A Fully Complete PER Model for ML Polymorphic Types

Samson Abramsky[1] and Marina Lenisa[2] *

[1] LFCS, Division of Informatics, University of Edinburgh,
The King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, UK.
e-mail: `samson@dcs.ed.ac.uk`
[2] Dipartimento di Matematica e Informatica, Università di Udine,
Viale delle Scienze 206, 33100 Udine, ITALY.
e-mail: `lenisa@dimi.uniud.it`

Abstract. We present a *linear realizability* technique for building *Partial Equivalence Relations (PER)* categories over *Linear Combinatory Algebras*. These PER categories turn out to be *linear categories* and to form an *adjoint model* with their *co-Kleisli categories*. We show that a special linear combinatory algebra of *partial involutions*, arising from *Geometry of Interaction* constructions, gives rise to a *fully* and *faithfully complete* model for *ML polymorphic types* of system F.
Keywords: ML-polymorphic types, linear logic, PER models, Geometry of Interaction, full completeness.

Introduction

Recently, Game Semantics has been used to define fully-complete models for various fragments of Linear Logic ([AJ94a,AM99]), and to give fully-abstract models for many programming languages, including PCF [AJM96,HO96,Nic94], richer functional languages [McC96], and languages with non-functional features such as reference types and non-local control constructs [AM97,Lai97].

All these results are crucially based on the *linear* analysis of the *intuitionistic* arrow which is possible in the *intensional* setting of game categories. However, the definitions of game and game categories are quite complex, often requiring cumbersome quotienting operations. In this paper, we present the technique of *linear realizability* as a simpler and more direct alternative to game constructions for addressing *full completeness* issues.

The linear realizability technique amounts to constructing a category of *Partial Equivalence Relations (PERs)* over a *Linear Combinatory Algebra (LCA)*, which turns out to be a *linear category*, and to form an *adjoint model* with its co-Kleisli category. The notion of Linear Combinatory Algebra introduced by the first author ([Abr97a]) refines the standard notion of Combinatory Algebra, in the same way in which intuitionistic linear logic refines intuitionistic logic. The construction of PER models from LCA's presented in this paper is quite simple

* Work partially supported by TMR Linear FMRX-CT98-0170.

and clear, and it yields models with *extensionality* properties, thus avoiding the quotienting operations which are often needed in defining game categories and models. Moreover, PER categories offer simple natural models for second order (polymorphic) λ-calculus, i.e. Girard's System F ([Gir72]).

Recently, there has been much interest in realizability techniques, and in particular in *linear* realizability, especially in connection with full completeness and full abstraction problems. Realizability can be regarded as a powerful tool for mediating between *intensional* and *extensional* aspects of computation, and it has been used for extensionalizing intensional constructions (e.g. in [AM99]), and as a technique for building directly interesting (possibly fully-complete/fully-abstract) models. Examples of this latter use of realizability appear in this paper, and in [AL99], where a fully-abstract PER model for PCF, alternative to the game model of [AJM96], is provided using the algebra of *well-bracketed strategies*.

A *categorical* model of a type theory (or logic) is said to be *fully-complete* ([AJ94a]) if, for all *types* (*formulae*) A, B, all morphisms $f : [\![A]\!] \to [\![B]\!]$, from the interpretation of A into the interpretation of B, are denotations of a *proof-term* of the entailment $A \vdash B$, i.e. if the interpretation function from the category of syntactical objects to the category of denotations is *full*. The notion of full-completeness is the counterpart of the notion of *full abstraction*, in the sense that, if the term language is executable, then a fully-complete model is (up-to a possible quotient) fully-abstract.

Besides full completeness, one can ask the question whether the theory induced by a model $\mathcal{M}$ coincides precisely with the syntactical theory or whether more equations are satisfied in $\mathcal{M}$. A model $\mathcal{M}$ is called *faithful* if it realizes exactly the syntactical theory.

The fully and faithfully complete model for ML-types built in this paper is obtained as an instance of the PER construction, by considering the special linear combinatory algebra of *partial involutions*. ML-types are universal closures of simple types, i.e. types of the form $\forall X_1. \ldots X_n.T$, where T is $\forall$-free and $FV(T) \subseteq \{X_1, \ldots, X_n\}$. The algebra of partial involutions arises in the context of the generalization of Girard's *Geometry of Interaction* due to the first author ([AJ94,Abr96,Abr97a,AHPS98]). This is a powerful construction, which allows to build many new combinatory algebras, as well as to recover previously known models by viewing them in an alternative perspective. The algebra of partial involutions is a highly constrained algebra, in which all computations are *reversible*. Partial involutions are reminiscent of the *copy-cat* strategies of game categories, in that all the combinators mediate the required interactions between the arguments simply by copying information between input and output ports.

The proof of full completeness consists in showing that this model satisfies the axioms in the axiomatization of fully-complete models for ML-types given in [AL99a]. This axiomatization is given on the models of system F which are called *hyperdoctrines* ([Cro93]). In particular, it works in the context of *adjoint models*. It consists of two main steps. The first is an axiomatization of the fact that every morphism $f : 1 \to [\![T]\!]$, where T is an ML-type generates, under *decomposition*, a possibly *infinite* typed Böhm tree. Then, an axiom which rules out infinite trees from the model is introduced.

Proving that the model of partial involutions considered in this paper does not contain infinite typed Böhm trees is quite difficult, and it requires the study of an intermediate model. This is the model generated by the *Sierpinski PER* and it consists of all (possibly infinite) Böhm trees of the typed λ-calculus, with constants $\bot, \top$. A crucial step in our proof consists in proving that, in the simply typed λ-calculus with *typical ambiguity* and $\bot$-constants, "totality tests" are λ-definable by finite typed trees. These totality tests allow us to tell apart normal forms in which $\bot$ appears, from those in which $\bot$ does not appear. A further ingredient is an *Approximation Lemma*, along the lines of [AJM96].

The full completeness result obtained in this paper is interesting, since, until now, the research on full completeness for System F has produced fully-complete *denotational* models only for a small subclass of ML-types, i.e. the *algebraic types* (see [HRR90]). In the literature, there are two fully-complete models for the whole system F: i.e. that of [BC88], and that of [Hug99]. The first model is based on a quotient of a term model, the latter is a game model. But both these models still have a somewhat syntactical flavor, and their constructions are extremely complex. The model in this paper can be viewed as the first *denotational* model which is fully-complete for the whole class of ML-types.

In Section 1, we recall the syntax of ML types of system F, and we present two results on the simply typed λ-calculus with a theory satisfying *Typical Ambiguity*. The first is due to Statman, the latter is a new *Typed Separability* result. In Section 2, we recall the notion of $2\lambda\times$-hyperdoctrine and the notion of *adjoint hyperdoctrine* introduced in [AL99a], and we formalize the definition of fully-complete hyperdoctrine. In Section 3, we present the linear realizability technique, for building PER categories over LCAs. In Section 4, the LCA of partial involutions is described. In Section 5, the proof of full completeness for the PER model over the LCA of partial involutions is sketched. Final remarks and directions for future work appear in Section 6.

The authors are thankful to F.Honsell, R.Jagadeesan, J.Laird, J.Longley, S.Martini, G.Plotkin, A.Simpson for useful discussions on some of the issues of the paper.

1 ML Polymorphism

First, we recall the syntax of the class of *ML-types* of system F. Then, we present two important results on the simply typed λ-calculus with a theory satisfying *Typical Ambiguity*. A theory is said to satisfy Typical Ambiguity if two terms are equated if and only if they are equated for all possible substitutions of type variables. The first result that we present is Statman's *Typical Ambiguity Theorem*, which asserts that there is exactly one consistent theory satisfying Typical Ambiguity on the simply typed λ-calculus with infinitely many type variables: this is the $\beta\eta$-theory. An immediate consequence of this result is that the only consistent theory on the fragment of system F consisting of *ML-types* is precisely the $\beta\eta$-theory. The second result concerns the definability of "convergence tests" in the simply typed λ-calculus with infinitely many type variables, $\perp$-constants, and satisfying Typical Ambiguity. In particular, we prove that, for any given type, there are convergence test terms, which detect the presence of $\perp$-constants in a term of that type. This implies immediately that, in a theory of Typical Ambiguity over the simply typed λ-calculus with infinite type variables and $\perp$-constants, a term containing $\perp$ in its normal form can never be equated to a term in whose normal form $\perp$ does not appear. This result is used in the proof of full completeness of the model of PERs over the LCA of partial involutions.

We assume that the reader is familiar with System F (see e.g. [AL91]).

The class of *ML-polymorphic types* of system F corresponds to the limited kind of polymorphism allowed in the language ML.

Definition 1 (ML-types). *The class* ML-Type *of ML-types is defined by:*

$$\mathit{ML\text{-}Type} = \{\forall \boldsymbol{X}.T \mid T \in \ \mathit{SimType} \ \wedge \ FV(T) \subseteq \boldsymbol{X}\} \ ,$$

where SimType *is the class of* simple types *of system F, i.e. simple types over an infinite set of type variables, and $\boldsymbol{X}$ stands for $X_1, \ldots, X_n$, for $n \geq 0$.*

Terms of ML-types have essentially the same "combinatorics" as the typically ambiguous terms of the simply typed λ-calculus. In fact, any theory on ML-terms induces a theory satisfying Typical Ambiguity.

The following is a result about simply typed λ-calculus with *infinitely many* type variables λ^∞, first proved in [Sta88].

Theorem 1 (Statman's Typical Ambiguity). *Let T be a type of λ^∞ s.t. $FV(T) \subseteq \{X_1, \ldots, X_n\}$. If $\not\vdash M =_{\beta\eta} N \ : \ T$, then, there exist types $S_1, \ldots, S_n$, and $Y \in TVar$, and a term L s.t. $\vdash L[\boldsymbol{S}/\boldsymbol{X}] : T[\boldsymbol{S}/\boldsymbol{X}] \to Bool_Y$, where $Bool_Y = Y \to Y \to Y$, s.t.*

$$\vdash (LM)[\boldsymbol{S}/\boldsymbol{X}] =_{\beta\eta} \mathit{true} \ : \ Bool_Y \ \wedge \ \vdash (LN)[\boldsymbol{S}/\boldsymbol{X}] =_{\beta\eta} \mathit{false} \ : \ Bool_Y \ ,$$

where $\mathit{true} = \lambda x : Y.y : Y.x$ and $\mathit{false} = \lambda x : Y.y : Y.y$.

Corollary 1. *i) The maximal consistent theory satisfying Typical Ambiguity on the simply typed λ-calculus with infinitely many type variables is the $\beta\eta$-theory.*
ii) The maximal consistent theory on the fragment of system F consisting of ML-types is the $\beta\eta$-theory.

As it will be clear in the following section from the definition of full completeness, by Corollary 1ii), any non-trivial fully-complete model for ML-types of system F is necessarily *faithful*, i.e. it realizes exactly the $\beta\eta$-theory at ML-types.

Now we show that "convergence tests" are λ-definable in the simply typed λ-calculus with infinitely many type variables and $\perp$-constants for any type variable, and a theory satisfying Typical Ambiguity, which we call $\lambda^\infty_\perp$.

Definition 2 (Typed Convergence Tests). *Let $T = T_1 \to \ldots \to T_n \to X_k \in \mathit{SimType}$, let ι be a distinguished type variable, and let $\alpha_T = T[\iota \to \iota/\boldsymbol{X}]$. We define, by induction on T, the* convergence test *term $\vdash S_{\alpha_T} : \alpha_T$ as follows:*
if $T = X$, then $\quad S_{\iota\to\iota} = I_{\iota\to\iota}$, $\quad$ otherwise,
let $T = T_1 \to \ldots \to T_n \to X_k$, where $T_i = U_{i1} \to \ldots \to U_{iq_i} \to X_i$, then

$$S_{\alpha_T} = \lambda x_1 : \alpha_{T_1} \ldots x_n : \alpha_{T_n}.\lambda z : \iota.(x_1 S_{\alpha_{U_{11}}} \ldots S_{\alpha_{U_{1q_1}}})(\ldots(x_n S_{\alpha_{U_{n1}}} \ldots S_{\alpha_{U_{nq_n}}} z)) \ .$$

The "convergence test" terms defined above give us a procedure for deciding whether a normal form of $\lambda_\perp^\infty$ contains a divergent subterm. Namely, let M be a normal form of $\lambda_\perp^\infty$ of type $T_1 \to \ldots \to T_n \to X_k$. We first instantiate all the free variables in M by $\iota \to \iota$, then we apply M to the sequence of convergence tests $S_{\alpha_{T_1}}, \ldots, S_{\alpha_{T_n}}$. The effect of this is that, in the head reduction of $MS_{\alpha_{T_1}}, \ldots, S_{\alpha_{T_n}}$, each subterm of M definitely appears in head position, and it reduces to the identity, until a $\perp$ is detected.

For a term $\boldsymbol{y} : \boldsymbol{U} \vdash M : T$, we denote by $\boldsymbol{y} : \boldsymbol{\alpha_U} \vdash M_{\alpha_T} : \alpha_T$ (or simply by M_{α_T}) the term of type α_T obtained from $\boldsymbol{y} : \boldsymbol{U} \vdash M : T$ by instantiating all the type variables free in T by $\iota \to \iota$.

Theorem 2 (Typed Separability). *Let $T = T_1 \to \ldots \to T_n \to X_k \in SimType$, and let $\vdash M : T$ be a term of $\lambda_\perp^\infty$. Then*

$$M_{\alpha_T} S_{\alpha_{T_1}} \ldots S_{\alpha_{T_n}} = \begin{cases} I_{\iota\to\iota} & \textit{if the normal form of } M \textit{ is } \perp\textit{-free} \\ \lambda x : \iota.\, \perp & \textit{otherwise}\ . \end{cases}$$

Theorem 2 above can be regarded as a *typed* Böhm-like Separability Theorem, in the sense that, if we think of $\perp$ as a generic *unsolvable* term, then Theorem 2 allows us to tell apart normal forms from unsolvable terms.

Corollary 2. *In any theory satisfying Typical Ambiguity on $\lambda_\perp^\infty$, a term in whose normal form $\perp$ appears cannot be equated to a term, in whose normal form $\perp$ does not appear.*

2 Models of System F

We recall first the notion of $2\lambda\times$-hyperdoctrine (see [Cro93]). This essentially corresponds to the notion of *external model* (see [AL91]). Then, we give the formal definition of fully (and faithfully) complete hyperdoctrine model. Finally, we define the categorical notion of *adjoint hyperdoctrine*, on which the axiomatization of full completeness at ML-types of [AL99a] is given. Adjoint hyperdoctrines arise as *co-Kleisli* indexed categories of *linear* indexed categories.

In what follows, we assume that all indexed categories which we consider are strict (see e.g. [AL91,Cro93] for more details).

Definition 3 ($2\lambda\times$-hyperdoctrine). *A* $2\lambda\times$-hyperdoctrine *is a triple $(\mathcal{C}, \boldsymbol{G}, \forall)$, where:*
- *$\mathcal{C}$ is the* base category*, it has with finite products, and it consists of a distinguished object $\mathcal{U}$ which generates all other objects using the product operation $\times$. We will denote by $\mathcal{U}^m$, for $m \geq 0$, the objects of $\mathcal{C}$.*
- *$\boldsymbol{G} : \mathcal{C}^{op} \to$ CCCat is a $\mathcal{C}$-indexed cartesian closed category, where* CCCat *is the category of cartesian closed categories and strict cartesian closed functors, such that: for all $\mathcal{U}^m$, the collection of objects of the cartesian closed fibre category $\boldsymbol{G}(\mathcal{U}^m)$ is indexed by the morphisms from $\mathcal{U}^m$ to $\mathcal{U}$ in $\mathcal{C}$, i.e. the objects of $\boldsymbol{G}(\mathcal{U}^m)$ are the morphisms in $Hom_{\mathcal{C}}(\mathcal{U}^m,\mathcal{U})$, and, for any $f : \mathcal{U}^m \to \mathcal{U}^n$ in $\mathcal{C}^{op}$, the cartesian closed functor $\boldsymbol{G}(f) : \boldsymbol{G}(\mathcal{U}^n) \to \boldsymbol{G}(\mathcal{U}^m)$, called* reindexing functor *and denoted by f^*, is s.t., for any object $h : \mathcal{U}^n \to \mathcal{U}$, $f^*(h) = f; h$;*

– *For each object $\mathcal{U}^m$ of $\mathcal{C}$, there are functors* $\forall_m : \boldsymbol{G}(\mathcal{U}^m \times \mathcal{U}) \to \boldsymbol{G}(\mathcal{U}^m)$ *s.t.*
  - $\forall_m$ *is right adjoint to the functor* $\pi_m^* : \boldsymbol{G}(\mathcal{U}^m) \to \boldsymbol{G}(\mathcal{U}^m \times \mathcal{U})$, *where* $\pi_m : \mathcal{U}^m \times \mathcal{U} \to \mathcal{U}^m$ *is the projection in* $\mathcal{C}$*;*
  - $\forall_m$ *satisfies the* Beck-Chevalley condition.

Any $2\lambda\times$-hyperdoctrine can be endowed with a notion of interpretation $[\![\ ]\!]$ for the language of system F.
Types with free variables in $X_1, \ldots, X_m$ are interpreted by objects of $\mathbf{G}(\mathcal{U}^m)$, i.e. by morphisms from $\mathcal{U}^m$ to $\mathcal{U}$ in $\mathcal{C}$: $[\![X_1, \ldots, X_m \vdash T]\!] : \mathcal{U}^m \to \mathcal{U}$.
Well-typed terms, i.e. $X_1, \ldots, X_m; x_1 : T_1, \ldots, x_n : T_n \vdash M : T$, are interpreted by morphisms in the category $\mathbf{G}(\mathcal{U}^m)$:
$[\![X_1, \ldots, X_m; x_1 : T_1, \ldots, x_n : T_n \vdash M : T]\!] : [\![\boldsymbol{X} \vdash T_1]\!] \times \ldots \times [\![\boldsymbol{X} \vdash T_n]\!] \to [\![\boldsymbol{X} \vdash T]\!]$.

Definition 4 (Full (and Faithful) Completeness). *Let* $\mathcal{M} = (\mathcal{C}, \boldsymbol{G}, \forall, [\![\ ]\!])$ *be a* $2\lambda\times$*-hyperdoctrine.* $\mathcal{M}$ *is* fully and faithfully complete *w.r.t. the class of closed types* $\mathcal{T}$ *if, for all* $T \in \mathcal{T}$,
$\forall f \in Hom_{\boldsymbol{G}(1)}(1, [\![\vdash T]\!])$. $\exists(!)\beta\eta$*-normal form* M. $\vdash M : T \ \wedge \ f = [\![\vdash M : T]\!]$.

In the following definition, we capture those $2\lambda\times$-hyperdoctrines which arise from a co-Kleisli construction over an *indexed linear category*, and on which the axiomatization of fully-complete models for ML-types given in [AL99a] is based.

Definition 5 (Adjoint Hyperdoctrine).
An adjoint hyperdoctrine *is a quadruple* $(\mathcal{C}, \boldsymbol{L}, \boldsymbol{G}, \forall)$, *where:*

– $\mathcal{C}$ *is the* base category, *it has finite products, which consists of a distinguished object* $\mathcal{U}$ *which generates all other objects using the product operation* $\times$*. We will denote by* $\mathcal{U}^m$, *for* $m \geq 0$, *the objects of* $\mathcal{C}$.
– $\boldsymbol{L} : \mathcal{C}^{op} \to \mathrm{LCat}$ *is a* $\mathcal{C}$*-indexed linear category, where* LCat *is the category of linear categories and strict monoidal closed functors preserving the comonad structure, s.t.: for all* $\mathcal{U}^m$, *the underlying collection of objects of the linear fibre category* $\boldsymbol{L}(\mathcal{U}^m)$ *is indexed by the morphisms from* $\mathcal{U}^m$ *to* $\mathcal{U}$ *in* $\mathcal{C}$.
– $\boldsymbol{G} : \mathcal{C}^{op} \to \mathrm{CCCat}$ *is the* $\mathcal{C}$*-indexed co-Kleisli category of* $\boldsymbol{L}$, *which we assume to be cartesian closed.*
– *For each object* $\mathcal{U}^m$ *of* $\mathcal{C}$, *there are functors* $\forall_m : \boldsymbol{G}(\mathcal{U}^m \times \mathcal{U}) \to \boldsymbol{G}(\mathcal{U}^m)$ *s.t.*
  - $\forall_m : \boldsymbol{G}(\mathcal{U}^m \times \mathcal{U}) \to \boldsymbol{G}(\mathcal{U}^m)$ *is right adjoint to the functor* $\boldsymbol{G}(\pi_m) : \boldsymbol{G}(\mathcal{U}^m) \to \boldsymbol{G}(\mathcal{U}^m \times \mathcal{U})$, *where* $\pi_m : \mathcal{U}^m \times \mathcal{U} \to \mathcal{U}^m$ *is the projection in* $\mathcal{C}$*;*
  - $\forall_m : \boldsymbol{G}(\mathcal{U}^m \times \mathcal{U}) \to \boldsymbol{G}(\mathcal{U}^m)$ *satisfies the* Beck-Chevalley condition.

3 Models of PERs over a Linear Combinatory Algebra

Canonical examples of $2\lambda\times$-hyperdoctrines arise from considering the *Partial Equivalence Relation* (PER) category over a *combinatory algebra* (see [Cro93], Chapter 5, Section 5.5 for more details). In this section, we show how to build a PER category from a linear combinatory algebra (LCA). This category turns out to form an adjoint model with its co-kleisli category, and it gives rise to an adjoint hyperdoctrine.
We start by recalling the definition of linear combinatory algebra ([Abr97a,AHPS98]):

Definition 6 (Linear Combinatory Algebra). *A* linear combinatory algebra *$\mathcal{A} = (A, \bullet, !)$ is an applicative structure $(A, \bullet)$ with a unary (injective) operation $!$, and distinguished elements (combinators) $B, C, I, K, W, D, \delta, F$ satisfying the following equations:*

Equation	**Principal type**	**Logical rule**
$\boldsymbol{I}x = x$	$\alpha \multimap \alpha$	Identity
$\boldsymbol{B}xyz = x(yz)$	$(\alpha \multimap \beta) \multimap (\gamma \multimap \alpha) \multimap \gamma \multimap \beta$	Cut
$\boldsymbol{C}xyz = (xz)y$	$(\alpha \multimap \beta \multimap \gamma) \multimap \beta \multimap \alpha \multimap \gamma$	Exchange
$\boldsymbol{K}x!y = x$	$\alpha \multimap !\beta \multimap \alpha$	Weakening
$\boldsymbol{W}x!y = x!y!y$	$(!\alpha \multimap !\alpha \multimap \beta) \multimap !\alpha \multimap \beta$	Contraction
$\boldsymbol{D}!x = x$	$!\alpha \multimap \alpha$	Dereliction
$\delta !x = !!x$	$!\alpha \multimap !!\alpha$	Comultiplication
$\boldsymbol{F}!x!y = !(xy)$	$!(\alpha \multimap \beta) \multimap !\alpha \multimap !\beta$	Closed Functoriality .

LCA's correspond to *Hilbert style* axiomatization of $\multimap, !$ fragment of Linear Logic. Given an LCA $\mathcal{A} = (A, \bullet, !)$, we can form a standard CA $\mathcal{A}_s = (A, \bullet_s)$ by the "combinatory version" of Girard's translation of Intuitionistic Logic into Linear Logic. We define: $\alpha \bullet_s \beta = \alpha \bullet ! \beta$ (standard combinators can be defined in terms of the linear ones, see [AHPS98] for details).

We recall that a *BCI-algebra* is an applicative structure $(A, \bullet)$ with B, C, I combinators. In the next definition, we define a PER category over a BCI-algebra, which turns out to be symmetric monoidal closed.

Definition 7. *Let $\mathcal{A} = (A, \bullet)$ be a BCI-algebra. We define the category $PER_{\mathcal{A}}$ as follows.*
Objects*: PERs $\mathcal{R} \subseteq A \times A$, i.e. symmetric and transitive relations.*
Morphisms*: a morphism f from $\mathcal{R}$ to $\mathcal{S}$ is an equivalence class of the PER $\mathcal{R} \multimap \mathcal{S}$, where the PER $\mathcal{R} \multimap \mathcal{S}$ is defined by*

$$\alpha(\mathcal{R} \multimap \mathcal{S})\beta \quad \textit{iff} \quad \forall \gamma \, \mathcal{R} \, \gamma'. \; \alpha \bullet \gamma \; \mathcal{S} \; \beta \bullet \gamma' \; .$$

On BCI-algebras, standard *pairing* gives rise to a *tensor product*, but the definition of tensor product requires some care:

Lemma 1. *Let $\mathcal{A} = (A, \bullet)$ be a BCI-algebra. Let P be the* pairing combinator*, i.e. (using λ-notation) $P = \lambda xyz.zxy$. Then, for all PERs $\mathcal{R}, \mathcal{S}$, let $\mathcal{R} \otimes \mathcal{S}$ be the PER defined as the* transitive closure *of the following relation:*

$$\mathcal{R} \otimes' \mathcal{S} = \{(P\alpha\beta, P\alpha'\beta') \mid \alpha \, \mathcal{R} \, \alpha' \;\wedge\; \beta \, \mathcal{S} \, \beta'\} \; .$$

Notice in particular that, if the BCI-algebra is *affine*, i.e. it is a BCK-algebra, then the relation $\mathcal{R} \otimes' \mathcal{S}$ is already transitive, since, using projections, we get: $P\alpha\beta = P\alpha'\beta' \implies \alpha = \alpha' \;\wedge\; \beta = \beta'$.

Proposition 1. *Let $\mathcal{A} = (A, \bullet)$ be a BCI-algebra. Then $PER_{\mathcal{A}}$ is a symmetric monoidal closed category.*

Now we show how an LCA gives rise to a linear category.

Proposition 2. *Let $\mathcal{A} = (A, \bullet, !)$ be an LCA. Let $\boldsymbol{!} : PER_{\mathcal{A}} \to PER_{\mathcal{A}}$ be the functor defined by*

$$\forall\, \mathcal{R}\ .\ \boldsymbol{!}\,\mathcal{R} = \{(!\alpha, !\beta) \mid \alpha\ \mathcal{R}\ \beta\}, \quad \forall f : \mathcal{R}_1 \to \mathcal{R}_2\ .\ \boldsymbol{!}f = [F!f]\ .$$

Then $(\boldsymbol{!}, D, \delta, \phi, \phi')$ is a symmetric monoidal comonad, where

- $\phi_{\mathcal{R}_1, \mathcal{R}_2} : \boldsymbol{!}\,\mathcal{R}_1 \otimes \boldsymbol{!}\,\mathcal{R}_2 \to \boldsymbol{!}(\mathcal{R}_1 \otimes \mathcal{R}_2)$ *is defined by* $\phi_{\mathcal{R}_1, \mathcal{R}_2} = [\lambda u.F!P(uF)]$;
- $\phi' : I \simeq \boldsymbol{!}I$ *is* $[\delta]_{I \to \boldsymbol{!}I}$.

The following isomorphisms hold immediately in PER categories over LCA's:

Lemma 2. *Let $\mathcal{A} = (A, \bullet, !)$ be an LCA. Then, for all PERs $\mathcal{R}, \mathcal{S}$,*

1. (Idempotency of !) $[D] : \boldsymbol{!!}\,\mathcal{R} \simeq \boldsymbol{!}\,\mathcal{R} : [\delta]$;
2. (Uniformity of Threads) $\psi : \boldsymbol{!}\,\mathcal{R} \multimap \boldsymbol{!}\,\mathcal{S} \simeq \boldsymbol{!}\,\mathcal{R} \multimap \mathcal{S} : (\cdot)^{\dagger}$ *, where* $\psi = [\lambda x.x; D]$*; or equivalently:* $\forall \alpha \in \boldsymbol{!}\,\mathcal{R} \multimap \boldsymbol{!}\,\mathcal{S}$, $(\alpha; [D])^{\dagger} = \alpha$*;*
3. (Commutativity of $\bigcap$ w.r.t. !) $\bigcap_X !\, \mathcal{R} \simeq !(\bigcap_X \mathcal{R})$.

The second isomorphism in Lemma 2 above is relevant for full completeness. In fact, this isomorphism amounts exactly to the *Uniformity of Threads Axiom* in the axiomatization of full completeness of [AL99a]. The isomorphisms of Lemma 2 above highlight the fact that the PER category is a "degenerate" model of linear logic.

Theorem 3. *Let $\mathcal{A} = (A, \bullet, !)$ be an LCA. Then*

- *The category $PER_{\mathcal{A}}$ is linear.*
- *The co-Kleisli category $(PER_{\mathcal{A}})_{!}$, induced by the comonad $\boldsymbol{!}$ on the category $PER_{\mathcal{A}}$, is cartesian closed.*
- *The categories $PER_{\mathcal{A}}$ and $(PER_{\mathcal{A}})_{!}$ form an adjoint model.*
- *The category $(PER_{\mathcal{A}})_{!}$ is isomorphic to the category $PER_{\mathcal{A}_s}$, where $PER_{\mathcal{A}_s}$ is the category obtained by standard realizability from the standard combinatory algebra $\mathcal{A}_s$.*

Finally, we show how to build an adjoint hyperdoctrine from an LCA:

Theorem 4 (PER Adjoint Hyperdoctrine). *Let $\mathcal{A} = (A, \bullet, !)$ be an LCA. Then $\mathcal{A}$ gives rise to an adjoint hyperdoctrine $(\mathcal{C}, \boldsymbol{L}, \boldsymbol{G}, \forall)$, by defining:*

$\mathcal{C}$ **:** *Let $\mathcal{U}$ be the set $\{\mathcal{R} \mid \mathcal{R}$ is a PER on $A\}$. The objects of $\mathcal{C}$, $\mathcal{U}^n$, for $n \geq 0$, are the finite products in* Set *of n copies of the set $\mathcal{U}$, in particular $\mathcal{U}^0$ is the terminal object in* Set*. A morphism in $\mathcal{C}$, $f : \mathcal{U}^n \to \mathcal{U}^m$, is a set-theoretic function from $\mathcal{U}^m$ to $\mathcal{U}^n$.*

L : *The morphisms in the fibre category $\boldsymbol{L}(\mathcal{U}^m)$ from $h_1 : \mathcal{U}^m \to \mathcal{U}$ to $h_2 : \mathcal{U}^m \to \mathcal{U}$ are the equivalence classes of the PER $\bigcap_{\boldsymbol{X} \in \mathcal{U}^m}(h_1\boldsymbol{X} \multimap h_2\boldsymbol{X})$. For any object $f : \mathcal{U}^m \to \mathcal{U}$ in $\boldsymbol{L}(\mathcal{U}^m)$, we define $!f$ to be $\lambda\boldsymbol{X}.!(f\boldsymbol{X})$. For any morphism $f : \mathcal{U}^m \to \mathcal{U}^n$ in $\mathcal{C}$, we define the behavior of the functor $\boldsymbol{L}(f) : \boldsymbol{L}(\mathcal{U}^n) \to \boldsymbol{L}(\mathcal{U}^m)$ on morphisms by: for any morphism $H : h_1 \to h_2$ in $\boldsymbol{L}(\mathcal{U}^n)$, $H = \Lambda\boldsymbol{X}.H' \in \bigcap_{\boldsymbol{X}}(h_1\boldsymbol{X} \multimap h_2\boldsymbol{X})$, let $\boldsymbol{L}(f)(H) : \boldsymbol{L}(f)(h_1) \to \boldsymbol{L}(f)(h_2)$ be $\Lambda\boldsymbol{X}.H' \circ f(\boldsymbol{X}) \in \bigcap_{\boldsymbol{X}}(\boldsymbol{L}(f)(h_1)\boldsymbol{X} \multimap \boldsymbol{L}(f)(h_2)\boldsymbol{X})$.*

$\forall$ **:** *The functor $\forall_m : \boldsymbol{L}(\mathcal{U}^m \times \mathcal{U}) \to \boldsymbol{L}(\mathcal{U}^m)$ is defined as follows. For any $h : \mathcal{U}^m \times \mathcal{U} \to \mathcal{U}$, $\forall_m(h) = \lambda\boldsymbol{X}.\bigcap_{\boldsymbol{X}} h(\boldsymbol{X})$. For any morphism $H : h_1 \to h_2$ in $\boldsymbol{L}(\mathcal{U}^m \times \mathcal{U})$, $\forall_m(H) = H$.*

4 Partial Involutions Affine Combinatory Algebra

Many examples of LCAs arise from the categorical version of Girard's Geometry of Interaction (GoI) construction, based on *traced symmetric monoidal* categories ([Abr97a,Abr96,AHPS98]). A basic example of GoI LCA, introduced in [Abr97a], can be defined on the space $[\mathbf{N} \rightharpoonup \mathbf{N}]$ of partial functions from natural numbers into natural numbers, by applying the GoI construction to the the traced category *Pfn* of sets and partial functions. Here we briefly recall the definition of this LCA, without discussing the categorical framework (see [Abr97a,Abr96,AHPS98] for more details). The LCA of partial involutions, which will be shown to provide a fully-complete model for ML-types (see Section 5), arises as subalgebra of this.

Let us consider the space $[\mathbf{N} \rightharpoonup \mathbf{N}]$ of partial functions from natural numbers to natural numbers. For any $\alpha \in [\mathbf{N} \rightharpoonup \mathbf{N}]$ injective, we denote by α^{-1} the inverse of α. Now we show how we can endow the space $[\mathbf{N} \rightharpoonup \mathbf{N}]$ with a structure of LCA. Actually, the algebra which we obtain is *affine*, i.e. it has a full $\boldsymbol{K}$-combinator. We start by fixing two injective *coding* functions t and p:

$$t : \mathbf{N} + \mathbf{N} \to \mathbf{N} \quad , \quad p : \mathbf{N} \times \mathbf{N} \to \mathbf{N} \ .$$

The first is used in order to define application, and it allows to transform an one-input/one-output function into a two-input/two-output function. The latter is used for creating *infinitely* many copies of an one-input/one-output function α, i.e. for defining $!\alpha$.

We now explain how application is computed *geometrically*, using the language of "boxes and wires" which arises in the general setting of traced symmetric monoidal categories (see [JSV96] for an abstract treatment).

Let us represent an one-input/one-output function $\alpha \in [\mathbf{N} \rightharpoonup \mathbf{N}]$ by the following one-input-port/one-output-port box (see Fig. 1(i) below).

In order to define the application $\alpha \bullet \beta$, for $\alpha, \beta \in [\mathbf{N} \rightharpoonup \mathbf{N}]$, we regard α as a two-input/two-output function via the coding t. In particular, $t; \alpha; t^{-1} : \mathbf{N}+\mathbf{N} \rightharpoonup \mathbf{N} + \mathbf{N}$ can be described as a matrix of 4 one-input/one-output functions, where each entry $\alpha_{ij} : \mathbf{N} \rightharpoonup \mathbf{N}$, $\alpha_{ij} = in_i; t; \alpha; t^{-1}; in_j^{-1}$ accounts for the contribution from the i-th input wire into the j-th output wire (see Fig. 1(ii)).

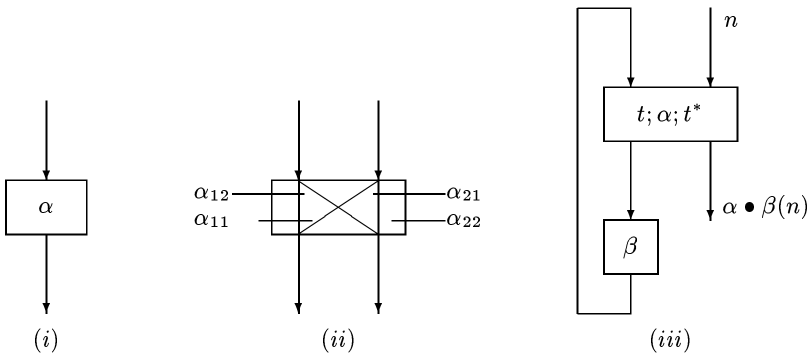


Fig. 1. Geometrical description of linear application.

The result of the application $\alpha \bullet \beta$ is the following one-input/one-output function (see Fig. 1(iii)):

$$\alpha \bullet \beta = \alpha_{22} \cup \alpha_{21}; (\beta; \alpha_{11})^\star; \beta; \alpha_{12} \ ,$$

where $\cup$ denotes union of graph relations, and $(\beta; \alpha_{11})^\star$ denotes $\bigcup_{n \geq 0} (\beta; \alpha_{11})^n$.

The above formula for computing the application is essentially the *Execution Formula* from Girard's Geometry of Interaction ([Gir89]).

The definition of the !-operation on our applicative structure is quite simple. The operation ! is intended to produce, from a single copy of α, *infinitely* many copies of α. These are obtained by simply tagging each of these copies with a natural number, i.e. we define:

$$!\alpha = p^{-1}; (id_{\mathbf{N}} \times \alpha); p \ .$$

Finally, we are left to show that (affine) combinators can be defined on the structure $([\mathbf{N} \rightharpoonup \mathbf{N}], \bullet, !)$. The formal (algebraic) definition of the combinators is the following:

Definition 8 (Combinators). *For* $\boldsymbol{X} \in \{\boldsymbol{I}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{K}, \boldsymbol{W}, \boldsymbol{D}, \delta, \boldsymbol{F}\}$*, let*

$$\boldsymbol{X} = s_{\boldsymbol{X}}^{-1}; f_{\boldsymbol{X}}; s_{\boldsymbol{X}} \ ,$$

where:

I : – $s_{\boldsymbol{I}} = t$.
- $f_{\boldsymbol{I}} : \boldsymbol{N} + \boldsymbol{N} \rightharpoonup \boldsymbol{N} + \boldsymbol{N}$ *is defined by:*
  $\forall n.\ f_{\boldsymbol{I}}(r, n) = (l, n) \quad \wedge \quad \forall n.\ f_{\boldsymbol{I}}(l, n) = (r, n)$.

B : – $s_{\boldsymbol{B}} : (((\boldsymbol{N} + \boldsymbol{N}) + (\boldsymbol{N} + \boldsymbol{N})) + \boldsymbol{N}) + \boldsymbol{N} \rightharpoonup \boldsymbol{N}$ *is defined by*
  $s_{\boldsymbol{B}} = ((t + t) + id_{\boldsymbol{N}}) + id_{\boldsymbol{N}}; (t + id_{\boldsymbol{N}}) + id_{\boldsymbol{N}}; t + id_{\boldsymbol{N}}; t$.
- $f_{\boldsymbol{B}} : (((\boldsymbol{N}+\boldsymbol{N})+(\boldsymbol{N}+\boldsymbol{N}))+\boldsymbol{N})+\boldsymbol{N} \rightharpoonup (((\boldsymbol{N}+\boldsymbol{N})+(\boldsymbol{N}+\boldsymbol{N}))+\boldsymbol{N})+\boldsymbol{N}$ *is the function defined by the following equations together with their symmetric closure:*
  - $\forall n.\ f_{\boldsymbol{B}}(r, n) = (l, (l, (l, (r, n))))$
  - $\forall n.\ f_{\boldsymbol{B}}(l, (l, (l, (l, n)))) = (l, (l, (r, (r, n))))$
  - $\forall n.\ f_{\boldsymbol{B}}(l, (l, (r, (l, n)))) = (l, (r, n))$.

C : – $s_{\boldsymbol{C}} : ((\boldsymbol{N} + \boldsymbol{N}) + ((\boldsymbol{N} + \boldsymbol{N}) + \boldsymbol{N})) + \boldsymbol{N} \rightharpoonup \boldsymbol{N}$ *is defined by*
  $s_{\boldsymbol{C}} = (t + (t + id_{\boldsymbol{N}})) + id_{\boldsymbol{N}}; (t + t) + id_{\boldsymbol{N}}; t + id_{\boldsymbol{N}}; t$.
- $f_{\boldsymbol{C}} : ((\boldsymbol{N}+\boldsymbol{N})+((\boldsymbol{N}+\boldsymbol{N})+\boldsymbol{N}))+\boldsymbol{N} \rightharpoonup ((\boldsymbol{N}+\boldsymbol{N})+((\boldsymbol{N}+\boldsymbol{N})+\boldsymbol{N}))+\boldsymbol{N}$ *is the function defined by the following equations together with their symmetric closure:*
  - $\forall n.\ f_{\boldsymbol{C}}(r, n) = (l, (r, (r, n)))$
  - $\forall n.\ f_{\boldsymbol{C}}(l, (r, (l, (r, n)))) = (l, (l, (r, n)))$
  - $\forall n.\ f_{\boldsymbol{C}}(l, (r, (l, (l, n)))) = (l, (l, (l, n)))$.

K : – $s_{\boldsymbol{K}} : (\boldsymbol{N} + \boldsymbol{N}) + \boldsymbol{N} \rightharpoonup \boldsymbol{N}$ *is defined by* $s_{\boldsymbol{K}} = t + id_{\boldsymbol{N}}; t$.
- $f_{\boldsymbol{K}} : (\boldsymbol{N} + \boldsymbol{N}) + \boldsymbol{N} \rightharpoonup (\boldsymbol{N} + \boldsymbol{N}) + \boldsymbol{N}$ *is the function defined by:*
  $\forall n.\ f_{\boldsymbol{K}}(r, n) = (l, (r, n)) \quad \wedge \quad \forall n.\ f_{\boldsymbol{K}}(l, (r, n)) = (r, n)$.

W : *In order to define* $\boldsymbol{W}$*, we need first to fix* $i, j \in \boldsymbol{N}$ *such that* $i \neq j$*. Then*
- $s_{\boldsymbol{W}} : ((\boldsymbol{N} \times \boldsymbol{N}) + ((\boldsymbol{N} + \boldsymbol{N}) + \boldsymbol{N})) + \boldsymbol{N} \rightharpoonup \boldsymbol{N}$ *is defined by*
  $s_{\boldsymbol{W}} = (p + (t + id_{\boldsymbol{N}})) + id_{\boldsymbol{N}}; (id_{\boldsymbol{N}} + t) + id_{\boldsymbol{N}}; t + id_{\boldsymbol{N}}; t$.
- $f_{\boldsymbol{W}} : ((\boldsymbol{N} \times \boldsymbol{N}) + ((\boldsymbol{N} + \boldsymbol{N}) + \boldsymbol{N})) + \boldsymbol{N} \rightharpoonup ((\boldsymbol{N} \times \boldsymbol{N}) + ((\boldsymbol{N} + \boldsymbol{N}) + \boldsymbol{N})) + \boldsymbol{N}$ *is the function defined by the following equations together with their symmetric closure:*

- $\forall n.\ f_{\boldsymbol{W}}(r,n) = (l,(r,(r,n)))$
- $\forall n.\ f_{\boldsymbol{W}}(l,(r,(l,(r,n)))) = (l,(l,(i,n)))$
- $\forall n.\ f_{\boldsymbol{W}}(l,(r,(l,(l,n)))) = (l,(l,(j,n)))$.

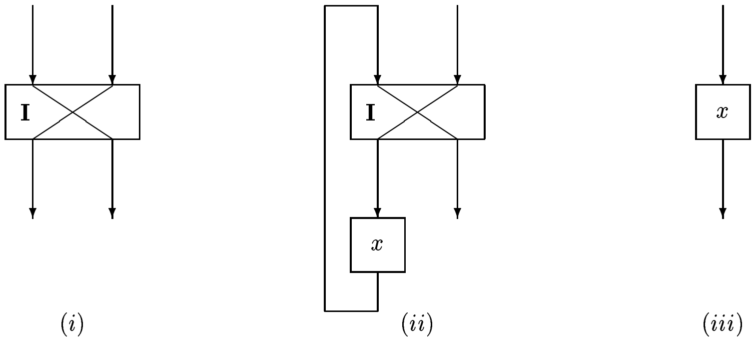
D : *In order to define* $\boldsymbol{D}$, *we need to fix* $i \in \boldsymbol{N}$. *Then*
- $s_{\boldsymbol{D}} : (\boldsymbol{N} \times \boldsymbol{N}) + \boldsymbol{N} \rightharpoonup \boldsymbol{N}$ *is defined by* $s_{\boldsymbol{D}} = p + id_{\boldsymbol{N}}; t$.
- $f_{\boldsymbol{D}} : (\boldsymbol{N} \times \boldsymbol{N}) + \boldsymbol{N} \rightharpoonup (\boldsymbol{N} \times \boldsymbol{N}) + \boldsymbol{N}$ *is the function defined by:* $\forall n.\ f_{\boldsymbol{D}}(r,n) = (l,(i,n)) \quad \wedge \quad \forall n.\ f_{\boldsymbol{D}}(l,(i,n)) = (r,n)$.

δ : *In order to define* δ, *we need to fix* $i, j \in \boldsymbol{N}$. *Then*
- $s_\delta : (\boldsymbol{N} \times (\boldsymbol{N} \times \boldsymbol{N})) + \boldsymbol{N} \rightharpoonup \boldsymbol{N}$ *is defined by* $s_\delta = (id_{\boldsymbol{N}} \times p) + id_{\boldsymbol{N}}; p + id_{\boldsymbol{N}}; t$.
- $f_\delta : (\boldsymbol{N} \times (\boldsymbol{N} \times \boldsymbol{N})) + \boldsymbol{N} \rightharpoonup (\boldsymbol{N} \times (\boldsymbol{N} \times \boldsymbol{N})) + \boldsymbol{N}$ *is the function defined by:* $\forall n.\ f_\delta(r,n) = (l,(i,(j,n))) \quad \wedge \quad \forall n.\ f_\delta(l,(i,(j,n))) = (r,n)$.

F : *In order to define* $\boldsymbol{F}$, *we need to fix* $i, j \in \boldsymbol{N}$. *Then*
- $s_{\boldsymbol{F}} : ((\boldsymbol{N} \times \boldsymbol{N}) + \boldsymbol{N} \times (\boldsymbol{N} + \boldsymbol{N})) + \boldsymbol{N} \rightharpoonup \boldsymbol{N}$ *is defined by* $s_{\boldsymbol{F}} = (p + (id_{\boldsymbol{N}} \times t)) + id_{\boldsymbol{N}}; (id_{\boldsymbol{N}} + p) + id_{\boldsymbol{N}}; t$.
- $f_{\boldsymbol{F}} : ((\boldsymbol{N} \times \boldsymbol{N}) + \boldsymbol{N} \times (\boldsymbol{N} + \boldsymbol{N})) + \boldsymbol{N} \rightharpoonup ((\boldsymbol{N} \times \boldsymbol{N}) + \boldsymbol{N} \times (\boldsymbol{N} + \boldsymbol{N})) + \boldsymbol{N}$ *is the function defined by the following equations together with their symmetric closure:*
  - $\forall n.\ f_{\boldsymbol{F}}(r,n) = (l,(r,(i,(r,n))))$
  - $\forall n.\ f_{\boldsymbol{F}}(l,(r,(i,(l,n)))) = (l,(l,(j,n)))$.

There is a simple, intuitive, geometrical explanation of these combinators, which makes use of the language of boxes and wires. For example, let us consider the identity combinator $\boldsymbol{I}$. Since $\boldsymbol{I}$ has to satisfy the equation $\boldsymbol{I}x = x$, in order to define $\boldsymbol{I}$, it is convenient to regard $\boldsymbol{I}$ as a two-input/two-output function, up-to-coding. The Identity combinator just copies informations from the left-hand input-wire to the right-hand output-wire, and vice versa from the right-hand input-wire to the left-hand output-wire (see Fig. 2(i)). The fact that $\boldsymbol{I}$ satisfies the identity equation has a simple geometrical explanation. Let us apply $\boldsymbol{I}$ to a partial function x (see Fig. 2(ii)). Now *yank* the string connecting the input and the output wires of the result of the application, forgetting about the box corresponding to $\boldsymbol{I}$. This gives us immediately the expected result (see Fig. 2(iii)). Our argument is based on the *Yanking Property* of the trace on the symmetric monoidal category *Pfn* underlying our combinatory algebra. In particular, *Yanking* is one of the axioms characterizing the trace operation in the setting of traced symmetric monoidal categories.

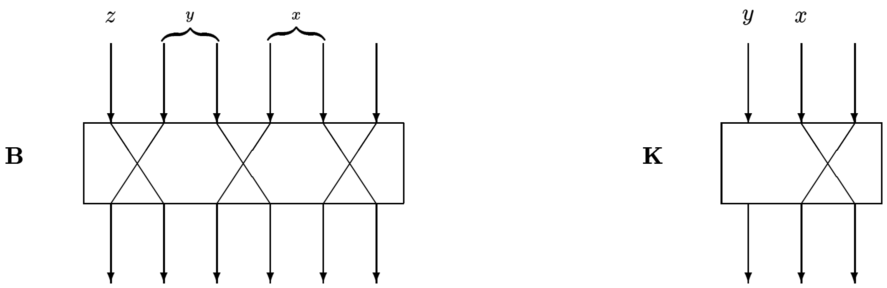
Let us now consider the combinator $\boldsymbol{B}$ which satisfies the equation $\boldsymbol{B}xyz = x(yz)$. Concretely, the box for $\boldsymbol{B}$ has two input (and two output) wires for x and two input (and two output) wires for y, since both x and y are applied to an argument, one input (and one output) wire for z, which appears only as argument, plus one extra input (and one output) wire, along which the input-token (output-token) is intended to enter (exit). The connections of the wires inside the box for $\boldsymbol{B}$ are determined by the control flow between x, y, z in the right-hand part of the equation. First of all, the control flow passes from the input port of $\boldsymbol{B}$ to the input port of x. The second port of x is then connected



(i) (ii) (iii)

Fig. 2. Geometrical description of **I**.

to the input port of y, while the second port of y is connected to the unique port of z. The remaining connections are then obtained by symmetry (see Fig. 3(i)). Using the Yanking Property, one can then check that the result of the application of $\boldsymbol{B}$ to x, y, z is the expected one.

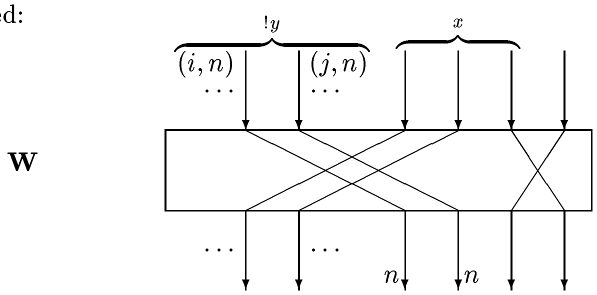


(i) (ii)

Fig. 3. Geometrical representations of **B**, **K**.

Now we briefly discuss the remaining combinators. The combinator $\boldsymbol{C}$ can be explained in a similar way as $\boldsymbol{B}$. The affine combinator $\boldsymbol{K}$ simply forgets about its second argument y (see Fig. 3(ii)).

In order to define $\boldsymbol{W}$, we need to fix two different indices $i, j \in \mathbf{N}$, tagging the copies of y which are used as arguments by x. The remaining copies of y are ignored:



The behavior of $\boldsymbol{D}, \delta, \boldsymbol{F}$ can be explained similarly (see [AL99a]).

Essentially, all the combinators of Definition 8 are functions that mediate the required interactions between the arguments simply by copying information between the various ports.

There are many possible conditions that can be imposed on partial functions in order to cut down the space $[\mathbf{N} \rightharpoonup \mathbf{N}]$, still maintaining closure under the application, !, and all the affine combinators. The subalgebra which gives rise to the fully-complete model of Section 5 is obtained by considering *partial involutions*:

Definition 9. *Let* $f : \boldsymbol{N} \rightharpoonup \boldsymbol{N}$*.* f *is a* partial involution *iff its graph is a symmetric relation. Let us denote by* $[\boldsymbol{N} \rightharpoonup_{\mathsf{Inv}} \boldsymbol{N}]$ *the space of partial involutions from* $\boldsymbol{N}$ *to* $\boldsymbol{N}$*.*

One can check that partial involutions are closed under the application, the !-operation, and all the combinators of Definition 8, i.e.:

Proposition 3. $\mathcal{A}_{\mathsf{PInv}} = ([\boldsymbol{N} \rightharpoonup_{\mathsf{Inv}} \boldsymbol{N}], \bullet, !)$ *is an affine combinatory algebra.*

$\mathcal{A}_{\mathsf{PInv}}$ is a highly constrained algebra, in which all computations are *reversible*. Partial involutions are reminiscent of the *copy-cat* strategies of game categories, in that the only computational effect that they have is that of *copying* informations from input to output wires.

5 A Fully Complete PER Model

In this section, we sketch the proof that the PER category over the LCA $\mathcal{A}_{\mathsf{PInv}}$ of Section 4 satisfies the Axioms of [AL99a], and hence it gives rise to a fully and faithfully complete PER model for ML-types.

The axiomatization of [AL99a] consists of two main steps. The first is an axiomatization of the fact that every morphism $f : 1 \to [\![T]\!]$, where T is an ML-type generates, under *decomposition*, a possibly *infinite* typed Böhm tree. The second step consists of an axiom which rules out infinite trees from the model is introduced. We start by discussing briefly the axioms for the decomposition. First of all, notice that the axiom which expresses the fact that the type $\forall \boldsymbol{X}.X_k$ is empty, and the *Uniformity of Threads* Axiom hold immediately on PER models. In fact, for the first axiom to hold, we need only to verify that the PER $\bigcap_{\boldsymbol{X}} X_k$ is the empty PER. This follows immediately, by instantiating X_k with the empty PER. The Uniformity of Threads Axiom follows from the isomorphism $\bigcap_{\boldsymbol{X}} ! \,\mathcal{R} \multimap !S \simeq \bigcap_{\boldsymbol{X}} ! \,\mathcal{R} \multimap S$, which is an immediate consequence of Lemma 2 of Section 3. The proof of the validity of the remaining axioms for the Decomposition Theorem is based essentially on the nature of partial involutions, and it requires a careful analysis of their applicative behavior. The details of the lengthy proof appear in [AL99a]. The most difficult part of the proof of full completeness for the model $\mathrm{PER}_{\mathcal{A}_{\mathsf{PInv}}}$ consists in proving the *Finiteness Axiom*, i.e. in ruling out infinite typed trees. In particular, we prove that the trees generated by elements of PERs which are denotations of ML-types, via repeated applications of the Decomposition Theorem, have *finite* height. In order to prove this finiteness result, we need to study an intermediate model, which contains also *approximant* terms of possibly infinite trees. This intermediate model consists of the hierarchy of *simple PERs* over the *Sierpinski PER*. This hierarchy gives rise to a model for the simply typed calculus with $\bot, \top$ constants at the base type. First of all, we prove an *Approximation Lemma* (along the lines of [AJM96]), which says that the graph of every partial involution f in a closed polymorphic PER can be viewed as the union of all its approximants. The approximants of f correspond, essentially, to the finite trees obtained by truncating at level k the tree generated from f by applying the Decomposition Theorem. Then, reasoning by contradiction, using the Typed Separability result of Section 1, and the fact that $\bot$ does not live in closed polymorphic PERs, we conclude that only trees with finite height belong to such PERs. The details of the proof appear in [AL99a].

6 Final Remarks and Directions for Future Work

We conclude this paper with a list of remarks and interesting issues which still remain to be addressed (some of them are currently under investigation).

• In this paper, we have presented a fully-complete model for ML-types. A natural question arises: what happens beyond ML-types. Here is a partial answer. Already at the type Nat $\rightarrow$ Nat, where *Nat* is the type of Church's numerals, i.e. $\forall X.(X \rightarrow X) \rightarrow X \rightarrow X$, the PER model of partial involutions is not fully-complete. In fact, not only all recursive functions, but even *all* functions from natural numbers to natural numbers, can be encoded in the type Nat $\rightarrow$ Nat. A similar problem arises even if we consider the term combinatory algebra. PER models as they are defined in this paper, do not seem to give full-completeness beyond ML-types. An innovative construction is called for here.

• Another question which arises naturally is whether the PER model over the linear term combinatory algebra is fully-complete at ML-types. We conjecture that this is the case, but a proof of this fact seems difficult. A logical relation technique relating the term algebra and the term subalgebra of partial involutions could be useful here. The interest of linear term algebras lies in the fact that the PER model generated by these is essentially the PER model shown to be fully-complete at algebraic types in [HRR90].

• We have presented a linear realizability technique for building PER categories over an LCA. These PER categories turn out to be linear categories. It would be interesting to carry on the investigation of the general properties of these categories, e.g. define coproducts, products, etc..

• Models of partial involutions are worthwhile investigating also for typed/untyped λ-calculi different from system F. E.g. strategies in the [AJM96] style, which are represented by partial involutions from Opponent moves to Player moves, should provide fully-complete models for simply typed λ-calculus with $\bot, \top$-base constants. In the untyped setting, partial involutions strategies could possibly provide fully-abstract models, alternative to those in [DFH99,KNO99].

• In the category $\mathrm{PER}_{\mathtt{PInv}}$, models of typed Böhm trees naturally arise (e.g. the model induced by the Sierpinski PER). These are in particular models of the simply typed λ-calculus together with a *fixed point combinator*, as suggested by Alex Simpson. All these "infinite" calculi seem interesting by themselves, but have not yet been property investigated.

References

[Abr96] S.Abramsky. Retracing some paths in Process Algebra, *Concur'96*, 1996.

[Abr97] S.Abramsky. Axioms for Full Abstraction and Full Completeness, 1997, to appear.

[Abr97a] S.Abramsky. Interaction, Combinators, and Complexity, Notes, Siena (Italy), 1997.

[AHPS98] S.Abramsky, E.Haghverdi, P.Panangaden, P.Scott. Geometry of Interaction and Models of Combinatory Logic, 1998, to appear.

[AJ94] S.Abramsky, R.Jagadeesan. New foundations for the Geometry of Interaction, *Inf. and Comp.* **111**(1), 1994, 53–119.

[AJ94a] S.Abramsky, R.Jagadeesan. Games and Full Completeness for Multiplicative Linear Logic, *J. of Symbolic Logic* **59**(2), 1994, 543–574.

[AJM96] S.Abramsky, R.Jagadeesan, P.Malacaria. Full Abstraction for PCF, 1996, Inf. and Comp. to appear.

[AL99] S.Abramsky, J.Longley. Realizability models based on history-free strategies, Draft paper, 1999.

[AL99a] S.Abramsky, M.Lenisa. Fully Complete Models for ML Polymorphic Types, Technical Report **ECS-LFCS-99-414**, LFCS, 1999 (available at http://www.dimi.uniud.it/~lenisa/Papers/Soft-copy-ps/lfcs99.ps.gz).

[AM97] S.Abramsky, G.McCusker. Full abstraction for idealized Algol with passive expressions, *TCS* **227**, 1999, 3–42.

[AM99] S.Abramsky, P.Mellies. *Concurrent Games and Full Completeness*, *LICS'99*.

[AL91] A.Asperti, G.Longo. *Categories, Types and Structures*, Foundations of Computing Series, The MIT Press, 1991.

[BC88] V.Breazu-Tannen, T.Coquand. Extensional models for polymorphism, *TCS* **59**, 1988, 85–114.

[Cro93] R.Crole, *Categories for Types*, Cambridge University Press, 1993.

[DFH99] P.Di Gianantonio, G.Franco, F.Honsell. Game Semantics for Untyped λ-calculus, *TLCA'99*, LNCS, 1999.

[Gir72] J.Y.Girard. Interprétation functionelle et élimunation des coupures de l'arithmètique d'ordre supérieur, Thèse d'Etat, Université Paris VII, 1972.

[Gir89] J.Y.Girard. Towards a Geometry of Interaction, *Contemporary Mathematics* **92**, 1989, 69–108.

[Hug99] D.Hughes. *Hypergame Semantics: Full Completeness for System F*, D.Phil. thesis, University of Oxford, submitted 1999.

[HRR90] J.Hyland, E.Robinson, G.Rosolini. Algebraic types in PER models, *MFPS*, M.Main *et al.* eds, LNCS **442**, 1990, 333–350.

[HO96] M.Hyland, L.Ong. On full abstraction for PCF, *Inf. and Comp.*, 1996, to appear.

[KNO99] A.Ker, H.Nickau, L.Ong. More Universal Game Models of Untyped λ-Calculus: The Böhm Tree Strikes Back, *CSL'99*, LNCS, 1999.

[JSV96] A.Joyal, R.Street, D.Verity. Traced monoidal categories, *Math. Proc. Comb. Phil. Soc.* **119**, 1996, 447–468.

[Lai97] J.Laird. Full abstraction for functional languages with control, *LICS'97*.

[McC96] G.McCusker. Games and full abstraction for FPC, *LICS'96*, 1996.

[Nic94] H.Nickau. Hereditarily sequential functionals, Proc. of the Symposium *Logical Foundations for Computer Science*, LNCS **813**, 1994.

[Sta88] R.Statman. λ-definable functionals and $\beta\eta$-conversion, *Arch. Math. Logik* **23**, 1983, 21–26.