Observational equivalence of 3rd-order Idealized Algol is decidable

C.-H. L. Ong[*]
Oxford University Computing Laboratory, UK

Abstract

We prove that observational equivalence of 3rd-order finitary Idealized Algol (IA) is decidable using Game Semantics. By modelling state explicitly in our games, we show that the denotation of a term M of this fragment of IA (built up from finite base types) is a *compactly innocent* strategy-with-state i.e. the strategy is generated by a *finite* view function f_M. Given any such f_M, we construct a real-time deterministic pushdown automata (DPDA) that recognizes the complete plays of the knowing-strategy denotation of M. Since such plays characterize observational equivalence, and there is an algorithm for deciding whether any two DPDAs recognize the same language, we obtain a procedure for deciding observational equivalence of 3rd-order finitary IA. This algorithmic representation of program meanings, which is *compositional*, provides a foundation for model-checking a wide range of behavioural properties of IA and other cognate programming languages. Another result concerns 2nd-order IA with recursion: we show that observational equivalence for this fragment is undecidable.

1 Introduction

Game Semantics has emerged as a powerful paradigm for giving semantics to a variety of programming languages and logical systems. In the game reading, types (specifications) are interpreted as games, and programs (implementations) as strategies on games. Game Semantics captures the quantitative and algorithmic aspects of a computation typical of Operational Semantics, while admitting *compositional* methods in the style of Denotational Semantics: it provides a very *concrete* way of building fully abstract models. This paper concerns a recent development of Game Semantics in a new, *algorithmic* direction, with a view to applications in computer-assisted verification and program analysis. The important first steps have been taken by Ghica and McCusker [5]; they show that the 2nd-order finitary (i.e. recursion-free) fragment of the fully abstract game semantics of Reynolds' language, Idealized Algol (IA), can be represented in a remarkably simple form by regular expressions. This yields a procedure for deciding observational equivalence for this fragment.

[*]OUCL, Wolfson Building, Parks Road, Oxford OX1 3QD, United Kingdom. www.comlab.ox.ac.uk/oucl/work/luke.ong/

The promise of this approach is to transfer the methods of Model Checking based on automata-theoretic representations, which has been so successful in the analysis of hardware designs, and communications and security protocols, to the much more *structured* setting of programming languages, in which data types and control flow are important.

Overview. A notable feature of Abramsky and McCusker's fully abstract knowing (or history-sensitive) strategy game semantics [2] for IA is the implicit nature of its model of state. Knowing strategies that denote finitary IA terms are in general infinite sets of *justified* sequences (i.e. sequences of moves, each of which is equipped with a pointer to an earlier move) of a certain kind. For games of up to 2nd order, such pointers can safely be ignored (because they are uniquely reconstructible), so that justified sequences may simply be represented as words over an alphabet of moves. Thus Ghica and McCusker have shown that knowing strategies that denote 2nd-order finitary IA can be represented, compositionally, as regular expressions. Can the approach be extended to higher orders? We show that using a scheme based on *view offsets*, justified sequences can be encoded as words over an appropriate alphabet. We then show that, in general, knowing strategies that denote the 3rd-order fragment are *not* regular (Example 5.1).

What automata-theoretic formalism then is needed to characterize (the knowing-strategy semantics of) 3rd-order finitary IA? Our approach to the question is via a more intensional version of game semantics which models state explicitly: we attach a *state* (i.e. a finite function from *locations* to data values) to each move of the play. Intuitively the locations that are defined at a move correspond to the assignable variables that have been created thus far in the computation. Because these variables are *local* (to the block in which they are allocated), their contents can only be changed by the assignment commands that are *in scope* i.e. within the same block in the program being modelled (P's perspective), and not directly by any assignment command of the program context (O's perspective). This translates to a basic principle governing state: Only P can change state, either by updating the contents at locations introduced by earlier P-moves which are currently *P-visible* (i.e. appearing in the *P-view* of the history of play), or by introducing new locations. The key point is that, for games of up to 3rd-order, we may take a completely *innocent* view of states by considering as relevant *only* those locations that have been introduced by earlier P-moves which are currently P-visible. This has the crucial effect of restraining the growth of states as the play unfolds.

We show that $[\![\Gamma \vdash M : A]\!]$, the strategy-with-state that denotes a 3rd-order IA term-in-context, is *compactly innocent* in the sense (of [7]) that the strategy is generated by a *finite* view function f_M. Moreover $[\![\Gamma \vdash M : A]\!]$ is closely related to the knowing-strategy denotation, written $[\![\Gamma \vdash M : A]\!]^\kappa$: we have (Theorem 4.3(i))

$$\textbf{\textit{erase}}\,[\![\Gamma \vdash M : A]\!] \;=\; [\![\Gamma \vdash M : A]\!]^\kappa$$

i.e. the knowing-strategy denotation may be recovered by erasing states from (each play of) the innocent strategy-with-state denotation. Given any such finite view function f_M, we then show how to construct an associated real-time (i.e. no ϵ-transition) deterministic pushdown automaton (DPDA) P_{f_M}. The control states of P_{f_M} are the P-views that define f_M; its input alphabet is the set of moves (without state) of the arena, augmented by certain symbols for the purpose of encoding pointers. Writing $\mathcal{L}(\mathsf{P}_{f_M})$ for the language recognized by P_{f_M}, we then prove a major technical result (Theorem 5.2) of the paper:

$$\mathcal{L}(\mathsf{P}_{f_M}) \;=\; \textbf{\textit{cplays}}\,[\![\Gamma \vdash M : A]\!]^\kappa$$

In other words, the set of *complete plays* (i.e. plays in which every question is answered) of the knowing-strategy semantics of 3rd-order finitary IA is *deterministic context-free*. This answers the question we posed earlier in the Overview.

Further since compactly innocent strategies-with-state are *effectively* compositional (Theorem 3.6), the innocent-strategy denotation $[\![\Gamma \vdash M : A]\!]$, regarded as a map $M \mapsto f_M$, is effectively computable, and hence, so is the deterministic context-free representation $M \mapsto \mathsf{P}_{f_M}$. Finally, thanks to a characterization of observational equivalence (written $\approx$) in terms of complete plays in [2] (i.e. $M \approx N$ iff $\textbf{\textit{cplays}}\,[\![\Gamma \vdash M : A]\!]^\kappa = \textbf{\textit{cplays}}\,[\![\Gamma \vdash N : A]\!]^\kappa$), we obtain a pleasing application of the DPDA representation:

Theorem 1.1. *For any 3rd-order finitary IA terms-in-contexts $\Gamma \vdash M : A$ and $\Gamma \vdash N : A$ that are constructed from finite base types, we have*

$$M \approx N \iff \mathcal{L}(\mathsf{P}_{f_M}) = \mathcal{L}(\mathsf{P}_{f_N}).$$

Further, because there is an effective procedure for deciding whether any two DPDAs recognize the same language, $M \approx N$ is decidable.

A desirable consequence of the above characterization is that whenever $M \not\approx N$, any complete play that witnesses the inequality of $\mathcal{L}(\mathsf{P}_{f_M})$ and $\mathcal{L}(\mathsf{P}_{f_N})$ can be used to construct a separating context for M and N.

Another result of the paper concerns 2nd-order IA with recursion (in which fixpoints of functions of up to second order are computable). We show that observational equivalence for this fragment is undecidable (Theorem 5.5).

Related work. Perhaps the most important result that we should mention here is Loader's undecidability theorem in [9]. He proves that observational equivalence of 4th-order finitary PCF is *undecidable*. Observational equivalence of (active) IA does not extend that of PCF conservatively; indeed it is not known whether observational equivalence of 4th or higher-order IA is decidable. On the (Algorithmic) Game Semantics front, Ghica has extended his earlier work with McCusker to a call-by-value language with arrays [4], and to model checking Hoare-style program correctness assertions [3]. In a foundational direction, Abramsky [1] has extended the regular language representation to games in whose plays there is at most one pending occurrence of a question at any time, so that each move has a unique justifier (and so pointers may be ignored).

2 Idealized Algol

Reynolds' *Idealized Algol* (IA for short) [16] is a compact language that elegantly combines state-based procedural and higher-order functional programming. The basis of IA is a simply-typed call-by-name λ-calculus in which the standard constructs of imperative programming are definable. For simplicity, we shall consider a version of IA generated from a single basic data set of natural numbers. The *base types* of IA (ranged over by β) are exp, loc and com, which are, respectively, expressions that have natural-number values, *locations* (or *assignable program variables*) at which natural numbers can be stored, and commands. The *types* of IA (ranged over by A, B, C, etc.) are then defined by the grammar: $A \;:=\; \beta \mid A \Rightarrow A$. The ***order*** of a type A, written *order*(A), is defined as: *order*$(\beta) = 0$, and *order*$(A \Rightarrow B) = \max(\textit{order}(A) + 1, \textit{order}(B))$.

Term candidates of IA (ranged over by M, N, P, etc.) are defined as follows:

$$\begin{array}{rcl} M & ::= & x \mid M M \mid \boldsymbol{\lambda} x : A.M \mid \Omega \mid n \mid \mathbf{Y}(M) \\ & \mid & \mathtt{succ} \mid \mathtt{pred} \mid \mathtt{ifzero}\, M \,\mathtt{then}\, M \,\mathtt{else}\, M \\ & \mid & X \mid M := M \mid \mathtt{!}M \mid \mathtt{mkvar}\, M\, M \\ & \mid & \mathtt{skip} \mid M \,\mathtt{;}\, M \mid \mathtt{new}\, x := n \,\mathtt{in}\, M \end{array}$$

where x ranges over a countable set of variables, X over a countable set of locations, and n over natural numbers. The first two clauses define the standard PCF constructs. We use infix := for variable assignment, prefix **!** for dereferencing, infix **;** for command sequencing, and write `skip` for the null command. *Block-allocated local variables* are introduced by the construct $\mathtt{new}\, x := n \,\mathtt{in}\, M$ (the local variable x is initialized to n as control enters the block). The *bad-variable constructor* `mkvar` takes a "read method" $M : \mathsf{exp}$ and a "write method" $N : \mathsf{exp} \Rightarrow \mathsf{com}$ and creates an object $\mathtt{mkvar}\, M\, N$ of type loc. The constructor is not part of Reynolds' original definition [16], but was introduced in [2] in order to prove a definability result, on which the full abstraction construction depends. Another departure from Reynolds' definition is that we allow *active* expressions i.e. expressions with side effects such as $f : \mathsf{com} \Rightarrow \mathsf{exp} \vdash \mathtt{new}\, z := 0 \,\mathtt{in}\, (f(z := \mathbf{!}z + 1) + \mathbf{!}z) : \mathsf{exp}$. Using the fixed-point operator, IA can express iterations such as while-loops, and recursively defined procedures.

Values of IA, ranged over by V, V', etc., are values of the base types (i.e. numerals, `skip` and locations) and λ-abstractions. The operational semantics of IA is given in terms of an evaluation relation of the form $M, S \Downarrow V, S'$ where M and V range over closed terms and values respectively, and S and S' range over ***states***, which are finite functions from locations to natural numbers. The evaluation relation $\Downarrow$ is defined by induction over a set of rules, a selection of which is presented in Table 1.

Let M and N be terms such that for some Γ and A, $\Gamma \vdash M : A$ and $\Gamma \vdash N : A$ are provable (the typing rules are standard, see e.g. [13] for a definition). We define $M \lessapprox N$ as: for any context $C[\text{-}]$ such that $C[M]$ and $C[N]$ are *programs* (i.e. closed terms of type com), for any states S, S',

$$C[M], S \Downarrow \mathtt{skip}, S' \quad \Longrightarrow \quad C[N], S \Downarrow \mathtt{skip}, S'.$$

Note that quantification over *all* program contexts $C[\text{-}]$ ensures that all potential side effects of M and N are taken fully into account. We call $\lessapprox$ ***observational preorder***, and define $M \approx N$ (read "M and N are ***observationally equivalent***") to mean $M \lessapprox N$ and $N \lessapprox M$.

The theory of observational equivalence is rich. We consider some examples.

Example 2.1. (i) "Snap back": This illustrates the consequences for observational equivalence of the inability of IA to "snap back" the state to some previous point in the thread of computation. For $p : \mathsf{com} \Rightarrow \mathsf{com}$, we have $p(\mathbf{\Omega}) \approx$

$$\mathtt{new}\, x := 0 \,\mathtt{in}\, p(x := 1) \,\mathbf{;}\, \mathtt{if}\, \mathbf{!}x = 1 \,\mathtt{then}\, \mathbf{\Omega} \,\mathtt{else}\, \mathtt{skip}$$

(ii) Parametricity: For $p : \mathsf{com} \Rightarrow \mathsf{exp}_2 \Rightarrow \mathsf{com}$ (where exp_2 is the type of binary-valued expressions), we have

$$\begin{array}{rl} & \mathtt{new}\, x := 1 \,\mathtt{in}\, p\, (x := -\mathbf{!}x)\, (\mathbf{!}x > 0) \\ \approx & \mathtt{new}\, y := \mathtt{tt} \,\mathtt{in}\, p\, (y := \mathtt{not}\, y)\, (\mathbf{!}y) \end{array}$$

Remark 2.2. In [11] McCusker has adapted the knowing-strategy semantics of IA to give a characterization of IA without the bad-variable constructor `mkvar`. Using the model he shows that adding `mkvar` is conservative for observational equivalence but not for observational preorder.

3 Game semantics

We shall consider *arenas* as introduced in [7, 10], and use square and round parentheses in bold type as meta-variables for moves as follows:

O-question	P-answer	P-question	O-answer
[	**]**	**(**	**)**

We assume the reader is familiar with the following standard notions of Game Semantics: justified sequences, P-view $\ulcorner s \urcorner$ and O-view $\llcorner s \lrcorner$ of a justified sequence s, local and interaction sequences (for composing strategies), and the conditions of Visibility and Well-Bracketing. Definitions for these terms can be obtained from [7] or [10].

State Change Conditions. We shall consider justified sequences of *moves-with-state*, and introduce new conditions of *State Change*. First some notations. A ***state*** (ranged over by S, S_i, T, etc.) is a finite function from *locations* (ranged over by X, X', X_i, Y, etc.) to natural numbers. A ***move-with-state*** of an arena A is a pair, written m^S (or simply m if S is understood), where m is a move of A and S is a state; we refer to a location in $\mathrm{dom}(S)$ as a location ***defined*** at m. Given states S_0 and S_1, we define the state $S_0[S_1]$ (read "S_0 *updated by* S_1") as: for each $X \in \mathrm{dom}(S_0[S_1]) = \mathrm{dom}(S_0)$,

$$S_0[S_1](X) \;=\; \begin{cases} S_1(X) & \text{if } X \in \mathrm{dom}(S_1) \\ S_0(X) & \text{otherwise} \end{cases}$$

Given an arena, we consider justified sequences of moves-with-state that satisfy Visibility, Well-Bracketing, and the following:

(P) The locations defined at a P-move m^S are the locations defined at the preceding O-move $m_1^{S_1}$, and possibly some *fresh*[1] locations which are said to be ***introduced*** by m^S. I.e. $\mathrm{dom}(S_1) \subseteq \mathrm{dom}(S)$.

(O) The opening move has a null state. Let m^S be an O-move which is explicitly justified by $m_0^{S_0}$, and let $m_1^{S_1}$ be the move immediately preceding m. Then $S = S_0[S_1]$.

We state a straightforward consequence of (P) and (O):

Lemma 3.1. *Every location that is defined at a move (in a play) is introduced by some P-move that appears in the P-view at that point.* $\square$

[1] I.e. none in $\mathrm{dom}(S) \setminus \mathrm{dom}(S_1)$ appears earlier in the play.

$$\frac{M,S \Downarrow X,S'}{\mathbf{!}M,S \Downarrow n,S'}\ X \in \mathrm{dom}(S') \wedge S'(X)=n \qquad \frac{M,S \Downarrow \mathtt{skip},S' \quad N,S' \Downarrow V,S''}{M\,\mathbf{;}\,N,S \Downarrow V,S''}\ V=n \text{ or } \mathtt{skip}$$

$$\frac{M,S \Downarrow \mathtt{mkvar}\,P\,Q,S' \quad N,S' \Downarrow n,S'' \quad Qn,S'' \Downarrow \mathtt{skip},S'''}{M:=N,S \Downarrow \mathtt{skip},S'''} \qquad \frac{M,S \Downarrow X,S' \quad N,S' \Downarrow n,S''}{M:=N,S \Downarrow \mathtt{skip},S''[X\mapsto n]}$$

$$\frac{M[X/x],S[X\mapsto n] \Downarrow V,S'[X\mapsto m]}{\mathtt{new}\,x:=n\,\mathtt{in}\,M,S \Downarrow V,S'}\ X \notin \mathrm{dom}(S)\cup\mathrm{dom}(S'),\ V=n \text{ or } \mathtt{skip}.$$

Table 1. Selected rules defining the evaluation relation $\Downarrow$.

We need to check that the State Change Conditions correctly model the scoping of block-allocated local variables. Take a location X introduced by some P-move m in a play $s\,m\,u$. We define the ***thread*** of X to be the subsequence of the play $s\,m\,u$ consisting of moves at which X is defined (i.e. when control is in the scope of X). Specifically we should verify that condition (O) has the effect that whenever the thread of X is re-entered by O, the contents at X are set to the value *last held* in the thread. This is indeed the case, as the next lemma shows.

Lemma 3.2. *We assume an arena of order at most three. Let $s\,m_0\,u\,m_1\,m$ be a legal position in which the O-move m is explicitly justified by m_0. Suppose the location X is defined at m_0 (so by condition (O), X is also defined at m). Then either X is defined at m_1 or X is not defined at any move in the intervening segment u.*

Remark 3.1. (i) The Lemma fails for 4th-order games. Consider the following legal position

$$\begin{array}{ccccccccccc} [_0 & (_1 & [_2 & (_3 & [_2' & (_3' & [_4 &]_4 &)_3' &]_2' &)_3 \\ & & & X & & X' & X & X & X' & X' & X \end{array} \qquad (1)$$

In play (1), the numeric subscript gives the (type-theoretic) order of the move, and the O-question $[_4$ is explicitly justified by $(_3$; note that the targets of the rest of the pointers are completely determined by Well-Bracketing. Suppose $(_3$ introduces the location X, and $(_3'$ introduces the location X'. Assume that $(_1$ has a null state. Then by condition (O), X is defined at $[_4$, and so, by condition (P), also at $]_4$. But X', not X, is defined at $)_3'$ and $]_2'$. Now X is defined at $m_1 =)_3$; it is not defined at $]_2'$ but defined at some move in the segment $u = [_2' \cdots)_3'$, which contradicts the Lemma. The point is that, as the move $)_3$ re-enters the thread of X, condition (O) says that X should be set to the value that was held at X at move $(_3$, failing to take into account possible updates to X at $]_4$.

(ii) The State Change conditions define an *innocent* (i.e. view-dependent) notion of state, in the sense that at any point in a play, *only* the states of those moves that appear in the *P-view* (of the history) is relevant (see Lemma 3.1). By using a "history-sensitive" notion of state, it is possible to avoid the problem identified in (i). Unfortunately the price one would then be forced to pay is that the states that must be carried along in an interaction quickly accumulate and become large. As a consequence, strategies denoting recursion-free IA terms fail to be compactly innocent.

We define a ***legal position*** (or ***play***) of an arena to be a justified sequence of moves-with-state satisfying Visibility, Well-Bracketing, and State Change. A play is said to be ***complete*** just in case every question in it is answered. A ***strategy-with-state*** σ of an arena A is a set of prefix-closed legal positions of moves-with-state of A satisfying: (i) If even-length $s \in \sigma$ and sm^S is a legal position then $sm^S \in \sigma$; and (ii) *Determinacy*: For any odd-length $s \in \sigma$ if $s\,m_1^{S_1}$ and $s\,m_2^{S_2}$ are both in σ then $m_1 = m_2$ and $S_1 = S_2$. In the sequel, we shall often consider *stateless* strategies-with-state (i.e. the state of each move in every legal position is null) which coincide with what are called *strategies* in [7] or ***knowing strategies*** in [2]. For any strategy-with-state σ, we write ***cplays*** σ for the set of complete plays in σ.

For any justified (or interaction) sequence u of moves-with-state, we define ***erase*** u to be the sequence that is obtained from u by erasing the state from each move. For any strategy-with-state σ, we define ***erase*** $\sigma = \{$ ***erase*** $s : s \in \sigma \}$. It is easy to verify that for any $s, s' \in \sigma$, if ***erase*** s = ***erase*** s' then $s = s'$, and ***erase*** σ is a knowing strategy.

Composition of strategies-with-state. Suppose σ and τ are strategies-with-state of arenas $A \Rightarrow B$ and $B \Rightarrow C$ respectively, and suppose both arenas are of order at most 3. Their composite $\sigma\,\mathbf{;}\,\tau$, which will be shown to be a strategy-with-state of $A \Rightarrow C$, is defined in the style of "parallel composition with hiding in CSP", as is standard in Game Semantics, as far as the underlying moves (with pointers) are concerned. Roughly speaking, the states of the composite are obtained by disjoint union of the states of the component strategies.

Let S and S^- be states. We define a new state $S^- \lhd S$ as follows: for any $X \in \mathrm{dom}(S^- \triangleleft S) = \mathrm{dom}(S^-) \cup \mathrm{dom}(S)$

$$(S^- \triangleleft S)(X) \ = \ \begin{cases} S(X) & \text{if } X \in \mathrm{dom}(S) \\ S^-(X) & \text{otherwise} \end{cases}$$

Let s and s' be justified sequences of an arena such that ***erase*** $s =$ ***erase*** s'. We define a relation $s \lessdot s'$ (read "s' is a ***state extension*** of s") by recursion as follows: we have $\epsilon \lessdot \epsilon$; and $s\, m^S \lessdot s'\, m^{S'}$ holds provided (i) $s \lessdot s'$, (ii) $S \subseteq S'$, and (iii) if m is a P-move then $S' = S^- \triangleleft S$ where S^- is the state of the last move of s'. (In the following we shall assume that $s \lessdot t$ has the force that ***erase*** $s =$ ***erase*** t.)

Take σ and τ as before. We define $\mathbf{ISeq}(\sigma, \tau)$ to be the set of local sequences u of moves-with-state of (A, B, C) satisfying conditions (I1), (I2) and (I3) as follows:

(I1) There is some $t \in \tau$ such that $t \lessdot u \upharpoonright (B, C)$.

(I2) For each occurrence of an initial B-move b in u, there exists some $s \in \sigma$ such that $s \lessdot u \upharpoonright (A, B, b)$.

(I3) Suppose $m_1^{S_1}$ and $m_2^{S_2}$ occur consecutively in u.

- (a) Suppose m_1 is a P-move in $A \Rightarrow C$. Let $m_0^{S_0}$ be the move that explicitly justifies m_2 in u. Then $S_2 = S_0[S_1]$.
- (b) If m_1 is an O-move in *component*[2] X, then $\mathrm{dom}(S_1) \subseteq \mathrm{dom}(S_2)$, and $S_2 \upharpoonright \overline{\mathcal{L}_X} = S_1 \upharpoonright \overline{\mathcal{L}_X}$, where $\overline{(\text{-})}$ means set-complementation.

We assume that $\mathcal{L}_{(A,B,b)}$ – the set of locations introduced by P-moves of the component (A, B, b), for each b, and $\mathcal{L}_{(B,C)}$ (which is similarly defined), are pairwise disjoint. We define $u \upharpoonright (B, C)$ to be the justified sequence of moves-with-state which is obtained from u by first taking the subsequence consisting of only moves-with-state from $B \Rightarrow C$ and then replacing the state S of each move by $S \upharpoonright \mathcal{L}_{(B,C)}$; similarly for $u \upharpoonright (A, B, b)$. For any $u \in \mathbf{ISeq}(\sigma, \tau)$ we define $u \upharpoonright (A, C)$ to be the justified sequence of moves-with-state of $A \Rightarrow C$ that is obtained from u by deleting all B-moves. We can now define

$$\sigma \mathbin{\mathbf{;}} \tau \ = \ \{\, u \upharpoonright (A, C) : u \in \mathbf{ISeq}(\sigma, \tau) \,\}.$$

An important point to note is that conditions (I1), (I2) and (I3) above ensure that the states of an interaction sequence from $\mathbf{ISeq}(\sigma, \tau)$ are completely determined by the projected components, namely, $u \upharpoonright (B, C)$ and $u \upharpoonright (A, B, b)$, which are required to be state extensions of legal positions in τ and σ respectively.

[2]The (B, C)-*component* of $u \in \mathbf{ISeq}(\sigma, \tau)$ is the subsequence of u consisting of moves from $B \Rightarrow C$; and for every occurrence b of an initial B-move in u, the (A, B, b)-*component* is the subsequence of u consisting of moves from $A \Rightarrow B$ that are hereditarily justified by b.

Lemma 3.3. *For any $u \in \mathbf{ISeq}(\sigma, \tau)$, there is a unique $t \in \tau$ such that $t \lessdot u \upharpoonright (B, C)$; similarly for each b, there is a unique $s \in \sigma$ such that $s \lessdot u \upharpoonright (A, B, b)$.*

Suppose the last move of $u \in \mathbf{ISeq}(\sigma, \tau)$ is an O-move in component (B, C); thanks to Lemma 3.3, we can define $\ulcorner u \upharpoonright (B, C) \urcorner$ to be $\ulcorner t \urcorner$ for the unique $t \in \tau$ such that $t \lessdot u \upharpoonright (B, C)$; similarly for $\ulcorner u \upharpoonright (A, B, b) \urcorner$.

We can now prove that composition is well-defined, and preserved by state-erasure.

Theorem 3.4. *For any strategies-with-state $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$ of order at most 3, we have*

(i) $\sigma \mathbin{\mathbf{;}} \tau$ is a strategy-with-state over $A \Rightarrow C$.

*(ii) Further **erase** $(\sigma \mathbin{\mathbf{;}} \tau) =$ **erase** σ **;** **erase** τ.*

Suitably quotiented, strategies-with-state do form a category, but we do not pursue that here, because we shall use such strategies (and, later, the innocent such), *not* to build semantics, but as programmable *representations* of one.

Innocent strategies-with-state. The *view* of a legal position of moves-with-state is defined in exactly the same way as the standard, stateless, case. However note that the view of a legal position is not in general a legal position, because condition (O) of State Change may fail. The notion of *innocence* in the sense of [7] extends to strategies-with-state straightforwardly. We say that a strategy-with-state σ is ***innocent*** if whenever the even-length $sa^S b^T \in \sigma$, if $ta^S \in \sigma$ and $\ulcorner ta^S \urcorner = \ulcorner sa^S \urcorner$ then $ta^S b^T \in \sigma$. As in the stateless case, innocent strategies are completely determined by ***view functions***, which are maps from odd-length P-views p to *justified* P-moves (i.e. a P-move together with a pointer into p). We say that σ is ***generated*** by a view function[3] f, written $\sigma =$ ***strat***(f), just in case for any odd-length $sa^S \in \sigma$, we have $sa^S b^T \in \sigma$ if and only if $\ulcorner sa^S \urcorner \in \mathrm{dom}(f)$ and $f(\ulcorner sa^S \urcorner) = b^T$. (Note that the definition does not require every P-view in the domain of f to be the P-view of some legal position in σ.)

Remark 3.2. Fix a view function f. Suppose $f(p) = a^S$, and X is a fresh location introduced by the P-move a^S (we assume that X is not defined at any move in p). Take $u \in$ ***strat***(f) and suppose $t_1\, a^S \leqslant t_2$ and $t_2\, a^S \leqslant u$ such that $\ulcorner t_1 \urcorner = \ulcorner t_2 \urcorner = p$. By the freshness assumption in condition (P), the location X introduced by the second occurrence of a (i.e. after t_2) should be a fresh copy, distinct from the X introduced by the first occurrence of a. For innocent strategies-with-state over arenas of up to order 3, the insistence on freshness may safely be relaxed. Specifically:

[3]We require view functions f to satisfy *location freshness*: Suppose $f(p_1) = m_1^{S_1}$ and $f(p_2) = m_2^{S_2}$. Set N_i to be the set of locations in $\mathrm{dom}(S_i)$ that are introduced at m_i, and are not defined at any move in p_i; if $N_1 \cap N_2 \neq \varnothing$ then $p_1 = p_2$.

(1) No two threads of location X can ever *overlap in time*.

(2) When a previous thread is re-entered, the contents of the location is set to the value last held in that thread.

For (2) to hold, it is enough to modify the definition of $S_0[S_1]$ in condition (O) slightly: Suppose m^S is an O-move which is explicitly justified by $m_0^{S_0}$, and suppose the move preceding m^S is $m_1^{S_1}$ which is explicitly justified by $m_{10}^{S_{10}}$, we define $\mathrm{dom}(S_0[S_1]) = \mathrm{dom}(S_0)$ and for $X \in \mathrm{dom}(S_0[S_1])$

$$S_0[S_1](X) = \begin{cases} S_1(X) & \text{if } X \in \mathrm{dom}(S_1) \cap \mathrm{dom}(S_{10}) \\ S_0(X) & \text{otherwise} \end{cases}$$

Theorem 3.5. *Suppose $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$ are innocent strategies-with-state of order at most 3, then the composite $\sigma\,;\tau$ is an innocent strategy-with-state.*

For the proof, we use essentially the same argument as the compositionality proof in [7]. The key idea therein is what McCusker calls the *core* of an interaction sequence.

Example 3.3. We consider some examples of innocent strategies-with-state. (i) The usual innocent strategies (without state) are of course innocent strategies-with-states. Thus the canonical maps such as identities and projections are examples. If $\sigma : C \Rightarrow A$ and $\tau : C \Rightarrow B$ are innocent strategies-with-state, so is the pairing $\langle \sigma, \tau \rangle : C \Rightarrow (A \times B)$. Note also that modulo renaming of moves, innocent strategies-with-state of $C \Rightarrow (A \Rightarrow B)$ are the same as those of $C \times A \Rightarrow B$. (ii) The strategy generated by the good-variable view function $\mathsf{loc}_{X,i}$ (see definition (3)) is an important example.

The view function f can be presented as the least prefix-closed set T_f of P-views such that *even-length* $p\,m^S \in T_f$ if $f(p) = m^S$. We shall refer to T_f as the ***evaluation tree*** of f (or of ***strat***(f)). We say that an innocent strategy-with-state is ***compact*** if it is generated by a view function whose domain of definition is finite. The last result of this section is the *effective* compositionality of *compact* innocent strategies-with-state, which we phrase as follows:

Theorem 3.6 (Effective Compositionality). *There is an algorithm that takes as input finite view functions f and g such that* ***strat***$(f) : A \Rightarrow B$ *and* ***strat***$(g) : B \Rightarrow C$ *are innocent strategies-with-state of finite arenas of order at most 3, and returns the evaluation tree T_h of a view function h such that* ***strat***$(h) =$ ***strat***(f) ; ***strat***$(g) : A \Rightarrow C$.

The algorithm generates interaction sequences between ***strat***(f) and ***strat***(g) which are *short-sighted* in the sense that any move in u which is an O-move of $A \Rightarrow C$ (C say) is explicitly justified by the preceding move in u (which must be a P-move in C, by the Switching Condition). Termination of the algorithm amounts to showing that there is no infinite short-sighted interaction sequences. The proof of the last fact uses a lemma due to Hyland [6]: For any infinite state-less play w over a finitary arena, either the set of P-views spanned by w (i.e. $\{ \ulcorner s \urcorner : s \leqslant w \}$) is infinite, or the set of O-views spanned by w is infinite.

4 Innocent representation of 3rd-order IA

Knowing-strategy semantics of IA. For notational simplicity, we shall write exp for the arena that denotes the type exp, similarly for loc and com. The arena exp is just the standard natural numbers arena with move-set $\{ q \} \cup \mathbb{N}$ where q is the initial question that justifies each answer $n \in \mathbb{N}$. The arena com is a two-move arena whose initial question *run* justifies the answer-move *done*. Following Reynolds, the type loc of locations is interpreted in an object-oriented style as a product of its "read method" and its "write method". Thus the arena loc is the product arena $\mathsf{exp} \times \mathsf{com}^\omega$, whereby the first component is the value that is held at the location, and the second component contains countably many commands to write 0, 1, 2, etc., to the location; we write the question-move and the answer-move in the i-th copy of com in loc as *write*(i) and *ok*$_i$ (or simply *ok*) respectively. We remark that com^ω is a *retract* of $\mathsf{exp} \Rightarrow \mathsf{com}$, a property that underpins the interpretation of dereferencing and assignment to variable.

We direct the reader to [11] (see also [2]) for an account of the interpretation of IA term-in-contexts as knowing strategies, written $[\![\Gamma \vdash M : A]\!]^\kappa$. Here we shall just consider the interpretation of the `new`-block, for which we need to introduce a non-innocent strategy. For each $i \geq 0$, we define a knowing strategy new_i over the arena $(\mathsf{loc} \Rightarrow \mathsf{exp}) \Rightarrow \mathsf{exp}$ by specifying its maximal plays to be plays that are (*qua* words over the relevant move-set) recognized by the left-linear[4] context-free grammar:

$$\begin{array}{rrcl} & S & \rightarrow & q_1 \cdot q_2 \cdot S_i \\ k \geq 0, & S_k & \rightarrow & \mathit{read} \cdot k \cdot S_k \\ & & + & \sum_{j \geq 0} \mathit{write}(j) \cdot \mathit{ok} \cdot S_j \\ & & + & \sum_{m \geq 0} m_2 \cdot m_1 \end{array} \tag{2}$$

The alphabet-set in question is the move-set of the arena $(\mathsf{loc} \Rightarrow \mathsf{exp}) \Rightarrow \mathsf{exp}$, which is an infinite set. To distinguish the two copies of exp, we mark moves from the rightmost copy of exp with subscript 1, and those from the other copy with subscript 2. The strategy new_i has precisely the behaviour expected of a "good variable" which has been initialized to i: the answer in response to a *read* is

[4]A context-free grammar is *left-linear* if every rule is either of the form $P \rightarrow w\,Q$ or of the form $P \rightarrow w$, where P and Q are non-terminal symbols and w is a word of the alphabet. A language is regular if and only if it is recognized by a left-linear grammar.

always the last value written, or i if no *write*(n) has yet been played. We can now define $[\![\,\Gamma \vdash \mathtt{new}\,x := n\,\mathtt{in}\,M : \beta\,]\!]^{\kappa}$ to be $[\![\,\Gamma \vdash \boldsymbol{\lambda} x : \mathsf{loc}.M : \mathsf{loc} \Rightarrow \beta\,]\!]^{\kappa}$ **;** new_n, where $\beta = \mathsf{exp}, \mathsf{com}$.

Using knowing-strategy semantics, observational preorder of IA may be characterized in terms of complete plays as follows (see [11] or [2] for a proof):

Theorem 4.1 (Characterization). *For any Γ, A, M_1 and M_2 such that $\Gamma \vdash M_i : A$, we have $M_1 \sqsubsetapprox M_2$ iff* ***cplays*** $[\![\,\Gamma \vdash M_1 : A\,]\!]^{\kappa} \subseteq$ ***cplays*** $[\![\,\Gamma \vdash M_2 : A\,]\!]^{\kappa}$.

The Characterization Theorem plays an important role in the main decidability result of this paper.

Innocent strategy-with-state representation. *Henceforth, without further mention, all arenas, strategies and IA terms-in-context are assumed to have order at most three.*

The innocent strategy-with-state denotation of 3rd-order IA terms-in-context, written $[\![\,\Gamma \vdash M : A\,]\!]$, is defined for all constructs, except the new-block, by exactly the same inductive clauses that define the knowing-strategy semantics. For the new-block, given any $i \geq 0$ and any location X, we define the view function $\mathsf{loc}_{X,i}$ over the arena $(\mathsf{loc} \Rightarrow \mathsf{exp}) \Rightarrow \mathsf{exp}$:

$$\mathsf{loc}_{X,i} \;:\; \left\{ \begin{array}{rcl} q_1 & \mapsto & q_2^{(X,i)} \\ q_1 \cdot q_2^{(X,i)} \cdot \mathit{read}^{(X,k)} & \mapsto & k^{(X,k)} \\ q_1 \cdot q_2^{(X,i)} \cdot \mathit{write}(j)^{(X,k)} & \mapsto & \mathit{ok}^{(X,j)} \\ q_1 \cdot q_2^{(X,i)} \cdot m_2^{(X,k)} & \mapsto & m_1^{(X,k)} \end{array} \right. \tag{3}$$

for each $j, k, m \geq 0$. As in (2), we use subscripts to distinguish moves from the two copies of exp; we write $m^{(X,i)}$ as a shorthand for the move whose state maps X to i. The idea behind the view function is simple: the state of a move in ***strat***$(\mathsf{loc}_{X,i})$ is given by the value held at the location X at that point. Thus the answer in response to a read-move $\mathit{read}^{(X,k)}$ is the value stored at X, which is k; the state can only be changed by a write-move: P's response to the O-move $\mathit{write}(j)^{(X,k)}$ is to change the state to (X, j). The reader may wish to check that the maximal plays of ***strat***$(\mathsf{loc}_{X,i})$, as sequences of moves, form a language recognized by the following left-linear grammar:

$$\begin{array}{rcl} S & \to & q_1 \cdot q_2^{(X,i)} \cdot S_i \\ k \geq 0, S_k & \to & \mathit{read}^{(X,k)} \cdot k^{(X,k)} \cdot S_k \\ & + & \sum_{j \geq 0} \mathit{write}(j)^{(X,k)} \cdot \mathit{ok}^{(X,j)} \cdot S_j \\ & + & \sum_{m \geq 0} m_2^{(X,k)} \cdot m_1^{(X,k)} \end{array} \tag{4}$$

This is just another description of the behaviour of a "good variable" initialized to i, which is equivalent to that specified by the knowing strategy new_i in the following sense:

Lemma 4.2. *For each $i \geq 0$,* ***erase*** $($***strat***$(\mathsf{loc}_{X,i})) = \mathsf{new}_i$.

The lemma is an immediate consequence of the fact that (2) is obtained from (4) by erasure. For $\beta = \mathsf{exp}$ and com, we are now in a position to define $[\![\,\Gamma \vdash \mathtt{new}\,x := n\,\mathtt{in}\,M : \beta\,]\!]$ to be

$$[\![\,\Gamma \vdash \boldsymbol{\lambda} x : \mathsf{loc}.M : \mathsf{loc} \Rightarrow \beta\,]\!]\;\boldsymbol{;}\;\textbf{\textit{strat}}(\mathsf{loc}_{X,n})$$

Thanks to Example 3.3(i), Lemma 4.2 and Theorems 3.4(ii) and 3.6, we can now clarify the relationship between the innocent-strategy denotation of a term and its knowing-strategy semantics: the former should be regarded as an algorithmic representation of the latter.

Theorem 4.3. *For any valid finitary 3rd-order IA term-in-context $\Gamma \vdash M : A$*

(i) we have ***erase*** $[\![\,\Gamma \vdash M : A\,]\!] = [\![\,\Gamma \vdash M : A\,]\!]^{\kappa}$.

(ii) Further if the types are built up from finite base types, then $[\![\,\Gamma \vdash M : A\,]\!]$, which is a compact innocent strategy-with-state, is effectively computable.

Strictly speaking, the innocent strategy-with-state denotation is not a semantics (as it is not a model of the theory of the language) of 3rd-order IA, but a *representation* of one such, namely, its knowing-strategy semantics.

Example 4.1. For ease of explanation we shall consider a 3rd-order type built up from a finite base type exp_2, the type of binary-valued expressions. Take the following terms:

$$\begin{array}{rcl} \Theta_1 & = & \boldsymbol{\lambda} F.\mathtt{new}\,x := 0\,\mathtt{in}\,F(\boldsymbol{\lambda} y.\Xi) \\ \Theta_2 & = & \boldsymbol{\lambda} F.F(\boldsymbol{\lambda} y.\mathtt{new}\,x := 0\,\mathtt{in}\,\Xi) \end{array}$$

of type $((\mathsf{exp}_2 \Rightarrow \mathsf{exp}_2) \Rightarrow \mathsf{exp}_2) \Rightarrow \mathsf{exp}_2$, where

$$\Xi \;=\; (\mathtt{ifzero}\,\mathbf{!}x\,\mathtt{then}\,x := y\,\mathtt{else}\,x := \neg y)\;\mathbf{;}\;\mathbf{!}x.$$

The view functions that generate $[\![\,\Theta_1\,]\!]$ and $[\![\,\Theta_2\,]\!]$ respectively are presented as evaluation trees in Table 2. In the Table, q_0, q_1, q_2 and q_3 are the four questions of the arena (the subscript is the type-theoretic order of the question); "m, i" is a shorthand for the move m whose state maps a fixed location X (say) to i, whereas "m" means that the state is null. (Note that the view function specified by the tree is just the map $p \mapsto m^{\{\,(X,i)\,\}}$ where $p \cdot$ "m, i" ranges over even-length path of the tree; the pointer of m into p is uniquely determined in each case.)

As an illustration of operational reasoning based on game semantics, we shall show $\Theta_1 \not\sqsubsetapprox \Theta_2$ and $\Theta_2 \not\sqsubsetapprox \Theta_1$. Using the left view function in Table 2, we construct the following legal position of moves-with-state:

$$\begin{array}{l} q_0\ (q_1, 0)\ (q_2, 0)\ (q_3, 0)\ (1, 0)\ (1, 1) \\ (q_2, 1)\ (q_3, 1)\ (0, 1)\ (1, 1)\ (1, 1)\ (1, 1) \end{array}$$

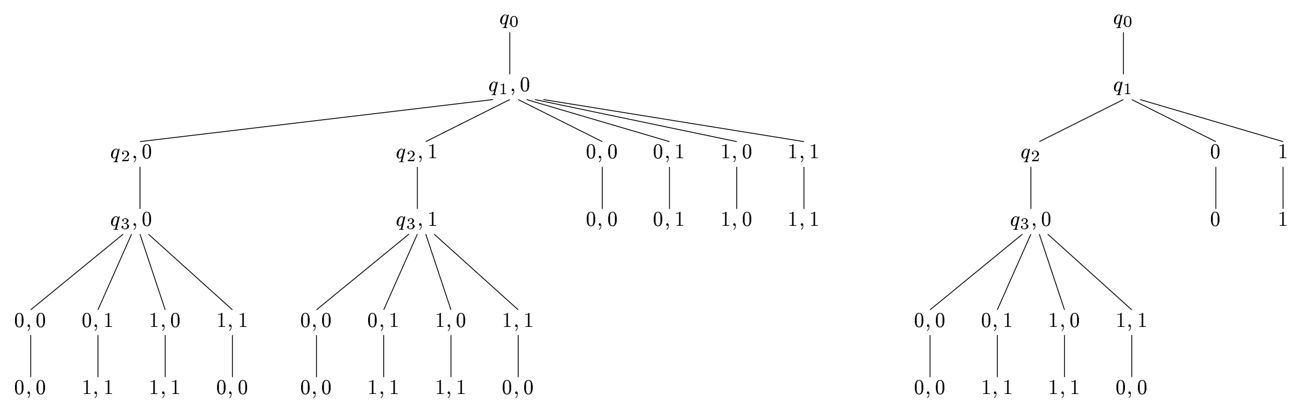

Table 2. View functions of Θ_1 (left) and Θ_2 (right) in Example 4.1

which is in $[\![\, \Theta_1 \,]\!]$. (Note that neither $q_0\ (q_1, 0)\ (q_2, 1)$ nor $q_0\ (q_1, 0)\ (q_2, 0)\ (q_3, 0)\ (1, 1)$ (say) are legal positions of $[\![\, \Theta_1 \,]\!]$ because condition (O) of State Change is violated.) Using the right view function in the Table, we have

$$q_0\ q_1\ q_2\ (q_3, 0)\ (1, 0)\ (1, 1)\ q_2\ (q_3, 0)\ (0, 0)\ (0, 0)\ 0\ 0$$

which is in $[\![\, \Theta_2 \,]\!]$. (Again the justification pointers in the legal positions above are uniquely reconstructible.) Thus, by Theorem 4.3, we obtain the following (complete) plays by erasure:

$$\begin{array}{lcl} q_0\ q_1\ q_2\ q_3\ 1\ 1\ q_2\ q_3\ 0\ 1\ 1\ 1 & \in & \textbf{\textit{erase}}\,[\![\, \Theta_1 \,]\!] = [\![\, \Theta_1 \,]\!]^{\kappa} \\ q_0\ q_1\ q_2\ q_3\ 1\ 1\ q_2\ q_3\ 0\ 0\ 0\ 0 & \in & \textbf{\textit{erase}}\,[\![\, \Theta_2 \,]\!] = [\![\, \Theta_2 \,]\!]^{\kappa} \end{array}$$

Observe that the first legal position above does not belong to $[\![\, \Theta_2 \,]\!]^{\kappa}$ by Determinacy; similarly nor does the second belong to $[\![\, \Theta_1 \,]\!]^{\kappa}$. Hence, thanks to Theorem 4.1, we have $\Theta_1 \not\precsim \Theta_2$ and $\Theta_2 \not\precsim \Theta_1$. As an exercise in much the same vein, the reader may wish to prove that $\Theta_2 \approx \boldsymbol{\lambda} F.F(\boldsymbol{\lambda} y.y)$ by showing that ***cplays*** $[\![\, \Theta_2 \,]\!]^{\kappa} =$ ***cplays*** $[\![\, \boldsymbol{\lambda} F.F(\boldsymbol{\lambda} y.y) \,]\!]^{\kappa}$.

Further, by the definability theorem in [2], given a complete play that belongs to the knowing-strategy denotation of one term but not that of the other, we can construct a program context that separates the two terms. For example, a context that is constructed from the complete play in $[\![\, \Theta_1 \,]\!]^{\kappa}$ above is

$$D[\text{-}] \;=\; [\text{-}](\boldsymbol{\lambda} g.\,\mathtt{ifzero}\,(g1)\,\mathtt{then}\,\Omega\,\mathtt{else}\,(g0));$$

we have $D[\Theta_1] \Downarrow 1$ but $D[\Theta_2] \Downarrow 0$.

5 DPDA characterization

Encoding pointers by view offset. We use a simple encoding scheme, called ***view offset***, for representing legal positions as words of an alphabet. For any even-length legal position $s\,m$, we encode the pointer of the P-move m to q (say), which must appear in $\ulcorner s \urcorner$ by Visibility, by a numeric offset n which is defined to be the number of pending O-questions that occur *after* q in the P-view $\ulcorner s \urcorner$. Similarly the pointer of an O-move can be given in terms of an offset relative to the O-view at that point. The move m with view offset n may then be encoded by (say) the word $o^n\, m$ – the prefix o^n is taken to be the unary numeral n. A legal position can thus be represented as a concatenation of such words. Note that the view-offset scheme is quite general: it is for legal positions of all finite orders.

The encoding scheme may be simplified in several ways. First there is no need to represent explicitly the pointer of an answer-move: by decreeing that such a pointer is always to the last pending question, we can use a pushdown stack to verify Well-Bracketing. Secondly, in the 3rd-order case, a further simplification is possible, thanks to the following lemma.

Lemma 5.1. *Let s be an even-length legal position of a 3rd-order arena. If there is a pending first-order P-question in s, then there is a* unique *pending first-order P-question (namely the 2nd element) in $\llcorner s \lrcorner$, and that question must also appear in $\ulcorner s \urcorner$. Further if another legal position t has the same P-view as s, then $\llcorner s \lrcorner$ and $\llcorner t \lrcorner$ contain the same unique pending first-order question.*

By the Lemma (applied to $t = \ulcorner s \urcorner$), the pointer of an O-question (which is necessarily 2nd-order, as the 0th-order O-question has no pointer by definition) is to the *unique* first-order pending P-question that appears in the O-view, and that P-question is guaranteed to appear in the *P-view* at that point. Therefore there is no need to consider O-views at all! Indeed because the justifying P-question is unique, there is no need to represent the pointer of any 2nd-order question. There is also no need to represent the pointer of any first-order question as it must be to the opening question. Thus *the only moves in any legal position (of a 3rd-order arena) whose pointers must be coded explicitly are 3rd-order P-questions.*

Finally in the case of 3rd-order *compact* innocent strategy $\boldsymbol{strat}(f)$ generated by some finite view function f, yet another simplification is possible. Instead of coding a 3rd-order P-question $\boldsymbol{(}$, whose pointer has offset i, as $o^i\boldsymbol{(}$, we can simply introduce a *new* symbol as a name for the word $o^i\boldsymbol{(}$ – this is quite harmless as the offsets i are bounded by the (half the) maximum length of the P-views in $\mathrm{dom}(f)$.

Using the view-offset encoding, we can consider the set of complete plays of a knowing strategy denoting a 3rd-order finitary IA term as a set of words. In general such sets are *not* regular, as the following example shows.

Example 5.1. Set $A = ((o \Rightarrow o) \Rightarrow o) \Rightarrow o$ where o is any arena that has only one question and one answer (such as com). Take σ to be $[\![\, \boldsymbol{\lambda} F.F(\boldsymbol{\lambda} x.x)\,]\!]^\kappa$ which is an innocent strategy over A. We write the four questions of A as $\boldsymbol{[}_0, \boldsymbol{(}_1, \boldsymbol{[}_2$ and $\boldsymbol{(}_3$, and their corresponding answers as $\boldsymbol{]}_0, \boldsymbol{)}_1, \boldsymbol{]}_2$ and $\boldsymbol{)}_3$ respectively; the (type-theoretic) order of each move is annotated as a subscript, and we take $\boldsymbol{(}_3$ to mean "the 3rd-order P-question with view offset 0". In the following we consider complete plays of σ as words over the alphabet $\Sigma = \{\, \boldsymbol{[}_0, \boldsymbol{(}_1, \boldsymbol{[}_2, \boldsymbol{(}_3, \boldsymbol{]}_0, \boldsymbol{)}_1, \boldsymbol{]}_2, \boldsymbol{)}_3 \,\}$. For each $n \geq 2$, we define a word

$$z_n \;=\; \boldsymbol{[}_0\, \boldsymbol{(}_1\, \underbrace{\boldsymbol{[}_2\, \boldsymbol{(}_3 \cdots \boldsymbol{[}_2\, \boldsymbol{(}_3}_{2(\lfloor n/2 \rfloor)\text{ moves}}\, \boldsymbol{)}_3\, \boldsymbol{]}_2 \cdots \boldsymbol{)}_3\, \boldsymbol{]}_2\, \boldsymbol{)}_1\, \boldsymbol{]}_0$$

Each z_n has length $4(\lfloor n/2 \rfloor) + 4 > n$ and is a complete play of σ. For any $u, v, w \in \Sigma^*$ such that $z_n = u\,v\,w \in \boldsymbol{cplays}\,\sigma$, and $|v| \geq 1$, and $u\,v^i\,w \in \boldsymbol{cplays}\,\sigma$ for each $i \geq 0$, it follows from Well-Bracketing that v must have the form

$$\underbrace{\boldsymbol{[}_2\, \boldsymbol{(}_3 \cdots \boldsymbol{[}_2\, \boldsymbol{(}_3}_{j\text{ moves}}\, \boldsymbol{)}_3\, \boldsymbol{]}_2 \cdots \boldsymbol{)}_3\, \boldsymbol{]}_2$$

such that $j \leq 2(\lfloor n/2 \rfloor)$, which implies that $|u\,v| > n$. Writing $L = \boldsymbol{cplays}\,\sigma$, we have shown: for every $n \geq 0$, there exists $z_n \in L$ such that $|z_n| \geq n$ and for all u, v, w, if $z_n = u\,v\,w \wedge |v| \geq 1 \wedge \forall i \geq 0 \,.\, u\,v^i\,w \in L$ then $|u\,v| > n$. Thus, by the Pumping Lemma for Regular Languages, $\boldsymbol{cplays}\,\sigma$ is not regular.

Recall that a ***deterministic pushdown automaton*** (DPDA) is specified by a 6-tuple $P = \langle\, Q, \mathsf{init}, \Sigma, \Gamma, \bot, \delta \,\rangle$ where Q is a finite set of *(control) states*, $\mathsf{init} \in Q$ is the initial state, Σ is a finite set of *input symbols*, Γ is a finite set of *stack symbols*, $\bot \in \Gamma$ is an auxiliary symbol indicating bottom-of-stack (initially the DPDA stack consists of one instance of $\bot$ and nothing else), and δ is a finite set of *transitions*, each of the form

$$p, Z \xrightarrow{a} q, \alpha$$

where p, q are states, $a \in \Sigma \cup \{\, \epsilon \,\}$, $Z \in \Gamma$, and α is a finite sequence of stack symbols, subject to the following restriction:

(1) If $p, Z \xrightarrow{a} q, \alpha$ and $p, Z \xrightarrow{a} r, \beta$ and $a \in \Sigma \cup \{\, \epsilon \,\}$ then $q = r$ and $\alpha = \beta$.

(2) If $p, Z \xrightarrow{\epsilon} q, \alpha$ and $p, Z \xrightarrow{a} r, \beta$ then $a = \epsilon$.

A DPDA is said to be ***real-time*** just in case it has no ϵ-transitions.

A *configuration* of a DPDA is a pair (q, α) where q is a state and $\alpha \in \Gamma^*$ is the stack contents. The transitions of a configuration are defined by the following *prefix rule*:

> If $p, Z \xrightarrow{a} q, \alpha$ is a transition in δ then for any $\gamma \in \Gamma^*$, we have $p, \gamma Z \xrightarrow{a} q, \gamma\alpha$

(which we read as: by consuming a from the input, and replacing Z on top of the stack by α, we can go from state p to state q). The transition relation $\xrightarrow{a}$ between configurations where $a \in \Sigma$, is then standardly extended to words $\xrightarrow{w}$ where $w \in \Sigma^*$. Acceptance is by empty stack. Formally the ***language recognized by the DPDA*** P, written $\mathcal{L}(P)$, is defined to be

$$\mathcal{L}(P) \;=\; \{\, w \in \Sigma^* : \exists q \in Q \,.\, \mathsf{init}, \bot \xrightarrow{w} q, \bot \,\}$$

I.e. $\mathcal{L}(P)$ is the set of inputs that P can consume and at the same time empty its stack.

The ***DPDA Equivalence Problem*** ("Is there an effective procedure for deciding whether any two DPDAs recognize the same language?") was first posed in 1966. Restricted to the real-time case, the problem was solved positively by Oyamaguchi *et al* in 1980 [14]. The general decidability problem was only solved recently, also positively, by Sénizergues [17] (solution announced in 1997). A simpler proof and a primitive recursive complexity bound were subsequently obtained by Stirling [18, 19].

Construction of the DPDA P_f. Let f be a finite view function that generates an innocent strategy-with-state over a 3rd-order arena A. We shall construct a real-time DPDA P_f that recognizes the complete plays of the knowing strategy $\boldsymbol{erase}\,(\boldsymbol{strat}(f))$. Formally the real-time DPDA

$$\mathsf{P}_f \;=\; \langle\, Q_f, \mathsf{init}, \Sigma_f, \Gamma_f, \bot, \delta_f \,\rangle$$

is defined as follows. The alphabet-set Σ_f contains the move-set M_A of the arena A, and for each 3rd-order P-question $\boldsymbol{(}_3$ with view-offset i that appears in the range of f, we introduce a new symbol for it in Σ_f. We define the state-set $Q_f = \{\, \mathsf{init} \,\} \cup \mathrm{dom}(f) \cup \{\, p\, m^S : f(p) = m^S \,\}$. I.e., other than the initial state, every state is either an (odd-length) P-view in the domain, or an (even-length) P-view that is a section of the graph of f. The set Σ_f of stack symbols consists of the f-reachable questions of A. An *f-reachable question* of A is either an O-question $[^S$ whereby $p\,[^S \in \mathrm{dom}(f)$ for some p, or it is a P-view of the form $p\,(^S$ such that f maps p to $(^S$. Clearly Γ_f is a finite set. The transitions of P_f, which are presented in Table 3, are organized into four types:

Type	*State Before* (P-views)	*Input Symbol*	*State After* (P-views)	*Stack Action*
1	odd	P qn.	even	push
2	odd	P ans.	even	pop
3	even (or init)	O qn.	odd	push
4	even	O ans.	odd	pop

For example a transition $p, Z \xrightarrow{a} q, \alpha$ is type-1 just in case p is an odd-length P-view, a is a P-question, q is an even-length P-view, and $\alpha = Z\,Z'$ for some Z'. For ease of reading, we present each transition $p, Z \xrightarrow{a} q, \alpha$ as a block of the form $\boxed{\;p \;\mid\; Z \;\mid\; a \;\mid\; q \;\mid\; \alpha\;}$ in Table 3; where appropriate, we shall write a move of order k as m_k to aid reading. By inspecting the four types of transitions in turn, it is straightforward to verify that the automaton P_f thus defined is a real-time DPDA.

We give the main technical result of the section:

Theorem 5.2. *For any compact innocent strategy-with-state **strat**(f) of a 3rd-order arena, we have $\mathcal{L}(\mathsf{P}_f)$ = **cplays** (**erase** (**strat**(f))). Hence, by Theorem 4.3, (the set of complete plays of) the knowing-strategy semantics of a 3rd-order finitary IA term is deterministic context-free.*

We briefly explain the idea behind Theorem 5.2. The tasks of verifying that an input word is a (complete) play in ***erase strat***(f) have been built into the transitions of the DPDA P_f. The state that is reached, at a given point during a computation of P_f, is precisely the P-view of the prefix, s say, of the input word which has been read thus far; and the stack at that point contains exactly the subsequence of pending (f-reachable) questions in s. The stack actions ensure that the computation admits only moves that observe Well-Bracketing. If the prefix s is odd-length, only the P-move $m^S = f(\ulcorner s \urcorner)$ may fire — see transitions of Types 1 and 2. In case s is even-length, all *legal* O-moves m (i.e. those satisfying O-visibility, Well-Bracketing and State Change) such that $\ulcorner s\,m^S \urcorner \in \mathrm{dom}(f)$ may fire — see transitions of Types 3 and 4.

For a simple example, we consider the DPDA of a stateless view function as follows.

Example 5.2. Take the term $\lambda F.F(\lambda x.x)$ of type $((o \Rightarrow o) \Rightarrow o) \Rightarrow o$, first considered in Example 5.1 (we use the notations therein). The transitions (less the stack actions) of the associated DPDA are shown in Figure 1. Consider the Type-4.3 transition marked † which we give in full as follows:

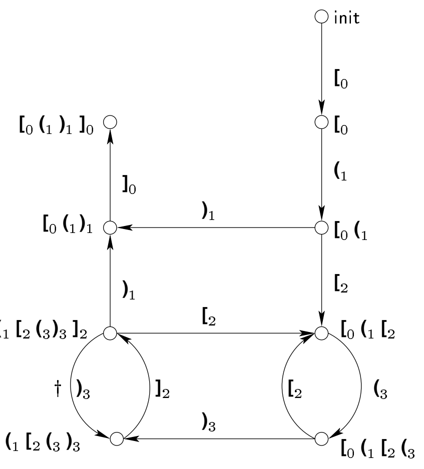

Figure 1. Transitions of the DPDA in E.g. 5.2

$$[_0\,(_1\,[_2\,(_3\,)_3\,]_2, \text{“}[_0\,(_1\,[_2\,(_3\text{”} \xrightarrow{)_3} [_0\,(_1\,[_2\,(_3\,)_3, \epsilon$$

Note that the P-question to which the input symbol $)_3$ is an answer does not appear in the state (i.e. P-view) before the transition (it occurs in the segment of the play between $(_1$ and $[_2$ left out of the P-view); we rely on the "symbol" on the top of stack – "$[_0\,(_1\,[_2\,(_3$" which is the last pending (f-reachable) question – to work out the P-view after the move. This is the reason why the stack symbols are *f-reachable* questions, and not simply questions.

Corollary 5.3. *For the same f as before, plays of the knowing strategy **erase** (**strat**(f)), plays of the innocent strategy-with-state **strat**(f), and complete plays of **strat**(f) are all real-time deterministic context-free.*

As a corollary of Theorem 5.2, we have the main result of the paper:

Theorem 5.4 (Decidability). *Observational preorder, and hence observational equivalence, of 3rd-order finitary IA (built up from finitary base types) is decidable.*

Proof. Take 3rd-order IA-terms M_1 and M_2 such that $\Gamma \vdash M_i : A$ is provable for some Γ and A. By Theorem 3.6, their respective innocent strategy-with-state denotations $[\![\,\Gamma \vdash M_i : A\,]\!]$ (as finite view functions) are computable. By Theorem 5.2 and since it is decidable whether any two real-time DPDAs recognize the same language, we have "***cplays*** (***erase*** $[\![\,\Gamma \vdash M_1 : A\,]\!]) \subseteq$ ***cplays*** (***erase*** $[\![\,\Gamma \vdash M_2 : A\,]\!]$)" is decidable. Hence, by Theorem 4.3, we have "***cplays*** $[\![\,\Gamma \vdash M_1 : A\,]\!]^{\kappa} \subseteq$ ***cplays*** $[\![\,\Gamma \vdash M_2 : A\,]\!]^{\kappa}$" is decidable. Finally, by Theorem 4.1, we have "$M_1 \sqsubseteq M_2$" is decidable, as required. $\square$

Type-1: For each odd-length P-view p, and each P-question $\boldsymbol{(}^S$ such that $f : p \mapsto \boldsymbol{(}^S$

p	Z	$\boldsymbol{(}$	$p\,\boldsymbol{(}^S$	Z “$p\,\boldsymbol{(}^S$”

Type-2: For each odd-length P-view p, and each P-answer $\boldsymbol{]}^S$, such that $f : p \mapsto \boldsymbol{]}^S$ which is explicitly justified by $\boldsymbol{[}^{S_0}$ in p

p	$\boldsymbol{[}^{S_0}$	$\boldsymbol{]}$	$p\,\boldsymbol{]}^S$	ϵ

Type-3.1: For each initial question $\boldsymbol{[}_0$

init	$\perp$	$\boldsymbol{[}_0$	$\boldsymbol{[}_0$	$\perp\boldsymbol{[}_0$

Type-3.2: For each odd-length P-view of the form $p\,\boldsymbol{(}_1^{S_0}\,w$ such that $f : p\,\boldsymbol{(}_1^{S_0}\,w \mapsto b^{S_1}$ where $\boldsymbol{(}_1^{S_0}$ is the unique first-order P-question that appears in $\llcorner p\,\boldsymbol{(}_1^{S_0}\,w\,b^{S_1}\lrcorner$ (see Lemma 5.1), and for each 2nd-order O-question $\boldsymbol{[}_2$ such that $\boldsymbol{(}_1 \vdash_A \boldsymbol{[}_2$

$p\,\boldsymbol{(}_1^{S_0}\,w\,b^{S_1}$	Z	$\boldsymbol{[}_2$	$p\,\boldsymbol{(}_1^{S_0}\,\boldsymbol{[}_2^{S_0[S_1]}$	$Z\,\boldsymbol{[}_2^{S_0[S_1]}$

Type-4.1: For each odd-length P-view p such that $f : p \mapsto \boldsymbol{(}^S$, for each O-answer $\boldsymbol{)}$ such that $\boldsymbol{(} \vdash_A \boldsymbol{)}$

$p\,\boldsymbol{(}^S$	“$p\,\boldsymbol{(}^S$”	$\boldsymbol{)}$	$p\,\boldsymbol{(}^S\,\boldsymbol{)}^S$	ϵ

Type-4.2: For each odd-length P-view of the form $p\,\boldsymbol{(}_1^{S_0}\,\boldsymbol{[}_2^T\,w$ where $\boldsymbol{(}_1^{S_0}$ is a first-order P-question such that $f : p\,\boldsymbol{(}_1^{S_0}\,\boldsymbol{[}_2^T\,w \mapsto \boldsymbol{]}_2^{S_1}$, which is explicitly justified by $\boldsymbol{[}_2^T$, and for each first-order O-answer $\boldsymbol{)}_1$ such that $\boldsymbol{(}_1 \vdash_A \boldsymbol{)}_1$

$p\,\boldsymbol{(}_1^{S_0}\,\boldsymbol{[}_2^T\,w\,\boldsymbol{]}_2^{S_1}$	“$p\,\boldsymbol{(}_1^{S_0}$”	$\boldsymbol{)}_1$	$p\,\boldsymbol{(}_1^{S_0}\,\boldsymbol{)}_1^{S_0[S_1]}$	ϵ

Type-4.3: For each odd-length P-view of the form $p\,\boldsymbol{(}_1^{T_0}\,\boldsymbol{[}_2^T\,w$ where $\boldsymbol{(}_1^{T_0}$ is a first-order P-question such that $f : p\,\boldsymbol{(}_1^{T_0}\,\boldsymbol{[}_2^T\,w \mapsto \boldsymbol{]}_2^{S_1}$, and for $f : p\,\boldsymbol{(}_1^{T_0}\,u \mapsto \boldsymbol{(}_3^{S_0}$ and each 3rd-order O-answer $\boldsymbol{)}_3$ such that $\boldsymbol{(}_3 \vdash_A \boldsymbol{)}_3$

$p\,\boldsymbol{(}_1^{T_0}\,\boldsymbol{[}_2^T\,w\,\boldsymbol{]}_2^{S_1}$	“$p\,\boldsymbol{(}_1^{T_0}\,u\,\boldsymbol{(}_3^{S_0}$”	$\boldsymbol{)}_3$	$p\,\boldsymbol{(}_1^{T_0}\,u\,\boldsymbol{(}_3^{S_0}\,\boldsymbol{)}_3^{S_0[S_1]}$	ϵ

Table 3. Transitions of the DPDA P_f

It is not clear what the complexity of deciding observational equivalence of 3rd-order finitary IA is. Our decision procedure is dominated by the computation of the view-function denotation of the two IA terms being compared, and by the procedure that decides equivalence of the two derived real-time DPDAs. The former is in essence a normalization procedure (which is essentially *linear head reduction* for a sublanguage consisting of appropriate canonical forms of 3rd-order finitary IA); the complexity of the latter is a topic of current research. Stirling has suggested that his algorithm in [19] can be modified to give an elementary upper bound for deciding the equivalence of *real-time* DPDAs.

An undecidability result. Ghica and McCusker's decidability result applies to 2nd-order finitary IA augmented by *iteration* in the form of while-loops. Is observational equivalence still decidable for the same fragment augmented by the fixpoint operator (so that one can compute fixpoints of terms of order 2)? The answer is no.

Theorem 5.5 (Undecidability). *Assuming that base types are finite, observational preorder of 2nd-order IA with recursion (thus admitting recursively-defined first-order functions) is undecidable.*

The proof uses an idea in [8]. Fix a finite alphabet Σ, and consider a programming system called Queue, which has a single memory cell X, a queue (unbounded, first-in-first-out) data structure, and four instructions, namely, `enqueue` a (write $a \in \Sigma$ onto the right end of the queue), `dequeue` X

(if the queue is empty, halt; otherwise remove the leftmost symbol from the queue and write it to X), `if` $X = a$ `goto` L, and `halt`, where $L \geq 0$ is a label. A *Queue program* is a finite sequence of the form $1 : I_1,\ 2 : I_2,\ \cdots\ n : I_n$, where each I_i is an instruction. By simulating Post's *Tag Systems* [12] in Queue, it is known that the problem QUEUE-HALTING: "Given a Queue program, will it halt eventually?" is undecidable. Given a Queue program P, we construct an IA program M_P of the fragment in question that simulates P. Plainly the IA program M_P eventually halts if and only if it is observationally equivalent to the program `skip`. Undecidability of observational equivalence for this fragment of IA then follows from the undecidability of QUEUE-HALTING.

6 Conclusions and further directions

In this paper we have answered the following questions:

1. Does Ghica and McCusker's regular-expression representation of 2nd-order IA extend to the 3rd-order fragment? *No: Example 5.1.*

2. What automata-theoretic formalism is needed to model 3rd-order IA without recursion? *Deterministic context-free languages: Theorem 5.2.*

3. Is observational preorder of 3rd-order IA without recursion decidable? *Yes: Theorem 5.4.*

4. Is observational preorder of 2nd-order IA with recursion decidable? *No: Theorem 5.5.*

We intend to investigate applications of this work to Software Model Checking. In addition to model-checking observational equivalence which is of basic importance, the same algorithmic representations of program meanings which are derived in this work can be put to use in verifying a wide range of program properties of IA and cognate programming languages, and in practice this is where the interesting applications are most likely to lie.

Some open problems. It is of practical importance to understand the complexity of deciding observational equivalence of 3rd-order finitary IA, and to extract an efficient algorithm tailored to the task at hand. A challenging open problem is to give an automata-theoretic characterization of (the knowing-strategy semantics of) 4th-order IA. In view of Theorem 5.5, a natural question is whether observational equivalence of 2nd-order *call-by-value* IA with recursion is decidable; a positive answer would be good news from a model-checking perspective because of the direct relevance to procedural languages such as C. Finally it would be interesting to identify fragments of *Reduced ML* (call-by-value PCF augmented by statically scoped, dynamically allocated, mutable state, with equality test for references, as studied in [15]) that have decidable observational equivalence. Our preferred approach is by Algorithmic Game Semantics, but to our knowledge, the prior (for this approach) problem of the existence of a fully abstract game semantics for Reduced ML is still open.

Acknowledgements. The author is grateful to Samson Abramsky and Andrzej Murawski for helpful discussions.

References

[1] S. Abramsky. Semantics via game theory. Lecture slides. Marktoberdorf International Summer School, 2001.
[2] S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In [13]
[3] D. R. Ghica. A regular-language model for Hoare-style correctness statements. In *Proc. VCL'2001*.
[4] D. R. Ghica. Regular language semantics for a call-by-value programming language. In *Proc. 17th Conf. MFPS*, 2001.
[5] D. R. Ghica and G. McCusker. Reasoning about idealized algol using regular languages. In *Proc. ICALP'00*, pages 103–116. Springer-Verlag, 2000. LNCS Vol. 1853.
[6] J. M. E. Hyland. Game semantics. In A. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*, pages 131–182. Cambridge Univ. Press, 1997.
[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*, 163:285–408, 2000.
[8] N. D. Jones and S. S. Muchnick. Complexity of finite memory programs with recursion. *JACM*, 25:312–321, 1978.
[9] R. Loader. Finitary PCF is not decidable. *TCS*, 266:342–364, 2001.
[10] G. McCusker. *Games for recursive types*. BCS Distinguished Dissertation. Cambridge University Press, 1998.
[11] G. McCusker. On the semantics of Idealized Algol without the bad-variable constructor. Technical Report 01/01, SOCS, University of Sussex, 2001.
[12] M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.
[13] P. W. O'Hearn and R. D. Tennent. *Algol-like Languages: Volumes I and II*. Biekhäuser, 1997.
[14] M. Oyamaguchi, H. Honda, and Y. Inagaki. The equivalence problem for real-time strict deterministic languages. *Information and Control*, 45:90–115, 1980.
[15] A. M. Pitts. Operational semantics and program equivalence. APPSEM 2000 Summer School Lectures, Portugal, 9-15 Sep. 2000.
[16] J. C. Reynolds. The essence of Algol. In de Bakker and van Vliet, editors, *Algorithmic Languages*, pages 345–372. North Holland, 1978.
[17] G. Sénizergues. Complete formal systems for equivalence problems. *TCS*, 231:309–334, 2001.
[18] C. Stirling, Decidability of DPDA equivalence. *Theoretical Computer Science*, 255:1–31, 2001.
[19] C. Stirling. Deciding DPDA equivalence is primitive recursive. Ftp-able preprint, 2001.