A Fragment of ML Decidable by Visibly Pushdown Automata

David Hopkins[1,⋆], Andrzej S. Murawski[2,⋆⋆], and C.-H. Luke Ong[1]

[1] Department of Computer Science, University of Oxford, UK
[2] Department of Computer Science, University of Leicester, UK

Abstract. The simply-typed, call-by-value language, RML, may be viewed as a canonical restriction of Standard ML to ground-type references, augmented by a "bad variable" construct in the sense of Reynolds. By a *short* type, we mean a type of order at most 2 and arity at most 1. We consider the *O-strict* fragment of (finitary) RML, $\mathsf{RML}_{\text{O-Str}}$, consisting of terms-in-context $x_1 : \theta_1, \cdots, x_n : \theta_n \vdash M : \theta$ such that θ is short, and every argument type of every θ_i is short. $\mathsf{RML}_{\text{O-Str}}$ is surprisingly expressive; it includes several instances of (in)equivalence in the literature that are challenging to prove using methods based on (state-based) logical relations. We show that it is decidable whether a given pair of $\mathsf{RML}_{\text{O-Str}}$ terms-in-context is observationally equivalent. Using the fully abstract game semantics of RML, our algorithm reduces the problem to the language equivalence of visibly pushdown automata. When restricted to terms in canonical form, the problem is EXPTIME-complete.

1 Introduction

The standard approaches to the verification of higher-order programs are *type-based program analysis* on the one hand, and *theorem-proving* and *dependent types* on the other. The former is sound, but often imprecise; the latter typically requires human intervention. The paper is concerned with a relatively recent, game-semantics based approach to the verification of higher-order procedural programs. We consider a call-by-value language, RML, which has both functional and (stateful) imperative features, mediated by Church's simple type theory. RML may be viewed as a canonical restriction of Standard ML to ground-type references, except that it includes a "bad variable" construct [17,2].

Observational equivalence is a compelling notion of program equivalence. Two terms M and N are observationally equivalent, written $M \cong N$, if they are mutually replaceable in every program without changing the computational outcome. Because of the quantification over all program contexts, the theory of observational equivalence is rich and hard to reason about, as illustrated by the following example.

Example 1. (i) $\mathsf{let}\, c = \mathsf{ref}\, \mathsf{in}\, \lambda f^{\mathsf{unit}\to\mathsf{unit}}.(c := 1; f(); !c) \;\cong\; \lambda f^{\mathsf{unit}\to\mathsf{unit}}.(f(); 1)$
(ii) $\mathsf{let}\, c = \mathsf{ref}\, \mathsf{in}\, \lambda f^{\mathsf{unit}\to\mathsf{unit}}.(c := 0; f(); c := 1; f(); !c) \;\cong\; \lambda f^{\mathsf{unit}\to\mathsf{unit}}.(f(); f(); 1)$

⋆ Supported by Microsoft Research through its PhD Scholarship Programme.
⋆⋆ Supported by an EPSRC Advanced Research Fellowship (EP/C539753/1).

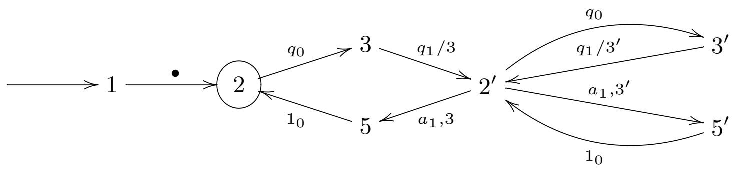
(iii) $\mathsf{let}\, a = \mathsf{ref}\,\mathsf{in}\,\mathsf{let}\, r = \mathsf{ref}\,\mathsf{in}\, \lambda f.(r := !r + 1; a := f(!r); r := !r - 1; !a) \ \not\cong\ \lambda f.f(1)$
The two equivalences above, due to Pitts and Stark [16] and Thamsborg [3] respectively, are notoriously tricky to verify using methods based on (state-based) logical relations. The inequivalence, a somewhat surprising instance due to Stark [19], requires a rather delicate separating context to exhibit.

Let θ_i and θ range over RML types. We say that *observational equivalence is decidable at a type sequent* $\theta_1, \cdots, \theta_n \vdash \theta$ (or simply, $\overline{\theta} \vdash \theta$ is *decidable*) just if the following problem is decidable: given terms-in-context $x_1 : \theta_1, \cdots, x_n : \theta_n \vdash M, N : \theta$ of finitary RML (henceforth, written $\mathsf{RML_f}$), are they observationally equivalent? This paper is concerned with the question of classifying the decidable type sequents of $\mathsf{RML_f}$.

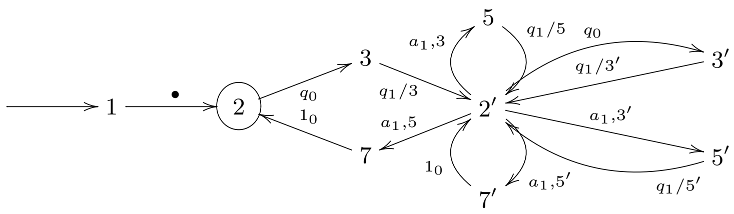
Following Ghica and McCusker [6], we use a method based on game semantics to decide observational equivalence of $\mathsf{RML_f}$. Take a term-in-context $\Gamma \vdash M : \theta$ with $\Gamma = x_1 : \theta_1, \cdots, x_n : \theta_n$. In game semantics [9,7], the term-in-context is interpreted as a P strategy $[\![\Gamma \vdash M : \theta]\!]$ for playing (against O, who takes the environment's perspective) in the prearena $[\![\overline{\theta} \vdash \theta]\!]$. A play between P and O is a sequence of moves in which each non-initial move has a justification pointer to some earlier move. Thanks to the fully abstract game semantics of RML, observational equivalence is characterised by *complete plays* i.e. $M \cong N$ iff the P-strategies, $[\![\Gamma \vdash M]\!]$ and $[\![\Gamma \vdash N]\!]$, trace out the same set of complete plays. (A play is complete if every question in it is subsequently answered in the play.) Strategies may be viewed as highly constrained processes, and are amenable to concrete, automata-theoretic representations. In certain prearenas—which we shall call *bi-strict*—plays may be represented simply by their underlying move sequence, because the justification pointers from both O- and P-moves are uniquely determined. Murawski studied bi-strict sequents in [11] and identified those that are decidable by reduction to the equivalence problem of deterministic FSA.

In this paper, we consider type sequents of $\mathsf{RML_f}$ that are *O-strict*. Plays over prearenas denoted by O-strict sequents enjoy the property that pointers from O-moves are uniquely determined by the underlying move sequence. We first give a simple characterisation of O-strict type sequents: $\theta_1, \cdots, \theta_n \vdash \theta$ is O-strict iff θ is short, and every argument type of every θ_i is short, where a type is said to be *short* if it has order at most 2 and arity at most 1. (Henceforth we write the O-strict fragment of $\mathsf{RML_f}$ as $\mathsf{RML_{O\text{-}Str}}$.) We then prove our first result: observational equivalence of $\mathsf{RML_{O\text{-}Str}}$ is decidable by reduction to the equivalence problem of visibly pushdown automata [4]. Our proof is by induction over the canonical forms of $\mathsf{RML_{O\text{-}Str}}$ terms-in-contexts. For each such term-in-context $\Gamma \vdash M$, we construct a visibly pushdown automaton $\mathcal{A}_{\Gamma \vdash M}$ that accepts the complete plays in the strategy denotation of the term, whereby P-questions are pushes, O-answers pops and all other moves no-ops. The key innovation of the construction lies in the encoding of the pointers from P-questions. Instead of trying to represent all such pointers present, we concentrate only on *representing a single pointer*. Note that for every pointer we need to represent, there must be an accepting run encoding its location. So even though each individual word accepted by the automaton may not have enough information to fully reconstruct the pointers, when we consider the full language we will be able to uniquely place all justification pointers. Our second result is that the observational equivalence of $\mathsf{RML_{O\text{-}Str}}$ terms-in-context in canonical form is EXPTIME-complete. EXPTIME-hardness is shown by a reduction of the equivalence problem for nondeterministic automata on binary trees [18] to the problem of deciding observational equivalence at the sequent $\mathsf{unit} \to \mathsf{int} \vdash (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}$.

Example 2. Our algorithm can decide the (in)equivalence[1] instances in Example 1 as the terms belong to $\mathsf{RML}_{\text{O-Str}}$. (i) The VPA below represents the complete plays of the game semantics of the terms of Example 1(i). In the diagram, the edge label 'q/n' means 'on reading q, push n', and 'a, n' means 'on reading a and stack top is n, pop'.



(ii) This VPA represents the complete plays of the game semantic strategy for the terms of Example 1(ii).



Our results may be viewed as the first steps towards a complete classification of the decidable $\mathsf{RML_f}$ type sequents. In the case of (finitary) Idealized Algol, decidability (and if so, the complexity) of a given type sequent depends only on its type-theoretic order [12]. In contrast, the decidability of $\mathsf{RML_f}$-sequents is not so neatly characterised by order (see the table below): there are undecidable sequents of order as low as 2 [11], amidst interesting classes of decidable sequents at each of orders 1 to 4. For comparison, we also give DFA-decidable (regular) and undecidable sequents [10,11].

$\mathsf{RML_f}$-Fragment	Examples of Type Sequents (writing o for unit)
bi-strict, regular	$(o \to o) \to o \vdash o \to o$
bi-strict, not-reg.	$\vdash (o \to o) \to o$
$\mathsf{RML}_{\text{O-Str}}$	$((o \to \ldots \to o) \to o) \to o \vdash (o \to \ldots \to o) \to o$
undecidable	$\vdash (o \to o) \to (o \to o) \to o, \quad (((o \to o) \to o) \to o) \to o \vdash o$

The remaining open cases are those in which pointers from O-moves need to be represented explicitly. At the moment we see no way of dealing with them, as they seem to require potentially unbounded references to past computations. What we can say is that our method of single-pointer representation cannot be extended beyond the O-strict fragment of $\mathsf{RML_f}$ (as there are distinct strategies that have the same single-pointer representation).

[1] Restricted to sequents in which $\mathsf{int\,ref}$ does *not* occur, observational equivalence of RML conservatively extends that of Reduced ML (because mkvar is only needed at $\mathsf{int\,ref}$ for definability).

2 RML, Game Semantics and Visibly Pushdown Automata

RML is a call-by-value functional language with state [1]. It is similar to Reduced ML [16], the canonical restriction of Standard ML to ground-type references, except that it includes a "bad-variable" constructor (in the absence of the "bad-variable" constructor the equality test is definable). Types are generated by the grammar $\theta ::= \mathsf{unit} \mid \mathsf{int} \mid \mathsf{int\,ref} \mid \theta \rightarrow \theta$. The type assignment rules are completely standard. The operational semantics, which is defined as a "big-step" relation [11], is also standard. For closed terms, we write $M\Downarrow$ just if M reduces to some value. This can be used to define a natural notion of equivalence; intuitively, two terms are observationally equivalent if one can always replace the other without affecting the result of the computation. Given two terms-in-context $\Gamma \vdash M_1, M_2 : \theta$, we say that M_1 *observationally approximates* M_2 (written $\Gamma \vdash M_1 \precapprox M_2$) if for all contexts $C[-]$ such that $C[M_1]$ and $C[M_2]$ are closed terms of type unit, we have that if $C[M_1]\Downarrow$ then $C[M_2]\Downarrow$. We say M_1 and M_2 are *observationally equivalent* (written $\Gamma \vdash M_1 \cong M_2$) if $\Gamma \vdash M_1 \precapprox M_2$ and $\Gamma \vdash M_2 \precapprox M_1$. It can be shown that every RML term is effectively convertible to an equivalent term in *canonical form*, defined by the following grammar ($\beta \in \{\mathsf{unit}, \mathsf{int}\}$).

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

In order to achieve a decidability result, we consider the *finitary* fragment of RML, written $\mathsf{RML_f}$. That is, we restrict the type int to be a finite subset of $\mathbb{Z}$.

Call-By-Value Game Semantics We present call-by-value game semantics in the style of Honda and Yoshida [7], as opposed to Abramsky and McCusker's isomorphic model [1], as Honda and Yoshida's constructions are more concrete, lead to more compact alphabets, and so are more suited to algorithmic analysis.

An *arena* A is a triple $(M_A, \vdash_A, \lambda_A)$ where M_A is a set of *moves* where $I_A \subseteq M_A$ consists of *initial* moves, $\vdash_A \subseteq M_A \times (M_A \backslash I_A)$ is called the *justification relation*, and $\lambda_A : M_A \rightarrow \{O, P\} \times \{Q, A\}$ a labelling function such that for all $i_A \in I_A$ we have $\lambda_A(i_A) = (P, A)$ and if $m \vdash_A m'$ then $(\pi_1 \lambda_A)(m) \neq (\pi_1 \lambda_A)(m')$ and $(\pi_2 \lambda_A)(m') = A \Rightarrow (\pi_2 \lambda_A)(m) = Q$.

The function λ_A labels moves as belonging to either *Opponent* or *Proponent* and as being either a *Question* or an *Answer*. Note that answers are always justified by questions, but questions can be justified by either a question or an answer. We will use arenas to model types. However, the actual games will be played over *prearenas*, which are defined in the same way except that initial moves are O-questions.

Three basic arenas are 0, the empty arena, 1, the arena containing a single initial move $\bullet$, and $\mathbb{Z}$, which has the integers as its set of moves, all of which are initial P-answers.

Some constructions on arenas are described below. Here we use $\overline{I_A}$ as an abbreviation for $M_A \backslash I_A$, and $\overline{\lambda_A}$ for the O/P-complement of λ_A. Intuitively $A \otimes B$ is the union of the arenas A and B, but with the initial moves combined pairwise. $A \Rightarrow B$ is slightly more complex. First we add a new initial move, $\bullet$. We take the O/P-complement of A, change the initial moves into questions, and set them to now be justified by $\bullet$. Finally, we take B and set its initial moves to be justified by A's initial moves. The final construction, $A \rightarrow B$, takes two arenas A and B and produces a prearena, as shown below. This is essentially the same as $A \Rightarrow B$ without the initial move $\bullet$.

$$M_{A\Rightarrow B} = \{\bullet\} \uplus M_A \uplus M_B \qquad M_{A\otimes B} = I_A \times I_B \uplus \overline{I_A} \uplus \overline{I_B}$$

$$I_{A\Rightarrow B} = \{\bullet\} \qquad I_{A\otimes B} = I_A \times I_B$$

$$\lambda_{A\Rightarrow B} = m \mapsto \begin{cases} PA & \text{if } m = \bullet \\ OQ & \text{if } m \in I_A \\ \overline{\lambda_A}(m) & \text{if } m \in \overline{I_A} \\ \lambda_B(m) & \text{if } m \in M_B \end{cases} \qquad \lambda_{A\otimes B} = m \mapsto \begin{cases} PA & \text{if } m \in I_A \times I_B \\ \lambda_A(m) & \text{if } m \in \overline{I_A} \\ \lambda_B(m) & \text{if } m \in \overline{I_B} \end{cases}$$

$$\vdash_{A\Rightarrow B} = \{(\bullet, i_A) | i_A \in I_A\} \cup \{(i_A, i_B) | i_A \in I_A, i_B \in I_B\} \cup \vdash_A \cup \vdash_B$$

$$\vdash_{A\otimes B} = \{((i_A, i_B), m) | i_A \in I_A \wedge i_B \in I_B \wedge (i_A \vdash_A m \vee i_B \vdash_B m)\} \cup (\vdash_A \cap (\overline{I_A} \times \overline{I_A})) \cup (\vdash_B \cap (\overline{I_B} \times \overline{I_B}))$$

$$M_{A\rightarrow B} \;=\; M_A \uplus M_B \qquad \lambda_{A\rightarrow B}(m) \;=\; \begin{cases} OQ & \text{if } m \in I_A \\ \overline{\lambda_A}(m) & \text{if } m \in \overline{I_A} \\ \lambda_B(m) & \text{if } m \in M_B \end{cases}$$

$$I_{A\rightarrow B} \;=\; I_A \qquad \vdash_{A\rightarrow B} \;=\; \{(i_A, i_B) | i_A \in I_A, i_B \in I_B\} \cup \vdash_A \cup \vdash_B$$

We intend arenas to represent types, in particular $[\![\mathsf{unit}]\!] = 1$, $[\![\mathsf{int}]\!] = \mathbb{Z}$ (or a finite subset of $\mathbb{Z}$ for $\mathsf{RML_f}$) and $[\![\theta_1 \rightarrow \theta_2]\!] = [\![\theta_1]\!] \Rightarrow [\![\theta_2]\!]$. A term $x_1 : \theta_1, \ldots, x_n : \theta_n \vdash M : \theta$ will be represented by a *strategy* for the prearena $[\![\theta_1]\!] \otimes \ldots \otimes [\![\theta_n]\!] \rightarrow [\![\theta]\!]$.

A *justified sequence* in a prearena A is a sequence of moves from A in which the first move is initial and all other moves m are equipped with a pointer to an earlier move m', such that $m' \vdash_A m$.

A *play* s is a justified sequence which additionally satisfies the following conditions.

(i) *Alternation*: O and P take it in turns to play moves. That is if $t\, m\, m' \sqsubseteq s$ then $\lambda^{OP}(m) \neq \lambda^{OP}(m')$.
(ii) *Well-Bracketing*: Questions asked first must be answered first. If $t\, q\, \widehat{t'}\, a \sqsubseteq s$ then all questions in t' must be answered in t'.
(iii) *Visibility*: If $t\, m\, \widehat{t'}\, m' \sqsubseteq s$ then m appears in $\mathsf{view}(t\, m\, t')$, where view is defined by, $\mathsf{view}(\epsilon) = \epsilon$, $\mathsf{view}(o) = o$ if o is initial, and $\mathsf{view}(t\, m\, \widehat{t'}\, m') = \mathsf{view}(t)\, m\, m'$.

We denote the set of all valid plays over prearena A as P_A.

A *strategy* σ for prearena A is a non-empty, even-prefix-closed set of plays from A, satisfying the condition that if $s\, m_1, s\, m_2 \in \sigma$ then $s\, m_1 = s\, m_2$.

We can think of a strategy as being a playbook telling P how to respond by mapping odd-length plays to moves.

A play is *complete* if all questions have been answered. Note that (unlike in the call-by-name case) a complete play is not necessarily maximal. We denote the set of complete plays in strategy σ by $\mathsf{comp}(\sigma)$.

Game Semantics of RML In the game semantic model of RML, a term-in-context $x_1 : \theta_1, \ldots, x_n : \theta_n \vdash M : \theta$ is represented by a strategy for the prearena $[\![\theta_1]\!] \otimes \ldots \otimes [\![\theta_n]\!] \rightarrow [\![\theta]\!]$. These strategies are built up compositionally over the syntax of the term. Essentially, free identifiers $x : \theta \vdash x : \theta$ are interpreted as *copy-cat* strategies where P always copies O's move into the other copy of $[\![\theta]\!]$, $\lambda x.M$ allows multiple copies of

$[\![M]\!]$ to be run, application MN requires a form of parallel composition plus hiding and the other constructions can be interpreted using special strategies. The game semantic model is fully abstract in the following sense.

Theorem 1 (Abramsky and McCusker 1997 [1,2]). *For all* RML*-terms-in-context* $\Gamma \vdash M, N : \theta$*, we have* $M \sqsubseteq\!\!\!\!\!\sim N$ *iff* $\mathsf{comp}([\![\Gamma \vdash M]\!]) \subseteq \mathsf{comp}([\![\Gamma \vdash N]\!])$.

We will show decidability of observational equivalence for a fragment of RML by representing the game semantics of terms as *Visibly Pushdown Automata* (VPA) [4]. VPA are a subclass of pushdown automata in which the stack action is uniquely determined by the input letter. The alphabet is partitioned into push-letters, pop-letters and noop-letters. On reading a letter the automaton must perform the appropriate action. (We write $s \xrightarrow{q/x} s'$ to mean "on reading q, push x" and $s \xrightarrow{a,x} s'$ to mean "on reading a and stack top is x, pop".) This gives them very attractive closure properties. In particular, equivalence of deterministic VPA is decidable in polynomial time.

3 Characterising the O-Strict Fragment of RML

In order to represent strategies using automata, we need to be able to encode plays (move sequence with pointers) as words. In some cases pointers can be uniquely reconstructed from the underlying move sequence, thanks to the visibility or well-bracketing conditions, which constrain the position of the justifying move. For instance, the targets of pointers from answer-moves can always be deduced from the underlying sequence of moves. In general, however, pointers must be encoded explicitly, and this poses a representational challenge because the target of a pointer can be arbitrarily far back in the history of the play. We say that a play is *O-strict* just if the pointer from every O-question in the play is uniquely determined by the underlying move sequence. A prearena is said to be *O-strict* if every play of the prearena is O-strict; a type sequent is *O-strict* if its denotation (in the call-by-value game semantics) is an O-strict prearena. It follows that when representing plays of an O-strict prearena, only pointers from P-questions need to be encoded. In this section, we aim to find a simple characterisation of the O-strict sequents.

Every RML-type θ can be written uniquely as $\theta_1 \to \cdots \to \theta_n \to \beta$ (by convention $\to$ associates to the right), where $n \geq 0$ and β stands for unit, int or int ref. In what follows we shall write $(\theta_1, \cdots, \theta_n, \beta)$ for θ. The *arity*, $ar(\theta)$, and *order*, $ord(\theta)$, of θ are defined as follows. $ar(\theta) := \begin{cases} n & \text{if } \beta = \mathsf{unit} \text{ or } \mathsf{int} \\ n+1 & \text{if } \beta = \mathsf{int\,ref}. \end{cases}$ $ord(\mathsf{unit}) = ord(\mathsf{int}) = 0$, $ord(\mathsf{int\,ref}) = 1$ and $ord(A \to B) = \max(ord(A)+1, ord(B))$. For clarity, we shall assume $\beta = \mathsf{unit}$ in the argument that follows; there is no loss of generality because essentially identical considerations work for the case of int, and int ref can be treated as unit $\to$ unit.

Types on the right of O-strict sequents. Consider an arena with the following enabling chain $q_0 \vdash a_0 \vdash q_1 \vdash a_1 \vdash q_2$. (For brevity, we shall say that the arena has a *qaqaq*-branch.) Then sequences of the form $q_0 a_0 (q_1 a_1)^n q_2$, where $n \geq 0$, are all plays, regardless of which occurrence of a_1 is used to justify q_2. Representing the pointer from the O-question q_2 would seem to require unbounded memory or an infinite alphabet.

Observe that the prearena of the type sequent $\vdash (\mathsf{unit}, \mathsf{unit}, \mathsf{unit})$ has a $qaqaq$-branch. In general, the same is the case for $\Gamma \vdash (\theta_1, \cdots, \theta_k, \mathsf{unit})$, where $k \geq 2$. In other words, the type on the right of an O-strict sequent has the shape (θ, unit) or is unit. Another troublesome sequent is $\vdash (((\mathsf{unit}, \mathsf{unit}), \mathsf{unit}), \mathsf{unit})$ which has a $qaqqq$-branch. In general, types of the form $((\theta_1, \cdots, \theta_k, \mathsf{unit}), \mathsf{unit})$ have a similar problem in case θ_i is functional for some $1 \leq i \leq k$. Thus, types on the right of an O-strict sequent must be of type Θ_2 (we shall call a type *short* just if it is in Θ_2) where

$$\Theta_1 ::= \mathsf{unit} \mid \mathsf{unit} \to \Theta_1 \qquad\qquad \Theta_2 ::= \mathsf{unit} \mid \Theta_1 \to \mathsf{unit}.$$

Equivalently, a type is in Θ_2 just if it has order at most 2 and arity at most 1.

Types on the left of O-strict sequents. Type sequents that contain $((\mathsf{unit}, \mathsf{unit}, \mathsf{unit}), \mathsf{unit})$ on the left are similarly problematic because the corresponding prearenas have a $qqqaq$-branch. Generally, sequents of the shape $\cdots, (\theta_1, \cdots, \theta_k, \mathsf{unit}), \cdots \vdash \cdots$ are not O-strict, if for some i, $\theta_i = (\theta_i^1, \cdots, \theta_i^{k'}, \mathsf{unit})$ and $k' \geq 2$.

Sequents that have the type $(((\mathsf{unit}, \mathsf{unit}), \mathsf{unit}), \mathsf{unit}), \mathsf{unit})$ on the left are also not O-strict because the corresponding prearenas have a $qqqqq$ branch. In general, this is the case for $\theta_i^1 = (\alpha_i^1, \cdots, \alpha_i^{k''}, \mathsf{unit})$, whenever some α_i^j is functional. Hence, if a sequent is O-strict, then each θ_i^1 must be of type Θ_1, i.e. each θ_i must be in Θ_2. This leads us to the class $\Theta_3 ::= \mathsf{unit} \mid \Theta_2 \to \Theta_3$. Equivalently a type is in Θ_3 just if it has shape $(\theta_1, \cdots, \theta_k, \mathsf{unit})$ where $k \geq 0$ and $\theta_i \in \Theta_2$ for each i. Note that Θ_3 contains Θ_1 but not Θ_2.

Lemma 1. *A type sequent, $\theta_1, \cdots, \theta_n \vdash \theta$, is O-strict iff $\theta \in \Theta_2$, and each $\theta_i \in \Theta_3$.*

So far we have omitted int and $\mathsf{int\,ref}$. To incorporate them into the characterisation, we treat int in the same way as unit, and $\mathsf{int\,ref}$ in the same way as $\mathsf{unit} \to \mathsf{unit}$. The revised definition of the collections, Θ_2 and Θ_3, thus reads as follows.

$$\begin{array}{llll} \Theta_0 ::= \mathsf{unit} \mid \mathsf{int} & & \Theta_2 ::= \Theta_0 \mid \Theta_1 \to \Theta_0 \mid \mathsf{int\,ref} \\ \Theta_1 ::= \Theta_0 \mid \Theta_0 \to \Theta_1 \mid \mathsf{int\,ref} & & \Theta_3 ::= \Theta_0 \mid \Theta_2 \to \Theta_3 \mid \mathsf{int\,ref} \end{array}$$

Definition 1. The *O-strict fragment of* RML, henceforth referred to as $\mathsf{RML}_{\text{O-Str}}$, consists of terms-in-context of the shape $x_1 : \Theta_3, \cdots, x_n : \Theta_3 \vdash M : \Theta_2$.

Since conversion to canonical form preserves types, canonical forms of $\mathsf{RML}_{\text{O-Str}}$-terms also belong to $\mathsf{RML}_{\text{O-Str}}$. Consequently, they satisfy the following properties.

(i) If $\Gamma \vdash \lambda x.\mathbb{C}$ is in $\mathsf{RML}_{\text{O-Str}}$, then $\Gamma, x : \Theta_1 \vdash \mathbb{C} : \Theta_0$.
(ii) If $\Gamma \vdash \mathsf{let}\, x = \mathsf{ref}\,\mathsf{in}\,\mathbb{C}$ is in $\mathsf{RML}_{\text{O-Str}}$, then $\Gamma, x \vdash \mathbb{C} : \Theta_2$.
(iii) If $\Gamma \vdash \mathsf{let}\, x = \cdots \,\mathsf{in}\,\mathbb{C}$ is in $\mathsf{RML}_{\text{O-Str}}$, then $\Gamma, x : \Theta_3 \vdash \mathbb{C} : \Theta_2$.
(iv) If $\Gamma \vdash \mathsf{let}\, x = z(\lambda y.\mathbb{C})\,\mathsf{in} \cdots$ is in $\mathsf{RML}_{\text{O-Str}}$, then $\Gamma, y : \Theta_1 \vdash \mathbb{C} : \Theta_0$.

Example 3. The following are $\mathsf{RML}_{\text{O-Str}}$ terms-in-contexts.

$$\text{(i)} \begin{cases} f : \mathsf{unit} \to \mathsf{unit} \to \mathsf{unit} \vdash \mathsf{let}\, g = f()\,\mathsf{in}\,(\mathsf{let}\, h = f()\,\mathsf{in}\, g()) : \mathsf{unit} \\ f : \mathsf{unit} \to \mathsf{unit} \to \mathsf{unit} \vdash \mathsf{let}\, g = f()\,\mathsf{in}\,(\mathsf{let}\, h = f()\,\mathsf{in}\, h()) : \mathsf{unit} \end{cases}$$

$$\text{(ii)} \begin{cases} f : ((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit} \vdash f(\lambda x^{\mathsf{unit} \to \mathsf{unit}}.f(\lambda y^{\mathsf{unit} \to \mathsf{unit}}.x())) : \mathsf{unit} \\ f : ((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit} \vdash f(\lambda x^{\mathsf{unit} \to \mathsf{unit}}.f(\lambda y^{\mathsf{unit} \to \mathsf{unit}}.y())) : \mathsf{unit} \end{cases}$$

(iii) The three pairs of terms in Example 1.

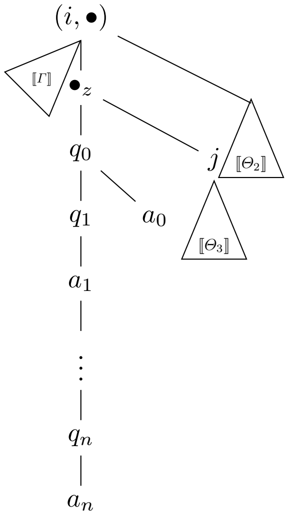
4 The O-Strict Fragment is VPA-Decidable

We shall show that the (fully abstract) game semantics of every $\mathsf{RML}_{\text{O-Str}}$-term can be faithfully represented using VPAs in the following sense.

Theorem 2. *There is an algorithm that transforms a given* $\mathsf{RML}_{\text{O-Str}}$*-term-in-context* $\Gamma \vdash M : \theta$ *to a VPA* $\mathcal{A}_{\Gamma \vdash M}$ *such that* $\Gamma \vdash M_1 \cong M_2$ *iff* $\mathcal{L}(\mathcal{A}_{\Gamma \vdash M_1}) = \mathcal{L}(\mathcal{A}_{\Gamma \vdash M_2})$.

Proof Outline. We wish to show that for all $\mathsf{RML}_{\text{O-Str}}$-terms $\Gamma \vdash M : \theta$ there exists (constructively) a VPA $\mathcal{A}_{\Gamma \vdash M}$ that accepts (some representation of) $[\![\Gamma \vdash M : \theta]\!]$.

To simplify this, we define $[\![\ldots]\!]_i$ by $[\![\Gamma \vdash M : \theta]\!] := \sum_{i \in I_{[\![\Gamma]\!]}} i\,[\![\Gamma \vdash M : \theta]\!]_i$. (To save space, we write $[\![\Gamma \vdash M : \theta]\!]$ simply as $[\![M]\!]$.) That is, $[\![M]\!]_i$ contains all plays of $[\![M]\!]$ which begin with initial move i, but with i removed. We will define automata $\mathcal{A}^i_M$ which accept the underlying move sequences of all complete plays in $[\![M]\!]_i$. To complete the proof, we will need to encode justification pointers, but for now we omit them. We partition our alphabet so that all P-questions are pushes, all O-answers pops and everything else noops.



Our construction proceeds inductively over the canonical forms. The simpler canonical forms can be described using regular expressions or as straightforward combinations of their subautomata. The case of λ-abstraction requires using the stack to nest copies of the body of the function. The construction for $\mathsf{let}\, x = \mathsf{ref}\, \mathsf{in}\, M$ stores the value of the variable in the state. The most complicated cases are those of the form $\mathsf{let}\, x = zM\, \mathsf{in}\, N$ and here we consider the hardest of them, $\mathsf{let}\, x = z(\lambda y.M)\, \mathsf{in}\, N$. The relevant prearena is shown in the figure on the right.

We assume the automata $\mathcal{A}^{(i,q_0)}_{\Gamma,y \vdash M}$ and $\mathcal{A}^{(i,j)}_{\Gamma,x \vdash N}$. To construct $\mathcal{A}^i_{\Gamma \vdash \mathsf{let}\, x = z(\lambda y.M)\, \mathsf{in}\, N}$ we take as our set of states:

$$Q^i_{\Gamma \vdash \mathsf{let}\, x=z(\lambda y.M)\, \mathsf{in}\, N} = \{(1),(2)\} \uplus \biguplus_{q_0 \in I_{[\![\theta_1]\!]}} Q^{(i,q_0)}_{\Gamma,y \vdash M} \uplus \biguplus_{j \in I_{[\![\theta_3]\!]}} Q^{(i,j)}_{\Gamma,x \vdash N}$$
$$\uplus \biguplus_{q_0 \in I_{[\![\theta_1]\!]},\, j \in I_{[\![\theta_3]\!]}} \left(Q^{(i,q_0)}_{\Gamma,y \vdash M} \times \widehat{Q^{(i,j)}_{\Gamma,x \vdash N}} \right)$$

where $\widehat{Q^{(i,j)}_{\Gamma,x \vdash N}}$ is the set of states s in $\mathcal{A}^{(i,j)}_{\Gamma,x \vdash N}$ such that $t \overset{m_x}{\rightarrow} s$ is a transition, with m_x a P-move in $[\![\Theta_3]\!]$. (1) is the initial state and the final states are those from $\mathcal{A}^{(i,j)}_{\Gamma,x \vdash N}$. The set of stack symbols is the disjoint union of the stack symbols used in the automata for M and N, plus the fresh symbol (1), plus the states of each $Q^{(i,q_0)}_{\Gamma,y \vdash M}$.

Large sections of the play will proceed as in $[\![M]\!]$ or $[\![N]\!]$. In particular, when in a $Q^{(i,j)}_{\Gamma,x \vdash N}$-state, play proceeds as in $\mathcal{A}^{(i,j)}_{\Gamma,x \vdash N}$ (although we will add additional transitions).

Similarly, when in a state with a $Q^{(i,q_0)}_{\Gamma,y\vdash M}$ component, play continues as in $\mathcal{A}^{(i,q_0)}_{\Gamma,y\vdash M}$ (although non-$[\![\Gamma]\!]$-transitions will be redirected). Hence we have that if $s_M \overset{m\Box}{\rightarrow} t_M$ in $\mathcal{A}^{(i,q_0)}_{\Gamma,y\vdash M}$ where m is a $[\![\Gamma]\!]$-move then in our new automaton we have $s_M \overset{m\Box}{\rightarrow} t_M$ and $(s_M, s_N) \overset{m\Box}{\rightarrow} (t_M, s_N)$ for all $s_N \in \widehat{Q^{(i,j)}_{\Gamma,x\vdash N}}$. Similarly, if $s_N \overset{m\Box}{\rightarrow} t_N$ in $\mathcal{A}^{(i,j)}_{\Gamma,x\vdash N}$ then we have $s_N \overset{m\Box}{\rightarrow} t_N$. Here we use $\overset{m\Box}{\rightarrow}$ to represent that this could be a push-, pop- or a noop-transition but whatever the case the transitions in the new automaton will perform the same stack action as in the old one.

The initial section of the play will correspond to evaluating $z(\lambda y.M)$. After the initial move, P will play $\bullet_z$. At this point, O can either play an initial $[\![\Theta_3]\!]$ move j, or play q_0, opening an M-thread. If O chooses the latter, play proceeds as in $[\![M]\!]$ until P plays in $[\![\Theta_1 \rightarrow \Theta_0]\!]$ (that is either P plays a_0, closing the M-thread, or some q_i). At this point O can choose to continue the current M-thread (unless P has closed it by playing a_0) or to open a new M-thread with q_0. Note that if O opens a new M-thread, while the old one is still open, the old thread will be left in a position where the only valid move is for O to answer the pending q_i with a_i. Thus bracketing ensures that we cannot revisit an old M-thread until we have closed the current one.

The transitions needed to represent this section of the play are:

- $(1) \overset{\bullet_z/(1)}{\rightarrow} (2)$.
- $(2) \overset{q_0}{\rightarrow} i^{q_0}_M$ where $i^{q_0}_M$ is the initial state in $\mathcal{A}^{(i,q_0)}_{\Gamma,y\vdash M}$.
- If $s_M \overset{a_0}{\rightarrow} t_M$ in $\mathcal{A}^{(i,q_0)}_{\Gamma,y\vdash M}$, then $s_M \overset{a_0}{\rightarrow} (2)$.
- If $s_M \overset{q_i/\gamma}{\rightarrow} t_M \overset{a_i,\gamma}{\rightarrow} u_M$, $i > 0$, in $\mathcal{A}^{(i,q_0)}_{\Gamma,y\vdash M}$ (note that this must be the only transition out of s_M), then $s_M \overset{q_i/(s_M)}{\rightarrow} (2)$ and $(2) \overset{a_i,(s_M)}{\rightarrow} u_M$.

Eventually we may reach a point where all M-threads are closed and O plays j, for which we have transitions $(2) \overset{j,(1)}{\rightarrow} i^j_N$ where i^j_N is the initial state in $\mathcal{A}^{(i,j)}_{\Gamma,x\vdash N}$. Play then proceeds as in $[\![N]\!]$, except that if P ever plays in x (that is in $[\![\Theta_3]\!]$), then O again gets the chance to play q_0 and open an M-thread. If this happens then as before the threads can be stacked. Further, whenever O has the chance to open a new M-thread, O also has the option of resuming play in N by playing in $[\![\Theta_3]\!]$. If there are currently open M-threads when O chooses to return to N, then to obey bracketing it must be a $[\![\Theta_3]\!]$-question which O plays. As before, the only way to resume an open M-thread is with an answer, so to obey bracketing this can only happen after the $[\![\Theta_3]\!]$-question is answered. To manage this, for all $s_N \in \widehat{Q^{(i,j)}_{\Gamma,x\vdash N}}$ we need to have the following transitions:

- $s_N \overset{q_0}{\rightarrow} (i^{q_0}_M, s_N)$, where $i^{q_0}_M$ is the initial state in $\mathcal{A}^{(i,q_0)}_{\Gamma,y\vdash M}$.
- If $s_M \overset{a_0}{\rightarrow} t_M$ in $\mathcal{A}^{(i,q_0)}_{\Gamma,y\vdash M}$, then $(s_M, s_N) \overset{a_0}{\rightarrow} s_N$.
- If $s_M \overset{q_i/\gamma}{\rightarrow} t_M \overset{a_i,\gamma}{\rightarrow} u_M$, $i > 0$, in $\mathcal{A}^{(i,q_0)}_{\Gamma,y\vdash M}$ then $(s_M, s_N) \overset{q_i/(s_M)}{\rightarrow} s_N$ and $s_N \overset{a_i,(s_M)}{\rightarrow} (u_M, s_N)$.

Remark 1. It follows from the construction that $\mathcal{A}_{\Gamma\vdash M:\theta}$ is regular if types of free variables are $((\mathsf{unit}, \mathsf{unit}), \cdots, (\mathsf{unit}, \mathsf{unit}), \mathsf{unit})$ and the type of M is $(\mathsf{unit}, \mathsf{unit})$ or unit.

Pointers. We now consider how to represent pointers. Since we are concerned with O-strict prearenas, we only try to encode pointers from P-moves. Further, instead of describing the location of every pointer in a play, in each run of the automaton we only give the position of a single pointer. However, for every pointer we need to represent there must be an accepting run encoding its location. Since our strategies are deterministic, each P-move has a unique justifier and so when we consider the full language accepted by the automaton this encoding scheme gives us sufficient information to reconstruct all justification pointers.

If $s\,m\,s'ns''$ is a sequence of moves, we will use $s\ \overset{\bullet}{m}\ s'\ \overset{\circ}{n}\ s''$ to represent that there is a pointer from (the P-move) n to m. We refer to moves tagged with • as target-moves and those tagged with ◦ as source-moves. We will construct automata $\mathcal{A}^i_M$ which accept all strings that are either the underlying move sequence of a complete play in $[\![M]\!]_i$ or the underlying move sequence plus the encoding of a single justification pointer from a P question. Note that as we omit the initial move, we cannot encode pointers that point to it. However, this is not a problem since there is only ever one occurrence of the initial move so any pointers to it are always uniquely reconstructible. All other justification pointers from P-questions must have a representation in the automaton's language. Note that if $\mathcal{L}(\mathcal{A}^i_{\Gamma\vdash M}) = \mathcal{L}(\mathcal{A}^i_{\Gamma\vdash N})$ for all $i \in I_{[\![\Gamma]\!]}$ then $\mathsf{comp}([\![\Gamma \vdash M]\!]) = \mathsf{comp}([\![\Gamma \vdash N]\!])$.

In the case of $\mathsf{let}\, x = z(\lambda y.M)\,\mathsf{in}\, N$ we must ensure we preserve all pointers from $[\![M]\!]$ and $[\![N]\!]$, plus that in each $[\![M]\!]$-thread q_1 can point to the q_0 that opened that thread and finally that if in $[\![N]\!]$ P plays an $[\![x]\!]$-move justified by j, this can point at the copy of j which started $[\![N]\!]$. We must also take care to enforce that each accepting run only contains the encoding of a single pointer.

Example 4. (i) Take the term $\mathsf{let}\, g = f()\,\mathsf{in}\,(\mathsf{while}\, b()\,\mathsf{do}\,(\mathsf{let}\, h = f()\,\mathsf{in}\,()));g()$, where $f : \mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}$ and $b : \mathsf{unit} \to \mathsf{int}$. The corresponding automaton will accept the following sequences: $q_f a_f (q_b 1_b q_f a_f)^* q_b 0_b q_{f'} a_{f'} a$ (no pointer information), $q_f\ \overset{\bullet}{a_f}\ (q_b 1_b q_f a_f)^* q_b 0_b q_{f'} a_{f'} a$ and $q_f a_f (q_b 1_b q_f a_f)^* q_b 1_b q_f\ \overset{\bullet}{a_f}\ (q_b 1_b q_f a_f)^* q_b 0_b q_{f'} a_{f'} a$ (information about single possible targets), and $q_f\ \overset{\bullet}{a_f}\ (q_b 1_b q_f a_f)^* q_b 0_b\ \overset{\circ}{q_{f'}}\ a_{f'} a$ (a single pointer is represented). Note that any occurrence of a_f could be a potential target for the pointer from $q_{f'}$. By annotating moves with • and ◦ we avoid the need for unbounded indices that would otherwise have to be used to represent pointers.

(ii) These automata represent the complete plays of the strategies for the terms of Example 3(i). As the language is regular we hide the stack actions. The underlying move sequences are identical but the encoding of pointers allows us to differentiate them.

$$\begin{tikzcd}
& & & 3 \arrow[r, "q_0"] & 5 \arrow[r, "a_0"] & 7 \arrow[drr, bend left, "\overset{\circ}{q_1}"] \arrow[drr, bend right, "q_1"'] & & & \\
{} \arrow[r] & 1 \arrow[r, "q_0"] & 2 \arrow[ur, "\overset{\bullet}{a_0}"] \arrow[dr, "a_0"'] & & & & & 9 \arrow[r, "a_1"] & 10 \arrow[r, "\bullet"] & 11 \\
& & & 4 \arrow[r, "q_0"] & 6 \arrow[r, bend left, "\overset{\bullet}{a_0}"] \arrow[r, bend right, "a_0"'] & 8 \arrow[urr, "q_1"'] & & &
\end{tikzcd}$$

$$\begin{tikzcd}
& & & 3 \arrow[r, "q_0"] & 5 \arrow[r, "a_0"] & 7 \arrow[drr, "q_1"] & & & \\
{} \arrow[r] & 1 \arrow[r, "q_0"] & 2 \arrow[ur, "\overset{\bullet}{a_0}"] \arrow[dr, "a_0"'] & & & & & 9 \arrow[r, "a_1"] & 10 \arrow[r, "\bullet"] & 11 \\
& & & 4 \arrow[r, "q_0"] & 6 \arrow[ur, "a_0"] \arrow[r, "\overset{\bullet}{a_0}"] & 8 \arrow[urr, bend left, "q_1"] \arrow[urr, bend right, "\overset{\circ}{q_1}"'] & & &
\end{tikzcd}$$

5 Complexity

Following [15], we define the size of a VPA to be the sum of the number of states and the number of stack symbols. The size of the alphabet is linear in the size of the input word and so we ignore it. The number of transitions is bounded by a polynomial in the size of the automaton.

In each case of the construction, the set of states consists of a number of fresh states, a number of copies of the states from sub-automata and a number of copies of pairs of states from different sub-automata. The set of stack symbols is similar. This means, that if automaton $\mathcal{A}_M$ is built up from n sub-automata $\mathcal{A}_{M_1} \ldots \mathcal{A}_{M_n}$ then $|\mathcal{A}_M| \leq c \times \left(1 + \sum |\mathcal{A}_{M_i}| + \sum_{i \neq j} |\mathcal{A}_{M_i}| \times |\mathcal{A}_{M_j}|\right)$, for some constant c. Given that at each step the size of the problem is greater than the sum of the size of the sub-problems, the implied recursion has an exponential bound.

Now the time required to construct each automaton is polynomial in its size (so exponential in the size of the input). The time taken to check whether two deterministic VPA are equivalent is polynomial in the size of the two VPA. Finally, the number of VPA we will need to check is exponential in the size of the input (in the number of int-components in the context). Altogether, this gives an exponential bound on the total amount of time required to check two $\mathsf{RML}_{\mathsf{O\text{-}Str}}$-terms in canonical form for observational equivalence.

It turns out that this bound is optimal. One can show EXPTIME-hardness using a reduction of the EXPTIME-complete equivalence problem for nondeterministic automata on binary trees [18]. Through that route, we can show that observational equivalence is EXPTIME-hard for canonical terms $gen : \mathsf{unit} \to \mathsf{int} \vdash \mathbb{C} : (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}$. The associated arena A is shown on the right.

$$\begin{tikzcd}
 & q \arrow[dl, no head] \arrow[drr, no head] & & \\
q_{gen} \arrow[d, no head] & & & \star \arrow[d, no head] \\
a_{gen} & & & q_0 \arrow[dl, no head] \arrow[d, no head] \\
 & & q_1 \arrow[d, no head] & a_0 \\
 & & a_1 &
\end{tikzcd}$$

In order to represent (ranked) binary trees, let us assume that values of type int are partitioned into the set of binary and nullary labels, ranged over l_2 and l_0 respectively. Then any ranked binary tree T over such labels can be represented by the play $q \star \mathcal{S}(T)$ on A, where $\mathcal{S}(T)$ is defined as $\mathcal{S}(l) := q_0\, q_{gen}\, l_{gen}\, a_0$; $\mathcal{S}(n(T_1, T_2)) := q_0\, q_{gen}\, n_{gen}\, q_1\, \mathcal{S}(T_1)\, a_1\, q_1\, \mathcal{S}(T_2)\, a_1\, a_0$. Note that $\mathcal{S}(T)$ can be seen as a record of a depth-first traversal of T. The key to the hardness argument is the construction of a term $gen : \mathsf{unit} \to \mathsf{int} \vdash \mathbb{C}_{\mathcal{A}} : (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}$ for a given tree automaton $\mathcal{A}$ such that $\mathsf{comp}([\![gen \vdash \mathbb{C}_{\mathcal{A}}]\!]) = \{q\star\} \cup \{q \star \mathcal{S}(T) \,|\, T \in \mathcal{T}(\mathcal{A})\}$, where $\mathcal{T}(\mathcal{A})$ is the set of trees accepted by $\mathcal{A}$. To that end we take advantage of the term $\vdash \lambda f.f(); f() : (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}$. Observe that it generates complete plays that are very similar to the plays used to represent trees: they have the form $q \star X$, where $X ::= \epsilon \mid q_0\, q_1\, X\, a_1\, q_1\, X\, a_1\, a_0\, X$. To construct $\mathbb{C}_{\mathcal{A}}$ we can equip the term above with additional code that tracks possible states of $\mathcal{A}$, as the input tree is being traversed. In order to cover all possible tree shapes the free identifier $gen : \mathsf{unit} \to \mathsf{int}$ is used as a label generator.

Alternatively, one could readily adapt the EXPTIME-hardness argument for third-order Idealized Algol [15] to the call-by-value setting. This would yield EXPTIME-hardness of observational equivalence for canonical forms typable as

$gen : \mathsf{unit} \to \mathsf{int}, f : ((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit} \vdash \mathbb{C} : \mathsf{unit}$.

Theorem 3. *Observational equivalence of* $\mathsf{RML}_{\text{O-Str}}$*-terms in canonical form is EXPTIME-complete.*

Further Directions Does $\mathsf{RML}_{\text{O-Str}}$ capture all the decidable sequents? (We think not.) It would be interesting to identify (and classify) the decidable type sequents of the following related languages. (i) Call-by-value Idealized Algol [5,13], which can be viewed as a fragment of RML with block-allocated storage. It is known that the order-2 fragment is decidable [13] and not order-5 [10]. (ii) The case of Reduced ML [19] (i.e. RML without mkvar) is significantly more complicated. Recently it was shown that terms-in-contexts of the shape$\cdots, x : \beta \to \beta, \cdots \vdash M : \beta \to \beta$, where $\beta = \mathsf{unit}, \mathsf{int}, \mathsf{int\,ref}$, can be represented with automata over infinite alphabets [14].

Another direction we intend to pursue is to implement the model checking algorithm described, building upon the infrastructure of the call-by-name tool Homer [8].

References

1. Abramsky, S., McCusker, G.: Call-by-value games. In: Nielsen, M. (ed.) CSL 1997. LNCS, vol. 1414. Springer, Heidelberg (1998)
2. Abramsky, S., McCusker, G.: Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In: Algol-like Languages. Birkhäuser, Basel (1997)
3. Ahmed, A., Dreyer, D., Rossberg, A.: State-dependent representation independence. In: POPL (2009)
4. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC (2004)
5. Ghica, D.R.: Regular-language semantics for a call-by-value programming language. In: MFPS (2001)
6. Ghica, D.R., McCusker, G.: The regular-language semantics of second-order Idealized Algol. Theor. Comput. Sci. 309(1-3) (2003)
7. Honda, K., Yoshida, N.: Game theoretic analysis of call-by-value computation. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256. Springer, Heidelberg (1997)
8. Hopkins, D., Ong, C.-H.L.: HOMER: A Higher-Order Observational Equivalence Model checkER. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 654–660. Springer, Heidelberg (2009)
9. Hyland, J.M.E., Ong, C.-H.L.: On full abstraction for PCF: I, II, and III. Inf. Comput. 163(2) (2000)
10. Murawski, A.S.: About the undecidability of program equivalence in finitary languages with state. ACM Transactions on Computational Logic 6(4) (2005)
11. Murawski, A.S.: Functions with local state: regularity and undecidability. Theoretical Computer Science 338(1/3) (2005)
12. Murawski, A.S., Ong, C.-H.L., Walukiewicz, I.: Idealized algol with ground recursion, and DPDA equivalence. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 917–929. Springer, Heidelberg (2005)
13. Murawski, A.S., Tzevelekos, N.: Block structure vs. Scope extrusion: Between innocence and omniscience. In: Ong, L. (ed.) FOSSACS 2010. LNCS, vol. 6014, pp. 33–47. Springer, Heidelberg (2010)

14. Murawski, A.S., Tzevelekos, N.: Algorithmic nominal game semantics. In: Barthe, G. (ed.) ESOP 2011. LNCS, vol. 6602, pp. 419–438. Springer, Heidelberg (2011)
15. Murawski, A.S., Walukiewicz, I.: Third-order idealized algol with iteration is decidable. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 202–218. Springer, Heidelberg (2005)
16. Pitts, A.M., Stark, I.D.B.: Operational reasoning for functions with local state. In: Higher Order Operational Techniques in Semantics (1998)
17. Reynolds, J.C.: The essence of Algol. In: de Bakker, J.W., van Vliet, J.C. (eds.) Algorithmic Languages, pp. 345–372. North Holland, Amsterdam (1978)
18. Seidl, H.: Deciding equivalence of finite tree automata. SIAM J. Comput. 19(3) (1990)
19. Stark, I.D.B.: Names and Higher-Order Functions. PhD thesis, Univ. of Cambridge (1995)