Idealized Algol with Ground Recursion, and DPDA Equivalence*

A. S. Murawski[1] C.-H. L. Ong[1] I. Walukiewicz[2]

[1] Oxford University Computing Laboratory, Parks Road, Oxford OX1 3QD, UK
[2] CNRS, Université Bordeaux-1, 351, Cours de la Libération, 33 405, Talence, France

Abstract. We prove that observational equivalence of $\mathsf{IA}_3 + \mathbf{Y}_0$ (3rd-order Idealized Algol with 0th-order recursion) is equivalent to the DPDA Equivalence Problem, and hence decidable. This completes the classification of decidable fragments of Idealized Algol. We also prove that observational approximation of $\mathsf{IA}_1 + \mathbf{Y}_0$ is undecidable by reducing the DPDA Containment Problem to it.

1 Introduction

Observational equivalence is an extensional notion of program equivalence. Two program phrases are *observationally equivalent* if one can be replaced by the other in any program without causing any observable difference to the computational outcome. Reynolds's Idealized Algol (IA) is an elegant and compact programming language that combines imperative programming with high-order features, mediated by a simple type theory. Observational equivalence in IA is in general undecidable even when ground types are finite sets. This paper is concerned with the question of decidability of observational equivalence for appropriate fragments of IA.

We begin with a quick review of IA. Ground types of IA, which are ranged over by β, are *exp* (expressions), *com* (commands) and *var* (assignable variables). Types of IA, ranged over by θ, θ' etc., are generated from ground types by the function space constructor $\theta \rightarrow \theta'$. The *order* of a type is defined by $\mathsf{ord}(\beta) = 0$ and $\mathsf{ord}(\theta \rightarrow \theta') = \max(\mathsf{ord}(\theta) + 1, \mathsf{ord}(\theta'))$. *Finitary* Idealized Algol, IA_f, is just recursion-free Idealized Algol over finite ground types. We can extend IA_f with iteration by adding the rule

$$\frac{\Gamma \vdash M : \mathit{exp} \quad \Gamma \vdash N : \mathit{com}}{\Gamma \vdash \mathbf{while}\ M\ \mathbf{do}\ N : \mathit{com}}$$

and with general recursion by adding the rule

$$\frac{\Gamma, x : \theta \vdash M : \theta}{\Gamma \vdash \mu x^\theta . M : \theta}.$$

* Work supported by the UK EPSRC (GR/R88861/01), European Research Training Network GAMES and St John's College, Oxford.

We call a term an *ith-order term* provided its typing derivation uses exclusively judgments of the shape $\Gamma \vdash M : \theta$ where the types of the free identifiers in Γ have order less than i and $\mathsf{ord}(\theta) \leq i$. The collection of ith-order IA_f (resp. IA_f+**while**) terms will be denoted by IA_i (resp. IA_i+**while**). We write $\mathsf{IA}_i+\mathbf{Y}_j$ to refer to ith-order IA_f terms with recursion that can be typed using the recursion rule in which $\mathsf{ord}(\theta) \leq j$. We tabulate all known results on the complexity of observational equivalence of β-normal terms in IA [1–4]:

	pure	+**while**	$+\mathbf{Y}_0$	$+\mathbf{Y}_1$
IA_1	coNP	Pspace	?	$= \mathsf{IA}_1 + \mathbf{Y}_0$
IA_2	Pspace	Pspace	?	undecidable
IA_3	Exptime	Exptime	?	undecidable
IA_4	undecidable	undecidable	undecidable	undecidable

The same results as above also hold for observational approximation.

This paper addresses the cases marked with question marks. In $\mathsf{IA}_i + \mathbf{Y}_0$ only programs of ground type can call themselves recursively. For example, while-loops **while** b **do** c can then be defined by

$$b : exp,\, c : com,\, z \vdash \mu z^{com}.\ \mathbf{ifzero}\, b\, skip\, (c; z).$$

We show that observational equivalence in $\mathsf{IA}_i + \mathbf{Y}_0$ is decidable for $i = 1, 2, 3$ by giving a reduction to the DPDA Equivalence Problem (recently proved decidable by Sénizergues [8]). This does not tell us much about the complexity, though. At the moment it is only known that the complexity of DPDA Equivalence is bounded by a primitive recursive function [5]. We also show that already for $i = 1$ observational equivalence is at least as hard as DPDA Equivalence. In consequence, no advance on the complexity of the former problem can be made without an advance on the latter. Another result is that observational approximation in $\mathsf{IA}_i + \mathbf{Y_0}$ is undecidable for $i = 1, 2, 3$, because the undecidable DPDA Containment Problem [6] can be reduced to it.

Let us comment on the relationship of our results to a recent paper [4] showing decidability of IA_3 + **while**. In that paper a simpler language was considered but it was translated to a weaker form of pushdown automata. This was essential to get a precise complexity bound. In this paper we model a richer language but, for reasons explained above, we do not concern ourselves with complexity, hence our constructions are not designed to optimize the size of the resulting automata. In [4] most constructions relied on parallel composition of automata. This is not possible here as DPDAs are not closed under that operation.

2 Game semantics: complete plays

We assume familiarity with the treatment of game semantics of IA as presented, for example, in [7]. Recall that the *multiplicative composition* $\sigma;_\mathrm{m} \tau : A \multimap C$ of two strategies $\sigma : A \multimap B$ and $\tau : B \multimap C$ is defined by parallel composition with hiding by letting the strategies interact in the shared subgame B and subsequently hiding the B-moves.

Let $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$. Recall that $A \Rightarrow B = !A \multimap B$ and $B \Rightarrow C = !B \multimap C$. In order to compose the strategies, one first defines $\sigma^\dagger : !A \multimap !B$ by

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

where $s \upharpoonright m$ stands for the subsequence of s (pointers included) whose moves are hereditarily justified by m. Then $\sigma;\tau : A \Rightarrow C$ is taken to be $\sigma^\dagger ;_{\mathrm{m}} \tau$.

Given a set σ of positions on G we write $\mathcal{L}(\sigma)$ for the set of the underlying sequences of moves from M_G. For a given strategy σ, we write $\mathsf{comp}\,\sigma$ for the set of its non-empty plays in which the number of questions matches the number of answers; such plays are called *complete*.

Remark 1. We will be interested in the $\dagger$ construction when $B = [\![\theta]\!]$ and $\mathsf{ord}(\theta) \leq 1$. Then $\sigma^\dagger$ can be characterized explicitly as follows.

- $B = [\![\beta]\!]$: Then $\sigma^\dagger = (\mathsf{comp}\,\sigma)^*\,\sigma$, i.e. $\sigma^\dagger$ simply iterates σ.
- $B = [\![\beta_1 \to \beta_0]\!]$: Then the switching conditions in the game $!A \multimap !B$ imply that a new copy of σ can be started each time σ is finished (as above) and, additionally, after each question q_1 from β_1. We can thus capture $K = \{\,\epsilon\,\} \cup \mathsf{comp}\,(\sigma^\dagger)$ by the equation below.

  $$K = \{\varepsilon\} \cup \bigcup \{ q_0 U q_1 K a_1 U \cdots q_1 K a_1 U a_0 K \mid q_0 U q_1 a_1 U \cdots q_1 a_1 U a_0 \in \mathsf{comp}\,\sigma \},$$

  where U's stand for (possibly empty and possibly different) segments of moves from A.

The operational semantics of IA can be found in [7]. We write $M \Downarrow$ if the closed term M reduces to *skip*. Recall that two terms $\Gamma \vdash M_1, M_2 : \theta$ are *observationally equivalent*, written $\Gamma \vdash M_1 \cong M_2$, if for any context $C[-]$ such that $C[M_1]$ and $C[M_2]$ are closed terms of type *com*, we have $C[M_1]\Downarrow$ iff $C[M_2]\Downarrow$. Similarly, M_1 *observationally approximates* M_2, written $\Gamma \vdash M_1 \sqsubseteq\!\!\!\sim M_2$, just if for all contexts satisfying the properties above, $C[M_1]\Downarrow$ implies $C[M_2]\Downarrow$.

Theorem 1 ([7]). *$\Gamma \vdash M_1 \sqsubseteq\!\!\!\sim M_2$ iff $\mathsf{comp}\,[\![\Gamma \vdash M_1]\!] \subseteq \mathsf{comp}\,[\![\Gamma \vdash M_2]\!]$. Consequently, $\Gamma \vdash M_1 \cong M_2$ iff $\mathsf{comp}\,[\![\Gamma \vdash M_1]\!] = \mathsf{comp}\,[\![\Gamma \vdash M_2]\!]$.*

3 Simple terms and pointer-free representation

Suppose $\theta = \theta_1 \to \cdots \to \theta_n \to \beta$. Then we write $\mathit{tail}(\theta) = \beta$. Given $\Gamma \vdash M : \theta$, depending on whether $\mathit{tail}(\beta)$ is *com*, *exp* or *var* respectively, we define the *sets of sequences of moves* $(\!|\Gamma \vdash M : \theta|\!)$ by the following decompositions:

$$\mathcal{L}(\mathsf{comp}\,[\![\Gamma \vdash M : \theta]\!]) = \mathit{run} \cdot (\!|\Gamma \vdash M|\!) \cdot \mathit{done}$$

$$\mathcal{L}(\mathsf{comp}\,[\![\Gamma \vdash M : \theta]\!]) = q \cdot \sum_{j=0}^{\mathit{max}} ((\!|\Gamma \vdash M|\!)_j \cdot j)$$

$$\mathcal{L}(\mathsf{comp}\,[\![\Gamma \vdash M : \theta]\!]) = \sum_{j=0}^{\mathit{max}} \mathit{write}(j) \cdot (\!|\Gamma \vdash M|\!)_j^w \cdot \mathit{ok} + \mathit{read} \cdot \sum_{j=0}^{\mathit{max}} ((\!|\Gamma \vdash M|\!)_j^r \cdot j).$$

It will turn out convenient to define automata accepting $\llparenthesis \cdots \rrparenthesis$ instead of $\mathcal{L}(\mathsf{comp}\,[\![\cdots]\!])$, because then it will not be necessary to interpret hiding in many cases (an operation under which DCFLs are not closed in general). Since $\llparenthesis \cdots \rrparenthesis$ are sets of sequences of moves, they do not always represent $\mathsf{comp}\,[\![\cdots]\!]$ faithfully because they ignore pointers. Nevertheless, we are going to identify a sufficiently rich class of terms for which $\mathsf{comp}\,[\![\cdots]\!]$ can be recovered from $\llparenthesis \cdots \rrparenthesis$.

First, we note that to establish decidability it suffices to consider β-normal terms only. This does not solve the pointer problem though. To address it we replace the application rule with its multiplicative version (left) and contraction (right):

$$\frac{\Gamma \vdash M : \theta \to \theta' \quad \Delta \vdash N : \theta}{\Gamma, \Delta \vdash MN : \theta'} \qquad \frac{\Gamma, x_1 : \theta, x_2 : \theta \vdash M : \theta'}{\Gamma, x : \theta \vdash M[x/x_1, x/x_2] : \theta'}.$$

All β-normal terms in $\mathsf{IA}_3+\mathbf{Y}_0$ are typable if we allow the above rules for $\mathsf{ord}(\theta) \leq 2$. We will call a β-normal $\mathsf{IA}_3 + \mathbf{Y}_0$ term *simple* if it can be typed by using the contraction rule only for θ such that $\mathsf{ord}(\theta) < 2$. For instance, $\lambda f.f(\lambda x.f(\lambda y.x))$ is *not* simple. Note that pointer reconstruction is uniquely defined for all moves *except* third-order questions (pointers for answers can be reconstructed uniquely thanks to the bracketing condition; first-order moves must point to the unique initial move; finally, because of Visibility, second-order questions must point to the necessarily unique first-order question in the appropriate O-view). As made precise in the lemma below, for simple terms, we can still recover $\mathsf{comp}\,[\![\cdots]\!]$ from $\llparenthesis \cdots \rrparenthesis$.

Lemma 1. *Suppose $\Gamma \vdash M : \theta$ is simple. If $sq_3 \in [\![\Gamma \vdash M : \theta]\!]$, and q_3 is a third-order question then q_3's justifier in sq_3 is the last unanswered enabler of q_3 in s.*

Proof. By induction on the structure of simple terms. The crucial point is that if $\sigma : A \multimap B$ satisfies the Lemma for $B = [\![\theta]\!]$ such that $\mathsf{ord}(\theta) \leq 1$, so does $\sigma^\dagger$. This follows from the description in Remark 1.

On the other hand, simple terms are good representatives of β-normal terms. Any β-normal term of $\mathsf{IA}_3 + \mathbf{Y}_0$ can be typed by extending a typing derivation of a simple term with a number of applications of the contraction rule for θ of order 2 followed by a number of applications of the λ-abstraction rule.

4 *G*-automata

We are going to use a variant of deterministic pushdown automata. Their states will be divided into O-states and P-states and the stack will be modified and inspected only during ϵ-moves.

- In O-states the automaton will only be able to read an O-move from the input without inspecting or changing the stack; after the transition the state will always change to a P-state.

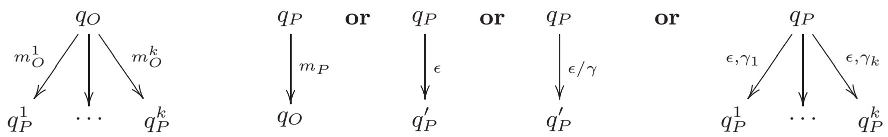

Fig. 1. Transitions in G-automata

– In P-states the automaton will either: read a *single* P-move without modifying the stack and move to an O-state, or perform an ϵ-move and go to a P-state possibly modifying the stack.

The constraints are summarized in Figure 1 and captured formally below.

Definition 1. *Let G be a game. A G-automaton $\mathcal{A}$ is a tuple $\langle Q, \Omega, i, F, \delta \rangle$ such that*

– *$Q = Q^O + Q^P$ is the set of states partitioned into Q^O and Q^P, called the sets of O-states and P-states respectively;*
– *Ω is the stack alphabet;*
– *$i \in Q^P$ is the initial state, i does not have any incoming transitions;*
– *$F \subseteq Q^P$ is the set of final states, final states do not have any outgoing transitions;*
– *$\delta : [Q^O \rightharpoonup (M_G^O \rightharpoonup Q^P)] + [Q^P \rightharpoonup \big((M_G^P \times Q^O) + Q^P + (\Omega \times Q^P) + (\Omega \rightharpoonup Q^P)\big)]$ is the transition function.*

The interpretation of a transition $\delta(q_O) \in M_G^O \to Q^P$ is that the automaton reads a letter $a \in M_G^O$ from the input and changes the state to $\delta(q_O)(a)$. Similarly for $\delta(q_P) = (b, q_O) \in M_G^P \times Q^O$, the automaton expects to see b as the input and then changes its state to q_O. On a transition $\delta(q_P) = q'_P$ the automaton changes its state but does not consume the input or change the stack. On a transition $\delta(q_P) = (\gamma, q'_P)$ the automaton changes the state and pushes γ on the stack. On a transition $\delta(q_P) \in \Omega \rightharpoonup Q^P$ the automaton makes a pop move and changes its state depending on the letter that was popped. We will use arrow notation for transitions as in Figure 1. Note that slash and comma denote push and pop operations, respectively.

We write $L(\mathcal{A})$ for the language that $\mathcal{A}$ accepts by final state and the empty stack. It is easy to see that G-automata are DPDAs. They have a particularity that they look at the stack only when doing a pop move. We make this restriction to simplify the definitions that follow. It is not difficult to see that for every deterministic pushdown automaton there is an equivalent one with this property.

By a G-automaton *configuration*, we mean a pair qw, where q is a state and $w \in \Omega^*$ is the content of the stack.

Definition 2. *We say that a G-automaton is* productive *if every non-initial reachable configuration is productive, i.e. the configuration occurs in some accepting run of the automaton.*

Lemma 2. *For every G-automaton there is an equivalent productive automaton.*

Proof. Let $\mathcal{A} = \langle Q, \Omega, i, F, \delta \rangle$ be a G-automaton. Consider the function $\beta : \Omega^* \to \mathcal{P}(Q)$ which for all contents of the stack $w \in \Omega^*$ returns the set of states q such that the configuration qw is productive. We will first modify $\mathcal{A}$ so that it keeps the current value of β in its state. Note that it is easy to update β after push-transitions, because $\beta(wa)$ depends only on $\beta(w)$ and a: $q \in \beta(wa)$ iff there exists q' such that $q' \in \beta(w)$ and when $\mathcal{A}$ starts with a on the stack in state q it can end up in q' with empty stack (states q' with this property can be precomputed at the very beginning of the construction). In order to handle pop-transitions, we simply force the automaton to push the old value of β in addition to the pushed stack symbol. Finally, it now suffices to suppress all transitions that are not consistent with β to obtain a productive automaton.

Remark 2. (i) The productiveness of the automaton will play an important role in our constructions for several reasons. One is that when $\mathcal{A}$ is productive we can be sure that when running as a subautomaton of some automaton construction (as in the proof of Theorem 2), it will only use the symbols it pushed (it cannot try to make a pop operation on the empty stack as that would be an unproductive configuration). Another consequence of productiveness is that the automaton cannot enter an accepting state while the stack is not empty. Indeed, since there are no outgoing transitions from an accepting state, the resulting configuration would not be productive. Further, as the automaton stack is necessarily empty whenever it reaches a final state, we also know that when it finishes the stack will be exactly as before it has started.

(ii) From an O-state there are in principle several input letters from M_G^O that can make the automaton advance. However, because of productiveness, in general, not every playable O-move can label an outgoing transition from the O-state. From a P-state there is only one input letter from M_G^P that the automaton is prepared to read. We have this asymmetry because G-automata are designed to accept strategies. Thus, an automaton works in cycles: it reads a letter from M_G^P, then a letter from M_G^O and then does some internal manipulations on the stack.

Definition 3. *We say that an automaton is* careful *if whenever it reaches a configuration qw after reading the sequence of moves s then the sequence of open second-order questions in s appears in the stack w. More formally, there is a function $\pi_{\mathcal{A}} : \Omega \to M_G \cup \{\epsilon\}$ such that $\pi^*_{\mathcal{A}}(w)$ is the sequence of open second-order questions in s.*

The final definition of this section makes it precise what kind of automata we want to construct. Below, by $\mathcal{A}(F')$ we will denote the automaton $\mathcal{A}$ with F' as the set of accepting states.

Definition 4. *We say that a tuple of G-automata $\langle \mathcal{A}^1, \cdots, \mathcal{A}^n \rangle$, where each $\mathcal{A}^j = \langle Q^j, \Omega^j, i^j, F^j, \delta^j \rangle$, is* fully productive *for $\Gamma \vdash M : \theta$ just if each $\mathcal{A}^j$ is productive and careful; further*

- *suppose* $tail(\theta) = com$*: we have* $n = 1$ *and* $L(\mathcal{A}^1) = (\!|\Gamma \vdash M : \theta|\!)$
- *suppose* $tail(\theta) = exp$*: we have* $n = 1$, $F^1 = \oplus_{j=0}^{max} F_j$ *and* $L(\mathcal{A}^1(F_j)) = (\!|\Gamma \vdash M : \theta|\!)_j$ *for any* $0 \leq j \leq max$*;*
- *in case* $tail(\theta) = var$*: we have* $n = max + 2$, $F^1 = \oplus_{j=0}^{max} F_j$ *and* $L(\mathcal{A}^1(F_j)) = (\!|\Gamma \vdash M : \theta|\!)_j^r$ *for any* $0 \leq j \leq max$*; for each* $0 \leq k \leq max$*, we have* $L(\mathcal{A}^{k+2}) = (\!|\Gamma \vdash M : \theta|\!)_k^w$*.*

5 Modelling simple terms

Here we focus on simple terms. The extension to other β-normal terms is discussed in Section 6.

Theorem 2. *For any simple* $\mathsf{IA}_3 + \mathbf{Y}_0$ *term* $\Gamma \vdash M : \theta$ *there exists a fully productive tuple of* $[\![\Gamma \vdash \theta]\!]$*-automata for* $\Gamma \vdash M : \theta$*.*

Proof. We use structural induction. Whenever our constructions fail to preserve productiveness, we simply appeal to Lemma 2 to obtain an equivalent productive automaton.

Thanks to the equalities below and the fact that productive G-automata compose well (see Remark 2 (i))

$$(\!|\Gamma \vdash \lambda x.M|\!) = (\!|\Gamma, x \vdash M|\!) \qquad (\!|\Gamma \vdash M; N|\!) = (\!|\Gamma \vdash M|\!) \cdot (\!|\Gamma \vdash N|\!)$$
$$(\!|\Gamma \vdash !M|\!)_j = (\!|\Gamma \vdash M|\!)_j^r \qquad (\!|\Gamma \vdash M := N|\!) = \textstyle\sum_{j=0}^{max} ((\!|\Gamma \vdash N|\!)_j \cdot (\!|\Gamma \vdash M|\!)_j^w)$$
$$(\!|\Gamma \vdash \mathbf{if}\ M\ \mathbf{then}\ N_1\ \mathbf{else}\ N_2|\!) = (\!|\Gamma \vdash M|\!)_0 \cdot (\!|\Gamma \vdash N_2|\!) + (\textstyle\sum_{j=1}^{max} (\!|\Gamma \vdash M|\!)_j) \cdot (\!|\Gamma \vdash N_1|\!)$$

the corresponding cases are easy. We can simply appeal to the inductive hypothesis and construct the new automata by connecting suitable final states with suitable initial states with ϵ-transitions. Note that this does not violate determinism as final states do not have outgoing transitions. Equivalently, one could "glue" final states with initial ones suitably. If one performs the constructions for reachable final states only, productiveness will be preserved. The remaining cases are treated as follows:

The case of $\Gamma \vdash \mathbf{new}\ X\ \mathbf{in}\ M : \beta$*.* Suppose $\beta = com$ and $\langle\, \mathcal{A}\,\rangle$ is fully productive for $\Gamma, X : var \vdash M : \beta$. Let us construct $max + 1$ copies of $\mathcal{A}$ denoted by $\mathcal{A}^0, \cdots, \mathcal{A}^{max}$. We will use superscripts to refer to their states. The unique automaton $\mathcal{B}$ in the fully productive tuple for $\Gamma \vdash \mathbf{new}\ X\ \mathbf{in}\ M$ will consist of all these copies, the idea being that the index of the copy corresponds to the interim value stored in the variable X. The new initial state will be i^0, i.e. the initial state of $\mathcal{A}^0$. The set of final states will be $H = \bigcup_{h=0}^{max} F^h$, i.e. the set of final states from all the copies $\mathcal{A}^h$ of $\mathcal{A}$. We make the following changes to the copies of $\mathcal{A}$. Because $\mathcal{A}$ is productive, we can assume that each $\mathcal{A}$-transition using the P-moves $read_X$ or $write(k)_X$ $(0 \leq k \leq max)$ originating from the distinguished copy of var is followed by transitions on $i_X^1, \cdots, i_X^d$ and ok_X respectively, where

$\{ i^1, \cdots, i^d \} \subseteq \{ 0, \cdots, max \}$. To construct $\mathcal{B}$ we redirect transitions in the various copies of $\mathcal{A}$ as shown below

$$\begin{tikzcd}
 & p^h \arrow[d, "read_X"] & & & p^h \arrow[dd, "\epsilon"] \\
 & o^h \arrow[dl, "i^1_X"'] \arrow[dr, "i^d_X"] & & \longmapsto & \\
p^h_{i^1} & \cdots & p^h_{i^d} & & p^h_h
\end{tikzcd}
\qquad
\begin{tikzcd}
p^h \arrow[d, "write(k)_X"] & & p^h \arrow[dd, "\epsilon"] \\
o^h \arrow[d, "ok_X"] & \longmapsto & \\
p^h_0 & & p^k_0
\end{tikzcd}$$

where $0 \leq h \leq max$. These transformations redirect only the transition from p^h, while the transitions from o^h remain as they were. In the first case the transformation is performed only if $h = i^c$ for some $1 \leq c \leq d$. Otherwise, the transition from p^h is just erased. Note that in the second case the new transition connects $\mathcal{A}^h$ to $\mathcal{A}^k$. After all the transitions on P-moves are dealt with, we delete the transitions from o^h. It is clear that the $\mathcal{B}$ is careful if $\mathcal{A}$ is. Using Lemma 2 we can make it productive. If $\beta = exp$ the same construction can be performed to give us a fully productive tuple for $\Gamma \vdash \mathbf{new}\ X\ \mathbf{in}\ M$. To define the required partition of states $H = \oplus_{j=0}^{max} H_j$ we simply take $H_j = \bigcup_{h=0}^{max} F_j^h$, where $F = \oplus_{j=0}^{max} F_j$ is the partition given by the inductive hypothesis. The case of $\beta = var$ combines the previous two cases.

The case of $\Gamma \vdash \mu x^\beta.M : \beta$. Suppose $\beta = com$ and $\langle \mathcal{A} \rangle$ is a fully productive tuple for $\Gamma, x : \beta \vdash M : \beta$, where $\mathcal{A} = \langle Q, \Omega, i, F, \delta \rangle$. The unique automaton $\mathcal{B}$ in the fully productive tuple for $\Gamma \vdash \mu x.M$ will be constructed from two copies $\mathcal{A}^0, \mathcal{A}^1$ of $\mathcal{A}$. Intuitively, $\mathcal{A}^1$ will be used to process recursive calls from M, whereas $\mathcal{A}^0$ will correspond to the base copy of M. Accordingly, the initial state will be i^0 and the final states will be those from F^0. The stack alphabet of $\mathcal{B}$ will, in addition to the stack alphabet of $\mathcal{A}$, contain two copies of the set of O-states of $\mathcal{A}$. By productiveness of $\mathcal{A}$, we can assume that each transition on run_x is followed by a transition on $done_x$. In order to define $\mathcal{B}$, for each $h = 0, 1$ and for each block of the shape $p^h \xrightarrow{run_x} o^h \xrightarrow{done_x} p^h_{done}$ we erase the transition from p^h and add the transitions: $p^h \xrightarrow{\epsilon/o^h} i^1$ and $f^1 \xrightarrow{\epsilon, o^h} p^h_{done}$ for any $f \in F$. The case of $\beta = exp$ is similar but, additionally, we have to pass on the result of the recursive call. The notion of a productive tuple will make that easy. Suppose $F = \oplus_{j=0}^{max} F_j$. Then we first create two copies $\mathcal{A}^0, \mathcal{A}^1$ of $\mathcal{A}$ and for $h = 0, 1$ and for each block

$$\begin{tikzcd}
 & p^h \arrow[d, "q_x"] & \\
 & o^h \arrow[dl, "i^1_x"'] \arrow[d] \arrow[dr, "i^d_x"] & \\
p^h_{i^1} & \cdots & p^h_{i^d}
\end{tikzcd}$$

we erase the transition from p^h and add the transitions: $p^h \xrightarrow{\epsilon/o^h} i^1$ and $f^1_{i^c} \xrightarrow{\epsilon, o^h} p^h_{i^c}$ for any $1 \leq c \leq d$ and $f_{i^c} \in F_{i^c}$. The initial state of the new automaton is i^0, the final states are those in $F^0 = \oplus_{j=0}^{max} F_j^0$. The remaining case of $\beta = var$ is more tedious but completely analogous to the previous two. Given a fully productive tuple $\langle \mathcal{A}^1, \cdots, \mathcal{A}^{max+2} \rangle$ for $\Gamma, x \vdash M$, blocks $write(j)_x ok_x$ are replaced with calls to a copy of $\mathcal{A}^{j+2}$ as for com, while blocks $read_x j_x$ are replaced by calls to a copy of $\mathcal{A}^1$. It is easy to see that the obtained automaton is careful. After the above constructions we can apply Lemma 2 to make the resulting automaton productive.

The case of $f M_1 \cdots M_k$. For demonstration we assume that $k = 1, M_1 = M$ and $fM : com$ and f does not occur in M, i.e. $f : \theta \to com, \Gamma \vdash fM : com$. Let us write r_f, d_f and r, d for the $[\![com]\!]$-moves on the left and right respectively. Observe that because fM is interpreted by interaction of the identity strategy (corresponding to f) with $([\![\Gamma \vdash M]\!])^\dagger$ we have $\mathsf{comp}\,[\![f : \theta \to com, \Gamma \vdash fM : com]\!] = r\ r_f\ (\{\,\epsilon\,\} \cup L)\ d_f\ d$, where $L = \phi(\mathsf{comp}\,[\![\Gamma \vdash M : \theta]\!]^\dagger)$ and ϕ is the (injective) renaming map which acts like identity on moves from $[\![\Gamma]\!]$ and maps $[\![\theta]\!]$-moves to their copies in $[\![\theta \to com]\!]$. Below we examine two representative examples in detail and give the precise shape of ϕ. We use subscripts to refer to (moves from) various copies of com.

- Suppose $\Gamma \vdash M : com_0$ and $f : com_{f,1} \to com_f, \Gamma \vdash fM : com$. Then $L = \mathsf{comp}\,[\![\Gamma \vdash M]\!]^\dagger[r_0 \mapsto r_{f,1}, d_0 \mapsto d_{f,1}]$.
- Suppose $\Gamma \vdash M : com_1 \to com_0$ and $f : (com_{f,2} \to com_{f,1}) \to com_f, \Gamma \vdash fM : com$. Then $L = \mathsf{comp}\,[\![\Gamma \vdash M]\!]^\dagger[r_0 \mapsto r_{f,1}, d_0 \mapsto d_{f,1}, r_1 \mapsto r_{f,2}, d_1 \mapsto d_{f,2}]$.

Consequently, the main difficulty is the construction of an automaton accepting $\mathsf{comp}\,(\sigma^\dagger) \cup \{\epsilon\}$, where $\sigma = [\![\Gamma \vdash M : \theta]\!]$ and $\mathsf{ord}(\theta) \leq 1$. What is constructed in the following is not a $[\![\Gamma \vdash \theta]\!]$-automaton (the initial state, which is an O-state, will coincide with the final one and it will have both incoming and outgoing transitions), but the crucial fact is that it becomes one when the transitions on r_f and d_f (which will become the new initial and final moves respectively) are added. We consider the case of $\theta = com_1 \to com_0$ for illustration.

Suppose $\langle \mathcal{A} \rangle$ is fully productive for $\Gamma \vdash M : \theta$, $\mathcal{A} = \langle Q, \Omega, i, F, \delta \rangle$ and $\mathsf{ord}(\theta) \leq 1$. Recall from Remark 1 that in this case $\dagger$ iterates as well as stacks copies of the original strategy and a new copy can start either if the previous one finished or after r_1. To model that, we create two copies $\mathcal{A}^0, \mathcal{A}^1$ of $\mathcal{A}$ with the aim of delegating the copies started after r_1 to $\mathcal{A}^1$. The outermost iterated copies will be processed by $\mathcal{A}^0$.

First we define a few new states: i^0_{new} (which will be the initial and final state of the automaton) and $(\circ, q), (\bullet, q)$, where $q \in Q^0 + Q^1 + \{\, i^0_{\text{new}} \,\}$. We make the following additions to $\mathcal{A}^0, \mathcal{A}^1$:

- $i^0_{\text{new}} \xrightarrow{r_0} (\bullet, i^0_{\text{new}}) \xrightarrow{\epsilon/(i^0_{\text{new}}, r_0)} i^0$ and $f^0 \xrightarrow{\epsilon,(i^0_{\text{new}}, r_0)} (\circ, i^0_{\text{new}}) \xrightarrow{d_0} i^0_{\text{new}}$ for any $f \in F$.
- Observe that r_1 is always followed by d_1 in σ. By productiveness of $\mathcal{A}$, we can assume that each transition $p^h \xrightarrow{r_1} o^h$ $(h = 0, 1)$ is followed by $o^h \xrightarrow{d_1} p^h_0$.

Then we add $o^h \xrightarrow{r_0} (\bullet, o^h) \xrightarrow{\epsilon/(o^h, r_0)} i^1$. This makes it possible for the automaton to start processing a new copy of σ, the return state is saved on the stack. To process call returns, we add $f^1 \xrightarrow{\epsilon,(o^h,r_0)} (\circ, o^h) \xrightarrow{d_0} o^h$ for any $f \in F$.

Note that in addition to return addresses in the form of O-states we have also arranged for the questions r_0 to be pushed on the stack. They will remain there as long as they are not answered in the corresponding position. These questions are redundant for accepting $\mathsf{comp}\,(\sigma^\dagger) \cup \{\epsilon\}$, but are necessary for the resulting automaton to be careful. Since r_0 will be substituted by $r_{f,1}$, the questions r_0 are exactly the second-order questions contributed by f. The case of $f M_1 \cdots M_k$ is analogous, except that we need to apply $\dagger$ to $\sum_{i=1}^{k} [\![\Gamma \vdash M_i]\!]$.

The contraction rule. The contraction rule for free identifiers of type θ, where $\mathsf{ord}(\theta) = 0, 1$, is interpreted simply by identifying moves from the two copies of $[\![\theta]\!]$. In general this might lead to nondeterminism, but thanks to the structure of our automata this can never happen in our case. There is no problem with P-moves as from each P-state the automaton can read at most one letter. Suppose that a nondeterminism arises from some O-state reachable from the initial configuration. This means that there exists a position s such that the automaton can read both $s\,o_1$ and $s\,o_2$, where o_1 and o_2 are the O-moves from two different copies of θ. However, this contradicts that fact that at most one of $s\,o_1$ and $s\,o_2$ can satisfy Visibility (because only one first-order question is O-visible in s). Consequently, contraction can be interpreted without loss of determinacy.

6 Representing pointers

We are going to introduce a representation of pointers for simple terms by following Lemma 1. Consider a position sq_3, where q_3 is a third-order question. We define $\pi(s, q_3) = k - j + 2$ where q_2^j is the last enabler of q_3 in the sequence $q_2^1, \cdots, q_2^k$ of all second-order unanswered questions in s (written in the order they appear in s).

Definition 5. *Suppose $\sigma = [\![\Gamma \vdash M : \theta]\!]$, where $\Gamma \vdash M : \theta$ is an $\mathsf{IA}_3 + \mathbf{Y}_0$ term. The language $\mathcal{P}(\sigma)$ over the alphabet $M_{[\![\Gamma\vdash\theta]\!]} + \{\, check \,\}$ is defined by $\mathcal{P}(\sigma) = \{\, s\, check^{\pi(s,q_3)} \mid sq_3 s' \in \mathsf{comp}\,\sigma \text{ for some } s' \,\}$.*

Lemma 3. *For any simple term $\Gamma \vdash M : \theta$ there exists a DPDA accepting $\mathcal{L}(\mathsf{comp}\,\sigma) \cup \mathcal{P}(\sigma)$, where $\sigma = [\![\Gamma \vdash M : \theta]\!]$.*

Proof. By Theorem 2 there is a fully productive tuple for $\Gamma \vdash M : \theta$. By adding transitions on the missing initial and final moves, we can construct an automaton $\mathcal{A}$ accepting $\mathcal{L}(\mathsf{comp}\,\sigma)$. Though $\mathcal{A}$ is not, strictly speaking, a G-automaton, it is nonetheless a DPDA that is productive and careful. (The definitions of productive and careful do carry over in this case.) Next, we will construct a

DPDA accepting $\mathcal{L}(\mathsf{comp}\,\sigma) \cup \mathcal{P}(\sigma)$. Suppose that the automaton $\mathcal{A}$ reads the sequence sq_3 where q_3 is a third-order question. Note that q_3 is always a P-move, so it is uniquely determined by s and σ. As $\mathcal{A}$ is productive, we know that sq_3 can be extended to a complete position. It remains to take care of *check* letters. As $\mathcal{A}$ is careful, all second-order questions that are open in sq_3 will be stored on the stack. Hence, in order to accept the right number of *check*'s after reading s, the automaton can move to a fresh state (reading the letter *check*) and enter a new mode in which it will repeatedly pop the stack reading *check* whenever the topmost stack symbol contains a second-order question and ϵ otherwise. It accepts exactly after it encounters the first enabler of q_3.

Now that we can represent the semantics of simple terms with pointers, the semantics of other β-normal terms can be obtained by renaming, because in order to type such terms we only need to perform contraction at order 2 and λ-abstraction. The former is done by relabelling (in the same manner as at order 0 or 1; the previous argument that determinism is preserved remains valid), the latter amounts to identity in the games setting. Consequently, we get:

Theorem 3. *For any* $\mathsf{IA}_3 + \mathbf{Y}_0$ *term* $\Gamma \vdash M : \theta$ *in* β*-normal form there exists a DPDA accepting* $\mathcal{L}(\mathsf{comp}\,\sigma) \cup \mathcal{P}(\sigma)$*, where* $\sigma = [\![\Gamma \vdash M : \theta]\!]$*. Hence, the theorem also holds for any* $\mathsf{IA}_3 + \mathbf{Y}_0$ *term.*

By Lemma 1, $\mathcal{L}(\mathsf{comp}\,\sigma) \cup \mathcal{P}(\sigma)$ is a faithful representation of $\mathsf{comp}\,\sigma$. Thus, by Theorem 3 and the decidability of DPDA equivalence [8], we get

Theorem 4. *Observational equivalence is decidable for terms in* $\mathsf{IA}_3 + \mathbf{Y}_0$*.*

7 Hardness

A DPDA [6] is a tuple $\mathcal{B} = \langle\, Q, \Sigma, \Gamma, \delta, q_0, Z_0 \,\rangle$, where $\delta : Q \times (\Sigma \cup \{\, \epsilon \,\}) \times \Gamma \rightharpoonup Q \times \Gamma^*$. Additionally, whenever $\delta(q, a, X)$ is defined for some $a \in \Sigma$, $\delta(q, \epsilon, X)$ must be undefined. We consider acceptance by empty stack (initially the stack contains Z_0) and write $N(\mathcal{B})$ for the language accepted by $\mathcal{B}$. For simplicity, we further assume that $\mathcal{B}$ can either pop the stack or push one symbol onto it, i.e. if $\delta(q, a, X) = (q', \alpha)$ then $\alpha = \epsilon$ or $\alpha = \alpha_0\alpha_1$ $(\alpha_0, \alpha_1 \in \Gamma)$ and $\alpha_0 = X$. Any DPDA can be easily converted into this form.

We identify values of type *exp* with Σ. Consider the game $G = [\![exp]\!] \Rightarrow [\![com]\!]$ so that we have $M_G = \{\, q \,\} \cup \Sigma \cup \{\, r, d \,\}$. Given $L \subseteq \Sigma^*$ define $\widehat{L} \subseteq M_G^*$ by $\widehat{L} = \{\, rqx_1 \cdots qx_n d \mid x_1 \cdots x_n \in L \,\}$. Note that $\widehat{L_1} = \widehat{L_2}$ iff $L_1 = L_2$.

Lemma 4. *For any DPDA* $\mathcal{B}$ *there exists a term* $x : exp \vdash M_{\mathcal{B}} : com$ *such that* $\mathcal{L}(\mathsf{comp}\,[\![x : exp \vdash M_{\mathcal{B}} : com]\!]) = \widehat{N(\mathcal{B})}$.

Proof. Push transitions are simulated by recursive calls, pop moves by call returns. Before each call, the symbol to be pushed is stored in the variable *TOP* which is used to initialize the local copy of X after the call. Take $M_{\mathcal{B}}$ to be

$$
\begin{array}{l}
x : exp \vdash \mathbf{new}\, Q := q_0,\ TOP := Z_0,\ CH\ \mathbf{in} \\
\qquad \mu z^{com}.\ \mathbf{new}\, POP := 0,\ X := !TOP\ \mathbf{in} \\
\qquad\qquad \mathbf{while}\ (\mathbf{not}\ !POP)\ \mathbf{do} \\
\qquad\qquad\quad (\mathbf{if}\ \delta(!Q, \epsilon, !X) = (q', \alpha)\ \mathbf{then} \\
\qquad\qquad\qquad (Q := q'; \\
\qquad\qquad\qquad\ \mathbf{if}\ \alpha = \epsilon\ \mathbf{then}\ POP := 1\ \mathbf{else}\ ((TOP := \alpha_1); z)) \\
\qquad\qquad\quad \mathbf{else} \\
\qquad\qquad\qquad (CH := x; \\
\qquad\qquad\qquad\ \mathbf{if}\ \delta(!Q, !CH, !X) = (q', \alpha)\ \mathbf{then} \\
\qquad\qquad\qquad\quad (Q := q'; \\
\qquad\qquad\qquad\quad\ \mathbf{if}\ \alpha = \epsilon\ \mathbf{then}\ POP := 1\ \mathbf{else}\ ((TOP := \alpha_1); z)) \\
\qquad\qquad\qquad\ \mathbf{else}\ \Omega_{com})) : com
\end{array}
$$

Proposition 1. *For any DPDAs $\mathcal{B}_1, \mathcal{B}_2$ we have $N(\mathcal{B}_1) \subseteq N(\mathcal{B}_2)$ iff $M_{\mathcal{B}_1} \sqsubseteq\!\!\!\sim M_{\mathcal{B}_2}$.*

Observe that the term used in Lemma 4 is from $\mathsf{IA}_1 + \mathbf{Y}_0$. Since the Containment Problem for DPDAs is undecidable [6] we have:

Corollary 1. *Observational approximation is undecidable for $\mathsf{IA}_1 + \mathbf{Y}_0$ terms; observational equivalence is at least as hard as DPDA Equivalence.*

Our results complete the classification of decidable and undecidable fragments of IA. The exact complexity of observational equivalence for β-normal terms is known in most cases, as shown in the table in Section 1. The complexity of $\mathsf{IA}_i + \mathbf{Y}_0$ depends on the complexity of DPDA Equivalence, which is not known at present. Our construction, when suitably optimized, yields DPDAs that are doubly exponentially larger than the given term in β-normal form. It is also not yet understood how the presence of β-redexes affects the complexity.

References

1. Ong, C.-H. L.: Observational equivalence of 3rd-order Idealized Algol is decidable. In: Proceedings of LICS (2002) 245–256
2. Murawski, A. S.: On program equivalence in languages with ground-type references. In: Proceedings of LICS (2003) 108–117
3. Murawski, A. S.: Games for complexity of second-order call-by-name programs. Theoretical Computer Science, to appear.
4. Murawski, A. S., Walukiewicz, I.: Third-order Idealized Algol with iteration is decidable. In: Proceedings of FOSSACS. LNCS **3441** (2005) 202–218
5. Stirling, C.: Deciding DPDA equivalence is primitive recursive. In: Proceedings of ICALP. LNCS **2380** (2002) 821–832
6. Hopcroft, J. E., Ullman, J. D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
7. Abramsky, S., McCusker, G.: Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In O'Hearn, P.W., Tennent, R.D., eds.: Algol-like languages, Birkhaüser (1997) 297–329
8. Sénizergues, G.: L(A)=L(B)? decidability results from complete formal systems. Theoretical Computer Science **251(1-2)** (2001) 1–166