Electronic Notes in Theoretical Computer Science 83 (2004)
URL: http://www.elsevier.nl/locate/entcs/volume83.html 20 pages

On the Semantics of the Bad-Variable Constructor in Algol-like Languages

Guy McCusker [1]

University of Sussex
Falmer
Brighton BN1 9QH
United Kingdom

Abstract

The fully abstract games model of Reynolds's Idealized Algol is adapted to provide a characterization of the language without the "bad variable constructor" `mkvar`. The model shows that the addition of `mkvar` to the language is conservative for observational equivalence but not for the observational preorder.

Key words: Algol-like languages, game semantics, full abstraction, bad variables, conservative extension, observational equivalence.

1 Introduction

This paper presents a contribution to the study of the semantics of Algol-like languages. In a seminal paper [10], John Reynolds introduced a prototypical higher-order imperative programming language which has become known as Idealized Algol (IA). This language elegantly combines the features of a basic imperative language with a full higher-order procedure mechanism in the form of the λ-calculus, and is both clean and powerful as a result. A good deal of research in the semantics of imperative programs has focussed on IA and its variants; some 20 papers on the subject have recently been published in a collection [9].

One of Reynolds's key observations was that assignable variables can be seen semantically as objects with two methods: a dereferencing method and an assignment method. This technique has been used in most semantic accounts of Algol-like languages, including the fully abstract games-based models [2,3].

However, the particular variants of the language for which these games models are fully abstract import the "variables as objects" view into their

[1] Email: `guym@cogs.susx.ac.uk`

syntax. That is to say, the language is augmented with a variable constructor `mkvar` which takes an arbitrary "dereferencing method" and "assignment method" and creates an entity of type `var`, the type of mutable variables. These hand-made objects can exhibit behaviour quite unlike that of a genuine storage cell. For example, the "bad variable" $\texttt{mkvar}(\texttt{3})(\lambda n.\texttt{skip})$ always returns `3` when dereferenced, and ignores any attempt to assign to it.

Bad-variables are not merely an artefact of the object-oriented view of variables. In call-by-name languages, apparent bad-variable behaviour arises through *aliasing*: consider for example a program phrase like

$$\lambda x : \texttt{var}.\lambda y : \texttt{var}.x := 3; y := 2; \texttt{if}\ !x = 3\ \texttt{then}\ C\ \texttt{else}\ C'.$$

Ordinary programming intuition about good variables would suggest that C' is never executed here; but if x and y are both bound to the same variable z, then $!x$ evaluates to 2, not 3, so C' is executed. In some sense, x is behaving like a bad variable.

Another source of bad-variable behaviour is provided by array subscripting. Consider the phrase

$$\lambda x : \texttt{var}.x := 3; \texttt{if}\ !x = 3\ \texttt{then}\ C\ \texttt{else}\ C'.$$

If we apply this procedure to the argument

$$a[a[0]]$$

where a is an array of integers which are initially all 0, then C' will be executed: first $a[0]$ is set to 3, then $a[a[0]]$ is evaluated, which looks up the value of $a[3]$, yielding 0. Again, x behaves like a bad variable under these circumstances.

Although bad-variable behaviours can and do arise in call-by-name languages without the use of `mkvar`, the explicit inclusion of a bad-variable constructor in a programming language seems unnatural. Since Algol-like languages have typically been studied in the presence of such a constructor, it is important to ask whether its inclusion makes any difference from the point of view of observational equivalence, i.e. whether the addition of `mkvar` to the language is *conservative*. If `mkvar` changes the semantics of the language, one would then ask whether the games models can be adapted to give a fully abstract account of a `mkvar`-free variant of Idealized Algol.

It is easy to see that in a call-by-value language, the inclusion of a constructor like `mkvar` would make a dramatic difference to the theory of program equivalence. Without `mkvar`, every value of type `var` must indeed be a genuine variable name, so for instance

$$\lambda v : \texttt{var}.v := !v \cong \lambda v : \texttt{var}.\texttt{skip}.$$

This equivalence is easy to violate using `mkvar`; consider, for example, what happens when the bad variable $\texttt{mkvar}(!y)(\lambda n.x := n)$ is supplied to these two functions.

Since IA is a call-by-name language, this argument does not apply: the question of `mkvar`'s conservativity remains. In this paper, we show that for Idealized Algol with *active expressions*, that is, where phrases of natural num-

ber type can have side-effects, the addition of `mkvar` has *no effect on observational equivalence*, so the existing fully abstract model of IA with `mkvar` remains fully abstract for equivalence.

Somewhat surprisingly, however, `mkvar` is *not* conservative with respect to the observational *preorder* on IA terms. We therefore have an unusual situation in which the games model of Idealized Algol with `mkvar`, which is fully abstract with respect to the observational preorder, is also fully abstract for the language without `mkvar`, but for *behavioural equivalence only*. That is, the model is equationally but not inequationally fully abstract for IA without `mkvar`. To our knowledge, the only other result of this kind is due to Allen Stoughton [11] who shows that there exists a model of the functional language PCF which is equationally but not inequationally fully abstract.

The essence of our result is contained in the following example. In the language without `mkvar`, the following holds:

$$\texttt{if}\ !x = 3\ \texttt{then skip else diverge} \precsim x := 3.$$

This is because the only way for the left hand side to terminate is if x is bound to a phrase which evaluates to a variable currently holding the value `3`; and in that case, the right hand side will also terminate and the assignment will have no observable effect.

This inequation is not valid in the presence of `mkvar`: for instance, replacing x with $\texttt{mkvar}(3)(\lambda n.\texttt{diverge})$ will make the left hand side converge while the right hand side diverges. However, this is in some sense the only kind of inequation which `mkvar` renders invalid. We make this precise by defining a new preorder on the games model of IA which essentially adds all such inequations, and showing that this preorder gives rise to an inequationally fully abstract model of IA without `mkvar`.

This is the only known fully abstract model of a higher-order programming language with mutable store, without an explicit bad-variable constructor. Though fully abstract models have been constructed for other imperative languages, including IA with passive expressions [5], a heap-allocated call-by-value variant of IA [4], and a language with higher-order store [1], using game semantics, in each case the inclusion of `mkvar` is crucial to the full abstraction result.

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 of natural numbers. The base types of Idealized Algol are then

$$B ::= \texttt{exp} \mid \texttt{var} \mid \texttt{com}$$

i.e. natural-number-valued expressions, assignable program variables in which natural numbers can be stored, and commands. The types of Idealized Algol

are given by

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

The syntax of the language is as follows:

$$\begin{aligned} M ::= {} & x \mid n \mid \lambda x : T.M \mid MM \\ \mid {} & \texttt{succ}(M) \mid \texttt{pred}(M) \\ \mid {} & \texttt{ifzero } M \texttt{ then } M \texttt{ else } M \mid \texttt{Y}(M) \\ \mid {} & M := M \mid !M \mid \texttt{new } x \texttt{ in } M \\ \mid {} & \texttt{skip} \mid M; M \\ \mid {} & \texttt{mkvar } M \ M \end{aligned}$$

As usual x ranges over a countable collection of variables, and n over the natural numbers. The construct $\lambda x : T.M$ binds x in M, as does $\texttt{new } x \texttt{ in } M$. We identify terms up to α-conversion. We use infix $:=$ for assignment to variables, prefix ! for dereferencing, and infix ; for sequential composition.

We will work with terms-in-context $\Gamma \vdash M : T$ where Γ is a finite function associating types to variables, which will usually be written in list form such as $x : A, y : B, z : C$, M is a term of the language and T is a type. The typing rules are mostly standard; we give just three important ones. Our language has *active expressions*, which means that sequential composition has the typing rule

$$\frac{\Gamma \vdash M : \texttt{com} \quad \Gamma \vdash N : B}{\Gamma \vdash M; N : B}$$

for any base-type B.

The typing rule for $\texttt{new } x \texttt{ in } M$ is as follows.

$$\frac{\Gamma, x : \texttt{var} \vdash M : \texttt{com}}{\Gamma \vdash \texttt{new } x \texttt{ in } M : \texttt{com}}$$

Finally, the rule for `mkvar` is the following.

$$\frac{\Gamma \vdash M : \texttt{exp} \quad \Gamma \vdash N : \texttt{exp} \Rightarrow \texttt{com}}{\Gamma \vdash \texttt{mkvar} MN : \texttt{var}}$$

Although we have not included booleans in the language, we will make use of extended syntax such as $\texttt{if } x = 3 \texttt{ then } M \texttt{ else } N$ in examples; this can easily be encoded in our language.

Since we will be interested in the language both with and without the `mkvar` constant, we shall write IA for the language without `mkvar`, and $IA_{\texttt{mkvar}}$ for the language including `mkvar`.

The operational semantics of the language is given in standard fashion as a "big-step" evaluation relation, with judgements of the form

$$s, M \Downarrow s', V$$

where M is a term, V is a *value* (a numeral, `skip`, a variable x of type `var`, a `mkvar` term, or an abstraction), and s and s' are stores: functions from the

free var-typed variables in M to the natural numbers. We use the notation $(s \mid x \mapsto n)$ to denote the store resulting from updating s so that x is mapped to n.

We give a few of the rules defining the operational semantics of $IA_{\mathtt{mkvar}}$ below; the other rules are all standard. Of course, the operational semantics of IA is obtained simply by omitting the rules for mkvar.

Dereferencing

$$\frac{s, M \Downarrow s', x \quad s'(x) = n}{s, !M \Downarrow s', n}$$

$$\frac{s, M \Downarrow s', \mathtt{mkvar}\ N_1\ N_2 \qquad s', N_1 \Downarrow s'', n}{s, !M \Downarrow s'', n}$$

Assignment

$$\frac{s, N \Downarrow s', n \quad s', M \Downarrow s'', x}{s, M := N \Downarrow (s'' \mid x \mapsto n), \mathtt{skip}}$$

$$\frac{s, N \Downarrow s', n \quad s', M \Downarrow s'', \mathtt{mkvar}\ N_1\ N_2 \quad s'', N_2(n) \Downarrow s''', \mathtt{skip}}{s, M := N \Downarrow s''', \mathtt{skip}}$$

Block structure

$$\frac{(s \mid x \mapsto 0), M \Downarrow (s' \mid x \mapsto n), \mathtt{skip}}{s, \mathtt{new}\ x\ \mathtt{in}\ M \Downarrow s', \mathtt{skip}.}$$

The ***observational preorder*** $\underset{\sim}{\sqsubset}$ on terms of IA is defined as usual. For closed terms of type com, we write $M\Downarrow$ if $s, M \Downarrow s, \mathtt{skip}$, where s is the unique store over no variables. Given terms $\Gamma \vdash M, N : T$, $M \underset{\sim}{\sqsubset} N$ iff for all IA contexts $C[-]$ such that $C[M]$ and $C[N]$ are closed terms of type com,

$$C[M]\Downarrow \Rightarrow C[N]\Downarrow.$$

The relation of ***observational equivalence*** $\cong$ between terms of IA is defined by $M \cong N \iff M \underset{\sim}{\sqsubset} N \wedge N \underset{\sim}{\sqsubset} M$.

The observational preorder and equivalence on $IA_{\mathtt{mkvar}}$ are defined similarly: we write $\underset{\sim}{\sqsubset}_m$ and $\cong_m$ for these relations. Note that for these relations, the quantification over contexts in the definition includes contexts which make use of mkvar.

3 The games model of $IA_{\mathtt{mkvar}}$

We now briefly recall the definitions of the category of games which provides our model of $IA_{\mathtt{mkvar}}$ and the semantic definitions from [2,3]. We should remark that the definitions below are not identical to those in *loc. cit.*, but they give rise to an isomorphic model.

The games and strategies we use are direct descendants of those used by

Hyland, Ong and Nickau [6,8] to provide fully abstract models of PCF.

3.1 Arenas

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

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

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

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

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

  and so on. If $\lambda^{\mathsf{OP}}(a) = \mathsf{O}$, we call a an O-move; otherwise, a is a P-move.
- $\vdash_A$ is a relation between $M_A + \{\star\}$ and M_A, called ***enabling***, which satisfies
  - $\star \vdash_A a \Rightarrow \lambda_A(a) = \mathsf{OQ} \wedge [b \vdash_A a \iff b = \star]$.
  - $a \vdash_A b \wedge \lambda_A^{\mathsf{QA}}(b) = \mathsf{A} \Rightarrow \lambda_A^{\mathsf{QA}}(a) = \mathsf{Q}$.
  - $a \vdash_A b \wedge a \neq \star \Rightarrow \lambda_A^{\mathsf{OP}}(a) \neq \lambda_A^{\mathsf{OP}}(b)$.

The enabling relation tells us either that a move a is ***initial*** and needs no justification ($\star \vdash_A a$), or that it can be justified by another move b, if b has been played ($b \vdash_A a$).

A ***justified sequence*** s of moves in an arena A is a sequence of moves together with ***justification pointers***: for each move a in s which is not initial, there is a pointer to an earlier move b of s such that $b \vdash_A a$. We say the move b ***justifies*** a, and extend this terminology to say that a move b ***hereditarily justifies*** a if the chain of pointers back from a passes through b.

Given a justified sequence s, its ***view*** $\ulcorner s \urcorner$ is defined as follows.

$$\ulcorner \varepsilon \urcorner = \varepsilon$$
$$\ulcorner s \cdot a \urcorner = a, \text{if } a \text{ is initial}$$
$$\ulcorner s \cdot \overset{\frown}{a \cdot t \cdot b} \urcorner = \ulcorner s \urcorner \cdot \overset{\frown}{a \cdot} b.$$

A justified sequence s satisfies the ***visibility condition*** iff for all prefixes $t \cdot m$ of s, if m is not initial then the move justifying m lies in $\ulcorner t \urcorner$.

If s is a justified sequence, 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

$$t \cdot \overset{\frown}{q \cdot u \cdot} a$$

of s, all questions asked in u are answered within u; in other words, when an answer is given, it is always to the most recent question which has not been

answered.

A justified sequence s is a ***legal position*** iff:

- s is ***alternating***: if $s = s_1 abs_2$ then $\lambda^{\mathsf{OP}}(a) \neq \lambda^{\mathsf{OP}}(b)$.
- The visibility condition holds.
- The bracketing condition holds.

The set of all legal positions of an arena A is written L_A. We will also refer to legal positions as ***plays*** of A. A play s is ***complete*** if all questions in s are answered.

3.2 Strategies

A strategy for an arena A is a set of even-length positions, such that

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

Given a non-empty legal position sa in an arena A, the ***current thread*** $\mathsf{thread}(sa)$ is the subsequence of sa containing all moves hereditarily justified by the same initial move as a. A strategy σ for A is ***single-threaded*** iff

- if $sab \in \sigma$ then b is justified by a move in $\mathsf{thread}(sa)$; and
- if $sab, t \in \sigma$, and $ta \in L_A$ is such that $\mathsf{thread}(sa) = \mathsf{thread}(ta)$, then $tab \in \sigma$, with the justification pointer on b such that $\mathsf{thread}(tab) = \mathsf{thread}(sab)$.

That is to say, a single-threaded strategy chooses its move at a position sa based just on the moves in the current thread $\mathsf{thread}(sa)$. From now on we will only be interested in single-threaded strategies. We write $\sigma : A$ to indicate that σ is a single-threaded strategy for A. We will also refer to the ***single-threaded plays*** of a strategy, meaning those plays which only contain one initial move. Clearly a single-threaded strategy is determined by its set of single-threaded plays.

3.3 Constructions on arenas

Given arenas A and B, the arenas $A \times B$ and $A \Rightarrow B$ are defined as follows.

$$\begin{array}{rcl} M_{A\times B} & = & M_A + M_B \text{ (disjoint union)} \\ \lambda_{A\times B} & = & [\lambda_A, \lambda_B] \\ \star \vdash_{A\times B} a & \iff & \star \vdash_A a \vee \star \vdash_B a \\ a \vdash_{A\times B} b & \iff & a \vdash_A b \vee a \vdash_B b \end{array}$$

$$\begin{array}{rcl} M_{A\Rightarrow B} & = & M_A + M_B \\ \lambda_{A\Rightarrow B} & = & [\overline{\lambda}_A, \lambda_B] \end{array}$$

$$\star \vdash_{A \Rightarrow B} a \iff \star \vdash_B a$$
$$a \vdash_{A \Rightarrow B} b \iff a \vdash_A b \vee a \vdash_B b \vee [\star \vdash_B a \wedge \star \vdash_A b]$$

The unit for $\times$ is the empty arena $\mathbf{1} = \langle \emptyset, \emptyset, \emptyset \rangle$.

3.4 The category $\mathcal{C}$

We define a category $\mathcal{C}$ as follows.

$$\text{Objects} : \text{Arenas.}$$
$$\text{Morphisms } A \to B : \text{Single-threaded strategies for } A \Rightarrow B.$$

As usual in game semantics, identities are given by copycat strategies and composition by "parallel composition plus hiding": see [6], for example. $\mathcal{C}$ is then a cartesian closed category.

3.5 Interpretation of $IA_{\mathtt{mkvar}}$

We have defined a cartesian closed category $\mathcal{C}$ which provides a model for simply typed λ-calculus in the standard fashion [7]. We now complete the definition of the games models of IA and $IA_{\mathtt{mkvar}}$ by giving an interpretation for the base types and constants of the languages in $\mathcal{C}$.

For brevity, we will use the same name for the arena interpreting a base type as for the type itself; this should not lead to any confusion.

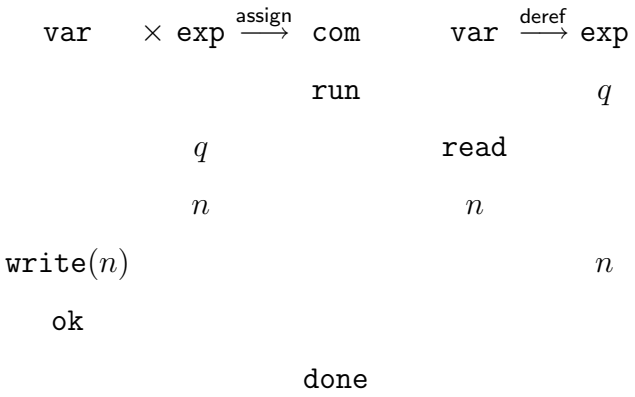
The arena `exp` is the familiar arena of natural numbers, defined as follows: it has a single initial question q to which P may respond with any natural number as an answer. The numeral n in IA or $IA_{\mathtt{mkvar}}$ is interpreted as the strategy which always responds to q with n.

The arena `com` is similar: there is a single initial question `run` and a single possible answer `done` to signal termination. The constant `skip` is interpreted as the strategy which always responds to `run` with `done`.

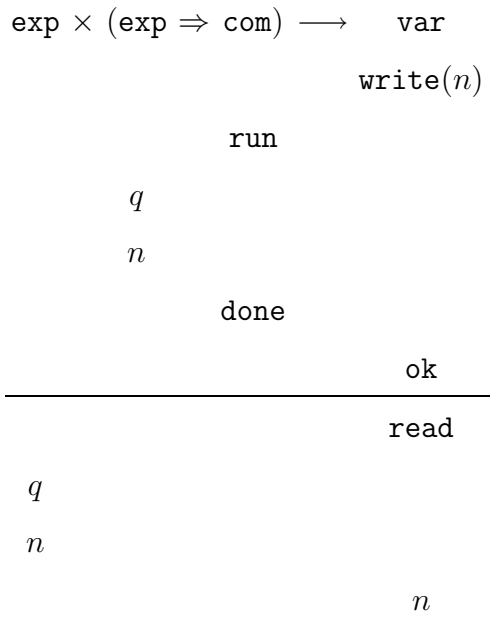
For `var`, we exploit Reynolds's idea of using a product of a "read method" and "write method" type: thus $\mathtt{var} = \mathtt{exp} \times \mathtt{com}^\omega$. Concretely, this game has a single initial question in the `exp`-component, which we write as `read`, to which P can answer with any natural number; and ω-many initial questions in the $\mathtt{com}^\omega$ part, which we write as $\mathtt{write}(n)$, to which P can respond with a single answer `ok`.

Assignment and dereferencing are interpreted using the strategies depicted

below.



`mkvar` is modelled using a strategy whose typical plays are



A typical legal position in the arena `var` is $\texttt{read}\cdot 3\cdot\texttt{write}(4)\cdot\texttt{ok}\cdot\texttt{write}(9)\cdot\texttt{ok}\cdot\texttt{read}\cdot 2\cdots$. Note in particular that the arena itself imposes no relationship between values written in and values read out. We say that such a play has the ***good variable property*** (with initial value 0) if the answer provided to a `read` is always the last value written, or 0 if no $\texttt{write}(n)$ has yet been played.

Define a map $\texttt{new} : (\texttt{var} \Rightarrow \texttt{com}) \to \texttt{com}$ to be the strategy whose single-

threaded plays are those of the form (an even length prefix of)

$$\begin{array}{rcl} (\mathtt{var} \Rightarrow \mathtt{com}) & \rightarrow & \mathtt{com} \\ & & \mathtt{run} \\ & \mathtt{run} & \\ s & & \\ & \mathtt{done} & \\ & & \mathtt{done} \end{array}$$

where s is a legal play in the `var` component which has the good variable property. Given a term $\Gamma, x : \mathtt{var} \vdash M : \mathtt{com}$ of $IA_{\mathtt{mkvar}}$, we can then define

$$[\![\mathtt{new}\ x\ \mathtt{in}\ M]\!] = \mathtt{new}([\![\lambda x.M]\!]) : \Gamma \rightarrow \mathtt{com}.$$

4 Full Abstraction for $IA_{\mathtt{mkvar}}$

We now briefly review the structure of the full abstraction proof for $IA_{\mathtt{mkvar}}$ which appeared previously in [3]. We begin by defining a preorder on strategies.

Definition 4.1 If σ and τ are strategies for an arena A, then $\sigma \sqsubseteq_m \tau$ if and only if

$$\forall \text{ complete } s \in \sigma . s \in \tau$$

That is, the complete plays of σ are contained in τ. Note that the equivalence relation induced by this preorder relates those strategies whose complete plays are identical.

Lemma 4.2 The strategy interpreting `skip` is maximal with respect to $\sqsubseteq_m$ on the arena `com`.

Lemma 4.3 Composition of strategies is monotone with respect to $\sqsubseteq_m$: if $\sigma \sqsubseteq_m \sigma' : A \rightarrow B$ and $\tau \sqsubseteq_m \tau' : B \rightarrow C$, then $\sigma ; \tau \sqsubseteq_m \sigma' ; \tau'$.

Lemma 4.4 (Soundness and Adequacy) Given a term $\vdash M : \mathtt{com}$ of $IA_{\mathtt{mkvar}}$, $M{\Downarrow} \iff [\![M]\!] = [\![\mathtt{skip}]\!]$.

Theorem 4.5 (Inequational Soundness) If $\Gamma \vdash M, N : T$ are terms of $IA_{\mathtt{mkvar}}$ and $[\![M]\!] \sqsubseteq_m [\![N]\!]$ then $M \underset{\sim}{\sqsubseteq}_m N$.

Proof Suppose $[\![M]\!] \sqsubseteq_m [\![N]\!]$ and let $C[-]$ be a closing context such that $C[M]{\Downarrow}$. By Lemma 4.4, $[\![C[M]]\!] = [\![\mathtt{skip}]\!]$. By the compositionality of the semantics together with Lemma 4.3, we know that $[\![C[M]]\!] \sqsubseteq_m [\![C[N]]\!]$, so $[\![C[N]]\!] = [\![\mathtt{skip}]\!]$ by maximality of $[\![\mathtt{skip}]\!]$. Using Lemma 4.4 again we have that $C[N]{\Downarrow}$ as required. □

The converse of this soundness result, completeness, depends upon the following *definability* property.

Lemma 4.6 (Definability) Let A be an arena interpreting a type of $IA_{\texttt{mkvar}}$ and let s be any complete play of A. There exists an $IA_{\texttt{mkvar}}$ term

$$x : A \vdash M : \texttt{com}$$

such that the only single-threaded complete play of $[\![M]\!]$ is $\texttt{run} \cdot s \cdot \texttt{done}$.

This is a special case of the definability result proved in [3], which shows that every finite strategy is the denotation of a term of $IA_{\texttt{mkvar}}$. We state this special case here because it is sufficient for our purposes and because it is closer to the analogous result we will prove for the language without `mkvar`. Note that, for this lemma, the fact that `mkvar` may be used in the term M is crucial.

Theorem 4.7 (Completeness) If $\Gamma \vdash M, N : T$ are terms of $IA_{\texttt{mkvar}}$ and $M \sqsubseteq_m N$ then $[\![M]\!] \sqsubseteq_m [\![N]\!]$.

Proof Suppose $M \sqsubseteq_m N$ and let s be any complete play of $[\![M]\!]$. We must show s is contained in $[\![N]\!]$.

By definability, there is a term $x : \Gamma \Rightarrow T \vdash C[x] : \texttt{com}$ which has $\texttt{run} \cdot s \cdot \texttt{done}$ as its only complete single-threaded play. Since $s \in [\![M]\!]$ it is easy to see that this implies that $[\![C[M]]\!] = [\![\texttt{skip}]\!]$. Therefore, by adequacy, $C[M]\Downarrow$ and hence $C[N]\Downarrow$, so by soundness, $[\![C[N]]\!] = [\![\texttt{skip}]\!]$. This is only possible if $[\![N]\!]$ contains a play t such that $\texttt{run} \cdot t \cdot \texttt{done} \in [\![C[x]]\!]$. Since $[\![C[x]]\!]$ has only one complete single-threaded play, this implies that $s \in [\![N]\!]$ as required. □

Putting Theorems 4.5 and 4.7 together gives the full abstraction result.

5 The language without `mkvar`

In this section we study the expressive power of contexts written without the use of `mkvar`, and show that the absence of `mkvar` has no impact on observational equivalence of programs, but does affect the observational preorder. This result is obtained by constructing a fully abstract model of `mkvar`-free IA, which consists of the same strategies used for the model of $IA_{\texttt{mkvar}}$ but with a different preorder.

To motivate the new preorder, let us consider the example from the introduction. The only complete single-threaded play of

$$[\![\texttt{if } !x = 3 \texttt{ then skip else diverge}]\!]$$

is

$$\begin{array}{rcl} \texttt{var} & \to & \texttt{com} \\ & & \texttt{run} \\ \texttt{read} & & \\ 3 & & \\ & & \texttt{done} \end{array}$$

while the only complete, single-threaded play of $[\![x := 3]\!]$ is

$$\begin{array}{rcl} \texttt{var} & \to & \texttt{com} \\ & & \texttt{run} \\ \texttt{write}(3) & & \\ \texttt{ok} & & \\ & & \texttt{done}. \end{array}$$

Thus we see that in strategies corresponding to terms related by $\underset{\sim}{\sqsubset}$, some places where P plays a $\texttt{read}$ move in one strategy can see a $\texttt{write}(-)$ in the other strategy. The following definitions set up a preorder on strategies which reflects this possibility.

Definition 5.1 Let A be a game interpreting an IA type and $s, t \in L_A$ be two plays of A. We write $s \lhd_{\mathsf{O}} t$ iff

$$s = s_1 \cdot \overset{\curvearrowleft}{\texttt{read}} \cdot s_2 \cdot n \cdot s_3$$

for some sequences s_1, s_2 and s_3, where $\texttt{read}$ is an O-move being the read-move in any occurrence of $\texttt{var}$ in the type A, n is the answer to the specified $\texttt{read}$, and

$$t = s_1 \cdot \overset{\curvearrowleft}{\texttt{write}(n)} \cdot s_2 \cdot \texttt{ok} \cdot s_3.$$

where the $\texttt{write}(n)$ is understood to be played in the same occurrence of $\texttt{var}$ as the $\texttt{read}$ in s.

We write $s \lhd_{\mathsf{P}} t$ for the analogous relation where the replaced $\texttt{read}$ is a P-move.

Let $\propto_{\mathsf{O}}$ be the reflexive, transitive closure of the relation $\lhd_{\mathsf{O}}$, and similarly for $\propto_{\mathsf{P}}$.

Thus $s \propto_{\mathsf{O}} t$ if and only if t can be obtained from s by replacing some segments of the form $\texttt{read} \ldots n$ with $\texttt{write}(n) \ldots \texttt{ok}$.

Definition 5.2 Given two strategies σ and τ for the same arena, $\sigma \sqsubseteq \tau$ iff

$$\forall \text{ complete } s \in \sigma. \exists t \in \tau. s \propto_{\mathsf{P}} t.$$

5.1 Definability, Completeness, Conservativity

We now set about proving a definability result similar to Lemma 4.6. This is essentially a programming task: for any play s we seek a term M which can test for this play, up to the $\propto_{\mathsf{O}}$ relation. The idea is that the term M will have access to some variables $x_1, \ldots, x_n$ in which it records information about the moves O plays. Thus M will contain some code designed to allow it to play the moves in the sequence s, and some *profiling code* which stores information in these extra variables and allows us to trap any deviations which O may make from the "script" given by s.

Definition 5.3 Let $x_1 : \mathtt{var}, \ldots, x_n : \mathtt{var} \vdash M : A$ be an IA term, and let s_1, s_2 be states over the variables x_i. The triple (M, s_1, s_2) is said to *accept* a play $s \in L_A$ iff there is a *complete* play $t \in [\![M]\!]$ such that

- O's play in the $\mathtt{var}$ components is that of a tuple of storage cells initialized with state s_1
- P's play in the $\mathtt{var}$ components is such that the last values written into the x_is leave state s_2
- $t \upharpoonright A = s$.

Thus for example the triple $(x \vdash x :=!x+1, x \mapsto 0, x \mapsto 1)$ accepts only the play $\mathtt{run} \cdot \mathtt{done}$: using x as a counter we are able to keep track of the number of times O plays the initial $\mathtt{run}$, so that the empty play or a play containing two occurrences of $\mathtt{run}$ is not accepted. The same term with starting state $x \mapsto 0$ and finishing state $x \mapsto 2$ accepts only the play $\mathtt{run} \cdot \mathtt{done} \cdot \mathtt{run} \cdot \mathtt{done}$.

The following transformation on plays gives us a hook onto which we can attach the profiling code we need.

Definition 5.4 Given a play $s \in L_A$, define $\mathsf{inst}(s) \in L_{\mathtt{com} \Rightarrow A}$ as follows:

$$\begin{aligned} \mathsf{inst}(\varepsilon) &= \varepsilon \\ \mathsf{inst}(s \cdot m) &= \mathsf{inst}(s) \cdot m && \text{if } m \text{ is a } \mathsf{P}\text{-move} \\ \mathsf{inst}(s \cdot m) &= \mathsf{inst}(s) \cdot m \cdot \mathtt{run} \cdot \mathtt{done} && \text{if } m \text{ is an } \mathsf{O}\text{-move} \end{aligned}$$

where the $\mathtt{run}$ and $\mathtt{done}$ are in the newly added $\mathtt{com}$ component.

The intention of this definition is that $\mathsf{inst}(s)$ is an "instrumented" version of s in which P runs the new command after every move by O.

We can now state the key lemma which leads to our definability result.

Lemma 5.5 Let A be any IA type and $s \in L_A$ any complete play. There exists a term

$$\vec{x} : \mathtt{var}, c : \mathtt{com} \vdash M : A$$

and stores s_1 and s_2 such that $(\lambda c.M, s_1, s_2)$ accepts a play $t \in L_{\mathtt{com} \Rightarrow A}$ if and only if $\mathsf{inst}(s) \propto_{\mathsf{O}} t$.

Proof The proof is by induction on the length of s. The base case is straightforward; for example, if A is of the form $B \Rightarrow \mathtt{exp}$ then

$$x : \mathtt{var}, c : \mathtt{com} \vdash \lambda b : B.(x := 1; 0)$$

with starting state $x \mapsto 0$ and finishing state $x \mapsto 0$ does the job: only the empty play is accepted because any non-empty play results in 1 being written into x.

The case in which s has length 2 is illustrative.

- If A is of the form $B \Rightarrow \mathtt{exp}$ and s is, for example, $q \cdot 3$, then the term

  $$x : \mathtt{var}, c : \mathtt{com} \vdash \lambda b : B.(x :=!x + 1; c; 3)$$

  with starting state $x \mapsto 0$ and finishing state $x \mapsto 1$ fulfils the requirements. The use of variable x ensures that the question q is played exactly once, and the rest of the term provides the appropriate behaviour for P.
- If A is of the form $B \Rightarrow \mathtt{com}$ and s is $\mathtt{run} \cdot \mathtt{done}$, then the term

  $$x : \mathtt{var}, c : \mathtt{com} \vdash \lambda b : B.(x :=!x + 1; c)$$

  with starting state $x \mapsto 0$ and finishing state $x \mapsto 1$ fulfils the requirements.
- If A is of the form $B \Rightarrow \mathtt{var}$ and s is $\mathtt{write}(3) \cdot \mathtt{ok}$, the term is

  $$x : \mathtt{var}, y : \mathtt{var}, c : \mathtt{com} \vdash \lambda b : B.(x :=!x + 1; c; y)$$

  with starting state $(x \mapsto 0, y \mapsto 0)$ and finishing state $(x \mapsto 1, y \mapsto 3)$. This works in the same way as the above examples, but note the use of a variable y to receive and trap the value written by O: if O's first move is $\mathtt{read}$ or any other $\mathtt{write}(n)$, y will hold the wrong value at the end.
- Finally, if A is of the form $B \Rightarrow \mathtt{var}$ and s is $\mathtt{read} \cdot 3$, the term is

  $$x : \mathtt{var}, y : \mathtt{var}, c : \mathtt{com} \vdash \lambda b : B.(x :=!x + 1; c; y)$$

  with starting state $(x \mapsto 0, y \mapsto 3)$ and finishing state $(x \mapsto 1, y \mapsto 3)$. Here y must be initialized with the value 3 which must be provided when O reads from the term; but notice that the same final state results if instead of performing a read, O plays $\mathtt{write}(3)$. If O writes any other value, that value will be stored in y so the play will not be accepted. Hence this term accepts all those plays t such that $\mathsf{inst}(s) \propto_{\mathsf{O}} t$.

For the inductive step, we must examine the form of longer plays and apply the decomposition techniques familiar from definability proofs in game semantics. We shall first deal with the case of single-threaded plays. A single-threaded play of a game such as $\Gamma \times (A \Rightarrow B) \Rightarrow C$, where B and C are base

types, has the form

$$\Gamma \times (A \Rightarrow B) \Rightarrow C$$

$$q$$

$$q$$

$$s$$

$$a$$

$$t$$

and by suitable relabelling of moves, s can be seen as a complete play of $\Gamma \times (A \Rightarrow B) \Rightarrow A$, and $q \cdot t$ is a complete play of $\Gamma \times (A \Rightarrow B) \Rightarrow C$.

By the inductive hypothesis we can find a term

$$\vec{x} : \mathtt{var}, c : \mathtt{com} \vdash \lambda(g : \Gamma, f : A \Rightarrow B).M : \Gamma \times (A \Rightarrow B) \Rightarrow A$$

and starting state s_1 and finishing state s_2 (over the variables $\vec{x}$), which accepts a play u iff $\mathsf{inst}(s) \propto_{\mathsf{O}} u$.

Similarly there is a term

$$\vec{y} : \mathtt{var}, c : \mathtt{com} \vdash \lambda(g, f).N : \Gamma \times (A \Rightarrow B) \Rightarrow C$$

and starting state t_1 and finishing state t_2 (over the variables $\vec{y}$), which accepts a play v iff $\mathsf{inst}(q \cdot t) \propto_{\mathsf{O}} v$.

Supposing for example that B is $\mathtt{var}$ and the moves q and a in B as illustrated above are $\mathtt{read}$ and 3, then the term

$$\vec{x}, \vec{y} : \mathtt{var}, c : \mathtt{com} \vdash \lambda(g, f).c; \mathtt{if}\ !(fM) = 3\ \mathtt{then}\ N\ \mathtt{else}\ \Omega$$

with starting state $s_1 \otimes t_1$ and finishing state $s_2 \otimes t_2$, where $\otimes$ denotes disjoint union of states, accepts only the required plays. For other types B and other question and answer moves in B, mild alterations to this term are necessary; we omit further details.

Finally, we must consider the inductive step in the case of a multi-threaded play. A multi-threaded play $s = i \cdot s'$ consists of $s_1 = s \upharpoonright i$, the first thread which is begun, interleaved with $s_2 = s \setminus s_1$, the other threads. If s is genuinely multi-threaded, each of s_1 and s_2 is shorter than s so by the inductive hypothesis we have terms and states

$$\vec{x} : \mathtt{var}, c_1 : \mathtt{com} \vdash \lambda\vec{z}.M : A, t_1, t_2$$

and

$$\vec{y} : \mathtt{var}, c_2 : \mathtt{com} \vdash \lambda\vec{z}.N : A, u_1, u_2$$

which accept only $\mathsf{inst}(s_1)$ and $\mathsf{inst}(s_2)$, and plays related to them by $\propto_{\mathsf{O}}$, respectively. The term

$$\vec{x}, \vec{y}, w : \mathtt{var}, c_1, c_2 : \mathtt{com} \vdash \lambda\vec{z}.\mathtt{if}\ !w = 0\ \mathtt{then}\ w := 1; M\ \mathtt{else}\ N : A$$

with starting state $t_1 \otimes u_1 \otimes (w \mapsto 0)$ and finishing state $t_2 \otimes u_2 \otimes (w \mapsto 1)$ then accepts any interleaving of such plays.

To complete the proof we must show that it is possible to restrict the accepted plays to those for which the interleaving of s_1 and s_2 moves matches that in s. Note that after each O-move in A, P plays $\mathtt{run}$ in the $\mathtt{com}$ component corresponding to c_1 if O was playing an s_1 move, and c_2 otherwise. Thus the play in these two $\mathtt{com}$ components reveals the interleaving of s_1 and s_2 which O is playing out. If we replace c_1 with $v :=!v \times 2; c$ and c_2 with $v := (!v \times 2)+1; c$, where $v : \mathtt{var}$ and $c : \mathtt{com}$ are fresh variables, this interleaving is encoded in variable v, and c provides the instrumentation hook required by the inductive hypothesis. We can now add to the starting state the requirement that $v \mapsto 1$, and to the finishing state $v \mapsto n$, where n is a number reflecting the appropriate interleaving, and obtain a term and states which accept only the required interleaving, completing the proof. □

Lemma 5.6 (Definability) Let A be an arena interpreting a type of IA and let s be any complete play of A. There exists an IA term $x : A \vdash M : \mathtt{com}$ such that the only single-threaded complete plays of $[\![M]\!]$ are those of the form $\mathtt{run} \cdot t \cdot \mathtt{done}$, where $s \propto_{\mathsf{P}} t$.

Proof By the previous Lemma, there is a term

$$\vec{x} : \mathtt{var}, c : \mathtt{com} \vdash M : A \Rightarrow \mathtt{com}$$

and starting state s_1 and finishing state s_2 such that M, s_1, s_2 accepts a complete play u iff $\mathsf{inst}(\mathtt{run} \cdot s \cdot \mathtt{done}) \propto_{\mathsf{O}} u$.

Then $M[\mathtt{skip}/c], s_1, s_2$ accepts u iff $\mathtt{run} \cdot s \cdot \mathtt{done} \propto_{\mathsf{O}} u$. Therefore, the term

$$\lambda a : A.\mathtt{new}\ \vec{x} := s_1\ \mathtt{in}(M[\mathtt{skip}/c](a); \mathtt{if}\ \vec{x} = s_2\ \mathtt{then}\ \mathtt{skip}\ \mathtt{else}\ \Omega)$$

(using obvious syntactic sugar for the initialization and final checking of the variables $\vec{x}$) has $\mathtt{run} \cdot t \cdot \mathtt{done}$ as a complete play iff $\mathtt{run} \cdot s \cdot \mathtt{done} \propto_{\mathsf{O}} \mathtt{run} \cdot t \cdot \mathtt{done}$, which is to say that $s \propto_{\mathsf{P}} t$. □

This lemma allows us to show that the semantics of IA with the preorder $\sqsubseteq$ is *complete* for the observational preorder on IA (note that we have not yet established soundness!) and that the extension of IA with $\mathtt{mkvar}$ conservative for observational equivalence, as follows.

Theorem 5.7 (Completeness) If $\Gamma \vdash M, N : T$ are terms of IA and $M \underset{\sim}{\sqsubseteq} N$ then $[\![M]\!] \sqsubseteq [\![N]\!]$.

Proof The proof is analogous to that of Theorem 4.7, using our new definability result. If $s \in [\![M]\!]$ is any complete play, by definability we can find a context $C[-]$ such that the complete plays of $[\![C[x]]\!]$ are just those of the form $\mathtt{run} \cdot t \cdot \mathtt{done}$ where $s \propto_{\mathsf{P}} t$. Then $[\![C[M]]\!] = [\![\mathtt{skip}]\!]$, so by adequacy $C[M]\Downarrow$ and hence also $C[N]\Downarrow$. This implies that there is some $t \in [\![N]\!]$ such that $\mathtt{run} \cdot t \cdot \mathtt{done} \in [\![C[x]]\!]$ and hence there is $t \in [\![N]\!]$ with $s \propto_{\mathsf{P}} t$ as required. □

Corollary 5.8 (Conservativity) If M and N are equivalent in IA ($M \cong N$), they are equivalent in $IA_{\mathtt{mkvar}}$ ($M \cong_m N$).

Proof If M and N are equivalent in IA, by the above we have $[\![M]\!] \sqsubseteq [\![N]\!]$ and $[\![N]\!] \sqsubseteq [\![M]\!]$. We shall show that M and N have the same complete plays, from which the result follows using the full abstraction of the semantics of $IA_{\mathtt{mkvar}}$.

Let $s \in [\![M]\!]$ be any complete play. Since $[\![M]\!] \sqsubseteq [\![N]\!]$, there exists $t \in [\![N]\!]$ such that $s \propto_{\mathsf{P}} t$. Since $[\![N]\!] \sqsubseteq [\![M]\!]$, there is some $u \in [\![M]\!]$ such that $t \propto_{\mathsf{P}} u$.

We therefore have $s, u \in [\![M]\!]$ with $s \propto_{\mathsf{P}} u$. If these are not identical, the first place they differ must be a P-move, which is impossible since $[\![M]\!]$ is a deterministic strategy. Hence $s = u$, from which it follows that $s = t$, so $s \in [\![N]\!]$ as required. Applying a symmetric argument we can conclude that $[\![M]\!]$ and $[\![N]\!]$ have identical complete plays. □

Since the converse of this corollary is immediate, we now know that the theories of observational equivalence of IA and $IA_{\mathtt{mkvar}}$ coincide.

5.2 Soundness

We shall now show that the preorder $\sqsubseteq$ is also *sound* for the observational preorder on IA terms, and hence that it captures $\sqsubseteq\!\!\!\!\sim$ precisely, so that we have an inequationally fully abstract model of IA.

We begin by identifying an important property enjoyed by all strategies interpreting terms of IA, but not by $[\![\mathtt{mkvar}]\!]$.

Definition 5.9 A strategy σ is $\propto$***-closed*** iff for every complete $s \in \sigma$ and every play t such that $s \propto_{\mathsf{O}} t$, there exists $u \in \sigma$ such that $t \propto_{\mathsf{P}} u$.

Lemma 5.10 If $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$ are $\propto$-closed, so is $\sigma ; \tau$.

Proof We can only sketch the idea, due to lack of space. $\propto$-closure says that given a sequence s in a strategy, if O plays almost according to s but changes some $\mathtt{read} \cdots n$ sequences to $\mathtt{write}(n) \cdots \mathtt{ok}$, the strategy's response changes in a similar way. Suppose we have a sequence $s \in \sigma ; \tau$, coming from an interaction between $s_1 \in \sigma$ and $s_2 \in \tau$. If O changes some $\mathtt{read} \cdots n$ sequences in s_1, σ responds by doing the same. If any of these changes are in the B-component, they can be fed to τ as changes to s_2. τ now responds with similar changes, which we can feed back to σ and so on. Since each such change replaces a $\mathtt{read}$ by a $\mathtt{write}(n)$, and since there are only finitely many $\mathtt{read}$s in the original interaction, this process terminates, and we are left with a new interaction which witnesses the $\propto$-closure of $\sigma ; \tau$. □

Corollary 5.11 For any term M of IA, $[\![M]\!]$ is $\propto$-closed.

Proof Since we have just shown that $\propto$-closure is preserved by composition, and it is clearly also preserved by currying and pairing, we just need to check that all the strategies used in the semantics of IA are $\propto$-closed.

For the strategies corresponding to constants of the language this is trivial: only $\mathtt{new}$ gives O the chance to play any $\mathtt{read}$s, and if a sequence $s_1 \cdot \mathtt{read} \cdot 3 \cdot s_2 \in [\![\mathtt{new}]\!]$ then clearly we also have $s_1 \cdot \mathtt{write}(3) \cdot \mathtt{ok} \cdot s_2 \in [\![\mathtt{new}]\!]$ so $\propto$-closure holds.

The only other strategies used are copycat-style strategies, such as identities, projections and so on. Taking as an example the identity on `var`, a typical sequence of plays related by $\propto$ is:

`var`	$\Rightarrow$	`var`		`var`	$\Rightarrow$	`var`		`var`	$\Rightarrow$	`var`
		`read`				`write`(7)				`write`(7)
`read`			$\propto_{\mathsf{O}}$	`read`			$\propto_{\mathsf{P}}$	`write`(7)		
7				7				`ok`		
		7				`ok`				`ok`

In fact all $\propto_{\mathsf{O}}$-related sequences are of the form above, and the rightmost sequence is again a play in the identity strategy, so this typical case shows that the strategy is $\propto$-closed. $\square$

Lemma 5.12 If $\sigma \sqsubseteq \sigma' : A \to B$ and $\tau \sqsubseteq \tau' : B \to C$, and all these strategies are $\propto$-closed, then $\sigma ; \tau \sqsubseteq \sigma' ; \tau'$.

Proof The argument is similar to that for Lemma 5.10. $\square$

Theorem 5.13 (Inequational Soundness) Let $\Gamma \vdash M : A$ and $\Gamma \vdash N : A$ be terms of IA. If $[\![M]\!] \sqsubseteq [\![N]\!]$ then $M \underset{\sim}{\sqsubseteq} N$.

Proof First note that $[\![\texttt{skip}]\!]$ is a maximal strategy for the `com` arena with respect to $\sqsubseteq$; cf. Lemma 4.2. Since we have just shown that for all the strategies used in the semantics of IA, composition is monotone with respect to $\sqsubseteq$, we can use the same argument as for Theorem 4.5 to arrive at the result. $\square$

Since $\sqsubseteq$ is both sound and complete for $\underset{\sim}{\sqsubseteq}$, we have an inequationally fully abstract model of IA, whose notion of equivalence coincides with that for $IA_{\texttt{mkvar}}$.

6 Discussion

It is not clear to what extent the techniques introduced in this paper can be applied to model other language variants in the absence of `mkvar`. In the case of IA with passive expressions, the kind of fine analysis of the fully abstract model which we perform in this paper is not available, because the fully abstract model is obtained via a quotient. What we can say is that our conservativity result does not carry over to this language: for example, the equivalence

$$\lambda v : \texttt{var}.\texttt{if}\ !v = 3\ \texttt{then}\ v := 3 \cong \lambda v : \texttt{var}.\texttt{if}\ !v = 3\ \texttt{then}\ \texttt{skip}$$

which holds in the absence of `mkvar` because v cannot itself have side-effects, can be broken by binding v to the phrase $\texttt{mkvar}(3)(\lambda n : \texttt{exp}.y := 2)$.

It also seems doubtful that our proof technique could be applied to call-by-value languages. The key insight for this paper was that `mkvar`-free programs

give rise to $\propto$-closed strategies. We expect that this is still true for call-by-value, but one probably needs more: an expression of type `var` in a call-by-value language will evaluate either to a storage cell or a `var`-typed identifier. It seems likely that one would need to capture this as a stronger restriction on the behaviour of strategies in order to obtain an appropriate definability result.

It therefore appears that the result presented here is something of an anomaly; there are certainly more questions left to answer in this area.

7 Acknowledgments

The author has benefited from interesting discussions with many people over the time it took to develop this result, including Samson Abramsky, Peter O'Hearn, Jim Laird, Dan Ghica, and Matthew Wall. Three anonymous referees gave excellent feedback and found many technical and typographical errors. Thank you all!

This research was supported in part by the EPSRC research project "Semantic approaches to control of interference in higher-order programming language".

References

[1] S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. In *Proceedings, Thirteenth Annual IEEE Symposium on Logic in Computer Science*, pages 334–344. IEEE Computer Society Press, 1998.

[2] 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.

[3] 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 [9], pages 297–329 of volume 2.

[4] S. Abramsky and G. McCusker. Call-by-value games. In M. Nielsen and W. Thomas, editors, *Computer Science Logic: 11th International Workshop Proceedings*, Lecture Notes in Computer Science, pages 1–17. Springer-Verlag, 1998.

[5] S. Abramsky and G. McCusker. Full abstraction for Idealized Algol with passive expressions. *Theoretical Computer Science*, 227:3–42, 1999.

[6] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III. *Information and Computation*, 162(2):285–408, 2000.

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

[8] 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.

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

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

[11] A. Stoughton. Equationally fully abstract models of PCF. In M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Fifth International Conference on the Mathematical Foundations of Programming Semantics*, volume 442 of *Lecture Notes in Computer Science*. Springer Verlag, 1990.