GAME THEORETIC ANALYSIS OF CALL-BY-VALUE COMPUTATION

KOHEI HONDA NOBUKO YOSHIDA

ABSTRACT. We present a general semantic universe of call-by-value computation based on elements of game semantics, and validate its appropriateness as a semantic universe by the full abstraction result for call-by-value PCF, a generic typed programming language with call-by-value evaluation. The key idea is to consider the distinction between call-by-name and call-by-value as that of the structure of information flow, which determines the basic form of games. In this way call-by-name computation and call-by-value computation arise as two independent instances of sequential functional computation with distinct algebraic structures. We elucidate the type structures of the universe following the standard categorical framework developed in the context of domain theory. Mutual relationship between the presented category of games and the corresponding call-by-name universe is also clarified.

1. INTRODUCTION

The *call-by-value* is a mode of calling procedures widely used in imperative and functional programming languages, e.g. [1, 30], in which one evaluates arguments before applying them to a concerned procedure. The semantics of higher-order computation based on call-by-value evaluation has been widely studied by many researchers in the context of domain theory, cf. [35, 23, 32, 12, 40, 11], through which it has become clear that the semantic framework for the call-by-value computation has a basic difference from the one for call-by-name computation (see [15, 42] for introduction to the topic). The difference between the semantics of call-by-value and that of call-by-name in this context may roughly be captured as the difference in the classes of involved functions: in call-by-name, we take any continuous functions between pointed cpos, while, in call-by-value, one takes *strict* continuous functions. The latter is also equivalently presentable as partial continuous functions between (possibly bottomless) cpos. This distinction leads to a basic algebraic difference of the induced categorical universes, cf.[11, 12].

The present paper offers a semantic analysis of call-by-value computation from a different angle, based on elements of game semantics. In game semantics, computation is modelled as specific classes of interacting processes (called *strategies*), which, together with a suitable notion of composition, form a categorical universe with appropriate type structures. One may compare this approach to Böhm trees or to sequential algorithms [6, 22], in both of which computation is modelled not by set-theoretic functions of a certain kind but by objects with internal structures which reflect computational behaviour of the concerned class of computation. Game semantics has its origin in Logics [7, 10] and has been used for the semantic analysis of programming languages, especially for characterising the notion of sequentiality [8, 34]. By concentrating on specific forms of interaction which obey a few basic constraints, the approach makes it possible to extract desired classes of interacting processes at a high-level of abstraction, offering suitable semantic universes for varied calculi and programming languages, cf. [2, 3, 4, 19, 20, 24]. The forms of interaction in these universes are however inherently call-by-name: it has

---

LFCS, Department of Computer Science, University of Edinburgh. e-mail: kohei@dcs.ed.ac.uk, ny@dcs.ed.ac.uk. Supported in part by EPSRC Fellowships and JSPS Research Fellowships.

not been clear how the call-by-value computation can be captured in the setting of game semantics, in spite of its equally significant status as a mode of computation.

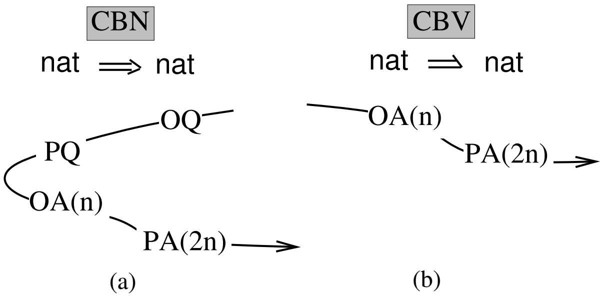

Figure 1

In the present work it will be shown that a general semantic universe of the call-by-value higher-order computation can indeed be simply constructed, employing basic elements of the foregoing game semantics, but with a key difference in the structures of interaction. More specifically, we find that the distinction between call-by-name and call-by-value in game semantics arises as the one in the form of the flow of information. Let us illustrate this point by simple examples. Figure 1 (a) depicts how a function which doubles a given natural number is modelled in the foregoing game semantics ("O" for Opponent, "P" for Player, "A" for Answer, and "Q" for Question). Computation starts when Opponent asks a question on the right, requesting an answer: then Player (the function) asks what the argument is on the left, from which the number is received, and finally it returns to the right to answer the initial question by the double of the received number. In Figure 1 (b), the same function is modelled in the call-by-value game. This time the flow starts at the *left* component, which already carries a value: then the function just returns the answer on the right. One may notice that this means the interaction should start from an *answer*, which might be regarded as an anomaly in the preceding convention in game semantics. However, it turns out that this parameter of games — whether one initiates a game by answers or by questions — is orthogonal to other basic elements of the game semantics, leading to a simple construction of a categorical universe in which representative functional calculi based on call-by-value evaluation can be faithfully interpreted. The independence of the parameter suggests we may obtain a suitable universe to model, say, imperative call-by-value computation by simply altering other parameters, cf. [4, 21]. We also note that the possibility to model "data-driven computation" in contrast to "demand-driven computation" as games is discussed in an early paper on game semantics by Abramsky and Jagadeesan [2].

The main technical contribution of the present work is the validation of the semantic exactness with which the induced universe captures the call-by-value sequential higher-order computation through the full abstraction result for the call-by-value version of PCF [35, 40], a paradigmatic functional calculus. The result seems the first one in this context[1] and is easily extendable to other languages as we shall indicate in Section 6. We also clarify the relationship between the present universe of games and the corresponding call-by-name universe by showing they are faithfully embeddable to each other. These results indicate, together with the preceding results on call-by-name PCF [3, 19], that the two basic notions of calling procedures in higher-order computation are representable in the game-based semantic framework in an exact way, and that they

---

[1] Independently and concurrently Riecke and Sandholm [38] obtained a similar result, see Section 6.

arise as two independent, though mutually related, semantic universes with equal status (which parallels the findings in domain theory, cf. [11]). It is also notable that, as we clarify later, the universe of call-by-value games assumes basic type structures which have arisen through the categorical analysis of domain-theoretic universes for call-by-value, or partial, computation, cf.[11, 12, 23, 31, 32, 36, 39], though with a strong intensional flavour. This suggests an abstract notion of "call-by-value computation" may be delineated apart from the standard domain theoretic constructions, cf. [11, 12].

The structure of games we shall use is a conservative extension of the construction by Hyland and Ong [19]. The relationship is detailed in [18].

This is an extended abstract of [18]. The reader may refer to [18] for proofs and detailed technical discussions. In the remainder, Section 2 introduces the basic notion of games and strategies. Sections 3 and 4 outline the algebraic structures of the category of games and its extensional quotient. Section 5 establishes the main result of the paper, the inequational full abstraction for call-by-value PCF. Section 6 discusses further results and remaining topics. Appendix briefly reviews call-by-value PCF.

2. GAMES AND STRATEGIES

This section introduces the basic construction of games and strategies which are to become objects and morphisms in the categorical universe. We start from sorting (the terminology is from [29]), from which call-by-value types arise as its specific subclass.

2.1. **Sorting and Type.**

(i) (sorting) A *sorting* $\mathbb{S}$ is a triple of: (1) $\underline{\mathbb{S}}$, which is a collection of mutually disjoint non-empty sets ranged over by $S, S', \ldots$ each called a *sort*, (2) $\lambda : \underline{\mathbb{S}} \to \{\, [,\ (,\ ],\ )\, \}$, a labelling function and (3) $\mathit{Obs} : \underline{\mathbb{S}} \to 2^{\underline{\mathbb{S}}}$, the justification relation (if $S' \in \mathit{Obs}(S)$ we say *S justifies S'*), where $S' \in \mathit{Obs}(S)$ implies:
- $\lambda(S) = [$ then $\lambda(S') \in \{\, (,\ ]\, \}$. Dually $\lambda(S) = ($ then $\lambda(S') \in \{\, [,\ )\, \}$.
- $\lambda(S) =]$ then $\lambda(S') = [$ always. Dually $\lambda(S) =)$ then $\lambda(S') = ($ always.

Elements of a sort are called *actions*, denoted $x, y, \ldots$, writing e.g. x^S when $x \in S$. The set of *initial sorts*, denoted $\mathrm{init}(\mathbb{S})$, is given as $\{S \mid \text{for no } S' \in \underline{\mathbb{S}}. S \in \mathit{Obs}(S')\}$.

(ii) (type) A *cbv-type*, or simply a *type*, is a sorting such that all initial sorts are labelled by "]" and any of its sorts is reachable from some initial sort, where reachability is understood regarding sortings as graphs (nodes are sorts, directed edges are given by Obs). Types are denoted by $A, B, C, \ldots$.

An action of a sort labelled by each of "(, [,],)" is called, respectively, *Player Question*, *Opponent Question*, *Player Answer*, and *Opponent Answer*, the first two collectively *Question*, the last two *Answer*, the first and third *P-action*, and the second and fourth *O-action*. Answers of initial sorts are often called *signals*. On labels we define a self-inverse function $\overline{(\cdot)}$, giving the *dual* of a label, satisfying: $\overline{[} = ($ and $\overline{]} =)$.

2.2. **Examples.** (sorting)

(i) $\mathbf{0}$ is the empty sorting, which is a type. $\mathbf{1}$ is a sorting whose unique]-labelled sort is a singleton, which is again a type. nat is made as $\mathbf{1}$ replacing a singleton with ω (the set of natural numbers), similarly bool with $\{\mathsf{true}, \mathsf{false}\}$.

(ii) Given $\mathbb{S}$, write $\overline{\mathbb{S}}$ for the sorting which is the result of changing labels by $\overline{(\cdot)}$. So $\overline{\mathsf{nat}}$ is the sorting with the same sort as nat which is however labelled by ")". Next, given $\mathbb{S}_1$ and $\mathbb{S}_2$, let $\mathbb{S}_1 \uplus \mathbb{S}_2$ denote their disjoint union, i.e. the sorts are the disjoint union of $\mathbb{S}_1$ and $\mathbb{S}_2$, inheriting labelling and justification. Then $\overline{\mathsf{nat}} \uplus \mathsf{nat}$ is the sorting with two copies of ω labelled by ")" and "]".

(iii) We define $\mathsf{nat}\mathrel{\underline{\rightharpoonup}}\mathsf{nat}$ as a type with three sorts, one is a singleton written $]^{\mathsf{nat}\mathrel{\underline{\rightharpoonup}}\mathsf{nat}}$, another a copy of ω written $[^{\mathsf{nat}}$, and the third again a copy of ω written $]^{\mathsf{nat}}$, for which labels are given as these notations indicate. The justification is given so that $]^{\mathsf{nat}\mathrel{\underline{\rightharpoonup}}\mathsf{nat}}$ only justifies $[^{\mathsf{nat}}$, which in turn only justifies $]^{\mathsf{nat}}$.

By a *sequence from a set* X we mean a partial function from ω to X defined for a finite initial segment of ω (called *indices*) and undefined for the rest. As an example, abc has $\{0, 1, 2\}$ as its indices. We often confuse elements and their occurrences in a sequence. ε denotes the empty sequence. We are interested in sequences of actions representing a certain kind of interaction between an agent (Player) and the outside (Opponent).

2.3. **Action Sequence.** Given a sorting $\mathbb{S}$, an *action sequence in* $\mathbb{S}$ or often simply a *sequence in* $\mathbb{S}$ is a sequence from actions in $\mathbb{S}$ (let is be $x_0x_1...x_{n-1}$), together with the relation on its indices denoted $\mapsto$ (writing $x_i \mapsto x_j$ for $i \mapsto j$), satisfying:

(consistency) (1) $x_i \mapsto x_j \Rightarrow i \lneq j$, (2) $(x_i \mapsto x_k \ \wedge\ x_j \mapsto x_k) \Rightarrow i = j$, (3) $x_i^S \mapsto x_k^{S'} \Rightarrow S' \in \mathit{Obs}(S)$, (4) $\neg\exists x_i.\ x_i \mapsto x_j^S \Rightarrow S$ initial (then x_j *occurs free*),

(linearity in answers) (5) $(x_i \mapsto x_j \ \wedge\ x_i \mapsto x_k \ \wedge\ x_k$ an answer$) \Rightarrow j = k$, (6) A free O-answer (resp. a free P-answer) occurs at most once, and:

(strict alternation) (7) If x_i is a P-action (resp. O-action), then x_{i+1} is an O-action (resp. P-action) for $0 \leq i \leq n-2$.

$s, s', \ldots$ range over action sequences, often leaving the associated $\mapsto$ implicit. We say x_i *justifies* x_j when $x_i \mapsto x_j$. On action sequences we define two functions, $\ulcorner s \urcorner$, the *P-view* of s, and $\llcorner s \lrcorner$, the *O-view* of s, as, inheriting $\mapsto$ whenever possible: **(pv0)** $\ulcorner \varepsilon \urcorner = \varepsilon$, **(pv1)** $\ulcorner s x_i \urcorner = x_i$ when x_i is a free O-action, **(pv2)** $\ulcorner s_0 x_i s_1 x_j \urcorner = \ulcorner s_0 \urcorner x_i x_j$ when $x_i \mapsto x_j$ and x_j is an O-action, and **(pv3)** $\ulcorner s_0 x_i \urcorner = \ulcorner s_0 \urcorner x_i$ if x_i is a P-action; $\llcorner s \lrcorner$ is defined dually, i.e. by exchanging "O-action" and "P-action" throughout. We then say:

(i) s is *well-bracketed* when: *if* $s_0 x_i s_1 x_j$ is a prefix of s such that (1) x_i is a question (2) x_j is an answer and (3) either x_j occurs free or x_j is justified by a question in s_0, *then* x_i justifies an answer in s_1.

(ii) s *satisfies the visibility condition* when, in any of its prefix $s_0 x_i$ where x_i is a P-action (resp. O-action) which is y_j s.t. $y_j \mapsto x_i$ always occurs in $\ulcorner s_0 \urcorner$ (resp. $\llcorner s_0 \lrcorner$).

An action sequence is *legal* when it is well-bracketed and satisfies the visibility condition. Legal action sequences are sometimes called *legal positions.* We can verify the set of legal sequences of any sorting is closed under prefix and view constructions.

We are now ready to give the main definition of this section, which determines the class of interacting processes we are concerned with in the present study.

2.4. **Definition.** (strategy) An *innocent strategy from* A *to* B, or simply a *strategy from* A *to* B, is a prefix-closed set σ of legal positions in $\overline{A} \uplus B$, such that:

(O-initial) $s \in \sigma$ implies the initial action of s (if any) is an O-action.

(contingency completeness) $s \in \sigma$ and sx_i is legal for an O-action x_i imply $sx_i \in \sigma$.

(innocence) If $s_1 x, s_2 \in \sigma$, x is a P-action and $\ulcorner s_1 \urcorner = \ulcorner s_2 \urcorner$, then $s_2 y \in \sigma$ such that (1) $\ulcorner s_1 x \urcorner = \ulcorner s_2 y \urcorner$ and (2) $s_2 z \in \sigma \ \Rightarrow\ s_2 z = s_2 y$.

We write $\sigma : A \rightarrow B$ when σ is a strategy from A to B. f_σ denotes the partial function determined by σ, mapping even-length P-views to next actions (if any) with justification. Given $\sigma, \tau : A \rightarrow B$, we set $\sigma \leq \tau$ when $\sigma \subset \tau$, equivalently when $f_\sigma \subset f_\tau$.

Using the function representation, it is easy to see the set of strategies from A to B forms a dI-domain under $\leq$, where compact elements are those with finite graphs. Further, given $sx_ix_{i+1} \in \sigma : A \to B$, if x_i and x_{i+1} come from different types then x_{i+1} is necessarily a P-action (switching condition). Also the projection of $s \in \sigma : A_1 \to A_2$ onto A_i $(i = 1, 2)$, written $s \upharpoonright A_i$, is always legal in A_i.

2.5. **Examples.** (strategies)

- (i) (undefined) For each A and B, there is a strategy from A to B which is totally undefined, so that it is least w.r.t. the ordering $\leq$. We write this strategy $\perp_{A \to B}$.
- (ii) (first-order function) The set of strategies from nat to nat precisely correspond to the set of partial functions from ω to ω.
- (iii) (higher-order function) We describe a strategy $\sigma : \mathsf{nat} \Rightarrow \mathsf{nat} \to \mathsf{nat}$ which corresponds to the behaviour of an open call-by-value PCF-term, $x : \iota \to \iota \rhd \mathsf{succ}(x3) : \iota$. After receiving a signal on the left, which is a function, σ asks the result of applying 3 to that function, and, on receiving the answer, returns its successor to the right. Except the last free answer, each action is justified by the preceding one.

Strategies denote a certain kind of deterministic processes, and are, as such, precisely representable as (name passing) synchronisation trees, see [18]. The presentation is often useful for describing, and reasoning about, strategies: indeed the full abstraction result was originally obtained in this setting [17]. The following inductive definition of composition of strategies is suggested by such representation.

2.6. **Definition.** (composition) Given $\sigma : A \to B$ and $\tau : B \to C$, we set:

$$\{s_1; s_2 \mid s_1 \in \sigma,\ s_2 \in \tau,\ s_1 \upharpoonright B = s_2 \upharpoonright \overline{B}\}$$

where $s_1; s_2$ with s_1 and s_2 as above is given: (1) $\varepsilon; \varepsilon = \varepsilon$, (2) $s_1 x^B; s_2 \overline{x^B} = s_1; s_2$ ($\overline{x^B}$ is the corresponding dual action of x^B), and (3) $s_1 x^A; s_2 = (s_1; s_2) x^A$, $s_1; s_2 x^C = (s_1; s_2) x^C$, in each case inheriting the justification relation from the original pair.

(3) above is well-defined since two cases are always disjoint due to the switching condition. We can also verify: (i) $\sigma; \tau$ is a strategy from A to C, (ii) ; is associative with identity given by the copy-cat strategy, i.e. that which exactly copies actions between $\overline{A}$ and A, and (iii) ; is bi-continuous with respect to $\leq$. Thus we define:

2.7. **Definition.** $\mathcal{CBV}$ denotes the category of cbv-types and innocent strategies.

By the preceding discussions, $\mathcal{CBV}$ is enriched over CPO, the category of possibly bottomless cpos and continuous functions. Each homset has a least element $\perp$ for which the composition is left strict, that is $\perp; \sigma = \perp$ always.

3. INTENSIONAL UNIVERSE

Type structures of a semantic universe offer the basic articulation of its algebraic structures needed, for example, for interpreting various programming languages in it. This section clarifies the basic type structure of $\mathcal{CBV}$ in the light of the distinction between total and partial maps. We first introduce the notion of totality, cf. [13].

3.1. **Definition.** σ is *total* when $\tau; \sigma = \perp$ implies $\tau = \perp$. We write $\sigma \Downarrow$ when σ is total.

The totality of $\sigma : A \to B$ is equivalent to any one of: (1) $\forall \tau : \mathbf{1} \to A.\ \tau \Downarrow \Rightarrow \tau; \sigma \Downarrow$, (2) the square $\langle \mathbf{0} \to A \xrightarrow{\sigma} B,\ \mathbf{0} \to \mathbf{0} \to B \rangle$ is a weak pullback (notice $\mathbf{0}$ is initial and weakly terminal), and (3) σ immediately emits the P-signal for each initial O-signal. (1) relates to a familiar idea of totality, (2) is a categorically basic one, and (3) gives the behavioural characterisation, clarifying the dynamic aspect of totality.

3.2. **Examples.** (total maps)

(i) The unique arrow $\perp$ from $\mathbf{0}$ to any type is total, by definition. All isomorphisms are total. Also, there is no total map to $\mathbf{0}$, except from itself.

(ii) There is a unique total map $!_A : A \to \mathbf{1}$ for each A. It reacts to the initial signal (if any) by the unique P-signal at $\mathbf{1}$, and no more action is possible.

(iii) $\sigma : \mathsf{nat} \to \mathsf{nat}$ is total iff the underlying number-theoretic function is total.

Let us denote $\mathcal{CBV}_{\mathtt{t}}$ for the category of types and total strategies. Since totality is closed upwards w.r.t. $\leq$, $\mathcal{CBV}_{\mathtt{t}}$ again CPO-enriches. It has finite products: 3.2 (ii) above shows $\mathbf{1}$ is terminal, while the product of A and B is given by a type $A \otimes B$ whose sorts are the disjoint union of non-initial sorts of A and B together with, for each pair of $S \in \mathrm{init}(A)$ and $S' \in \mathrm{init}(B)$, a sort $]^{S,S'} = S \times S'$ (the set theoretic product), which justifies what S and S' justify in A and B, the rest as in A and B ($A \otimes \mathbf{0}$ and $\mathbf{0} \otimes A$ are set as $\mathbf{0}$). Projection maps are evidently given. $\otimes$ is often denoted $\times$ in $\mathcal{CBV}_{\mathtt{t}}$. We also note that $\mathcal{CBV}_{\mathtt{t}}$ has arbitrary (small) products and co-products, but we do not need them here.

The relationship between total maps and usual (often called *partial*) maps is clarified by the notion of *lifting*. Write $A_\perp$ for the type given by adding two singleton sorts to A, one initial which justifies the other one, the latter justifying all $S \in \mathrm{init}(A)$, the rest as in A. Then we can see the set of *total* arrows from $\mathbf{1}$ to $A_\perp$ is order-isomorphic to the set of *partial* arrows from $\mathbf{1}$ to A. These two are mediated by two copy-cat like strategies, $\mathsf{up} : A \to A_\perp$ and $\mathsf{dn} : A_\perp \to A$, with obvious behaviours (up reacts to an initial action at A by going though two added actions at $A_\perp$ then does the copy-cat: dn just does the dual). In a familiar way this induces the adjoint situation as described below.

3.3. **Proposition.**

(i) Let F be the inclusion functor from $\mathcal{CBV}_{\mathtt{t}}$ to $\mathcal{CBV}$. Then F has the right adjoint T, with $T(A) = A_\perp$, the unit $\eta_A = \mathsf{up}$, and the co-unit $\epsilon = \mathsf{dn}$, which CPO-enriches. The monad $\langle T,\ \eta_A,\ T(\mathsf{dn})\rangle$ is denoted $\mathbf{T}$, which has a tensorial strength $\mathtt{st}_{A,B}$ and a co-strength (in the sense of [37]) $\mathtt{st}'_{A,B}$.

(ii) The Kleisli category of $\mathbf{T}$ on $\mathcal{CBV}_{\mathtt{t}}$ is isomorphic to $\mathcal{CBV}$. We write $\sigma^\dagger$ for $\mathsf{up}; T(\sigma) : A \to B_\perp$ where $\sigma : A \to B$ is partial, and $\sigma_\dagger$ for $\sigma; \mathsf{dn} : A \to B$ where $\sigma : A \to B_\perp$ is total.

Using the monad $\mathbf{T}$, we can now present the basic type structures of $\mathcal{CBV}$. In (iii) below $A \Rightarrow B$ is a type whose sorts are the disjoint union of those of A and B together with new $]^{A \Rightarrow B}$ which is a singleton, with the label of each $S \in \mathrm{init}(A)$ changed into $[$ and those of A's non-initial sorts dualised. Justification is as in A and B, with the addition of $]^{A \Rightarrow B}$ justifying what were in $\mathrm{init}(A)$, each of which in turn justifying what were in $\mathrm{init}(B)$ ($\mathbf{0} \Rightarrow B$ is set as $\mathbf{1}$). Notice the similarity with the construction of $\overline{A} \uplus B$.

3.4. **Definition and Proposition.**

(i) (partial pairing [32]) Given $\sigma_1 : C \to A$ and $\sigma_2 : C \to B$, their *left pairing*, $\langle\!\langle \sigma_1,\ \sigma_2 \rangle\!\rangle_l : C \to A \otimes B$, and the *right paring*, $\langle\!\langle \sigma_1,\ \sigma_2 \rangle\!\rangle_r : C \to A \otimes B$ are given as: $\langle\!\langle \sigma_1,\ \sigma_2 \rangle\!\rangle_l \overset{\mathrm{def}}{=} (\langle \sigma_1^\dagger,\ \sigma_2^\dagger \rangle; \psi_{A,B})_\dagger$ and $\langle\!\langle \sigma_1,\ \sigma_2 \rangle\!\rangle_r \overset{\mathrm{def}}{=} (\langle \sigma_1^\dagger,\ \sigma_2^\dagger \rangle; \tilde{\psi}_{A,B})_\dagger$ where $\psi_{A,B} = \mathtt{st}'_{A,TB}; T(\mathtt{st}_{A,B}; \mathsf{dn})$ and $\tilde{\psi}_{A,B} = \mathtt{st}_{A,TB}; T(\mathtt{st}'_{A,B}; \mathsf{dn})$.

(ii) (premonoidal tensor [37]) Given A, we define $A\otimes$ and $\otimes A$ by: (i) $A \otimes \cdot B = A \otimes B$ and $B \cdot \otimes A = B \otimes A$, and (ii) $A \otimes \sigma \overset{\mathrm{def}}{=} \langle\!\langle \pi_1,\ \pi_2; \sigma \rangle\!\rangle$, and $\sigma \otimes A \overset{\mathrm{def}}{=} \langle\!\langle \pi_1; \sigma,\ \pi_2 \rangle\!\rangle$ where π_i denote projections. Then $A\otimes$ and $\otimes A$ both define functors on $\mathcal{CBV}$

which CPO-enrich. We then define, for $\sigma : A \to B$ and $\tau : C \to D$: (i) $\sigma \otimes_l \tau = (\sigma \otimes C); (B \otimes \tau)$ and (ii) $\sigma \otimes_r \tau = (A \otimes \sigma); (\tau \otimes C)$.

(iii) (partial exponential [23]) The functor $\underline{\cdot} \otimes A : \mathcal{CBV}_{\mathsf{t}} \to \mathcal{CBV}$ has the right adjoint $A \rightharpoonup \underline{\cdot} : \mathcal{CBV} \to \mathcal{CBV}_{\mathsf{t}}$, which CPO-enriches. Equivalently, there exists an arrow $\mathsf{ev} : (A \rightharpoonup B) \otimes A \to B$ such that, for any $\sigma : C \otimes A \to B$, there is a unique total arrow $p\lambda(\sigma) : C \to B$ satisfying $(p\lambda(\sigma) \otimes \mathsf{id}); \mathsf{ev} = \sigma$, and $p\lambda$ is a continuous operator.

An outstanding fact on partial pairing is that the right and left pairings of the same tuple do *not* coincide in general. This exhibits a strongly intensional character of $\mathcal{CBV}$, substantiating Moggi's remark that $\langle\!\langle \sigma_1, \sigma_2 \rangle\!\rangle_l$ and $\langle\!\langle \sigma_1, \sigma_2 \rangle\!\rangle_r$ reflect the "order of evaluation" [32]. This also implies the tensor in $\mathcal{CBV}$ does not give a bifunctor, cf. Corollary 4.3 of [37]. We write $\langle\!\langle \sigma_1, \sigma_2 \rangle\!\rangle$ when two versions coincide (as when either is total).

The final structure we need is *recursion*, here presented as an operator on each homset. [18] gives an alternative presentation as constants. Below we say A is *pointed* when it has a unique initial sort which is a singleton, equivalently when $\mathrm{hom}(\mathbf{1}, A)$ in $\mathcal{CBV}_{\mathsf{t}}$ is a pointed cpo. For such A, $\mathsf{dn}'_A : TA \to A$ denotes the unique total map such that $\mathsf{up}_A; \mathsf{dn}'_A = \mathsf{id}_A$. Pointed types are precisely objects in the category of Eilenberg-Moore algebra of the monad $\mathbf{T}$. Also, any type of form $A \rightharpoonup B$ is pointed.

3.5. **Proposition.** Let A be pointed and $\sigma : C \times A \to A$. Then there is a strategy $\mathbf{rec}(\sigma) : C \to A$ which satisfies: (i) $\tau; \mathbf{rec}(\sigma) = \tau; \langle\!\langle \mathsf{id}_C, \mathbf{rec}(\sigma) \rangle\!\rangle; \sigma$ for $\tau : \mathbf{1} \to C$ (if σ is total we can take off τ from the equation). (ii) $\mathbf{rec}(\tau \otimes \mathsf{id}_A; \sigma) = \tau; \mathbf{rec}(\sigma)$ for each $\tau : B \to C$, and: (iii) Given $\tau : \mathbf{1} \to C$, if $\{\rho_i : \mathbf{1} \to A\}_{i \in \omega}$ is defined as: (1) $\rho_0 = \bot$, (2) $\rho_{i+1} = \langle\!\langle \tau, \rho_i^{\dagger}; \mathsf{dn}' \rangle\!\rangle; \sigma$, then $\{\rho_i\}$ is an increasing ω-chain such that $\sqcup \rho_i = \tau; \mathbf{rec}(\sigma)$.

4. EXTENSIONAL UNIVERSE

$\mathcal{CBV}$ represents an abstract notion of execution of call-by-value computation. For the interpretation of programming languages at the same abstraction level as in the standard semantic universe like the category of domains, we may need a more abstract universe, which we construct from $\mathcal{CBV}$ by a simple quotient construction. The universe is also useful for understanding the behaviour of arrows in $\mathcal{CBV}$ in an abstract way. Below we briefly outline the basic structure of this universe, leaving details to [18]. We start from the following ordering (cf. [36, 11]):

$$\sigma_1 \precsim \sigma_2 \quad \overset{\mathrm{def}}{\Leftrightarrow} \quad \forall C, C', \tau : C \to A, \tau' : B \to C'. \ \tau; \sigma_1; \tau' \Downarrow \quad \Rightarrow \quad \tau; \sigma_2; \tau' \Downarrow .$$

Immediately $\precsim$ is a preorder for which the composition is monotone (thus the quotient is well-defined), and $\leq \subseteq \precsim$. We now define $\widehat{\mathcal{CBV}}$ as the category of types and $\precsim$-equivalence classes of strategies. $f, g, \ldots$ range over arrows in $\widehat{\mathcal{CBV}}$. The induced partial order is still denoted $\precsim$. $\widehat{\mathcal{CBV}}$ is enriched over Poset, the category of posets with monotone maps, since monotonicity carries over from $\mathcal{CBV}$. Observing $\mathbf{0}$ is the zero object in $\widehat{\mathcal{CBV}}$ (i.e. both terminal and initial), we define $\bot : A \to B$ as the unique map that factors through $\mathbf{0}$, cf. [13]. Then $\bot$ is indeed the least element in each homset, and the composition is strict at both sides. We can then define total maps as before: $f \Downarrow$ when $g; f = \bot$ implies $g = \bot$ for each g, equivalently when the square $\langle \mathbf{0} \to A \xrightarrow{f} B, \ \mathbf{0} \to \mathbf{0} \to B \rangle$ is a pullback, from which all properties of total maps as in $\mathcal{CBV}$ follow. Notice also $f \Downarrow \Leftrightarrow \forall \sigma \in f. \, \sigma \Downarrow \Leftrightarrow \exists \sigma \in f. \, \sigma \Downarrow$. We write $\widehat{\mathcal{CBV}}_{\mathsf{t}}$ for the subcategory of total maps.

We can then show $\widehat{\mathcal{CBV}}_{\mathsf{t}}$ is well-pointed, with finite products (indeed all small products and co-products) inducing Poset-enriched bi-functors, all inheriting from $\mathcal{CBV}_{\mathsf{t}}$. Again as in $\mathcal{CBV}$, the inclusion functor from $\widehat{\mathcal{CBV}}_{\mathsf{t}}$ to $\widehat{\mathcal{CBV}}$ has the right adjoint inheriting constructions from T, which we write again T, which Poset-enriches. The corresponding monad, again denoted $\mathbf{T}$, has strengths and is now commutative, i.e. $\psi_{A,B} = \tilde{\psi}_{A,B}$ in 3.4 (i). Again the Kleisli category of $\mathbf{T}$ on $\widehat{\mathcal{CBV}}_{\mathsf{t}}$ is isomorphic to $\widehat{\mathcal{CBV}}$. Using the monad, we can now clarify the basic type structures of $\widehat{\mathcal{CBV}}$. Thus, again from the general result by Power and Robinson [37], we know $\widehat{\mathcal{CBV}}$ is a Poset-enriched symmetric monoidal category, which has all type structures as given in Proposition 3.4 (i)(ii)(iii) inheriting the constructions from $\mathcal{CBV}$, where left and right pairings are identified. Finally the recursion in $\mathcal{CBV}$ carries over to $\widehat{\mathcal{CBV}}$, though all $\tau : \mathbf{1} \to C$ in 3.5 can be replaced with id_C. We also note that $\widehat{\mathcal{CBV}}$ allows the treatment of recursive types for a large class of functors, but we do not use them in the present paper.

5. INTERPRETATION OF $\mathrm{PCF}_{\mathrm{V}}$

$\mathrm{PCF}_{\mathrm{V}}$[35, 36] is a typed programming language based on call-by-value evaluation. The syntax and evaluation rules can be found in the standard literature, cf.[15, 42, 40], which are briefly reviewed in Appendix (following [15] except the recursion is only defined for function types, cf.[42, 40]). $\mathcal{CBV}$ and its extensional quotient are conceived to represent call-by-value, or partial, higher-order functional computation. Moreover it has a type structure which does include that of $\mathrm{PCF}_{\mathrm{V}}$. Thus we may seek to represent $\mathrm{PCF}_{\mathrm{V}}$-terms and its computation in these universes. We primarily consider the interpretation in $\mathcal{CBV}$, and only move to $\widehat{\mathcal{CBV}}$ at the last step. The interpretation follows.

5.1. **Definition.** First we define the mapping from the set of types and environments of $\mathrm{PCF}_{\mathrm{V}}$ to objects in $\mathcal{CBV}$ as: $\llbracket \iota \rrbracket \overset{\mathrm{def}}{=} \mathsf{nat}$, $\llbracket o \rrbracket \overset{\mathrm{def}}{=} \mathsf{bool}$, $\llbracket \alpha \Rightarrow \beta \rrbracket \overset{\mathrm{def}}{=} \llbracket \alpha \rrbracket \rightharpoonup \llbracket \beta \rrbracket$, $\llbracket \varepsilon \rrbracket \overset{\mathrm{def}}{=} \mathbf{1}$ and $\llbracket \Gamma, x : \alpha \rrbracket \overset{\mathrm{def}}{=} \llbracket \Gamma \rrbracket \otimes \llbracket \alpha \rrbracket$. Then the mapping from $\mathrm{PCF}_{\mathrm{V}}$-terms to arrows in $\mathcal{CBV}$ is given inductively as follows, assuming either of the left/right pairings is selected uniformly.

(i) $\llbracket \Gamma, x : \alpha, \Delta \rhd x : \alpha \rrbracket \overset{\mathrm{def}}{=} \pi : \llbracket \Gamma \rrbracket \otimes \llbracket \alpha \rrbracket \otimes \llbracket \Delta \rrbracket$, where π is an appropriate projection.

(ii) $\llbracket \Gamma \rhd \lambda x^{\alpha}.M : \alpha \Rightarrow \beta \rrbracket \overset{\mathrm{def}}{=} p\lambda(\sigma) : \llbracket \Gamma \rrbracket \to \llbracket \beta \rrbracket$, where $\llbracket \Gamma, x : \alpha \rhd M : \beta \rrbracket = \sigma$.

(iii) $\llbracket \Gamma \rhd MN : \beta \rrbracket \overset{\mathrm{def}}{=} \langle\!\langle \sigma_1, \sigma_2 \rangle\!\rangle ; \mathsf{ev} : \llbracket \Gamma \rrbracket \to \llbracket \beta \rrbracket$, where $\llbracket \Gamma \rhd M : \alpha \Rightarrow \beta \rrbracket = \sigma_1$ and $\llbracket \Gamma \rhd N : \alpha \rrbracket = \sigma_2$.

(iv) $\llbracket \Gamma \rhd \boldsymbol{\mu} x^{\alpha}.M : \alpha \rrbracket \overset{\mathrm{def}}{=} \mathbf{rec}(\sigma) : \llbracket \Gamma \rrbracket \to \llbracket \alpha \rrbracket$, where $\llbracket \Gamma, x : \alpha \rhd M : \alpha \rrbracket = \sigma$

(v) $\llbracket \Gamma \rhd \mathsf{cond}\ L\ M_1\ M_2 : \alpha \rrbracket \overset{\mathrm{def}}{=} (\langle\!\langle \tau, \langle\!\langle \sigma_1^{\dagger}, \sigma_2^{\dagger} \rangle\!\rangle \rangle\!\rangle ; \gamma_{T(\llbracket \alpha \rrbracket)})_{\dagger} : \llbracket \Gamma \rrbracket \to \llbracket \alpha \rrbracket$ where $\llbracket \Gamma \rhd L : o \rrbracket = \tau$, $\llbracket \Gamma \rhd M_1 : \alpha \rrbracket = \sigma_1$, $\llbracket \Gamma \rhd M_2 : \alpha \rrbracket = \sigma_2$ and $\gamma_A : \mathsf{bool} \otimes A \otimes A \to A$ is a strategy with an appropriate behaviour.

(vi) For a constant c of type α, we set: $\llbracket \Gamma \rhd c : \alpha \rrbracket \overset{\mathrm{def}}{=} !_{\llbracket \Gamma \rrbracket} ; \overline{c} : \llbracket \Gamma \rrbracket \to \mathbf{1} \to \llbracket \alpha \rrbracket$ where $\overline{c} : \mathbf{1} \to \llbracket \alpha \rrbracket$ is given as a strategy with obvious behaviour for each c.

The descriptions of γ and $\overline{c}$ for each c are given in [18]. As basic properties of the mapping, we know $\llbracket \Gamma \rhd V : \alpha \rrbracket$ is always total, where V denotes a value, i.e. an abstraction or a non-Ω constant; $\llbracket \Gamma \rhd M\{V/x\} : \beta \rrbracket = \langle\!\langle \mathsf{id}_{\llbracket \Gamma \rrbracket}, \tau \rangle\!\rangle ; \sigma : \llbracket \Gamma \rrbracket \to \llbracket \beta \rrbracket$ for any $\tau = \llbracket \Gamma \rhd V : \alpha \rrbracket$ and $\sigma = \llbracket \Gamma, x : \alpha \rhd M : \beta \rrbracket$; and that $\Gamma \rhd M \Downarrow V$ implies $\llbracket M \rrbracket = \llbracket V \rrbracket$. We can then verify the following key properties of the interpretation.

5.2. **Proposition.**

(i) (computational adequacy) $\llbracket M \rrbracket \neq \bot$ iff $\exists V.\ M \Downarrow V$ for a closed M.

(ii) (adequacy) $\llbracket M \rrbracket \precsim \llbracket N \rrbracket$ implies $M \preceq_{obs} N$ for closed M, N of the same type.

Given the adequacy result, if we show its converse, i.e. $\preceq_{obs}$ implies $\precsim$ via the interpretation, then we obtain the full abstraction. For the purpose it suffices to prove all compact elements of appropriate types are $\mathrm{PCF_V}$-definable, cf.[25, 35]. The definability argument is carried out using a subset of $\mathrm{PCF_V}$-terms defined as follows.

5.3. **Definition.** *Finite canonical forms* (FCFs for short) are inductively given as:

(i) $\Gamma \triangleright \Omega : \alpha$ and $\Gamma \triangleright n : \iota$ are FCF's.

(ii) $\Gamma \triangleright \lambda y^{\alpha}.M : \alpha \rightarrow \beta$ is a FCF if $\Gamma, y : \alpha \triangleright M : \beta$ is.

(iii) $\Gamma \triangleright \mathbf{let}\ y^{\alpha} = zV\ \mathbf{in}\ N : \beta$ is a FCF if (1) $\Gamma, y : \alpha \triangleright N : \beta$ is a FCF, (2) z has a type $\beta \Rightarrow \alpha$ in Γ, and (3) $\Gamma \triangleright V : \beta$ is a FCF (which is also a value).

(iv) $\Gamma \triangleright (\mathbf{case}\ x\ \mathbf{of}\ n_1 : M_1 \,[]\, n_2 : M_2 \,[]..[]\, n_k : M_k\ ) : \alpha$ is a FCF if $x : \iota \in \Gamma$ and, for each i, $\Gamma \triangleright M_i : \alpha$ is a FCF.

where, in (iii), $\mathbf{let}\ y^{\alpha} = zM\ \mathbf{in}\ N$ stands for $(\lambda y^{\alpha}.N)(zM)$, and, in (iv), $\mathbf{case}\ y\ \mathbf{of}\ n_1 : M_1 \,[]..[]\, n_k : M_k$ stands for $\mathsf{cond}\ (y = n_1)\ M_1(...(\mathsf{cond}\ (y = n_k)\ M_k\ \Omega)..)$, the latter assuming the equality check is suitably encoded in $\mathrm{PCF_V}$.

FCFs faithfully capture the behaviour of compact strategies of PCF-types:

(i) Ω denotes $\perp$. $m : \iota$ immediately returns m after an initial O-signal.

(ii) $\lambda x^{\alpha}.M : \alpha \Rightarrow \beta$ represents a strategy which, after an initial O-signal, does a sequence of actions $]^{\alpha \Rightarrow \beta}\ [^{\alpha}$ (here an annotated label denotes an action of that kind) where $]^{\alpha \Rightarrow \beta} \mapsto [^{\alpha}$, then behaves as M.

(iii) $\Gamma, x_i : \gamma_1 \Rightarrow \gamma_2, \Delta \triangleright \mathbf{let}\ y^{\alpha} = x_i M\ \mathbf{in}\ N : \beta$ first interacts at x_i by $(^{\gamma_i}$, then Opponent may ask at M (when γ_1 is a higher-order type) which, after some interactions, will be answered by Player, followed by an Opponent Answer $)^{\gamma_2}$. Then the actions move to N. Here the "let" construct is used to make the order of evaluation explicit (see [32] for a similar use of the construct in a different context).

(iv) The case statement corresponds to the situation when a strategy acts according to the received ground values (here natural numbers). A vector of values can be handled by nesting the construct.

Using FCFs we can prove:

5.4. **Theorem.** (definability) For each compact element $\sigma : \mathbf{1} \rightarrow [\![\alpha]\!]$ for any $\mathrm{PCF_V}$-type α in $\mathcal{CBV}$, there is a FCF $F : \alpha$ such that $[\![F : \alpha]\!] = \sigma$. Conversely, the interpretation of any FCF is a compact element in the respective type.

The proof is by induction on the cardinality of compact elements, translating the behaviour of strategies into the corresponding FCFs based on the correspondence between actions and strategies we illustrated above. We note that, like FCFs themselves, the argument is much simpler than the corresponding one in call-by-name PCF, cf.[19]. See [18] for details. Write $[\![\Gamma \triangleright M : \alpha]\!]_e$ for $[[\![\Gamma \triangleright M : \alpha]\!]]_{\precsim}$. From the definability result we can now conclude:

5.5. **Theorem.** (full abstraction) For closed $\mathrm{PCF_V}$-terms $M : \alpha$ and $N : \alpha$, we have $M : \alpha \preceq_{obs} N : \alpha$ iff $[\![M : \alpha]\!]_e \precsim [\![N : \alpha]\!]_e$.

6. DISCUSSIONS

6.1. **Further Results.** First we briefly outline how call-by-name universe and the call-by-value universe are mutually embeddable, as in the context of domains. Let *cbn-types* be sortings in which (1) initial sorts are all opponent questions and (2) each sort is reachable from some initial sort. The strategies are then as in Definition 2.4 with an added condition which ensures the switching condition. The composition of strategies is just as in Section 2, based on which we obtain the category of cbn-types and innocent strategies which is cartesian-closed and is enriched over CPO, which we denote $\mathcal{CBN}$. There is a full embedding of $\mathbb{CA}$ of [19] in $\mathcal{CBN}$ and its extensional quotient allows interpretation of call-by-name FPC as in the category in [24]. Now we say a $\mathcal{CBN}$ type is *pointed* when it has a unique initial sort which is a singleton, just as in $\mathcal{CBV}$. Let us also say a strategy in $\mathcal{CBN}$ is *linear* when, after the initial question at the codomain, it immediately asks the question at the domain, and never asks an initial question at the domain again. Writing $\mathcal{CBN}_1$ for the subcategory of $\mathcal{CBN}$ of pointed types and linear strategies, the embedding result says (i) $\mathcal{CBN}$ is isomorphic to the full subcategory of $\mathcal{CBV}_{\mathrm{t}}$ of pointed types, and (ii) $\mathcal{CBV}$ is isomorphic to the full subcategory of $\mathcal{CBN}_1$ of pointed types whose initial questions justify no questions. The proof is by the translation of information flow. See [18] for details.

Next we discuss how we would extend the full abstraction result in Section 5 to other call-by-value programming languages. Firstly it is straightforward to extend the argument in Section 5 to $\mathrm{PCF}_{\mathrm{V}}$ with sums and products or to the untyped call-by-value λ-calculus. Recursively typed languages such as FPC [15] can also be handled (though the premonoidal tensor in $\mathcal{CBV}$ poses a problem), as observed by Fiore and as will be reported elsewhere. For the interpretation of imperative constructs, we would consider, as noted in Introduction, variants of the present universe by changing parameters of games following [4, 21], which does lead to coherent semantic universes. One interesting topic in this context would be whether one needs refined type structures as in [4] for the interpretation of the impure constructs: indeed a much simpler, and more direct, approach seems possible in the present setting. Some results on these topics will be reported elsewhere.

6.2. **Related works.** After completing the full version of this paper [18], the authors were informed of an independent (and essentially concurrent) work by Riecke and Sandholm [38] in which they obtained a full abstraction for call-by-value FPC (which easily implies that of $\mathrm{PCF}_{\mathrm{V}}$). The construction is based on Kripke logical relations on pCPO, and is thus quite different from the present one. No quotienting is necessary to reach the semantic universe, while the construction of the universe itself is substantially more complicated. In a brief comparison, one may say that their approach would give better insights for understanding why some (continuous) function is *not* sequential; while their construction does not directly model the dynamic aspects of sequential call-by-value computation, thus may not lead to the insights in that context. Thus two methods would play different roles in semantic analysis.

In game semantics, Abramsky and McCusker are working on game semantics on call-by-value languages, based on McCusker's early idea and also suggested by the present work, which tries to extract call-by-value strategies from the universes of call-by-name games in [24, 4] (personal communication).[2] In another vein, Harmer and Malacaria are working on game semantics for call-by-value computation based on games originally introduced in [3]. [16] gives a preliminary study in this direction.

6.3. **Intensionality and relationship with process theories.** The strongly intensional character of $\mathcal{CBV}$ is not at the same level of abstraction as, say, pCPO. The same can be said about its call-by-name counterpart and other categories of games, in the sense

---

[2] At the final stage of preparation of this camera-ready version, we obtained their typescript [5], which exploits the type structures of the original universe in [4] to interpret a functional language with a certain imperative feature. Detailed discussions, especially the comparison with an approach we mentioned in 6.1, should be left for a future occasion.

that they reflect some notion of execution, albeit abstractly, cf. [9, 19]. From the viewpoint that the primary purpose of semantic representation of programming languages lies in giving (in)equations over programs as general as possible, this feature may be considered as a drawback. However we can take a different perspective, and ask whether this novel way of representing programs can be put to a significant use, especially once given the full abstraction result as the semantic justification of the representation. As a first such step, one may exploit the representation for the development of abstract theory of execution, including the formal optimisation techniques. Type structures as we studied in Section 4 may be put to an effective use in this context. One interest in this regard is that our interpretation of $\mathrm{PCF_V}$ in $\mathcal{CBV}$ already gives a concise abstract implementation of the language in the form name passing processes. The representation is comparable to Milner's direct encoding in [27], performing the β_v-reduction by three name passing interactions. Such a "physical" character of the abstract universe suggests we may study the execution of, say, call-by-value programming languages from a new level of mathematical abstraction (this is in line with Girard's studies on the semantics of cut elimination [14]). Relatedly the induced encodings also suggest the possibility of relating game semantics and process theories at the fundamental level. The study of behavioural types by Milner [28] may suggest possible directions (from which the present study actually started).

Acknowledgments. Special thanks go to Marcelo Fiore for his suggestions concerning pertinent categorical structures. We thank Samson Abramsky, Paul Mellies, Pasquale Malacaria, Guy McCusker, Jon Riecke and anonymous referees for comments and/or discussions, and N. Raja for his hospitality in Bombay.

REFERENCES

[1] Abelson, H., Sussman, G.J., *Structure and Interpretation of Computer Program*, MIT Press, 1985.

[2] Abramsky, S. and Jagadeesan, R., Games and Full Completeness for Multiplicative Linear Logic, Journal of Symbolic Logic, 59(2), pp. 543–574, 1994.

[3] Abramsky, S., Jagadeesan, R. and Malacaria, P., Full Abstraction for PCF, 1994. To appear.

[4] Abramsky, S. and McCusker, G., Linearity, Sharing and State: a fully abstract game semantics for Idealized Algol with active expressions, ENTCS, Vol.3, North Holland, 1996.

[5] Abramsky, S. and McCusker, G., Call-by-value games, a typescript, 12p, Apr. 1997.

[6] Berry, G. and Curien, P. L., Sequential algorithms on concrete data structures. *TCS* Vol.20, pp.265–321, North-Holland, 1982.

[7] Blass, A., A game semantics for linear logic, *Annuals of Pure and Applied Logic*, 56:183–220, 1992.

[8] Curien, P. L., Sequentiality and full abstraction. In Proc. of *Application of Categories in Computer Science*, LNM 177, pp.86–94, Cambridge Press, 1995.

[9] Danos, V. and Regnier, L., Games and abstract machines. *LICS'96*, IEEE, 1994.

[10] Felshcer, W., Dialogue games as a foundation for intuitionistic logic, *Handbook of Philosophical logic*, Vol.3, pp.341–372, D. Reidel Publishing Company, 1986.

[11] Fiore, M., *Axiomatic Domain Theory in Category of Partial Maps*, PhD thesis, ECS-LFCS-94-307, Univ. of Edinburgh, 1994.

[12] Fiore, M. and Plotkin, G., An Axiomatisation of Computationally Adequate Domain Theoretic Models of FPC, *LICS'94*, pp.92–102, IEEE, 1994.

[13] Freyd, P., Algebraically Complete Categories, In Proc. of *Como. Category Theory Conference*, LNM 1488, pp.95–104, Springer Verlag, 1991.

[14] Girard, J.-Y., Linear Logic, *TCS*, Vol.50, pp.1–102, North-Holland, 1987.

[15] Gunter, C., *Semantics of Programming Languages: Structures and Techniques*, MIT Press, 1992.

[16] Harmer, R., Malacaria, P., Linear foundations of game semantics, a typescript, Sep. 1996.

[17] Honda, K., Yoshida, N., Name-Passing Games: a functional universe, a typescript, 35p, Nov. 1996.

[18] Honda, K. and Yoshida, N., Game-theoretic Analysis of Call-by-value Computation (full version of this paper), ftp-able at `ftp.dcs.ed.ac.uk/export/kohei/cbvfull.ps.gz`, Feb, 1997.

[19] Hyland, M. and Ong, L., On Full Abstraction for PCF: I, II and III, 130 pages, ftp-able at `theory.doc.ic.ac.uk/papers/Ong`, 1994.

[20] Hyland, M. and Ong, L., Pi-calculus, dialogue games and PCF, *FPCA'93*, ACM, 1995.

[21] Laird, J., Full abstraction for functional languages with control, *LICS'97*, IEEE, 1997.

[22] Kahn, G. and Plotkin, D., Domaines Concrets. INRIA Report 336, 1978.

[23] Longo, G. and Moggi, E., Cartesian closed categories of enumarations for effective type-structures, LNCS 173, Springer-Varlag, 1984.

[24] McCusker, G., Games and Full Abstraction for FPC. *LICS'96*, IEEE, 1996.

[25] Milner, R., Fully abstract models of typed lambda calculi. *TCS*, Vol.4, 1–22, North-Holland, 1977.

[26] Milner, R., *A Calculus of Communicating Systems*, LNCS 76, Springer-Verlag, 1980.

[27] Milner, R., Functions as Processes. *MSCS*, 2(2), pp.119–146, 1992.

[28] Milner, R., Sorts and Types of π-Calculus, a manuscript, 43pp, 1990.

[29] Milner, R., Polyadic π-Calculus: a tutorial. *Proceedings of the International Summer School on Logic Algebra of Specification*, Marktoberdorf, 1992.

[30] Milner, R., Tofte, M. and Harper, R., *The Definition of Standard ML*, MIT Press, 1990.

[31] Moggi, E., Partial morphisms in categories of effective objects, *Info.&Comp.*, 76:250–277, 1988.

[32] Moggi, E., Notions of Computations and Monads. *Info.&Comp.*, 93(1):55–92, 1991.

[33] Nickau, M., Hereditarily Sequential Functionals, LNCS 813, pp.253–264, Springer-Verlag, 1994.

[34] Ong, L., Correspondence between Operational Semantics and Denotational Semantics, *Handbook of Logic in Computer Science*, Vol.4, pp.269–356, Oxford University Press, 1995.

[35] Plotkin, G., LCF considered as a programming language, *TCS*, 5:223–255, North-Holland, 1975.

[36] Plotkin, G., Lecture on Predomains and Partial Functions. Notes for a course given at the Center for the Study of Language and Information, Stanford, 1985.

[37] Power, J., Robinson, E., Premonoidal Categories and Notions of Computation, To appear in *MSCS*.

[38] Riecke, J., and Sandholm,A. Relational Account of Call-by-value Sequentiality, *LICS'97*, 1997.

[39] Robinson, E. and Rosolini, P., Categories of Partial Maps, *Info.&Comp.*, 79:95–130, 1988.

[40] Sieber, K., Relating Full Abstraction Results for Different Programming Languages, *FST/TCS'10*, pp. 373–387, LNCS 472, Springer-Verlag, 1990.

[41] Winskel, G., Synchronization Trees, *TCS*, Vol.34, pp. 33–82, North-Holland, 1985.

[42] Winskel, G., *The Formal Semantics of Programming Languages*, MIT Press, 1993.

APPENDIX: PCF$_{\mathrm{V}}$

We give a brief review of syntax and operational semantics of the call-by-value PCF [15, 42, 40]: our treatment is nearest to [15]. Given an infinite set of *variables*, ranged over by $x, y, z, \ldots$, the syntax of the language is given as follows.

$$\alpha ::= \iota \mid o \mid \alpha \Rightarrow \beta \qquad M ::= x \mid \lambda x^{\alpha}.M \mid MM \mid \mathsf{cond}\ L\ M_1\ M_2 \mid \boldsymbol{\mu} x^{\alpha\Rightarrow\beta}.M \mid c$$

where c is a constant. An *environment* is a list of pairs of a variable and a type, where all variables are distinct, ranged over by $\Gamma, \Delta, \ldots$ The typing rules of PCF$_{\mathrm{V}}$ is given as:

$$\Gamma, x : \alpha, \Gamma' \rhd x : \alpha \qquad \frac{c \text{ is a constant of type } \alpha}{\Gamma \rhd c : \alpha} \qquad \frac{\Gamma \rhd M : \alpha \Rightarrow \beta \quad \Gamma \rhd N : \alpha}{\Gamma \rhd MN : \beta}$$

$$\frac{\Gamma, x : \alpha \rhd M : \beta}{\Gamma \rhd \lambda x^{\alpha}.M : \alpha \Rightarrow \beta} \qquad \frac{\Gamma \rhd L : o \quad \Gamma \rhd M : \alpha \quad \Gamma \rhd N : \alpha}{\Gamma \rhd \mathsf{cond}\ L\ M\ N : \alpha} \qquad \frac{\Gamma, x : \alpha \Rightarrow \beta \rhd M : \alpha \Rightarrow \beta}{\Gamma \rhd \boldsymbol{\mu} x.M : \alpha \Rightarrow \beta}$$

As a set of constants, we assume: $n : \iota$ for each numeral n, $\Omega : \alpha$ for each α, $\mathsf{succ} : \iota \Rightarrow \iota$, and $\mathsf{zero?} : \iota \Rightarrow o$. Terms of form $\rhd M : \alpha$ (often written $M : \alpha$) are called *closed terms*. Abstractions and constants except Ω are called *values*.

On the set of terms we define an evaluation relation $\Downarrow$ in the style of natural semantics.

$$V \Downarrow V \qquad \frac{M \Downarrow \lambda x.M_0 \quad N \Downarrow V \quad M_0\{V/x\} \Downarrow U}{MN \Downarrow U} \qquad \frac{M\{\boldsymbol{\mu} x.M/x\} \Downarrow V}{\boldsymbol{\mu} x.M \Downarrow V} \qquad \frac{M \Downarrow n}{\mathsf{succ}\ M \Downarrow n+1}$$

$$\frac{M \Downarrow 0}{\mathsf{zero?}M \Downarrow \mathsf{true}} \qquad \frac{M \Downarrow n+1}{\mathsf{zero?}M \Downarrow \mathsf{false}} \qquad \frac{L \Downarrow \mathsf{true} \quad M_1 \Downarrow V}{\mathsf{cond}\ L\ M_1\ M_2 \Downarrow V} \qquad \frac{L \Downarrow \mathsf{false} \quad M_2 \Downarrow U}{\mathsf{cond}\ L\ M_1\ M_2 \Downarrow U}$$

Finally an *observational preorder* on closed terms is defined as follows: $M \preceq_{obs} N$ iff, for any well-typed context of a program type $C[\cdot]$, we have $C[M] \Downarrow n$ iff $C[N] \Downarrow n$. We note that this is the same thing as considering convergence at *all* types, a situation quite different from the case of call-by-name evaluation.