Linearity, Sharing and State: a fully abstract game semantics for Idealized Algol with active expressions

Samson Abramsky Guy McCusker

Abstract

The manipulation of objects with state which changes over time is all-pervasive in computing. Perhaps the simplest example of such objects are the program variables of classical imperative languages. An important strand of work within the study of such languages, pioneered by John Reynolds, focusses on "Idealized Algol", an elegant synthesis of imperative and functional features.

We present a novel semantics for Idealized Algol using games, which is quite unlike traditional denotational models of state. The model takes into account the irreversibility of changes in state, and makes explicit the difference between *copying* and *sharing* of entities. As a formal measure of the accuracy of our model, we obtain a full abstraction theorem for Idealized Algol with active expressions.

1 Introduction

Our starting point is the elegant synthesis of functional and imperative programming due to John Reynolds [28]. His approach is to take an applied simply-typed λ-calculus in which, by suitable choices of the base types and constants, imperative features can be represented. The resulting language is referred to as *Idealized Algol* (IA for short).

The language is in fact parametrized by the choice of basic data types and operations on them. In Reynolds' approach, a sharp distinction is drawn between these basic data types—which can be regarded as discrete sets—and the phrase types of the programming language. For each basic data type = set X, there will be two basic phrase types of the language:

- $\mathtt{exp}[X]$—the type of expressions which yield values in the set X.
- $\mathtt{var}[X]$—the type of (assignable) program variables which can contain values from the set X.

In addition, there is a basic type $\mathtt{com}$ of commands. Reynolds makes a sharp distinction between expressions, which he does not allow to have side-effects, and commands, which are executed precisely to have side-effects. We shall relax this distinction, and can consequently identify $\mathtt{com}$ with $\mathtt{exp}[\mathbf{1}]$ where $\mathbf{1}$ is a one-point set—the "unit" type.

The syntax of basic types is then

$$B \ ::= \ \mathtt{exp}[X] \mid \mathtt{var}[X] \mid \mathtt{com}.$$

The general types are defined by

$$T \ ::= \ B \mid T \Rightarrow T.$$

The constants of the language will be as follows:

- Recursion combinators $\mathbf{Y}_T : (T \Rightarrow T) \Rightarrow T$ for each type T.
- Conditionals
$$\mathtt{cond}_B : \mathtt{exp}[\mathbf{B}] \Rightarrow B \Rightarrow B \Rightarrow B$$
for each base type B.
- For each operation
$$f : X_1 \times \cdots \times X_k \longrightarrow X$$
on the basic data types, a constant
$$\tilde{f} : \mathtt{exp}[X_1] \Rightarrow \cdots \mathtt{exp}[X_k] \Rightarrow \mathtt{exp}[X].$$
- Command sequencing:
$$\mathtt{seq}_X : \mathtt{com} \Rightarrow \mathtt{exp}[X] \Rightarrow \mathtt{exp}[X]$$
for each set X. Taking $X = \mathbf{1}$ this yields
$$\mathtt{seq} : \mathtt{com} \Rightarrow \mathtt{com} \Rightarrow \mathtt{com}$$
as a special case. Also,
$$\mathtt{skip} : \mathtt{com}.$$
- Variable dereferencing and assignment:
$$\mathtt{deref}_X : \mathtt{var}[X] \Rightarrow \mathtt{exp}[X]$$
$$\mathtt{assign}_X : \mathtt{var}[X] \Rightarrow \mathtt{exp}[X] \Rightarrow \mathtt{com}.$$

- Block structure:

$$\mathtt{new}_{X,Y} : (\mathtt{var}[X] \Rightarrow \mathtt{exp}[Y]) \Rightarrow \mathtt{exp}[Y].$$

  Again, taking $Y = \mathbf{1}$ yields

$$\mathtt{new}_X : (\mathtt{var}[X] \Rightarrow \mathtt{com}) \Rightarrow \mathtt{com}$$

  as a special case.

The first three items mean that we can regard Idealized Algol as an *extension* of PCF [25], identifying $\mathtt{exp}[\mathbf{N}]$, $\mathtt{exp}[\mathbf{B}]$ with the basic types **nat**, **bool** of PCF. Reynolds' version of the language, in which expressions cannot have side-effects, is obtained by removing all constants $\mathtt{seq}_X$, $\mathtt{new}_{X,Y}$ except for $\mathtt{seq}$, $\mathtt{new}_X$.

With these constants, we can encode the usual imperative constructs.

Examples

1. $x := x + 1$ is represented by

$$x : \mathtt{var}[\mathbf{N}] \vdash \mathtt{assign}\ x\ (\mathtt{succ}\ (\mathtt{deref}\ x)) : \mathtt{com}.$$

2. $\mathtt{while}\ \neg(x = 0)\ \mathtt{do}\ x := x - 1$ is represented by

$$\begin{aligned} x : \mathtt{var}[\mathbf{N}] \vdash \mathbf{Y}_{\mathtt{com}}(\lambda c : \mathtt{com}.\ & \mathtt{cond}\ (\mathtt{iszero}\ (\mathtt{deref}\ x)) \\ & \mathtt{skip} \\ & (\mathtt{seq}\ (\mathtt{assign}\ x\ (\mathtt{pred}(\mathtt{deref}\ x)))\ c)) : \mathtt{com}. \end{aligned}$$

3. The code
$\mathtt{new}\ x\ \mathtt{in}$
$(x := 0;$
$p(x := x + 1);$
$\mathtt{if}\ x = 0\ \mathtt{then}\ \Omega\ \mathtt{else}\ \mathtt{skip}$
$)$
   translates into

$$\begin{aligned} p : \mathtt{com} \Rightarrow \mathtt{com} \vdash\ & \mathtt{new}(\lambda x : \mathtt{var}[\mathbf{N}]. \\ & \mathtt{seq}\ (\mathtt{assign}\ x\ 0) \\ & \quad (\mathtt{seq}\ (p(\mathtt{assign}\ x\ (\mathtt{succ}(\mathtt{deref}\ x)))) \\ & \quad\quad (\mathtt{cond}\ (\mathtt{iszero}\ (\mathtt{deref}\ x))\ \Omega\ \mathtt{skip}))) : \mathtt{com}. \end{aligned}$$

It is worth noting how the translation into Idealized Algol forces us to be precise about the types of the variables, and whether they occur free or bound.

Idealized Algol is surprisingly expressive. For example, scoped arrays with dynamically computed bounds can be introduced by definitional extension [33], as can classes, objects and methods [27]. It can thus serve as a prototypical language combining state and block structure with higher-order functional features in the same way that PCF has been studied as a prototypical functional language.

2 Models for Idealized Algol

What should a model for Idealized Algol look like? Since the language is an applied simply typed λ-calculus, we should expect to model it in a cartesian-closed category $\mathcal{C}$ [9]. To accommodate the recursion in the language, we can ask for $\mathcal{C}$ to be cpo-enriched [5], or more minimally, to be *rational* [2]. This says that $\mathcal{C}$ is enriched over pointed posets, and that the least upper bounds of chains $\{f^k \circ - \mid k \in \omega\}$ of iterated endomorphisms $f : A \to A$ exist in a suitably enriched and parametrized sense (see [2] for details).

As regards the basic data types, following [1] we can ask that the functor

$$\mathcal{C}(1, -) : \mathcal{C}_0 \longrightarrow \mathbf{Set}_\star$$

have a left adjoint left inverse $\widetilde{(\cdot)}$, where $\mathcal{C}_0$ is a suitable sub-category of $\mathcal{C}$, and $\mathbf{Set}_\star$ is the category of pointed sets. (If $\mathcal{C}$ is **Cpo**, then $\mathcal{C}_0$ will be the sub-category of strict maps, and $\widetilde{X_\star}$ is the flat domain $X_\perp$. If $\mathcal{C}$ is a category of games, then $\mathcal{C}_0$ will be the sub-category of total morphisms on π-atomic objects, and $\widetilde{X_\star}$ will be the flat game over the set X; see [1].)

Unpacking this definition, it says that for each object A of $\mathcal{C}_0$ and pointed set $X_\star$ (where X is the set of elements of $X_\star$ excluding the basepoint), there is a bijection

$$\mathcal{C}_0(\widetilde{X_\star}, A) \cong \mathbf{Set}_\star(X_\star, \mathcal{C}(1, A))$$

natural in $X_\star$ and A, and such that the unit

$$\eta_{X_\star} : X_\star \longrightarrow \mathcal{C}(1, \widetilde{X_\star})$$

is a bijection. This means that each $x \in X$ corresponds to a unique morphism $\bar{x} : 1 \to \widetilde{X_\star}$ in $\mathcal{C}$, and for each family

$$(f_x : 1 \longrightarrow A \mid x \in X)$$

there is a unique $\mathcal{C}_0$-morphism

$$[f_x \mid x \in X] : \widetilde{X_\star} \longrightarrow A$$

such that

$$\overline{x_0}; [f_x \mid x \in X] = f_{x_0}$$

for all $x_0 \in X$. As a special case, each $f : X_\star \to Y_\star$ in $\mathbf{Set}_\star$ (i.e. each partial function $f : X \rightharpoonup Y$) is represented by

$$[g_x \mid x \in X] : \widetilde{X_\star} \longrightarrow \widetilde{Y_\star}$$

where

$$g_x = \begin{cases} \overline{f(x)}, & f(x) \text{ defined} \\ -_{\widetilde{Y_\star}}, & \text{otherwise.} \end{cases}$$

Thus this adjunction gives us a "constant" or "numeral" for each of the elements of a datatype, and a `case` construct to branch on the values of a datatype, with (partial) functions between datatypes represented as special cases.

This data is used in modelling the basic types and constants of Idealized Algol as follows:

- $\mathtt{exp}[X]$ is interpreted by $\widetilde{X_\star}$, and $\mathtt{com} = \mathtt{exp}[1]$ by $\widetilde{1_\star}$.
- The conditionals and basic operations on data types are interpreted using the `case` construct. For example, the constant

  $$\mathtt{if0} : \mathtt{exp}[\mathbf{N}] \Rightarrow \mathtt{exp}[\mathbf{N}] \Rightarrow \mathtt{exp}[\mathbf{N}] \Rightarrow \mathtt{exp}[\mathbf{N}]$$

  is represented by $[g_n \mid n \in \mathbf{N}]$, where

  $$g_0 = [\![\lambda x, y : \mathtt{exp}[\mathbf{N}].\ x]\!]$$

  $$g_{n+1} = [\![\lambda x, y : \mathtt{exp}[\mathbf{N}].\ y]\!].$$

- Skip and sequencing are represented as the degenerate cases of constants and case constructs for the underlying data type $X = 1 = \{\bullet\}$. That is,

  $$[\![\mathtt{skip}]\!] = \top \stackrel{\mathrm{df}}{=} \bar{\bullet} : 1 \longrightarrow \widetilde{1_\star}$$

  $$[\![\mathtt{seq}_A]\!] = [g_\bullet \mid \bullet \in \{\bullet\}] : \widetilde{1_\star} \Rightarrow A \Rightarrow A$$

  where $g_\bullet = [\![\lambda x : A.\ x]\!]$.

  To understand this modelling of sequencing, think of $\widetilde{1_\star}$ as a type with a single defined value (the denotation $\top$ of `skip`); this value being returned when a command is invoked corresponds to the successful termination of the command. Now just as a conditional firstly evaluates the test, and then selects one of the branches for execution, so `seq` as a "unary case statement" will firstly evaluate the command, and then, provided the command terminates successfully, proceed with the continuation.

What about variables? Here we can make use of another important idea due to Reynolds [28]. If we analyze the "methods" we use to manipulate variables, in the spirit of object-oriented programming, we see that there are just two:

- the reading (or dereferencing) method: $\mathtt{deref}_X : \mathtt{var}[X] \Rightarrow \mathtt{exp}[X]$.
- the writing (or assignment) method: $\mathtt{assign}_X : \mathtt{var}[X] \Rightarrow \mathtt{exp}[X] \Rightarrow \mathtt{com}$.

We can *identify* $\mathtt{var}[X]$ with the product of its methods:

$$\mathtt{var}[X] = \mathtt{acc}[X] \times \mathtt{exp}[X]$$

where $\mathtt{acc}[X]$ is the type of "acceptors" or write capabilities. It is then tempting to take a further step (which Reynolds has often, but not always, taken [28];

cf. also [33]), and identify acceptors with the type indicated by the assignment operation:

$$\mathtt{acc}[X] = \mathtt{exp}[X] \Rightarrow \mathtt{com}.$$

However, we do not wish to do this, for the following reason. Idealized Algol is a call-by-name language, but it is intended that only *values* of basic data types, not "closures" or "thunks", be stored in program variables. Thus we prefer to define

$$\mathtt{acc}[X] = \mathtt{com}^X,$$

the product of X copies of $\mathtt{com}$. We can then define a *retraction*

$$c : \mathtt{acc}[X] \lhd (\mathtt{exp}[X] \Rightarrow \mathtt{com}) : r$$

where $c : \mathtt{acc}[X] \to (\mathtt{exp}[X] \Rightarrow \mathtt{com})$ is defined by exponential transposition from

$$\hat{c} : \widetilde{X_\star} \longrightarrow (\mathtt{com}^X \Rightarrow \mathtt{com})$$

$$\hat{c} = [\pi_x \mid x \in X],$$

and

$$r = \langle r_x \mid x \in X \rangle : (\mathtt{exp}[X] \Rightarrow \mathtt{com}) \longrightarrow \mathtt{acc}[X]$$

where $r_x : (\mathtt{exp}[X] \Rightarrow \mathtt{com}) \to \mathtt{com}$ is defined by:

$$\mathtt{exp}[X] \Rightarrow \mathtt{com} \cong (\widetilde{X_\star} \Rightarrow \mathtt{com}) \times 1 \xrightarrow{\mathtt{id} \times \bar{x}} (\widetilde{X_\star} \Rightarrow \mathtt{com}) \times \widetilde{X_\star} \xrightarrow{\mathtt{Ap}} \mathtt{com}.$$

If these definitions are worked out in the case of **Cpo**, it will be seen that r is "strictification", while c is just inclusion of strict functions, as expected.

Given these definitions, we can now define

$$[\![\mathtt{deref}]\!] = \mathtt{snd}$$

$$[\![\mathtt{assign}]\!] = c \circ \mathtt{fst}.$$

Since we have omitted products from our version of Idealized Algol for notational simplicity, it also makes sense to define a variable constructor

$$\mathtt{mkvar}_X : (\mathtt{exp}[X] \Rightarrow \mathtt{com}) \Rightarrow \mathtt{exp}[X] \Rightarrow \mathtt{var}[X]$$

by

$$[\![\mathtt{mkvar}_X]\!] f e = \langle r \circ f, e \rangle.$$

3 The Functional/Imperative Boundary

At this point, the reader should be experiencing a sense of vertigo, or at least puzzlement. We have provided a notion of model for Idealized Algol which is only the mildest extension of the usual notion of model for PCF, and yet which appears to account for all the imperative features of the language, without introducing states or any other device for explicitly modelling assignable variables! What is going on?

The answer is indeed a very interesting consequence of Reynolds' analysis of imperative languages, although it is one which, as far as we are aware, he has not himself explicitly drawn. Firstly, note that a more precise statement is that the notion of model we have developed to this point accounts for everything in Idealized Algol *except* the `new` constants, i.e. block structure. We refer to the sub-language obtained by omitting the `new` constants as $\mathsf{IA} - \{\texttt{new}\}$. We can now formulate the following thesis:

$\boxed{\mathsf{IA} - \{\texttt{new}\} \text{ is a pure functional language.}}$

At first sight, this seems nonsensical, since the usual "basic imperative language" [34], which does not include block structure, can be represented in $\mathsf{IA} - \{\texttt{new}\}$, as shown in Section 1. However, recall that the process of translating an imperative language into IA forced us to be more explicit about free and bound variables. The "basic imperative language" of the textbooks actually relies on an implicit convention by which the program variables, which are all global, are bound (and possibly initialized) at the top level. We claim that it is only when identifiers of type $\texttt{var}[X]$ are bound to actual "storage cell objects"—which is exactly what the `new` constants do—that real imperative behaviour arises.

Of course, to substantiate this claim, we must show, not only that our simple specification of a "functional model" for $\mathsf{IA} - \{\texttt{new}\}$ suffices to interpret the syntax, but that actual models so arising do faithfully reflect the concepts in the language, and capture the operational behaviour of programs. We can in fact do this in a very strong sense. As we shall see in Section 6, the categories of games used to give the first syntax-independent constructions of fully abstract models for PCF [2, 11], when used to give models for $\mathsf{IA} - \{\texttt{new}\}$ in the way we have described, again yield fully abstract models. Moreover, the *proof* of full abstraction is a very easy extension of that for PCF, and can be given at the axiomatic level introduced in [1]. This latter point means that *any* model of the axioms in [1] yields a fully abstract model of $\mathsf{IA} - \{\texttt{new}\}$.

Firstly, however, we shall turn to the question of modelling the `new` constants.

4 The semantics of `new`

Our previous discussion has located the functional/imperative boundary, the point at which genuinely "stateful" behaviour arises, in the semantics of the

`new` constant. What are the key features of this construct?

Locality The "object" created by a local declaration

$$\texttt{new}\ x\ \texttt{in}\ C$$

must be "private" to C. This causes problems for traditional models based on representing the state in a global, monolithic fashion by a mapping from "locations" to values. The functor-category approaches [22, 33] address this problem by replacing the global state by a functor varying over "stages".

Irreversibility When a variable is updated, the previous value is lost. Again, models based on representing states as functions find it hard to account for this feature. For good discussions of this point see [18, 26].

Sharing Multiple occurrences of a variable in a functional program refer, conceptually at least, to different "copies" of the same, unchanging value ("referential transparency"); this implies that the temporal order in which these occurrences are dereferenced makes no difference to the outcome. By contrast, multiple references to an assignable variable refer to different time-slots in the life of a single underlying object with state which changes over time; this is *sharing* rather than *copying*.

How can we capture these features? The point of view we wish to adopt is one we have already hinted at, and indeed appears in a significant line of previous work [15, 16, 26]. We want to understand $\texttt{new}\ x\ \texttt{in}\ C$ as binding the free identifier x of type $\texttt{var}[X]$ to an "object" or "process" which gives the behaviour of a storage cell. The behaviour of $\texttt{new}\ x\ \texttt{in}\ C$ then arises from the *interaction* between C and this cell, which is "internalized", i.e. hidden from the environment. Such an account immediately addresses two of the key features of `new` noted above:

- Locality is addressed, since the interaction between C and the storage cell process is hidden from the environment.
- Irreversibility is addressed, since the state of the storage cell will change as C interacts with it.

How can we formalize this idea in our current framework? A first attempt is to consider introducing a constant

$$\texttt{cell}_X : 1 \longrightarrow \texttt{var}[X]$$

such that, if $f : \texttt{var}[X] \longrightarrow A$, $\texttt{new}(f)$ is given by the composition

$$\texttt{new}(f) = 1 \xrightarrow{\texttt{cell}_X} \texttt{var}[X] \xrightarrow{f} A.$$

The idea is that $\mathtt{cell}_X$ gives the "behaviour" of our storage cell process. However, recalling that

$$\mathtt{var}[X] = \mathtt{com}^X \times \mathtt{exp}[X]$$

this is clearly hopeless, since a constant of this type, which in particular will supply a constant value every time we read from the variable, is clearly just what we *don't* want!—We need to take account of the changing state of the variable.

At this point we produce our *deus ex machina*: Linear Logic! Up to this point, we have been working exclusively with intuitionistic types; since everything except `new` was essentially functional, this was all we needed, at least to get a model. But now we need a loop-hole to get some access to the dynamics, and Linear Logic provides such a loop-hole. Suppose then that our cartesian closed category $\mathcal{C}$ arises as $\mathcal{C} = K_!(\mathcal{L})$, the co-Kleisli category of a Linear category $\mathcal{L}$ with respect to the ! comonad [6, 29]. The intuitionistic function types we have been using get their standard decompositions into the Linear types:

$$A \Rightarrow B = !A \multimap B.$$

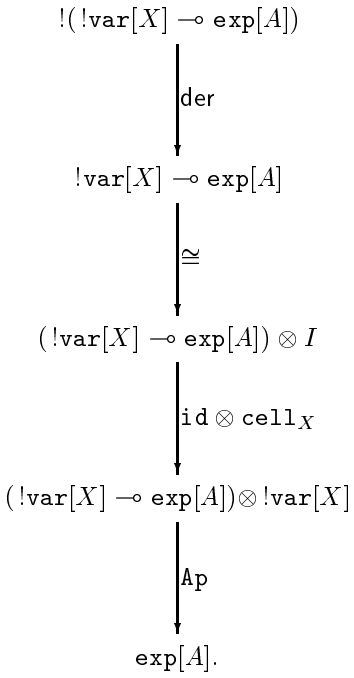
In particular, we see that the type of $\mathtt{new}_A$ is:

$$\mathtt{new}_{X,A} : !(!\mathtt{var}[X] \multimap \mathtt{exp}[A]) \multimap \mathtt{exp}[A].$$

Now suppose that we have a morphism

$$\mathtt{cell}_X : I \longrightarrow !\mathtt{var}[X].$$

Then we can define $\mathtt{new}_A$ as the composition



(Here der is the dereliction map (the counit of !), and Ap is the linear application.)

Note that Linear types really are necessary here. If we had a constant $\mathtt{cell}_X : \mathtt{var}[X]$ in the language, and tried to define

$$\mathtt{new}f = f\ \mathtt{cell}_X,$$

then this would be interpreted in $K_!(\mathcal{L})$ by

$$I \xrightarrow{\mathtt{cell}_0^\dagger} !\mathtt{var}[X] \xrightarrow{\quad f \quad} A$$

where $\mathtt{cell}_0^\dagger$ is the "promotion" of a morphism

$$\mathtt{cell}_0 : I \longrightarrow \mathtt{var}[X].$$

But the promotion will behave "uniformly" in each copy of $\mathtt{var}[X]$, whereas we clearly need behaviour which is history-sensitive, and depends on the previous history of accesses to other "copies" (which are really the previous time slots of the single shared underlying object with state). Thus the cell morphism we require will *not* be of the form $\mathtt{cell}_0^\dagger$ for any $\mathtt{cell}_0 : I \to \mathtt{var}[X]$.

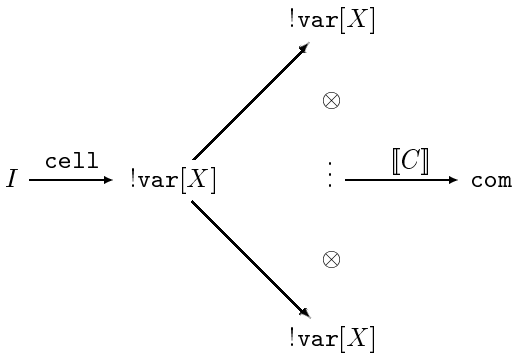
Provided that we can define a suitable morphism

$$\mathtt{cell}_X : I \longrightarrow !\mathtt{var}[X]$$

which does capture the behaviour of a storage cell object, then we have completed our semantics of Idealized Algol. In Section 7 we shall see that this can indeed be done for a suitable category of games, and by this means we will obtain the first fully abstract model of Idealized Algol.

The point to be emphasized here is how small an increment from the modelling of PCF is required to capture Idealized Algol, *provided* a sufficient handle on the dynamics is present in our semantics in order to define the `cell` morphism. The key feature of game semantics is that *the dynamics is already there*.

How is sharing represented in this approach? Firstly, the multiple references to a variable are interpreted using the cocommutative comonoid structure of $!\mathtt{var}[X]$, i.e. the contraction rule, so that the interpretation of a block $\mathtt{new}\ x\ \mathtt{in}\ C$ looks like:



The contraction merges the accesses to the variable x arising from the various occurrences of it in C into a single "event stream". The task of the `cell` morphism is to impose the appropriate causality on this event stream, so that in particular a read will return the last value written.

5 Games

We will describe three categories of games in this paper:

- $\mathcal{G}$: a linear category of games;
- $\mathcal{C}$: a cartesian closed category derived from $\mathcal{G}$, essentially by a co-Kleisli construction; and
- $\mathcal{C}/\lesssim$: an "extensional" (but not well-pointed!) category, obtained from $\mathcal{C}$ by quotienting by the intrinsic preorder.

$\mathcal{G}$ will be basic to all our work. We will present a model of Idealized Algol in $\mathcal{C}$ and prove the key results on Computational Adequacy and Definability for that model. The full abstraction of $\mathcal{C}/\lesssim$ will then be an easy consequence.

An important technical role will be played by a lluf (same objects, fewer morphisms) subcategory $\mathcal{G}_{\rm inn} \hookrightarrow \mathcal{G}$ of *innocent* strategies. For contrast, we will call the strategies in the larger category *knowing*.

The category $\mathcal{G}_{\rm inn}$ was introduced in [13, 14] and used there to construct a fully abstract model for FPC, a functional language with sums, products, functions and recursive types. It is a generalization of Hyland and Ong's previous category of dialogue games and innocent strategies [11], with many improvements in definitions, presentation and identification of key lemmas based on the ideas in [2].

Most of the verification that the structures described below have the required properties is identical to that for the categories described in [13], so we shall refer extensively to that work for proofs.

5.1 Arenas, views and legal positions

Definition An *arena* A is specified by a structure $\langle M_A, \lambda_A, \vdash_A \rangle$ 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 the set $\{\mathsf{O},\mathsf{P}\} \times \{\mathsf{Q},\mathsf{A}\}$ as $\{\mathsf{OQ},\mathsf{OA},\mathsf{PQ},\mathsf{PA}\}$, and use λ_A^{OP} to mean λ_A followed by left projection, so that $\lambda_A^{\mathsf{OP}}(m) = \mathsf{O}$ if $\lambda_A(m) = \mathsf{OQ}$ or $\lambda_A(m) = \mathsf{OA}$. Define λ_A^{QA} in a similar way. Finally, $\overline{\lambda}_A$ 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 $M_A + \{\star\}$ (where $\star$ is just a dummy symbol) and M_A, called *enabling*, which satisfies

  (e1) $\star \vdash_A m \Rightarrow \lambda_A(m) = \mathsf{OQ} \wedge [n \vdash_A m \iff n = \star]$;

  (e2) $m \vdash_A n \wedge \lambda_A^{\mathsf{QA}}(n) = \mathsf{A} \Rightarrow \lambda_A^{\mathsf{QA}}(m) = \mathsf{Q}$;

  (e3) $m \vdash_A n \wedge m \neq \star \Rightarrow \lambda_A^{\mathsf{OP}}(m) \neq \lambda_A^{\mathsf{OP}}(n)$.

The idea of the enabling relation is that when a game is played, a move can only be made if a move has already been made to enable it. The $\star$ enabler is special—it says which moves are enabled at the outset. A move m such that $\star \vdash_A m$ is called *initial*. Conditions (e2) and (e3) say that answers are enabled by questions, and that the protagonists always enable each other's moves, never their own.

Given an arena, we are interested in sequences of moves of a certain kind. Before defining these, let us fix our notation for operations on sequences. If s and t are sequences, we write st for their concatenation. We also write sa for the sequence s with element a appended. Sometimes we use the notation $s \cdot t$ or $s \cdot a$ when it aids legibility. The empty sequence is written as ε.

Definition A *justified sequence* in an arena A is a sequence s of moves of A, together with an associated sequence of pointers: for each non-initial move m in s, there is a pointer to a move n earlier in s such that $n \vdash_A m$. We say that the move n *justifies* m. Note that the first move in any justified sequence must be initial, since it cannot possibly have a pointer to an earlier move attached to it; so by (e1), justified sequences always start with an opponent question.

Given a justified sequence s, define the *player view* $\ulcorner s \urcorner$ and *opponent view* $\llcorner s \lrcorner$ of s by induction on $|s|$, as follows.

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

The view of a sequence is intended to represent the "currently relevant" subsequence of moves. However, notice that the view of a justified sequence need not itself be justified: the appearance of a move m in the view does not guarantee the appearance of its justifier. This will be rectified when we impose the *visibility condition*, to follow.

A justified sequence s is *well-formed* if it satisfies

(w1) Players alternate: if $s = s_1 mns_2$ then $\lambda^{\mathsf{OP}}(m) \neq \lambda^{\mathsf{OP}}(n)$.

(w2) 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 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*.

A useful intuition is to think of questions as left parentheses, (, and answers as right parentheses,). In order to satisfy the bracketing condition, the string of brackets must be a prefix of a well-formed string of brackets, and

furthermore each) must be justified by the corresponding (. Of course this is where the name "bracketing condition" comes from.

A well-formed sequence s is *legal*, or is a *legal position*, if it also satisfies the following *visibility condition*:

- if $tm \sqsubseteq s$ where m is a P-move, then the justifier of m occurs in $\ulcorner t \urcorner$.
- if $tm \sqsubseteq s$ where m is a non-initial O-move, then the justifier of m occurs in $\llcorner t \lrcorner$.

We write L_A for the set of legal positions of A.

5.2 Games and strategies

Definition Let s be a legal position of an arena A and let m be a move in s. We say that m is *hereditarily justified* by an occurrence of a move n in s if the chain of justification pointers leading back from m ends at n, i.e. m is justified by some move m_1, which is in turn justified by m_2 and so on until some m_k is justified by an initial move n. We write $s \upharpoonright n$ for the subsequence of s containing all moves hereditarily justified by n. This notation is slightly ambiguous, because it confuses the move n with a particular occurrence of n; however, no difficulty will arise in practice. We similarly define $s \upharpoonright I$ for a set I of (occurrences of) initial moves in s to be the subsequence of s consisting of all moves hereditarily justified by a move of I.

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

- $\langle M_A, \lambda_A, \vdash_A \rangle$ is an arena.
- P_A is a non-empty, prefix-closed subset of L_A, called the *valid positions*, and satisfying

$$\text{if } s \in P_A \text{ and } I \text{ is a set of initial moves of } s \text{ then } s \upharpoonright I \in P_A.$$

5.2.1 Multiplicatives

Given games A and B, define new games $A \otimes B$ and $A \multimap B$ as follows.

$$\begin{array}{rcl} M_{A\otimes B} & = & M_A + M_B. \\ \lambda_{A\otimes B} & = & [\lambda_A, \lambda_B]. \\ \star \vdash_{A\otimes B} n & \iff & \star \vdash_A n \vee \star \vdash_B n. \\ m \vdash_{A\otimes B} n & \iff & m \vdash_A n \vee m \vdash_B n. \\ P_{A\otimes B} & = & \{s \in L_{A\otimes B} \mid s \upharpoonright A \in P_A \wedge s \upharpoonright B \in P_B\}. \end{array}$$

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

In the above, $s \upharpoonright A$ denotes the subsequence of s consisting of all moves from M_A; $s \upharpoonright B$ is analogous. The conflict with the previously introduced notation $s \upharpoonright I$ should not cause any confusion.

The tensor unit is defined by $I = \langle \emptyset, \emptyset, \emptyset, \{\varepsilon\} \rangle$.

5.2.2 Strategies

Definition A *strategy* σ for a game A is a non-empty set of even-length positions from P_A, satisfying

(s1) $sab \in \sigma \Rightarrow s \in \sigma$.

(s2) $sab, sac \in \sigma \Rightarrow b = c$, and the justifier of b is the same as that of c. In other words, the justified sequences sab and sac are identical.

The *identity* strategy for a game A is a strategy for $A \multimap A$ defined by

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

We use subscripts to distinguish the two occurrences of A, and write $t \sqsubseteq^{\text{even}} s$ to mean that t is an even-length prefix of s.

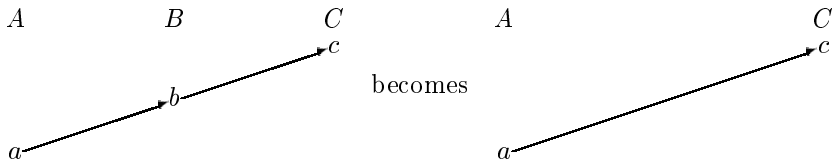
All that id_A does is to copy the move made by Opponent in one copy of A to the other copy of A. The justifier for Player's move is the copy of the justifier of Opponent's move. It is easy to check that this does indeed define a strategy.

5.2.3 Composition

The categories we will work in have games as objects and strategies as morphisms. Therefore, given strategies $\sigma : A \multimap B$ and $\tau : B \multimap C$, we would like to compose them to form a strategy $\sigma ; \tau : A \multimap C$. First, some auxiliary definitions are necessary.

Definition Let u be a sequence of moves from games A, B and C together with justification pointers from all moves except those initial in C. Define $u \upharpoonright B, C$ to be the subsequence of u consisting of all moves from B and C; if a pointer from one of these points to a move of A, delete that pointer. Similarly define $u \upharpoonright A, B$. We say that u is an *interaction sequence* of A, B and C if $u \upharpoonright A, B \in P_{A \multimap B}$ and $u \upharpoonright B, C \in P_{B \multimap C}$. The set of all such sequences is written as $\mathsf{int}(A, B, C)$.

Suppose $u \in \mathsf{int}(A, B, C)$. A pointer from a C-move must be to another C-move, and a pointer from an A-move a must be either to another A-move, or to an *initial* B-move, b, which in turn must have a pointer to an initial C-move, c. Define $u \upharpoonright A, C$ to be the subsequence of u consisting of all moves from A and C, except that in the case outlined above, the pointer from a is changed to point to c.



Given strategies $\sigma : A \multimap B$ and $\tau : B \multimap C$, define $\sigma \parallel \tau$ to be

$$\{u \in \mathsf{int}(A, B, C) \mid u \upharpoonright A, B \in \sigma \wedge u \upharpoonright B, C \in \tau\}.$$

We are now ready to define the composite of two strategies.

Definition If $\sigma : A \multimap B$ and $\tau : B \multimap C$, define $\sigma ; \tau : A \multimap C$ by

$$\sigma ; \tau = \{u \upharpoonright A, C \mid u \in \sigma \parallel \tau\}.$$

5.2.4 Innocent strategies

Definition A strategy $\sigma : A$ is *innocent* if and only if it satisfies

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

In other words, how σ plays depends only on the current P-view.

5.2.5 The categories $\mathcal{G}$ and $\mathcal{G}_{\mathrm{inn}}$

The objects of $\mathcal{G}$ and its lluf subcategory $\mathcal{G}_{\mathrm{inn}}$ are games. A morphism from A to B in $\mathcal{G}$ is a strategy $\sigma : A \multimap B$. The lluf subcategory $\mathcal{G}_{\mathrm{inn}}$ has as morphisms only the innocent strategies. Composition and identities are as described above.

Proposition 1 $\mathcal{G}$ and $\mathcal{G}_{\mathrm{inn}}$ are categories.

5.2.6 Monoidal structure

We have already given the object part of the tensor product. We now describe the corresponding action on morphisms which makes tensor into a bifunctor and $\mathcal{G}$ into a symmetric monoidal category.

Given $\sigma : A \to B$ and $\tau : C \to D$, define $\sigma \otimes \tau : (A \otimes C) \multimap (B \otimes D)$ by

$$\sigma \otimes \tau = \{s \in L_{A \otimes C \multimap B \otimes D} \mid s \upharpoonright A, B \in \sigma \wedge s \upharpoonright C, D \in \tau\}.$$

We can now define natural isomorphisms unit, assoc and comm with components $\mathsf{unit_A} : \mathsf{A} \otimes \mathsf{I} \to \mathsf{A}$, $\mathsf{assoc_{A,B,C}} : \mathsf{A} \otimes (\mathsf{B} \otimes \mathsf{C}) \to (\mathsf{A} \otimes \mathsf{B}) \otimes \mathsf{C}$ and $\mathsf{comm_{A,B}} : \mathsf{A} \otimes \mathsf{B} \to \mathsf{B} \otimes \mathsf{A}$ given by the obvious copycat strategies—in each case the set of moves of the domain game is isomorphic to the set of moves of the codomain game. It is then trivial to verify the following.

Proposition 2 The structure described above makes $\mathcal{G}$ into a symmetric monoidal category. $\mathcal{G}_{\mathrm{inn}}$ is a sub-symmetric monoidal category of $\mathcal{G}$.

5.2.7 Closed structure

To make $\mathcal{G}$ into a symmetric monoidal *closed* category, we need to show that each functor $- \otimes B$ has a (specified) right adjoint. Observe first that the only difference between games $A \otimes B \multimap C$ and $A \multimap (B \multimap C)$ is in the tagging of moves in the disjoint unions. Therefore

$$\begin{aligned} \mathcal{G}(A \otimes B, C) &= \{\sigma \mid \sigma \text{ is a strategy for } A \otimes B \multimap C\} \\ &\cong \{\sigma \mid \sigma \text{ is a strategy for } A \multimap (B \multimap C)\} \\ &= \mathcal{G}(A, B \multimap C). \end{aligned}$$

Denote this isomorphism by $\Lambda_B(-)$. This structure gives us:

Proposition 3 $\mathcal{G}$ is an autonomous (i.e. symmetric monoidal closed) category. $\mathcal{G}_{\mathrm{inn}}$ is a sub-autonomous category of $\mathcal{G}$.

5.2.8 Products

Given games A and B, define a game $A\&B$ as follows.

$$\begin{aligned} M_{A\&B} &= M_A + M_B \\ \lambda_{A\&B} &= [\lambda_A, \lambda_B] \\ \star \vdash_{A\&B} n &\iff \star \vdash_A n \vee \star \vdash_B n \\ m \vdash_{A\&B} n &\iff m \vdash_A n \vee m \vdash_B n \\ P_{A\&B} &= \{s \in L_{A\&B} \mid s \upharpoonright A \in P_A \wedge s \upharpoonright B = \varepsilon\} \\ &\cup \{s \in L_{A\&B} \mid s \upharpoonright B \in P_B \wedge s \upharpoonright A = \varepsilon\}. \end{aligned}$$

We can now define projections $\pi_1 : A\&B \to A$ and $\pi_2 : A\&B \to B$ by the obvious copycat strategies. Given $\sigma : C \to A$ and $\tau : C \to B$, define $\langle \sigma, \tau \rangle : C \to A\&B$ by

$$\begin{aligned} \langle \sigma, \tau \rangle &= \{s \in L_{C \multimap A\&B} \mid s \upharpoonright C, A \in \sigma \wedge s \upharpoonright B = \varepsilon\} \\ &\cup \{s \in L_{C \multimap A\&B} \mid s \upharpoonright C, B \in \tau \wedge s \upharpoonright A = \varepsilon\}. \end{aligned}$$

Proposition 4 $A\&B$ is the product of A and B in both $\mathcal{G}$ and $\mathcal{G}_{\mathrm{inn}}$, with projections given by π_1 and π_2.

It should be clear how this definition generalizes to give all products.

5.3 Exponential

Definition Given a game A, define the game $!A$ as follows.

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

5.3.1 Promotion

Given a map $\sigma : !A \to B$, we wish to define its promotion $\sigma^\dagger : !A \to !B$ to be a strategy which plays "several copies of σ". However, in general this cannot be done because there is no way for $\sigma^\dagger$ to know how the many threads of dialogue in $!A \multimap !B$ should be grouped together to give dialogues in $!A \multimap B$. There is a class of games B for which this can be done, however: the well-opened games.

Definition A game A is *well-opened* iff for all $sm \in P_A$ with m initial, $s = \varepsilon$.

In a well-opened game, initial moves can only happen at the first move, so there is only ever a single thread of dialogue. Note that if B is well-opened then so is $A \multimap B$ for any game A, so while $!A$ is not well-opened except in pathological cases, the game $!A \multimap B$ is well-opened whenever B is. We are going to construct a cartesian closed category in which all games are well-opened and exponentials (in the ordinary sense, not the linear logic one) are given by $!A \multimap B$, so this observation is important.

Given a map $\sigma : !A \to B$, define its promotion $\sigma^\dagger : !A \to !B$ by

$$\sigma^\dagger = \{s \in L_{!A \multimap !B} \mid \text{for all initial } m, s \upharpoonright m \in \sigma\}.$$

Proposition 5 If A and B are well-opened games, and σ is a strategy for $!A \multimap B$, then $\sigma^\dagger$ is a strategy for $!A \multimap !B$. If σ is innocent then so is $\sigma^\dagger$.

5.3.2 Dereliction

For well-opened games A, we can define $\mathsf{der}_A : !A \to A$ to be the copycat strategy

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

Dereliction and promotion behave as expected where they are defined.

Proposition 6 Let A, B and C be well-opened games, and let $\sigma : !A \multimap B$ and $\tau : !B \multimap C$ be strategies. Then

- $\sigma^\dagger ; \mathsf{der}_B = \sigma$,
- $\mathsf{der}_A^\dagger = \mathsf{id}_{!A}$, and

- $\sigma^\dagger ; \tau^\dagger = (\sigma^\dagger ; \tau)^\dagger$.

We now note an important lemma.

Lemma 7 (Bang Lemma) If B is well-opened and $\sigma : !A \to !B$ is innocent then $\sigma = (\sigma ; \mathsf{der}_B)^\dagger$.

5.3.3 Contraction

We define $\mathsf{con}_A : !A \to !A \otimes !A$. For any $s \in P_{!A_0 \multimap !A_1 \otimes !A_2}$, let I be the set of occurrences of initial moves in A_1 and J be the set of occurrences of initial moves in A_2. Let $s_1 = s \upharpoonright I$ and $s_2 = s \upharpoonright J$. Then define con_A as

$$\{s \in P_{!A_0 \multimap !A_1 \otimes !A_2} \mid \forall t \sqsubseteq^{\mathrm{even}} s.(t_1 \upharpoonright !A_0 = t_1 \upharpoonright !A_1) \wedge (t_2 \upharpoonright !A_0 = t_2 \upharpoonright !A_2)\}.$$

5.3.4 Exponential isomorphisms

These reduce to *identities* in the present setting:

$$\begin{aligned} !(A \& B) &= !A \otimes !B. \\ I &= !I. \end{aligned}$$

5.4 The cartesian closed categories $\mathcal{C}$ and $\mathcal{C}_{\mathrm{inn}}$

We can now define the cartesian closed category of games $\mathcal{C}$, in which we will model Idealized Algol.

$$\begin{aligned} \text{Objects} &: \text{Well-opened games} \\ \text{Morphisms } \sigma : A \to B &: \text{Strategies for } !A \multimap B \end{aligned}$$

For any well-opened game A, the strategy $\mathsf{der}_A : !A \multimap A$ is the identity map on A, and given morphisms $\sigma : A \to B$ and $\tau : B \to C$, that is to say strategies $\sigma : !A \multimap B$ and $\tau : !B \multimap C$, we define the composite morphism $\sigma \fatsemi \tau : A \to C$ to be $\sigma^\dagger ; \tau$.

The innocent strategies form a lluf subcategory $\mathcal{C}_{\mathrm{inn}}$ of $\mathcal{C}$.

Products in $\mathcal{C}$ and $\mathcal{C}_{\mathrm{inn}}$ are constructed as in $\mathcal{G}$: set $A \times B = A \& B$.

Moreover,

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

So we can define $A \Rightarrow B$ to be the game $!A \multimap B$, giving cartesian closure.

Proposition 8 $\mathcal{C}$ is a cartesian closed category, and $\mathcal{C}_{\mathrm{inn}}$ is a lluf sub-cartesian closed category of $\mathcal{C}$.

5.4.1 Order enrichment

The strategies for a game A are easily seen to form a directed-complete partial order under the inclusion ordering, with least element $-= \{\varepsilon\}$. Moreover, composition, tensor, currying etc. are all continuous with respect to this order. Applying this to the hom-objects $A \multimap B$, we obtain:

Proposition 9 $\mathcal{G}$ is a CPO-enriched autonomous category. $\mathcal{C}$ is a CPO-enriched cartesian closed category.

For any innocent strategy $\sigma : A$, define the *view-function* of σ to be the partial function f from P-views to P-moves defined by

$$f(v) = b \iff \exists sa.sab \in \sigma \wedge \ulcorner sa \urcorner = v.$$

Proposition 10 The knowing strategies for a game A form a dI-domain; the compact strategies are those with a finite set of positions.

The innocent strategies for a game A also form a dI-domain, with the compact strategies being those with finite view-function.

Note that an innocent strategy with finite view-function is not necessarily finite qua set of positions—this was the point of introducing view-functions in the first place. For example, by the Bang Lemma any innocent strategy for $!A$ other than $-$ has an infinite set of positions. Thus we shall speak of an innocent strategy $\sigma : A$ with finite view function being *innocently compact* i.e. compact in $\mathcal{G}_{\mathrm{inn}}(I, A)$.

5.5 Standard datatypes

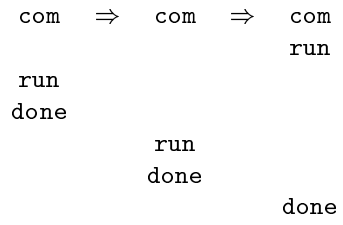
As described in [1], the category $\mathcal{C}$ has standard datatypes given by flat games. For any set X, the game $\widetilde{X_\star}$ has one initial question q, and one answer for each element $x \in X$. The valid positions of the game are given by the set

$$\{\varepsilon, q\} \cup \{qx \mid x \in X\}.$$

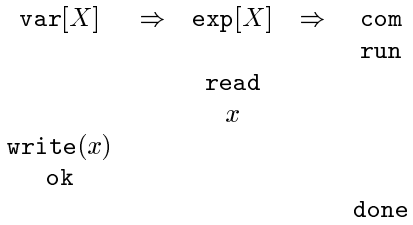
For any $x \in X$, the morphism $\bar{x} : 1 \to \widetilde{X_\star}$ is the strategy $\{\varepsilon, qx\}$ which responds to the question q with answer x. For a family of maps $(f_x : 1 \to A \mid x \in X)$, the morphism $[f_x \mid x \in X] : \widetilde{X_\star} \to A$ is the strategy which responds to the initial move of A by playing q in $\widetilde{X_\star}$, and after receiving the answer x, continues to play in A according to f_x.

Following the interpretation of $\mathsf{IA} - \{\mathtt{new}\}$ outlined in Section 2, the type $\mathtt{exp}[X]$ is interpreted as the game $\widetilde{X_\star}$. In the case of $\mathtt{com} = \mathtt{exp}[\mathsf{1}]$, we shall call the initial question $\mathtt{run}$ and the unique answer $\mathtt{done}$. The type $\mathtt{var}[X]$ is interpreted as $\mathtt{exp}[X] \times \mathtt{com}^X$; we shall refer to the initial question of the first

component as `read`, and to the moves of the xth copy of `com` as `write`(x) and `ok`. The strategy interpreting $\mathtt{seq}_{\mathtt{com}}$ is therefore:



and the interpretation of `assign` is as follows.



Note that these strategies are innocent; in fact the whole machinery of standard datatypes works inside $\mathcal{C}_{\mathrm{inn}}$, so our interpretation of $\mathsf{IA} - \{\mathtt{new}\}$ uses only innocent strategies.

5.6 Intrinsic Preorder

Our full abstraction result will in fact hold not in $\mathcal{C}$, but in the quotient of $\mathcal{C}$ with respect to a certain preorder, which we now define. Let Σ be the game with a single question q and one answer a (so Σ is isomorphic to the game interpreting `com` as described above). There are only two strategies for Σ: $-\!\!= \{\varepsilon\}$ and $\top = \{\varepsilon, qa\}$. Maps $\alpha : A \to \Sigma$ in $\mathcal{C}$ can be thought of as tests on strategies for A: a strategy σ passes the test if $\sigma \fatsemi \alpha = \top$. The intrinsic preorder for strategies on A is defined as follows.

$$\sigma \lesssim \tau \text{ iff } \forall \alpha : A \to \Sigma . \sigma \fatsemi \alpha = \top \Rightarrow \tau \fatsemi \alpha = \top.$$

So $\sigma \lesssim \tau$ if τ passes every test passed by σ. It is straightforward to show the following.

Proposition 11 $\lesssim$ is a preorder on each hom-set of $\mathcal{C}$, and the quotient $\mathcal{C}/\!\lesssim$ of $\mathcal{C}$ by $\lesssim$ is a poset-enriched cartesian closed category.

6 $\mathsf{IA} - \{\mathtt{new}\}$

The structure exhibited for $\mathcal{G}_{\mathrm{inn}}$ in the previous section allows us to model $\mathsf{IA} - \{\mathtt{new}\}$ according to the definitions given in Section 2. Since innocent strate-

gies correspond precisely to functional programs, as shown in [11, 13, 14], this provides some immediate confirmation of our thesis that $\mathsf{IA} - \{\mathtt{new}\}$ is a pure functional language.

The aim of this section is to prove the following result.

Theorem 12 (Innocent Definability) Let T be an IA type. Every compact innocent strategy σ on the game $[\![T]\!]$ is definable in $\mathsf{IA} - \{\mathtt{new}\}$; i.e. for some closed term $\vdash M : T$, $[\![M]\!] = \sigma$.

We will in fact prove a more general result at the axiomatic level introduced in [1]. (We shall assume the material in [1] as background.)

Theorem 13 (Definability for Normed Sequential Categories) Let $\mathcal{C}$ be a normed sequential category with all small products (where it is understood that π-atomicity now refers to arbitrary products rather than just finite ones). Any such category gives rise to a model of $\mathsf{IA} - \{\mathtt{new}\}$ as specified in Section 2. Then for any IA type T and compact morphism $f : 1 \to [\![T]\!]$ in $\mathcal{C}$, f is definable in $\mathsf{IA} - \{\mathtt{new}\}$.

Since $\mathcal{C}_{\mathrm{inn}}$ is a normed sequential category, as shown in [1] and [13], Theorem 13 implies Theorem 12.

Proof The proof is a straightforward extension of the proofs of the Decomposition Theorem (Theorem 4.1) and the Definability Theorem for Normed Categories (Theorem 8.2) in [1]. We shall just describe the additional steps in the argument. For notational simplicity, we shall assume that the only basic data types are $\mathbf{N}$ and the unit type.

Suppose then that $T = T_1 \Rightarrow T_2 \Rightarrow \ldots \Rightarrow T_k \Rightarrow B$ is an IA type, where $T_i = U_{i,1} \Rightarrow \ldots \Rightarrow U_{i,q_i} \Rightarrow B_i$, $1 \leq i \leq k$, and that

$$f : 1 \to [\![T]\!]$$

is a compact morphism in $\mathcal{C}$.

Suppose first that $[\![B]\!] = \widetilde{X_\star}$ (so that $B = \mathtt{exp}[X]$ or $B = \mathtt{com} = \mathtt{exp}[1]$). Since $[\![B]\!]$ is discrete, we can apply a case analysis:

- $f = -$. Then $f = [\![\lambda \vec{x} : \vec{T}.\Omega]\!]$, where we write $\vec{T} = T_1, \ldots, T_k$.
- f is (the exponential transpose of)

$$!(T_1 \& \cdots \& T_k) \longrightarrow 1 \xrightarrow{\bar{c}} \widetilde{X_\star}$$

  for a (necessarily unique) $c \in X$. Then if $B = \mathtt{exp}[\mathbf{N}]$,

$$f = [\![\lambda \vec{x} : \vec{T}.\mathbf{c}]\!],$$

  while if $B = \mathtt{com}$, $f = [\![\lambda \vec{x} : \vec{T}.\mathtt{skip}]\!]$.

- $f = C_i(f_1, \ldots, f_{q_i}, (g_0, \ldots, g_{l\perp 1}))$ where $B_i = \mathtt{exp}[\mathbf{N}]$. Then

$$f = [\![\lambda \vec{x} : \vec{T}.\mathtt{case}_l(x_i(M_1\vec{x})\cdots(M_{q_i}\vec{x}))(P_0\vec{x})\cdots(P_{l\perp 1}\vec{x})]\!]$$

  where $[\![M_j]\!] = f_j$ for $1 \leq j \leq q_i$, and $[\![P_j]\!] = g_j$ for $0 \leq j < l$.

- $f = C_i(f_1, \ldots, f_{q_i}, (g_\bullet))$ where $B_i = \mathtt{com}$. Then

$$f = [\![\lambda \vec{x} : \vec{T}.\mathtt{seq}_B(x_i(M_1\vec{x})\cdots(M_{q_i}\vec{x}))(P\vec{x})]\!]$$

  where $[\![M_j]\!] = f_j$ for $1 \leq j \leq q_i$ and $[\![P]\!] = g_\bullet$.

- $f = C_i(f_1, \ldots, f_{q_i}, (g_0, \ldots, g_{l\perp 1}))$ where $B_i = \mathtt{var}[\mathbf{N}]$. In this case, by the π-atomicity of B, f must factor through one of the projections

$$\mathtt{var}[\mathbf{N}] \xrightarrow{\mathtt{snd}} \mathtt{exp}[\mathbf{N}]$$

  or

$$\mathtt{var}[\mathbf{N}] \xrightarrow{\mathtt{fst};\pi_n} \mathtt{com}.$$

  In the first case,

$$f = [\![\lambda \vec{x} : \vec{T}.\mathtt{case}_l(\mathtt{deref}(x_i(M_1\vec{x})\cdots(M_{q_i}\vec{x}))(P_0\vec{x})\cdots(P_{l\perp 1}\vec{x}))]\!],$$

  while in the second case,

$$f = [\![\lambda \vec{x} : \vec{T}.\mathtt{seq}_B(\mathtt{assign}(x_i(M_1\vec{x})\cdots(M_{q_i}\vec{x}))\mathbf{n})(P\vec{x})]\!]$$

  where M_j, P_j and P are defined inductively as in the previous two cases.

Finally, we consider the case where $B = \mathtt{var}[\mathbf{N}]$. By the universal property of the product,

$$f = \langle \langle f_n \mid n \in \mathbf{N} \rangle, f_e \rangle$$

and by the previous cases, $f_n = [\![P_n]\!]$ for all $n \in \mathbf{N}$, and $f_e = [\![M]\!]$, while $f_n = \perp$ for almost all n by compactness of f. Then for some l,

$$f = [\![\lambda \vec{x} : \vec{T}.\mathtt{mkvar}(\lambda x : \mathtt{exp}[X].\mathtt{case}_l x P_0 \cdots P_{l\perp 1})M]\!]. \qquad \square$$

A normal form for compact terms of $\mathsf{IA} - \{\mathtt{new}\}$ can be extracted from this proof, extending the PCF "evaluation trees" [2].

7 Modelling new

Returning to the discussion in Section 4, to complete our semantics for Idealized Algol, we must specify a morphism

$$\mathtt{cell}_X : I \to !\mathtt{var}[X].$$

Plays in $!\mathtt{var}[X]$ have the form

$$\mathtt{read} \cdot v_0 \cdot \mathtt{write}(v_1) \cdot \mathtt{ok} \cdot \mathtt{read} \cdot v_2 \ldots$$

where each read and each write$(-)$ is initial, and all other moves are justified by the immediately preceding move. Note that the type $!\mathtt{var}[X]$ imposes no causality constraints between the result returned by a read and the value previously written. Thus in the above example we could have $v_2 \neq v_1$. The definition of cell as a deterministic strategy on this game is quite clear; it should respond to a write(v) with ok, and to a read with the value last written, if any. If there has been no write performed, the uninitialized cell will have no response to a read, while an initialized cell $\mathtt{cell}_{X,v_0}$ with initial value v_0 will return v_0 in this case.

This strategy clearly implements the required behaviour, and is a well-defined deterministic strategy. However, it is *not* innocent. Indeed, note that since all Opponent moves are initial, the P-view when Opponent has just moved consists only of the move Opponent has just played. Thus innocent and history-free strategies coincide on this game. This lack of innocence is exactly what allows cell to take account of the previous accesses to the variable, and hence to correctly implement sharing rather than copying.

Thus we find that our model of $\mathsf{IA} - \{\mathtt{new}\}$ lives entirely in the subcategory $\mathcal{C}_{\mathrm{inn}}$ (and implicitly in $\mathcal{G}_{\mathrm{inn}}$), while to model new we have to use the larger category $\mathcal{G}$ of knowing strategies. An obvious question then arises: what subcategory of $\mathcal{G}$ is generated by $\mathcal{G}_{\mathrm{inn}}$ and cell? The answer is provided by the following result.

Theorem 14 (Innocent Factorization) Let A be a game such that

$$sa \in P_A^{\mathrm{odd}} \wedge \ulcorner sa \urcorner b \in P_A \implies sab \in P_A,$$

and $\sigma : I \longrightarrow A$ be any (knowing) strategy. Then there exists a set X, $x_0 \in X$, and an innocent strategy $\tau : !\mathtt{var}[X] \longrightarrow A$ such that

$$\sigma = \mathtt{cell}_{X,x_0}; \tau.$$

Moreover, if σ is compact, τ is innocently compact. Note that this is a weak orthogonality property in the sense of factorization systems [7].

Proof The idea is that τ simulates σ using its view of the play in A together with (a code for) the full previous history of the play, which it keeps in the

variable. Thus we use state to encode history, a standard idea in automata theory; the interesting thing here is that we find a point of contact between machine simulations and factorization systems.

The set X is taken to be (a set of codes for) the positions of A (so it must be a set of cardinality greater than or equal to that of the set of positions of A), and x_0 to be the code for the initial position ε. Note that the P-view of a position s in $!\mathtt{var}[X] \multimap A$ contains the string $\ulcorner s \upharpoonright A \urcorner$. At each O-move a in A, following a previous play s, the strategy τ behaves as follows:

If σ does not have a response at any position of A with player view $\ulcorner sa \upharpoonright A \urcorner$, τ has no response. Otherwise, it reads $\mathtt{code}(t)$ from the variable. Let $b = \sigma(ta)$. If $\ulcorner sa \urcorner b \notin P_A$, τ has no response. Otherwise it writes $\mathtt{code}(tab)$ back into the variable, and then plays b in A.

Note that the assumption on A together with the fact that σ is a valid strategy ensures that τ is well-defined. It is clear that τ is innocent, and that the composite $\mathtt{cell}_{X,x_0} ; \tau = \sigma$, as required. Further, if σ is compact, it only has a response b at a finite number of positions ta. In this case, the strategy τ only has a response at a finite number of views, so τ is innocently compact. □

We must now show that for any IA type T, the game $[\![T]\!]$ satisfies the hypothesis of the above theorem.

Lemma 15 For any IA type T, the valid positions of the game $[\![T]\!]$ are precisely the legal positions with at most one initial move.

Proof By induction on the structure of T. The base cases are trivial, while for the inductive step we make use of several technical lemmas from [13]. Suppose $T = A \Rightarrow B$, so that $[\![T]\!] = ![\![A]\!] \multimap [\![B]\!]$. It is clear that each valid position contains at most one initial move and is legal. For the converse, we proceed by induction on the length of a legal sequence containing at most one initial move. Again the base case is trivial, so consider a legal position sa with at most one initial move, and suppose that $s \in P_{[\![T]\!]}$. To complete the proof we must show that $sa \upharpoonright ![\![A]\!] \in P_{![\![A]\!]}$ and $sa \upharpoonright [\![B]\!] \in P_{[\![B]\!]}$. By the outer inductive hypothesis, it suffices to show that each of these sequences is *legal*. For the alternation condition, in the case a is an O-move, the visibility condition in $[\![T]\!]$ implies that a is in the same component as the last P-move in s, so both so the alternation condition is satisfied. If a is a P-move, the "state diagram" of Lemma 3.2.1 of [13] shows that O played last in both components, so P cannot possibly violate the alternation condition. The bracketing condition is clearly satisfied in each component. For the visibility condition, one can adapt the proof of Lemma 3.1.6 of [13] to show that $\ulcorner s \urcorner \upharpoonright [\![B]\!]$ is a subsequence of $\ulcorner s \upharpoonright B \urcorner$, and similarly for the O-view, and for the other component. Therefore, since the justifier of a is in the appropriate view of s, it is also in the appropriate view of s restricted to the component of a, so the visibility condition is satisfied. □

Proposition 16 For every IA type T, the game $[\![T]\!]$ satisfies the hypothesis of the Innocent Factorization Theorem.

Proof We must show that if $sa \in P^{\mathrm{odd}}_{[\![T]\!]}$ and $\ulcorner sa \urcorner b \in P_{[\![T]\!]}$ then $sab \in P_{[\![T]\!]}$. Since $\ulcorner sa \urcorner b \in P_{[\![T]\!]}$, we know that sab is a justified sequence and that players alternate. The bracketing condition holds in sab because the pending question at sa is the same as that at $\ulcorner sa \urcorner$. The visibility condition for sab is clearly satisfied. Therefore sab is a legal position, and contains at most one initial move since sa does, and b is a Player-move and therefore not initial. So by the previous lemma, $sab \in P_{[\![T]\!]}$. □

Theorem 17 (Definability) Every compact strategy $\sigma : I \longrightarrow [\![T]\!]$ in $\mathcal{C}$, where T is an IA type, is definable in Idealized Algol.

Proof By Theorem 14 and Proposition 16, σ factors as

$$\sigma = \mathtt{cell}_{X,v_0} ; \tau$$

where τ is a compact innocent strategy. By the Innocent Definability Theorem, for some term $\vdash M : \mathtt{var}[X] \Rightarrow T$ of $\mathsf{IA} - \{\mathtt{new}\}$, $\Lambda\tau = [\![M]\!]$. Writing $T = \vec{T} \Rightarrow B$ for some base type B, if $B = \mathtt{exp}[X]$ for some X then

$$\sigma = [\![\lambda \vec{x} : \vec{T}.\mathtt{new}(\lambda v : \mathtt{var}[X].\mathtt{seq}_B(\mathtt{assign}\ v\ \mathbf{v_0})(M\ v\ \vec{x}))]\!].$$

If $B = \mathtt{var}[X]$, one can use the universal property of products together with the above and the constant $\mathtt{mkvar}$ to find a term defining σ. The details are left as an exercise. □

Note that, if the basic data types of IA are countable, then the set of positions of $[\![T]\!]$ is countable for every IA type T, and we can take $X = \mathbf{N}$ in the Definability Theorem.

8 Computational Adequacy

In this section, we introduce the operational semantics of Idealized Algol and demonstrate that the the games model is both sound and computationally adequate, yielding an inequational soundness theorem. We present the operational semantics as a "big-step" evaluation relation, with an auxiliary relation of "structural congruence" (cf. [17]) denoted by $\equiv$. The structural congruence is that generated by β-conversion and all instances of

$$M \equiv \mathbf{Y}M.$$

We shall assume that the only program variables are of type $\mathbf{N}$ for simplicity. Let us introduce some notation. A $\mathtt{var}[\mathbf{N}]$-context $?$ is one of the form $? = x_1 : \mathtt{var}[\mathbf{N}], \ldots, x_n : \mathtt{var}[\mathbf{N}]$, with the x_i distinct. A $?$-store is a partial function $s : \{x_1, \ldots, x_n\} \rightharpoonup \{\mathbf{n} \mid n \in \mathbf{N}\}$. We write $\mathrm{St}(?)$ for the set of all such stores. If $s \in \mathrm{St}(?, x : \mathtt{var}[\mathbf{N}])$, then $s \upharpoonright x$ is the $?$-store obtained by restricting the domain of s, while $(s \mid x \mapsto \mathbf{n})$ denotes the result of updating the contents of x

in s to be $\mathbf{n}$. Note that this operation may extend the domain of s, if $s(x)$ was previously undefined, or may override an existing value for x.

We now define relations of the form

$$?\,, s \vdash M \Downarrow_B c, s'$$

where

- $?$ is a $\mathtt{var}[\mathbf{N}]$-context
- $s, s' \in \mathrm{St}(?)$
- B is a base type
- $? \vdash M : B$ is a term-in-context
- c is a *canonical form* such that $? \vdash c : B$ is derivable.

The canonical forms are:

- at type $\mathtt{exp}[\mathbf{N}]$, the numerals $\mathbf{n}$, and similarly at other expression types;
- at type $\mathtt{com}$, the command $\mathtt{skip}$;
- at type $\mathtt{var}[\mathbf{N}]$, variables x and expressions $\mathtt{mkvar}\ f\ e$.

The relations are defined inductively below. We omit the type tags, and leave the reader to fill in the rules for the extra, unspecified, operations on base types.

Structural congruence

$$\frac{M \equiv M' \quad ?\,, s \vdash M' \Downarrow c, s'}{?\,, s \vdash M \Downarrow c, s'}$$

Canonical forms

$$\frac{}{?\,, s \vdash c \Downarrow c, s}$$

Conditional

$$\frac{?\,, s \vdash M \Downarrow \mathbf{tt}, s' \quad ?\,, s' \vdash N \Downarrow c, s''}{?\,, s \vdash \mathtt{cond}\ M\ N\ P \Downarrow c, s''} \qquad \frac{?\,, s \vdash M \Downarrow \mathbf{ff}, s' \quad ?\,, s' \vdash P \Downarrow c, s''}{?\,, s \vdash \mathtt{cond}\ M\ N\ P \Downarrow c, s''}$$

Arithmetic

$$\frac{?\,, s \vdash M \Downarrow \mathbf{n}, s'}{?\,, s \vdash \mathtt{succ}\ M \Downarrow \mathbf{n}+\mathbf{1}, s'} \qquad \frac{?\,, s \vdash M \Downarrow \mathbf{n}+\mathbf{1}, s'}{?\,, s \vdash \mathtt{pred}\ M \Downarrow \mathbf{n}, s'}$$

Sequencing

$$\frac{?\,, s \vdash M \Downarrow \mathtt{skip}, s' \quad ?\,, s' \vdash N \Downarrow c, s''}{?\,, s \vdash \mathtt{seq}\ M\ N \Downarrow c, s''}$$

Dereferencing

$$\frac{?\,, s \vdash V \Downarrow x, s' \quad s'(x) = \mathbf{n}}{?\,, s \vdash \mathtt{deref}\ V \Downarrow \mathbf{n}, s'} \qquad \frac{?\,, s \vdash V \Downarrow \mathtt{mkvar}\ f\ e, s' \quad ?\,, s' \vdash e \Downarrow \mathbf{n}, s''}{?\,, s \vdash \mathtt{deref}\ V \Downarrow \mathbf{n}, s''}$$

Assignment

$$\frac{?\,, s \vdash E \Downarrow \mathbf{n}, s' \quad ?\,, s' \vdash V \Downarrow x, s''}{?\,, s' \vdash \mathtt{assign}\ V\ E \Downarrow \mathtt{skip}, (s'' \mid x \mapsto \mathbf{n})}$$

$$\frac{?\,, s \vdash E \Downarrow \mathbf{n}, s' \quad ?\,, s' \vdash V \Downarrow \mathtt{mkvar}\ f\ e, s'' \quad ?\,, s'' \vdash f\mathbf{n} \Downarrow \mathtt{skip}, s'''}{?\,, s' \vdash \mathtt{assign}\ V\ E \Downarrow \mathtt{skip}, s'''}$$

Block structure

$$\frac{?\,, x : \mathtt{var}[\mathbf{N}], s \vdash M \Downarrow c, s'}{?\,, s \vdash \mathtt{new}\ \lambda x.M \Downarrow c, s' \upharpoonright x}$$

At first sight, the rule for `new` may seem restricted, since it applies only to terms of the form $\mathtt{new}\ \lambda x.M$. However, there is no loss of expressive power. In our denotational model, the η-law is valid, so for any term M of type $\mathtt{var}[X] \Rightarrow \mathtt{exp}[Y]$, the terms $\mathtt{new}\ M$ and $\mathtt{new}\ \lambda x.Mx$ have the same denotation. In the light of the adequacy result to follow, this means that these two terms are observationally equivalent, so our rule is sufficiently general.

A *program* is a closed term of type `com`. We write $P\Downarrow$ as an abbreviation for $, () \vdash P \Downarrow \mathtt{skip}, ()$ in this case, and $P\Uparrow$ for $\neg P\Downarrow$. Let $\mathsf{Trm}(?\,, T)$ be the set of all M such that $? \vdash M : T$ is derivable. The program contexts $\mathsf{Ctxt}(?\,, T)$ are those contexts $C[-]$ such that $C[M]$ is a program for all $M \in \mathsf{Trm}(?\,, T)$. The

observational preorder $\sqsubseteq_{\cong}$ on $\mathsf{Trm}(?\,, T)$ is then defined by

$$M \sqsubseteq_{\cong} N \text{ iff } \forall C[-] \in \mathsf{Ctxt}(?\,, T).C[M]\Downarrow \Rightarrow C[N]\Downarrow.$$

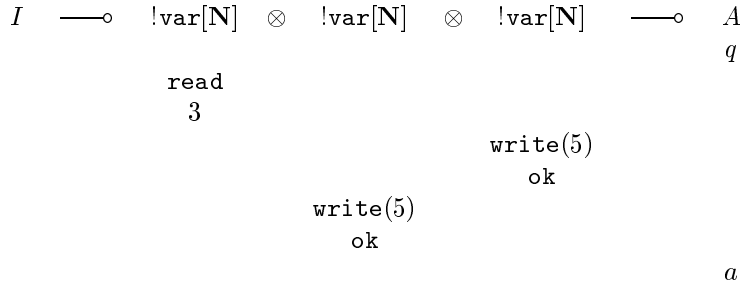
8.1 Making state explicit

The correspondence between the behaviour of the model and the operational semantics is intuitively compelling, but nonetheless the proofs of the soundness and adequacy results which follow are slightly unusual. This is a consequence of the implicit nature of our model of state: the state needs to be made explicit in order for the connection between terms and their interpretations to be made precise.

Consider a $\mathtt{var}[\mathbf{N}]$-context $?$ and a state $s \in \mathrm{St}(?)$. This state can be represented in our model by a strategy $[\![s]\!] : I \multimap\ !\mathtt{var}[\mathbf{N}] \otimes \ldots \otimes\ !\mathtt{var}[\mathbf{N}]$ consisting of a tuple of suitably initialized `cell` strategies. For example, if $?$ is $x : \mathtt{var}[\mathbf{N}], y : \mathtt{var}[\mathbf{N}], z : \mathtt{var}[\mathbf{N}]$ and s is the state $(x \mapsto 3, z \mapsto 7)$ then $[\![s]\!]$ is the strategy

$$I \xrightarrow{\cong} I \otimes I \otimes I \xrightarrow{\mathtt{cell}_3 \otimes \mathtt{cell} \otimes \mathtt{cell}_7} !\mathtt{var}[\mathbf{N}] \otimes !\mathtt{var}[\mathbf{N}] \otimes !\mathtt{var}[\mathbf{N}].$$

The interpretation in the games model of a term $? \vdash M$ in state s is given by the linear composition $[\![s]\!]\,;[\![? \vdash M]\!]$. To study these behaviours more closely, it will be necessary for us to consider the interaction $[\![s]\!] \parallel [\![? \vdash M]\!] \in \mathsf{int}(I, !\mathtt{var}[\mathbf{N}] \otimes \ldots \otimes !\mathtt{var}[\mathbf{N}], A)$, where A is the object interpreting the type of M. For any sequence $qt_1t_2 \in [\![s]\!] \parallel [\![? \vdash M]\!]$, where q is an initial question and t_1 is an even-length sequence of moves in the $!\mathtt{var}[\mathbf{N}] \otimes \ldots \otimes !\mathtt{var}[\mathbf{N}]$ component, we say that t_1 *leaves state* s' if the last write moves in each $!\mathtt{var}[\mathbf{N}]$ are such that each cell is set to the value given by s'. For example, the sequence



in the state s described above leaves the state $(x \mapsto 3, y \mapsto 5, z \mapsto 5)$.

8.2 Soundness

We are now in a position to state the soundness lemma.

Lemma 18 For any $\mathtt{var}[\mathbf{N}]$-context $?$, any term $? \vdash M : A$ and any $s \in \mathrm{St}(?)$, if $?, s \vdash M \Downarrow c, s'$ then each sequence qt in $[\![s]\!] \parallel [\![M]\!]$ has the form qt_1t_2, where t_1 is a sequence of reads and writes in the $[\![?]\!]$ component leaving state s', and $[\![s']\!] \parallel [\![c]\!]$ contains the string qt_2. Further, each sequence in $[\![s']\!] \parallel [\![c]\!]$ arises in this way.

Proof By induction on the derivation of $?, s \vdash M \Downarrow c, s'$. The base cases, for canonical forms, are trivial, and the case of the rule making use of the structural congruence follows from the fact that if $M \equiv N$ then $[\![M]\!] = [\![N]\!]$.

For `new`, we have the rule

$$\frac{?, x : \mathtt{var}[\mathbf{N}], s \vdash M \Downarrow \mathtt{skip}, s'}{?, s \vdash (\mathtt{new}\ \lambda x.M) \Downarrow \mathtt{skip}, s' \upharpoonright x}$$

Notice that $[\![s]\!]$ interpreted as a $?, x : \mathtt{var}[\mathbf{N}]$ state is the extension of the interpretation of $[\![s]\!]$ as a $?$ state by a single uninitialized `cell`, that is

$$I \xrightarrow{\ \cong\ } I \otimes I \xrightarrow{[\![s]\!] \otimes \mathtt{cell}} [\![?]\!] \otimes !\mathtt{var}[\mathbf{N}].$$

By the inductive hypothesis, the sequence $[\![s]\!] \parallel [\![M]\!]$ has the form

$$I \quad \xrightarrow{[\![s]\!]} \quad [\![?]\!] \otimes !\mathtt{var}[\mathbf{N}] \quad \xrightarrow{[\![M]\!]} \quad \begin{array}{l}\mathtt{com}\\ \mathtt{run}\\ \\ \mathtt{done}\end{array}$$
$$t$$

where t leaves state s'. But by definition of $[\![\mathtt{new}]\!]$, the strategy $[\![\mathtt{new}(\lambda x.M)]\!]$ behaves as $\mathtt{cell}; [\![M]\!]$, so $[\![s]\!] \parallel [\![\mathtt{new}(\lambda x.M)]\!]$ behaves as required.

The cases for all the other constants are similar. We consider that of `seq` for illustration. The rule in question is below.

$$\frac{?, s \vdash M \Downarrow \mathtt{skip}, s' \quad ?, s' \vdash N \Downarrow c, s''}{?, s \vdash \mathtt{seq} M N \Downarrow c, s''}$$

By the inductive hypothesis, $[\![s]\!] \parallel [\![M]\!]$ has the form $\mathtt{run} \cdot t_1 \cdot \mathtt{done}$, where t_1 leaves state s', and the sequences of moves in $[\![s']\!] \parallel [\![N]\!]$ have the form qt_2t_3, where t_2 leaves state s''. The sequences in $[\![s'']\!] \parallel [\![c]\!]$ are precisely those of the form qt_3. Because of the copycat nature of $[\![\mathtt{seq}]\!]$, it is then easy to show that $[\![s]\!] \parallel [\![\mathtt{seq} M N]\!]$ consists of strings of the form $qt_1t_2t_3$, giving the result. □

The following simple corollary expresses the sense in which our model is sound.

Proposition 19 If $?, s \vdash M \Downarrow c, s'$ then $[\![s]\!]; [\![M]\!] = [\![s']\!]; [\![c]\!]$.

8.3 Computational Adequacy

We now strengthen our soundness result to show that the model is *computationally adequate*, that is if for some $? \vdash M$ of base type and some state $s \in \mathrm{St}(?)$ we have $[\![s]\!]\,;[\![M]\!] \neq -$, then $?, s \vdash M \Downarrow$. As usual our proof of this result makes use of logical relations; we shall only describe the proof in outline, indicating the points at which it differs from standard such proofs for functional languages.

Define for each type T of IA and each $\mathtt{var}[\mathbf{N}]$-context $?$ a relation $\preccurlyeq^{\Gamma}_{T}$ between strategies $\sigma : ![\![?]\!] \multimap [\![T]\!]$ and terms $? \vdash M : T$ as follows.

- $\sigma \preccurlyeq^{\Gamma}_{\mathtt{com}} M$ iff for all $s \in \mathrm{St}(?)$,

$$[\![s]\!]\,;\sigma \neq - \text{ leaving state } s' \Rightarrow ?, s \vdash M \Downarrow \mathtt{skip}, s'.$$

- $\sigma \preccurlyeq^{\Gamma}_{\mathtt{exp}[\mathbf{N}]} M$ iff for all $s \in \mathrm{St}(?)$,

$$[\![s]\!]\,;\sigma = [\![\mathbf{n}]\!] \text{ leaving state } s' \Rightarrow ?, s \vdash M \Downarrow \mathbf{n}, s'.$$

- $\sigma \preccurlyeq^{\Gamma}_{\mathtt{var}[\mathbf{N}]} M$ iff

$$\sigma\,;\pi_1 \preccurlyeq^{\Gamma}_{\mathtt{exp}[\mathbf{N}]} \mathtt{deref}\ M \wedge \sigma\,;\pi_2\,;\pi_n \preccurlyeq^{\Gamma}_{\mathtt{com}} \mathtt{assign}\ M\ \mathbf{n}.$$

- $\sigma \preccurlyeq^{\Gamma}_{A \Rightarrow B} M$ iff for any $\tau \preccurlyeq^{\Gamma}_{A} N$, $\langle \sigma, \tau \rangle\,;\mathsf{ev} \preccurlyeq^{\Gamma}_{B} MN$ and if $A \Rightarrow B$ is $\mathtt{var}[\mathbf{N}] \Rightarrow \mathtt{exp}[X]$, then $\mathtt{cell}\,;\Lambda^{\perp 1}(\sigma) \preccurlyeq^{\Gamma}_{\mathtt{exp}[X]} \mathtt{new}M$.

In the above and in what follows, given a strategy $\sigma : A \multimap (!\mathtt{var}[\mathbf{N}] \multimap B)$ we use $\mathtt{cell}\,;\Lambda^{\perp 1}(\sigma)$ as an abbreviation for the map

$$A \xrightarrow{\ \cong\ } A \otimes I \xrightarrow{\ \mathsf{id} \otimes \mathtt{cell}\ } A \otimes !\mathtt{var}[\mathbf{N}] \xrightarrow{\ \Lambda^{\perp 1}(\sigma)\ } B.$$

The extra clause in the case of $\mathtt{var}[\mathbf{N}] \Rightarrow \mathtt{com}$ reflects the fact that program variables can be bound in two ways: either by application to a term of type $\mathtt{var}[\mathbf{N}]$, or by allocation of a new local variable.

Lemma 20 Let $?, \Delta \vdash M : T$, where $?$ is a part of the context containing only variables of type $\mathtt{var}[\mathbf{N}]$ while Δ may contain variables of any type (including $\mathtt{var}[\mathbf{N}]$). Suppose $\Delta = y_1 : T_1, \ldots, y_n : T_n$ and let σ_i be strategies and N_i be terms such that $\sigma_i \preccurlyeq^{\Gamma}_{T_i} N_i$ for $i = 1, \ldots, n$. Write $\vec{\sigma_i} \fatsemi [\![M]\!]$ for the composite

$$[\![?]\!] \xrightarrow{\ \langle \mathsf{id}, \sigma_1, \ldots, \sigma_n \rangle\ } [\![?, \Delta]\!] \xrightarrow{\ [\![M]\!]\ } [\![T]\!].$$

Then $\vec{\sigma_i} \fatsemi [\![M]\!] \preccurlyeq^{\Gamma}_{T} M[\vec{N_i}/\vec{y_i}]$.

Proof By induction on the length of the derivation of $?, \Delta \vdash M$. We shall just highlight some of the interesting cases.

- If M is a variable, there are two subcases: either the variable is from $?$ or it is from Δ. In the latter case the result is trivial, while in the former it is just a matter of showing that $[\![? \vdash x]\!] \preccurlyeq^{\Gamma}_{\mathtt{var[N]}} x$, which is simple.

- If M is an abstraction, again there are two cases: either it has type $\mathtt{var}[\mathbf{N}] \Rightarrow \mathtt{exp}[X]$, or some other type. In the latter case the result follows straightforwardly from the inductive hypothesis. Otherwise, let us just consider the case of the type $\mathtt{var}[\mathbf{N}] \Rightarrow \mathtt{com}$. Suppose we have $?, \Delta \vdash \lambda x.M : \mathtt{var}[\mathbf{N}] \Rightarrow \mathtt{com}$, derived from $?, \Delta, x : \mathtt{var}[\mathbf{N}] \vdash M : \mathtt{com}$. There are two parts to check, one corresponding to application, the other to allocation of a new program variable. The first part is proved in the usual way, by considering x as being in the "Δ-part" of the context when applying the inductive hypothesis. For the second part, x must be construed as being in the "$?$-part". The inductive hypothesis gives us

  $$\vec{\sigma_i} \wp [\![M]\!] \preccurlyeq^{\Gamma,x}_{\mathtt{com}} M[\vec{N_i}/\vec{y_i}].$$

  We must show that

  $$\mathtt{cell}\,; \Lambda^{\perp 1}(\vec{\sigma_i} \wp [\![\lambda x.M]\!]) \preccurlyeq^{\Gamma}_{\mathtt{com}} \mathtt{new}\ \lambda x.M[\vec{N_i}/\vec{y_i}],$$

  i.e. that for any $s \in \mathrm{St}(?)$, if $[\![s]\!]\,;(\mathtt{cell}\,; \Lambda^{\perp 1}(\vec{\sigma_i} \wp [\![\lambda x.M]\!])) \neq -$ leaving state s' then $?, s \vdash \mathtt{new}\ \lambda x.M \Downarrow \mathtt{skip}, s'$. But the left hand side of the above is equal to

  $$[\![s]\!]\,;(\vec{\sigma_i} \wp [\![M]\!])$$

  so the result follows from the inductive hypothesis.

- For the case when M is $\mathtt{new}$, it suffices to show that $[\![\mathtt{new}]\!] \preccurlyeq_{(\mathtt{var[N]} \Rightarrow \mathtt{com}) \Rightarrow \mathtt{com}} \mathtt{new}$. Let $e \preccurlyeq_{\mathtt{var[N]} \Rightarrow \mathtt{com}} M$. Then by definition of the logical relation, $\mathtt{cell}\,; \Lambda^{\perp 1}(e) \preccurlyeq_{\mathtt{com}} \mathtt{new}\ M$. But by definition of $[\![\mathtt{new}]\!]$, $\langle [\![\mathtt{new}]\!], e \rangle\,; \mathsf{ev} = \mathtt{cell}\,; \Lambda^{\perp 1}(e)$, so the result follows immediately. □

Proposition 21 (Computational Adequacy) Let P be a program. Then $P\Downarrow \iff [\![P]\!] \neq -$.

Proof The left-to-right implication is given by Proposition 19. For the right-to-left direction, we use the previous Lemma, which tells us that $[\![P]\!] \preccurlyeq_{\mathtt{com}} P$, so that if $[\![P]\!] \neq -$, then $P\Downarrow$ by definition of $\preccurlyeq_{\mathtt{com}}$. □

At last we can prove the inequational soundness result in the usual way.

Theorem 22 (Inequational Soundness) Let $M, N \in \mathsf{Trm}(?, T)$. If $[\![M]\!] \subseteq [\![N]\!]$ then $M \sqsubseteq_{\sim} N$.

Proof Suppose that $[\![M]\!] \subseteq [\![N]\!]$, and that for some $C[-] \in \mathsf{Ctxt}(?, T)$ we have $C[M]\Downarrow$. By the previous Proposition, $[\![C[M]]\!] \neq -$, and by compositionality of the semantics, $[\![C[M]]\!] \subseteq [\![C[N]]\!]$. Therefore $[\![C[N]]\!] \neq -$, and hence by the previous Proposition again, $C[N]\Downarrow$. Therefore $M \sqsubseteq_{\sim} N$. □

Note that this result applies to the model in $\mathcal{C}$. To extend it to the model in $\mathcal{C}/\lesssim$, just observe that Proposition 21 immediately extends to that model. The above proof can therefore be used to show the following.

Theorem 23 (Inequational Soundness in $\mathcal{C}/\lesssim$) Let $M, N \in \mathsf{Trm}(?, T)$. If $[\![M]\!] \lesssim [\![N]\!]$ then $M \sqsubseteq_{\lesssim} N$.

9 Full Abstraction

With the Inequational Soundness and Definability results in place, it is a routine matter to show that $\mathcal{C}/\lesssim$ is a fully abstract model of Idealized Algol.

Theorem 24 (Full Abstraction) Let $M, N \in \mathsf{Trm}(?, T)$. Then

$$[\![M]\!] \lesssim [\![N]\!] \iff M \sqsubseteq_{\lesssim} N.$$

Proof The left-to-right direction is the Inequational Soundness result above. For the converse, suppose that $[\![M]\!] \not\lesssim [\![N]\!]$. Without loss of generality we may assume M and N are closed terms; then by definition of the intrinsic preorder there exists a strategy $\alpha : ![\![T]\!] \multimap \Sigma$ such that $[\![M]\!] \fatsemi \alpha = \top$ and $[\![N]\!] \fatsemi \alpha = \bot$. Clearly α can be chosen to be compact. Since the game Σ is the same as $[\![\mathtt{com}]\!]$, the Definability Theorem implies that there is an IA term $x : T \vdash C[x] : \mathtt{com}$ such that $[\![C[x]]\!] = \alpha$. Therefore $[\![M]\!] \fatsemi \alpha = [\![C[M]]\!]$, and similarly for N. By Computational Adequacy, $C[M]\Downarrow$ and $C[N]\Uparrow$. Therefore $M \not\sqsubseteq_{\lesssim} N$, completing the proof. $\square$

10 The Extensional Model

Our fully abstract model $\mathcal{C}/\lesssim$ involves two ingredients: the intensional category $\mathcal{C}$ and the intrinsic preorder $\lesssim$. In the case of PCF, our understanding of the intrinsic preorder on innocent strategies is limited; the recent result of Loader [12], which implies that it is undecidable even on compact strategies, shows that this is unavoidable. By contrast, we can give an explicit characterization of the intrinsic preorder on knowing strategies, from which the effective presentability of the fully abstract model of Idealized Algol follows easily.

Say that a play in a game is *complete* if it is maximal and all questions have been answered. Note that, for any IA type T, the complete plays in $[\![T]\!]$ are exactly those in which the initial question has been answered. We call a game *simple* in this case. We write comp_A for the set of complete plays in A. If σ is a strategy on A, we write $\mathsf{comp}(\sigma) = \sigma \cap \mathsf{comp}_A$. We write $\mathsf{Pref}(S)$ for the prefix-closure of a set of sequences S.

Theorem 25 (Characterization Theorem) Let σ and τ be strategies on a simple game A in $\mathcal{C}$. Then

$$\begin{aligned} \sigma \lesssim \tau &\iff \mathsf{comp}(\sigma) \subseteq \mathsf{comp}(\tau) \\ &\iff \mathsf{Pref}(\mathsf{comp}(\sigma)) \subseteq \mathsf{Pref}(\mathsf{comp}(\tau)). \end{aligned}$$

Proof Suppose that $s = m_1 \ldots m_n \in \mathsf{comp}(\sigma) \backslash \mathsf{comp}(\tau)$. We define a strategy $\alpha : A \to \Sigma$ which "follows the script of s" (playing as O) as long as P does. If s is completed, it replies in Σ:

$$\begin{array}{ccc} A & \xrightarrow{\alpha} & \Sigma \\ & & q \\ m_1 & & \\ \vdots & & \\ m_n & & \\ & & a \end{array}$$

Note that α is a well-defined (but not in general innocent) strategy. Then by assumption $\sigma ; \alpha = \top$ while $\tau ; \alpha = -$. Thus $\sigma \lesssim \tau \Rightarrow \mathsf{comp}(\sigma) \subseteq \mathsf{comp}(\tau)$.

For the converse, suppose that $\sigma^\dagger ; \alpha = \top$ and $\tau^\dagger ; \alpha = -$. The interaction between σ and α has the form

$$\begin{array}{ccccc} 1 & \xrightarrow{\sigma} & !A & \xrightarrow{\alpha} & \Sigma \\ & & & & q \\ & & m_1 & & \\ & & \vdots & & \\ & & m_n & & \\ & & & & a \end{array}$$

The sequence $s = m_1 \ldots m_n$ can be decomposed into the "threads" $s_1 = s \upharpoonright i_1$, $\ldots$, $s_k = s \upharpoonright i_k$, where $i_1, \ldots, i_k$ are the occurrences of initial moves in s. Each s_i must be a complete play in A by the bracketing condition. Clearly we must have $s_j \notin \tau$ for some j, $1 \leq j \leq n$, since otherwise we would have $\tau^\dagger ; \alpha = \top$.

Finally, since Pref is monotone, $\mathsf{comp}(\sigma) \subseteq \mathsf{comp}(\tau) \Rightarrow \mathsf{Pref}(\mathsf{comp}(\sigma)) \subseteq \mathsf{Pref}(\mathsf{comp}(\tau))$. Since the elements of $\mathsf{comp}(\sigma)$ are maximal plays, they form an anti-chain under the prefix ordering; Pref is order-reflecting on anti-chains, yielding the converse. □

Now given the dI-domain $\mathcal{C}(1, A)$, where A is a simple game, define $p : \mathcal{C}(1, A) \to \mathcal{C}(1, A)$ by

$$p(\sigma) = \mathsf{Pref}(\mathsf{comp}(\sigma)).$$

Proposition 26 The map p is a finitary projection [3] whose image is a dI-domain.

Proof Note that $p(\sigma) = \downarrow(\sigma \cap \mathsf{comp}_A)$. It is easily seen from this description that p has the stated properties. □

Note that it follows from the Characterization Theorem that

$$\mathcal{C}(A,B)/\lesssim \;\cong\; (\{p(\sigma) \mid \sigma \in \mathcal{C}(A,B)\}, \subseteq).$$

The compact elements of $\mathcal{C}(A,B)/\lesssim$ correspond to the finite consistent sets of complete plays, where s consistent with t means that $s \sqcap t$ has even length. It is then clear that, if we start from basic data types which are given as enumerated sets [4], then for each IA type T, $\mathcal{C}(1, [\![T]\!])/\lesssim$ is an effectively presentable domain.

Theorem 27 (Effective Presentability) The fully abstract model of Idealized Algol is effectively presentable.

Example Consider the type $\mathtt{com} \Rightarrow \mathtt{com}$. The interpretation of this type in $\mathcal{C}$ is isomorphic to the lazy natural numbers. That is, for each $k \in \mathbf{N}$, there is a strategy denoted by

$$\lambda c : \mathtt{com}.\, \underbrace{c; \ldots; c}_{k \text{ times}},$$

but also a strategy denoted by

$$\lambda c : \mathtt{com}.\, \underbrace{c; \ldots; c}_{k \text{ times}}; -,$$

and there is a strategy denoted by

$$\mathbf{Y}(\lambda F : \mathtt{com} \Rightarrow \mathtt{com}. \lambda c : \mathtt{com}.\, c; (Fc)).$$

In $\mathcal{C}/\lesssim$ all these partial strategies, which have no complete plays, collapse to $-$, and the interpretation of $\mathtt{com} \Rightarrow \mathtt{com}$ is isomorphic to the flat natural numbers.

11 Related work

There have been two main strands of work addressing the issue of locality of store in programming languages from a semantic point of view. The first, based on the use of functor categories, was pioneered by Reynolds and Oles [22], and has since been considerably refined, notably by O'Hearn and Tennent [21]. The idea is essentially to take a traditional "global state" model and parameterize it with respect to store shapes, to account for the allocation and later deallocation of local variables. Stark has also used functor categories to model ML-style references [31], and similar ideas have led to denotational models of the π-calculus [10, 32]. The second, perhaps computationally more compelling, method has been termed "Object based semantics" by Reddy [26]. In this view, commands, procedures and variables are seen as *objects* or *processes* which interact with one another during the course of a computation. Milner adopted this approach in an operational setting, translating a language with local variable

declarations into the process calculus CCS [16], while Reddy realized the same ideas denotationally, using coherence spaces [26]. Our work can be seen as the next stage in this line of enquiry.

A somewhat different investigation into the behaviour of Algol-like languages has recently been undertaken by Pitts [24], who uses the operational semantics directly to prove certain program equivalences and establish reasoning principles, such as a context lemma. It is to be hoped that our games model can be used for similar purposes, perhaps to strengthen Pitts' context lemma, as has been done for functional languages [13], or to develop a novel logic for reasoning about Algol programs.

As regards full abstraction, the strongest results previously achieved have been by O'Hearn and Reddy [19], for a model combining object-based and functor-category ideas, and by Sieber [30], for a model using logical relations. In both cases, the results are that the model captures definability of first-order functions, and hence full abstraction for closed second-order terms. O'Hearn and Reddy's results are for essentially the same version of Idealized Algol as ours. Sieber's results are for a language without side-effecting expressions, but with "snap-back" and parallel conditional.

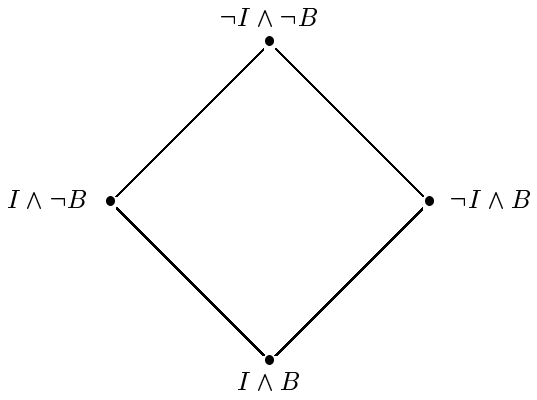
12 Further directions

As we mentioned in the Introduction, Reynolds' original version of IA does not allow side-effects in expressions. To capture this restricted language, we must reflect the distinction between *active* and *passive* types in the model [27, 20]. A passive type P does not allow side-effects to the environment in the course of computing a value of type P. This distinction is also crucial to the Syntactic Control of Interference [27], which is a type discipline for controlling interference and aliasing in imperative programs.

It is possible to refine the model we have presented here to incorporate the active/passive distinction, and hence to get a fully abstract model for Reynolds' original language with passive expressions. This will be described in a sequel to the present paper. From the point of view of game semantics, what is interesting about this refined model is that, instead of a simple innocent/knowing dichotomy, we get a whole spectrum of possibilities, with innocent strategies (when all moves are passive) and knowing strategies (when all moves are active) as the extreme cases.

This model should also prove a good setting for studying the Syntactic Control of Interference from a semantic point of view.

Another point for further investigation is suggested by the following diagram.



Here I denotes innocence and B the bracketing condition. The category $\mathcal{G}_{\mathrm{inn}}$ embodies both constraints on strategies, and very successfully captures pure functional programming [11, 13, 14]. As we have seen in the present paper, the category of knowing (but well-bracketed) strategies captures Idealized Algol. If we conversely retain innocence but weaken the bracketing condition then we get a model of PCF extended with non-local control operators [8, 23]. Thus we begin to develop a semantic taxonomy of constraints on strategies mirroring the presence or absence of various kinds of computational features.

References

[1] S. Abramsky. Axioms for full abstraction and full completeness. Submitted for publication, 1996.

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

[3] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science Volume 3*, pages 1–168. Oxford University Press, 1994.

[4] A. Asperti and G. Longo. *Categories, Types and Structures : An introduction to category theory for the working computer scientist.* Foundations of Computing Series. MIT Press, 1991.

[5] G. Berry, P.-L. Curien, and J.-J. Lévy. Full abstraction for sequential languages: the state of the art. In M. Nivat and J. C. Reynolds, editors, *Algebraic Semantics*, pages 89–132. Cambridge University Press, 1986.

[6] G. Bierman. What is a categorical model of intuitionistic linear logic? In *International Conference on Typed Lambda Calculi and Applications*. Springer-Verlag, 1995. Lecture Notes in Computer Science.

[7] F. Borceux. *Handbook of Categorical Algebra, volume 1*. Cambridge University Press, 1994.

[8] R. Cartwright, P.-L. Curien, and M. Felleisen. Fully abstract semantics for observably sequential languages. *Information and Computation*, 111(1):297–401, 1994.

[9] R. Crole. *Categories for Types*. Cambridge University Press, 1994.

[10] M. P. Fiore, E. Moggi, and D. Sangiorgi. A fully abstract model for the π-calculus. In *11th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1996.

[11] M. Hyland and C.H. L. Ong. On full abstraction for PCF. Submitted for publication, 1996.

[12] R. Loader. Finitary PCF is undecidable. Unpublished manuscript, available from `http://info.ox.ac.uk/~loader/`, 1996.

[13] G. McCusker. *Games and Full Abstraction for a functional metalanguage with recursive types*. PhD thesis, Imperial College, University of London, 1996. to appear.

[14] G. McCusker. Games and full abstraction for FPC. In *11th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1996.

[15] R. Milner. Processes: a mathematical model of computing agents. In *Logic Colloquium '73*, pages 157–173. North Holland, 1975.

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

[17] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–142, 1992.

[18] P. W. O'Hearn and U. Reddy. Objects, interference and the Yoneda embedding. In M. Main and S. Brookes, editors, *Mathematical Foundations of Programming Semantics: Proceedings of 11th International Conference*, Electronic Notes in Theoretical Computer Science. Elsevier Science Publishers B.V., 1995.

[19] P. W. O'Hearn and U. Reddy. Objects, interference and the Yoneda embedding. In *MFPS XI, Conference on Mathematical Foundations of Program Semantics*, volume 1 of *Electronic Notes in Theoretical Computer Science*. Elsevier, March 1995.

[20] P. W. O'Hearn, M. Takayama, A. J. Power, and R. D. Tennent. Syntactic control of interference revisited. In *MFPS XI, Conference on Mathematical Foundations of Program Semantics*, volume 1 of *Electronic Notes in Theoretical Computer Science*. Elsevier, March 1995.

[21] P. W. O'Hearn and R. D. Tennent. Parametricity and local variables. *Journal of the ACM*, 42(3):658–709, May 1995.

[22] F. J. Oles. Type categories, functor categories and block structure. In M. Nivat and J. C. Reynolds, editors, *Algebraic Semantics*. Cambridge University Press, 1985.

[23] C.-H. L. Ong and C. A. Stewart. A Curry-Howard foundation for functional computation with control (summary). Preprint, July 1996.

[24] A. M. Pitts. Reasoning about local variables with operationally-based logical relations. In *11th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1996.

[25] G. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[26] U. Reddy. Global state considered unncessary: an object-based semantics for Algol. *Lisp and Functional Programming*, 1996.

[27] J. C. Reynolds. Syntactic control of interference. In *Conf. Record 5th ACM Symposium on Principles of Programming Languages*, pages 39–46, 1978.

[28] J. C. Reynolds. The essence of Algol. In J. W. de Bakker and J. C. van Vliet, editors, *Algorithmic Languages*, pages 345–372. North Holland, 1981.

[29] R. A. G. Seely. Linear logic, $*$-autonomous categories and cofree coalgebras. In *Category theory, computer science and logic*. American Mathematical Society, 1987.

[30] K. Sieber. Full Abstraction for the Second Order Subset of an ALGOL-like Language. Technischer Bericht A/04/95, FB14, Universität des Saarlandes, 1995.

[31] I. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, December 1994.

[32] I. Stark. A fully abstract domain model for the π-calculus. In *11th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1996.

[33] R. D. Tennent. Denotational semantics. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 169–322. Oxford University Press, 1994.

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