the links and jumping over lambda-binders. So the normal form of the term can be defined in the unfolding of the term with links. Thanks to the compatibility of the unfolding, every MSOL formula over the unfolding can be translated into a formula over a term itself.

This method does not work when there are bound variables of type other than 0. The

$$\begin{tikzcd}
\varphi \arrow[d, mapsto] & BT(M) \vDash \varphi \arrow[d, phantom, "\text{iff}"] & & M \vDash F^{-1}(\gamma_{win}) \arrow[d, phantom, "\text{iff}"] & M \arrow[d, mapsto, "F"] \\
\mathcal{A} & BT(M) \in L(\mathcal{A}) \arrow[d, phantom, "\text{iff}"] & & G(\mathcal{A}, M) \vDash \gamma_{win} \arrow[d, phantom, "\text{iff}"] & G(\mathcal{A}, M) \\
& \text{Eve wins in } \mathcal{K}(\mathcal{A}, M) \arrow[rr, phantom, "\text{iff}"] & & \text{Eve wins in } G(\mathcal{A}, M) &
\end{tikzcd}$$

Figure 1 Schema of the main part of the proof.

reason is that when reducing a redex of a variable, say of a type $0 \to 0$, a new redex can be created. Moreover, in order to avoid variable capture, variable renaming may be needed. In consequence, after doing the operation described in the previous paragraph we can get a term with much more, or even infinitely many variables. This phenomenon would not happen if we started from an infinite safe term [21, 4] since no variable renaming would be needed when reducing such a term. So one can use the method above to reduce simultaneously all the redexes of the highest order in a safe term. This would give a β-equivalent safe term over the same set of variables but with the smaller maximal order of a redex. Hence one can repeat the argument until all the redexes are eliminated. In consequence, this gives yet another method to show the decidability of MSOL theory of Böhm trees generated by safe terms, that is all the trees in the pushdown hierarchy. As noted in the introduction this method cannot be extended to all infinite simply-typed terms. The Krivine machine [28] provides a solution by deferring substitutions till "the last moment". The issues discussed here are related to β-reduction, the fixpoints do not enter into the picture. Indeed, as remarked on page 106, if we admit infinite terms then fixpoints can be simply eliminated. So the core of the Transfer Theorem is essentially about β-reduction in infinite terms.

The proof of Transfer Theorem is presented in the full version of the paper. Its schema is presented in Figure 1. It simplifies the exposition to think that all constants are binary, so that $BT(M)$ is a binary tree. There is a non-deterministic parity automaton $\mathcal{A}$ on infinite binary trees that recognizes trees that are models of φ. For a term M we define a game $\mathcal{K}(\mathcal{A}, M)$. This game is presented in terms of configurations of a Krivine machine. Eve wins in $\mathcal{K}(\mathcal{A}, M)$ iff $BT(M)$ is accepted by $\mathcal{A}$. An important step is to construct a reduced game $G(\mathcal{A}, M)$ with the property that Eve wins in the later game iff she wins in $\mathcal{K}(\mathcal{A}, M)$. For this we use Krivine machines in a similar way as in [39] the difference being that now we need to handle infinite terms. According to [35] the reduction from "$BT(M) \in L(\mathcal{A})$" to some equivalent of $\mathcal{G}(\mathcal{A}, M)$ can be also done by adapting the methods from [31] and [34]. Finally, and this is another important step, we 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 $\widehat{\varphi}$. Actually this schema works only for terms some special form that we call canonical. To complete the proof we show that the translation of M into a canonical form is an MSOL transduction [12].

5 Consequences of the Transfer Theorem for evaluation

The objective of this section is to present several corollaries of the Transfer Theorem. The first concerns the so-called global model-checking problem. For a given finite term M and a formula φ we want to find a finite representation of the set of nodes in the Böhm tree of M where the formula holds. For this we need to have a way of representing subtrees of the Böhm tree. One simple method is to just use the term obtained by the reduction sequence producing that subtree. This method gives an effective way of representing every node v of $BT(M)$ by a term N_v from $\mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ (where $\mathcal{T}$ is the set of types of subterms of M, and $\mathcal{X}$ is the set variables appearing in M) so that $BT(N_v)$ is the subtree of $BT(M)$ rooted at node v. The Transfer Theorem gives immediately the following corollary:

▶ **Corollary 3.** *Let M be a finite term of λY-calculus and φ an MSOL formula. Nodes of $BT(M)$ can be represented by terms of $\mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ so that $N \vDash \widehat{\varphi}$ iff the tree rooted in the node of $BT(M)$ represented by N satisfies φ.*

Let us explain in more detail how nodes of $BT(M)$ can be represented by terms over the same vocabulary as M. One can observe that a term obtained by a reduction sequence from M can be written as a term with explicit substitutions $(N_0[\sigma_0])(N_1[\sigma_1])\dots(N_k[\sigma_k])$ where $N_0 \dots N_k$ are terms and $\sigma_0, \dots, \sigma_k$ are substitutions ranging over terms with explicit substitutions. The important point of this representation is that the terms $N_0, \dots, N_k$ as well as all the terms appearing in substitutions are subterms of M. One can easily prove this observation directly by induction on the number of reductions. It also follows immediately from the formalization of the reduction with the Krivine machine. Explicit substitution notation can then be eliminated by introducing λ-abstractions. An expansion of a term with substitutions, $E(N[\sigma])$, is defined by induction as $E(N[\sigma]) = (\lambda x_0 \dots x_n.N)E(\sigma(x_0))\dots E(\sigma(x_n))$; where $x_0, \dots, x_n$ are the variables free in N. Hence $E(N_0[\sigma_0])E(N_1[\sigma_1])\dots E(N_k[\sigma_k])$ belongs to $\mathit{Terms}(S, \mathcal{T}, \mathcal{X})$ which allows us to use the Transfer Theorem. Recall that, but for [39], other papers on global model-checking make a detour to collapsible pushdown automata [9, 7].

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: can one substitute terms for the free variables of M so that the result is β-equal to K?

▶ **Corollary 4.** *Fix $\Sigma, \mathcal{T}, \mathcal{X}$. For every pair of finite 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 in $\mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ 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 term M, but in places where M has free variables the tree is arbitrary. Since K is a finite 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 $\mathit{shape}(M) \land \widehat{\varphi}$. 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. So if there is a solution to a matching problem then there is one in finite terms. 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 been shown by Schmidt-Schauß [42].

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 5.** *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 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 would 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, the variables $\{x_1, \dots, x_l\}$, 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 $\mathit{shape}((\lambda x_1 \dots x_l.z)M_1 \dots M_l) \land \gamma$, 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 stated in the corollary is to decide the satisfiability of the formula

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

This formula asks for a term having a particular shape and whose Böhm tree satisfies $\widehat{\varphi}$. Once again, if there is a solution then there is a regular solution, that is a finite term.

6 Conclusions

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 [31] stating that the MSOL properties of Böhm trees of λY-terms is decidable; Courcelle and Knapik's [13] result showing a similar theorem for a variant of evaluation of first-order terms; and finally, a result of Salvati [41] demonstrating that in the context of finite λ-calculus, recognizability is preserved under inverse homomorphism.

The translation we propose for going from 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 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 [14] and Muchnik iteration [43, 45], 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 [11, 10].

We cannot compare directly the three results since the Transfer Theorems for unfolding and Muchnik iteration work on graphs, while our theorem works on 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 Böhm tree of a finite term. Hence, similarly to Muchnik's iteration, $\beta\delta$-reduction allows to obtain all trees in the pushdown hierarchy. Moreover, recent result of Parys [36] shows that there is a term whose Böhm tree does not belong to the pushdown hierarchy. This proves that our Transfer Theorem is not a consequence of 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

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

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

3 H. Barendregt. *The Lambda Calculus*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.

4 William Blum and C.-H. Luke Ong. The safe lambda calculus. *Logical Methods in Computer Science*, 5(1), 2009.

5 Achim Blumensath, Thomas Colcombet, and Christof Löding. Logical theories and compatible operations. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata*, volume 2 of *Texts in Logic and Games*, pages 73–106. AUP, 2008.

6 Achim Blumensath and Stephan Kreutzer. An extension to Muchnik's theorem. *Journal of Logic and Computation*, 13:59–74, 2005.

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

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

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

10 A. Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FSTTCS'03*, volume 2914 of *LNCS*, pages 112–124, 2003.

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

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

13 Bruno Courcelle and Teodor Knapik. The evaluation of first-order substitution is monadic second-order compatible. *TCS*, 281:177–206, 2002.

14 Bruno Courcelle and Igor Walukiewicz. Monadic second-order logic, graphs and unfoldings of transition systems. *Ann. of Pure and Appl. Log.*, 92:35–62, 1998.

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

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

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

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

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

20 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.

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

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

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

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

25 N. Kobayashi. Types and recursion schemes for higher-order program verification. In *APLAS*, volume 5904 of *LNCS*, pages 2–3, 2009.

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

27 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.

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

29 Dietrich Kuske. Compatibility of Shelah and Stupp's and Muchnik's iteration with fragments of monadic second order logic. In *STACS*, volume 1 of *LIPIcs*, pages 467–478, 2008.

30 Dietrich Kuske and Markus Lohrey. Monadic chain logic over iterations and applications to pushdown systems. In *LICS*, pages 91–100, 2006.

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

32 C.-H. L. Ong. On model-checking trees generating by higher-order recursion schemes. Long version available from the author's web page, 2006.

33 C.-H. Luke Ong and Steven James Ramsay. Verifying higher-order functional programs with pattern-matching algebraic data types. In *POPL*, pages 587–598, 2011.

34 Luke Ong. Local computation of beta-reduction by game semantics. Version of February 2013, received by email form the author.

35 Luke Ong. Private communication, February 2013.

36 Pawel Parys. On the significance of the collapse operation. In *LICS'12*, pages 521–530, 2012.

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

38 M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the AMS*, 141:1–23, 1969.

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

40 S. Salvati and I. Walukiewicz. Evaluation is MSOL compatible. Technical report, INRIA-CNRS, 2013. http://hal.inria.fr/hal-00773126.

41 Sylvain Salvati. Recognizability in the simply typed lambda-calculus. In *WoLLIC*, volume 5514 of *LNCS*, pages 48–60, 2009.

42 Manfred Schmidt-Schauß. Decidability of arity-bounded higher-order matching. In *CADE*, volume 2741 of *LNCS*, pages 488–502, 2003.

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

44 Jonathan Stupp. The lattice-model is recursive in the original model. Institute of Mathematics, The Hebrew University, Jerusalem, January 1975.

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