Full Abstraction for Idealized Algol with Passive Expressions

Samson Abramsky
University of Edinburgh
Department of Computer Science
James Clerk Maxwell Building
Edinburgh EH9 3JZ
Scotland
samson@dcs.ed.ac.uk

Guy McCusker
St John's College
Oxford OX1 3JP,
England
mccusker@comlab.ox.ac.uk

Abstract

A fully abstract games model of Reynolds' Idealized Algol is described. The model gives a semantic account of the distinction between *active types*, such as commands, which admit side-effecting behaviour, and *passive types*, such as expressions, which do not.

Keywords: Algol-like languages, game semantics, full abstraction.

1 Introduction

Our aim in this paper is to give the first syntax-independent construction of a fully abstract model for Idealized Algol. John Reynolds proposed Idealized Algol as capturing the essence of Algol 60 [32]; it is an elegant synthesis of the features of a simple block-structured imperative programming language with those of higher-order functional programming. As such it has become the focus of a body of research aimed at understanding the—potentially very subtle—behaviour of programs which combine the use of local variables with procedures; a recently published collection [26] contains reprints of many of these papers. The intricacies arising from the combination of computational features in the language mean that the search for a good, syntax-free description of the fully abstract model of Idealized Algol raises significant problems over and above those associated with the full abstraction problem for PCF [20, 29].

In a previous paper [5, 6], we gave the first fully abstract model of Idealized Algol with *active*, i.e. possibly side-effecting, expressions. It is significantly harder to capture the "pure" version of Idealized Algol originally proposed by Reynolds, in which a sharp distinction is drawn between "active" types such as commands, which are allowed to have side-effects, and "passive" types such as expressions, which are not. In the present paper we will give a fully abstract model for the pure language. This will be done by reflecting the distinction between active and passive types in our model, which is based on game semantics [1, 11, 16, 17, 4, 3, 9, 22]. It is further evidence of the flexibility and tractability of game semantics that it is possible to capture such a subtle behavioural condition in a compositional fashion, preserving the good properties of categories of games.

2 Idealized Algol

Idealized Algol is an applied simply-typed λ-calculus, with a suitable stock of constants to express basic imperative features. For notational simplicity, we shall consider the version of Idealized Algol with a single basic data type $\mathbb{N}$ of natural numbers. The base types of Idealized Algol are then

$$B ::= \texttt{exp}[\mathbb{N}] \mid \texttt{var}[\mathbb{N}] \mid \texttt{com}$$

i.e. natural-number-valued expressions, assignable program variables in which natural numbers can be stored, and commands. We shall generally write simply `exp` and `var` rather than $\texttt{exp}[\mathbb{N}]$ and $\texttt{var}[\mathbb{N}]$. The types of Idealized Algol are given by

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

The constants of the language are as follows:

- Recursion combinators $\mathbf{Y}_T : (T \Rightarrow T) \Rightarrow T$ for each type T;
- conditionals $\texttt{if0}_B : \texttt{exp} \Rightarrow B \Rightarrow B \Rightarrow B$ for each base type B. We shall also make use of a family of conditional operators

  $$\texttt{case}_m : \texttt{exp} \Rightarrow \underbrace{B \Rightarrow \cdots \Rightarrow B}_{m} \Rightarrow B$$

  for each natural number m and base type B. The term $\texttt{case}_m M\ N_1 \dots N_m$ first evaluates M to some number n, and then evaluates the term N_n provided $1 \leq n \leq m$. Note that the constants $\texttt{case}_m$ can be defined up to observation equivalence in terms of the ordinary conditional. The reason for adding them to the language explicitly is to ease the proof of our full abstraction result;

- arithmetic operations

$$\begin{aligned} \mathbf{n} :&\ \ \mathtt{exp}\ \ (\text{for } n \in \mathbb{N}) \\ \mathtt{succ}, \mathtt{pred} :&\ \ \mathtt{exp} \Rightarrow \mathtt{exp} \end{aligned}$$

- command sequencing

$$\begin{aligned} \mathtt{seq} :&\ \ \mathtt{com} \Rightarrow \mathtt{com} \Rightarrow \mathtt{com} \\ \mathtt{skip} :&\ \ \mathtt{com} \end{aligned}$$

- variable dereferencing and assignment

$$\begin{aligned} \mathtt{deref} :&\ \ \mathtt{var} \Rightarrow \mathtt{exp} \\ \mathtt{assign} :&\ \ \mathtt{var} \Rightarrow \mathtt{exp} \Rightarrow \mathtt{com} \end{aligned}$$

- variable constructor

$$\mathtt{mkvar} : \mathtt{exp} \Rightarrow (\mathtt{exp} \Rightarrow \mathtt{com}) \Rightarrow \mathtt{var}$$

- block structure

$$\mathtt{new} : (\mathtt{var} \Rightarrow \mathtt{com}) \Rightarrow \mathtt{com}.$$

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

Examples

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

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

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

$$\begin{aligned} x : \mathtt{var} \vdash \mathbf{Y}_{\mathtt{com}}(\lambda c : \mathtt{com}.\ \ &\mathtt{if0}_{\mathtt{com}}\ (\mathtt{deref}\ x) \\ &\mathtt{skip} \\ &(\mathtt{seq}\ (\mathtt{assign}\ x\ (\mathtt{pred}(\mathtt{deref}\ x)))\ c)) : \mathtt{com}. \end{aligned}$$

3. The code

$$\begin{aligned} &\mathtt{new}\ x\ \mathtt{in} \\ &(x := 0; \\ &p(x := x + 1); \\ &\mathtt{if}\ x = 0\ \mathtt{then}\ \Omega\ \mathtt{else}\ \mathtt{skip} \\ &) \end{aligned}$$

translates into

$$\begin{aligned} p : \mathtt{com} \Rightarrow \mathtt{com} \vdash\ \ &\mathtt{new}(\lambda x : \mathtt{var}. \\ &\quad \mathtt{seq}\ \ (\mathtt{assign}\ x\ 0) \\ &\qquad (\mathtt{seq}\ \ (p(\mathtt{assign}\ x\ (\mathtt{succ}(\mathtt{deref}\ x)))) \\ &\qquad\quad (\mathtt{if0}_{\mathtt{com}}\ (\mathtt{deref}\ x))\ \Omega\ \mathtt{skip}))) : \mathtt{com}. \end{aligned}$$

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 [37], as can classes, objects and methods [31]. 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.

A remark about `mkvar` It is well-known that Algol-like languages may contain so-called *bad variables*, that is, entities of type `var` which do not have the behvaiour of storage cells. For example, consider the behaviour of

$$x : \mathtt{var}, y : \mathtt{var} \vdash \mathtt{if0}_{\mathtt{var}}\ (\mathtt{deref}\ x)\ x\ y$$

over several uses.

Reynolds [32] analyses such variables in an object-oriented fashion, treating them as objects with a "read method" of type `exp` and a "write method" of type $\mathtt{exp} \Rightarrow \mathtt{com}$. The `mkvar` construct allows such an object to be constructed from its methods. For example,

$$x : \mathtt{var} \vdash \mathtt{mkvar}\ (\mathtt{deref}\ x)\ (\lambda n : \mathtt{exp}.\ \mathtt{assign}\, x\ (\mathtt{succ}\ (\mathtt{deref}\ x)))$$

gives a variable object whose write method always increments x by one, regardless of the value passed in to be written.

3 Active vs Passive Types

The simplicity of the syntactic presentation of Idealized Algol is misleading: the operational intentions behind the imperative features impose non-trivial constraints on a model if they are to be captured accurately. In particular, notice that the set of constants has been carefully chosen so that expression evaluation cannot have side-effects. This means, for example, that in any Idealized Algol term-in-context

$$\Gamma, x : \mathtt{var} \vdash M : \mathtt{exp}$$

the term M cannot perform any assignments to x. On the other hand, if `exp` is replaced by `com`, this clearly no longer holds, e.g.

$$\Gamma, x : \mathtt{var} \vdash x := \mathbf{1} : \mathtt{com}.$$

(In contrast, the language considered in [5, 6] has explicit constants to allow expressions to have side-effects. We shall refer to this extended language as Idealized Algol *with active expressions* or IAa for short; the pure language will be known as IA.)

Say that a type T is ***passive*** if no term $\Gamma \vdash M : T$ can have side-effects on the variables in Γ; otherwise T is ***active***. The basic example of a passive type is

$\mathtt{exp}$; and the passive types form an *exponential ideal*, i.e. if P is passive then so is $T \Rightarrow P$ for any type T (since for example the activity of $\Gamma \vdash \lambda x.\, M : T \Rightarrow P$ would contradict the assumed passivity of $\Gamma, x : T \vdash M : P$).

The basic example of an active type is `com`. What about `var`? Following Reynolds [32], we can identify `var` with the product of its two "access methods"; reading (`deref`) and writing (`assign`):

$$\mathtt{var} = \mathtt{exp} \times (\mathtt{exp} \Rightarrow \mathtt{com}).$$

Under this identification, `deref` and `assign` are simply first and second projection respectively. Dually, the constant `mkvar` is essentially the pairing operation for this type. (The reader should be warned that this is not *quite* the interpretation we shall give to `var` in our model.) Then we see that `var` combines both aspects, since reading is passive, while writing is active.

4 Models of Idealized Algol

What should a model of 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 $\mathbf{C}$ [14]. To accommodate the recursion in the language, we can ask for $\mathbf{C}$ to be **cpo**-enriched [13], or more minimally to be rational [1].

To capture the distinction between active and passive types, we assume that $\mathbf{C}$ has a full subcategory $\mathbf{P}$ of "passive objects", which forms an exponential ideal. For modelling `exp`, we require a passive object $\mathbf{N}$ such that there is an order-isomorphism

$$\mathbf{C}(1, \mathbf{N}) \cong \mathbb{N}_\perp$$

between the "points" of $\mathbf{N}$ and the usual flat domain of natural numbers. We will use this to interpret the numerals $\mathbf{n}$ as morphisms $[\![\mathbf{n}]\!] : 1 \to \mathbf{N}$ (for $n \in \mathbb{N}$). We also assume there are morphisms $s, p : \mathbf{N} \to \mathbf{N}$ which track the action of `succ` and `pred` on $\mathbb{N}_\perp$. The conditionals are to be interpreted similarly. All of this can be stated more succinctly by saying that $\mathbf{C}$ has *standard datatypes*: see [2].

We further assume an active object `com` of $\mathbf{C}$, and morphisms $[\![\mathtt{skip}]\!] : 1 \to \mathtt{com}$, $[\![\mathtt{seq}]\!] : 1 \to (\mathtt{com} \Rightarrow \mathtt{com} \Rightarrow \mathtt{com})$. One should understand `com` as a unit type, i.e. a one-element data-type, with the proviso that it is an *active* unit type. Thus we expect the points of `com` to be in order-isomorphism with the Sierpinski poset $1_\perp$. We think of the single non-bottom element of `com` (the point $[\![\mathtt{skip}]\!]$) as signaling successful termination; $[\![\mathtt{seq}]\!]$ forces each of its arguments to be evaluated in turn, thus having exactly the intended effect of sequential composition. (We can also think of $[\![\mathtt{seq}]\!]$ as a "unary conditional".)

Finally, as already indicated, we think of `var` as a product

$$\mathtt{var} = \mathtt{exp} \times \mathtt{com}^{\mathbb{N}}$$

where $\mathtt{com}^{\mathbb{N}}$ is the product of countably many copies of `com`. We use $\mathtt{com}^{\mathbb{N}}$ rather than $\mathtt{exp} \Rightarrow \mathtt{com}$ since assignment is intended to be a *strict* operation, and assume a retraction

$$c : \mathtt{com}^{\mathbb{N}} \lhd \mathtt{exp} \Rightarrow \mathtt{com} : r$$

so that we can define

$$\begin{array}{rcl} [\![\mathtt{deref}]\!] & = & \mathtt{fst} \\ [\![\mathtt{assign}]\!] & = & c \circ \mathtt{snd} \\ [\![\mathtt{mkvar}]\!]ef & = & \langle e, r \circ f \rangle. \end{array}$$

5 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 $\mathsf{IA} - \{\mathtt{new}\}$. We can now formulate the following thesis:

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

At first sight, this seems nonsensical, since the usual "basic imperative language" [38], which does not include block structure, can be represented in $\mathsf{IA} - \{\mathtt{new}\}$, as shown in Section 2. 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 `var` 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 $\mathsf{IA} - \{\mathtt{new}\}$ 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 8, a small modification of the categories of games used to give the first syntax-independent constructions of fully abstract models for PCF [1, 11, 22], when used to give models for $\mathsf{IA} - \{\mathtt{new}\}$ in the way we have described, again yield fully abstract models. Indeed, for IAa, exactly the same categories can be used; the refinement we introduce here is necessary to capture the distinction between active and passive types. 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 [2]. This latter point means that *any* model of the axioms in [2] yields a fully abstract model of $\mathsf{IAa} - \{\texttt{new}\}$; refined axioms can be given which describe fully abstract models of pure $\mathsf{IA} - \{\texttt{new}\}$.

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

6 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 [27, 37] 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 [23, 30].

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 [19, 21, 30]. We want to understand $\texttt{new}\,x\,\texttt{in}\,C$ as binding the free identifier x of type `var` 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

$$\mathtt{cell} : 1 \longrightarrow \mathtt{var}$$

such that, if $f : \mathtt{var} \longrightarrow \mathtt{com}$, $\mathtt{new}(f)$ is given by the composition

$$\mathtt{new}(f) = 1 \xrightarrow{\mathtt{cell}} \mathtt{var} \xrightarrow{f} \mathtt{com}.$$

The idea is that `cell` gives the "behaviour" of our storage cell process. However, recalling that

$$\mathtt{var} = \mathtt{exp} \times \mathtt{com}^{\mathbb{N}}$$

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 [7, 33]. 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 `new` is:

$$\mathtt{new} : !(\,!\mathtt{var} \multimap \mathtt{com}) \multimap \mathtt{com}.$$

Now suppose that we have a morphism

$$\mathtt{cell} : I \longrightarrow !\mathtt{var}.$$

Then we can define `new` as the composition

$$
\begin{tikzcd}
!(\,!\mathtt{var} \multimap \mathtt{com}) \arrow[d, "\mathsf{der}"] \\
!\mathtt{var} \multimap \mathtt{com} \arrow[d, "\cong"] \\
(\,!\mathtt{var} \multimap \mathtt{com}) \otimes I \arrow[d, "\mathsf{id} \otimes \mathtt{cell}"] \\
(\,!\mathtt{var} \multimap \mathtt{com}) \otimes !\mathtt{var} \arrow[d, "\mathsf{Ap}"] \\
\mathtt{com}.
\end{tikzcd}
$$

(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 $\mathtt{cell} : \mathtt{var}$ in the language, and tried to define

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

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

$$I \xrightarrow{\mathtt{cell}_0^\dagger} !\mathtt{var} \xrightarrow{f} \mathtt{com}$$

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

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

But the promotion will behave "uniformly" in each copy of `var`, 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 $\mathtt{cell}_0 : I \to \mathtt{var}$.

Provided that we can define a suitable morphism

$$\mathtt{cell} : I \longrightarrow !\mathtt{var}$$

which does capture the behaviour of a storage cell object, then we have completed our semantics of Idealized Algol. In Section 9 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 $!\mathtt{var}$, i.e. the contraction rule, so that the interpretation of a block $\mathtt{new}\ x\ \mathtt{in}\ C$ looks like:

$$
\begin{tikzcd}
 & & !\mathtt{var} & \\
 & & \otimes & \\
I \arrow[r, "\mathtt{cell}"] & !\mathtt{var} \arrow[uur] \arrow[ddr] & \vdots \arrow[r, "[\![C]\!]"] & \mathtt{com} \\
 & & \otimes & \\
 & & !\mathtt{var} &
\end{tikzcd}
$$

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.

7 A refined games model

We describe a cartesian closed category of games, $\mathcal{C}_{a/p}$, built over a linear category $\mathcal{G}_{a/p}$, which refines those presented in [17, 16, 5, 6]; in fact, both the category of innocent strategies used to model functional languages and the category of history-sensitive strategies, which captures IAa, are contained as full subcategories of $\mathcal{C}_{a/p}$. This hybrid nature allows us to model precisely the combination of functional and imperative behaviour present in pure IA.

The refinement is quite simple: moves of our games are classified as either *active* or *passive*. The idea is that active moves make a change to the state, having a persistent effect, while passive moves do not. Strategies are constrained to play innocently (i.e. functionally) with respect to passive moves but may exhibit arbitrary history-sensitive behaviour with respect to active moves. In our model of IA, the only active moves are those pertaining to the assignment operation. Assignments change the state for good; all other actions merely make passive use of the store.

7.1 Games

As indicated above, the games to be introduced below are a reasonably straightforward refinement of those previously used in [17, 16, 5, 6]. In the development which follows, we will place special emphasis on the new features introduced by the active/passive distinction, and will only give those proofs which depend on the new features. For a full account of the simpler underlying setup, see [17].

Definition An ***a/p-arena*** A is specified by a structure $\langle M_A, \lambda_A, \vdash_A \rangle$ where

- M_A is a set of ***moves***;
- $\lambda_A : M_A \to \{\mathsf{O}, \mathsf{P}\} \times \{\mathsf{Q}, \mathsf{A}\} \times \{\mathsf{act}, \mathsf{pass}\}$ is a ***labelling*** function which indicates whether a move is by Opponent (O) or Player (P), whether it is a question (Q) or an answer (A), and whether it is active (act) or passive (pass). We refer to the three components of λ_A as λ_A^{OP}, λ_A^{QA} and $\lambda_A^{a/p}$ and write $\overline{\lambda}_A$ for the function λ_A with the O/P part reversed, so that

$$\overline{\lambda}_A(m) = (\mathsf{O}, \mathsf{Q}, \mathsf{pass}) \iff \lambda_A(m) = (\mathsf{P}, \mathsf{Q}, \mathsf{pass})$$

  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) $\star \vdash_A m \Rightarrow \lambda_A^{\mathsf{OP}}(m) = \mathsf{O} \wedge \lambda_A^{\mathsf{QA}}(m) = \mathsf{Q} \wedge [n \vdash_A m \iff n = \star]$;

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

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

  (e4) $m \vdash_A n \wedge n$ active $\Rightarrow m$ active or $m = \star$.

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. Condition (e4) was not present in previous definitions of games. It ensures that active moves can only be enabled by other active moves, thus constraining when active play can begin.

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. We use $\sqsubseteq$ for the prefix ordering on sequences. If n is an element of a sequence s, then $s_{\leq n}$ denotes the prefix of s up to and including n, while $s_{<n}$ is the same but excludes n.

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.

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

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 $s = s_1 m n s_2$ 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.

(w3) The ***activity condition***. Say that a position s is ***active*** if there is no pending question at s, or if the pending question is labelled as active. A position is ***passive*** otherwise. The activity condition is satisfied by s if for every prefix ta of s with a active, the position t is active.

Conditions (w1) and (w2) above are familiar in game semantics. Condition (w3) is new; it further constrains when active moves can be played. In particular, if there is a passive question pending, the whole discussion is deemed passive, so only passive moves can be played until the question is answered. In our model of IA, this is the key condition which prevents expressions, which are passive, from having active side-effects.

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.

The following lemma is useful in analysing legal positions of a/p-arenas.

Lemma 1 Let $s \cdot m$ be an active position, and let the justifier of m in s be n. Then $s_{<n}$ is active.

Proof The sequence s has the form

$$s_{<n} \cdot n \dots m.$$

If m is an answer, it answers n so the pending question of $s \cdot m$ is the same as that of $s_{<n}$. Hence $s_{<n}$ is active since $s \cdot m$ is. Otherwise, m is a question so is itself active (lest $s \cdot m$ be a passive position). Therefore its justifier n is active, so by the activity condition, $s_{<n}$ is active. $\square$

Corollary 2 Let $s \cdot m$ be an active position, with m an O-move. For every O-move n in $\ulcorner s \cdot m \urcorner$, $s_{\leq n}$ is active.

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 ***a/p-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 a/p-arena.
- P_A is a non-empty, prefix-closed subset of L_A, called the ***valid positions***, and satisfying

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

7.1.1 A/P-views

Given a valid position $s \in P_A$, the P-view $\ulcorner s \urcorner$ is a subsequence of s in which each pair of moves

$$\cdots m \cdot n \cdots$$

in s, with m an O-move, is either included or excluded in its entirety. Corresponding to the intuition that active moves are "persistent", we introduce the notion of *a/p-view* which takes activity into account.

Definition The ***a/p-view*** $\widetilde{s}$ of a legal position s is the subsequence of s consisting of those pairs $m \cdot n$ as above such that either $m \cdot n$ appears in $\ulcorner s \urcorner$ or $s_{\leq m}$ is an active position.

Lemma 3 For a legal position s, $\widetilde{s}$ is a justified sequence.

Proof Let m be a move in $\widetilde{s}$. We must show that the justifier of m also appears in $\widetilde{s}$. If m is in $\widetilde{s}$ because it appears in $\ulcorner s \urcorner$, this follows easily. Otherwise, if m is an O-move, then $s_{\leq m}$ is an active position. Let the justifier of m in s be n. By Lemma 1, $s_{<n}$ is an active position, so both n and its predecessor appear in $\widetilde{s}$. If m is a P-move then $s_{<m}$ is an active position and the justifier of m appears in $\ulcorner s_{<m} \urcorner$, by the visibility condition. But by Corollary 2, this move must occur at an active position, and hence must appear in $\widetilde{s}$. □

Remark Later, we will need to consider an extended notion of a/p-view for sequences of moves which are not necessarily legal positions. First note that the definition of P-view makes sense for any sequence of moves, some with justification pointers, some without: if an O-move without a pointer is encountered, it is treated as an initial move; other O-moves are treated as usual. The notion of pending question also makes sense for arbitrary sequences. Say that a sequence s ending in an O-move is active if the pending question of $\ulcorner s \urcorner$ is active. We can now extend the definition of a/p-view above to arbitrary sequences of moves and justification pointers, provided the players alternate. Note also that this extended definition coincides with the original one for legal positions, because the pending question of $\ulcorner s \urcorner$ is the same as that of s for sequences satisfying the bracketing condition.

7.1.2 Multiplicatives

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

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

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

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.

Note the additional constraint on the definition of $A \multimap B$: where a new enabling is introduced between an initial move of B and one of A, it must be ensured that an active move is only enabled by another active move.

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

7.1.3 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.

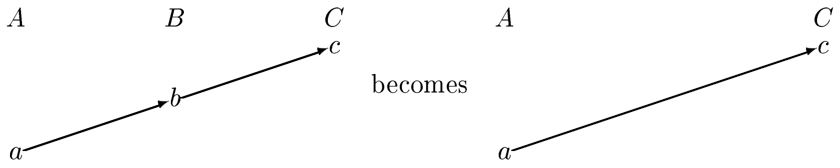
7.1.4 Composition

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

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

Proposition 4 Composition is well-defined and associative.

Proof The definition of composition here is identical to that in [17], so we need only check that the extra conditions required for well-definedness in the presence of the active/passive classification of moves are met. This amounts to ensuring that for any $u \in \sigma \parallel \tau$, the string $u \upharpoonright A, C$ satisfies the activity condition. Suppose we have prefix $s \cdot m$ of $u \upharpoonright A, C$ with m active, but that s is a passive position. There must be a passive pending question q in s; suppose this is in C. Therefore $u_{<m} \upharpoonright B, C$ contains the same unanswered question q, so it is a passive position. If m is in C, the activity condition is contravened in B, C. If m is in A, there must be an odd number of moves in B between q and m. By the bracketing condition, these must contain more questions than answers, and must therefore leave at least one question unanswered. By the activity condition in B, C, all these moves are passive. Therefore there is an unanswered passive question in B, so m contravenes the activity condition in A, B. $\square$

7.1.5 Innocent strategies and a/p-strategies

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

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

where the justification pointer from b in tab points to the same move of $\ulcorner ta \urcorner = \ulcorner sa \urcorner$ as does that from b in sab.

In other words, how σ plays depends only on the current P-view. Similarly we may define ***a/p-strategies*** to be those satisfying

$$sab \in \sigma \wedge t \in \sigma \wedge ta \in P_A \wedge \widetilde{ta} = \widetilde{sa} \Rightarrow tab \in \sigma.$$

So a/p-strategies depend only on the current view and all those moves played at active positions. There is now an informative hierarchy of strategies: any history-free strategy (in the sense of [1]) is innocent; any innocent strategy is an a/p-strategy; and any a/p-strategy is a history-sensitive strategy.

From earlier work [16, 17, 11, 22, 5, 6], we know that the innocent strategies correspond precisely to functional programs, while arbitrary history-sensitive strategies correspond to programs in IAa. The result of this paper will show that a/p-strategies correspond exactly to programs of IA.

7.1.6 The categories $\mathcal{G}_{a/p}$ and $\mathcal{G}_{\text{inn}}$

The objects of $\mathcal{G}_{a/p}$ and its lluf (i.e. same objects, fewer arrows: the dual of "full"!) subcategory $\mathcal{G}_{\text{inn}}$ are a/p-games. A morphism from A to B in $\mathcal{G}$ is an a/p-strategy $\sigma : A \multimap B$. The lluf subcategory $\mathcal{G}_{\text{inn}}$ has as morphisms only the innocent strategies. Composition and identities are as described above.

The main result of this section is the following.

Proposition 5 Let $\sigma : A \multimap B$ and $\tau : B \multimap C$ be strategies. If σ and τ are a/p-strategies, then so is $\sigma ; \tau$. If σ and τ are both innocent strategies then $\sigma ; \tau$ is again innocent.

In order to prove this Proposition, we will make use of the following definitions and lemmas.

Definition Let $u \in \mathsf{int}(A, B, C)$.

- The sequence u consists of segments $m \ldots n$ where m is an O-move in $u \upharpoonright A, C$, n is a P-move in $u \upharpoonright A, C$ and all the intervening moves are in B (there need not be any), with a possibly incomplete segment at the end. The ***core*** $\overline{u}$ of u is the subsequence consisting of all those segments $m \ldots n$ such that the pair $m \cdot n$ appears in $\widetilde{u \upharpoonright A, C}$, including the final, possibly incomplete segment.

- A ***generalized* O*-move*** of u is a move m such that either m is an O-move of $u \upharpoonright A, C$ or m is a B-move. Thus m is either an O-move in $u \upharpoonright A, B$ or in $u \upharpoonright B, C$. In the former case, we say the ***component*** of m in u is A, B; in the latter, the component is B, C. An arbitrary component will be written as X, with Y implicitly denoting the other component. The component of a generalized O-move m will sometimes be denoted by X_m.

Lemma 6 Let $u = u_1 \cdot q \cdot u_2$ be an interaction sequence with q a passive question unanswered in u_2. For each m in $q \cdot u_2$, if m is a generalized O-move with component X, the sequence $u_{\leq m} \upharpoonright X$ is passive. Otherwise, if m is in A then $u_{\leq m} \upharpoonright A, B$ is passive and if m is in C, $u_{\leq m} \upharpoonright B, C$ is passive.

Proof By induction on $|u_{\leq m}| - |u_1 \cdot q|$. The base case, when m is q, is trivial. For the inductive step, if m is in A or C, the preceding move m^- has the same component, so $u_{\leq m^-} \upharpoonright X$ is passive by the inductive hypothesis. Hence m is a passive move, so if m is a question the result holds. If m is an answer, it answers some question n occurring in u_2 by the bracketing condition. It is easy to show that the move n^- immediately before n is in component X, so by the inductive hypothesis $u_{\leq n^-} \upharpoonright X$ is passive. Hence $u_{\leq m} \upharpoonright X$ is also passive. If m is in B, justified by n, again the move n^- has component X, so the same argument applies. $\square$

Corollary 7 If m is a generalized O-move of u such that $u_{\leq m} \upharpoonright X$ is active, then m appears in $\overline{u}$.

Proof Suppose m does not appear in $\overline{u}$. Then m is contained in one of the segments $n \ldots n'$ of u excluded from $\overline{u}$. Therefore $u_{\leq n}$ contains an unanswered passive question, which remains unanswered at m. By the above Lemma, $u_{\leq m} \upharpoonright X$ is passive. $\square$

Lemma 8 Let $u \in \mathsf{int}(A, B, C)$ and let m be a generalized O-move with component X. If m is not initial in X, write n for its justifier and n^- for the predecessor of n in u.

1. If m is not initial in X, n^- is a generalized O-move with component X.
2. If m appears in $\overline{u}$ and m is not initial in X, both n and n^- appear in $\overline{u}$.
3. If m appears in $\overline{u}$ then $\ulcorner \overline{u}_{\leq m} \upharpoonright X \urcorner = \ulcorner u_{\leq m} \upharpoonright X \urcorner$. (Note that on the left hand side we may need the extended notion of view given above.)

Proof Part (1) is proved by a straightforward case analysis. We prove parts (2) and (3) simultaneously by induction on the length of $u_{\leq m}$. The case when m is initial is trivial and encompasses the base case. If m is not initial, suppose m appears in $\overline{u}$. If m is in A or C, there are two possibilities. Either m is in $\ulcorner u \upharpoonright A, C \urcorner$, in which case so is n by definition of view, and it is then easy to see that both n and n^- appear in $\overline{u}$, or $u_{\leq m} \upharpoonright A, C$ is active. Let n' be the predecessor of n in $u \upharpoonright A, C$. By Lemma 1, $u_{\leq n'} \upharpoonright A, C$ is active, so the pair $n' \cdot n$ appears in $\overset{\frown}{u \upharpoonright A, C}$, so the segment $n' \ldots n$, incorporating both n and n^-, appears in $\overline{u}$. Suppose now m is in B. Since m is an O-move in component X, it is a P-move in the other component Y and its predecessor m^- is an O-move in Y. It is clear that m^- appears in $\overline{u}$, so by the inductive hypothesis, $\ulcorner \overline{u}_{\leq m^-} \upharpoonright Y \urcorner = \ulcorner u_{\leq m} \upharpoonright Y \urcorner$. By the visibility condition the justifier n of m appears in this view and hence in $\overline{u}$. Since n is in B, n^- also appears in $\overline{u}$. This proves part (2). For part (3) we calculate

$$
\begin{array}{rcll}
\ulcorner u_{\leq m} \upharpoonright X \urcorner & = & \ulcorner u_{\leq n^-} \upharpoonright X \urcorner \cdot n \cdot m & \\
 & = & \ulcorner \overline{u}_{\leq n^-} \upharpoonright X \urcorner \cdot n \cdot m & \text{by IH and (1)} \\
 & = & \ulcorner \overline{u}_{\leq m} \upharpoonright X \urcorner. &
\end{array}
$$
$\square$

The above lemma is the key to proving that innocent strategies compose. We now extend it to handle the case of a/p-strategies.

Lemma 9 Let $u \in \mathsf{int}(A, B, C)$ and let m be a generalized O-move of u with component X such that m appears in $\overline{u}$. Then $\widetilde{u_{\leq m}} \restriction X = \widetilde{\overline{u}_{\leq m}} \restriction X$. (Here we are using the extended notion of a/p-view on the right hand side.)

Proof By Lemma 8(3), we know that $\ulcorner u_{\leq m} \restriction X \urcorner = \ulcorner \overline{u}_{\leq m} \restriction X \urcorner$, so we need only show that the O-moves n of $u_{\leq m} \restriction X$ such that $u_{\leq n} \restriction X$ is active are the same as those of $\overline{u}_{\leq m}$. But by Lemma 8(3) again, for any O-move n of $\overline{u}_{\leq m} \restriction X$, the P-view of $\overline{u}_{\leq n} \restriction X$ is the same as that of $u_{\leq n} \restriction X$, so the former is active if and only if the latter is. It just remains to show that if $u_{\leq n} \restriction X$ is active for some move n, then n appears in $\overline{u}$; but this follows from Corollary 7. $\square$

We now give the proof that composing a/p-strategies gives an a/p-strategy, omitting the (simpler) case of innocent strategies.

Proof Let $\sigma : A \multimap B$ and $\tau : B \multimap C$ be a/p-strategies, and suppose $sab, t \in \sigma ; \tau$, $ta \in P_{A \multimap C}$ and $\widetilde{sa} = \widetilde{ta}$. We must show that $tab \in \sigma ; \tau$. By the definition of composition, there exists $v \in \sigma \parallel \tau$ such that $va \restriction A, C = ta$, and similarly $u \cdot a \ldots b \in \sigma \parallel \tau$ which restricts to give sab.

Let $n \cdot n'$ be a pair of moves appearing in $\widetilde{sa} = \widetilde{ta}$, so that both $\overline{ua}$ and $\overline{va}$ contain a segment $n \ldots n'$, with all the intervening moves in B. Suppose that $\overline{ua}_{\leq n} = \overline{va}_{\leq n}$. We shall show that for each move m in the segment $n \ldots n'$ of ua, m appears in the segment of va and $\overline{ua}_{\leq m} = \overline{va}_{\leq m}$. For the case of the move n, this is trivial. Suppose now that the statement holds for a move m. Since σ and τ are a/p-strategies, the move immediately after m in ua is determined by $\widetilde{ua_{\leq m}} \restriction X_m$. By Lemma 9, this equals $\widetilde{\overline{ua}_{\leq m}} \restriction X_m = \widetilde{\overline{va}_{\leq m}} \restriction X_m$. By Lemma 9 again, this is equal to $\widetilde{va_{\leq m}} \restriction X_m$, which determines the next move of va. Hence the next move of va is the same as that of ua, so by induction, the whole of the segment $n \ldots n'$ is the same in each.

The above argument can be used as the inductive step in a proof of the fact that $\overline{ua} = \overline{va}$, and then used again to show that since $ua \ldots b \in \sigma \parallel \tau$, we also have $va \ldots b \in \sigma \parallel \tau$ (with the same moves between a and b). We can conclude that $tab \in \sigma ; \tau$. $\square$

Since composition preserves both innocent and a/p-strategies, and since the identity strategy on any game is innocent, we have the following.

Proposition 10 $\mathcal{G}_{a/p}$ and $\mathcal{G}_{\mathrm{inn}}$ are categories.

7.1.7 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 $\mathcal{G}_{a/p}$ 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

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

We can now define natural isomorphisms unit, assoc and comm with components $\mathsf{unit}_A : A \otimes I \to A$, $\mathsf{assoc}_{A,B,C} : A \otimes (B \otimes C) \to (A \otimes B) \otimes C$ and $\mathsf{comm}_{A,B} : A \otimes B \to B \otimes A$ 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 11 The structure described above makes $\mathcal{G}_{a/p}$ into a symmetric monoidal category. $\mathcal{G}_{\text{inn}}$ is a sub-symmetric monoidal category of $\mathcal{G}_{a/p}$.

7.1.8 Closed structure

To make $\mathcal{G}_{a/p}$ 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

$$\begin{array}{rcl} \mathcal{G}_{a/p}(A \otimes B, C) & = & \{\sigma \mid \sigma \text{ is an a/p-strategy for } A \otimes B \multimap C\} \\ & \cong & \{\sigma \mid \sigma \text{ is an a/p-strategy for } A \multimap (B \multimap C)\} \\ & = & \mathcal{G}_{a/p}(A, B \multimap C). \end{array}$$

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

Proposition 12 $\mathcal{G}_{a/p}$ is an autonomous (i.e. symmetric monoidal closed) category. $\mathcal{G}_{\text{inn}}$ is a sub-autonomous category of $\mathcal{G}_{a/p}$.

7.1.9 Products

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

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

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

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

Proposition 13 $A\&B$ is the product of A and B in both $\mathcal{G}_{a/p}$ and $\mathcal{G}_{\text{inn}}$, with projections given by π_1 and π_2.

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

7.2 Exponential

Definition Given an a/p-game A, define the game $!A$ as follows.

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

Notice that $L_{!A} = L_A$, so it is only the set of valid positions $P_{!A}$ which distinguishes $!A$ from A: a valid position of $!A$ is an interleaving of several valid positions of A.

7.2.1 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 An a/p-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 14 If A and B are well-opened games, and σ is an a/p-strategy for $!A \multimap B$, then $\sigma^\dagger$ is an a/p-strategy for $!A \multimap !B$. If σ is innocent then so is $\sigma^\dagger$.

7.2.2 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 15 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 16 (Bang Lemma) If B is well-opened and $\sigma : !A \to !B$ is innocent then $\sigma = (\sigma\,;\mathsf{der}_B)^\dagger$.

7.2.3 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 $s_1 = s \restriction I$ and $s_2 = s \restriction J$. Then define con_A as

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

7.2.4 Exponential isomorphisms

These reduce to *identities* in the present setting:

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

7.3 The cartesian closed categories $\mathcal{C}_{a/p}$ and $\mathcal{C}_{\text{inn}}$

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

$$\begin{array}{rcl} \text{Objects} & : & \text{Well-opened a/p-games} \\ \text{Morphisms } \sigma : A \to B & : & \text{A/P-strategies for } !A \multimap B \end{array}$$

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 $\mathcal{C}_{\text{inn}}$ of $\mathcal{C}_{a/p}$.

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

Moreover,

$$\begin{array}{rcl} \mathcal{C}_{a/p}(A \times B, C) & = & \mathcal{G}_{a/p}(!(A\&B), C) \\ & = & \mathcal{G}_{a/p}(!A \otimes !B, C) \\ & \cong & \mathcal{G}_{a/p}(!A, !B \multimap C) \\ & = & \mathcal{C}_{a/p}(A, !B \multimap C). \end{array}$$

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

Proposition 17 $\mathcal{C}_{a/p}$ is a cartesian closed category, and $\mathcal{C}_{\text{inn}}$ is a lluf sub-cartesian closed category of $\mathcal{C}_{a/p}$.

There are two full subcategories of $\mathcal{C}_{a/p}$ which deserve mention: the first consists of those games in which all moves are passive, and the second contains the games in which all moves are active. These two extreme cases have been considered before. The former is precisely the category presented in [16, 17] and used to provide a fully abstract model of the functional language FPC, while the latter is the category of arbitrarily history-sensitive strategies of [5, 6], which contains the fully abstract model of IAa. Thus $\mathcal{C}_{a/p}$ contains a whole spectrum of computational possibilities, with the model of pure IA lying in the middle.

7.3.1 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 $\bot = \{\varepsilon\}$. 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 18 $\mathcal{G}_{a/p}$ is a **cpo**-enriched autonomous category. $\mathcal{C}_{a/p}$ 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 (together with justification pointers) defined by

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

Similarly we define the ***a/p-view-function*** of an a/p-strategy.

Proposition 19 The a/p-strategies for a game A form a dI-domain; the compact strategies are those with a finite a/p-view-function.

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 does not necessarily have a finite a/p-view-function. For example, by the Bang Lemma any innocent strategy for $!A$ other than $\bot$ has an infinite set of positions; if all moves of A are active, this gives rise to an infinite a/p-view-function. Thus we shall speak of an innocent strategy $\sigma : A$ with finite view function being ***innocently compact*** i.e. compact in $\mathcal{G}_{\text{inn}}(I, A)$. For such strategies, we write $|\sigma|$ for the size of the view-function of σ.

7.4 Interpreting Algol types

Given any set X, let $\widetilde{X_\star}$ be the game with one initial question q and one answer for each move in the set X. All moves are passive. The valid positions of the game are given by the set

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

Thus in the hom-set $\mathcal{C}_{a/p}(1, \widetilde{X_\star})$, there is the empty strategy $\{\varepsilon\}$ and for each $x \in X$ a strategy $\overline{x} = \{\varepsilon, qx\}$. For any family of maps $(f_x : 1 \to A \mid x \in X)$, there is a morphism $[f_x] : \widetilde{X_\star} \to A$ given by the strategy which responds to the initial move of A by q in $\widetilde{X_\star}$ and after receiving the answer x continues to play in A according to f_x. This morphism has the property that for each $x_0 \in X$,

$$\overline{x_0}\,;[f_x] = f_{x_0}.$$

All this says that $\mathcal{C}_{a/p}$ has *standard datatypes* in the terminology of [2], and allows us to interpret the expression types of IA. Notice that, in order to allow us to define the strategy $[f_x]$, it is necessary that the moves of $\widetilde{X_\star}$ be passive.

Notice also that the formation of $[f_x]$ can be parameterized, that is, for any family of morphisms $(f_x : A \to B \mid x \in X)$ we can form a map $[f_x] : A \times \widetilde{X_\star} \to B$ with the obvious parameterized version of the property above. Furthermore, $\widetilde{X_\star}$ is used linearly, so when working in $\mathcal{G}_{a/p}$, we can consider $[f_x]$ as a strategy for $!A \otimes \widetilde{X_\star} \multimap B$.

The IA type `exp` is therefore interpreted by the game $\widetilde{\mathbb{N}_\star}$, which we shall write as $\mathbf{N}$ for convenience. The type `com` is interpreted as the game with a single initial question `run` and one answer `done`, both active. The valid positions are simply

$$\{\varepsilon, \mathtt{run}, \mathtt{run} \cdot \mathtt{done}\}.$$

Finally, the type `var` is interpreted as $\mathtt{exp} \times \mathtt{com}^{\mathbb{N}}$; we shall refer to the initial move of the first component as `read` and to the moves of the xth copy of `com` as $\mathtt{write}(x)$ and `ok`. The strategy interpreting `seq` is therefore:

$$\begin{array}{ccccc}
\mathtt{com} & \Rightarrow & \mathtt{com} & \Rightarrow & \mathtt{com} \\
 & & & & \mathtt{run} \\
\mathtt{run} & & & & \\
\mathtt{done} & & & & \\
 & & \mathtt{run} & & \\
 & & \mathtt{done} & & \\
 & & & & \mathtt{done}
\end{array}$$

and the interpretation of `assign` is as follows.

$$\begin{array}{ccccc}
\mathtt{var} & \Rightarrow & \mathtt{exp} & \Rightarrow & \mathtt{com} \\
 & & & & \mathtt{run} \\
 & & \mathtt{read} & & \\
 & & x & & \\
\mathtt{write}(x) & & & & \\
\mathtt{ok} & & & & \\
 & & & & \mathtt{done}
\end{array}$$

The command `skip` is interpreted by the strategy which responds to `run` with `done`; dereferencing is interpreted by the first projection from `var` to `exp`.

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.

7.5 Intrinsic Preorder

Our full abstraction result will in fact hold not in $\mathcal{C}_{a/p}$, but in the quotient of $\mathcal{C}_{a/p}$ with respect to a certain preorder, which we now define. Let Σ be the game with a single question q and one answer a, both active (so Σ is isomorphic to the game interpreting `com` as described above). There are only two strategies for Σ: $\perp = \{\varepsilon\}$ and $\top = \{\varepsilon, qa\}$. Maps $\alpha : A \to \Sigma$ in $\mathcal{C}_{a/p}$ 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 20 $\lesssim$ is a preorder on each hom-set of $\mathcal{C}_{a/p}$, and the quotient $\mathcal{C}_{a/p}/\lesssim$ of $\mathcal{C}_{a/p}$ by $\lesssim$ is a poset-enriched cartesian closed category.

8 $\mathsf{IA} - \{\texttt{new}\}$

The structure exhibited for $\mathcal{G}_{\text{inn}}$ in the previous section allows us to model $\mathsf{IA} - \{\texttt{new}\}$ according to the definitions given in Section 4. Since innocent strategies correspond precisely to functional programs, as shown in [11, 17, 16], this provides some immediate confirmation of our thesis that $\mathsf{IA} - \{\texttt{new}\}$ is a pure functional language.

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

Theorem 21 (Innocent Definability) Let T be an IA type. Every compact innocent strategy σ on the game $[\![T]\!]$ is definable in $\mathsf{IA} - \{\texttt{new}\}$; i.e. for some closed term $\vdash M : T$, $[\![M]\!] = \sigma$.

The proof of this result is very similar to those of similar results given in [17, 16, 11] and follows the general approach of the axiomatic proof of full abstraction introduced in [2]. However, the novel distinction between active and passive moves in the present setting introduces some differences, so we shall describe the proof again, omitting those parts of the verification which have been covered in previous papers and can be regarded as standard.

Consider an innocent strategy

$$\sigma : !T \otimes (!A \multimap B) \multimap C$$

where B and C are either `exp` or `com`. (Note the linearity of the type $(!A \multimap B)$.) Suppose that σ responds to the initial question q in C with the initial question q' in B, so that plays in σ all have the form (a prefix of)

$$q \cdot q' \cdot s \cdot b \cdot t$$

where b answers q'. Note that by the bracketing condition the moves of s are all in $!T$ or $!A$, and that at any point in s with P to play there is exactly one initial move of $!A$ in the P-view. Consider an initial move of $!T$ occurring in s. If this is active, the unique visible initial $!A$-move must also be active, since it is an unanswered question. Therefore we can turn s into a position of

$$!T \multimap !A$$

by altering the justifier of each initial move in $!T$ to point to the appropriate initial move of $!A$. (The argument above shows that the activity condition is satisfied and that this rearrangement of justification pointers is well-defined; the remaining conditions are straightforward to verify.) We now have:

Lemma 22 After the manipulation described above, the set

$$\{s \mid q \cdot q' \cdot s \in \sigma, q' \text{ unanswered in } s\}$$

is an innocent strategy for $!T \multimap !A$, which we call $\mathsf{arg}(\sigma)$. If σ is compact, then $|\mathsf{arg}(\sigma)| < |\sigma|$.

We can also form the sequences t into strategies.

Lemma 23 With σ as above, suppose $B = \texttt{exp}$. Then for each n, let

$$\sigma_n = \{t \mid q \cdot q' \cdot n \cdot t \in \sigma\}.$$

Each σ_n is an innocent strategy for $!T \multimap C$. If $|\sigma|$ is finite, then $|\sigma_n| < |\sigma|$.

Furthermore, the strategy given by

$$
\begin{tikzcd}
!T \otimes (!A \multimap \mathbf{N}) \arrow[d, "\mathsf{con} \otimes \mathsf{id}"] \\
!T \otimes !T \otimes (!A \multimap \mathbf{N}) \arrow[d, "\mathsf{arg}(\sigma) \otimes \mathsf{id}"] \\
!A \otimes !T \otimes (!A \multimap \mathbf{N}) \arrow[d, "\cong"] \\
!T \otimes (!A \multimap \mathbf{N}) \otimes !A \arrow[d, "\mathsf{id} \otimes \mathsf{ev}"] \\
!T \otimes \mathbf{N} \arrow[d, "{[\sigma_n]}"] \\
C
\end{tikzcd}
$$

is equal to σ. A corresponding result holds when $B = \mathtt{com}$.

The following lemma lets us transform strategies at IA types into strategies of the form discussed above.

Lemma 24 (Linearization of Head Occurrence) Let $\sigma : A \otimes !B \multimap C$ where C has a unique initial move, and suppose that σ responds to this move by a move in $!B$. Then there exists a strategy $\sigma' : A \otimes !B \otimes B \multimap C$ which responds to the first move by a move in B and satisfies

$$
\begin{tikzcd}
A \otimes !B \arrow[r, "\mathsf{id} \otimes \mathsf{con}"] \arrow[d, "\sigma"'] & A \otimes !B \otimes !B \arrow[d, "\mathsf{id} \otimes \mathsf{id} \otimes \mathsf{der}"] \\
C & A \otimes !B \otimes B \arrow[l, "\sigma'"]
\end{tikzcd}
$$

Proof To construct σ', simply relabel all of the moves of σ hereditarily justified by the first move in $!B$ to be in the separate tensor factor B. $\square$

We can now give the proof of the Innocent Definability Theorem. Let $T = T_1 \Rightarrow T_2 \Rightarrow \ldots \Rightarrow B$ be an IA type, where B is a base type. Let $\sigma : 1 \to [\![T]\!]$ be a compact innocent strategy. We must find a term M such that $[\![M]\!] = \sigma$. Clearly if we can find M' such that $[\![x_1 : T_1, \ldots, x_n : T_n \vdash M']\!] : [\![T_1 \times \ldots \times T_n]\!] \to [\![B]\!]$ is the uncurrying of σ then setting $M = \lambda\vec{x}.M'$ completes the proof, so let us consider instead strategies

$$\sigma : !T_1 \otimes \ldots \otimes !T_n \multimap B,$$

where we abuse notation by confusing IA types with their game interpretations. We shall only consider the case where B is `exp` or `com`: the case of `var` can be handled similarly using `mkvar`. The proof proceeds by induction on $|\sigma|$. We apply case analysis according to σ's response to the initial question of B. There are three cases.

- σ has no response (this incorporates the base case). Then $M = \Omega$ defines σ.
- σ responds with n, if B is `exp`, or with `done` if B is `com`. Then $M = \mathtt{n}$ or $M = \mathtt{skip}$ defines σ.
- σ responds by asking the initial question in some $!T_i$. Then by Linearization of Head Occurrence we obtain a strategy

  $$\sigma' : !T_1 \otimes \ldots \otimes !T_n \otimes T_i \multimap B.$$

  Let $T_i = !T_{i,1} \otimes \ldots \otimes !T_{i,l} \multimap C$ where C is a base type. (We are being sloppy about currying in this type, eliding the necessary natural isomorphisms. A more careful treatment of a similar argument can be found in [1].) If C is `exp`, by Lemmas 22 and 23 we obtain strategies

  $$\mathsf{arg}(\sigma') : !T_1 \otimes \ldots !T_k \multimap !(T_{i,1} \times \ldots \times T_{i,l})$$

  and

  $$\sigma'_n : !T_1 \otimes \ldots !T_k \multimap B.$$

  By the Bang Lemma and the universal property of product, $\mathsf{arg}(\sigma')$ can be written as $\langle \tau_1, \ldots, \tau_n \rangle^\dagger$ where

  $$\tau_j : !T_1 \otimes \ldots !T_k \multimap T_{i,j}.$$

  By the inductive hypothesis, each τ_j and each σ_n is definable by some term M_j or N_n respectively. Moreover, since σ is compact, there is some m such that $\sigma_n = \bot$ for all $n \geq m$. It is now easy to check that the term

  $$\mathtt{case}_m(x_i M_1 \ldots M_l) \;\; N_1 \ldots N_m$$

  defines σ, using Lemma 23.

  If C is `com`, as above we obtain

  $$\tau_j : !T_1 \otimes \ldots !T_k \multimap T_{i,j}$$

and a strategy $\sigma_\bullet : !T_1 \otimes \ldots !T_k \multimap B$; again these can be defined by some terms $M_1, \ldots, M_l$ and N. Note that by the activity condition, the type B must also be `com`. It is this fact which allows us to establish definability without the need for constants which allow expressions to have side-effects. It can now be checked that the term

$$\texttt{seq}\ (x_i M_1 \ldots M_l)\ N$$

defines σ.

Finally, if C is `var`, the first move made by σ is either `read` or $\texttt{write}(n)$ for some n. In either case we can proceed exactly as above, obtaining a term

$$\texttt{case}_m(\texttt{deref}\ (x_i M_1 \ldots M_l))\ N_1 \ldots N_m$$

or

$$\texttt{seq}(\texttt{assign}\ (x_i M_1 \ldots M_l)\ \mathbf{n})\ \ N.$$

9 Modelling `new`

Returning to the discussion in Section 6, to complete our semantics for Idealized Algol, we must specify a morphism

$$\texttt{cell} : I \rightarrow !\texttt{var}.$$

Plays in $!$`var` have the form

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

where each `read` and each $\texttt{write}(-)$ is initial, and all other moves are justified by the immediately preceding move. Note that the type $!$`var` 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 $\texttt{write}(v)$ 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 $\texttt{cell}_{v_0}$ 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 a/p-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 $\mathsf{IA} - \{\texttt{new}\}$ lives entirely in the subcategory $\mathcal{C}_{\text{inn}}$ (and implicitly in $\mathcal{G}_{\text{inn}}$), while to model `new` we have to use the larger category $\mathcal{G}_{a/p}$ of a/p-strategies. An obvious question then arises: what subcategory of $\mathcal{G}_{a/p}$ is generated by $\mathcal{G}_{\mathrm{inn}}$ and `cell`? The answer is provided by the following result.

Theorem 25 (Innocent Factorization) Let A be a game such that M_A is countable and satisfies

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

and $\sigma : I \longrightarrow A$ be any a/p-strategy. Then there exists $x_0 \in \mathbb{N}$, and an innocent strategy $\tau : !\mathtt{var} \longrightarrow A$ such that

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

Moreover, if σ is compact, τ is innocently compact. Note that this is a weak orthogonality property in the sense of factorization systems [8].

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 active moves in 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.

Fix an encoding $\mathsf{code}(-)$ of justified sequences of moves of A as natural numbers, and let x_0 be the code for the initial position ε. After each O-move in A, the strategy τ will perform one `read` move, obtaining the sequence of moves played at active positions so far (provided O plays according to $\mathtt{cell}_{x_0}$ in $!\mathtt{var}$). Then, if the current position is active, τ makes a $\mathtt{write}(-)$ move, updating the stored sequence to include the move just made by O and the move τ is about to make in A. Finally, τ moves in A. Hence the P-view after an O-move in A looks something like

$$m_0 \cdot \mathtt{read} \cdot \mathsf{code}(t_0) \cdot n_1 m_1 \cdot \mathtt{read} \cdot \mathsf{code}(t_1) \cdot \mathtt{write}(-) \cdot \mathtt{ok} \cdot n_2 m_2 \cdot \mathtt{read} \cdot \mathsf{code}(t_2) n_3 m_3 \cdots.$$

In this example, the sequence $m_0 n_1 m_1 n_2 m_2 \ldots$ is the P-view of the play in A, while the sequence $t_0, t_1, t_2, \ldots$ is an increasing chain of sequences of moves, namely those moves played at active positions up to each O-move m_j of the view. The move m_1 is at an active position, while all other O-moves are at passive positions. (In fact, by the activity condition, this example cannot arise. We hope the reader will forgive this "poetic licence", which helps keep the example short.) From this information we can calculate the a/p-view of play in A so far, by merging the view $m_0 n_1 m_1 n_2 m_2 \ldots$ with the moves from the sequences $t_0, t_1, t_2, \ldots$ in the following way. Suppose that

$$\begin{array}{rcl} t_1 & = & t_0 \cdot t_1' \\ t_2 & = & t_1 \cdot t_2' \\ & \vdots & \end{array}$$

The sequence $t_0 \cdot t'_1 \cdot t'_2 \cdots$ contains all the moves played at active positions. We just need to insert into this all the moves that appear in the P-view which are not already there, that is, those moves in the P-view which occur at passive positions. Since the sequence of moves up to and including m_1 is an active position, there is no need to insert $m_1 \cdot n_2$. The other moves in the view do need to be added. The a/p-view of play in A is therefore given by

$$t_0 \cdot mn_1 \cdot t'_1 \cdot t'_2 \cdot m_2 \cdots$$

In general, to construct the a/p-view sequence, take the sequence

$$t_0 \cdot t'_1 \cdot t'_2 \cdot t'_3 \cdots$$

and insert the moves $m_i \cdot n_{i+1}$ between t'_i and t'_{i+1} whenever the position at m_i is passive.

Thus, provided O plays according to the strategy $\mathtt{cell}_{x_0}$ in $!A$, the P-view of the whole game contains sufficient information to calculate the a/p-view of the play in A. It is this which allows the innocent strategy τ to simulate the a/p-strategy σ. We now describe the behaviour of τ at a position sa, where a is an O-move in A, taking into account the fact that in general O need not play according to `cell` in the variable part of the game.

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 $\mathsf{code}(t_i)$ from the variable. If the sequence of codes $t_0, t_1, \ldots, t_i$ read from the variable do not form an increasing chain, τ has no further response. Otherwise, calculate the sequence u as indicated above which will be the a/p-view of play in A provided O plays according to `cell`. Let $b = \sigma(u)$. If $\ulcorner sa \upharpoonright A \urcorner b \notin P_A$, τ has no further response. Otherwise, if $sa \upharpoonright A$ is an active position, τ writes $\mathsf{code}(t_i \cdot ab)$ back into the variable. Finally, τ plays the move 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 $\mathtt{cell}_{x_0} ; \tau = \sigma$, as required. Further, if σ is compact, it only has a response b at a finite number of a/p-views, and hence at a finite number of views. In this case, the strategy τ only has a response at a finite number of views, so τ is innocently compact. $\square$

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

Lemma 26 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 [17]. 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 $sa \restriction ![\![A]\!] \in P_{![\![A]\!]}$ and $sa \restriction [\![B]\!] \in P_{[\![B]\!]}$. 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 [17] 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, as is the activity condition. For the visibility condition, one can adapt the proof of Lemma 3.1.6 of [17] to show that $\ulcorner s \urcorner \restriction [\![B]\!]$ is a subsequence of $\ulcorner s \restriction B \urcorner$, 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 27 For every IA type T, the game $[\![T]\!]$ satisfies the hypotheses of the Innocent Factorization Theorem.

Proof Countability of the set of moves is trivial. We must now 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 same fact implies that the activity condition is satisfied. 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 28 (Definability) Let $T = T_1 \Rightarrow T_2 \Rightarrow \ldots \Rightarrow \mathtt{com}$ be an IA type. Every compact strategy $\sigma : I \longrightarrow [\![T]\!]$ in $\mathcal{C}_{a/p}$ is definable in Idealized Algol.

Proof By Theorem 25 and Proposition 27, σ factors as

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

where τ is a compact innocent strategy. By the Innocent Definability Theorem, for some term $\vdash M : \mathtt{var} \Rightarrow T$ of $\mathsf{IA} - \{\mathtt{new}\}$, $\Lambda\tau = [\![M]\!]$. Then

$$\sigma = [\![\lambda\vec{x} : \vec{T}.\mathtt{new}(\lambda v : \mathtt{var}.\,\mathtt{seq}\,(\mathtt{assign}\,v\,\mathtt{x_0})\,(M\ v\ \vec{x}))]\!]. \qquad \square$$

10 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. [18]) denoted by $\equiv$. The structural congruence is that generated by β-conversion and all instances of

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

Let us introduce some notation. A `var`-context Γ is one of the form $\Gamma = x_1 : \texttt{var}, \ldots, x_n : \texttt{var}$, with the x_i distinct. A Γ-store is a partial function $s : \{x_1, \ldots, x_n\} \rightharpoonup \{\mathbf{n} \mid n \in \mathbb{N}\}$. We write $\mathrm{St}(\Gamma)$ for the set of all such stores. We write $(s \mid x \mapsto n)$ for the store obtained by 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. Similarly, $(s \mid x \mapsto \bot)$ updates a store so that x is undefined.

We now define relations of the form

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

where

- Γ is a `var`-context
- $s, s' \in \mathrm{St}(\Gamma)$
- B is a base type
- $\Gamma \vdash M : B$ is a term-in-context
- c is a *canonical form* such that $\Gamma \vdash c : B$ is derivable.

The canonical forms are:

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

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

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

Canonical forms

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

Conditional

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

Arithmetic

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

Sequencing

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

Dereferencing

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

Assignment

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

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

Block structure

$$\frac{\Gamma, x : \mathtt{var}, (s \mid x \mapsto \bot) \vdash M \Downarrow \mathtt{skip}, (s' \mid x \mapsto v)}{\Gamma, s \vdash \mathtt{new}\ \lambda x.M \Downarrow \mathtt{skip}, s'}$$

At first sight, the rule for `new` may seem restricted, since it applies only to terms of the form `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 $\mathtt{var} \Rightarrow \mathtt{exp}$, the terms `new` M and `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 $\mathsf{Trm}(\Gamma, T)$ be the set of all M such that $\Gamma \vdash M : T$ is derivable. The program contexts $\mathsf{Ctxt}(\Gamma, T)$ are those contexts $C[-]$ such that $C[M]$ is a program for all $M \in \mathsf{Trm}(\Gamma, T)$. The ***observational preorder*** $\underset{\sim}{\sqsubset}$ on $\mathsf{Trm}(\Gamma, T)$ is then defined by

$$M \underset{\sim}{\sqsubset} N \text{ iff } \forall C[-] \in \mathsf{Ctxt}(\Gamma, T). C[M]\Downarrow \Rightarrow C[N]\Downarrow.$$

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

$$I \xrightarrow{\cong} I \otimes I \otimes I \xrightarrow{\mathtt{cell}_3 \otimes \mathtt{cell} \otimes \mathtt{cell}_7} !\mathtt{var} \otimes !\mathtt{var} \otimes !\mathtt{var}.$$

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

$$\begin{array}{ccccccccc}
I & \multimap & !\mathtt{var} & \otimes & !\mathtt{var} & \otimes & !\mathtt{var} & \multimap & A \\
 & & & & & & & & q \\
 & & \mathtt{read} & & & & & & \\
 & & 3 & & & & & & \\
 & & & & & & \mathtt{write}(5) & & \\
 & & & & & & \mathtt{ok} & & \\
 & & & & \mathtt{write}(5) & & & & \\
 & & & & \mathtt{ok} & & & & \\
 & & & & & & & & a
\end{array}$$

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

10.2 Soundness

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

Lemma 29 For any var-context Γ, any term $\Gamma \vdash M : A$ and any $s \in \mathrm{St}(\Gamma)$, if $\Gamma, s \vdash M \Downarrow c, s'$ 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 $[\![\Gamma]\!]$ 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 $\Gamma, s \vdash M \Downarrow c, s'$. 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

$$\frac{\Gamma, x : \mathtt{var}, (s \mid x \mapsto \bot) \vdash M \Downarrow \mathtt{skip}, (s' \mid x \mapsto v)}{\Gamma, s \vdash (\mathtt{new}\,\lambda x.M) \Downarrow \mathtt{skip}, s'}$$

Notice that $[\![(s \mid x \mapsto \bot)]\!]$ is the extension of the interpretation of $[\![s]\!]$ as a Γ state by a single uninitialized cell, that is

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

By the inductive hypothesis, the sequence $[\![(s \mid x \mapsto \bot)]\!] \parallel [\![M]\!]$ has the form

$$\begin{array}{ccccc} I & \xrightarrow{[\![(s|x\mapsto\bot)]\!]} & [\![\Gamma]\!] \otimes !\mathtt{var} & \xrightarrow{[\![M]\!]} & \mathtt{com} \\ & & & & \mathtt{run} \\ & & t & & \\ & & & & \mathtt{done} \end{array}$$

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.

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

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 $[\![s']\!] \parallel [\![N]\!]$ has the form $\mathtt{run} \cdot t_2 \cdot \mathtt{done}$, where t_2 leaves state s''. The unique maximal sequence in $[\![s'']\!] \parallel [\![\mathtt{skip}]\!]$ is $\mathtt{run} \cdot \mathtt{done}$. Because of the copycat nature of $[\![\mathtt{seq}]\!]$, it is then easy to show that $[\![s]\!] \parallel [\![\mathtt{seq} MN]\!]$ consists of strings of the form $\mathtt{run} \cdot t_1 \cdot t_2 \cdot \mathtt{done}$, giving the result. □

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

Proposition 30 If $\Gamma, s \vdash M \Downarrow c, s'$ then $[\![s]\!]\,;[\![M]\!] = [\![s']\!]\,;[\![c]\!]$.

10.3 Computational Adequacy

We now strengthen our soundness result to show that the model is *computationally adequate*, that is if for some $\Gamma \vdash M$ of base type and some state $s \in \mathrm{St}(\Gamma)$

we have $[\![s]\!]\,;[\![M]\!] \neq \perp$, then $\Gamma, s \vdash M \Downarrow$. 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 $\mathtt{var}$-context Γ a relation $\precsim^{\Gamma}_{T}$ between strategies $\sigma : ![\![\Gamma]\!] \multimap [\![T]\!]$ and terms $\Gamma \vdash M : T$ as follows.

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

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

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

$$[\![s]\!]\,;\sigma = [\![\mathbf{n}]\!] \Rightarrow \Gamma, s \vdash M \Downarrow \mathbf{n}, s.$$

  Note that in this case, the interaction of $[\![s]\!]$ with σ can contain no active moves, so the state s is left unchanged.

- $\sigma \precsim^{\Gamma}_{\mathtt{var}} M$ iff

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

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

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

$$A \xrightarrow{\ \cong\ } A \otimes I \xrightarrow{\ \mathsf{id} \otimes \mathtt{cell}\ } A \otimes !\mathtt{var} \xrightarrow{\ \Lambda^{-1}(\sigma)\ } B.$$

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

Lemma 31 Let $\Gamma, \Delta \vdash M : T$, where Γ is a part of the context containing only variables of type $\mathtt{var}$ while Δ may contain variables of any type (including $\mathtt{var}$). Suppose $\Delta = y_1 : T_1, \ldots, y_n : T_n$ and let σ_i be strategies and N_i be terms such that $\sigma_i \precsim^{\Gamma}_{T_i} N_i$ for $i = 1, \ldots, n$. Write $\vec{\sigma_i} \mathbin{⨾} [\![M]\!]$ for the composite

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

Then $\vec{\sigma_i} \mathbin{⨾} [\![M]\!] \precsim^{\Gamma}_{T} M[\vec{N_i}/\vec{y_i}]$.

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

- If M is a variable, there are two subcases: either the variable is from Γ 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 $[\![\Gamma \vdash x]\!] \precsim^{\Gamma}_{\mathtt{var}} x$, which is simple.

- If M is an abstraction, again there are two cases: either it has type $\mathtt{var} \Rightarrow \mathtt{com}$, or some other type. In the latter case the result follows straightforwardly from the inductive hypothesis. Otherwise, suppose we have $\Gamma, \Delta \vdash \lambda x.M : \mathtt{var} \Rightarrow \mathtt{com}$, derived from $\Gamma, \Delta, x : \mathtt{var} \vdash M : \mathtt{com}$. 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 thought of as being in the "Γ-part". The inductive hypothesis gives us

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

  We must show that

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

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

$$[\![(s \mid x \mapsto \perp)]\!]\,;(\vec{\sigma_i} \fatsemi [\![M]\!])$$

  so the result follows from the inductive hypothesis.

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

Proposition 32 (Computational Adequacy) Let P be a program. Then $P\Downarrow \iff [\![P]\!] \neq \perp$.

Proof The left-to-right implication is given by Proposition 30. For the right-to-left direction, we use the previous Lemma, which tells us that $[\![P]\!] \preccurlyeq_{\mathtt{com}} P$, so that if $[\![P]\!] \neq \perp$, then $P\Downarrow$ by definition of $\preccurlyeq_{\mathtt{com}}$. $\square$

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

Theorem 33 (Inequational Soundness) Let $M, N \in \mathsf{Trm}(\Gamma, T)$. If $[\![M]\!] \subseteq [\![N]\!]$ then $M \lesssim N$.

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

Note that this result applies to the model in $\mathcal{C}_{a/p}$. To extend it to the model in $\mathcal{C}_{a/p}/\lesssim$, just observe that Proposition 32 immediately extends to that model. The above proof can therefore be used to show the following.

Theorem 34 (Inequational Soundness in $\mathcal{C}_{a/p}/\lesssim$)
Let $M, N \in \mathsf{Trm}(\Gamma, T)$. If $[\![M]\!] \lesssim [\![N]\!]$ then $M \mathrel{\underset{\sim}{\sqsubset}} N$.

11 Full Abstraction

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

Theorem 35 (Full Abstraction) Let $M, N \in \mathsf{Trm}(\Gamma, T)$. Then

$$[\![M]\!] \lesssim [\![N]\!] \iff M \mathrel{\underset{\sim}{\sqsubset}} N.$$

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 $[\![M]\!] \,;\, \alpha = \top$ and $[\![N]\!] \,;\, \alpha = \perp$. Clearly α can be chosen to be compact. Since the game Σ is the same as $[\![\texttt{com}]\!]$, the Definability Theorem implies that there is an IA term $x : T \vdash C[x] : \texttt{com}$ such that $[\![C[x]]\!] = \alpha$. Therefore $[\![M]\!] \,;\, \alpha = [\![C[M]]\!]$, and similarly for N. By Computational Adequacy, $C[M]\Downarrow$ and $C[N]\Uparrow$. Therefore $M \not\mathrel{\underset{\sim}{\sqsubset}} N$, completing the proof. $\square$

11.1 Some remarks

It is very easy to show that the full subcategory of $\mathcal{C}_{a/p}$ consisting of passive games (games in which all moves are passive) is equivalent to the category of games described in [16, 17], which is a mild variant of Hyland and Ong's category [11]. Furthermore, for PCF types, the quotient by $\lesssim$ is the same as the extensional quotient used to arrive at a fully abstract model of PCF. Therefore the category giving the fully abstract model of IA also gives a fully abstract model of PCF. We therefore have:

Theorem 36 (Conservativity) IA is a conservative extension of PCF, that is to say, two PCF terms are equivalent if and only if they are equivalent as terms of IA.

This fact, which has long been held to be folklore, was first proved formally by Peter O'Hearn [25] for versions of PCF and IA augmented with parallel conditional constants. He remarked that his proof can easily be adapted to handle the purely sequential versions of the languages, but did not give the details; our model seems to be the first to establish this result explicitly.

In [5, 6] it was shown that the fully abstract model of Idealized Algol with active expressions is *effectively presentable*, and noted that no such result is possible for the games models of PCF, since this would contradict Loader's result on the undecidability of finitary PCF [15]. The theorem above implies that the model presented here is not effectively presentable, for precisely the same reasons.

12 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 [27], and has since been considerably refined, notably by O'Hearn and Tennent [24]. 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 [35], and similar ideas have led to denotational models of the π-calculus [10, 36]. The second, perhaps computationally more compelling, method has been termed "Object based semantics" by Reddy [30]. 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 [21], while Reddy realized the same ideas denotationally, using coherence spaces [30]. Our work, both here and in [5, 6], can be seen as the next stage in this line of enquiry.

As regards full abstraction, the strongest results previously achieved have been by O'Hearn and Reddy [23], for a model combining object-based and functor-category ideas, and by Sieber [34], for a model using logical relations. In both cases, the results are that the model captures definability of first-order functions, and full abstraction for closed second-order terms. O'Hearn and Reddy's results are for a version of Idealized Algol similar to the one we considered in [5, 6], in which expressions may have side-effects. Sieber's results are for a language without side-effecting expressions, but with "snap-back" (a feature which allows state changes to be reversed) and parallel conditional.

A somewhat different investigation into the behaviour of Algol-like languages has recently been undertaken by Pitts [28], 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 [17], or to develop a novel logic for reasoning about Algol programs. This is the subject of current research.

References

[1] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. Accepted for publication in Information and Computation, 1997.

[2] S. Abramsky. Axioms for full abstraction and full completeness. In G. Plotkin, C. Sterling, and M. Tofte, editors, *Essays in Honour of Robin Milner*. MIT Press, to appear.

[3] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic*, 59(2):543 – 574, June 1994. Also appeared as Technical Report 92/24 of the Department of Computing, Imperial College of Science, Technology and Medicine.

[4] S. Abramsky and G. McCusker. Games and full abstraction for the lazy λ-calculus. In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 234–243. IEEE Computer Society Press, 1995.

[5] S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions (extended abstract). In *Proceedings of 1996 Workshop on Linear Logic*, volume 3 of *Electronic notes in Theoretical Computer Science*. Elsevier, 1996.

[6] S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In O'Hearn and Tennent [26], pages 297–329 of volume 2.

[7] 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.

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

[9] V. Danos, H. Herbelin, and L. Regnier. Game semantics and abstract machines. In *Proceedings, Eleventh Annual IEEE Symposium on Logic in Computer Science* [12], pages 394–405.

[10] M. P. Fiore, E. Moggi, and D. Sangiorgi. A fully-abstract model for the π-calculus (extended abstract). In *Proceedings, Eleventh Annual IEEE Symposium on Logic in Computer Science* [12], pages 43–54.

[11] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III. Accepted for publication in Information and Computation, 1997.

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

[13] G. M. Kelly. *Basic Concepts of Enriched Category Theory.* Cambridge University Press, 1982.

[14] J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic.* Cambridge University Press, 1986.

[15] R. Loader. Finitary PCF is not decidable. Unpublished manuscript, 1996.

[16] G. McCusker. Games and full abstraction for FPC. In *Proceedings, Eleventh Annual IEEE Symposium on Logic in Computer Science* [12], pages 174–183.

[17] G. McCusker. *Games and Full Abstraction for a Functional Metalanguage with Recursive Types*. Distinguished Dissertations in Computer Science. Springer-Verlag, 1998.

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

[19] R. Milner. Processes, a mathematical model of computing agents. In *Logic Colloquium, Bristol 1973*, pages 157–174. North Holland, Amsterdam, 1975.

[20] R. Milner. Fully abstract models of typed lambda-calculi. *Theoretical Computer Science*, 4:1–22, 1977.

[21] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1980.

[22] H. Nickau. Hereditarily sequential functionals. In *Proceedings of the Symposium on Logical Foundations of Computer Science: Logic at St. Petersburg*, Lecture notes in Computer Science. Springer, 1994.

[23] 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.

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

[25] P. W. O'Hearn. Note on Algol and conservatively extending functional programming. *Journal of Functional Programming*, 6(1):171–180, January 1996.

[26] P. W. O'Hearn and R. D. Tennent, editors. *Algol-like Languages*. Birkhaüser, 1997.

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

[28] A. M. Pitts. Reasoning about local variables with operationally-based logical relations. In *Proceedings, Eleventh Annual IEEE Symposium on Logic in Computer Science* [12], pages 152–163.

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

[30] U. S. Reddy. Global state considered unnecessary: Object-based semantics for interference-free imperative programs. *Lisp and Symbolic Computation*, 9(1), 1996.

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

[32] J. C. Reynolds. The essence of Algol. In *Proceedings of the 1981 International Symposium on Algorithmic Languages*, pages 345–372. North-Holland, 1981.

[33] R. A. G. Seely. Linear logic, $*$-autonomous categories and cofree coalgebras. In *Category Theory, Computer Science and Logic*. American Mathematical Society, 1987.

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

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

[36] I. Stark. A fully abstract domain model for the π-calculus. In *Proceedings, Eleventh Annual IEEE Symposium on Logic in Computer Science* [12], pages 36–42.

[37] 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.

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