Block structure vs scope extrusion: between innocence and omniscience

Andrzej S. Murawski[⋆] and Nikos Tzevelekos[⋆⋆]

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD, UK

Abstract. We study the semantic meaning of block structure using game semantics and introduce the notion of block-innocent strategies, which turns out to characterise call-by-value computation with block-allocated storage through soundness, finitary definability and universality results. This puts us in a good position to conduct a comparative study of purely functional computation, computation with block storage and dynamic memory allocation respectively. For example, we show that dynamic variable allocation can be replaced with block-allocated variables exactly when the term involved (open or closed) is of base type and that block-allocated storage can be replaced with purely functional computation when types of order two are involved. To illustrate the restrictive nature of block structure further, we prove a decidability result for a finitary fragment of call-by-value Idealized Algol for which it is known that allowing for dynamic memory allocation leads to undecidability.

1 Introduction

Most programming languages manage memory by employing a stack for local variables and heap storage for dynamically allocated data that may live beyond their initial context. A prototypical example of the former mechanism is Reynolds's Idealized Algol [16], in which local variables can only be introduced inside blocks of ground type. Memory is then allocated on entry to the block and deallocated on exit. In contrast, languages such as ML permit variables to escape from their current context under the guise of pointers or references. In this case, after memory is allocated at the point of reference creation, the variable is allowed to persist indefinitely (in practice, garbage collection or explicit deallocation is used to put an end to its life).

In this paper we would like to compare the expressivity of the two paradigms. As a simple example of heap-based memory allocation we consider the language RML, introduced by Abramsky and McCusker in [2], which is a fragment of ML featuring integer-valued references. They also constructed a fully abstract game model of RML based on strategies (also referred to as *knowing* strategies) that allow the player to base his decisions on the full history of play. On the other hand, at around the same time Honda and Yoshida [6] showed that the purely functional core of RML, better known as call-by-value PCF [14], corresponds to *innocent* strategies [7], i.e. those that can only rely on a

[⋆] Supported by an EPSRC Advanced Research Fellowship (EP/C539753/1).
[⋆⋆] Supported by EPSRC (EP/F067607/1).

restricted view of the play when deciding on the next move. Since block-structured storage of Idealized Algol seems less expressive than dynamic memory allocation of ML and more expressive than PCF, it is natural to ask about its exact position in the range of strategies between innocence and omniscience. Our first result is an answer to this question. We introduce the family of *block-innocent* strategies, situated strictly between innocent and knowing strategies. As a vehicle for our study we use a call-by-value variant $\mathsf{IA}_{\mathsf{cbv}}$ of Idealized Algol and prove that each $\mathsf{IA}_{\mathsf{cbv}}$-term can be interpreted by a block-innocent strategy (soundness), each finitary block-innocent strategy corresponds to an $\mathsf{IA}_{\mathsf{cbv}}$-term (finitary definability) and each recursively presentable block-innocent strategy corresponds to an $\mathsf{IA}_{\mathsf{cbv}}$-term (universality). Block-innocence captures the particular kind of uniformity exhibited by strategies originating from block-structured programs, akin to innocence yet strictly weaker. In fact, we define block-innocence through innocence in a setting enriched with explicit store annotations added to standard moves. For instance, in the play shown below[1], if P follows a block-innocent strategy, he is free to use different moves as the fourth and sixth moves, but the tenth one and the twelfth one have to be the same.

$$\begin{tikzcd}[column sep=tiny] q & q & q & 0 & q \arrow[lll, bend right] & 1 & a \arrow[lllll, bend right] & a \arrow[lllllll, bend right] & q & 0 & q \arrow[lll, bend right] & 0 \end{tikzcd}$$

Additionally, our framework detects "storage violations" resulting from an attempt to access a variable from outside of its block. For instance, no $\mathsf{IA}_{\mathsf{cbv}}$-term will ever produce the following play (the last move is the offending one).

$$\begin{tikzcd}[column sep=tiny] q & q & q & 0 & q \arrow[lll, bend right] & 1 & a \arrow[lllll, bend right] & a \arrow[lllllll, bend right] & q & q \arrow[lll, bend right] \end{tikzcd}$$

The notion of block-innocence provides us with a systematic methodology to address expressivity questions related to block structure such as "Does a given strategy originate from a stack-based memory discipline?" or "Can a given program using dynamic memory allocation be replaced with an equivalent program featuring stack-based storage?". To illustrate the approach we conduct a complete study of the relationship between the three classes of strategies according to type-thereotic order. We find that knowingness implies block-innocence when terms of base types (open or closed) are involved, that block-innocence implies innocence exactly for types of at most second order, and that knowingness implies innocence if the term is of base type and its free identifiers are of order 1.

As a further confirmation of the restrictive nature of the stack discipline of $\mathsf{IA}_{\mathsf{cbv}}$, we prove that program equivalence is decidable for a finitary variant of $\mathsf{IA}_{\mathsf{cbv}}$ which properly contains all second-order types as well as some third-order types (interestingly, our type discipline covers the available higher-order types in PASCAL). In contrast, the corresponding restriction of RML is known to be undecidable [10].

Related work. The stack discipline has always been regarded as part of the essence of Algol [16]. Accordingly, finding models embodying stack-oriented storage management has become an important goal of research into Algol-like languages. In this spirit, in the early 1980s, Reynolds [16] and Oles [11] devised a semantic model of Algol-like

[1] For the sake of clarity, we only include pointers pointing more than one move behind.

languages using a category of functors from a category of store shapes to the category of predomains. Perhaps surprisingly, in the 1990s Pitts and Stark [13, 17] managed to adapt the techniques to languages with dynamic allocation. This would appear to create a common platform suitable for a comparative study such as ours. However, despite the valuable structural insights, the relative imprecision of the functor category semantics (failure of definability and full abstraction) makes it unlikely that the results obtained by us can be proved via this route. The semantics of local effects has also been investigated from the category-theoretic point of view in [15].

As for the game semantics literature, Ong's work [12] based on strategies-with-state is the work closest to ours. His paper defines a compositional framework that is proved sound for the third-order fragment of call-by-name Idealized Algol. Adapting the results to call-by-value and all types is far from immediate. For a start, to handle higher-order types, we note that the state of O-moves is no longer determined by its justifier and the preceding move. Instead, the right state has to be computed "globally" using the whole history of play. However, the obvious adaptation of so modified framework to call-by-value does not capture the block-structure of $\mathsf{IA}_{\mathsf{cbv}}$. Quite the opposite: it seems to be more compatible with RML than $\mathsf{IA}_{\mathsf{cbv}}$! Consequently, further changes are needed to characterize $\mathsf{IA}_{\mathsf{cbv}}$. Firstly, to restore definability, the explicit stores have to become lists instead of sets. Secondly, conditions controlling state changes must be tightened. In particular, P must be forbidden from introducing fresh variables at any step and, in a similar vein, must be forced to drop some variables from his moves in certain circumstances.

The paper cited above is part of a series that has eventually led to a complete classification of the decidable cases of call-by-name (finitary) Idealized Algol. Much less is known about the call-by-value case, we are only aware of two papers: one by Ghica [5] and the other by the first-named author [10]. Both rely on regular languages to capture the game semantics of fragments of $\mathsf{IA}_{\mathsf{cbv}}$ and RML respectively. Their other common feature is that the types considered are selected in such a way that no pointers need to be represented in the induced plays. Our results represent further progress with regard to $\mathsf{IA}_{\mathsf{cbv}}$. The type system of our language, named $\mathsf{IA}^{2+}_{\circlearrowleft}$, is designed in such a way that only pointers from O-moves need not be represented, but we must include an explicit representation of pointers from certain P-moves. In particular, in contrast to [5], we can account for all second-order types, as we allow all first-order types to occur in contexts. "Curried" types of the form $A \to B \to C$ are especially tricky to handle here, because they can only be dealt with correctly if pointers from P-moves are encoded explicitly (recall that in the call-by-value setting $A \to B \to C$ and $A \times B \to C$ are not isomorphic). Any further extension of the type system of $\mathsf{IA}^{2+}_{\circlearrowleft}$ leads either to context-free languages or to plays in which pointers from O-moves of unbounded length would have to be handled, which seemingly requires an infinite alphabet.

2 Syntax

To set a common ground for our investigations, we introduce a higher-order programming language $\mathcal{L}$ that features syntactic constructs for both block and dynamic memory allocation. Its types are generated by the grammar below, where β ranges over the

$$\frac{\Gamma, x:\mathsf{var} \vdash M:\beta}{\Gamma \vdash \mathsf{new}\, x \,\mathsf{in}\, M:\beta} \qquad \frac{}{\Gamma \vdash \mathsf{ref} : \mathsf{var}} \qquad \frac{}{\Gamma \vdash () : \mathsf{unit}} \qquad \frac{i \in \mathbb{Z}}{\Gamma \vdash i : \mathsf{int}} \qquad \frac{(x:\theta) \in \Gamma}{\Gamma \vdash x : \theta}$$

$$\frac{\Gamma \vdash M_1 : \mathsf{int} \qquad \Gamma \vdash M_2 : \mathsf{int}}{\Gamma \vdash M_1 \oplus M_2 : \mathsf{int}} \qquad \frac{\Gamma \vdash M : \mathsf{int} \quad \Gamma \vdash N_0 : \theta \quad \Gamma \vdash N_1 : \theta}{\Gamma \vdash \mathsf{if}\, M \,\mathsf{then}\, N_1 \,\mathsf{else}\, N_0 : \theta}$$

$$\frac{\Gamma \vdash M : \mathsf{var}}{\Gamma \vdash !M : \mathsf{int}} \qquad \frac{\Gamma \vdash M : \mathsf{var} \quad \Gamma \vdash N : \mathsf{int}}{\Gamma \vdash M := N : \mathsf{unit}} \qquad \frac{\Gamma \vdash M : \mathsf{unit} \to \mathsf{int} \quad \Gamma \vdash N : \mathsf{int} \to \mathsf{unit}}{\Gamma \vdash \mathsf{mkvar}(M, N) : \mathsf{var}}$$

$$\frac{\Gamma \vdash M : \theta \to \theta' \quad \Gamma \vdash N : \theta}{\Gamma \vdash MN : \theta'} \qquad \frac{\Gamma, x : \theta \vdash M : \theta'}{\Gamma \vdash \lambda x^{\theta}.M : \theta \to \theta'} \qquad \frac{\Gamma \vdash M : (\theta \to \theta') \to (\theta \to \theta')}{\Gamma \vdash \mathsf{Y}(M) : \theta \to \theta'}$$

ground types unit and int.

$$\theta ::= \quad \beta \quad | \quad \mathsf{var} \quad | \quad \theta \to \theta$$

The syntax of $\mathcal{L}$ is given in the Figure above. Note in particular the first two rules concerning variables. The order of a type is defined as follows: $\mathsf{ord}(\beta) = 0$, $\mathsf{ord}(\mathsf{var}) = 1$, $\mathsf{ord}(\theta_1 \to \theta_2) = \max(\mathsf{ord}(\theta_1) + 1, \mathsf{ord}(\theta_2))$. For any $i \geq 0$, terms that are typable using exclusively judgments of the form $x_1 : \theta_1, \cdots, x_n : \theta_n \vdash M : \theta$, where $\mathsf{ord}(\theta_j) < i$ $(1 \leq j \leq n)$ and $\mathsf{ord}(\theta) \leq i$, are said to form the ith-order fragment. To spell out the operational semantics of $\mathcal{L}$, we need to assume a countable set Loc of *locations*, which are added to the syntax as auxiliary constants of type var. We shall write α to range over them. The semantics then takes the form of judgments $s, M \Downarrow s', V$, where s, s' are finite partial functions from Loc to integers, M is a term and V is a value. Terms of the following shapes are values: $()$, integer constants, elements of Loc, λ-abstractions or terms of the form $\mathsf{mkvar}(\lambda x^{\mathsf{unit}}.M, \lambda y^{\mathsf{int}}.N)$. Here we only reproduce the two evaluation rules related to variable creation.

$$\frac{s \cup (\alpha \mapsto 0), M[\alpha/x] \Downarrow s', V}{s, \mathsf{new}\, x \,\mathsf{in}\, M \Downarrow s' \setminus \alpha, V}\ {\scriptstyle \alpha \notin \mathsf{dom}\, s} \qquad \frac{}{s, \mathsf{ref} \Downarrow s \cup (\alpha \mapsto 0), \alpha}\ {\scriptstyle \alpha \notin \mathsf{dom}\, s}$$

$s' \setminus \alpha$ is the restriction of s' to $\mathsf{dom}\, s' \setminus \{\alpha\}$. The former rule encapsulates the state within the newly created block, while the latter creates a reference to a new memory cell that can be passed around without restrictions on its scope.

Given a closed term $\vdash M : \mathsf{unit}$, we write $M \Downarrow$ if there exists s such that $\emptyset, M \Downarrow s, ()$. We shall call two programs equivalent if they behave identically in every context. This is captured by the following definition, parameterized by the kind of contexts that are considered, to allow for testing of terms with contexts originating from a designated subset of the language.

Definition 1. *Suppose $\mathcal{L}'$ is a subset of $\mathcal{L}$. We say that the terms-in-context $\Gamma \vdash M_1, M_2 : \theta$ are $\mathcal{L}'$-equivalent (written $\Gamma \vdash M_1 \cong_{\mathcal{L}'} M_2$) if, for any $\mathcal{L}'$-context $C[-]$ such that $\vdash C[M_1], C[M_2] : \mathsf{unit}$, $C[M_1] \Downarrow$ if and only if $C[M_2] \Downarrow$.*

We shall study three sublanguages of $\mathcal{L}$ called PCF^+, $\mathsf{IA}_{\mathsf{cbv}}$ and RML. The latter two have appeared in the literature as paradigmatic examples of programming languages with stack discipline and dynamic memory allocation respectively.

$$\begin{aligned} M_{A\Rightarrow B} &= I_{A\Rightarrow B} \uplus I_A \uplus \overline{I}_A \uplus M_B & M_{A\otimes B} &= I_{A\otimes B} \uplus \overline{I}_A \uplus \overline{I}_B,\;\; I_{A\otimes B} = I_A \times I_B \\ I_{A\Rightarrow B} &= \{*\} & \lambda_{A\otimes B} &= [((i_A, i_B), PA), \lambda_A \upharpoonright \overline{I}_A, \lambda_B \upharpoonright \overline{I}_B] \\ \lambda_{A\Rightarrow B} &= [(*, PA), (i_A, OQ), \bar{\lambda}_A \upharpoonright \overline{I}_A, \lambda_B] & \vdash_{A\otimes B} &= \{((i_A, i_B), m) \mid i_A \vdash_A m \vee i_B \vdash_B m\} \\ \vdash_{A\Rightarrow B} &= \{(*, i_A), (i_A, i_B)\} \cup \vdash_A \cup \vdash_B & &\quad \cup (\vdash_A \upharpoonright \overline{I}_A{}^2) \cup (\vdash_B \upharpoonright \overline{I}_B{}^2) \end{aligned}$$

– PCF^+ is a purely functional language obtained from $\mathcal{L}$ by removing $\mathsf{new}\, x\, \mathsf{in}\, M$ and ref. It extends the language PCF [14] with primitives for variable access, but not for memory allocation.
– $\mathsf{IA}_{\mathsf{cbv}}$ is $\mathcal{L}$ without the ref constant. It can be viewed as a call-by-value variant of Idealized Algol [16]. Only block-allocated storage is available in $\mathsf{IA}_{\mathsf{cbv}}$.
– RML is $\mathcal{L}$ save the construct $\mathsf{new}\, x\, \mathsf{in}\, M$. It is exactly the language introduced in [2] as a prototypical language for ML-like integer references.

We shall often write $\mathsf{let}\, x = M\, \mathsf{in}\, N$ instead of $(\lambda x.N)M$. Note that, since $\mathsf{new}\, x\, \mathsf{in}\, M$ is equivalent to $\mathsf{let}\, x = \mathsf{ref}\, \mathsf{in}\, M$, RML and $\mathcal{L}$ merely differ on a syntactic level in that $\mathcal{L}$ contains "syntactic sugar" for blocks. In the opposite direction, our results will show that ref cannot in general be replaced with an equivalent term that uses $\mathsf{new}\, x\, \mathsf{in}\, M$. Indeed, our paper provides a general methodology for identifying and studying scenarios in which this expressivity gap occurs.

3 Game semantics

Here we introduce the game models used throughout the paper, which are based on the Honda-Yoshida approach to modelling call-by-value computation [6].

Definition 2. *An* **arena** $A = (M_A, I_A, \vdash_A, \lambda_A)$ *is given by*
– *a set M_A of moves, and a subset $I_A \subseteq M_A$ of* initial *moves,*
– *a justification relation $\vdash_A \subseteq M_A \times (M_A \setminus I_A)$, and*
– *a labelling function $\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$*

such that $\lambda_A(I_A) = \{PA\}$ and, whenever $m \vdash_A m'$, we have $(\pi_1 \lambda_A)(m) \neq (\pi_2 \lambda_A)(m')$ and $(\pi_2 \lambda_A)(m') = A \implies (\pi_2 \lambda_A)(m) = Q$.

The role of λ_A is to label moves as *Opponent* or *Proponent* moves and as *Questions* or *Answers*. We typically write them as $m, n, \ldots$, or $o, p, q, a, q_P, q_O, \ldots$ when we want to be specific about their kind. The simplest arena is $0 = (\emptyset, \emptyset, \emptyset, \emptyset)$. Other "flat" arenas are 1 and $\mathbb{Z}$, defined by $M_1 = I_1 = \{*\}$, $M_{\mathbb{Z}} = I_{\mathbb{Z}} = \mathbb{Z}$. The two standard constructions on arenas are presented in the figure above, where $\overline{I}_A$ stands for $M_A \setminus I_A$, the OP-complement of λ_A is written as $\bar{\lambda}_A$, and i_A, i_B range over initial moves in the respective arenas. Types of $\mathcal{L}$ can now be interpreted with arenas in the following way: $[\![\mathsf{unit}]\!] = 1$, $[\![\mathsf{int}]\!] = \mathbb{Z}$, $[\![\mathsf{var}]\!] = (1 \Rightarrow \mathbb{Z}) \otimes (\mathbb{Z} \Rightarrow 1)$ and $[\![\theta_1 \to \theta_2]\!] = [\![\theta_1]\!] \Rightarrow [\![\theta_2]\!]$. Although arenas model types, the actual games will be played in ***prearenas***, which are defined in the same way as arenas with the exception that initial moves must be O-questions. Given arenas A and B, we can construct the prearena $A \to B$ by setting:

$$\begin{aligned} M_{A\to B} &= M_A \uplus M_B & \lambda_{A\to B} &= [(i_A, OQ) \cup (\bar{\lambda}_A \upharpoonright \overline{I}_A)\;,\; \lambda_B] \\ I_{A\to B} &= I_A & \vdash_{A\to B} &= \{(i_A, i_B)\} \cup \vdash_A \cup \vdash_B\;\;. \end{aligned}$$

For $\Gamma = \{x_1 : \theta_1, \cdots, x_n : \theta_n\}$, typing judgments $\Gamma \vdash \theta$ will eventually be interpreted by strategies for the prearena $[\![\theta_1]\!] \otimes \cdots \otimes [\![\theta_n]\!] \to [\![\theta]\!]$ (if $n = 0$ we take the left-hand side to be 1), which we shall denote by $[\![\Gamma \vdash \theta]\!]$ or $[\![\theta_1, \cdots, \theta_n \vdash \theta]\!]$.

A ***justified sequence*** in a prearena A is a finite sequence s of moves of A satisfying the following condition: the first move must be initial, but all other moves m must be equipped with a pointer[2] to an earlier occurrence of a move m' such that $m' \vdash_A m$. A ***play*** in A is a justified sequence s satisfying the standard conditions of *Alternation*, *Well-Bracketing* and *Visibility* [7]. Visibility is based on the notions of *O-view* $\llcorner s \lrcorner$ and *P-view* $\ulcorner s \urcorner$ of a justified sequence s, given by: $\llcorner \epsilon \lrcorner = \epsilon$, $\llcorner s\, o \lrcorner = \llcorner s \lrcorner\, o$, $\llcorner s\, \overset{\frown}{o \cdots p} \lrcorner = \llcorner s \lrcorner\, o\, p$; $\ulcorner \epsilon \urcorner = \epsilon$, $\ulcorner s\, p \urcorner = \ulcorner s \urcorner\, p$, $\ulcorner s\, \overset{\frown}{p \cdots o} \urcorner = \ulcorner s \urcorner\, p\, o$. We write P_A to denote the set of plays in A.

Definition 3. *A* **(knowing) strategy** *σ on a prearena A, written $\sigma : A$, is a prefix-closed set of plays from A satisfying the first two conditions below. A strategy is* **innocent** *if, in addition, the third condition holds.*

O-CLOSURE *If even-length $s \in \sigma$ and $sm \in P_A$ then $sm \in \sigma$.*
DETERMINACY *If even-length $sm_1, sm_2 \in \sigma$ then $m_1 = m_2$.*
INNOCENCE *If $s_1 m, s_2 \in \sigma$ with odd-length s_1, s_2 and $\ulcorner s_1 \urcorner = \ulcorner s_2 \urcorner$ then $s_2 m \in \sigma$.*

Now we shall extend the framework to allow moves to be decorated with stores that contain *name*-integer pairs. The names should be viewed as semantic analogues of locations. When employing such moves-with-store, we are not interested in what exactly the names are, but we would like to know how they relate to names that have already been in play. Hence, the objects of study are rather the induced equivalence classes with respect to name-invariance, and all ensuing constructions and reasoning need to be compatible with it. This overhead can be dealt with robustly using the language of nominal set theory [4].

Let us fix a countably infinite set $\mathbb{A}$, the set of *names*, the elements of which we shall denote by α, β and variants. Consider the group PERM($\mathbb{A}$) of finite permutations of $\mathbb{A}$, denoted by π and variants. A ***strong nominal set*** [18] is a set equipped with a group action of PERM($\mathbb{A}$) such that each of its elements has *finite strong support*. That is to say, for any $x \in X$, there exists a finite set $\nu(x) \subseteq \mathbb{A}$, called ***the support of*** x, such that, for all permutations π, $(\forall \alpha \in \nu(x).\, \pi(\alpha) = \alpha) \iff \pi \cdot x = x$. Intuitively, $\nu(x)$ is the set of names "involved" in x. For example, the set $\mathbb{A}^{\#}$ of finite lists of distinct names with permutations acting elementwise is a strong nominal set. Name-variance in a strong nominal set X is represented by the relation: $x \sim x'$ if there exists π such that $x = \pi \cdot x'$.

We define a strong nominal set of ***stores***, the elements of which are finite sequences of name-integer pairs. Formally,

$$\Sigma, T ::= \quad \epsilon \quad | \quad (\alpha, i) :: \Sigma$$

where $i \in \mathbb{Z}$ and $\alpha \in \mathbb{A} \setminus \nu(\Sigma)$. We view stores as finite functions from names to integers, though their domains are lists rather than sets. Thus, we define the ***domain*** of a store to be the *list* of names obtained by applying the first projection to

[2] We then say that m' *justifies* m. If m is an answer, we might also say that m *answers* m'. If a question remains unanswered in s, it is *open*.

all of its elements. In particular, $\nu(\mathsf{dom}\,(\Sigma)) = \nu(\Sigma)$. If $\alpha \in \nu(\Sigma)$ then we write $\Sigma(\alpha)$ for the unique i such that (α, i) is an element of Σ. For stores Σ, T we write: $\Sigma \leq T$ for $\mathsf{dom}\,(\Sigma) \sqsubseteq \mathsf{dom}\,(T)$; $\Sigma \leq_p T$ for $\mathsf{dom}\,(\Sigma) \sqsubseteq_p \mathsf{dom}\,(T)$; $\Sigma \leq_s T$ for $\mathsf{dom}\,(\Sigma) \sqsubseteq_s \mathsf{dom}\,(T)$, where $\sqsubseteq, \sqsubseteq_p, \sqsubseteq_s$ denote the subsequence, prefix and suffix relations respectively. Note that $\Sigma \leq_{(p/s)} T \leq_{(p/s)} \Sigma$ implies $\mathsf{dom}\,(\Sigma) = \mathsf{dom}\,(T)$ but not $\Sigma = T$. Finally, let us write $\Sigma \setminus T$ for Σ restricted to $\nu(\Sigma) \setminus \nu(T)$.

An ***S-move*** (or *move-with-store*) in a prearena A is a pair consisting of a move and a store. We typically write S-moves as $m^{\Sigma}, n^{T}, o^{\Sigma}, p^{T}, q^{\Sigma}, a^{T}$. The first-projection function is viewed as *store erasure* and denoted by $\mathsf{erase}(\_)$. Note that moves contain no names and therefore, for any m^{Σ}, $\nu(m^{\Sigma}) = \nu(\Sigma) = \nu(\mathsf{dom}\,(\Sigma))$. A ***justified S-sequence*** in A is a sequence of S-moves equipped with justifiers, so that its erasure is a justified sequence. The notions of *O-view* and *P-view* are extended to S-sequences in the obvious manner. We say that a name α ***is closed*** in s if there are no open questions in s containing α.

Definition 4. *A justified S-sequence s in a prearena A is called an* **S-play**, *also written $s \in SP_A$, if it satisfies the following conditions, for all $\alpha \in \mathbb{A}$.*

INIT *If $s = m^{\Sigma} \cdots$ then $\Sigma = \epsilon$.*
JUST-P *If $s = \cdots o^{\Sigma} \overset{\curvearrowleft}{\cdots} p^{T} \cdots$ then $\Sigma \leq_p T$. If $\lambda_A(p) = PA$ then* $\mathsf{dom}\,(\Sigma) = \mathsf{dom}\,(T)$.
JUST-O *If $s = \cdots p^{\Sigma} \overset{\curvearrowleft}{\cdots} o^{T} \cdots$ then* $\mathsf{dom}\,(\Sigma) = \mathsf{dom}\,(T)$.
PREV-PQ *If $s = \cdots o^{\Sigma} q_P^{T} \cdots$ then $\Sigma \setminus T \leq_s \Sigma$ and $\Sigma \setminus (\Sigma \setminus T) \leq_p T$ and*
(a). *if $\alpha \in \nu(T \setminus \Sigma)$ then $\alpha \notin \nu(s_{<q_P^T})$,*
(b). *if $\alpha \in \nu(\Sigma \setminus T)$ then α is closed in $s_{<q_P^T}$.*
VAL-O *If $s = \cdots p^{\Sigma} s' o^{T} \cdots$ and $\alpha \in (\nu(T) \cap \nu(\Sigma)) \setminus \nu(s')$ then $T(\alpha) = \Sigma(\alpha)$.*

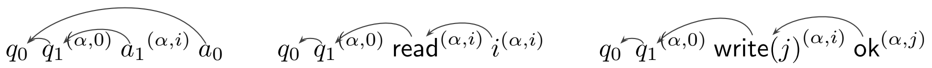
For example, PREV-PQ stipulates that P-questions may drop some names from the store and append some others, but these changes may only take place *in blocks* at the tail of the store. Moreover, appended names must be fresh in the whole play, and a name can be dropped only if it has been closed.

Let us remark that, as stores have strong support, the set of S-plays SP_A is a strong nominal set. Further properties of S-plays include:
- If $s = \cdots m^{\Sigma} a_P^{T} \cdots$ then $\Sigma \setminus T \leq_s \Sigma$ and $\Sigma \setminus (\Sigma \setminus T) \leq_p T$ and
  (a) if $\alpha \in \nu(T)$ then $\alpha \in \nu(\Sigma)$,
  (b) if $\alpha \in \nu(\Sigma \setminus T)$ then α is closed in $s_{<a_P^T}$.
- If $s = s_1 o^{\Sigma} p^{T} s_2$ with $\alpha \in \nu(\Sigma) \setminus \nu(T)$ then $\alpha \notin \nu(s_2)$.

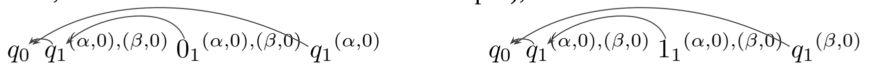
Definition 5. *An* **S-strategy** *σ on an arena A, written $\sigma : A$, is a prefix-closed set of S-plays from A satisfying the first three of the following conditions. An S-strategy is* **innocent** *if it also satisfies the last condition.*

NOMINAL CLOSURE *If $s' \sim s \in \sigma$ then $s' \in \sigma$.*
O-CLOSURE *If even-length $s \in \sigma$ and $sm^{\Sigma} \in SP_A$ then $sm^{\Sigma} \in \sigma$.*
DETERMINACY *If even-length $sm_1^{\Sigma_1}, sm_2^{\Sigma_2} \in \sigma$ then $sm_1^{\Sigma_1} \sim sm_2^{\Sigma_2}$.*
INNOCENCE *If $s_1 m^{\Sigma_1}, s_2 \in \sigma$ with s_1, s_2 odd-length and $\ulcorner s_1 \urcorner = \ulcorner s_2 \urcorner$ then there exists $s_2 m^{\Sigma_2} \in \sigma$ with $\ulcorner s_1 m^{\Sigma_1} \urcorner \sim \ulcorner s_2 m^{\Sigma_2} \urcorner$.*

Example 6. For any base type β, let us define the S-strategy $\mathsf{cell}_\beta : [\![\mathsf{var} \to \beta]\!] \to [\![\beta]\!]$ as the least innocent S-strategy containing the plays below. We use read and $\mathsf{write}(i)$ $(i \in \mathbb{Z})$ to refer to the question-moves of $[\![\mathsf{var}]\!]$, and i $(i \in \mathbb{Z})$ and ok for the non-initial answers.


$q_0\ q_1{}^{(\alpha,0)}\ a_1{}^{(\alpha,i)}\ a_0$
$q_0\ q_1{}^{(\alpha,0)}\ \mathsf{read}^{(\alpha,i)}\ i^{(\alpha,i)}$
$q_0\ q_1{}^{(\alpha,0)}\ \mathsf{write}(j)^{(\alpha,i)}\ \mathsf{ok}^{(\alpha,j)}$

Example 7. Had we used sets instead of lists for representing stores, the following "S-strategy", which represents incorrect overlap of scopes (α and β are in scope of one another, but at the same time have different scopes), would be innocent.


$q_0\ q_1{}^{(\alpha,0),(\beta,0)}\ 0_1{}^{(\alpha,0),(\beta,0)}\ q_1{}^{(\alpha,0)}$
$q_0\ q_1{}^{(\alpha,0),(\beta,0)}\ 1_1{}^{(\alpha,0),(\beta,0)}\ q_1{}^{(\beta,0)}$

Arenas and S-strategies form a category, which we call $\mathcal{S}$, and so do innocent S-strategies. $\mathcal{S}$ turns out to exhibit the same kind of categorical structure as that discussed in [6], which can be employed to model call-by-value higher-order computation with recursion. Thus, the functional part of $\mathsf{IA}_{\mathsf{cbv}}$ can be interpreted in $\mathcal{S}$ according to the standard recipe. Assignment, dereferencing and mkvar can in turn be modelled using the innocent strategies without stores from [2]. Finally, the denotation of $\mathsf{new}\,x\,\mathsf{in}\,M$ is obtained by composing the denotation of $\lambda x^{\mathsf{var}}.M$ with the innocent S-strategy cell_β. Let us write $[\![\cdots]\!]_{\mathrm{S}}$ for the resultant semantic map.

Proposition 8 (Soundness). *For any* $\mathsf{IA}_{\mathsf{cbv}}$*-term* $\Gamma \vdash M : \theta$, $[\![\Gamma \vdash M : \theta]\!]_{\mathrm{S}}$ *is an innocent S-strategy.*

Innocent S-strategies can be *decomposed* in a similar way to the innocent strategies of [6]. There is one important exception, though, which occurs when the second-move introduces a non-empty store (our rules of play imply that the move must be a question). Let α be the first variable from the non-empty store. In order to decompose the strategy, consider a P-view s in which α occurs in the second move q_α. It turns out that $s = qq_\alpha s_\alpha s'$, where (the store of) a move m^Σ from s contains α if, and only if, it is q_α or in s_α. In addition, no justification pointers connect s' to $q_\alpha s_\alpha$. This separation can be applied to decompose the view-function of an innocent S-strategy. The s_α parts, put together as a single S-strategy, can subsequently be dealt with in the style of factorization arguments, which remove α from moves at the cost of an additional var-component. Finally, to relate s_α's to the suitable s' one can use numerical codes for $q_\alpha s_\alpha$. These ideas lie at the heart of the following result. By a finitary innocent S-strategy we mean an innocent strategy whose view-function quotiented by name-variance is finite.

Proposition 9 (Finitary Definability and Universality).
– *Any finitary innocent S-strategy is* $\mathsf{IA}_{\mathsf{cbv}}$*-definable.*
– *Any recursively presentable innocent S-strategy is* $\mathsf{IA}_{\mathsf{cbv}}$*-definable.*

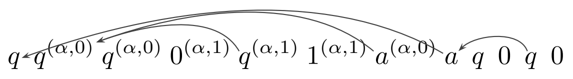
It is worth noting that the universality result for innocent S-strategies implies an analogous result for innocent strategies and PCF. Thanks to call-by-value, the result is actually sharper than the universality results of [1, 7], which had to be proved "up to observational equivalence". This was due to the fact that partial recursive functions could not always be represented in the canonical way (i.e. by terms for which the corresponding strategy contained plays of the form $q\,q\,n\,f(n)$). This is no longer the case under the call-by-value regime, where each partially recursive function f can be coded by a term whose denotation will be the strategy based on plays of the shape $n\,f(n)$.

With the soundness and definability results in place, we could now proceed in the familiar way to define a fully abstract model of $\mathsf{IA}_{\mathsf{cbv}}$ via the intrinsic quotient. However, this would be somewhat counterproductive. It turns out that RML is a conservative extension of $\mathsf{IA}_{\mathsf{cbv}}$ (Corollary 14), so the (simpler) fully abstract model of RML from [2], based on knowing strategies, is already fully abstract for $\mathsf{IA}_{\mathsf{cbv}}$. In fact, our model can be related to knowing strategies more precisely. Observe that by erasing storage annotations in an innocent S-strategy σ we obtain a knowing strategy, which we call $\mathsf{erase}(\sigma)$ (determinism follows from the fact that stores in O-moves are uniquely determined and from block-innocence). Let us write $[\![\cdots]\!]$ for the knowing strategy semantics (cast in [6]).

Lemma 10. *For any* $\mathsf{IA}_{\mathsf{cbv}}$*-term* $\Gamma \vdash M : \theta$, $[\![\Gamma \vdash M : \theta]\!] = \mathsf{erase}([\![\Gamma \vdash M : \theta]\!]_{\mathrm{S}})$.

This means the intrinsic quotient we would construct in the setting with stores can be represented more explicitly via the induced *complete* plays (without stores)[3]. Even though innocent S-strategies have not led us to a direct account of full abstraction for $\mathsf{IA}_{\mathsf{cbv}}$, we have obtained important insights into the structure of knowing strategies representing $\mathsf{IA}_{\mathsf{cbv}}$-terms: they are erasures of innocent S-strategies. Knowing strategies with this property will be referred to as ***block-innocent***. The knowledge that strategies determined by $\mathsf{IA}_{\mathsf{cbv}}$ are block-innocent will be crucial in establishing a series of results in the following sections.

Example 11. Let us revisit the two plays from the Introduction. The first one indeed comes from an innocent S-strategy (we reveal the stores below). For the second one to become innocent (in the setting with stores), a store with variable α, say, would need to be introduced in the second move. Then α must also occur in the seventh move by JUST-O, but it must not occur in the eighth move by JUST-P (the PA clause). Hence, it will not be present in the ninth move by JUST-O. Consequently, the last move (justified by the seventh move) is bound to break either PREV-PQ(a) (if it contains α) or JUST-P (if it does not).


$q\ q^{(\alpha,0)}\ q^{(\alpha,0)}\ 0^{(\alpha,1)}\ q^{(\alpha,1)}\ 1^{(\alpha,1)}\ a^{(\alpha,0)}\ a\ q\ 0\ q\ 0$

4 From omniscience to innocence

In Section 2 we introduced the three languages: PCF^+, $\mathsf{IA}_{\mathsf{cbv}}$ and RML. By the soundness and universality results of the previous section (as well as the soundness results from [6, 2]) the languages correspond respectively to innocent, block-innocent and knowing strategies. Let A be an arena. We write $\mathcal{I}_A$, $\mathcal{B}_A$ and $\mathcal{K}_A$ for the corresponding classes

[3] A play is complete if it does not contain unanswered questions. That such plays capture program equivalence in RML follows from the argument in [3], readily adaptable to RML. By Corollary 14, the same characterization will apply to $\mathsf{IA}_{\mathsf{cbv}}$.

of (store-free) strategies in A. Obviously, $\mathcal{I}_A \subseteq \mathcal{B}_A \subseteq \mathcal{K}_A$. Next we shall study type-theoretic conditions under which one kind of strategy collapses to another. Thanks to the universality results, this corresponds to the existence of an equivalent program in a weaker language.

Lemma 12. *Let* $A = [\![\theta_1, \cdots, \theta_n \vdash \theta \to \theta']\!]$*. Then* $\mathcal{B}_A \subsetneq \mathcal{K}_A$*.*

Proof. Observe that there exist moves q_0, a_0, q_1, a_1 such that $q_0 \vdash_A a_0 \vdash_A q_1 \vdash_A a_1$ and consider $\sigma = \{\epsilon, q_0, q_0a_0, q_0a_0q_1, q_0a_0q_1a_1\}$, i.e. σ has no response at $q_0a_0q_1a_1q_1$. Then $\sigma \in \mathcal{K}_A \setminus \mathcal{B}_A$. It is worth remarking that a strategy of the above kind denotes the RML-term $\vdash \mathsf{let}\, v = \mathsf{ref}\, \mathsf{in}\, \lambda x^{\mathsf{unit}}.\mathsf{if}\, !v\, \mathsf{then}\, \Omega\, \mathsf{else}\, v := !v + 1 : \mathsf{unit} \to \mathsf{unit}$. □

Lemma 12 confirms that, in general, block structure restricts expressivity. However, the next result shows this not to be the case for open terms of base type.

Lemma 13. *Let* $A = [\![\theta_1, \cdots, \theta_n \vdash \beta]\!]$*. Then* $\mathcal{B}_A = \mathcal{K}_A$*.*

Proof. Observe that any knowing strategy for A becomes block-innocent if in the second move P introduces a store with one variable that keeps track of the history of play (this is reminiscent of the factorization arguments in game semantics). The variable should be removed from the store by P only when he plays an answer to the initial question, in which case the play becomes complete and cannot be extended further. □

By universality, we can conclude that each RML-term of base type is equivalent to an $\mathsf{IA}_{\mathsf{cbv}}$-term. Since contexts used for testing equivalence are exactly of this kind, we obtain the following corollaries. The first one amounts to saying that RML is a conservative extension of $\mathsf{IA}_{\mathsf{cbv}}$. The second one states that block-structured contexts suffice to distinguish terms that might use scope extrusion.

Corollary 14. *For any* $\mathsf{IA}_{\mathsf{cbv}}$*-terms* $\Gamma \vdash M_1, M_2$ *and* RML*-terms* $\Gamma \vdash N_1, N_2$

- $\Gamma \vdash M_1 \cong_{\mathsf{RML}} M_2$ *if, and only if,* $\Gamma \vdash M_1 \cong_{\mathsf{IA}_{\mathsf{cbv}}} M_2$*.*
- $\Gamma \vdash N_1 \cong_{\mathsf{RML}} N_2$ *if, and only if,* $\Gamma \vdash N_1 \cong_{\mathsf{IA}_{\mathsf{cbv}}} N_2$*.*

Now we investigate the boundary between block structure and lack of state.

Lemma 15. *Let A be an arena such that each question enables an answer* [4]*. The following conditions are equivalent.*

1. $\mathcal{B}_A \subseteq \mathcal{I}_A$*.*
2. *No O-question is enabled by a P-question:* $m \vdash_A q_O$ *implies* $\lambda_A(m) = PA$*.*
3. *Store content of O-questions is trivial:* $sq_O^{\Sigma} \in SP_A$ *implies* $\mathsf{dom}\,(\Sigma) = \epsilon$*.*

We can now determine at which types block-innocence implies innocence.

Lemma 16. $[\![\theta_1, \cdots, \theta_n \vdash \theta]\!]$ *satisfies condition 2 of Lemma 15 iff* $\mathsf{ord}(\theta_i) \leq 1$ *(*$i = 1, \cdots, n$*) and* $\mathsf{ord}(\theta) \leq 2$*.*

Consequently, second-order $\mathsf{IA}_{\mathsf{cbv}}$-terms always have purely functional equivalents. Finally, we can pinpoint the types at which strategies are bound to be innocent: it suffices to combine the previous findings.

[4] All denotable arenas enjoy this property.

Lemma 17. *Let $A = [\![\theta_1, \cdots, \theta_n \vdash \theta]\!]$. Then $\mathcal{K}_A = \mathcal{I}_A$ iff* $\mathsf{ord}(\theta_i) \le 1$ *($i = 1, \cdots, n$) and* $\mathsf{ord}(\theta) = 0$.

In the next section we demonstrate that the gap in expressivity between $\mathcal{K}_A$ and $\mathcal{B}_A$ also bears practical consequences. The undecidable equivalence problem for second-order finitary RML becomes decidable in second-order finitary $\mathsf{IA}_{\mathsf{cbv}}$ (as well as at some third-order types).

5 Decidability of a finitary fragment of $\mathsf{IA}_{\mathsf{cbv}}$

To prove program equivalence decidable we restrict the base datatype of integers to the finite segment $\{0, \cdots, N\}$ ($N > 0$) and replace recursive definitions ($\mathsf{Y}(M)$) with looping ($\mathsf{while}\, M\, \mathsf{do}\, N$). Let us call the resultant language $\mathsf{IA}_{\circlearrowleft}$. Our decidability result will hold for a subset $\mathsf{IA}^{2+}_{\circlearrowleft}$ of $\mathsf{IA}_{\circlearrowleft}$, in which type order is restricted. $\mathsf{IA}^{2+}_{\circlearrowleft}$ will reside inside the third-order fragment of $\mathsf{IA}_{\circlearrowleft}$ and contain its second-order fragment. Note that the second-order fragment of similarly restricted RML is known be undecidable (even without loops) [10].

The decidability of program equivalence in $\mathsf{IA}^{2+}_{\circlearrowleft}$ will be shown by translating terms to regular languages representing the corresponding *knowing* strategies. We stress that we are *not* going to work with the induced S-plays. Nevertheless, the translation will crucially rely on insights gleaned from the semantics with explicit stores. More precisely, we will be interested in capturing the induced *complete* (store-free) plays. It is worth mentioning that, unlike in the (single-threaded) call-by-name setting, complete plays need not be maximal.

To represent plays as words, one needs to consider carefully how to represent pointers, should that be necessary. For example, this can be done by decorating moves with integers that encode the distance from the target in some way. Only pointers from questions require attention, since those from answers are uniquely reconstructible through the well-bracketing condition. Next we analyse two typing scenarios that look hopeless from the point of view of encoding pointers, since the distance from the pointer can grow arbitrarily. In the first case, thanks to block-innocence, we will be able to overcome the difficulties. The other case must remain a challenge for future work (or an undecidability result). On the basis of our discussion we shall subsequently introduce the type system of $\mathsf{IA}^{2+}_{\circlearrowleft}$.

Consider the arena $[\![\theta \vdash \theta_1 \to \ldots \to \theta_k \to \beta]\!]$. Due to the presence of the k arrows on the right-hand side we obtain chains of enablers $q_0 \vdash a_0 \vdash \cdots \vdash q_k \vdash a_k$, where q_0 is initial and each q_i ($i = 1, \cdots, k$) is initial in $[\![\theta_i]\!]$. We shall call the moves ***spinal***. Consider $[\![\vdash \lambda x^{\mathsf{unit}}.\lambda y^{\mathsf{unit}}.() : \mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}]\!]$ (i.e. $k = 2$), which contains plays of the form $q_0 a_0 (q_1 a_1)^j$ for any $j \ge 0$. Pointers are still uniquely determined in these plays, but everything changes once O plays q_2 next. Then the target might be any of the j occurrences of q_1. The strategy in question actually offers responses in all such cases, so it would seem that all of these plays need to be represented (thus necessitating the use of an infinite alphabet). Fortunately, thanks to block-innocence, we can restrict ourselves to the case $j = 1$ and make the problem disappear. To see why, observe that none of the moves q_i, a_i will ever carry a non-empty store in an S-play, by

Definition 4. Thus, because the strategy is block-innocent, its behaviour is already represented faithfully by the single play $q_0 a_0 q_1 a_1 q_2 a_2$. In fact, this is one of the cases when block-innocence implies innocence, but in general this will not be true for denotations of $\mathsf{IA}^{2+}_{\circlearrowleft}$-terms. Hence, we generalize the observation as follows. Since the move q_1 never carries a non-trivial store, it follows that no additional information about the strategy is hidden in plays containing two occurrences of q_1. This is because a block-innocent strategy has to behave uniformly after each q_1 and in general will depend only on what happened between q_0 and a_0, and not on what happened after a previous copy of q_1 was played (there can be no communication between the "threads" started with q_1 because q_1 cannot carry a non-trivial store). Now that it is known that O need only play one occurrence of q_1, we can apply a similar reasoning to q_2, and so on. This yields the following lemma. Note that, due to Visibility, insisting on the presence of a unique copy of $q_1, \cdots, q_k$ in a play amounts to asking that each q_i be preceded by a_{i-1}.

Lemma 18. *Call a play* **spinal** *if each spinal question q_i ($0 < i \le k$) occurring in it is the immediate successor of a_{i-1}. Let P^{sp}_A be the set of spinal plays of A. Let $\sigma, \tau : A$ be block-innocent strategies. Then $\sigma \cap P^{sp}_A = \tau \cap P^{sp}_A$ implies $\sigma = \tau$.*

Hence, for the purpose of checking program equivalence, it suffices to compare the induced sets of *spinal* complete plays.

Now that we have dealt with one challenge, let us introduce another one, which cannot be overcome so easily. Consider the arena $[\![(\theta_1 \to \theta_2 \to \theta_3) \to \theta_4 \vdash \theta]\!]$ and the enabling sequence $q_0 \vdash q_1 \vdash q_2 \vdash a_2 \vdash q_3$ it contains. Now consider the plays $q_0 q_1 (q_2 a_2)^j q_3$, where $j \ge 0$. Again, to represent the pointer from q_3 to one of the j occurrences of a_2, one would need an unbounded number of indices. This time it is not sufficient to restrict j to 1, because the behaviour need not be uniform after each q_2 (this is because in the setting with stores a non-empty store can be introduced as soon as in the second move q_1). To see that the concern is real, consider the term $f : (\mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit} \vdash \mathsf{new}\, x \,\mathsf{in}\, f(\lambda y^{\mathsf{unit}}.\cdots \lambda z^{\mathsf{unit}}.\cdots) : \mathsf{unit}$, where $(\cdots)$ contain some code inspecting and changing the value of x.

This leads us to introduce $\mathsf{IA}^{2+}_{\circlearrowleft}$ via a type system that will not generate the configuration just discussed. Another restriction is to omit third-order types in the context, as they lead beyond the realm of regular languages (cf. $f : ((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit} \vdash f(\lambda g^{\mathsf{unit}\to\mathsf{unit}}.g())$). Since var leads to identical problems as $\mathsf{unit} \to \mathsf{unit}$, we restrict its use accordingly.

Definition 19. *$\mathsf{IA}^{2+}_{\circlearrowleft}$ consists of $\mathsf{IA}_{\circlearrowleft}$-terms whose typing derivations rely solely on typing judgments of the shape $x_1 : ctype_1, \cdots, x_n : ctype_n \vdash M : ttype$, where $ctype$ and $ttype$ are defined by the grammar below.*

$$\begin{array}{llllllllll} ctype ::= & \beta & \mid & \mathsf{var} & \mid & \beta \to ctype & \mid & \mathsf{var} \to ctype & \mid & (\beta \to \beta) \to ctype \\ ttype ::= & \beta & \mid & \mathsf{var} & \mid & ctype \to ttype & & & & \end{array}$$

A lot of pointers from questions become uniquely determined in strategies representing $\mathsf{IA}^{2+}_{\circlearrowleft}$ terms, namely, all pointers from any O-questions and all pointers from P-questions to O-questions.

Lemma 20. *Let $A = [\![ctype_1, \cdots, ctype_n \vdash ttype]\!]$ and s_1, s_2 be spinal plays of A that are equal after all pointers from O-questions and all pointers from P-questions to O-questions have been erased. Then $s_1 = s_2$.*

Thus, the only pointers that need to be accounted for are those from P-questions to O-answers. Here is the simplest scenario illustrating that they can be ambiguous. Consider the terms

$$f : \mathsf{unit} \to \mathsf{unit} \to \mathsf{unit} \vdash \mathsf{let}\, g_1 = f()\,\mathsf{in}\,(\mathsf{let}\, g_2 = f()\,\mathsf{in}\, g_i()) : \mathsf{unit}$$

where $i = 1, 2$. They lead to the following plays, respectively for $i = 1$ and $i = 2$, which are equal up to pointers from P-questions to O-answers.

$$\begin{tikzcd}[column sep=small]
q_0 & q_1 \arrow[l, bend right] & a_1 \arrow[l, bend right] & q_1 \arrow[lll, bend right] & a_1 \arrow[l, bend right] & q_2 \arrow[lll, bend right] & \qquad & q_0 & q_1 \arrow[l, bend right] & a_1 \arrow[l, bend right] & q_1 \arrow[lll, bend right] & a_1 \arrow[l, bend right] & q_2 \arrow[l, bend right]
\end{tikzcd}$$

We are going to represent such pointers with numerical indices encoding the target of the pointer inside the current P-view. More precisely, let us enumerate (starting from 0) all question-enabling O-answers in the P-view. Then pointers from P-questions to O-answers can be encoded by decorating the P-question with the index of the O-answer. The plays above will be encoded as $q_0 q_1 a_1 q_1 a_1 q_2^0$ and $q_0 q_1 a_1 q_1 a_1 q_2^1$ respectively (other pointers are uniquely recoverable by Lemma 20 and will not be represented explicitly). So that we need not study the behaviour of the representation scheme for pointers under general composition (after which the indices might need to be recalculated), we restrict our translation to terms in a canonical shape, to be defined next. Any $\mathsf{IA}_{\circlearrowleft}$-term can be converted effectively to such a form and the conversion preserves denotation.

The canonical forms are defined by the following grammar. We use types as superscripts, whenever we want to highlight the type of an identifier (u, v, x, y, z range over identifier names). Note that the only identifiers in canonical form are those of base type, represented by x^β below.

$$\begin{aligned}
\mathbb{C} ::= {} & () \mid i \mid x^\beta \mid x^\beta \oplus y^\beta \mid \mathsf{if}\, x^\beta\, \mathsf{then}\, \mathbb{C}\, \mathsf{else}\, \mathbb{C} \mid x^{\mathsf{var}} := y^{\mathsf{int}} \mid !x^{\mathsf{var}} \mid \lambda x^\theta.\mathbb{C} \mid \\
& \mathsf{mkvar}(\lambda x^{\mathsf{unit}}.\mathbb{C}, \lambda y^{\mathsf{int}}.\mathbb{C}) \mid \mathsf{new}\, x^{\mathsf{var}}\, \mathsf{in}\, \mathbb{C} \mid \mathsf{while}\, \mathbb{C}\, \mathsf{do}\, \mathbb{C} \mid \mathsf{let}\, x^\beta = \mathbb{C}\, \mathsf{in}\, \mathbb{C} \mid \\
& \mathsf{let}\, x = z y^\beta\, \mathsf{in}\, \mathbb{C} \mid \mathsf{let}\, x = z\, \mathsf{mkvar}(\lambda u^{\mathsf{unit}}.\mathbb{C}, \lambda v^{\mathsf{int}}.\mathbb{C})\, \mathsf{in}\, \mathbb{C} \mid \mathsf{let}\, x = z(\lambda x^\theta.\mathbb{C})\, \mathsf{in}\, \mathbb{C}
\end{aligned}$$

Lemma 21. *Let $\Gamma \vdash M : \theta$ be an $\mathsf{IA}_{\circlearrowleft}$-term. There is an $\mathsf{IA}_{\circlearrowleft}$-term $\Gamma \vdash N : \theta$ in canonical form, effectively constructible from M, such that $[\![\Gamma \vdash M]\!] = [\![\Gamma \vdash N]\!]$.*

Proof. N can be obtained via a series of η-expansions, β-reductions and commuting conversions involving let and if. $\square$

A useful feature of the canonical form is that the problems with pointers can be related to the syntactic shape: they concern references to let-bound identifiers x^θ such that θ is *not* a base type (i.e. $\theta = \mathsf{var}$ or θ is a function type). The representation scheme for pointers corresponds then to enumerating such let bindings along branches of the syntactic tree of the canonical form (using 0 for topmost bindings). Below we state our representability theorem for $\mathsf{IA}_{\circlearrowleft}^{2+}$-terms. The definition of $\mathcal{A}_M$ is actually too generous, as we shall only need indices to decorate P-questions enabled by O-answers (in concrete examples the indices will be superscripts).

Proposition 22. *Suppose $\Gamma \vdash M : \theta$ is an $\mathsf{IA}^{2+}_{\circlearrowleft}$-term. Let $\mathcal{A}_M = M_A + (M_A \times \mathbb{N})$, where $A = [\![\Gamma \vdash \theta]\!]$. Let $\mathcal{C}_{\Gamma\vdash M}$ be the set of non-empty spinal complete plays from $[\![\Gamma \vdash M : \theta]\!]$. Then $\mathcal{C}_{\Gamma\vdash M}$ can be represented as a regular language over a* finite *subset of $\mathcal{A}_M$.*

Proof. For brevity, we shall write $\mathcal{C}_M$ instead of $\mathcal{C}_{\Gamma\vdash M}$ whenever it is clear what Γ should be. $\mathcal{C}_M$ can be decomposed as $\sum_{i\in I_A}(i\ \mathcal{C}^i_M)$. Obviously $\mathcal{C}_M$ is regular if, and only if, so is any of $\mathcal{C}^i_M$ ($i \in I_A$). Hence, it suffices to show that $\mathcal{C}^i_M$ is regular for any relevant i. The proof proceeds by induction on the structure of canonical forms. The most difficult cases are those involving let. Note that whenever a canonical form of an $\mathsf{IA}^{2+}_{\circlearrowleft}$-term is of the shape $\mathsf{let}\, x = z(\lambda x^{\theta}.\mathbb{C})\,\mathsf{in}\,\mathbb{C}$, z's type must be of the form $(\beta_1 \to \beta_2) \to (\theta_1 \to \theta_2)$ (and θ is a base type). We handle this case below. Consider the terms:

$$\Gamma, z : (\beta_1 \to \beta_2) \to (\theta_1 \to \theta_2), y : \beta_1 \vdash M : \beta_2,$$
$$\Gamma, z : (\beta_1 \to \beta_2) \to (\theta_1 \to \theta_2), x : \theta_1 \to \theta_2 \vdash N : \theta'.$$

Assuming that M and N satisfy the Proposition, we show that so does $N' \equiv \mathsf{let}\, x = z(\lambda y^{\beta_1}.M)\,\mathsf{in}\, N$. We shall refer to moves contributed by $x : \theta$ with m_x. If we want to range solely over O- or P-moves from the component, we use o_x and p_x respectively. Moreover, we use $m_{z,x}, o_{z,x}, p_{z,x}$ to refer to copies of m_x, o_x, p_x in the $z : \theta' \to \theta$ component. The most common operation performed using this notation will be the relabelling of m_x to $m_{z,x}$. If θ is a function type, then there is a unique P-question q_x enabled by the initial move $\star_x$. Whenever we have a separate substitution rule for q_x, the rule for m_x or p_x will not apply to q_x. In most cases we will want to substitute $q^0_{z,x}$ ($q_{z,x}$ decorated with index 0 represent a topmost binding) for q_x. In addition, $i + 1/i$ is used to increment all numerical indices by 1. Then we have

$$\mathcal{C}^{(i_\Gamma,\star_z)}_{N'} = q_z\,\mathcal{C}'\,\star_{z,x}\,\mathcal{C}^{(i_\Gamma,\star_z,\star_x)}_{N}[i+1/i,\ q^0_{z,x}\mathcal{C}'/q_x,\ p_{z,x}\mathcal{C}'/p_x,\ o_{z,x}/o_x]$$

where $\mathcal{C}' = (\sum_{i\in I_{[\![\beta_1]\!]}} i_z\,\mathcal{C}^{(i_\Gamma,\star_z,i_y)}_M[j_z/j])^*$ and j ranges over $I_{[\![\beta_2]\!]}$. □

Theorem 23. *Program equivalence of $\mathsf{IA}^{2+}_{\circlearrowleft}$-terms is decidable.*

We remark that adding dynamic memory allocation in the form of ref to $\mathsf{IA}^{2+}_{\circlearrowleft}$, or its second-order sublanguage, results in undecidability [10]. Hence, at second order, block structure is "strictly weaker" than scope extrusion.

6 Summary

In this paper we have introduced the notion of block-innocence that has been linked with call-by-value Idealized Algol in a sequence of results. Thanks to the faithfulness of block-innocence, we could investigate the interplay between type theory, functional computation and stateful computation with block structure and dynamic allocation respectively. We have also shown a new decidability result for a carefully designed fragment of $\mathsf{IA}_{\mathsf{cbv}}$. Its extension to product types poses no particular difficulty. In fact, it suffices to follow the way we have tackled the var type, which is itself a product type.

The result thus extends those from [5] and is a step forward towards a full classification of decidable fragments of $\mathsf{IA}_{\mathsf{cbv}}$: the language $\mathsf{IA}^{2+}_{\circlearrowleft}$ we considered features all second-order types and some third-order types, while finitary $\mathsf{IA}_{\mathsf{cbv}}$ is known to be undecidable at order 5 [9]. Interestingly, $\mathsf{IA}^{2+}_{\circlearrowleft}$ features restrictions that are compatible with the use of higher-order types in PASCAL [8], in which procedure parameters cannot be procedures with procedure parameters. An interesting topic for future work is a category-theoretic characterization of block-innocence.

References

1. S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163:409–470, 2000.
2. S. Abramsky and G. McCusker. Call-by-value games. In *Proceedings of CSL*, volume 1414 of *Lecture Notes in Computer Science*, pages 1–17. Springer-Verlag, 1997.
3. S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In P. W. O'Hearn and R. D. Tennent, editors, *Algol-like languages*, pages 297–329. Birkhaüser, 1997.
4. M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2002.
5. D. R. Ghica. Regular-language semantics for a call-by-value programming language. In *Proceedings of MFPS*, volume 45 of *Electronic Notes in Computer Science*. Elsevier, 2001.
6. K. Honda and N. Yoshida. Game-theoretic analysis of call-by-value computation. *Theoretical Computer Science*, 221(1–2):393–456, 1999.
7. J. M. E. Hyland and C.-H. L. Ong. On Full Abstraction for PCF: I. Models, observables and the full abstraction problem, II. Dialogue games and innocent strategies, III. A fully abstract and universal game model. *Information and Computation*, 163(2):285–408, 2000.
8. J. C. Mitchell. *Concepts in programming languages*. Cambridge University Press, 2002.
9. A. S. Murawski. About the undecidability of program equivalence in finitary languages with state. *ACM Transactions on Computational Logic*, 6(4):701–726, 2005.
10. A. S. Murawski. Functions with local state: regularity and undecidability. *Theoretical Computer Science*, 338(1/3):315–349, 2005.
11. F. Oles. Type algebras, functor categories and block structure. In M. Nivat and J. C. Reynolds, editors, *Algebraic Methods in Semantics*, pages 543–573. Cambridge University Press, 1985.
12. C.-H. L. Ong. Observational equivalence of 3rd-order Idealized Algol is decidable. In *Proceedings of IEEE Symposium on Logic in Computer Science*, pages 245–256. Computer Society Press, 2002.
13. A. M. Pitts and I. Stark. On the observable properties of higher order functions that dynamically create local names, or: What's new? In *Proc. 18th Int. Symp. on Math. Foundations of Computer Science*, pages 122–141. Springer-Verlag, 1993. LNCS Vol. 711.
14. G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.
15. J. Power. Semantics for local computational effects. *Electr. Notes Theor. Comput. Sci.*, 158:355–371, 2006.
16. J. C. Reynolds. The essence of Algol. In J. W. de Bakker and J.C. van Vliet, editors, *Algorithmic Languages*, pages 345–372. North Holland, 1981.
17. I. D. B. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge Computing Laboratory, 1995. Technical Report No. 363.
18. N. Tzevelekos. Full abstraction for nominal general references. *Logical Methods in Computer Science*, 5(3), 2009.