Game Semantics and Normalization by Evaluation

Pierre Clairambault[1] and Peter Dybjer[2]

[1] CNRS, ENS Lyon, Inria, UCBL, Université de Lyon, Lyon, France
[2] Chalmers Tekniska Högskola, Gothenburg, Sweden

Abstract. We show that Hyland and Ong's game semantics for PCF can be presented using normalization by evaluation (nbe). We use the bijective correspondence between innocent well-bracketed strategies and PCF Böhm trees, and show how operations on PCF Böhm trees, such as composition, can be computed lazily and simply by nbe. The usual equations characteristic of games follow from the nbe construction without reference to low-level game-theoretic machinery. As an illustration, we give a Haskell program computing the application of innocent strategies.

1 Introduction

In game semantics [17,3] types are interpreted as games between two players (Player/Opponent), and programs as strategies for Player. Combinators for programs become operations on strategies that can be quite complex. Composition of strategies for instance, involves an intricate mechanism of parallel interaction plus hiding *à la* CCS. The proof that they satisfy required equations is typically lengthy and non-trivial. In Hyland and Ong's game semantics of PCF [17] in particular, strategies interpreting programs are *innocent*: recall that a strategy is a set of admissible plays for Player, and is innocent when Player's action only depends on a subset of the play called the *P-view*. So innocent strategies are specified – and often defined as – a set of P-views (the *view functions*). Composing two such strategies involves computing the full set of plays of both strategies, composing these using parallel interaction plus hiding, and computing the P-views of the compound strategy. These computations are quite complex.

Several authors have tried to give more direct or elegant presentations of innocent strategies and their composition. Quite early, Curien gave syntactic representations of innocent strategies as *abstract Böhm trees* [10] and gave an abstract machine (the VAM) to compose them – this machinery is also quite involved. Amadio and Curien [5] reason about innocent strategies for PCF syntactically as *PCF Böhm trees* and compose them via infinitary rewriting. Finally and more recently, Harmer, Hyland and Melliès gave an elegant categorical reconstruction of innocent strategies from a basic category of simple games [16]. The resulting algorithm to compose strategies is however still quite involved.

In the present paper we provide yet another presentation of the innocent game semantics for PCF. Like Amadio and Curien we represent strategies as

© Springer-Verlag Berlin Heidelberg 2015
A. Pitts (Ed.): FOSSACS 2015, LNCS 9034, pp. 56–70, 2015.
DOI: 10.1007/978-3-662-46678-0_4

PCF Böhm trees. However, we use normalization by evaluation (nbe) [6] rather than infinitary rewriting. As Aehlig and Joachimski [4] showed, nbe can be used for computing the potentially partial and infinite Böhm trees of the untyped lambda calculus, and not only for computing normal forms. To this end they used lazy evaluation for computing the finite approximations of the Böhm tree.

We here adapt the nbe technique to PCF Böhm trees. To compute an operation on terms (PCF Böhm trees, strategies), we first evaluate them in a non-standard semantic domain where we then perform the corresponding semantic operation. Finally, the result is read back to the resulting PCF Böhm tree. For example, composition of PCF Böhm trees is performed by ordinary function composition in the semantic domain.

Finally note that our construction and its soundness are independent of the standard presentation of game semantics. The fact that our model agrees with the standard presentation of the innocent model for PCF follows from high-level reasons that use the soundness, adequacy and definability properties of game semantics. In particular, our contribution does not (and does not aim to) give insights into the low-level combinatorics of innocent interaction.

Related work. The first author and Murawski [9] use nbe to generate representations of innocent strategies in boolean PCF by higher-order recursion schemes. They exploit the fact that booleans can be replaced by their Church encoding. In contrast we consider here PCF with a datatype for lazy natural numbers, which is infinite. Thus our proof requires different techniques and is significantly more complex. Our approach is not particular to natural numbers and should smoothly extend to McCusker's games for recursive types [18].

Our non-standard semantic domain is similar to those used for previous work on untyped nbe [15,14], where semantic elements can be thought of as infinitary terms in higher order abstract syntax (hoas). We define a semantic domain for PCF Böhm trees in hoas, and show how to compute semantic operations in such a way that certain commuting conversions are executed. This is a key difference to the semantic domain used for normalizing terms in Gödel system T [1], which does not include these commuting conversions. We remark that although this approach to nbe initially was used for untyped nbe, it can also be used to advantage for typed languages. For example, it was a crucial step in devising nbe for dependent type theory [2] to use a similar semantic domain of untyped normal forms in hoas.

Plan of the paper. The rest of the paper is organized as follows. In Section 2 we introduce the syntax and reduction rules of PCF and our notion of model. We recall some notions from Hyland-Ong game semantics including the notions of innocent strategy and PCF Böhm tree. We also write a Haskell program for application of PCF Böhm trees using nbe. In Section 3 we provide a domain interpretation of the non-standard model used in the Haskell program. We prove that the interpretation function preserves all syntactic conversions of PCF, and that the interpretation of a term is identical to its PCF Böhm tree. In Section 4 we use these results to reconstruct the game model of PCF from nbe.

2 PCF, Innocent Strategies, and PCF Böhm Trees

2.1 PCF

Our version of PCF is close to Plotkin's original [20], except that we consider *lazy* (rather than *flat*) natural numbers. (Ultimately, we are interested in the connection between game semantics and Martin-Löf's meaning explanations [13], which are based on lazy evaluation of the terms of intuitionistic type theory.) Nothing in our approach is particular to natural numbers and we believe the approach extends to a more general setting of recursive types.

Types and terms. The **types** of PCF are generated by the type N of natural numbers, and function types $A \to B$. A **context** is a list of types denoted by Γ, Δ. The empty context is []. We define the set of **raw** terms by the following grammar, where $n \in \mathbb{N}$ is a natural number.

$$a, b, c ::= \underline{n} \mid \mathrm{app}\, a\, b \mid \lambda\, a \mid 0 \mid \mathrm{suc}\, a \mid \mathrm{case}\, a\, b\, c \mid \mathrm{fix}\, a \mid \Omega$$

Note that we use *de Bruijn indices*: the variable $\underline{n}$ refers to (if it exists) the first λ encountered after crossing n occurrences of λ when going up the syntax tree from the variable to the root. We include the non-terminating program Ω. Terms are assigned types using standard typing rules, displayed in Figure 1.

$$\frac{}{A_{n-1}, \ldots, A_0 \vdash \underline{i} : A_i} \qquad \frac{\Gamma \vdash b : A \to B \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathrm{app}\, b\, a : B} \qquad \frac{\Gamma, A \vdash b : B}{\Gamma \vdash \lambda\, b : A \to B} \qquad \frac{}{\Gamma \vdash 0 : \mathrm{N}}$$

$$\frac{\Gamma \vdash a : \mathrm{N} \qquad \Gamma \vdash b : A \qquad \Gamma, \mathrm{N} \vdash c : A}{\Gamma \vdash \mathrm{case}\, a\, b\, c : A} \qquad \frac{\Gamma \vdash a : \mathrm{N}}{\Gamma \vdash \mathrm{suc}\, a : \mathrm{N}} \qquad \frac{\Gamma, A \vdash c : A}{\Gamma \vdash \mathrm{fix}\, c : A} \qquad \frac{}{\Gamma \vdash \Omega : \mathrm{N}}$$

Fig. 1. Typing rules for PCF

Substitution. A **substitution** is a sequence of terms, written $\Gamma \vdash \langle a_{n-1}, \ldots, a_0 \rangle : A_{n-1}, \ldots, A_0$ if for all $0 \leq i \leq n-1$ we have $\Gamma \vdash a_i : A_i$. For $|\Gamma| = n$ we define abbreviations $\mathrm{id}_\Gamma = \langle \underline{n-1}, \ldots, \underline{0} \rangle$, $\mathrm{p}_{\Gamma,A} = \langle \underline{n}, \ldots, \underline{1} \rangle$ and $\mathrm{q}_{\Gamma,A} = \underline{0}$. We have $\Gamma \vdash \mathrm{id}_\Gamma : \Gamma$ and $\Gamma, A \vdash \mathrm{p}_{\Gamma,A} : \Gamma$. We will often just write id, p, and q.

$$\begin{array}{lll}
(\mathrm{case}\, a\, b\, c)[\gamma] = \mathrm{case}\, a[\gamma]\, b[\gamma]\, c[\langle \gamma \circ \mathrm{p}, \mathrm{q} \rangle] & (\mathrm{suc}\, a)[\gamma] = \mathrm{suc}\, (a[\gamma]) & 0[\gamma] = 0 \\
(\mathrm{app}\, f\, a)[\gamma] = \mathrm{app}\, (f[\gamma])\, (a[\gamma]) & (\mathrm{fix}\, c)[\gamma] = \mathrm{fix}\, (c[\langle \gamma \circ \mathrm{p}, \mathrm{q} \rangle]) & \Omega[\gamma] = \Omega \\
(\lambda\, f)[\gamma] = \lambda\, (f[\langle \gamma \circ \mathrm{p}, \mathrm{q} \rangle]) & &
\end{array}$$

Fig. 2. Substitution on term constructors

We define the action $a[\gamma]$ of a substitution $\Delta \vdash \gamma : \Gamma$ on a term $\Gamma \vdash a : A$ by induction on a, with $\underline{i}[\langle a_{n-1}, \ldots, a_0 \rangle] = a_i$ for variables and following the rules

of Figure 2 for term constructors. The composition of substitutions is defined by $\langle a_{n-1}, \ldots, a_0 \rangle \circ \gamma = \langle a_{n-1}[\gamma], \ldots, a_0[\gamma] \rangle$. When composing substitutions we will sometimes omit the operator $\circ$ and just use juxtaposition. By abuse of notation, we write $\langle \gamma, a \rangle$ for the sequence obtained by adding a at the end of γ.

$$\begin{array}{rcl}
\mathrm{app}\,(\lambda\, a)\, b & \rightarrow_{\beta_1} & a[\langle \mathrm{id}, b\rangle] \\
\mathrm{case}\, 0\, b\, c & \rightarrow_{\beta_2} & b \\
\mathrm{case}\,(\mathrm{suc}\, a)\, b\, c & \rightarrow_{\beta_3} & c[\langle \mathrm{id}, a\rangle] \\
\mathrm{fix}\, f & \rightarrow_{\delta} & f[\langle \mathrm{id}, \mathrm{fix}\, f\rangle] \\
\Omega & \rightarrow_{\Omega} & \Omega
\end{array}
\qquad
\begin{array}{rcl}
c & \rightarrow_{\eta_1} & \lambda\,(\mathrm{app}\,(c[\mathrm{p}])\,\mathrm{q}) \\
a & \rightarrow_{\eta_2} & \mathrm{case}\, a\, 0\,(\mathrm{suc}\,\mathrm{q}) \\
\mathrm{case}\,(\mathrm{case}\, a\, b\, f)\, b'\, f' & \rightarrow_{\gamma_1} & \mathrm{case}\, a\,(\mathrm{case}\, b\, b'\, f) \\
 & & \qquad (\mathrm{case}\, f\,(b'[\mathrm{p}]) \\
 & & \qquad\qquad (f'[\langle \mathrm{pp}, \mathrm{q}\rangle])) \\
\mathrm{app}\,(\mathrm{case}\, a\, b\, f)\, c & \rightarrow_{\gamma_2} & \mathrm{case}\, a\,(\mathrm{app}\, b\, c)(\mathrm{app}\, f\,(c[\mathrm{p}]))
\end{array}$$

Fig. 3. Reduction rules for PCF

Reduction. We equip PCF with the (context closure of the) reduction rules in Figure 3. The rules are *typed*: a reduction applies when both sides typecheck. We write $\approx$ for **convertibility**, i.e. the contextual equivalence closure of these relations. The two columns of Figure 3 will be treated quite differently in our development. The left hand side contains the *computation* rules which are used for evaluating a closed term of ground type. The right hand side contains η-*expansion* and *commutating conversions*, which are additional rules needed in Section 3.4 for transforming an arbitrary term to its *PCF Böhm tree.*

We will also need *head reduction.* For that, define **head environments** as:

$$H[] ::= [] \mid \mathrm{case}\ H[]\ b\ c \mid \mathrm{app}\ H[]\ b \mid \lambda\ H[]$$

Head reduction $\rightarrow_h$ is $H[\rightarrow_\alpha]$ for $\alpha \in \{\beta_1, \beta_2, \beta_3, \delta, \Omega\}$ a computation rule. Head reduction is deterministic: for each term, at most one head reduction applies. A term a is a **head normal form** if it is $\rightarrow_h$-normal. If $a \rightarrow_h^* a'$ where a' is $\rightarrow_h$-normal, then a' is the **head normal form** of a.

Non-dependent cwfs. Usually a model of PCF is a cartesian closed category with extra structure. We prefer to use a notion of model which separates contexts and types, and thus more closely matches the structure of our syntax. For that we use categories with families (cwfs) [12]. Cwfs provide a notion of model of dependent type theory which is both close to the syntax, and completely algebraic: it can be presented as a generalized algebraic theory in the sense of Cartmell [7].

Since we do not have dependent types we use *non-dependent cwfs*, that is cwfs where the set of types $\mathrm{Type}(\Gamma)$ does not depend on the context Γ.

Definition 1. *A* ***non-dependent cwf*** *consists of a set of* types Type, *plus:*

- *A base category* $\mathbb{C}$. *Its objects represent* contexts *and its morphisms represent* substitutions. *We write* $\Delta \vdash \gamma : \Gamma$ *for a context morphism from* Δ *to* Γ. *The identity is written* $\Gamma \vdash \mathrm{id}_\Gamma : \Gamma$ *and composition is written* $\gamma \circ \delta$, *or just* $\gamma\delta$.

- *A functor* $T : \mathbb{C}^{op} \to \mathbf{Set}^{\mathrm{Type}}$. *For each context* Γ *and type* A *this gives a set* $T(\Gamma)(A)$, *written* $\Gamma \vdash A$, *of terms of type* A *in context* Γ. *We write* $\Gamma \vdash a : A$ *for* $a \in \Gamma \vdash A$. *For* $\gamma : \Delta \to \Gamma$ *a morphism in* $\mathbb{C}$, *then* $T(\gamma)(A) : \Gamma \vdash A \to \Delta \vdash A$ *provides a* substitution *operation, written* $\Delta \vdash a[\gamma] : A$.
- *A* terminal object $[]$ *of* $\mathbb{C}$ *which represents the* empty context *and a terminal morphism* $\langle\rangle : \Delta \to []$ *which represents the* empty substitution.
- *A* context comprehension *which to an object* Γ *in* $\mathbb{C}$ *and a type* $A \in \mathrm{Type}$ *associates an object* $\Gamma{\cdot}A$ *of* $\mathbb{C}$, *a morphism* $\mathrm{p}_{\Gamma,A} : \Gamma{\cdot}A \to \Gamma$ *of* $\mathbb{C}$ *and a term* $\Gamma{\cdot}A \vdash \mathrm{q}_{\Gamma,A} : A$ *such that the following universal property holds: for each object* Δ *in* $\mathbb{C}$, *morphism* $\gamma : \Delta \to \Gamma$, *and term* $a : \Delta \vdash A$, *there is a unique morphism* $\theta = \langle\gamma, a\rangle : \Delta \to \Gamma{\cdot}A$, *such that* $\mathrm{p}_{\Gamma,A} \circ \theta = \gamma$ *and* $\mathrm{q}_{\Gamma,A}[\theta] = a$.

Democratic [8] non-dependent cwfs are equivalent to categories with finite products, but mimic more closely the structure of syntax. To describe the intended models of PCF we equip non-dependent cwfs with more structure, as follows.

Definition 2. *A non-dependent cwf* ***supports*** *PCF, or is a* ***pcf-cwf****, iff it is closed under the types and term constructors of Figure 1 (other than variable) and validates the equations and reduction rules of Figures 2 and 3, which have been chosen to make formal sense in an arbitrary non-dependent cwf as well as in the syntax of PCF. (A de Bruijn variable* $\underline{n}$ *in PCF is interpreted as an iterated projection* $\mathrm{q}[\mathrm{p}^n]$ *and all other syntactic constructs have a direct interpretation.)*

2.2 Innocent Strategies for PCF

We start with a simple operational presentation of the pcf-cwf of *innocent well-bracketed strategies* playing on (arenas for) PCF types.

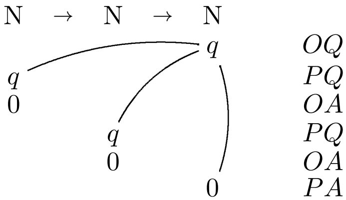

Fig. 4. A play on N → N → N

Game semantics formalize the intuition that a program is a strategy, and that execution is a play of this strategy against its execution environment according to rules determined by the type. For instance, a dialogue of type N → N → N could be that of Figure 4. Moves are either Questions (Q) or Answers (A) by either Player (P) or Opponent (O). Questions correspond to variable calls, and Answers to evaluation to terminating calls. The lines between moves are *justification pointers*: they convey information about *thread indexing* – here they are redundant, but become necessary on higher types. The diagram above should be read as follows: Opponent asks for the output of a function $f : \mathrm{N} \to \mathrm{N} \to \mathrm{N}$. This function f (Player) proceeds to interrogate its first argument. This argument is part of the execution environment of f, so it is played by Opponent – if it evaluates to 0, then f evaluates its second argument. If it also evaluates to 0, then f answers 0. A *strategy* is a collection of such interactions, informing the full behaviour of a program under execution.

The dialogue above, considered as a branch of a strategy, can also be represented syntactically by the term $\vdash \lambda\ (\lambda\ (\text{case}\ \underline{1}\ (\text{case}\ \underline{0}\ 0\ \Omega)\ \Omega)) : \mathrm{N} \to \mathrm{N} \to \mathrm{N}$, where the occurrences of Ω indicate parts of the term for which the dialogue above gives no information. In general a dialogue such as the above where *(1)* Opponent moves are *justified* by their immediate predecessor and *(2)* every Answer is justified by the last unanswered Question, can always be represented syntactically as a partial term – such dialogues are usually called *well-bracketed P-views*. In our example, the dialogue is a branch of the strategy for *left-plus* that first evaluates its first argument (and copies lazily each successor), then the second. The strategy for left-plus contains countably many dialogues. As a typical example, we show the dialogue for the computation of 2 + 2 in Figure 5.

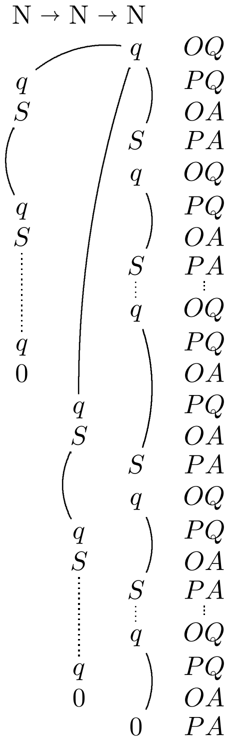

Fig. 5. *left-plus*

Representing these plays syntactically and pasting them together, one obtains the infinitary term $\vdash$ plus : $\mathrm{N} \to \mathrm{N} \to \mathrm{N}$ defined by

$$\begin{aligned} \text{plus} &= \lambda\,(\lambda\,(\text{case}\,\underline{1}\,\text{cc}\,(\text{suc}\,\text{plus}_1))) \\ \text{cc} &= \text{case}\,\underline{0}\,0\,(\text{suc}\,\text{cc}) \\ \text{plus}_n &= \text{case}\,\underline{0}\,(\text{case}\,\underline{n}\,0\,(\text{suc}\,\text{cc}))\,(\text{suc}\,\text{plus}_{n+1}) \end{aligned}$$

Here cc stands for *copycat*, the back-and-forth copying performed by the strategy when evaluating its second argument. Note that the de Bruijn indices grow. Indeed the second argument of case is an abstraction which binds a new variable, so the address of the second argument of f corresponds to larger and larger integers. Alternatively one can say that each Opponent Answer S in the plays provide a possible justifier that has to be crossed before reaching the second argument of f. We call the infinitary term above the *PCF Böhm tree* of *left-plus.*

The ideas above can easily be extended to represent syntactically (as an infinitary term) any *innocent well-bracketed strategy*, i.e. set of well-bracketed P-views on first-order PCF types $\mathrm{N} \to \ldots \to \mathrm{N} \to \mathrm{N}$. In general however, types of PCF have the form $A_{n-1} \to \ldots \to A_0 \to \mathrm{N}$, where each A_i has itself the form $A_{i,p_i-1} \to \ldots \to A_{i,0} \to \mathrm{N}$. In that case the discussion above still applies to the restriction of a strategy to its "first-order sub-type", which provides the backbone of a PCF Böhm tree. However, Player also needs to specify its behaviour should Opponent interrogate any of the arguments $A_{i,j}$ of a Player Question at the root of A_i. For each such Opponent Question, following (co-)inductively the same reasoning, a strategy for Player would inform a new strategy playing on $A_{n-1} \to \ldots \to A_0 \to A_{i,j}$ that can be set as an argument to the variable call matching the Player Question under consideration. This presentation of a strategy is called its *PCF Böhm tree.*

PCF Böhm trees. We now describe the terms obtained by this process. There are two kinds. On the one hand we define the *neutral PCF Böhm trees* which specify one Player Question (a variable call) along with the Player sub-strategies for the *arguments* of this call. On the other hand we define *PCF Böhm trees* wrap neutral PCF Böhm trees in a case statement, specifying the Player sub-strategies to play if Opponent answers 0 or S. We write $\Gamma \vdash_{\mathrm{Ne}} e : A$ if e is a *finite neutral Böhm tree* and $\Gamma \vdash_{\mathrm{Bt}} t : A$ if t is a *finite PCF Böhm trees* of type A in context Γ. They are defined as follows:

$$\frac{}{A_{n-1}, \ldots, A_i, \ldots, A_0 \vdash_{\mathrm{Ne}} \underline{i} : A_i} \qquad \frac{\Gamma \vdash_{\mathrm{Ne}} e : A \to B \quad \Gamma \vdash_{\mathrm{Bt}} t : A}{\Gamma \vdash_{\mathrm{Ne}} \mathrm{app}\, e\, t : B} \qquad \frac{\Gamma, A \vdash_{\mathrm{Bt}} t : B}{\Gamma \vdash_{\mathrm{Bt}} \lambda\, t : A \to B}$$

$$\frac{\Gamma \vdash_{\mathrm{Bt}} a : \mathrm{N}}{\Gamma \vdash_{\mathrm{Bt}} \mathrm{suc}\, a : \mathrm{N}} \qquad \frac{\Gamma \vdash_{\mathrm{Ne}} e : \mathrm{N} \quad \Gamma \vdash_{\mathrm{Bt}} t : \mathrm{N} \quad \Gamma, \mathrm{N} \vdash_{\mathrm{Bt}} t' : \mathrm{N}}{\Gamma \vdash_{\mathrm{Bt}} \mathrm{case}\, e\, t\, t' : \mathrm{N}} \qquad \frac{}{\Gamma \vdash_{\mathrm{Bt}} \Omega : \mathrm{N}} \qquad \frac{}{\Gamma \vdash_{\mathrm{Bt}} 0 : \mathrm{N}}$$

Fig. 6. Typing rules for finite PCF Böhm trees

These two sets are partially ordered by (the contextual closure of) $\Omega \leq a$ for all a, and their infinitary counterparts are defined as ideals (non-empty downward directed sets) for this order [21] – we will often keep this ideal completion implicit. From now on all PCF Böhm trees are considered infinitary,

The representation process outlined above yields a PCF Böhm tree in this formal sense. Moreover, this PCF Böhm tree is an infinitary PCF term, so can be sent back to a strategy using (by continuity) the usual game-theoretic interpretation of terms. This operation is inverse to the reification process described above, yielding an isomorphism between PCF Böhm trees and innocent strategies. In fact the correspondence is so direct that in the remainder of this paper we will identify them, and simply consider PCF Böhm trees as our representation of innocent strategies.

This correspondence is nothing new – it is one of the fundamental properties of the game model leading to definability: see *eg* Theorem 5.1 in [11]. It is easy to adapt this to lazy PCF. It is implicit in McCusker's definability process for a language with lazy recursive types (see Proposition 5.8 in [18]). We have not spelled out this connection more formally, since it would take too much space to introduce the required game-theoretic machinery.

2.3 Computing Operations on Strategies by nbe

To conclude this section we present (one aspect of) our result: that we can compute operations on innocent well-bracketed strategies, regarded as PCF Böhm trees, by nbe. As an example, a Haskell program that, given two infinite PCF Böhm trees as input, produces lazily the application of one to the other. All other operations of pcf-cwfs can be defined in an analogous way. In the remaining sections we will then show that application, and all the other pcf-cwf operations,

satisfy the expected equations, and that we get a pcf-cwf **PCFInn** of PCF-Böhm trees (or innocent, well-bracketed strategies).

The Haskell datatype for representing PCF Böhm trees and types of PCF is:

```
data Tm = ZeroTm | SuccTm Tm | LamTm Tm | CaseTm Tm Tm Tm
        | VarTm Int | AppTm Tm Tm
data Ty = Nat | Arr Ty Ty
```

It should be clear to the reader how inhabitants of `Tm` include representatives for PCF Böhm trees, hence for innocent strategies. The first step is to interpret PCF Böhm trees in a semantic domain `D`, which is a hoas version of `Tm`:

```
data D = ZeroD | SuccD D | LamD (D -> D) | CaseD D D (D -> D)
       | VarD Int | AppD D D
```

We then introduce two semantic operations `appD :: D -> D -> D` for application, and `caseD :: D -> D -> (D -> D) -> D` for case construction.

```
appD (LamD f)      d' = f d'
appD (CaseD e d f) d' = CaseD e (appD d d') (\x -> appD (f x) d')
appD (VarD n)      d' = AppD (VarD n) d'
appD (AppD e d)    d' = AppD (AppD e d) d'

caseD ZeroD         e' f' = e'
caseD (SuccD d)     e' f' = f' d
caseD (CaseD e d f) e' f' = CaseD e (caseD d e' f')
                                (\x -> caseD (f x) e' f')
caseD (AppD e d)    e' f' = CaseD (AppD e d) e' f'
caseD (VarD i)      e' f' = CaseD (VarD i) e' f'
```

We can interpret a PCF Böhm tree in the semantic domain `D` by the function `eval :: Tm -> [D] -> D`. It takes a term and interprets it in a given environment, encoded as a list of elements of the domain.

```
eval ZeroTm            env = ZeroD
eval (SuccTm t)        env = SuccD (eval t env)
eval (LamTm t)         env = LamD (\x -> eval t (x:env))
eval (CaseTm t1 t2 t3) env = caseD (eval t1 env) (eval t2 env)
                                (\x -> eval t3 (x:env))
eval (VarTm i)         env = env !! i
eval (AppTm t1 t2)     env = appD (eval t1 env) (eval t2 env)
```

An element of the semantic domain can be read back to a term using the function `readbackD :: Int -> D -> Tm`, defined as follows.

```
readbackD n ZeroD         = ZeroTm
readbackD n (SuccD d)     = SuccTm (readbackD n d)
readbackD n (LamD f)      = LamTm (readbackD (n+1) (f (VarD n)))
readbackD n (CaseD e d f) = CaseTm (readbackD n e) (readbackD n d)
                                (readbackD (n+1) (f (VarD n)))
readbackD n (VarD i)      = VarTm (n-i-1)
readbackD n (AppD e d)    = AppTm (readbackD n e) (readbackD n d)
```

Finally, we obtain `readback :: Int -> ([D] -> D) -> Tm` as

```
readback n f          = readbackD n (f [VarD (n-i-1) | i <- [0..(n-1)]])
```

The application of a PCF Böhm tree (innocent strategy) $\Gamma \vdash_{\mathrm{Bt}} t : A \to B$ to $\Gamma \vdash_{\mathrm{Bt}} t' : A$ with $|\Gamma| = n$ can now be computed lazily as `app n t t'`, where `app :: Int -> Tm -> Tm -> Tm` is defined by

```
app n t t' = readback n (\x -> appD (eval t x) (eval t' x))
```

We will in the following sections prove that this simple definition computes the claimed result. We can define functions for all other pcf-cwf combinators in a similar way, and prove that they satisfy the pcf-cwf-laws. In this way we get a pcf-cwf **PCFInn** which is an alternative nbe-based presentation of the innocent strategies model of PCF – these will come as a by-product of a nbe procedure producing the innocent strategy for a term.

3 The Domain Interpretation

To prove the correctness of the nbe program we use its denotational semantics in Scott domains. We first show that the interpretation function is sound w.r.t. syntactic conversion and computationally adequate. Then we show that the η-expanded interpretation of a term is equal to that of its PCF Böhm tree.

3.1 A Semantic Domain

D *and its combinators.* The Haskell datatype `D` can be interpreted as a Scott domain D which is the solution of the domain equation given by the constructors:

$$\mathrm{Lam_D} : (\mathrm{D} \to \mathrm{D}) \to \mathrm{D} \qquad \mathrm{Suc_D} : \mathrm{D} \to \mathrm{D} \qquad 0_\mathrm{D} : \mathrm{D}$$
$$\mathrm{App_D} : \mathrm{D} \to \mathrm{D} \to \mathrm{D} \qquad \mathrm{Case_D} : \mathrm{D} \to \mathrm{D} \to (\mathrm{D} \to \mathrm{D}) \to \mathrm{D} \qquad \mathrm{Var_D} : \mathrm{N} \to \mathrm{D}$$

We write Ω_D for the bottom element. The two semantic operations $\mathrm{app_D} : \mathrm{D} \to \mathrm{D} \to \mathrm{D}$ and $\mathrm{case_D} : \mathrm{D} \to \mathrm{D} \to (\mathrm{D} \to \mathrm{D}) \to \mathrm{D}$ are defined as their Haskell counterparts in Section 2.3.

Interpretation of PCF. Let Tm be the set of elements of the raw syntax of PCF. If E is a set, $[E]$ denotes the set of lists of elements of E. As for substitutions, we write $\langle\rangle$ for the empty list. If $\rho \in [\mathrm{D}]$ and $d \in \mathrm{D}$, we write $\rho :: d$ for the addition of d at the end of ρ[1] Finally, $\rho(i)$ is the i-th element of ρ *starting from the right and from* 0, if it exists, and Ω_D otherwise.

The **interpretation of PCF** is defined as a function $[\![-]\!] : \mathrm{Tm} \to [\mathrm{D}] \to D$:

$$[\![\mathrm{fix}\, f]\!]\ \rho = \textstyle\bigsqcup_{n\in\mathbb{N}} (\lambda d.\ [\![f]\!]\ (\rho :: d))^n (\Omega_\mathrm{D}) \qquad [\![\underline{n}]\!]\ \rho = \rho(n)$$
$$[\![\mathrm{app}\, s\, t]\!]\ \rho = \mathrm{app_D}\ ([\![s]\!]\ \rho)\ ([\![t]\!]\ \rho) \qquad [\![\mathrm{suc}\, a]\!]\ \rho = \mathrm{Suc_D}\ ([\![a]\!]\ \rho)$$
$$[\![\lambda\, t]\!]\ \rho = \mathrm{Lam_D}\ (\lambda x.[\![t]\!]\ (\rho :: x)) \qquad [\![\Omega]\!]\ \rho = \Omega_\mathrm{D}$$
$$[\![\mathrm{case}\, a\, b\, c]\!]\ \rho = \mathrm{case_D}\ ([\![a]\!]\ \rho)\ ([\![b]\!]\ \rho)\ (\lambda x.\ [\![c]\!]\ (\rho :: x)) \qquad [\![0]\!]\ \rho = 0_\mathrm{D}$$

[1] This notational difference from the Haskell program ensures that the order of environments matches that of the context.

It is extended to substitutions $\gamma = \langle a_{n-1}, \ldots, a_0 \rangle$ by $[\![\gamma]\!]\,\rho = \langle [\![a_{n-1}]\!]\,\rho, \ldots, [\![a_0]\!]\,\rho \rangle$.

3.2 Soundness for Conversion

Here, we prove that the interpretation described above is sound with respect to conversion in PCF. Reduction rules of PCF come in three kinds: the *computation rules* ($\beta_1, \beta_2, \beta_3, \delta, \Omega$), the *commutation rules* (γ_1, γ_2) and the *η-expansion rules* (η_1, η_2). Soundness w.r.t. computation rules follows from standard (and simple) verifications, and we verify only the commutation and η-expansion rules.

Commutation rules. The interpretation of PCF validates γ_1 and γ_2.

Lemma 1. *For all $d_1, d_2, d_3 \in \mathrm{D}$ and $f, f_1, f_2 \in \mathrm{D} \to \mathrm{D}$, we have:*

$$\mathrm{app}_\mathrm{D}\,(\mathrm{case}_\mathrm{D}\,d_1\,d_2\,f)\,d_3 = \mathrm{case}_\mathrm{D}\,d_1\,(\mathrm{app}_\mathrm{D}\,d_2\,d_3)\,(\lambda x.\,\mathrm{app}_\mathrm{D}\,(f\,x)\,d_3)$$
$$\mathrm{case}_\mathrm{D}\,(\mathrm{case}_\mathrm{D}\,d_1\,d_2\,f_1)\,d_3\,f_2 = \mathrm{case}_\mathrm{D}\,d_1\,(\mathrm{case}_\mathrm{D}\,d_2\,d_3\,f_2)\,(\lambda x.\,\mathrm{case}_\mathrm{D}\,(f_1\,x)\,d_3\,f_2)$$

Proof. We apply Pitts' co-induction principle [19] for proving inequalities in recursively defined domains. The proof proceeds by defining a relation R containing the identity relation on D and all pairs (for $d_1, d_2, d_3 \in \mathrm{D}$ and $f \in \mathrm{D} \to \mathrm{D}$):

$$(\mathrm{app}_\mathrm{D}\,(\mathrm{case}_\mathrm{D}\,d_1\,d_2\,f)\,d_3, \mathrm{case}_\mathrm{D}\,d_1\,(\mathrm{app}_\mathrm{D}\,d_2\,d_3)\,(\lambda x.\,\mathrm{app}_\mathrm{D}\,(f\,x)\,d_3))$$

Then, R is a bisimulation. For $d_1 = \Omega_\mathrm{D}, \mathrm{Var}_\mathrm{D}\,i, \mathrm{App}_\mathrm{D}\,d_1'\,d_2', \mathrm{Lam}_\mathrm{D}\,f', 0_\mathrm{D}$ or $\mathrm{Suc}_\mathrm{D}\,d'$, both sides evaluate to the same, so the bisimulation property is trivial. For $d_1 = \mathrm{Case}_\mathrm{D}\,d_1'\,d_2'\,f'$, by direct calculations both sides start with Case_D, followed by arguments related by R. By Pitts' result [19] the equality follows.

η-expansion rules. Note that these cannot be true in general since the interpretation ignores type information. So we define a semantic version of η-expansion:

$$\eta_N\,d = \mathrm{case}_\mathrm{D}\,d\,0_\mathrm{D}\,(\lambda x.\,\eta_N\,x) \qquad \eta_{A\to B}\,d = \mathrm{Lam}_\mathrm{D}\,(\lambda x.\,\eta_B\,\mathrm{app}_\mathrm{D}\,d\,(\eta_A\,x))$$

It extends to environments by $\eta_{[]}\,\rho = []$, $\eta_{\Gamma,A}\,[] = (\eta_\Gamma\,[]) :: \Omega_\mathrm{D}$ and $\eta_{\Gamma,A}\,(\rho :: d) = (\eta_\Gamma\,\rho) :: (\eta_A\,d)$. For $f : [\mathrm{D}] \to \mathrm{D}$ we define $\eta_{\Gamma\vdash A}\,f = \lambda\rho.\,\eta_A\,(f\,(\eta_\Gamma\,\rho))$.

From semantic η-expansion we get a *typed* notion of equality *up to η-expansion* in the domain: for a type A and elements $d, d' \in \mathrm{D}$ we write $d =_A d'$ iff $\eta_A\,d = \eta_A\,d'$. This generalizes to $f =_{\Gamma\vdash A} f'$ in the obvious way. We now prove that *syntactic* η-expansion is validated up to *semantic* η-expansion.

It is easy to see that for $\Gamma \vdash b : A \to B$ we have $[\![b]\!] =_{\Gamma\vdash A\to B} [\![\lambda\,(\mathrm{app}\,b[\mathrm{p}]\,\underline{0})]\!]$ and also to verify the η-rule for N. However, reduction rules are closed under context, so we need to check that typed equality is a congruence.

This relies on the fact that the interpretation of terms cannot distinguish an input from its η-expansion. To prove that we start by defining a realizability predicate on D, by $d \Vdash \mathrm{N}$ for all $d \in \mathrm{D}$, and $d \Vdash A \to B$ iff for all $d' \Vdash A$, *(1)* $\mathrm{app}_\mathrm{D}\,d\,d' =_B \mathrm{app}_\mathrm{D}\,d\,(\eta_A\,d')$ and *(2)* $\mathrm{app}_\mathrm{D}\,d\,d' \Vdash B$. This predicate generalizes

to contexts (written $\rho \Vdash \Gamma$) in the obvious way. We observe by induction on A that η-expanded elements of D are realizers: for all $d \in \mathrm{D}$, $\eta_A\, d \Vdash A$. But the interpretation of terms, despite not being η-expanded, also satisfy it. Indeed we prove, by induction on typing judgments, the following adequacy lemma.

Lemma 2. *For $\Gamma \vdash a : A$, $\rho \Vdash \Gamma$, then (1) $[\![a]\!]\,\rho =_A [\![a]\!]\,(\eta_\Gamma\,\rho)$ and (2) $[\![a]\!]\,\rho \Vdash A$.*

It is then immediate that the interpretation of term constructors preserves typed equality. All conversion rules hold up to typed equality in D, which is a congruence w.r.t. the interpretation of terms. Putting it all together we conclude:

Proposition 1 (Soundness). *For $\Gamma \vdash a, a' : A$ with $a \approx a'$, $[\![a]\!] =_{\Gamma \vdash A} [\![a']\!]$.*

3.3 Computational Adequacy

As a step towards our main result, we prove computational adequacy:

Proposition 2. *If $\Gamma \vdash a : A$ and $[\![a]\!] \neq_{\Gamma \vdash A} \Omega_{\mathrm{D}}$ then a has a head normal form.*

First, we reduce the problem to closed terms by noting two properties:

- Firstly, $\Gamma, A \vdash b : B$ has a head normal form iff $\lambda\, b$ has,
- Secondly, $[\![b]\!] =_{\Gamma, A \vdash B} \Omega_{\mathrm{D}}$ iff $[\![\lambda\, b]\!] =_{\Gamma \vdash A \to B} \Omega_{\mathrm{D}}$, as can be checked by a simple calculation.

So we abstract all free variables of a term and reason only on closed terms.

We now aim to prove it for closed terms. The proof has two steps: *(1)* we prove it for closed terms of ground type using logical relations, and *(2)* we deduce it for closed terms of higher-order types. To obtain *(2)* from *(1)* we will need to temporarily enrich the syntax with an *error* constant $*$. This $*$ has all types – we write $\Gamma \vdash_* a : A$ for typing judgments in the extended syntax. We also add two head reductions $\mathrm{app}\; *\; a \to_h *$ and $\mathrm{case}\; *\; a\, b\, c \to_h *$.

We also need to give an interpretation of $*$ in D. At this point it is tempting to enrich the domain D with a constructor for $*$. Fortunately we can avoid that; indeed the reader can check that setting $[\![*]\!]\,\rho = \mathrm{Case}_{\mathrm{D}}\, \Omega_{\mathrm{D}}\, \Omega_{\mathrm{D}}\, (\lambda x.\, \Omega_{\mathrm{D}}) = *_{\mathrm{D}}$, the interpretation validates the two reduction rules above. Note that the term $*$ is only an auxiliary device used in this section: we will never attempt to apply nbe on a term with error, so this coincidence will be harmless.

Ground type. We first define our logical relations.

Definition 3. *We define a relation $\sim^n_{\mathrm{N}}$ between closed terms $\vdash_* a : \mathrm{N}$ and elements of D by induction on n. First $a \sim^0_{\mathrm{N}} d$ always. Then, $a \sim^{n+1}_{\mathrm{N}} d$ iff either $d = \Omega_D$, or $a \to^*_h 0$ and $d = 0_{\mathrm{D}}$, or $a \to^*_h *$ and $d = *_{\mathrm{D}}$, or finally, if $a \to^*_h \mathrm{suc}\, a'$ and $d = \mathrm{Suc}_{\mathrm{D}}\, d'$ and $a' \sim^n_{\mathrm{N}} d'$. We then define $a \sim_N d$ iff for all $n \in \mathbb{N}$, $a \sim^n_N d$. Finally, $b \sim_{A \to B} d$ iff for any $a \sim_A e$, $\mathrm{app}\, b\, a \sim_B \mathrm{app}_{\mathrm{D}}\, d\, e$.*

This relation is closed under backward head reduction, and satisfies the continuity property that for any ω-chain $(d_i)_{i \in \mathbb{N}}$, if $a \sim_A d_i$ for all i then $a \sim_A \sqcup_i d_i$. The fundamental lemma of logical relations follows by induction on a.

Lemma 3. *For any term $\Gamma \vdash_* a : A$, for any $\delta \sim_\Gamma \rho$, we have $a[\delta] \sim_A [\![a]\!]\,\rho$.*

By definition of $\sim_A$, computational adequacy follows for closed terms of type N.

Higher-order types. Suppose $\vdash a : A_{n-1} \to \ldots \to A_0 \to \mathrm{N} = A$ satisfies $[\![a]\!] \neq_A \Omega_{\mathrm{D}}$. We need to show that a has a head normal form, but our previous analysis only applies to terms of ground type. By hypothesis we know that for $[\![a]\!]$ there are some arguments $d_{n-1}, \ldots, d_0$ making $[\![a]\!]$ non-bottom. However, in order to apply our earlier result for ground type, we need to find syntactic counterparts to $d_{n-1}, \ldots, d_0$ – and there is no reason why those would exist. So instead we replace the d_is with $*_{\mathrm{D}}$, which *does* have a syntactic counterpart.

The core argument is that replacing arguments of $[\![a]\!]$ with $*_{\mathrm{D}}$ only increases chances of convergence. To show that we introduce:

Definition 4. *We define* $\lesssim^n_{\mathrm{N}} \subseteq \mathrm{D}^2$ *by induction on* n*. Let* $d_1 \lesssim^0_{\mathrm{N}} d_2 \Leftrightarrow \top$ *and*

$$d_1 \lesssim^{n+1}_{\mathrm{N}} d_2 \Leftrightarrow \begin{cases} \textit{If } d_1 = d_2 = 0_{\mathrm{D}} \\ \textit{If } d_1 = \mathrm{Suc}_{\mathrm{D}}\, d_1',\ d_2 = \mathrm{Suc}_{\mathrm{D}}\, d_2' \textit{ and } d_1' \lesssim^n_{\mathrm{N}} d_2' \\ \textit{If } d_1 = \Omega_D \textit{ or } d_2 \geq *_D \end{cases}$$

We set $d_1 \lesssim_{\mathrm{N}} d_2$ *iff for all* $n \in \mathbb{N}$, $d_1 \lesssim^n_{\mathrm{N}} d_2$*. We lift this to all types by stating that* $d_1 \lesssim_{A\to B} d_2$ *iff for all* $d_1' \lesssim_A d_2'$, $\mathrm{app}_{\mathrm{D}}\, d_1\, d_1' \lesssim_B \mathrm{app}_{\mathrm{D}}\, d_2\, d_2'$.

This generalizes to a relation on environments $\rho_1 \lesssim_\Gamma \rho_2$, but unlike what the notation suggests, $\lesssim_A$ is *not* an ordering: it is neither reflexive, nor transitive, nor antisymmetric. However, for all A and $d \in \mathrm{D}$ we have $\Omega_{\mathrm{D}} \lesssim_A d \lesssim_A *_{\mathrm{D}}$.

The following fundamental lemma is proved by induction on a.

Lemma 4. *For any term* $\Gamma \vdash_* a : A$*, for any* $\rho_1 \lesssim_\Gamma \rho_2$*,* $[\![a]\!]\,\rho_1 \lesssim_A [\![a]\!]\,\rho_2$*.*

Putting the ingredients above together, we prove the following lemma.

Lemma 5. *For* $\vdash_* a : A$*,* $d \lesssim_A [\![a]\!]$ *and* $d \neq_A \Omega_{\mathrm{D}}$*,* a *has a head normal form.*

Proof. For N, assume there is $d \lesssim_{\mathrm{N}} [\![a]\!]$ such that $d \neq_{\mathrm{N}} \Omega_{\mathrm{D}}$, so $d \neq \Omega_{\mathrm{D}}$. It follows by definition of $\lesssim_{\mathrm{N}}$ that either $[\![a]\!] = *_D$, or d and $[\![a]\!]$ respectively start both with 0_{D} or both with $\mathrm{Suc}_{\mathrm{D}}$. But by Lemma 3 we have $a \sim_{\mathrm{N}} [\![a]\!]$, so by definition of logical relations, a has a head normal form.

For $A \to B$, take $\vdash_* b : A \to B$ and assume there is $d \lesssim_{A\to B} [\![b]\!]$ such that $d \neq_{A\to B} \Omega_{\mathrm{D}}$. So there is $d' \in D$ such that $\mathrm{app}_{\mathrm{D}}\, d\, (\eta_A\, d') \neq_B \Omega_{\mathrm{D}}$. But we have observed above that $\eta_A\, d' \lesssim_A *_D$. Therefore, by definition of $\lesssim_{A\to B}$, we have:

$$\mathrm{app}_{\mathrm{D}}\, d\, (\eta_A\, d') \lesssim_B \mathrm{app}_{\mathrm{D}}\, [\![b]\!]\, *_D = [\![\mathrm{app}\, b\, *]\!]$$

By induction hypothesis, $\mathrm{app}\, b\, *$ has a head normal form. But head reduction is deterministic, and any potentially infinite head reduction chain on b would transport to $\mathrm{app}\, b\, *$, so b has a head normal form.

Finally, it remains to deduce computational adequacy for closed terms. But for arbitrary $\vdash_* a : A$ such that $[\![a]\!] \neq_A \Omega_{\mathrm{D}}$, by Lemma 4 we have $[\![a]\!] \lesssim_A [\![a]\!]$, so we are in the range of Lemma 5 – therefore, a has a head normal form.

3.4 PCF Böhm Trees Defined by Repeated Head Reduction

In the next section we show how the PCF Böhm tree of a term can be computed by nbe. We also show that the result of this computation coincides with the traditional way of defining a PCF Böhm tree as obtained by repeated head reduction. This definition relies on the following lemma.

Lemma 6. *The system* $\{\to_{\gamma_1}, \to_{\gamma_2}\}$ *of commutations is strongly normalizing.*

Proof. Local confluence follows from a direct analysis of the (two) critical pairs, and one gets a decreasing measure by defining $|a| = 1$ on all leaves of the syntax tree, $|\mathrm{case}\, a\, b\, f| = 2|a| + \max(|b|, |f|)$ and $|\mathrm{app}\, a\, b| = 2|a| + |b|$ and $| - |$ behaves additively on all other constructors.

The PCF Böhm tree of a term. The PCF Böhm tree $\mathrm{BT}(a)$ of a term $\Gamma \vdash a : A$ is defined as follows. If $A = \mathrm{N}$ and a has no head normal form then $\mathrm{BT}(a) = \Omega$. Otherwise we convert a to head normal form and then to $\to_{\gamma_1,\gamma_2}$-normal form a' by Lemma 6 which is still a head normal form. The only possible cases are:

- $\mathrm{BT}(a) = 0$ if $a' = 0$.
- $\mathrm{BT}(a) = \mathrm{suc}\, \mathrm{BT}(a'')$ if $a' = \mathrm{suc}\, a''$
- $\mathrm{BT}(a) = \mathrm{case}\ (\mathrm{app}\ \underline{i}\ \overrightarrow{\mathrm{BT}(a_j)})\ 0\ (\mathrm{suc}\ \mathrm{BT}(\underline{0}))$ if $a' = \mathrm{app}\, \underline{i}\, \overrightarrow{a_j}$
- $\mathrm{BT}(a) = \mathrm{case}\, (\mathrm{app}\, \underline{i}\, \overrightarrow{\mathrm{BT}(a_j)})\, \mathrm{BT}(b)\, \mathrm{BT}(c)$ if $a' = \mathrm{case}\, (\mathrm{app}\, \underline{i}\, \overrightarrow{a_j})\, b\, c$.

If $A = B \to C$ then $\mathrm{BT}(a) = \lambda\, \mathrm{BT}(\mathrm{app}\, (a[\mathrm{p}])\, \underline{0})$. ($\mathrm{BT}(a)$ could be more explicitly defined as an ideal of finite approximations, see e.g. [14] for details.)

Together with the earlier results, we get the main result of this section.

Proposition 3. *If* $\Gamma \vdash a : A$ *then* $\Gamma \vdash_{\mathrm{Bt}} \mathrm{BT}(a) : A$ *and* $\eta_{\Gamma \vdash A}\, [\![a]\!] = [\![\mathrm{BT}(a)]\!]$.

Proof. As we aim to prove the equality of two elements of D we use again Pitts' method [19] and define the following relation

$$R = \{([\![\mathrm{BT}(a)]\!]\, (\eta_\Gamma\, \rho), \eta_A\, ([\![a]\!]\, (\eta_\Gamma\, \rho))) \mid \Gamma \vdash a : A \ \&\ \rho \in [\mathrm{D}]\}$$

and show that it is a bisimulation. Proposition 1 ensures that the conversion steps needed to transform a term to its PCF Böhm tree are sound in the model, Proposition 2 ensures that both sides are Ω_{D} at the same time.

This ends the core of the technical development. The same proof scheme can be used to show that the game interpretation of Section 2.2 validates conversion to PCF Böhm trees. Details can be obtained by adapting McCusker's proof [18].

4 Game Semantics of PCF Based on nbe

Normalization by evaluation. We are now ready to show the correctness of an nbe algorithm which computes innocent strategies for *infinitary terms*. Recall that in Section 2.2 we defined PCF Böhm trees as ideals of finite PCF Böhm

trees. The same construction on arbitrary PCF terms yields a notion of infinitary term on which BT and $[\![-]\!]$ automatically extends, along with Proposition 3.

The readback function $\mathrm{R}_n : ([\mathrm{D}] \to \mathrm{D}) \to \mathrm{Tm}$ is the semantic counterpart of the Haskell `readback` function in Section 2.3. The following is proved on finitary terms by a direct induction and extends to PCF Böhm trees by continuity.

Lemma 7. *If $\Gamma \vdash_{\mathrm{Bt}} t : A$ is a PCF Böhm tree with $|\Gamma| = n$, then $\mathrm{R}_n\, [\![t]\!] = t$.*

We now define an nbe algorithm which maps a PCF term to its PCF Böhm tree, and use this lemma together with Proposition 3 to show its correctness:

Theorem 1. *Let $\Gamma \vdash a : A$ be an (infinitary) PCF term. If $\mathrm{nbe}(a) = \mathrm{R}_n\,(\eta_{\Gamma \vdash A}[\![a]\!])$, where $|\Gamma| = n$, then $\mathrm{nbe}(a) = \mathrm{BT}(a)$.*

The pcf-cwf of PCF Böhm trees. We conclude this paper by showing how to recover the Hyland-Ong game model of PCF, up to isomorphism of pcf-cwfs. If $d \in \mathrm{D}$, say that d has **(semantic) type** A iff $\eta_A\, d = d$. Likewise a function $f : [\mathrm{D}] \to \mathrm{D}$ has type $\Gamma \vdash A$ iff $\eta_{\Gamma \vdash A}\, f = f$, and a function $\gamma : [\mathrm{D}] \to [\mathrm{D}]$ has type $\Gamma \vdash \Delta$ iff it is obtained by tupling functions of the appropriate types. This generalizes to a pcf-cwf **D** having PCF contexts as objects and functions (resp. elements) of the appropriate type as morphisms (resp. terms).

As a pcf-cwf, **D** supports the interpretation of PCF. But the plain domain interpretation $[\![a]\!]$ (of Section 3) of a PCF Böhm tree $\Gamma \vdash_{\mathrm{Bt}} a : A$ automatically has semantic type $\Gamma \vdash A$, and so is a term in the sense of **D**. Furthermore, this map from PCF Böhm trees to **D** is injective by Theorem 1. Finally, the image of PCF Böhm trees in **D** is closed under all pcf-cwf operations: each of these can be replicated in the infinitary PCF syntax then normalized using nbe, yielding by Theorem 1 and Proposition 3 a PCF Böhm tree whose interpretation matches the result of the corresponding operation in **D**. So, the interpretation of PCF Böhm trees forms a sub-pcf-cwf of **D**, called **PCFInn**, satisfying:

Theorem 2. PCFInn *is isomorphic to the pcf-cwf of PCF contexts/types, and innocent well-bracketed strategies between the corresponding arenas.*

If we unfold this definition we get the Haskell program for application in Section 2.3 and similar programs for composition and the other pcf-cwf operations. The pcf-cwf laws for these programs, such as associativity of composition, β, and η, follow from the corresponding laws for their domain interpretation in **D**.

Acknowledgments. The first author acknowledges the support of the French ANR Project RAPIDO, ANR-14-CE25-0007 and the second author of the Swedish VR Frame Programme Types for Proofs and Programs.

References

1. Abel, A.: Normalization by Evaluation: Dependent Types and Impredicativity. Institut für Informatik, Ludwig-Maximilians-Universität München, Habilitation thesis (May 2013)

2. Abel, A., Aehlig, K., Dybjer, P.: Normalization by evaluation for Martin-Löf type theory with one universe. In: 23rd Conference on the Mathematical Foundations of Programming Semantics, MFPS XXIII. Electronic Notes in Theoretical Computer Science, pp. 17–40. Elsevier (2007)
3. Abramsky, S., Jagadeesan, R., Malacaria, P.: Full abstraction for PCF. Inf. Comput. 163(2), 409–470 (2000)
4. Aehlig, K., Joachimski, F.: Operational aspects of untyped normalisation by evaluation. Mathematical Structures in Computer Science 14(4) (2004)
5. Amadio, R., Curien, P.-L.: Domains and lambda-calculi, vol. 46. Cambridge University Press (1998)
6. Berger, U., Schwichtenberg, H.: An inverse to the evaluation functional for typed λ-calculus. In: Proceedings of the 6th Annual IEEE Symposium on Logic in Computer Science, Amsterdam, pp. 203–211 (July 1991)
7. Cartmell, J.: Generalized algebraic theories and contextual categories. Annals of Pure and Applied Logic 32, 209–243 (1986)
8. Clairambault, P., Dybjer, P.: The biequivalence of locally cartesian closed categories and Martin-Löf type theories. Mathematical Structures in Computer Science 24(5) (2013)
9. Clairambault, P., Murawski, A.S.: Böhm trees as higher-order recursive schemes. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, Guwahati, India, December 12-14, pp. 91–102 (2013)
10. Curien, P.-L.: Abstract Böhm trees. Mathematical Structures in Computer Science 8(6), 559–591 (1998)
11. Curien, P.-L.: Notes on game semantics. From the authors web page (2006)
12. Dybjer, P.: Internal type theory. In: Berardi, S., Coppo, M. (eds.) TYPES 1995. LNCS, vol. 1158, pp. 120–134. Springer, Heidelberg (1996)
13. Dybjer, P.: Program testing and the meaning explanations of intuitionistic type theory. In: Epistemology versus Ontology - Essays on the Philosophy and Foundations of Mathematics in Honour of Per Martin-Löf, pp. 215–241 (2012)
14. Dybjer, P., Kuperberg, D.: Formal neighbourhoods, combinatory Böhm trees, and untyped normalization by evaluation. Ann. Pure Appl. Logic 163(2), 122–131 (2012)
15. Filinski, A., Rohde, H.K.: Denotational aspects of untyped normalization by evaluation. Theor. Inf. and App. 39(3), 423–453 (2005)
16. Harmer, R., Hyland, M., Melliès, P.-A.: Categorical combinators for innocent strategies. In: 22nd IEEE Symposium on Logic in Computer Science, Wroclaw, Poland, Proceedings, pp. 379–388. IEEE Computer Society (2007)
17. Hyland, J.M.E., Luke Ong, C.-H.: On full abstraction for PCF: I, II, and III. Inf. Comput. 163(2), 285–408 (2000)
18. McCusker, G.: Games and full abstraction for FPC. Inf. Comput. 160(1-2), 1–61 (2000)
19. Pitts, A.M.: A co-induction principle for recursively defined domains. Theor. Comput. Sci. 124(2), 195–219 (1994)
20. Plotkin, G.D.: LCF considered as a programming language. Theor. Comput. Sci. 5(3), 223–255 (1977)
21. Plotkin, G.D.: Post-graduate lecture notes in advanced domain theory (incorporating the “Pisa Notes”). Dept. of Computer Science, Univ. of Edinburgh (1981)