THE BULLETIN OF SYMBOLIC LOGIC
Volume 3, Number 3, Sept. 1997

GAMES AND DEFINABILITY FOR **FPC**

GUY McCUSKER

Abstract. A new games model of the language **FPC**, a type theory with products, sums, function spaces and recursive types, is described. A definability result is proved, showing that every finite element of the model is the interpretation of some term of the language.

§1. Introduction. The work of Lorenzen [24, 23] proposed dialogue games as a foundation for intuitionistic logic. The idea is simple: associated to a formula A is a set of moves for two players, each of which is either an attack on A—an attempt to refute its validity—or a defence. The players, O who wants to refute A and P who wants to prove A, take turns to make moves according to some rules. The rules determine which player has won when play ends, and the formula A is semantically valid if there is a strategy by which P can always win: a ***winning strategy***.

More recently, games of this kind have been applied in computer science to give programming languages a new kind of semantics with a strong intensional flavour. The game interpretations of propositional connectives are transferred to model type constructors via the Curry-Howard isomorphism; and the strategies which interpret proofs become the semantics of programs.

Thanks to the intensional information which they carry, games models have strong completeness properties. For example, in Abramsky and Jagadeesan's model of Multiplicative Linear Logic [2], not only is every proof interpreted as an element of the model, but also every element of the model is the interpretation of a unique cut-free proof. On the programming language side, games models of the functional programming language **PCF** have been constructed which satisfy a similar ***definability*** result [3, 17, 27], and hence provide the first syntax-free constructions of fully abstract models of **PCF** [29]. These results have since been extended to handle languages more complex than **PCF**, including untyped and recursively typed languages [5, 4], languages with control features [21], call-by-value languages [13, 6] and languages with side-effects and store [7]. In a slightly different vein, games models have also been discovered for the constructive classical type theory $\lambda\mu$ [28] and for System F [14].

Received July 11, 1997.
The author acknowledges the support of Oxford University Computing Laboratory.

© 1997, Association for Symbolic Logic
1079-8986/97/0303-0004/$2.60

In this paper, we give an account of a model of **FPC**, a type theory with product, sum, function space and recursive types. This model is a refinement of one first presented in [26], and satisfies a stronger definability property than did that model. When **FPC** is considered as a programming language in its own right, our construction gives rise to a fully abstract model isomorphic to that previously described.

1.1. Modelling the connectives. Consider a formula $A \wedge B$. To attack this, O need only attack either A or B, and P must defend against these attacks. Thus the moves of the game $A \wedge B$ are the moves of A and the moves of B, and the rules specify that to play $A \wedge B$ is to play either A or B, the choice being made by O. Therefore a winning strategy for P must contain winning strategies for both A and B as expected.

Now let us consider $A \rightarrow B$. O can attack this in two ways: either by attacking B, or by defending A. Dually, P can defend either by defending B or by attacking A. Thus the game $A \rightarrow B$ has the moves of A and the moves of B, but those moves of A which previously belonged to O (attacks) now belong to P and *vice versa*.

In the game $A \rightarrow A$, P has a winning strategy: if O attacks the right hand side, P can copy this move as an attack of the left hand side. When O defends against this attack, P can use the same defence against the original move on the right. Thus a pattern of moves is played out as follows

	A	$\rightarrow$	A
O			attack
P	attack		
O	defence		
P			defence
		$\vdots$	

consisting of a play of A on the right, and the same play on the left. However O plays, P can always respond, and therefore wins by playing this "copycat" strategy. For a thorough account of this kind of interpretation of intuitionistic logic, see [9].

1.2. Linear Types. In the example above, a single play of A was carried out on each side of the arrow. This need not be the case in general, because of the logical rules of weakening and contraction. For example, consider the formula $A \wedge (A \rightarrow B) \rightarrow B$. There is a winning copycat strategy, under which P copies moves between the two occurrences of A and between the two occurrences of B, as in Figure 1. Here P has started play in $A \wedge (A \rightarrow B)$ twice: first by copying the move b_1 from the right hand side, and again by copying the move a_1 from one occurrence of A to the other. The fact that the information about how many times a strategy uses its resources is manifest in the model means that games are particularly well-suited to modelling *linear*

$$
\begin{array}{ccccccc}
A & \wedge & (A & \rightarrow & B) & \rightarrow & B \\
 & & & & & & b_1 \\
 & & & & b_1 & & \\
 & & a_1 & & & & \\
a_1 & & & & & & \\
a_2 & & & & & & \\
 & & a_2 & & & & \\
 & & & & b_2 & & \\
 & & & & & & b_2 \\
 & & & & & \vdots &
\end{array}
$$

FIGURE 1. A winning strategy for $A \wedge (A \rightarrow B) \rightarrow B$.

logic [12]. This was first noticed by Blass [8], whose ideas were refined by several others [2, 16, 22], leading to the first definability results.

Games make the distinction between the additive conjunction $A \,\&\, B$ and the multiplicative conjunction $A \otimes B$ clear: in $A \,\&\, B$, a play consists of a single play of either A or B, while in $A \otimes B$, moves can be made on *both* sides. The linear arrow $A \multimap B$ allows A and B to be used once each, while the exponential ! allows reuse, so that a play of $!A$ consists of many plays of A. We will also make use of an interpretation of the additive disjunction $A \oplus B$: O first plays a dummy attack, to which P responds by stating which of A or B he will defend, and play then continues as in A or B accordingly. Given these constructions, we can use games to interpret type theories with the usual intuitionistic connectives $\times$, $\Longrightarrow$ and $+$ via Girard's translation [12].

$$
\begin{array}{rcl}
(A \times B)^\circ & = & A^\circ \,\&\, B^\circ \\
(A \Longrightarrow B)^\circ & = & !A^\circ \multimap B^\circ \\
(A + B)^\circ & = & !A^\circ \oplus !B^\circ
\end{array}
$$

In the remainder of this paper, we give formal definitions of games and strategies, and of the interpretations of the above connectives. We sketch how recursive types are handled, and outline the game interpretation of **FPC**. Finally, a definability result for this model is established.

§2. The language FPC. **FPC** is a typed λ-calculus with product, sum, exponential (function space) and recursive types. The syntax of **FPC** is given by the grammar in Figure 2, where T comes from a countable collection of type variables and x from a countable collection of expression variables. As usual we identify types and expressions up to α-equivalence; the usual notions of free and bound variables, and of substitution, apply. As expected $\mu\mathtt{T}.\tau$ binds

$$
\begin{array}{rcl}
\tau \in \text{Types} & ::= & \mathtt{T} \mid \tau_1 + \tau_2 \mid \tau_1 \times \tau_2 \mid \tau_1 \to \tau_2 \mid \mu\mathtt{T}.\tau. \\
M \in \text{Exp} & ::= & \mathtt{x} \mid \mathtt{inl}_{\tau_1,\tau_2}(M) \mid \mathtt{inr}_{\tau_1,\tau_2}(M) \\
& \mid & \mathtt{case}\ M\ \mathtt{of}\ \mathtt{inl}(\mathtt{x}_1).M_1\ \mathtt{or}\ \mathtt{inr}(\mathtt{x}_2).M_2 \\
& \mid & (M_1, M_2) \mid \mathtt{fst}(M) \mid \mathtt{snd}(M) \\
& \mid & \lambda\mathtt{x} : \tau.M \mid M_1(M_2) \\
& \mid & \mathtt{intro}_{\mu\mathtt{T}.\tau}(M) \mid \mathtt{elim}(M)
\end{array}
$$

FIGURE 2. Syntax of **FPC**.

$\mathtt{T}$ in τ, $\lambda\mathtt{x} : \tau.M$ binds $\mathtt{x}$ in M, and $\mathtt{case}\ M\ \mathtt{of}\ \mathtt{inl}(\mathtt{x}_1).M_1\ \mathtt{or}\ \mathtt{inr}(\mathtt{x}_2).M_2$ binds $\mathtt{x}_1$ in M_1 and $\mathtt{x}_2$ in M_2.

The term formation rules for product and exponential types are standard, so we just give those for the sum and recursive types, which may be less familiar: see Figure 3. In a judgement $\Theta, \Gamma \vdash M : \tau$, Θ is a list of type

Sums

$$
\frac{\Theta, \Gamma \vdash M : \tau \quad \Theta \vdash \tau'}{\Theta, \Gamma \vdash \mathtt{inl}_{\tau,\tau'}(M) : \tau + \tau'} \qquad \frac{\Theta, \Gamma \vdash M : \tau \quad \Theta \vdash \tau'}{\Theta, \Gamma \vdash \mathtt{inr}_{\tau',\tau}(M) : \tau' + \tau}
$$

$$
\frac{\Theta, \Gamma \vdash M : \tau_1 + \tau_2 \quad \Theta, \Gamma, \mathtt{x}_1 : \tau_1 \vdash M_1 : \tau \quad \Theta, \Gamma, \mathtt{x}_2 : \tau_2 \vdash M_2 : \tau}{\Theta, \Gamma \vdash \mathtt{case}\ M\ \mathtt{of}\ \mathtt{inl}(\mathtt{x}_1).M_1\ \mathtt{or}\ \mathtt{inr}(\mathtt{x}_2).M_2 : \tau}
$$

Recursive types

$$
\frac{\Theta, \Gamma \vdash M : \tau[\mathtt{T} \mapsto \mu\mathtt{T}.\tau]}{\Theta, \Gamma \vdash \mathtt{intro}_{\mu\mathtt{T}.\tau}(M) : \mu\mathtt{T}.\tau}
$$

$$
\frac{\Theta, \Gamma \vdash M : \mu\mathtt{T}.\tau}{\Theta, \Gamma \vdash \mathtt{elim}(M) : \tau[\mathtt{T} \mapsto \mu\mathtt{T}.\tau]}
$$

FIGURE 3. Well-formed expressions of **FPC**.

variables and Γ is a list of expression variables together with types. The judgement means that M is a well-formed expression of type τ, with free type variables among Θ and free expression variables among Γ.

The ability to define arbitrary recursive types is very powerful. In particular, it guarantees that recursion is available in the language, because it is possible to code up self-application and hence the fixed point combinator of the untyped λ-calculus

$$\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

can be typed in **FPC**. There is therefore no need to include extra syntax for recursion. This also means that there is a divergent term Ω of every type, given by the fixed point of the identity function; we will use this fact later.

§**3. A games model.** We now review the definitions of the category of games introduced in [25]; proofs that the definitions have the claimed properties can be found there. This category builds on the earlier work of [17], while being strongly influenced by the presentation of [3]. An important distinction between games models of logic and of programming languages is that, since there are perfectly good programs which do not terminate, we are no longer interested solely in winning strategies, but in all strategies. Therefore there will be no further mention of winning or losing games.

3.1. Games. A game has two participants: Player (P) and Opponent (O). A *play* of the game consists of a finite or infinite sequence of moves, alternately by O and P. In addition, each move is explicitly *justified* by an earlier move of the play, unless it is a special kind of move, called *initial*, which needs no justification. In the games we consider, O always moves first.

Before embarking on a formal definition, let us fix notation for sequences and operations on them. We use $s, t, \dots$ to range over sequences and a, $b, \dots$ over elements of sequences. If s and t are sequences, then st is their concatenation; ε is the empty sequence. A move m will often be identified with the singleton sequence consisting just of m. If S is a set, then $s \upharpoonright S$ is the restriction of s to S, i.e., s with all elements not in S deleted.

An ***arena*** is specified by a structure $A = \langle M_A, \lambda_A, \vdash_A \rangle$ where

- M_A is a set of ***moves***.
- $\lambda_A : M_A \to \{\mathsf{O}, \mathsf{P}\} \times \{\mathsf{Q}, \mathsf{A}\}$ is a ***labelling*** function which indicates whether a move is by Opponent (O) or Player (P), and whether it is a question (Q) or an answer (A). We write

$$\begin{aligned} \{\mathsf{O}, \mathsf{P}\} \times \{\mathsf{Q}, \mathsf{A}\} &= \{\mathsf{OQ}, \mathsf{OA}, \mathsf{PQ}, \mathsf{PA}\} \\ \lambda_A &= \langle \lambda_A^{\mathsf{OP}}, \lambda_A^{\mathsf{QA}} \rangle. \end{aligned}$$

  The function $\overline{\lambda}_A$ is λ_A with the O/P part reversed, so that

$$\overline{\lambda}_A(m) = \mathsf{OQ} \iff \lambda_A(m) = \mathsf{PQ}$$

  and so on. If $\lambda^{\mathsf{OP}}(m) = \mathsf{O}$, we call m an O-move; otherwise, m is a P-move.

- $\vdash_A$ is a relation between $M_A + \{\star\}$ and M_A, called ***enabling***, which satisfies
  - $\star \vdash_A m \Longrightarrow \lambda_A(m) = \mathsf{OQ} \wedge$
    $[n \vdash_A m \iff n = \star]$.
  - $m \vdash_A n \wedge \lambda_A^{\mathsf{QA}}(n) = \mathsf{A} \Longrightarrow \lambda_A^{\mathsf{QA}}(m) = \mathsf{Q}$.
  - $m \vdash_A n \wedge m \neq \star \Longrightarrow \lambda_A^{\mathsf{OP}}(m) \neq \lambda_A^{\mathsf{OP}}(n)$.

The enabling relation tells us either that a move m is ***initial*** and needs no justification $(\star \vdash_A m)$, or that it can be justified by another move n, if n has been played $(n \vdash_A m)$. In what follows, we will be concerned with finite sequences s of moves together with justification information: for each non-initial move m of s, there must be a pointer to an earlier move n such that $n \vdash_A m$. Note that the first move of such a sequence must be an initial move, but initial moves may appear after the first move. We call a sequence together with its justification information a ***justified sequence***, and suppress mention of the justification pointers whenever it does not cause confusion.

Given a justified sequence s, define the ***Player view*** $\ulcorner s \urcorner$ of s by

$$\begin{array}{rcll} \ulcorner \varepsilon \urcorner & = & \varepsilon. & \\ \ulcorner sm \urcorner & = & \ulcorner s \urcorner m, & \text{if } m \text{ is a P-move.} \\ \ulcorner sm \urcorner & = & m, & \text{if } \star \vdash m. \\ \ulcorner smtn \urcorner & = & \ulcorner s \urcorner mn, & \text{if } m \text{ justifies } n \text{ and} \\ & & & n \text{ is an O-move.} \end{array}$$

The ***Opponent view*** $\llcorner s \lrcorner$ is defined dually. The idea behind views is that during the play of a game, various different threads and sub-threads of discussion are opened up; the view represents that part of the play so far which is relevant to the current thread. The notions of justification and views should be compared to the restrictions placed upon so-called *E-strategies* in dialogue interpretations of intuitionistic logic [9].

We can now define an important class of justified sequences: the legal positions. A justified sequence s is ***legal*** if and only if

- Players alternate: if $s = s_1 mn s_2$ then $\lambda^{\mathsf{OP}}(m) \neq \lambda^{\mathsf{OP}}(n)$.
- The ***bracketing condition***. We say that a question q in s is ***answered*** by a later answer a in s if q justifies a. The bracketing condition is satisfied by s if for each prefix $tqua$ of s with q answered by a, the last asked, still unanswered question in tqu is q; in other words, when an answer is given, it is always to the most recent question which has not been answered—the ***pending question***.
- The ***visibility condition***. Suppose tm is a prefix of s, and m is not initial. If m is a P-move then the justifier of m appears in $\ulcorner t \urcorner$, and dually if m is an O-move then its justifier appears in $\llcorner t \lrcorner$.

The set of all legal positions of an arena A is written L_A.

Given a justified sequence s, we say that a move m of s is ***hereditarily justified*** by the move n if the chain of justification pointers leading back from m ends at n. Note that n is always an initial move in this case; we shall have no need for a more general notion of hereditary justification. Write $s \upharpoonright n$ for the subsequence of s consisting of all moves hereditarily justified by n, and generalize this to $s \upharpoonright I$ for a set of initial moves I. (This overloading of notation with the previously introduced $s \upharpoonright S$ should not cause any difficulty.)

A ***game*** is a structure $A = \langle M_A, \lambda_A, \vdash_A, P_A \rangle$ where

- $\langle M_A, \lambda_A, \vdash_A \rangle$ is an arena
- P_A is a non-empty, prefix closed set of legal positions, called the set of ***valid positions*** of A, such that if $s \in P_A$ then $s \upharpoonright I \in P_A$ for any set I of initial moves of s.

3.2. Strategies. A strategy for a game A can be thought of as a rule telling Player what move to make in a given position. Formally, this can be represented as a set σ of valid positions in which P has just moved, i.e., a set of even-length positions, such that

- $\varepsilon \in \sigma$
- $sab \in \sigma \Longrightarrow s \in \sigma$
- $sab, sac \in \sigma \Longrightarrow b = c$.

We are interested only in the innocent strategies: those strategies whose responses depend only on the view of the position, rather than on the whole play so far. A strategy σ for A is ***innocent*** if and only if

- $sab \in \sigma \wedge t \in \sigma \wedge ta \in P_A \wedge \ulcorner sa \urcorner = \ulcorner ta \urcorner \Longrightarrow tab \in \sigma$.

An innocent strategy can therefore be seen as a partial function from P-views to P-moves; we call this function the ***view-function*** of σ. For any game A, the smallest possible strategy is $\{\varepsilon\}$, which never makes any response. It is called the ***empty strategy*** and denoted $\perp$.

We write $\sigma : A$ to indicate that σ is an innocent strategy for A.

3.3. Multiplicatives. Given games A and B, the game $A \multimap B$ is defined as follows.

$$
\begin{array}{rcl}
M_{A\multimap B} & = & M_A + M_B \text{ (disjoint union)} \\
\lambda_{A\multimap B} & = & [\overline{\lambda}_A, \lambda_B] \\
\star \vdash_{A\multimap B} m & \Longleftrightarrow & \star \vdash_B m \\
m \vdash_{A\multimap B} n & \Longleftrightarrow & m \vdash_A n \vee m \vdash_B n \vee [\star \vdash_B m \wedge \star \vdash_A n] \\
P_{A\multimap B} & = & \{ s \in L_{A\multimap B} \mid s \upharpoonright M_A \in P_A \wedge s \upharpoonright M_B \in P_B \}.
\end{array}
$$

A consequence of this definition is that O must play the first move in the game B, and that only P can switch between A and B: whenever two consecutive moves are in different components, the second of these moves is by P.

The game $A \otimes B$ is defined similarly. The labelling is different: $\lambda_{A\otimes B} = [\lambda_A, \lambda_B]$. Initial moves of A and B remain initial moves. The unit for tensor is the empty game

$$I = \langle \emptyset, \emptyset, \emptyset, \{\varepsilon\}\rangle.$$

In the game $A \otimes B$, only O can switch components.

3.4. The category $\mathcal{G}$. We can now define a category $\mathcal{G}$ of games and strategies.

$$\begin{array}{rcl} \text{Objects} & : & \text{Games.} \\ \text{Morphisms } \sigma : A \to B & : & \text{Innocent strategies} \\ & & \text{for } A \multimap B. \end{array}$$

Identity. For any game A, the identity map $\mathsf{id} : A \to A$ is given by the 'copycat' strategy

$$\{s \in P_{A_1 \multimap A_2} \mid \forall t \sqsubseteq^{\text{even}} s. t \upharpoonright A_1 = t \upharpoonright A_2\}.$$

Here $t \sqsubseteq^{\text{even}} s$ means that the sequence t is an even-length prefix of s. We use subscripts on the 'A's to distinguish the two occurrences. This strategy responds to any move simply by copying it to the other component. Many of the strategies we will encounter have this sort of behaviour.

Composition. Given maps $\sigma : A \to B$ and $\tau : B \to C$, i.e., innocent strategies $\sigma : A \multimap B$ and $\tau : B \multimap C$, we define an innocent strategy $\sigma ; \tau$ for $A \multimap C$ by

$$\{s \upharpoonright M_A, M_C \mid s \upharpoonright M_A, M_B \in \sigma \wedge s \upharpoonright M_B, M_C \in \tau\}$$

where s ranges over sequences of moves from $M_A + M_B + M_C$. This can be shown to be well-defined and associative.

$\mathcal{G}$ as an autonomous category. Tensor and linear implication extend to functors. For example, given $\sigma : A \to B$ and $\tau : A' \to B'$ we can define $\sigma \otimes \tau : A \otimes A' \to B \otimes B'$ by

$$\{s \in P_{A\otimes A' \multimap B \otimes B'} \mid s \upharpoonright M_A, M_B \in \sigma, s \upharpoonright M_{A'}, M_{B'} \in \tau\}.$$

If $\sigma : A \otimes B \to C$, there is a strategy $\Lambda(\sigma) : A \to B \multimap C$ defined simply by relabelling moves in σ. $\mathcal{G}$ is now a symmetric monoidal closed (autonomous) category.

3.5. Additives. As outlined earlier, $\mathcal{G}$ can also interpret the additive connectives $\&$ and (less successfully) $\oplus$. The interpretation of $\&$ is a genuine categorical product, as expected. However, our interpretation of $\oplus$ does not yield a coproduct, but only a weak coproduct, and only then in a certain subcategory of $\mathcal{G}$. This somewhat unsatisfactory situation is nonetheless good enough to provide a model of **FPC**.

Product. Given games A and B, $A \,\&\, B$ is defined as the 'disjoint sum' of A and B, so that a valid position of $A \,\&\, B$ is either a valid position of A or one of B. Together with the obvious copycat maps $\pi_1 : A \,\&\, B \to A$ and $\pi_2 : A \,\&\, B \to B$, this defines a product in $\mathcal{G}$. The terminal object is I.

Sums. The sum $A \oplus B$ of games A and B has as its set of moves

$$M_{A \oplus B} = M_A + M_B + \{q, l, r\}.$$

The move q is an initial question, asking which side of the sum we are to work in. There are two possible answers, l and r. If l is played, the game continues as a play of A; if r is played, play continues as in B. Initial moves of A are justified by l, and initial moves of B are justified by r. It is for this reason that we allow answers to justify later questions.

There are obvious injection maps

$$\iota_1 : A \to A \oplus B \quad \text{and} \quad \iota_2 : B \to A \oplus B.$$

For example, ι_1 responds to the initial q with l, and then plays copycat between the two copies of A.

Unfortunately, this definition does not give a good interpretation of $\oplus$ in the whole games model. To see the extent to which it *does* work, let us make a further definition. A game A is ***well-opened*** if and only if for every $s \in P_A$, the only initial move of s is the first one. Notice that if both A and B are non-empty, then $A \otimes B$ is not well-opened, so restricting to well-opened games will not give a model of linear logic; but $A \oplus B$ is always well-opened, and $A \multimap B$ is well-opened whenever B is.

The notion of well-openedness allows us to identify the following property of $\oplus$. For any $\sigma : A \to C$ and $\tau : B \to C$, where C is well-opened, there is a strategy $[\sigma, \tau] : A \oplus B \to C$ such that $\iota_1 ;[\sigma, \tau] = \sigma$, $\iota_2 ;[\sigma, \tau] = \tau$. In response to the initial move in C, the strategy $[\sigma, \tau]$ asks the initial question q in $A \oplus B$. If the answer l (resp. r) is given, it carries on playing as σ (resp. τ). Hence $\oplus$ gives a weak coproduct in the subcategory of well-opened games; "weak" because the strategy $[\sigma, \tau]$ is not necessarily unique in satisfying the above equations.

3.6. Exponential. The game $!A$ is defined as a 'repeatable version' of A.

$$\begin{aligned} M_{!A} &= M_A \\ \lambda_{!A} &= \lambda_A \\ \vdash_{!A} &= \vdash_A \\ P_{!A} &= \{s \in L_{!A} \mid \text{for all initial } m,\ s \upharpoonright m \in P_A\}. \end{aligned}$$

A play of $!A$ therefore consists of a number of interleaved plays of A. As for the tensor product, only Opponent is able to switch between the different threads; indeed, Player's view only allows him to see moves from the current thread at any time.

For any game A, there are innocent strategies $\mathsf{weak} : !A \to I$ witnessing ***weakening*** (the empty strategy) and $\mathsf{con} : !A \to !A \otimes !A$ witnessing ***contraction*** (a copycat strategy). Given any $\sigma : !A \to B$ we can define another strategy $\sigma^{\dagger} : !A \to !B$ using the view-function of σ, since P-views in the game $!A \multimap B$ are the same as those in $!A \multimap !B$; this construction witnesses ***promotion***. All this means that we almost have an interpretation of the 'of course' operator of linear logic, but we still need to interpret dereliction. However, in general the copycat strategy for $!A \multimap A$ (which uses only one thread of $!A$) is not well defined, so we have to restrict to a subclass of games for which it is: the well-opened games. This problem can be remedied by moving to a more complicated definition of game (see [25]), but here we choose not to do so, because all the games which are used in modelling **FPC** are in fact well-opened, and because the model of **FPC** remains unchanged.

We will need the following lemma, which follows from the definition of promotion above.

LEMMA 1. *Every innocent strategy for* $!A \multimap !B$ *has the form* $\sigma^{\dagger}$ *for some* $\sigma : !A \multimap B$.

3.7. A cartesian closed category. We can now define a new category, the ***intensional*** category of games $\mathcal{I}$, as follows.

Objects	:	Well-opened games.
Morphisms $\sigma : A \to B$	:	Innocent strategies for $!A \multimap B$.

Notice how this definition of map reflects Girard's translation of intuitionistic logic into linear logic.

Identity.The identity is given by the dereliction map $\mathsf{der} : !A \multimap A$, which is well-defined for well-opened games A.

Composition.Given $\sigma : A \to B$ and $\tau : B \to C$, i.e., strategies $\sigma : !A \multimap B$ and $\tau : !B \multimap C$, we define their composite $\sigma \mathbin{⨟} \tau : A \to C$ by

$$\sigma \mathbin{⨟} \tau = \sigma^{\dagger} ; \tau.$$

It can easily be checked that $!(A \mathbin{\&} B) = !A \otimes !B$, so we have

$$\begin{aligned} \mathcal{I}(A \mathbin{\&} B, C) &= \mathcal{G}(!(A \mathbin{\&} B), C) \\ &= \mathcal{G}(!A \otimes !B, C) \\ &\cong \mathcal{G}(!A, !B \multimap C) \\ &= \mathcal{I}(A, !B \multimap C) \end{aligned}$$

so setting $A \Rightarrow B = !A \multimap B$, we see that $\mathcal{I}$ is cartesian closed.

3.8. Recursive types. Games admit a treatment of recursive types very similar to that of information systems [30]. We define an ordering $\trianglelefteq$ on games as follows. For any games A and B, $A \trianglelefteq B$ iff

$$\begin{array}{rcl} M_A & \subseteq & M_B \\ \lambda_A & = & \lambda_A \restriction M_A \\ \vdash_A & = & \vdash_B \cap (M_A + \{\star\} \times M_A) \\ P_A & = & P_B \cap M_A^*. \end{array}$$

This is a (large) dcpo, with least element I and directed sups given by taking the union of each component, just as for information systems. Given a type constructor (i.e., an operation on games) F which is continuous with respect to $\trianglelefteq$, we can find a game $D = F(D)$ in the usual way by setting $D = \bigsqcup_{n=0}^{\infty} F^n(I)$.

In fact this can be generalised and strengthened to obtain *parameterized minimal invariants* [11, 10] for a large class of functors $F : (\mathcal{I}^{\mathrm{op}} \times \mathcal{I})^n \to \mathcal{I}$, including all the functors built out of the product, sum and function space constructions described here.

3.9. A model of FPC. We have seen that $\mathcal{I}$ is a cartesian closed category, and that sums can be modelled in $\mathcal{I}$ using $\oplus$ and Girard's translation:

$$A + B = {!A} \oplus {!B}.$$

Given any maps $\sigma : A \to C$ and $\tau : B \to C$ (that is, strategies for $!A \multimap C$ and $!B \multimap C$, with C well-opened), we can construct a map $[\sigma, \tau] : A + B \to C$ using the constructions of $\mathcal{G}$:

$$[\sigma, \tau]_{\mathcal{I}} = \mathsf{der}\,;[\sigma, \tau]_{\mathcal{G}}$$

using subscripts to distinguish the category in which the $[-, -]$ operation is working. Again this is only a weak coproduct; indeed, $\mathcal{I}$ is a cartesian closed category with fixed points of all endomorphisms, so it is impossible for genuine coproducts to exist [15].

Recursive types can also be modelled, as outlined above. We therefore have everything we need to interpret **FPC**. The type constructors $\times$, $+$ and $\to$ are modelled using their counterparts in the category, and a recursive type $\mu\mathtt{T}.\tau$ is interpreted as the canonical solution of the "domain equation"

$$D = \tau[D/\mathtt{T}].$$

Any closed type τ is therefore interpreted as an object $[\![\tau]\!]$.

The interpretation of terms is standard; we only give details in the cases of sums and recursive types. For recursive types, note that because we are solving domain equations up to identity, we can interpret a term $\mathtt{intro}(M)$ with exactly the same map as is used to interpret M, and similarly $[\![\mathtt{elim}(M')]\!] =$

$[\![M']\!]$. For sums, one more piece of structure is needed, namely a map witnessing the distribution of product over sum. For any games A, B and C there is a map $\mathsf{dist}_{A,B,C} : A \times (B + C) \to (A \times B) + (A \times C)$ given by the obvious copycat strategy, such that the family dist forms a natural transformation satisfying $\mathsf{id} \times \iota_1 \mathbin{;} \mathsf{dist} = \iota_1$, $\mathsf{id} \times \iota_2 \mathbin{;} \mathsf{dist} = \iota_2$ and $\mathsf{id} \times \bot \mathbin{;} \mathsf{dist} = \bot$. With this in place we can interpret terms of sum type as shown in Figure 4.

Sums

$$\frac{[\![\Gamma \vdash M : \tau]\!] = m : [\![\Gamma]\!] \to [\![\tau]\!]}{[\![\Gamma \vdash \mathtt{inl}_{\tau,\tau'}(M)]\!] = m \mathbin{;} \iota_1 : [\![\Gamma]\!] \to [\![\tau]\!] + [\![\tau']\!]}$$

$$\frac{[\![\Gamma \vdash M : \tau]\!] = m : [\![\Gamma]\!] \to [\![\tau]\!]}{[\![\Gamma \vdash \mathtt{inr}_{\tau',\tau}(M)]\!] = m \mathbin{;} \iota_2 : [\![\Gamma]\!] \to [\![\tau']\!] + [\![\tau]\!]}$$

$$\frac{\begin{array}{rcl} [\![\Gamma \vdash M : \tau_1 + \tau_2]\!] & = & m : [\![\Gamma]\!] \to [\![\tau_1]\!] + [\![\tau_2]\!] \\ [\![\Gamma, \mathtt{x}_1 : \tau_1 \vdash M_1 : \tau]\!] & = & m_1 : [\![\Gamma]\!] \times [\![\tau_1]\!] \to [\![\tau]\!] \\ [\![\Gamma, \mathtt{x}_2 : \tau_2 \vdash M_2 : \tau]\!] & = & m_2 : [\![\Gamma]\!] \times [\![\tau_2]\!] \to [\![\tau]\!] \end{array}}{\begin{array}{l} [\![\Gamma \vdash \mathtt{case}\ M\ \mathtt{of}\ \mathtt{inl}(\mathtt{x}_1).M_1\ \mathtt{or}\ \mathtt{inr}(\mathtt{x}_2).M_2 : \tau]\!] \\ \qquad = \langle \mathsf{id}_{[\![\Gamma]\!]}, m \rangle \mathbin{;} \mathsf{dist} \mathbin{;} [m_1, m_2] : [\![\Gamma]\!] \to [\![\tau]\!] \end{array}}$$

FIGURE 4. Denotational semantics of sums in **FPC**.

Remark. The only difference between this model of **FPC** and that of [25] is the interpretation of sum. Previously, the linear sum $A \oplus B$ was used, rather than Girard's translation $!A \oplus !B$. This gave rise to certain complications which meant that **FPC** could not be modelled in $\mathcal{I}$ at all but only in its extensional quotient $\mathcal{E}$. It is also worth remarking that using Girard's translation renders the set P_A of valid positions in the definition of a game slightly redundant, since for all **FPC** types, the valid positions are precisely the legal positions with at most one initial move.

§**4. Definability.** Having set up our model of **FPC**, we now demonstrate that it satisfies a certain definability property. The proof follows the lines pioneered in [3, 17] and axiomatized in [1], but generalized slightly to handle the sum types. This is in contrast to the previous **FPC**-definability result, which was considerably more complicated to establish.

For any games A and B, the maps $\sigma : A \to B$ in $\mathcal{I}$ form an algebraic cpo under subset inclusion; a strategy is compact iff it has a finite view-function. The result we want is that *every compact strategy at **FPC** type is the denotation of some **FPC** term.* For simplicity, we shall work only with finite types, generated by the grammar

$$A ::= I \mid A + A \mid A \,\&\, A \mid A \Rightarrow A.$$

Each such type is isomorphic to one in basic form, that is, of the form $\prod_i (A_i \Rightarrow B_i + C_i)$ for some A_i, B_i and C_i. In [25], it is shown that a definability result for types of this kind implies the general result, so we aim to prove

THEOREM 2. *Let $T = \prod_i (A_i \Rightarrow B_i + C_i)$, D and E be games in basic form. Every finite strategy $\sigma : T \to D + E$ is definable by some term of **FPC**.*

To do this, we analyse the finite strategies

$$\sigma : \prod_i (A_i \Rightarrow B_i + C_i) \to D + E$$

in detail. The idea is to decompose σ into sub-strategies which completely determine its behaviour, but are strictly smaller than σ, to facilitate an inductive proof of definability. For these purposes, the linear type structure of the underlying category $\mathcal{G}$ is very useful.

In fact we prove this result simultaneously for all suitable types T, D and E by induction on the size of the view-function of σ. For each T, D, E and σ, we must find a term $\Gamma \vdash M : D + E$ where $\Gamma = \mathtt{x}_1 : A_1 \to B_1 + C_1, \ldots, \mathtt{x}_n : A_n \to B_n + C_n$ such that $\sigma = [\![\Gamma \vdash M]\!]$. The fact that recursive types have canonical solutions means, among other things, that the fixed point operator described earlier delivers least fixed points. Thus the base case of our induction is established: the empty strategy is defined by the term Ω of the correct type.

For the inductive step, there are three cases according to σ's response to the initial question q, which we call $\sigma(q)$.

- $\sigma(q) = l$. Then the set $\{s \mid qls \in \sigma\}$ forms an innocent strategy $\sigma' : !T \multimap !D$, which by Lemma 1 can be written as $\tau^\dagger$ for some τ. Hence $\sigma = \tau \mathbin{;} \imath_1$. The type $T \Longrightarrow D$ can be manipulated into basic form, so by the inductive hypothesis, $\tau = [\![M]\!]$ for some term M, and then σ is definable by $[\![\mathtt{inl}(M)]\!]$.
- $\sigma(q) = r$. Similar to the above, $\sigma = [\![\mathtt{inr}(M)]\!]$ for some term M.
- $\sigma(q) = q'$, the initial question in one thread of some $A \Rightarrow B + C$ on the left hand side. We will denote the corresponding answers by l' and r'. By manipulating the types, we can separate the thread of $A \Rightarrow B + C$ which σ has opened out into a separate tensor component, arriving at a strategy
$$\sigma' : !T \otimes (A \Rightarrow B + C) \multimap D + E.$$
The P-views of the positions of this strategy fall into two classes: those that contain an answer to q', and those that do not. The definition of view means that those which contain an answer to q' do not contain

any moves of the game A. We use this classification to decompose the strategy σ into smaller substrategies as follows.

The P-views in which q' is not answered have the form $qq's$, for some sequence of moves s. By the bracketing condition, q cannot be answered in s either, so s is a sequence of moves just in $!T$ and $!A$. The set of all such sequences gives rise to an innocent strategy for $!T \multimap !A$, after relabelling the moves, and hence to a strategy

$$\mathsf{arg}(\sigma) : !T \multimap A$$

by Lemma 1. This is the first of the substrategies we need.

The remaining views have the form $qq'l't$ or $qq'r't$, containing no moves from the game A. These views therefore give rise to a strategy for $!T \otimes (!B \oplus !C) \multimap D + E$, and hence to smaller strategies

$$\begin{array}{rl} \mathsf{cont_L}(\sigma) : & !T \otimes !B \multimap D + E \\ \mathsf{cont_R}(\sigma) : & !T \otimes !C \multimap D + E. \end{array}$$

The three substrategies $\mathsf{arg}(\sigma)$, $\mathsf{cont_L}(\sigma)$ and $\mathsf{cont_R}(\sigma)$ are together enough to determine the whole behaviour of σ. Furthermore, they have smaller view-functions than σ, so by the inductive hypothesis they are definable by terms $\Gamma \vdash M : A$, $\Gamma, \mathtt{y} : B \vdash M_L$ and $\Gamma, \mathtt{z} : C \vdash M_R$. These terms can now be put together to create a term defining σ: if $\mathtt{x}$ is the variable corresponding to the component $A \Longrightarrow B + C$ of T, then we have

$$\sigma = [\![\mathtt{case}\ \mathtt{x}M\ \mathtt{of}\ \mathtt{inl}(\mathtt{y}).M_L\ \mathtt{or}\ \mathtt{inr}(\mathtt{z}).M_R]\!]$$

completing the proof.

From definability to full abstraction. In previous work [25], **FPC** was endowed with an operational semantics and treated as a programming language in its own right. It was shown that the games model respected the operational semantics (computational adequacy), and the weaker definability result proved in *loc. cit.* was used to establish a *full abstraction* theorem, stating that observable equivalence of terms in the language coincides with equality in the model. All of this work was carried out in a certain quotient of the category $\mathcal{I}$, called the ***extensional category*** $\mathcal{E}$. It is possible, and straightforward, to carry out the same programme for the model presented here. The main differences are that previously sums (and hence the language **FPC**) were not modelled in $\mathcal{I}$ at all, but only in $\mathcal{E}$, and that the previous definability result was weaker as a consequence. However, the new model coincides with the old one up to extensional equivalence; that is to say, the models in $\mathcal{E}$ are identical.

REFERENCES

[1] S. Abramsky, *Axioms for full abstraction and full completeness*, Unpublished manuscript, 1996.

[2] S. Abramsky and R. Jagadeesan, *Games and full completeness for multiplicative linear logic*, ***Journal of Symbolic Logic***, vol. 59 (1994), no. 2, pp. 543 – 574, Also appeared as Technical Report 92/24 of the Department of Computing, Imperial College of Science, Technology and Medicine.

[3] S. Abramsky, R. Jagadeesan, and P. Malacaria, *Full abstraction for PCF*, Submitted for publication, 1996.

[4] S. Abramsky and G. McCusker, *Games and full abstraction for the lazy λ-calculus*, In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science* [18], pp. 234–243.

[5] ———, *Games for recursive types*, ***Theory and formal methods of computing 1994: Proceedings of the Second Imperial College Department of Computing Workshop on Theory and Formal Methods*** (Chris L. Hankin, Ian C. Mackie, and Rajagopal Nagarajan, editors), Imperial College Press, October 1995.

[6] ———, *Call-by-value games*, Submitted for publication, 1997.

[7] ———, *Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions*, ***Algol-like languages*** (Peter W. O'Hearn and Robert D. Tennent, editors), Birkhaüser, 1997, pp. 297–329 of volume 2.

[8] A. Blass, *A game semantics for linear logic*, ***Annals of Pure and Applied Logic***, vol. 56 (1992).

[9] W. Felscher, *Dialogues as a foundation for intuitionistic logic*, ***Handbook of philosophical logic, volume iii*** (D. Gabbay and F. Guenther, editors), D. Reidel Publishing Company, 1986, pp. 341–372.

[10] M. P. Fiore, ***Axiomatic domain theory in categories of partial maps***, Distinguished Dissertations in Computer Science, Cambridge University Press, 1996.

[11] P. J. Freyd, *Algebraically complete categories*, ***Proc. 1990 como category theory conference*** (Berlin) (A. Carboni et al., editors), Springer-Verlag, 1991, Lecture Notes in Mathematics Vol. 1488, pp. 95–104.

[12] J-Y. Girard, *Linear Logic*, ***Theoretical Computer Science***, vol. 50 (1987), no. 1, pp. 1–102.

[13] K. Honda and N. Yoshida, *Game theoretic analysis of call-by-value computation*, Submitted for publication, 1996.

[14] D. Hughes, *Games and definability for System F*, In *Proceedings, Twelfth Annual IEEE Symposium on Logic in Computer Science* [20], pp. 76–86.

[15] H. Huwig and A. Poigné, *A note on inconsistencies caused by fixpoints in a cartesian closed category*, ***Theoretical Computer Science***, vol. 73 (1990), pp. 101–112.

[16] J. M. E. Hyland and C.-H. L. Ong, *Fair games and full completeness for Multiplicative Linear Logic without the Mix-Rule*, available from `ftp://ftp.comlab.ox.ac.uk/pub/Documents/techpapers/Luke.Ong`, 1993.

[17] ———, *On full abstraction for PCF: I, II and III*, 130 pages, ftp-able at `theory.doc.ic.ac.uk`directory `papers/Ong`, 1994.

[18] IEEE Computer Society Press, *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science*, 1995.

[19] IEEE Computer Society Press, *Proceedings, Eleventh Annual IEEE Symposium on Logic in Computer Science*, 1996.

[20] IEEE Computer Society Press, *Proceedings, Twelfth Annual IEEE Symposium on Logic in Computer Science*, 1997.

[21] J. Laird, *Full abstraction for functional languages with control*, In *Proceedings, Twelfth*

Annual IEEE Symposium on Logic in Computer Science [20], pp. 58–67.

[22] F. Lamarche, *Games semantics for full propositional linear logic*, In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science* [18], pp. 464–473.

[23] P. Lorenzen, *Logik und Agon*, ***Atti del congresso internazionale di filosofia*** (Firenze), Sansoni, 1960, pp. 187–194.

[24] ———, *Ein dialogisches Konstruktivitätskriterium*, ***Infinitistic methods*** (Warszawa), PWN, 1961, Proceed. Symp. Foundations of Math., pp. 193–200.

[25] G. McCusker, *Games and full abstraction for a functional metalanguage with recursive types*, ***Ph.D. thesis***, Department of Computing, Imperial College, University of London, 1996.

[26] ———, *Games and full abstraction for FPC*, In *Proceedings, Eleventh Annual IEEE Symposium on Logic in Computer Science* [19], pp. 174–183.

[27] H. Nickau, *Hereditarily sequential functionals*, ***Proceedings of the symposium on logical foundations of computer science: Logic at st. petersburg***, Lecture notes in Computer Science, Springer, 1994.

[28] C. H. L. Ong, *A semantic view of classical proofs: type-theoretic, categorical and denotational characterizations*, In *Proceedings, Eleventh Annual IEEE Symposium on Logic in Computer Science* [19], pp. 230–241.

[29] G. Plotkin, *LCF considered as a programming language*, ***Theoretical Computer Science***, vol. 5 (1977), pp. 223–255.

[30] G. Winskel, ***The formal semantics of programming languages***, Foundations of Computing, The MIT Press, Cambridge, Massachusetts, 1993.

ST JOHN'S COLLEGE
OXFORD, UNITED KINGDOM OX1 3JP

E-mail: mccusker@comlab.ox.ac.uk