Böhm trees as higher-order recursion schemes

Pierre Clairambault, Andrzej Murawski

Böhm Trees as Higher-Order Recursive Schemes

P. Clairambault[1] and A. S. Murawski[2]

1 **CNRS, ENS de Lyon, Inria, UCBL, Université de Lyon, Laboratoire LIP**
2 **DIMAP and Department of Computer Science, University of Warwick**

Abstract

Higher-order recursive schemes (HORS) are schematic representations of functional programs. They generate possibly infinite ranked labelled trees and, in that respect, are known to be equivalent to a restricted fragment of the λY-calculus consisting of ground-type terms whose free variables have types of the form $o \to \cdots \to o$ (with o being a special case).

In this paper, we show that *any* λY-term (with no restrictions on term type or the types of free variables) can actually be represented by a HORS. More precisely, for any λY-term M, there exists a HORS generating a tree that faithfully represents M's (η-long) Böhm tree. In particular, the HORS captures higher-order binding information contained in the Böhm tree. An analogous result holds for finitary PCF.

As a consequence, we can reduce a variety of problems related to the λY-calculus or finitary PCF to problems concerning higher-order recursive schemes. For instance, Böhm tree equivalence can be reduced to the equivalence problem for HORS. Our results also enable MSO model-checking of Böhm trees, despite the general undecidability of the problem.

1998 ACM Subject Classification F.3.3, F.4.1

Keywords and phrases Lambda calculus, Böhm trees, Recursion Schemes

1 Introduction

Higher-order recursive schemes (HORS) are a class of programming schemes introduced to account for recursive procedures with higher-order parameters [7]. They can be viewed as grammars describing a potentially infinite tree. As tree generating devices, HORS can be identified with a limited fragment of the λY-calculus [19] consisting of ground-type terms whose free variables have types of order at most 1. The free variables play the role of tree constructors. HORS have recently been intensively investigated in connection with program verification. Notably, Ong [16] showed that monadic second-order logic (MSO) is decidable over trees generated by HORS and Kobayashi [12] took advantage of the result to propose a novel approach to the verification of higher-order functional programs.

As already mentioned, HORS are naturally viewed as a fairly small fragment of the λY-calculus: they generate potentially infinite trees, whereas normal forms of arbitrary λY-terms additionally contain variable bindings. This binding information cannot be represented in HORS explicitly. In fact, it turns out that MSO becomes undecidable over Böhm trees of λY-terms, if the binding relation is included in the signature. Nevertheless, as we show in the paper, HORS still allow one to generate faithful representations of (η-long) Böhm trees of λY-terms, where binding is encoded indirectly via *De Bruijn levels* [8]. We also prove an analogous representation theorem for the finitary (finite datatypes) variant PCF_f of PCF [17].

Our results make it possible to recast a variety of problems related to the λY-calculus or finitary PCF as problems concerning higher-order recursive schemes. For example, we obtain a reduction of Böhm tree equivalence in the λY-calculus or PCF_f to the (tree) equivalence problem for HORS. Unfortunately, the latter is currently not known to be


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

decidable and is closely related to the equivalence problem for *deterministic* collapsible pushdown automata [10]. The Böhm tree equivalence problem for PCF_f also has semantic significance, because PCF Böhm trees [2] are concrete representations of the strategies representing the terms in game semantics, and therefore characterize contextual equivalence in PCF with respect to contexts featuring state and control effects.

Other consequences, in the form of decidability results, can be derived by applying Ong's decidability result to representations of Böhm trees obtained through our theorems. In this way, one can show that numerous problems for λY or PCF_f are decidable. Examples include normalizability, finiteness, solvability or having a Böhm tree prefixed by a given finite term. Some of the results are already known, while others appear new.

Thus far, higher-order verification based on HORS focussed on model-checking trees extracted from programs. Our contribution opens the perspective of applying model-checking to terms with binding and arbitrary free variables, such as higher-order components of closed programs.

2 λY-calculus

2.1 Böhm trees of λY-terms

We work with simple types built from a single atom o using the arrow type constructor. They are defined by the grammar given below.

$$\theta ::= o \mid \theta \to \theta$$

The order of a type is defined as follows.

$$ord(o) = 0 \qquad ord(\theta_1 \to \theta_2) = \max(ord(\theta_1) + 1, ord(\theta_2))$$

Terms are considered up to α-equivalence, and equipped with β-reduction and η-expansion (respectively written $\to_\beta$ and $\to_\eta$). We write $\simeq_{\beta\eta}$ for the symmetric reflexive and transitive closure of $\to_\beta$ and $\to_\eta$. We shall consider several extensions of the simply-typed λ-calculus over the types introduced above.

- The $\lambda\bot$-calculus additionally contains a constant $\bot_o : o$. More generally, we shall write $\bot_\theta$ for terms specified as follows.

$$\bot_\theta = \begin{cases} \bot_o & \theta \equiv o \\ \lambda x^{\theta_1}.\bot_{\theta_2} & \theta \equiv \theta_1 \to \theta_2 \end{cases}$$

  For $\lambda\bot$-terms, let us define a partial order $\sqsubseteq$ (relating only terms with equal types) by

$$\frac{}{M \sqsubseteq M} \qquad \frac{}{\bot_o \sqsubseteq M} \qquad \frac{M_1 \sqsubseteq M_2}{\lambda x.M_1 \sqsubseteq \lambda x.M_2} \qquad \frac{M_1 \sqsubseteq M_1' \qquad M_2 \sqsubseteq M_2'}{M_1\ M_2 \sqsubseteq M_2'\ M_2'}.$$

  We will also consider the extension of the $\lambda\bot$-calculus to infinite terms, which we refer to as the $\lambda\bot^\infty$-calculus. The partial order $\sqsubseteq$ can be extended to $\lambda\bot^\infty$-terms to yield an ω-cpo.
- The λY-calculus contains a family of fixed-point operators $Y_\theta : (\theta \to \theta) \to \theta$, where θ ranges over arbitrary types, equipped with the reduction rule $Y_\theta\ M \to_Y M\ (Y_\theta\ M)$.

▸ **Definition 1.** Given a λY-term M and $n \in \mathbb{N}$, the nth approximant of M, written $M \upharpoonright n$, is a $\lambda\bot$-term defined by

$$\begin{array}{rclcrcl} x \upharpoonright n & = & x & & \lambda x.M \upharpoonright n & = & \lambda x.(M \upharpoonright n) \\ M\ N \upharpoonright n & = & (M \upharpoonright n)\ (N \upharpoonright n) & & Y_\theta \upharpoonright n & = & \lambda f^{\theta\to\theta}.f^n(\bot_\theta). \end{array}$$

▸ **Definition 2.** The η-long Böhm tree of a $\lambda\perp$-term $\Gamma \vdash M : \theta$, written $\mathsf{BT}(M)$, is its β-normal η-long form. For a λY-term $\Gamma \vdash M$, the η-long Böhm tree, also denoted by $\mathsf{BT}(M)$, is defined to be $\bigsqcup_n \mathsf{BT}(M \upharpoonright n)$.

Being η-long, these infinite normal forms might be more adequately called Nakajima trees [15]. However, their PCF counterparts are generally called PCF *Böhm trees* [6]. So, for consistency, we call them *η-long Böhm trees*, and for conciseness we will often omit *η-long*.

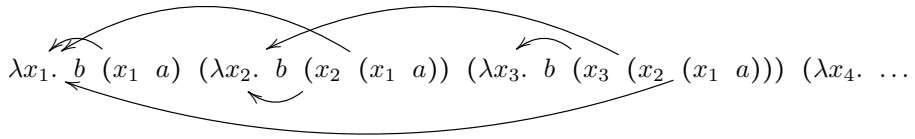
▸ **Definition 3.** λY-terms satisfying $\Gamma \vdash M : o$ are called *ground*. Given $n \geq 0$, we shall say that a ground term $\Gamma \vdash M : o$ is of *level* n if, for all $(x : \theta) \in \Gamma$, we have $ord(\theta) < n$.

Note that the free variables in a ground term of level 2 can have types of order 0 or 1, i.e. they are of the form $o \to \cdots \to o$, where o has at least one occurrence. Thus, Böhm trees of such terms are (possibly infinite) trees.

The restrictions on types of free variables in ground terms of level 2 are analogous to the restriction concerning types of terminal symbols in higher-order recursion schemes [7]. In fact, it can be shown that the Böhm trees of level-2 ground terms coincide with trees generated by higher-order recursive schemes [18], and therefore have a decidable MSO theory.

In this paper, we are interested in the study of the Böhm trees of *arbitrary* λY-terms. Unlike in the case of level-2 ground terms, Böhm trees of arbitrary λY-terms involve *binders*, as illustrated by the following example.

▸ **Example 4.** Consider $G = Y_{o\to(o\to o)\to o}\ (\lambda f^{o\to(o\to o)\to o}.\lambda y^{o}.\lambda x^{o\to o}.\ b\ (x\ y)\ (f\ (x\ y)))\ a$, which has type $(o \to o) \to o$ in context $a : o, b : o \to ((o \to o) \to o) \to o$. Its Böhm tree is



where the arrows indicate *binding information*: for each variable occurrence, the arrow indicates the location of the associated binder. Note that because all bound variables are used infinitely often, countably many variable names are required to represent binding.

This binding information is far from innocent: in fact, we prove in the next section that it makes MSO undecidable. In particular, the term G above has an undecidable MSO theory.

2.2 Undecidability of MSO with binders

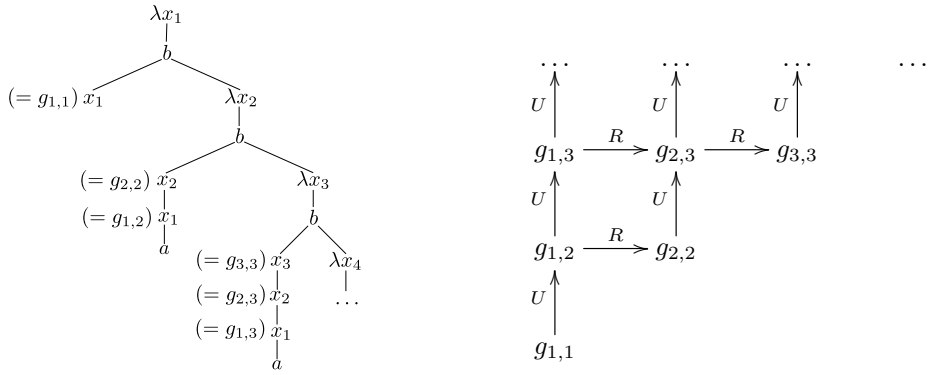
Let us first make formal what we mean by MSO on Böhm trees with binders.

▸ **Definition 5.** A **binding structure** $\mathcal{B} = (\mathcal{S}, \to)$ is a labelled transistion system, namely a set of *states* $\mathcal{S}$ and a relation $\to \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$, where $\mathcal{L}$ is the set of *labels* $\mathcal{L} = \mathbb{N} \cup \{\lambda\}$. For $l \in \mathcal{L}$, we write $a \xrightarrow{l} b$ for $(a, l, b) \in \to$.

Every $\lambda\perp^{\infty}$-term M defines a binding structure $\mathcal{B}(M)$ in which the states are nodes in the syntax tree of M, $a \xrightarrow{n} b$ holds if b is the n-th child of a and $a \xrightarrow{\lambda} b$ expresses the fact that b is a binder-node λx and a an occurrence of x. The MSO formulas over binding structures are given as follows.

$$\phi \quad ::= \quad X\ \mathrm{x} \mid \mathrm{x} \xrightarrow{\lambda} \mathrm{y} \mid \mathrm{x} \xrightarrow{n} \mathrm{y} \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \exists \mathrm{x}.\ \phi \mid \forall \mathrm{x}.\ \phi \mid \exists X.\ \phi \mid \forall X.\ \phi$$

We write the first-order variables x, y in a different font to distinguish them from the variable names of the λ-calculus. The validity of an MSO formula ϕ on a binding structure $\mathcal{B}$, written $\mathcal{B} \vDash \phi$, is defined as usual.



(a) The Böhm tree of G **(b)** The half-grid in $\mathsf{BT}(G)$

Figure 1 Construction of the half-grid

▸ **Theorem 6.** *MSO is undecidable on binding structures corresponding to Böhm trees of λY-terms.*

Proof. We reduce the well-known undecidable problem of deciding MSO on the infinite half-grid to MSO over binding structures generated by λY-terms. In particular, we will show that the binding structure $\mathcal{B}(\mathsf{BT}(G))$ contains an MSO-definable half-grid. The nodes of the half-grid will be occurrences of bound variables in $\mathsf{BT}(G)$, which are definable by the formula $\mathrm{grid}(\mathrm{x}) \triangleq \exists \mathrm{y}.\ \mathrm{x} \xrightarrow{\lambda} \mathrm{y}$.

We say that a bound variable occurrence x_i is at *layer* j if it occurs after j applications of b. Then for any $1 \le i \le j$, there is a unique occurrence of the bound variable x_i at layer j, we denote it by $g_{i,j}$. The $g_{i,j}$'s will be the nodes of the half-grid; in Figure 1a we describe the intended correspondence between occurrences of bound variables in $\mathsf{BT}(G)$ and the half-grid. The following two relations correspond respectively to moving right and up inside the grid.

$$\mathrm{x} \xrightarrow{R} \mathrm{y} \quad \triangleq \quad \mathrm{y} \xrightarrow{0} \mathrm{x}$$
$$\mathrm{x} \xrightarrow{U} \mathrm{y} \quad \triangleq \quad \exists \mathrm{z}.\ \mathrm{z} \xrightarrow{0^*} \mathrm{x} \wedge \mathrm{z} \xrightarrow{10^*} \mathrm{y} \wedge \exists \mathrm{x}'.\ \mathrm{x} \xrightarrow{\lambda} \mathrm{x}' \wedge \mathrm{y} \xrightarrow{\lambda} \mathrm{x}'$$

where $\xrightarrow{e_1 e_2}$ and $\xrightarrow{e^*}$ are respectively the relational composition of $\xrightarrow{e_1}$ and $\xrightarrow{e_2}$ and the transitive closure of $\xrightarrow{e}$, both MSO-definable. The relations define the half-grid in the sense that for any $1 \le i \le j$ and $1 \le i' \le j'$ we have $g_{i,j} \xrightarrow{R} g_{i',j'}$ iff $i' = i + 1$ and $j' = j$, and $g_{i,j} \xrightarrow{U} g_{i',j'}$ iff $i' = i$ and $j' = j + 1$. Consequently, the half-grid is MSO-definable within $\mathsf{BT}(G)$ and, thus, the latter must have an undecidable MSO theory. ◂

2.3 De Bruijn representations of binders

Theorem 6 exhibits a difference in expressivity between HORS and the λY-calculus. Nonetheless, in the following, we shall show how to construct HORS that faithfully capture Böhm trees generated by λY-terms. To that end, we shall use binder-free representations. As a consequence, we will be able to recast problems concerning the λY-calculus in the setting of HORS. In particular, we will retain the decidability of MSO model-checking such representations, which will provide us with a technique to verify a variety of properties of λY-terms in spite of Theorem 6.

Our binder-free representation scheme will rely on *De Bruijn levels* [8]: each bound variable x is given an index i, where i is the sum of a starting index and the number of

lambdas enclosing the binding location (De Bruijn levels should not be confused with De Bruijn *indices*, which associate numbers to variable uses and not their points of introduction, and can associate different numbers to different occurrences of the same variable).

▸ **Example 7.** Consider the term $M = \lambda f^{(o\to o)\to o}.Y_o\ (\lambda y^o.f\ (\lambda x^o.b\ x\ y))$, of type $((o \to o) \to o) \to o$ in context $b : o \to o \to o$. The Böhm tree of M, annotated with De Bruijn levels, has the shape $\lambda x_1.x_1\ (\lambda x_2.x_0\ x_2\ (x_1\ (\lambda x_3.x_0\ x_3\ (x_1\ (\lambda x_4.\ldots$, where $b : o \to o \to o$ is given an index of 0. Note that each variable introduction and use are now labelled with a natural number. This representation is unique, in the sense that, for a fixed choice of indices for free variables and a choice of a starting index, two α-equivalent terms must have the same representation.

With De Bruijn levels, $\lambda\bot$-terms can be represented as level-2 ground terms, typable in a special context defined below.

▸ **Definition 8.** Let Γ_{rep} be the following context.

$$\{\, z : o,\ succ : o \to o,\ var : o \to o,\ app : o \to o \to o,\ lam : o \to o \to o \,\}$$

Given $n \in \mathbb{N}$, we write $\overline{n}$ for the term $succ^n(z)$.

In particular, the terms $\overline{n}$ will be used to represent binding information as De Bruijn levels.

▸ **Definition 9.** Let $\Gamma \vdash M : \theta$ be a $\lambda\bot$-term and let $\nu : \Gamma \to \mathbb{N}$ be an injection. We define another $\lambda\bot$-term $\Gamma_{\mathsf{rep}} \vdash \mathsf{rep}_\nu(M, n) : o$ by

$$\begin{array}{lcllcl} \mathsf{rep}_\nu(\bot, n) & = & \bot & \qquad \mathsf{rep}_\nu(\lambda x.M, n) & = & lam\ \overline{n}\ \mathsf{rep}_{\nu\oplus\{x\mapsto n\}}(M, n+1) \\ \mathsf{rep}_\nu(x, n) & = & var\ \overline{\nu(x)} & \mathsf{rep}_\nu(MN, n) & = & app\ \mathsf{rep}_\nu(M, n)\ \mathsf{rep}_\nu(N, n). \end{array}$$

Note that, as long as $n > \max_{(x:\theta)\in\Gamma} \nu(x)$, $\mathsf{rep}_\nu(M, n)$ faithfully represents the syntactic structure of M. Given $\Gamma \vdash M : \theta$, where $\Gamma = \{x_1 : \theta_1, \cdots, x_k : \theta_k\}$, let $\nu : \Gamma \to \mathbb{N}$ be defined by $\nu(x_i) = i - 1$. We define $\mathsf{rep}(M)$ to be $\mathsf{rep}_\nu(M, k)$.

For any finite $\lambda\bot$-term M, $\mathsf{rep}(M)$ gives a binder-free representation of M as a ground $\lambda\bot$-term in context Γ_{rep}. Being monotonic, the function rep preserves ω-chains, and therefore can be used to represent the Böhm tree of any λY-term as a ground infinite $\lambda\bot^\infty$-term in context Γ_{rep}. This invites the question whether this infinite $\lambda\bot^\infty$-term can always be obtained as the Böhm tree of a level-2 ground λY-term.

▸ **Example 10.** Consider again the term M of Example 7. The binder-free representation of M's Böhm tree can be captured as an open (infinite) term in context Γ_{rep}, namely, $M_\infty = lam\ \overline{1}\ T(2)$, where

$$T(n) = app\ (var\ \overline{1})\ (lam\ \overline{n}\ (app\ (app\ (var\ \overline{0})\ (var\ \overline{n})\ T(n+1))))$$

More precisely, we have $\bigsqcup_n \mathsf{rep}(\mathsf{BT}(M \upharpoonright n)) = M_\infty$. In this case, it is easy to extract a λY-term $\Gamma_{\mathsf{rep}} \vdash M_{rep} : o$ such that $\mathsf{BT}(M_{rep}) = M_\infty$ from the definition of M_∞. This can be done by expressing the coinductive definition of M_∞ inside the λY-calculus as $M_{rep} = lam\ \overline{1}\ (Y_{o\to o}\ M_{step}\ \overline{2})$, where

$$M_{step} = \lambda T^{o\to o}.\ \lambda n^o.app\ (var\ \overline{1})\ (lam\ n\ (app\ (app\ (var\ \overline{0})\ (var\ n))\ (T\ (succ\ n))))).$$

Consequently, the Böhm tree of M_{rep} is a binder-free representation of the Böhm tree of M.

In our paper, we show how to obtain such representations in a systematic way.

▸ **Theorem 11.** *Let $\Gamma \vdash M : \theta$ be a λY-term. There exists a level-2 λY-term $\Gamma_{\mathsf{rep}} \vdash M_{rep} : o$ such that $\mathsf{BT}(M_{rep}) = \bigsqcup_n \mathsf{rep}(\mathsf{BT}(M \restriction n))$.*

Next we describe the construction underlying our proof of Theorem 11. In this section we only provide the necessary definitions, delegating the proof and a methodological discussion to Section 3. The construction uses *normalization by evaluation* [9] and, in particular, the observation that the technique can be internalized within the λY-calculus in our case.

Consider $\Gamma \vdash M : \theta$ with $\Gamma = \{x_1 : \theta_1, \cdots, x_k : \theta_k\}$. First we apply a simple transformation on type annotations inside M and substitute $o \to o$ for o: let M^* be defined as $M[o \to o/o]$. The substitution is meant to be applied to types θ occurring in λ-abstractions $(\lambda x^{\theta}.M)$ and fixed-point combinators (Y_θ). Observe that $x_1 : \theta_1^*, \cdots, x_k : \theta_k^* \vdash M^* : \theta^*$, where

$$o^* = o \to o, \qquad (\theta' \to \theta'')^* = (\theta')^* \to (\theta'')^*.$$

Next we define by mutual recursion two families of λ-terms $\{\downarrow_\theta\}$, $\{\uparrow_\theta\}$, subject to the following conditions: $\Gamma_{\mathsf{rep}} \vdash \downarrow_\theta : \theta^* \to o \to o$ and $\Gamma_{\mathsf{rep}} \vdash \uparrow_\theta : (o \to o) \to \theta^*$. We write $\lambda.M$ for $\lambda z^o.M$, where z does *not* occur in M.

$$\begin{aligned}
\downarrow_o &= \lambda x^{o \to o}.x \\
\downarrow_{\theta' \to \theta''} &= \lambda x^{(\theta' \to \theta'')^*}.\ \lambda v^o.\ \mathit{lam}\ v\ (\downarrow_{\theta''}\ (\ x\ (\uparrow_{\theta'}\ \lambda.(\mathit{var}\ v))\ )\ (\mathit{succ}\, v)\ ) \\
\uparrow_o &= \lambda x^{o \to o}.x \\
\uparrow_{\theta' \to \theta''} &= \lambda e^{o \to o}.\ \lambda a^{(\theta')^*}.\ \uparrow_{\theta''}\ (\ \lambda v^o.\ \mathit{app}\ (e\ v)\ (\downarrow_{\theta'}\ a\ v)\ )
\end{aligned}$$

Now, with all the definitions in place, it turns out that one can take M_{rep} to be

$$\downarrow_\theta\ M^*[\ \uparrow_{\theta_1} \lambda.(\mathit{var}\ \overline{0})/x_1\ ,\ \cdots\ ,\ \uparrow_{\theta_k} \lambda.(\mathit{var}\ \overline{k-1})/x_k\ ]\ \ \overline{k}.$$

In the next section we shall prove that M_{rep} indeed represents the Böhm tree of M in the sense of Theorem 11. Before that, we illustrate the outcome of the construction on an example.

▸ **Example 12.** Let us revisit the term M from Example 7. We have

$$M^* = \lambda f^{((o \to o) \to (o \to o)) \to (o \to o)}.Y_{o \to o}\ (\lambda y^{o \to o}.f\ (\lambda x^{o \to o}.b\ x\ y))$$

and, thus, $M_{rep} = \downarrow_{((o \to o) \to o) \to o}\ \ M^*[\uparrow_{o \to o \to o} \lambda.(\mathit{var}\ \overline{0})/b]\ \ \overline{1}$. By unfolding the definitions of $\downarrow_\theta, \uparrow_\theta$ and applying β-reduction the term can be rewritten. In fact, taking advantage of the β-equivalences

$$\begin{aligned}
\uparrow_{o \to o \to o} &\simeq_\beta \lambda e_1^{o \to o}.\lambda a_1^{o \to o}.\lambda a_2^{o \to o}.\lambda v_2^o.\ \mathit{app}\ (\mathit{app}\ (e_1\ v_2)\ (a_1\ v_2))\ (a_2\ v_2) \\
\downarrow_{((o \to o) \to o) \to o} &\simeq_\beta \lambda x^{(((o \to o) \to o) \to o)^*}.\lambda v_1^o.\ \mathit{lam}\ v_1\ (x\ M_{aux}\ (\mathit{succ}\ v_1)),
\end{aligned}$$

where $M_{aux} = \lambda a^{(o \to o)^*}.\lambda v_2^o.\ \mathit{app}\ (\mathit{var}\ v_1)\ (\mathit{lam}\ v_2\ (a\ (\lambda.\mathit{var}\ v_2)\ (\mathit{succ}\ v_2)))$, one can show that M_{rep} is β-equivalent to

$$\mathit{lam}\ \overline{1}\ (Y\ (\lambda y.\ \lambda v.\mathit{app}\ (\mathit{var}\ \overline{1})\ (\mathit{lam}\ v\ (\mathit{app}\ (\mathit{app}\ (\mathit{var}\ \overline{0})\ (\mathit{var}\ v))\ (y\ (\mathit{succ}\ v))))))\ \overline{2}),$$

the manually constructed term from Example 10.

3 Internalized normalization by evaluation

In this section, we present the mathematical development that led to the term transformation of the previous section. As already mentioned, the main idea comes from *normalization by evaluation* (NBE). NBE is a general method used to construct normal forms for terms through their denotational semantics; the reader is referred to [9] for an introduction. The basic idea is to interpret terms in a suitable semantic universe. By soundness, the interpretation will be invariant under syntactic reduction. Moreover, if one uses a class of domains expressive enough to encode representations of normal forms, then one can attempt to extract a representation of the normal form of a term from the semantics. Notably, the extraction can be performed by evaluating the interpretation map on well-chosen elements of the domain, which yields an entirely semantic normalization procedure. NBE has been described for a variety of languages, including the simply-typed λ-calculus [5] as well as richer languages with coproducts [4], or type theory [1]. Our presentation of NBE follows along the lines of [9]. The main novelty is the internalization of the process within the λY-calculus itself, which ultimately allows us to present NBE as a term transformation.

3.1 Semantic NBE for the λY-calculus

We first describe a domain semantics for the λY-calculus that will be exploited to obtain an NBE procedure. We assume the basic vocabulary of domain theory, e.g. [20]. Let us recall that an ω-cpo is a partial order where each ω-chain (of the form $x_1 \leq x_2 \leq \dots$) has a supremum. We will interpret types as *pointed* ω-cpos, i.e., ω-cpos with a bottom element written $\bot$. A function $f : A \to B$ is *continuous* if it preserves suprema of ω-chains. The set of continuous functions from A to B, ordered pointwise, is itself a pointed ω-cpo, denoted by $A \to B$.

Our NBE procedure will generate representations of Böhm trees of λY-terms as elements of the pointed ω-cpo of $\lambda\bot^\infty$-terms of ground type in context Γ_{rep}, henceforth referred to as E. Representations based on De Bruijn levels lack compositionality: the indices present in a subterm depend on the number of variables abstracted in the context in which the subterm occurs. In order to overcome the difficulty, instead of constructing directly De Bruijn terms in E, we will construct *term families* [9], intuitively functions $\mathbb{N} \to E$, which - unlike De Bruijn terms - can be manipulated compositionally.

We now describe our semantic NBE for λY. In our case, E will also be used to represent natural numbers, so term families will be interpreted as elements of the pointed ω-cpo $\widehat{E} = E \to E$, where the first E plays the role of $\mathbb{N}$. For the sake of clarity, we will sometimes write N instead of E to highlight subterms representing natural numbers. This should not create any confusion.

Our NBE relies on the standard interpretation of the λY-calculus with $[\![o]\!] = \widehat{E}$. Note that we write $\boldsymbol{\lambda}$, rather than λ, to stress that the semantic operation of function abstraction (on continuous functions between pointed ω-cpos) is meant.

$$[\![o]\!] = \widehat{E} \qquad [\![x]\!]_\rho = \rho(x) \qquad [\![\bot]\!]_\rho = \bot \qquad [\![\lambda x^\theta.M]\!]_\rho = \boldsymbol{\lambda} a^{[\![\theta]\!]}.[\![M]\!]_{\rho \oplus \{x \mapsto a\}}$$

$$[\![\theta_1 \to \theta_2]\!] = [\![\theta_1]\!] \to [\![\theta_2]\!] \qquad [\![M\ N]\!]_\rho = [\![M]\!]_\rho([\![N]\!]_\rho) \qquad [\![Y_\theta]\!]_\rho = \bigsqcup_n \boldsymbol{\lambda} f^{[\![\theta]\!] \to [\![\theta]\!]}.f^n(\bot)$$

We claimed above that unlike raw De Bruijn terms, term families can be constructed compositionally. This is done with the help of the following continuous functions, which

generalize term constructors of E to $\widehat{E}$.

$$\begin{array}{lcl}\widehat{var} & = & \boldsymbol{\lambda} v^N.\boldsymbol{\lambda} n^N.var\ v : N \to \widehat{E}\\ \widehat{app} & = & \boldsymbol{\lambda} e_1^{\widehat{E}}.\boldsymbol{\lambda} e_2^{\widehat{E}}.\boldsymbol{\lambda} n^N.app\ (e_1(n))\ (e_2(n)) : \widehat{E} \to \widehat{E} \to \widehat{E}\\ \widehat{lam} & = & \boldsymbol{\lambda} f^{N\to\widehat{E}}.\boldsymbol{\lambda} n^N.lam\ n\ (f(n)(succ\ n)) : (N \to \widehat{E}) \to \widehat{E}\end{array}$$

Using these extended constructors, one can define families of continuous functions, traditionally called *reification* ($\Downarrow_\theta \colon [\![\theta]\!] \to \widehat{E}$) and *reflection* ($\Uparrow_\theta \colon \widehat{E} \to [\![\theta]\!]$).

$$\begin{array}{lcllcl}\Downarrow_o x & = & x & \Downarrow_{\theta_1\to\theta_2} x & = & \widehat{lam}(\boldsymbol{\lambda} n^N.\ \Downarrow_{\theta_2} (x(\Uparrow_{\theta_1} (\widehat{var}(n)))))\\ \Uparrow_o e & = & e & \Uparrow_{\theta_1\to\theta_2} e & = & \boldsymbol{\lambda} x^{[\![\theta_1]\!]}.\ \Uparrow_{\theta_2} (\widehat{app}(e)(\Downarrow_{\theta_2} (x)))\end{array}$$

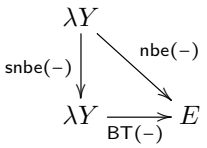
For closed terms $\vdash M : \theta$, the normalization-by-evaluation function can now be defined by setting $\mathsf{nbe}(M) = \Downarrow_\theta ([\![M]\!])(\overline{0})$. In order to apply the same method to open terms $\Gamma \vdash M : \theta$, where $\Gamma = \{x_1 : \theta_1, \ldots, x_k : \theta_k\}$, we need to build a canonical inhabitant of $[\![\Gamma]\!]$ by defining the *canonical* semantic environment $\rho_\Gamma : \Pi_{x_i \in \Gamma} [\![\theta_i]\!]$, associating $\Uparrow_{\theta_i} (\boldsymbol{\lambda}\_^N.var\ (\overline{i-1})) \in [\![\theta_i]\!]$ to any x_i. This leads to a generalized variant of the normalization function, defined by: $\mathsf{nbe}(M) = \Downarrow_\theta ([\![M]\!]_{\rho_\Gamma})(\overline{k})$.

▸ Proposition 13. For any $\lambda\bot$-term $\Gamma \vdash M$ in β-normal η-long form, $\mathsf{nbe}(M) = \mathsf{rep}(M)$. For any λY-term $\Gamma \vdash M : \theta$, $\mathsf{nbe}(M) = \bigsqcup_n \mathsf{rep}(\mathsf{BT}(M \upharpoonright n))$.

Proof. The first part is proved by induction on M. For the second part, we reason equationally as follows: $\mathsf{nbe}(M) \overset{(1)}{=} \Downarrow_\theta ([\![M]\!]_{\rho_\Gamma})(\overline{k}) \overset{(2)}{=} \Downarrow_\theta (\bigsqcup_n [\![M \upharpoonright n]\!]_{\rho_\Gamma})(\overline{k}) \overset{(3)}{=} \bigsqcup_n \Downarrow_\theta ([\![M \upharpoonright n]\!]_{\rho_\Gamma})(\overline{k}) \overset{(4)}{=} \bigsqcup_n \Downarrow_\theta ([\![\mathsf{BT}(M \upharpoonright n)]\!]_{\rho_\Gamma})(\overline{k}) \overset{(5)}{=} \bigsqcup_n \mathsf{rep}(\mathsf{BT}(M \upharpoonright n))$ where (1) is by definition of NBE, (2) is by induction on M from the definition of interpretation, (3) is by continuity of $\Downarrow_\theta$ and application. Since $M \upharpoonright n$ is a simply-typed $\lambda\bot$-term, it has a normal form $\mathsf{BT}(M \upharpoonright n)$ such that $M \upharpoonright n \simeq_{\beta\eta} \mathsf{BT}(M \upharpoonright n)$. By soundness of the domain semantics, this implies that $[\![M \upharpoonright n]\!]_\rho = [\![\mathsf{BT}(M \upharpoonright n)]\!]_\rho$ and (4) is true. Finally, (5) follows from the first part. ◂

3.2 Internalization

Following Section 2, for any λY-term $\Gamma \vdash M : \theta$, we define the syntactic NBE by $\mathsf{snbe}(M) = \downarrow_\theta M^*[\uparrow_{\theta_i} \lambda.(var\ \overline{i-1})/x_i]\ \overline{k}$ so that $\Gamma_{\mathsf{rep}} \vdash \mathsf{snbe}(M) : o$ ($\downarrow_\theta$ and $\uparrow_\theta$ are the terms defined in Section 2). In the remainder of this section we show the correctness of snbe, namely $\mathsf{BT}(\mathsf{snbe}(M)) = \bigsqcup_n \mathsf{rep}(\mathsf{BT}(M \upharpoonright n))$, by noting that snbe amounts to an internalization of the NBE procedure inside the λY-calculus, i.e., that the following triangle commutes.



To carry out the argument, below we introduce another interpretation of terms $\Gamma_{\mathsf{rep}}, \Gamma \vdash \theta$, written $\langle\!-\!\rangle$, which will treat variables from Γ_{rep} as constants. Accordingly, the sequents will be interpreted by a continuous function from $\langle\Gamma\rangle$ to $\langle\theta\rangle$, with Γ_{rep} omitted. Similarly, the semantic environment for a context $\Gamma_{\mathsf{rep}}, x_1 : \theta_1, \ldots, x_k : \theta_k$ is simply $\rho : \Pi_{x_i \in \Gamma} \langle\theta_i\rangle$. In the same vein, whenever we write $\Gamma \vdash M : \theta$ from now on, we will assume that none of the variables from Γ_{rep} occurs in Γ (this does not affect the generality of our results, as free variables of a λY-term can be renamed to avoid clashes with Γ_{rep}). Below we let z range over the variables from Γ_{rep}.

$$\begin{array}{rclcrclcrcl}
\langle o\rangle & = & E & & \langle x\rangle_\rho & = & \rho(x) & & \langle var\rangle_\rho & = & \boldsymbol{\lambda} n^E.var\ n \\
\langle \theta\to\theta'\rangle & = & \langle\theta\rangle\to\langle\theta'\rangle & & \langle M\ N\rangle_\rho & = & \langle M\rangle_\rho(\langle N\rangle_\rho) & & \langle app\rangle_\rho & = & \boldsymbol{\lambda} e_1^E.\boldsymbol{\lambda} e_2^E.app\ e_1\ e_2 \\
 & & & & \langle \lambda x^\theta.M\rangle_\rho & = & \boldsymbol{\lambda} a^{\langle\theta\rangle}.\langle M\rangle_{\rho\oplus\{x\mapsto a\}} & & \langle lam\rangle_\rho & = & \boldsymbol{\lambda} n^E.\boldsymbol{\lambda} e^E.lam\ n\ e \\
\langle z\rangle_\rho & = & z & & \langle Y_\theta\rangle_\rho & = & \bigsqcup_n \boldsymbol{\lambda} f^{\langle\theta\rangle\to\langle\theta\rangle}.f^n(\bot) & & \langle succ\rangle_\rho & = & \boldsymbol{\lambda} n^E.succ\ n
\end{array}$$

The interpretations $[\![-]\!]$ and $\langle - \rangle$ are related by the following lemma, which is easily proved by induction. We apply $\langle - \rangle$ to $M^\star$ on the understanding that $\Gamma \vdash M : \theta$ implies $\Gamma_{\mathsf{rep}}, \Gamma^\star \vdash M^\star : \theta^\star$ and that $[\![\Gamma]\!] = \langle \Gamma_{\mathsf{rep}}, \Gamma^\star\rangle$.

▸ **Lemma 14.** *For any type θ, $[\![\theta]\!] = \langle\theta^\star\rangle$. For any λY-term $\Gamma \vdash M : \theta$ and any semantic environment $\rho \in [\![\Gamma]\!]$, we have $[\![M]\!]_\rho = \langle M^\star\rangle_\rho$.*

On the other hand, $\langle - \rangle$ is related to Böhm trees by the following lemma.

▸ **Lemma 15.** *For any λY-term $\Gamma_{\mathsf{rep}} \vdash M : o$, we have $\langle M\rangle = \mathsf{BT}(M)$.*

Proof. Suppose first that $\Gamma_{\mathsf{rep}} \vdash M : o$ is a $\lambda\bot$-term in β-normal η-long form, i.e. $M = \mathsf{BT}(M)$. Then M must be an applicative term built from $\bot, z, succ, var, lam$ and app. By induction on M it follows that $\langle M\rangle = M$ and, thus, $\langle M\rangle = \mathsf{BT}(M)$.

Now consider arbitrary $\Gamma_{\mathsf{rep}} \vdash M : o$, and $n \in \mathbb{N}$. By construction, $M \upharpoonright n$ is a $\lambda\bot$-term. By normalization of the simply-typed λ-calculus, $M \upharpoonright n$ has a Böhm tree $\mathsf{BT}(M \upharpoonright n)$ such that $M \upharpoonright n \simeq_{\beta\eta} \mathsf{BT}(M \upharpoonright n)$. By soundness of the denotational model, it follows that $\langle M \upharpoonright n\rangle = \langle \mathsf{BT}(M \upharpoonright n)\rangle$. By the first part above, we can deduce $\langle M \upharpoonright n\rangle = \mathsf{BT}(M \upharpoonright n)$. The lemma then follows by continuity. ◂

Putting all the lemmas together, we arrive at our main result.

▸ **Theorem 16.** *For any λY-term $\Gamma \vdash M : \theta$, $\mathsf{BT}(\mathsf{snbe}(M)) = \bigsqcup_n \mathsf{rep}(\mathsf{BT}(M \upharpoonright n))$.*

Proof. From the following calculations. $\mathsf{BT}(\mathsf{snbe}(M)) \overset{(1)}{=} \langle \mathsf{snbe}(M)\rangle \overset{(2)}{=} \langle \downarrow_\theta\ M^*[\ \uparrow_{\theta_i} \lambda.(var\ \overline{i-1})/x_i]\ \overline{k}\rangle \overset{(3)}{=} \langle\downarrow_\theta\rangle(\langle M^*\rangle_{x_i\mapsto\langle\uparrow_{\theta_i}\lambda.(var\ \overline{i-1})\rangle})(\langle\overline{k}\rangle) \overset{(4)}{=} \Downarrow_\theta ([\![M]\!]_{x_i\mapsto\Uparrow_{\theta_i}(\boldsymbol{\lambda}.(var\ \overline{i-1}))})(\overline{k}) \overset{(5)}{=} \mathsf{nbe}(M) \overset{(6)}{=} \bigsqcup_n \mathsf{rep}(\mathsf{BT}(M \upharpoonright n))$. (1) follows from Lemma 15, (2) and (3) from the definitions of snbe and $\langle - \rangle$ respectively, (4) is a consequence of $\langle\downarrow_\theta\rangle = \Downarrow_\theta$, $\langle\uparrow_\theta\rangle = \Uparrow_\theta$ and Lemma 14. Finally, (5) follows from the definition of nbe and (6) from Proposition 13. ◂

We conclude this section with a discussion on the effect of the $M \mapsto \mathsf{snbe}(M)$ transformation on the order of terms (the *order* of a λY-term, written $ord(M)$, is the maximal order of the types of its subterms). We note that, given $\Gamma \vdash M : \theta$, the term $\mathsf{snbe}(M)$ is of order at most $ord(M) + 2$, because it contains $\downarrow_\theta$, which has type $\theta^* \to (o \to o)$ of order $ord(\theta) + 2$. However, as in Example 12, $\mathsf{snbe}(M)$ can be simplified by one size-decreasing β-reduction step to M' of order $ord(M) + 1$. Furthermore, if one is interested in obtaining an equivalent HORS then M' can be transformed into a HORS of order $ord(M)$ [18].

4 Generalization to PCF

Here we show that the methodology of the previous section can be applied to PCF [17]. This brings us closer to realistic programs, which can branch on data values. PCF extends the λY-calculus in that the ground type o is replaced with the type of natural numbers, basic arithmetic and zero testing. It was designed to be Turing-complete, so in order to avoid obvious undecidability we focus on its finitary variants. In what follows we consider boolean PCF_2 in which o is the type of booleans, but it should be clear that the techniques extend to any finite ground type.

4.1 PCF_2 and PCF_2 Böhm trees

PCF_2 types are defined by the grammar $\theta ::= \mathtt{B} \mid \theta \to \theta$ and terms are given by $M, N ::= tt \mid ff \mid \text{if } M \text{ then } N_1 \text{ else } N_2 \mid x \mid \lambda x^\theta.M \mid M\ N \mid Y_\theta\ M$. The following typing rules apply.

$$\frac{}{\Gamma \vdash tt : \mathtt{B}} \qquad \frac{}{\Gamma \vdash ff : \mathtt{B}} \qquad \frac{\Gamma \vdash M : \mathtt{B} \qquad \Gamma \vdash N_1 : \theta \qquad \Gamma \vdash N_2 : \theta}{\Gamma \vdash \text{if } M \text{ then } N_1 \text{ else } N_2 : \theta}$$

For PCF_2, the canonical forms that we wish to generate are PCF Böhm trees: they are (potentially infinite) trees generated by the rules displayed below.

$$\frac{}{\Gamma \vdash \bot, tt, ff : \mathtt{B}} \qquad \frac{\Gamma, \overrightarrow{\ldots, x_i : A_i, \ldots} \vdash M : \mathtt{B}}{\Gamma \vdash \lambda\overrightarrow{x}.M : \overrightarrow{A} \to \mathtt{B}} \qquad \frac{\overrightarrow{\Gamma \vdash M_i : \theta_i} \quad \Gamma \vdash N_1, N_2 : \mathtt{B} \quad (x : \overrightarrow{\theta} \to \mathtt{B}) \in \Gamma}{\Gamma \vdash \text{if } x\ \overrightarrow{M} \text{ then } N_1 \text{ else } N_2 : \mathtt{B}}$$

Such trees were first introduced along with the game semantics for PCF [11, 2], serving as a syntactic counterpart to *strategies*. Later developments in game semantics [3, 13] showed that two terms of PCF have the same strategy, i.e. the same PCF Böhm tree, if and only if they are indistinguishable by evaluation contexts having access to computational effects such as state and control operators. This makes PCF Böhm trees a natural candidate for a representation of higher-order programs to be used for verification purposes, since purely functional programs are eventually executed in runtime environments that have access to such effects.

We also consider the variant $\mathrm{PCF}_{2,\bot}$ consisting of Y-free terms of PCF_2, extended with a constant $\bot : \mathtt{B}$. They are partially ordered by the obvious adaptation of the partial order $\sqsubseteq$ on $\lambda\bot$-terms, still written $\sqsubseteq$. As before, this partial order extends to an ω-cpo on infinite terms of $\mathrm{PCF}_{2,\bot}$, referred to as $\mathrm{PCF}^\infty_{2,\bot}$.

The operational semantics for PCF_2 for which PCF Böhm trees can be taken to be canonical representatives is obtained by adding the rules listed below to β, η and Y. For brevity, if M then N_1 else N_2 is written as $\mathrm{if}(M, N_1, N_2)$. The rules are restricted to the cases where both sides typecheck.

$$\mathrm{if}(tt, N_1, N_2) \to_{\beta'_1} N_1 \qquad M \to_{\eta'} \mathrm{if}(M, tt, ff) \qquad \mathrm{if}(M, N_1, N_2)\ N' \to_{\gamma_1} \mathrm{if}(M, N_1\ N', N_2\ N')$$
$$M \to_{\eta'} \mathrm{if}(M, tt, ff) \qquad \mathrm{if}(\mathrm{if}(M, N_1, N_2), N_3, N_4) \to_{\gamma_2} \mathrm{if}(M, \mathrm{if}(N_1, N_3, N_4), \mathrm{if}(N_2, N_3, N_4))$$

We write $\simeq_{\mathrm{PCF}}$ for the corresponding equational theory. For $\mathrm{PCF}_{2,\bot}$ we also add the rule $\mathrm{if}(\bot, N_1, N_2) \to_\bot \bot$. The following property can then be established by standard means.

▸ Proposition 17. For any term $\Gamma \vdash M : \theta$ of $\mathrm{PCF}_{2,\bot}$ there is a PCF Böhm tree $\Gamma \vdash \mathsf{BT}(M) : \theta$ such that $M \simeq_{\mathrm{PCF}} \mathsf{BT}(M)$.

To define Böhm trees of arbitrary PCF_2-terms $\Gamma \vdash M : \theta$, we define the nth approximant of M, written $M \upharpoonright n$, by replacing fixpoint combinators Y_θ with $\lambda f^\theta.f^n(\bot_\theta)$, yielding a term of $\mathrm{PCF}_{2,\bot}$. In the general case, the PCF Böhm tree is then defined by $\mathsf{BT}(M) = \bigsqcup_n \mathsf{BT}(M \upharpoonright n)$.

4.2 Representability

We would like to represent PCF Böhm trees of PCF_2-terms as level-2 ground λY-terms. To that end, we shall use a new context Γ_{pcf}, defined as Γ_{rep} augmented with $\{\, tt : o, ff : o, \mathit{if} : o \to o \to o \to o \,\}$ to take the shape of PCF Böhm trees into account. Just as before, the set of ground $\lambda\bot^\infty$-terms typed in context Γ_{pcf} forms a pointed ω-cpo. It will be the target of our transformation and, again, we will denote it by E. The representation function $\mathsf{rep}(-)$ of Section 2 extends easily to a function mapping any $\mathrm{PCF}_{2,\bot}$-term $\Gamma \vdash M : \theta$ to a $\lambda\bot$-term $\Gamma_{\mathsf{pcf}} \vdash \mathsf{rep}(M) : o$ using De Bruijn levels. Our representation result for PCF_2 can then be stated as follows.

▸ **Theorem 18.** *For any* PCF_2 *term* $\Gamma \vdash M : \theta$*, there is a level-2* λY*-term* $\Gamma_{\mathsf{pcf}} \vdash M_{rep} : o$ *such that* $\mathsf{BT}(M_{rep}) = \bigsqcup_n \mathsf{rep}(\mathsf{BT}(M \upharpoonright n))$.

To construct M_{rep} from M, we first apply the following syntactic transformation translating types and terms of PCF_2 to λY-terms, combining the operation $(-)^*$ on λY-terms defined in Section 2 with an elimination of booleans based on Church encodings.

$$\begin{array}{rcl@{\qquad}rcl} \mathtt{B}^* & = & (o \to o) \to (o \to o) \to (o \to o) & tt^* & = & \lambda a^{o\to o}.\lambda b^{o\to o}.a \\ (\theta' \to \theta'')^* & = & (\theta')^* \to (\theta'')^* & ff^* & = & \lambda a^{o\to o}.\lambda b^{o\to o}.b \end{array}$$

$$(\text{if } M \text{ then } N_1 \text{ else } N_2)^* = \lambda \overrightarrow{x}^{\overrightarrow{\theta^*}}.\lambda a^{o\to o}.\lambda b^{o\to o}.M^*\ (N_1^*\ \overrightarrow{x}\ a\ b)\ (N_2^*\ \overrightarrow{x}\ a\ b)$$

where, in the last equation, N_1 and N_2 have type $\overrightarrow{\theta} \to \mathtt{B}$. The translation $(-)^*$ can be extended to all the other term constructors in the obvious way. Note that, since this translation gives us a λY-term, we can already apply the transformation of Section 2 and obtain a λY-term generating the Böhm tree of M^*. To get the PCF *Böhm tree of* M instead one can modify the $\downarrow_\theta, \uparrow_\theta$ terms as follows.

$$\begin{array}{rcl@{\qquad}rcl} \downarrow_{\mathtt{B}} & = & \lambda x^{\mathtt{B}^*}.x\ (\lambda.tt)\ (\lambda.ff) & \uparrow_{\mathtt{B}} & = & \lambda e^{o\to o}.\lambda a^{o\to o}.\lambda b^{o\to o}.\lambda n^o.if\ (e\ n)\ (a\ n)\ (b\ n) \\ \downarrow_{\theta'\to\theta''} & = & \lambda x^{(\theta'\to\theta'')^*}.\ \lambda v^o.\ lam\ v\ (\downarrow_{\theta''}(\ x\ (\uparrow_{\theta'} \lambda.(var\ v))\ )\ )\ (succ\, v)\ ) \\ \uparrow_{\theta'\to\theta''} & = & \lambda e^{o\to o}.\ \lambda a^{(\theta')^*}.\ \uparrow_{\theta''}(\ \lambda v^o.\ app\ (e\ v)\ (\downarrow_{\theta'} a\ v)\ ) \end{array}$$

Assuming $\Gamma = x_1 : \theta_1, \ldots, x_k : \theta_k$, we can then set $M_{rep} = \downarrow_\theta M^*[\uparrow_{\theta_i} \lambda.var\ \overline{i-1}/x_i]\ \overline{k}$.

The notion of *order* on types and terms is generalised to PCF_2 by setting $ord(\mathtt{B}) = 0$. As for λY, M_{rep} can be simplified by one β-reduction step to M' of order $ord(M) + 2$. M' can then be transformed into a HORS of order $ord(M) + 1$ [18].

5 Conclusion

We have shown that faithful representations of η-long Böhm trees of arbitrary λY-terms and PCF Böhm trees of PCF_2-terms can be generated by higher-order recursion schemes. We can use the result to reduce various decision problems related to the λY-calculus or PCF_2 to problems for higher-order recursive schemes.

The first class of problems to which the reduction technique can be applied are equivalence problems.

▸ **Corollary 19.** *The following problems are recursively equivalent: HORS equivalence, Böhm tree equivalence in the* λY*-calculus,* PCF *Böhm tree equivalence in* PCF_2.

The decidability status of these problems is currently unknown, but they are known to be related to two other interesting problems: the equivalence problem for deterministic collapsible pushdown automata [10] and contextual equivalence for PCF_2-terms with respect to contexts with state and control effects (contextual equivalence for PCF_2-terms with respect to purely functional contexts is undecidable, even without Y [14]).

In a different direction, we can exploit the decidability result for MSO on trees generated by HORS [16] to decide a variety of properties of Böhm trees. Here the undecidability result for binders places a limit on the kind of binding-related information that can be captured by MSO on our representations. However, many interesting problems on Böhm trees still fall within the range of our representation.

▸ **Corollary 20.** *The following problems are decidable for* λY*- and* PCF_2*-terms: normalizability (is a given term equivalent to a* Y*-free term?), finiteness (is the Böhm tree/*PCF

Böhm tree finite, i.e. a $\lambda\bot$/$\mathrm{PCF}_{2,\bot}$*-term?), solvability (is the Böhm tree/*PCF *Böhm tree non-empty after leading abstractions?), prefix (does the Böhm tree/*PCF *Böhm tree have a given finite prefix?).*

The results follow from the fact that all the relevant properties are expressible in MSO. Normalizability and solvability for the λY-calculus were already known to be decidable [19]. To the best of our knowledge, the other consequences are new. In future work, we would like to understand the exact complexity of these problems and see whether our results yield optimal bounds.

Acknowledgment. We would like to thank Sylvain Salvati for pointing out to us the undecidability argument for MSO on trees with binding. This research was conducted while the first author was in Cambridge, supported by the Advanced Grant ECSYM. The second author gratefully acknowledges the support of the EPSRC (EP/J019577/1).

References

1 A. Abel, T. Coquand, and P. Dybjer. Normalization by evaluation for Martin-Löf type theory with typed equality judgements. In *LICS*, pp 3–12, 2007.
2 S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Inf. and Comput.*, 163:409–470, 2000.
3 S. Abramsky and G. McCusker. Linearity, sharing and state. In *Algol-like languages*, pp 297–329, Birkhaüser, 1997.
4 T. Altenkirch, P. Dybjer, M. Hofmann, and P. J. Scott. Normalization by evaluation for typed lambda calculus with coproducts. In *LICS*, pp 303–312, 2001.
5 U. Berger and H. Schwichtenberg. An inverse of the evaluation functional for typed lambda-calculus. In *LICS*, pp 203–212, 1991.
6 P.-L. Curien. Abstract Böhm trees. *Math. Struct. in Comput. Sci.*, 8(6):559–591, 1998.
7 W. Damm. The IO- and OI-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982.
8 N. G. De Bruijn. Lambda calculus notation with nameless dummies. In *Indagationes Mathematicae (Proceedings)*, volume 75, pages 381–392. Elsevier, 1972.
9 P. Dybjer and A. Filinski. Normalization and partial evaluation. In *APPSEM*, LNCS 2395, pp 137–192, 2000.
10 M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In LICS, pp 452–461, 2008.
11 J. M. E. Hyland and C.-H. L. Ong. On Full Abstraction for PCF. *Inf. and Comput.*, 163(2):285–408, 2000.
12 N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL*, pp 416–428, 2009.
13 J. Laird. Full abstraction for functional languages with control. In *LICS*, pp 58–67, 1997.
14 R. Loader. Finitary PCF is not decidable. *Theor. Comput. Sci.*, 266(1-2):341–364, 2001.
15 R. Nakajima. Infinite normal forms for the λ-calculus. In *Proc. Symp. λ-calculus and Computer Science*, pages 62–82. Springer-Verlag, 1975.
16 C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pp 81–90, 2006.
17 G. D. Plotkin. LCF considered as a programming language. *Theor. Comput. Sci.*, 5:223–255, 1977.
18 S. Salvati and I. Walukiewicz. Recursive schemes, Krivine machines, and collapsible pushdown automata. In *RP*, LNCS 7550, pp 6–20, 2012.
19 R. Statman. On the lambda Y-calculus. *Ann. Pure Appl. Logic*, 130(1-3):325–337, 2004.
20 G. Winskel. *The Formal Semantics of Programming Languages.* MIT Press, 1993. Foundations of Computing Series.