Fragments of ML Decidable by Nested Data Class Memory Automata

Conrad Cotton-Barratt[1,⋆], 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 Warwick, UK

Abstract. The 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. We consider the fragment of (finitary) RML terms of order at most 1 with free variables of order at most 2, and identify two subfragments of this for which we show observational equivalence to be decidable. The first subfragment, $\mathrm{RML}_{2|-1}^{\text{P-Str}}$, consists of those terms in which the P-pointers in the game semantic representation are determined by the underlying sequence of moves. The second subfragment consists of terms in which the O-pointers of moves corresponding to free variables in the game semantic representation are determined by the underlying moves. These results are shown using a reduction to a form of automata over data words in which the data values have a tree-structure, reflecting the tree-structure of the threads in the game semantic plays. In addition we show that observational equivalence is undecidable at every third- or higher-order type, every second-order type which takes at least two first-order arguments, and every second-order type (of arity greater than one) that has a first-order argument which is not the final argument.

1 Introduction

RML is a call-by-value functional language with state [2]. It is similar to Reduced ML [17], the canonical restriction of Standard ML to ground-type references, except that it includes a "bad variable" constructor (in the absence of the constructor, the equality test is definable). This paper concerns the decidability of observational equivalence of finitary RML, $\mathrm{RML}_{\mathrm{f}}$. Our ultimate goal is to classify the decidable fragments of $\mathrm{RML}_{\mathrm{f}}$ completely. In the case of finitary Idealized Algol (IA), the decidability of observational equivalence depends only on the type-theoretic order [13] of the type sequents. In contrast, the decidability of $\mathrm{RML}_{\mathrm{f}}$ sequents is not so neatly characterised by order (see Figure 1): there are undecidable sequents of order as low as 2 [12], amidst interesting classes of decidable sequents at each of orders 1 to 4.

Following Ghica and McCusker [6], we use game semantics to decide observational equivalence of $\mathrm{RML}_{\mathrm{f}}$. Take a sequent $\Gamma \vdash M : \theta$ with $\Gamma = x_1 : \theta_1, \cdots, x_n : \theta_n$. In

⋆ Supported by an EPSRC Doctoral Training Grant
⋆⋆ Supported by Microsoft Research and Tony Hoare. Now at Ensoft Limited, UK.
⋆⋆⋆ Supported by EPSRC (EP/J019577/1)
† Partially supported by Merton College Research Fund

game semantics [7][10], the type sequent 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 – its justifier. Thanks to the fully abstract game semantics of RML, observational equivalence is characterised by *complete plays* i.e. $\Gamma \vdash M \cong N$ iff the P-strategies, $[\![\Gamma \vdash M]\!]$ and $[\![\Gamma \vdash N]\!]$, contain the same set of complete plays. Strategies may be viewed as highly constrained processes, and are amenable to automata-theoretic representations; the chief technical challenge lies in the encoding of pointers.

In [9] we introduced the *O-strict* fragment of $\mathrm{RML_f}$, $\mathrm{RML_{O\text{-}Str}}$, consisting of sequents $x_1 : \theta_1, \cdots, x_n : \theta_n \vdash M : \theta$ such that θ is *short* (i.e. order at most 2 and arity at most 1), and every argument type of every θ_i is short. Plays over prearenas denoted by O-strict sequents enjoy the property that the pointers from O-moves are uniquely determined by the underlying move sequence. The main result in [9] is that the set of complete plays of a $\mathrm{RML_{O\text{-}Str}}$-sequent is representable as a visibly pushdown automaton (VPA). A key idea is that it suffices to require each word of the representing VPA to encode the pointer from only *one* P-question. The point is that, when the full word language is analysed, it will be possible to uniquely place all justification pointers.

The simplest type that is not O-strict is $\beta \to \beta \to \beta$ where $\beta \in \{\mathsf{int}, \mathsf{unit}\}$. Encoding the pointers from O-moves is much harder because O-moves are controlled by the environment rather than the term. As observational equivalence is defined by a quantification over all contexts, the strategy for a term must consider *all* legal locations of pointer from an O-move, rather than just a single location in the case of pointer from a P-move. In this paper, we show that automata over data words can precisely capture strategies over a class of non-O-strict types.

Contributions. We identify two fragments of $\mathrm{RML_f}$ in which we can use deterministic weak nested data class memory automata [4] (equivalent to the locally prefix-closed nested data automata in [5]) to represent the set of complete plays of terms in these fragments. These automata operate over a data set which has a tree structure, and we use this structured data to encode O-pointers in words.

Both fragments are contained with the fragment $\mathrm{RML}_{2\vdash 1}$, which consists of terms-in-context $\Gamma \vdash M$ where every type in Γ is order at most 2, and the type of M is order at most 1. The first fragment, the *P-Strict subfragment*, consists of those terms in $\mathrm{RML}_{2\vdash 1}$ for which in the game semantic arenas have the property that the P-pointers in plays are uniquely determined by the underlying sequence of moves. This consists of terms-in-context $\Gamma \vdash M : \theta$ in which θ is any first order type, and each type in Γ has arity at most 1 and order at most 2. The second fragment, $\mathrm{RML}^{\mathrm{res}}_{2\vdash 1}$, consists of terms-in-context $\Gamma \vdash M : \theta$ in which θ, again, is any first order type, and each type $\theta' \in \Gamma$ is at most order 2, such that each argument for θ' has arity at most 1. Although these two fragments are very similar, they use different encodings of data values, and we discuss the difficulties in extending these techniques to larger fragments of $\mathrm{RML_f}$.

Finally we show that observational equivalence is undecidable at every third- or higher-order type, every second-order type which takes at least two first-order arguments, and every second-order type (of arity greater than one) that has a first-order argument which is not the final argument. See Figure 1 for a summary.

Fragment	Representative Type Sequent	Recursion	Ref.
Decidable			
O-Strict / $\mathrm{RML}_{\text{O-Str}}$ (ExpTime-Complete)	$((\beta \to \ldots \to \beta) \to \beta) \to \ldots \to \beta \vdash (\beta \to \ldots \to \beta) \to \beta$	**while**	[8,9]
O-Strict + Recursion (DPDA-Hard)	$((\beta \to \ldots \to \beta) \to \beta) \to \ldots \to \beta \vdash (\beta \to \ldots \to \beta) \to \beta$	$\beta \to \beta$	[8]
$\mathrm{RML}_{2\vdash 1}^{\text{P-Str}}$	$(\beta \to \cdots \to \beta) \to \beta \vdash \beta \to \cdots \to \beta$	**while**	†
$\mathrm{RML}_{2\vdash 1}^{\text{res}}$	$(\beta \to \beta) \to \cdots \to (\beta \to \beta) \to \beta \vdash \beta \to \cdots \to \beta$	**while**	†
Undecidable			
Third-Order	$\vdash ((\beta \to \beta) \to \beta) \to \beta$ $(((\beta \to \beta) \to \beta) \to \beta) \to \beta \vdash \beta$	$\bot$	[8],†
Second-Order	$\vdash (\beta \to \beta) \to \beta \to \beta$ $((\beta \to \beta) \to \beta \to \beta) \to \beta \vdash \beta$	$\bot$	[8],†
Recursion	Any	$(\beta \to \beta) \to \beta$	[8],†
Unknown			
$\mathrm{RML}_{2\vdash 1}$	$(\beta \to \cdots \to \beta) \to \cdots \to (\beta \to \cdots \to \beta) \to \beta \vdash \beta \to \cdots \to \beta$	$\bot$	-
RML_X	$\vdash \beta \to (\beta \to \beta) \to \beta$ $((\beta \to \beta) \to \beta) \to \beta \vdash \beta \to \beta \to \beta$	$\bot$	-
FO RML + Recursion	$\vdash \beta \to \cdots \to \beta$	$\beta \to \beta \to \beta$	-

Fig. 1: Summary of RML Decidability Results. († marks new results presented here; $\beta \in \{\mathsf{int}, \mathsf{unit}\}$; we write $\bot$ to mean an undecidability result holds (or none is known) even if no recursion or loops are present, and the only source of non-termination is through the constant Ω)

Related Work. A related language with full ground references (i.e. with a int ref ref type) was studied in [15], and observational equivalence was shown to be undecidable even at types $\vdash \mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}$. In contrast, for $\mathrm{RML}_{\mathrm{f}}$ terms, we show decidability at the same type. The key technical innovation of our work is the use of automata over infinite alphabets to encode justification pointers. Automata over infinite alphabets have already featured in papers on game semantics [14,15] but there they were used for a different purpose, namely, to model fresh-name generation. The nested data class memory automata we use in this paper are an alternative presentation of locally prefix-closed data automata [5].

2 Preliminaries

RML We assume base types unit, for commands, int for a finite set of integers, and a integer variable type, int ref. Types are built from these in the usual way. The *order* of a type $\theta \to \theta'$ is given by $max(order(\theta) + 1, order(\theta'))$, where base types unit and int have order 0, and int ref has order 1. The *arity* of a type $\theta \to \theta'$ is $arity(\theta') + 1$ where unit and int have arity 0, and int ref has arity 1. A full syntax and set of typing rules for RML is given in Figure 2. Note though we include only the arithmetic operations $\mathbf{succ}(i)$ and $\mathbf{pred}(i)$, these are sufficient to define all the usual comparisons and operations. We will write $\mathbf{let}\, x = M \,\mathbf{in}\, N$ as syntactic sugar for $(\lambda x.N)M$, and $M; N$ for $(\lambda x.N)M$ where x is a fresh variable.

$$\frac{}{\Gamma \vdash () : \mathsf{unit}} \qquad \frac{i \in \mathbb{N}}{\Gamma \vdash i : \mathsf{int}} \qquad \frac{\Gamma \vdash M : \mathsf{int}}{\Gamma \vdash \mathbf{succ}(M) : \mathsf{int}} \qquad \frac{\Gamma \vdash M : \mathsf{int}}{\Gamma \vdash \mathbf{pred}(M) : \mathsf{int}}$$

$$\frac{\Gamma \vdash M : \mathsf{int} \qquad \Gamma \vdash M_0 : \theta \qquad \Gamma \vdash M_1 : \theta}{\Gamma \vdash \mathbf{if}\, M\, \mathbf{then}\, M_1\, \mathbf{else}\, M_0 : \theta} \qquad \frac{\Gamma \vdash M : \mathsf{int\ ref}}{\Gamma \vdash !M : \mathsf{int}}$$

$$\frac{\Gamma \vdash M : \mathsf{int\ ref} \qquad \Gamma \vdash N : \mathsf{int}}{\Gamma \vdash M := N : \mathsf{unit}} \qquad \frac{\Gamma \vdash M : \mathsf{int}}{\Gamma \vdash \mathbf{ref}\, M : \mathsf{int\ ref}} \qquad \frac{}{\Gamma, x : \theta \vdash x : \theta}$$

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

$$\frac{\Gamma \vdash M : \mathsf{int} \qquad \Gamma \vdash N : \mathsf{unit}}{\Gamma \vdash \mathbf{while}\, M\, \mathbf{do}\, N : \mathsf{unit}} \qquad \frac{\Gamma \vdash M : \mathsf{unit} \to \mathsf{int} \qquad \Gamma \vdash N : \mathsf{int} \to \mathsf{unit}}{\Gamma \vdash \mathbf{mkvar}(M, N) : \mathsf{int\ ref}}$$

Fig. 2: Syntax of RML

The operational semantics, defined in terms of a big-step relation, are standard [12]. For closed terms $\vdash M$ we write $M\Downarrow$ just if there exist s, V such that $\emptyset, M \Downarrow s, V$. Two terms $\Gamma \vdash M : \theta$ and $\Gamma \vdash N : \theta$ are *observationally equivalent* (or *contextually equivalent*) if for all (closing) contexts $C[-]$ such that $\emptyset \vdash C[M], C[N] : \mathsf{unit}$, $C[M]\Downarrow$ if and only if $C[N]\Downarrow$.

It can be shown that every RML term is effectively convertible to an equivalent term in *canonical form* [8, Prop. 3.3], defined by the following grammar ($\beta \in \{\mathsf{unit}, \mathsf{int}\}$).

$$\begin{aligned}\mathbb{C} ::= {} & () \mid i \mid x^{\beta} \mid \mathbf{succ}(x^{\beta}) \mid \mathbf{pred}(x^{\beta}) \mid \mathbf{if}\, x^{\beta}\, \mathbf{then}\, \mathbb{C}\, \mathbf{else}\, \mathbb{C} \mid x^{\mathsf{int\ ref}} := y^{\mathsf{int}} \mid !x^{\mathsf{int\ ref}} \mid \\ & \lambda x^{\theta}.\mathbb{C} \mid \mathbf{mkvar}(\lambda x^{\mathsf{unit}}.\mathbb{C}, \lambda y^{\mathsf{int}}.\mathbb{C}) \mid \mathbf{let}\, x = \mathbf{ref}\, 0\, \mathbf{in}\, \mathbb{C} \mid \mathbf{while}\, \mathbb{C}\, \mathbf{do}\, \mathbb{C} \mid \mathbf{let}\, x^{\beta} = \mathbb{C}\, \mathbf{in}\, \mathbb{C} \mid \\ & \mathbf{let}\, x = zy^{\beta}\, \mathbf{in}\, \mathbb{C} \mid \ \mathbf{let}\, x = z\, \mathbf{mkvar}(\lambda u^{\mathsf{unit}}.\mathbb{C}, \lambda v^{\mathsf{int}}.\mathbb{C})\, \mathbf{in}\, \mathbb{C} \mid \mathbf{let}\, x = z(\lambda x^{\theta}.\mathbb{C})\, \mathbf{in}\, \mathbb{C}\end{aligned}$$

Game Semantics We use a presentation of call-by-value game semantics in the style of Honda and Yoshida [7], as opposed to Abramsky and McCusker's isomorphic model [2], as Honda and Yoshida's more concrete constructions lend themselves more easily to recognition by automata. We recall the following presentation of the game semantics for RML from [9].

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 $\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$ 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. The constructions on arenas are defined in Figure 3. 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, $A \to B$, 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 $\bullet$.

$$
\begin{array}{ll}
M_{A \Rightarrow B} = \{\bullet\} \uplus M_A \uplus M_B & M_{A \otimes B} = I_A \times I_B \uplus \overline{I_A} \uplus \overline{I_B} \\
I_{A \Rightarrow B} = \{\bullet\} & I_{A \otimes B} = I_A \times I_B \\
\lambda_{A \Rightarrow B} = m \mapsto \begin{cases} PA & \text{if } m = \bullet \\ OQ & \text{if } m \in I_A \\ \overline{\lambda_A}(m) & \text{if } m \in \overline{I_A} \\ \lambda_B(m) & \text{if } m \in M_B \end{cases} & \lambda_{A \otimes B} = m \mapsto \begin{cases} PA & \text{if } m \in I_A \times I_B \\ \lambda_A(m) & \text{if } m \in \overline{I_A} \\ \lambda_B(m) & \text{if } m \in \overline{I_B} \end{cases} \\
\vdash_{A \Rightarrow B} = \{(\bullet, i_A) | i_A \in I_A\} & \vdash_{A \otimes B} = \{((i_A, i_B), m) | i_A \in I_A \wedge i_B \in I_B \\
\quad \cup \{(i_A, i_B) | i_A \in I_A, i_B \in I_B\} & \qquad \wedge (i_A \vdash_A m \vee i_B \vdash_B m)\} \\
\quad \cup \vdash_A \cup \vdash_B & \quad \cup (\vdash_A \cap (\overline{I_A} \times \overline{I_A})) \\
& \quad \cup (\vdash_B \cap (\overline{I_B} \times \overline{I_B}))
\end{array}
$$

$$
\begin{array}{ll}
M_{A \to B} = M_A \uplus M_B & \lambda_{A \to B}(m) = \begin{cases} OQ & \text{if } m \in I_A \\ \overline{\lambda_A}(m) & \text{if } m \in \overline{I_A} \\ \lambda_B(m) & \text{if } m \in M_B \end{cases} \\
I_{A \to B} = I_A & \vdash_{A \to B} = \{(i_A, i_B) | i_A \in I_A, i_B \in I_B\} \cup \vdash_A \cup \vdash_B
\end{array}
$$

Fig. 3: Constructions on Arenas

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

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 standard conditions of Alternation, Well-Bracketing, and Visibility.

A *strategy* σ for prearena A is a non-empty, even-prefix-closed set of plays from A, satisfying the determinism condition: 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 $\mathbf{comp}(\sigma)$.

In the game model of RML, a term-in-context $x_1 : \theta_1, \ldots, x_n : \theta_n \vdash M : \theta$ is interpreted by a strategy of the prearena $[\![\theta_1]\!] \otimes \ldots \otimes [\![\theta_n]\!] \to [\![\theta]\!]$. These strategies are defined by recursion over the syntax of the term. 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 [1,2]). *If* $\Gamma \vdash M : \theta$ *and* $\Gamma \vdash N : \theta$ *are* RML *type sequents, then* $\Gamma \vdash M \cong N$ *iff* $\mathbf{comp}([\![\Gamma \vdash M]\!]) = \mathbf{comp}([\![\Gamma \vdash N]\!])$.

Nested Data Class Memory Automata We will be using automata to recognise game semantic strategies as languages. Equality of strategies can then be reduced to equivalence of the corresponding automata. However, to represent strategies as languages we must encode pointers in the words. To do this we use data languages, in which every position in a word has an associated *data value*, which is drawn from an infinite set (which we call the *data set*). Pointers between positions in a play can thus be encoded in the word by the relevant positions having suitably related data values. Reflecting the hierarchical structure of the game semantic prearenas, we use a data set with a tree-structure.

Recall a *tree* is a simple directed graph $\langle D, pred \rangle$ where $pred : D \rightharpoonup D$ is the predecessor map defined on every node of the tree except the root, such that every node has a unique path to the root. A node n has level l just if $pred^l(n)$ is the root (thus the root has level 0). A tree is of level l just if every node in it has level $\leq l$. We define a *nested data set* of level l to be a tree of level l such that each data value of level strictly less than l has infinitely many children. We fix a nested data set of level l, $\mathcal{D}$, and a finite alphabet Σ, to give a data alphabet $\mathbb{D} = \Sigma \times \mathcal{D}$.

We will use a form of automaton over these data sets based on class memory automata [3]. Class memory automata operate over an unstructured data set, and on reading an input letter (a, d), the transitions available depend both on the state the automaton is currently in, and the state the automaton was in after it last read an input letter with data value d. We will be extending a weaker variant of these automata, in which the only acceptance condition is reaching an accepting state. The variant of class memory automata we will be using, nested data class memory automata [4], works similarly: on reading input (a, d) the transitions available depend on the current state of the automaton, the state the automaton was in when it last read a descendant (under the $pred$ function) of d, and the states the automaton was in when it last read a descendant of each of d's ancestors. We also add some syntactic sugar (not presented in [4]) to this formalism, allowing each transition to determine the automaton's memory of where it last saw the read data value and each of its ancestors: this does not extend the power of the automaton, but will make the constructions we make in this paper easier to define.

Formally, a Weak Nested Data Class Memory Automaton (WNDCMA) of level l is a tuple $\langle Q, \Sigma, \Delta, q_0, F \rangle$ where Q is the set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states, and the transition function $\delta = \bigcup_{i=0}^{l} \delta_i$ where each δ_i is a function:

$$\delta_i : Q \times \Sigma \times (\{i\} \times (Q \uplus \{\bot\})^{i+1}) \to \mathcal{P}(Q \times Q^{i+1})$$

We write $Q_\bot$ for the set $Q \uplus \{\bot\}$, and may refer to the $Q_\bot^j$ part of a transition as its *signature*. The automaton is *deterministic* if each set in the image of δ is a singleton. A configuration is a pair (q, f) where $q \in Q$, and $f : \mathcal{D} \to Q_\bot$ is a class memory function (i.e. $f(d) = \bot$ for all but finitely many $d \in \mathcal{D}$). The initial configuration is (q_0, f_0) where f_0 is the class memory function mapping every data value to $\bot$. The automaton can transition from configuration (q, f) to configuration (q', f') on reading input (a, d) just if d is of level-i, $(q', (t_0, t_1, \ldots, t_i)) \in \delta(q, a, (i, f(pred^i(d)), \ldots, f(pred(d)), f(d)))$,

and $f' = f[d \mapsto t_i, pred(d) \mapsto t_{i-1}, \ldots, pred^{i-1}(d) \mapsto t_1, pred^i(d) \mapsto t_0]$. A run is defined in the usual way, and is accepting if the last configuration (q_n, f_n) in the run is such that $q_n \in F$. We say $w \in L(\mathcal{A})$ if there is an accepting run of $\mathcal{A}$ on w.

Weak nested data class memory automata have a decidable emptiness problem, reducible to coverability in a well-structured transition system [4,5], and are closed under union and intersection by the standard automata product constructions. Further, Deterministic WNDCMA are closed under complementation again by the standard method of complementing the final states. Hence they have a decidable equivalence problem.

3 P-Strict $\mathrm{RML}_{2\vdash 1}$

In [9], the authors identify a fragment of RML, the O-strict fragment, for which the plays in the game-semantic strategies representing terms have the property that the justification pointers of O-moves are uniquely reconstructible from the underlying moves. Analogously, we define the P-strict fragment of RML to consist of typed terms in which the pointers for P-moves are uniquely determined by the underlying sequence of moves. Then our encoding of strategies for this fragment will only need to encode the O-pointers: for which we will use data values.

3.1 Characterising P-Strict RML

In working out which type sequents for RML lead to prearenas which are P-strict, it is natural to ask for a general characterisation of such prearenas. The following lemma, which provides exactly that, is straightforward to prove:

Lemma 1. *A prearena is P-strict iff there is no enabling sequence $q \vdash \cdots \vdash q'$ in which both q and q' are P-questions.*

Which type sequents lead to a P-question hereditarily justifying another P-question? It is clear, from the construction of the prearena from the type sequent, that if a free variable in the sequent has arity > 1 or order > 2, the resulting prearena will have a such an enabling sequence, so not be P-strict. Conversely, if a free variable is of a type of order at most 2 and arity at most 1, it will not break P-strictness. On the RHS of the type sequent, things are a little more complex: there will be a "first" P-question whenever the type has an argument of order ≥ 1. To prevent this P-question hereditarily justifying another P-question, the argument must be of arity 1 and order ≤ 2. Hence the P-strict fragment consists of type sequents of the following form:

$$(\beta \to \cdots \to \beta) \to \beta \vdash ((\beta \to \cdots \to \beta) \to \beta) \to \cdots \to ((\beta \to \cdots \to \beta) \to \beta) \to \beta$$

(where $\beta \in \{\mathsf{unit}, \mathsf{int}\}$.)

From results shown here and in [8], we know that observational equivalence of all type sequents with an order 3 type or order 2 type with order 1 non-final argument on the RHS are undecidable. Hence the only P-strict types for which observational equivalence may be decidable are of the form: $(\beta \to \cdots \to \beta) \to \beta \vdash \beta \to \cdots \to \beta$ or $(\beta \to \cdots \to \beta) \to \beta \vdash \beta \to \cdots \to \beta \to (\beta \to \beta) \to \beta$. In this section we show that the first of these, which is the intersection of the P-strict fragment and $\mathrm{RML}_{2\vdash 1}$, does lead to decidability.

Definition 1. *The P-Strict fragment of* $\mathrm{RML}_{2\vdash 1}$*, which we denote* $\mathrm{RML}_{2\vdash 1}^{\text{P-Str}}$*, consists of typed terms of the form* $x_1 : \widehat{\Theta_1}, \ldots, x_n : \widehat{\Theta_1} \vdash M : \Theta_1$ *where the type classes* Θ_i *are as described below:*

$$\Theta_0 ::= \mathsf{unit} \mid \mathsf{int} \qquad \Theta_1 ::= \Theta_0 \mid \Theta_0 \to \Theta_1 \mid \mathsf{int\ ref} \qquad \widehat{\Theta_1} ::= \Theta_0 \mid \Theta_1 \to \Theta_0 \mid \mathsf{int\ ref}$$

This means we allow types of the form $(\beta \to \cdots \to \beta) \to \beta \vdash \beta \to \cdots \to \beta$ where $\beta \in \{\mathsf{unit}, \mathsf{int}\}$.

3.2 Deciding Observational Equivalence of $\mathrm{RML}_{\mathbf{2\vdash 1}}^{\text{P-Str}}$

Our aim is to decide observational equivalence by constructing, from a term M, an automaton that recognises a language representing $[\![M]\!]$. As $[\![M]\!]$ is a set of plays, the language representing $[\![M]\!]$ must encode both the moves and the pointers in the play. Since answer moves' pointers are always determined by well-bracketing, we only represent the pointers of question moves, and we do this with the nested data values. The idea is simple: if a play s is in $[\![M]\!]$ the language $L([\![M]\!])$ will contain a word, w, such that the string projection of w is the underlying sequence of moves of s, and such that:

- The initial move takes the (unique) level-0 data value; and
- Answer moves take the same data value as that of the question they are answering; and
- Other question moves take a fresh data value whose predecessor is the data value taken by the justifying move.

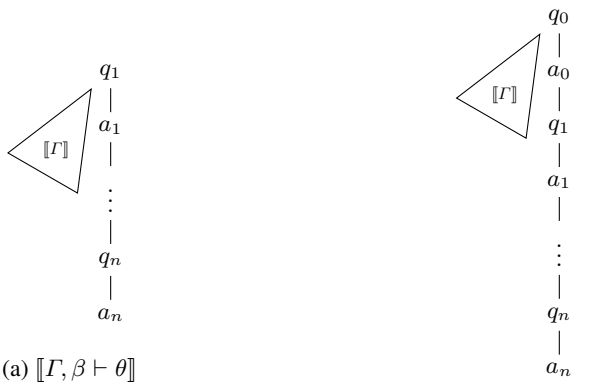
Of course, the languages recognised by nested data automata are closed under automorphisms of the data set, so in fact each play s will be represented by an infinite set of data words, all equivalent to one another by automorphism of the data set.

Theorem 2. *For every typed term* $\Gamma \vdash M : \theta$ *in* $\mathrm{RML}_{2\vdash 1}^{\text{P-Str}}$ *that is in canonical form we can effectively construct a deterministic weak nested data class memory automata,* $\mathcal{A}^M$*, recognising the complete plays of* $L([\![\Gamma \vdash M]\!])$.

Proof. We prove this by induction over the canonical forms. We note that for each canonical form construction, if the construction is in $\mathrm{RML}_{2\vdash 1}^{\text{P-Str}}$ then each constituent canonical form must also be. For convenience of the inductive constructions, we in fact construct automata $\mathcal{A}_\gamma^M$ recognising $[\![\Gamma \vdash M]\!]$ restricted to the initial move γ. Here we sketch two illustrative cases.

$\boldsymbol{\lambda x^{\beta}.M : \beta \to \theta}$**.** The prearenas for $[\![M]\!]$ and $[\![\lambda x^{\beta}.M]\!]$ are shown in Figure 4. Note that in this case we must have that $\Gamma, x : \beta \vdash M : \theta$, and so the initial moves in $[\![M]\!]$ contain an x-component. We therefore write these initial moves as (γ, i_x) where γ is the Γ-component and i_x is the x-component.

P's strategy $[\![\lambda x^{\beta}.M]\!]$ is as follows: after an initial move γ, P plays the unique a_0-move $\bullet$, and waits for a q_1-move. Once O plays a q_1-move i_x, P plays as in $[\![\Gamma, x \vdash M]\!]$ when given an initial move (γ, i_x). However, as the q_1-moves are not initial, it is possible that O will play another q_1-move, i'_x. Each time O does this it opens a new thread which P plays as per $[\![\Gamma, x \vdash M]\!]$ when given initial move (γ, i'_x). Only O may switch



(a) $[\![\Gamma, \beta \vdash \theta]\!]$

(b) $[\![\Gamma \vdash \beta \to \theta]\!]$

Fig. 4: Prearenas for $[\![\Gamma, x : \beta \vdash M : \theta]\!]$ and $[\![\Gamma \vdash \lambda x^{\beta}.M : \beta \to \theta]\!]$

between threads, and this can only happen immediately after P plays an a_j-move (for any j).

By our inductive hypothesis, for each initial move (γ, i_x) of $[\![\Gamma, x : \beta \vdash \theta]\!]$ we have an automaton $\mathcal{A}^{M}_{\gamma,i_x}$ recognising the complete plays of $[\![\Gamma, x : \beta \vdash M : \theta]\!]$ starting with the initial move (γ, i_x). We construct the automaton $\mathcal{A}^{\lambda x.M}_{\gamma}$ by taking a copy of each $\mathcal{A}^{M}_{\gamma,i_x}$, and quotient together the initial states of these automata to one state, p, (which by conditions on the constituent automata we can assume has no incoming transitions). This state p will hold the unique level-0 data value for the run, and states and transitions are added to have initial transitions labelled with q_0 and a_0, ending in state p. The final states will be the new initial state, the quotient state p, and the states which are final in the constituent automata. The transitions inside the constituent automata fall into two categories: those labelled with moves corresponding to the RHS of the term in context $\Gamma \vdash M$, and those labelled with moves corresponding to the LHS. Those transitions corresponding to moves on the RHS are altered to have their level increased by 1, with their signature correspondingly altered by requiring a level-0 data value in state p. Those transitions corresponding to moves on the LHS retain the same level, but have the top value of their data value signature replaced with the state p. Finally, transitions are added between the constituent automata to allow switching between threads: whenever there is a transition out of a final state in one of the automata, copies of the transition are added from every final state (though keeping the data-value signature the same). Note that the final states correspond to precisely the points in the run where the environment is able to switch threads.

$\mathbf{let}\ x^{\beta} = M\ \mathbf{in}\ N$. Here we assume we have automata recognising $[\![M]\!]$ and $[\![N]\!]$. The strategy $[\![\mathbf{let}\, x^{\beta} = M\, \mathbf{in}\, N]\!]$ essentially consists of a concatenation of $[\![M]\!]$ and $[\![N]\!]$, with the result of playing $[\![M]\!]$ determining the value of x to use in $[\![N]\!]$. Hence the automata construction is very similar to the standard finite automata construction for concatenation of languages, though branching on the different results for $[\![M]\!]$ to different automata for $[\![N]\!]$.

Corollary 1. *Observational equivalence of terms in* $\mathrm{RML}^{\text{P-Str}}_{2\vdash 1}$ *is decidable*

4 A Restricted Fragment of $\mathrm{RML}_{2\vdash 1}$

It is important, for the reduction to nested data automata for $\mathrm{RML}_{2\vdash 1}^{\text{P-Str}}$, that variables cannot be partially evaluated: in prearenas where variables have only one argument, once a variable is evaluated those moves cannot be used to justify any future moves. If we could later return to them we would need ensure that they were accessed only in ways which did not break visibility. We now show that this can be done, using a slightly different encoding of pointers, for a fragment in which variables have unlimited arity, but each argument for the variable must be evaluated all at once. This means that the variables have their O-moves uniquely determined by the underlying sequence of moves.

4.1 Fragment definition

Definition 2. *The fragment we consider in this section, which we denote* $\mathrm{RML}_{2\vdash 1}^{\text{res}}$*, consists of typed terms of the form* $x_1 : \Theta_2^1, \ldots, x_n : \Theta_2^1 \vdash M : \Theta_1$ *where the type classes* Θ_i *are as described below:*

$$\Theta_0 ::= \mathsf{unit} \mid \mathsf{int} \qquad \Theta_1^1 ::= \Theta_0 \mid \Theta_0 \to \Theta_0 \mid \mathsf{int\ ref}$$

$$\Theta_1 ::= \Theta_0 \mid \Theta_0 \to \Theta_1 \mid \mathsf{int\ ref} \qquad \Theta_2^1 ::= \Theta_1 \mid \Theta_1^1 \to \Theta_2^1$$

This allows types of the form $(\beta \to \beta) \to \cdots \to (\beta \to \beta) \to \beta \vdash \beta \to \cdots \to \beta$ where $\beta \in \{\mathsf{unit}, \mathsf{int}\}$. The shape of the prearenas for this fragment is shown in Figure 5. Note that moves in section A of the prearena (marked in Figure 5) relate to the type Θ_1 on the RHS of the typing judgement, and that we need only represent O-pointers for this section, since the P-moves are all answers so have their pointers uniquely determined by well-bracketing. Moves in sections B and C of the prearena correspond to the types on the LHS of the typing judgement. Moves in section B need only have their P-pointers represented, since the O-moves are all answer moves. Moves in section C have both their O- and P-pointers represented by the underlying sequence of moves: the P-pointers because all P-moves in this section are answer moves, the O-pointers by the visibility condition.

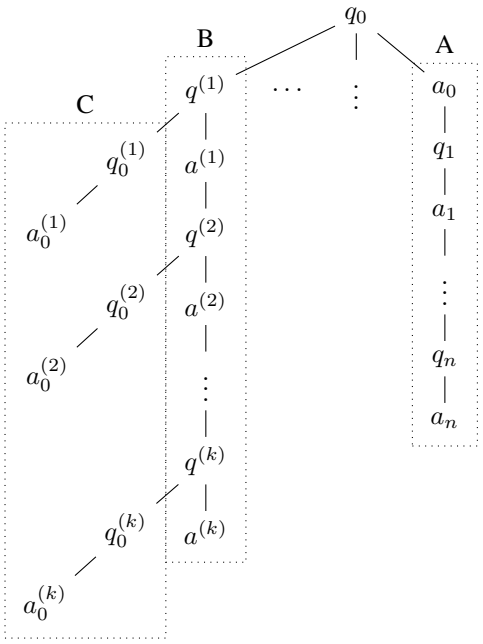


Fig. 5: Shape of arenas in $\mathrm{RML}_{2\vdash 1}^{\text{res}}$

4.2 Deciding Observation Equivalence

Similarly to the P-Strict case, we provide a reduction to weak nested data class memory automata that uses data values to encode O-pointers. However, this time we do not need to represent any O-pointers on the LHS of the typing judgement, so use data values only to represent pointers of the questions on the RHS. We do, though, need to represent P-pointers of moves on the LHS. This we do using the same technique used for representing P-pointers in [9]: in each word in the language we represent only one pointer by using a "tagging" of moves: the string $s \overset{\circ}{m} s' \overset{\bullet}{m'}$ is used to represent the pointer $s \overset{\curvearrowleft}{m} s' m'$. Because P's strategy is deterministic, representing one pointer in each word is enough to uniquely reconstruct all P-pointers in the plays from the entire language. Due to space constraints we do not provide a full explanation of this technique in this paper: for a detailed discussion see [8,9]. Hence for a term $[\![\Gamma \vdash M : \theta]\!]$ the data language we seek to recognise, $L([\![\Gamma \vdash M]\!])$ represents pointers in the following manner:

- The initial move takes the (unique) level-0 data value;
- Moves in $[\![\Gamma]\!]$ (i.e. in section B or C of the prearena) take the data value of the previous move;
- Answer moves in $[\![\theta]\!]$ (i.e. in section A of the prearena) take the data value of the question they are answering; and
- Non-initial question moves in $[\![\theta]\!]$ (i.e. in section A of the prearena) take a fresh data value nested under the data value of the justifying answer move.

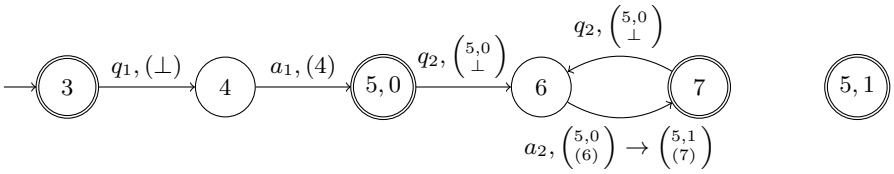
Theorem 3. *For every typed term $\Gamma \vdash M : \theta$ in* $\mathrm{RML}^{\mathrm{res}}_{2\vdash 1}$ *that is in canonical form we can effectively construct a deterministic weak nested data class memory automaton, $\mathcal{A}_M$, recognising the complete plays of $L([\![\Gamma \vdash M]\!])$.*

Proof. This proof takes a similar form to that of Theorem 2: by induction over canonical forms. We here sketch the λ-abstraction case.

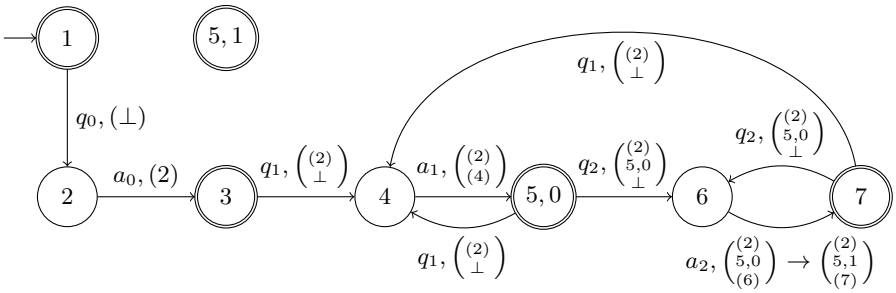
$\boldsymbol{\lambda x^{\beta}.M : \beta \to \theta}$**.** This construction is almost identical to that in the proof of Theorem 2: again the strategy for P is interleavings of P's strategy for $M : \theta$. The only difference in the construction is that where in the encoding for Theorem 2 the moves in each $\mathcal{A}^{M}_{\gamma,i_x}$ corresponding to the LHS and RHS of the prearena needed to be treated separately, in this case they can be treated identically: all being nested under the new level-0 data value. We demonstrate this construction in Example 1

Example 1. Figure 6 shows two weak nested data class memory automata. We draw a transition $p, a, (j, \begin{pmatrix} s_0 \\ \vdots \\ s_j \end{pmatrix}) \to p', \begin{pmatrix} s'_0 \\ \vdots \\ s'_j \end{pmatrix} \in \delta$ as an arrow from state p to p' labelled with "$a, \begin{pmatrix} s_0 \\ \vdots \\ s_j \end{pmatrix} \to \begin{pmatrix} s'_0 \\ \vdots \\ s'_j \end{pmatrix}$". We omit the "$\to \begin{pmatrix} s'_0 \\ \vdots \\ s'_j \end{pmatrix}$" part of the label if $s'_j = p'$ and $s_i = s'_i$ for all $i \in \{0, 1, \ldots, j-1\}$.

The automaton obtained by the constructions in Theorem 3 for the term-in-context $[\![\vdash \mathbf{let}\, c = \mathbf{ref}\, 0\, \mathbf{in}\, \lambda y^{\mathsf{unit}}.\mathbf{if}\, !c = 0\, \mathbf{then}\, c := 1\, \mathbf{else}\, \Omega]\!]$ is shown in Figure 6a (to aid



(a) Automaton for $[\![\vdash \mathbf{let}\, c = \mathbf{ref}\, 0\, \mathbf{in}\, \lambda y^{\mathsf{unit}}.\mathbf{if}\, !c = 0\, \mathbf{then}\, c := 1\, \mathbf{else}\, \Omega]\!]$



(b) Automaton for $[\![\vdash \lambda x^{\mathsf{unit}}.\mathbf{let}\, c = \mathbf{ref}\, 0\, \mathbf{in}\, \lambda y^{\mathsf{unit}}.\mathbf{if}\, !c = 0\, \mathbf{then}\, c := 1\, \mathbf{else}\, \Omega]\!]$

Fig. 6: Automata recognising strategies

readability, we have removed most of the dead and unreachable states and transitions). Note that we have the states $(5, 0)$ and $(5, 1)$ - here the second part of the state label is the value of the variable c: the top-level data value will remain in one of these two states, and by doing so store the value of c at that point in the run. The move q_2 in this example corresponds to the environment providing an argument y: note that in a run of the automaton the first time a y argument is passed, the automaton proceeds to reach an accepting state, but in doing so sets the top level data value to the state $(5, 1)$. This means the outgoing transition shown from state 7 cannot fire.

The automaton for $[\![\vdash \lambda x^{\mathsf{unit}}.\mathbf{let}\, c = \mathbf{ref}\, 0\, \mathbf{in}\, \lambda y^{\mathsf{unit}}.\mathbf{if}\, !c = 0\, \mathbf{then}\, c := 1\, \mathbf{else}\, \Omega]\!]$ is shown in Figure 6b (again, cleaned of dead/unreachable transitions for clarity). Note that this contains the first automaton as a sub-automaton, though with a new top-level data value added to the transitions. The q_1 move now corresponds to providing a new argument for x, thus starting a thread. Transitions have been added from the accepting states (5) and (7), allowing a new x-thread to be started from either of these locations. Note that the transition from (7) to (6), which could not fire before, now can fire because several data values (corresponding to different x-threads) can be generated and left in the state $(5, 0)$.

5 Undecidable Fragments

In this section we consider which type sequents and forms of recursion are expressive enough to prove undecidability. The proofs of the results this section proceed by identifying terms such that the induced complete plays correspond to runs of Turing-complete machine models.

On the Right of the Turnstile. In [11] it is shown that observational equivalence is undecidable for 5th-order terms. The proof takes the strategy that was used to show undecidability for 4th-order IA and finds an equivalent call-by-value strategy. It is relatively straightforward to adapt the proof to show that observational equivalence is undecidable at 3rd-order types, e.g. $((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit}$. A further result in [12] showed that the problem is undecidable at the type $(\mathsf{unit} \to \mathsf{unit}) \to (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}$. Both results easily generalise to show that the problem is undecidable at *every* 3rd-order type and *every* 2nd-order type which takes at least two 1st-order arguments. We modify the second of these proofs to show undecidability at $(\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit} \to \mathsf{unit}$. Our proof of this easily adapts to a proof of the following.

Theorem 4. *Observational equivalence is undecidable at every 2nd-order type (of arity at least two) which contains a 1st-order argument that is not the final argument.*

On the Left of the Turnstile. Note that $\vdash M \cong N : \theta$ if, and only if, $f : \theta \to \mathsf{unit} \vdash fM \cong fN : \mathsf{unit}$. Thus, for any sequent $\vdash \theta$ at which observational equivalence is undecidable, the sequent $\theta \to \mathsf{unit} \vdash \mathsf{unit}$ is also undecidable. So the problem is undecidable if, on the left of the turnstile, we have a fourth-order type or a (third-order) type which has a second-order argument whose first-order argument is not the last.

Recursion. In IA, observational equivalence becomes undecidable if we add recursive first-order functions [16]. The analogous results for RML with recursion also hold:

Theorem 5. *Observational equivalence is undecidable in* $\mathrm{RML}_{\text{O-Str}}$ *equipped with recursive functions* $(\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}$

6 Conclusion

We have used two related encodings of pointers to data values to decide two related fragments of $\mathrm{RML}_{2\vdash 1}$: $\mathrm{RML}^{\text{P-Str}}_{2\vdash 1}$, in which the free variables were limited to arity 1, and $\mathrm{RML}^{\mathrm{res}}_{2\vdash 1}$, in which the free variables were unlimited in arity but each argument of the free variable was limited to arity 1. It is natural to ask whether we can extend or combine these approaches to decide the whole of $\mathrm{RML}_{2\vdash 1}$. Here we discuss why this seems likely to be impossible with the current machinery used.

In deciding $\mathrm{RML}^{\text{P-Str}}_{2\vdash 1}$ we used the nested data value tree-structure to mirror the shape of the prearenas. These data values can be seen as names for different threads, with the sub-thread relation captured by the nested structure. What happens if we attempt to use this approach to recognise strategies on types where the free variables have arity greater than 1? With free variables having arity 1, whenever they are interrogated by P, they are entirely evaluated immediately: they cannot be partially evaluated. With arity greater than 1, this partial evaluation can happen: P may provide the first argument at some stage, and then at later points evaluate the variable possibly several times with different second arguments. P will only do this subject to visibility conditions though: if P partially evaluates a variable x while in a thread T, it can only continue that partial evaluation of x in T or a sub-thread of T. This leads to problems when our automata recognise interleavings of similar threads using the same part of the automaton. If P's

strategy for the thread T is the strategy $[\![M]\!]$ for a term M, and recognised by an automaton $\mathcal{A}^M$, then $[\![\lambda y.M]\!]$ will consist of interleavings of $[\![M]\!]$. The automaton $\mathcal{A}^{\lambda y.M}$ will use a copy of $\mathcal{A}^M$ to simulate an unbounded number of M-threads. If T is one such thread, which performs a partial evaluation of x, this partial evaluation will be represented by input letters with data values unrelated to the data value of T. If a sibling of T, T', does the same, the internal state of the automaton will have no way of telling which of these partial evaluations was performed by T and which by T'. Hence it may recognise data words which represent plays that break the visibility condition.

Therefore, to recognise strategies for terms with free variables of arity greater than 1, the natural approach to take is to have the data value of free-variable moves be related to the thread we are in. This is the approach we took in deciding $\mathrm{RML}^{\mathrm{res}}_{2\vdash 1}$: the free variable moves precisely took the data value of the part of the thread they were in. Then information about the partial evaluation was stored by the thread's data value. This worked when the arguments to the free variables had arity at most 1: however if we allow the arity of this to increase we need to start representing O-pointers in the evaluation of these arguments. For this to be done in a way that makes an inductive construction work for $\mathbf{let}\, x = (\lambda y.M)\, \mathbf{in}\, N$, we must use some kind of nesting of data values for the different M-threads. The naïve approach to take is to allow the M-thread data values to be nested under the data value of whatever part of the N-thread they are in. However, the M-thread may be started and partially evaluated in one part of the N-thread, and then picked up and continued in a descendant part of that N-thread. The data values used in continuing the M-thread must therefore be related to the data values used to represent the partial evaluation of the M-thread, but also to the part of the N-thread the play is currently in. This would break the tree-structure of the data values, and so seem to require a richer structure on the data values.

Further Work. A natural direction for further work, therefore, is to investigate richer data structures and automata models over them that may provide a way to decide $\mathrm{RML}_{2\vdash 1}$.

The automata we used have a non-primitive recursive emptiness problem, and hence the resulting algorithms both have non-primitive recursive complexity also. Although work in [8] shows that this is not the best possible result in the simplest cases, the exact complexities of the observational equivalence problems are still unknown.

To complete the classification of $\mathrm{RML}_{\mathrm{f}}$ also requires deciding (or showing undecidable) the fragment containing order 2 types (on the RHS) with one order 1 argument, which is the last argument. A first step to deciding this would be the fragment labelled RML_X in figure 1. Deciding this fragment via automata reductions similar to those in this paper would seem to require both data values to represent O-pointers, and some kind of visible stack to nest copies of the body of the function, as used in [9]. In particular, recognising strategies of second-order terms such as $\lambda f.f()$ requires the ability to recognise data languages (roughly) of the form $\{d_1 d_2 ... d_n d_n ... d_2 d_1 \mid n \in N, \text{each } d_i \text{ is distinct}\}$. A simple pumping argument shows such languages cannot be recognised by nested data class memory automata, and so some kind of additional stack would seem to be required.

References

1. Samson Abramsky and Guy McCusker. Linearity, sharing and state: a fully abstract game semantics for idealized algol with active expressions. *Electr. Notes Theor. Comput. Sci.*, 3:2–14, 1996.
2. Samson Abramsky and Guy McCusker. Call-by-value games. In Mogens Nielsen and Wolfgang Thomas, editors, *CSL 1997*, volume 1414 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 1997.
3. Henrik Björklund and Thomas Schwentick. On notions of regularity for data languages. *Theor. Comput. Sci.*, 411(4-5):702–715, 2010.
4. Conrad Cotton-Barratt, Andrzej S. Murawski, and C.-H. Luke Ong. Weak and nested class memory automata. *Proceedings of LATA 2015*, to appear, 2015.
5. Normann Decker, Peter Habermehl, Martin Leucker, and Daniel Thoma. Ordered navigation on multi-attributed data words. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014*, volume 8704 of *Lecture Notes in Computer Science*, pages 497–511. Springer, 2014.
6. Dan R. Ghica and Guy McCusker. The regular-language semantics of second-order idealized algol. *Theor. Comput. Sci.*, 309(1-3):469–502, 2003.
7. Kohei Honda and Nobuko Yoshida. Game-theoretic analysis of call-by-value computation. *Theor. Comput. Sci.*, 221(1-2):393–456, 1999.
8. David Hopkins. *Game Semantics Based Equivalence Checking of Higher-Order Programs*. PhD thesis, Department of Computer Science, University of Oxford, 2012.
9. David Hopkins, Andrzej S. Murawski, and C.-H. Luke Ong. A fragment of ML decidable by visibly pushdown automata. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP 2011*, volume 6756 of *Lecture Notes in Computer Science*, pages 149–161. Springer, 2011.
10. J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: i, ii, and III. *Inf. Comput.*, 163(2):285–408, 2000.
11. Andrzej S. Murawski. On program equivalence in languages with ground-type references. In *18th IEEE Symposium on Logic in Computer Science (LICS 2003)*, page 108. IEEE Computer Society, 2003.
12. Andrzej S. Murawski. Functions with local state: Regularity and undecidability. *Theor. Comput. Sci.*, 338(1-3):315–349, 2005.
13. Andrzej S. Murawski, C.-H. Luke Ong, and Igor Walukiewicz. Idealized algol with ground recursion, and DPDA equivalence. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP 2005*, volume 3580 of *Lecture Notes in Computer Science*, pages 917–929. Springer, 2005.
14. Andrzej S. Murawski and Nikos Tzevelekos. Algorithmic nominal game semantics. In Gilles Barthe, editor, *20th European Symposium on Programming, ESOP 2011*, volume 6602 of *Lecture Notes in Computer Science*, pages 419–438. Springer, 2011.
15. Andrzej S. Murawski and Nikos Tzevelekos. Algorithmic games for full ground references. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *ICALP 2012*, volume 7392 of *Lecture Notes in Computer Science*, pages 312–324. Springer, 2012.
16. C.-H. Luke Ong. An approach to deciding the observational equivalence of algol-like languages. *Ann. Pure Appl. Logic*, 130(1-3):125–171, 2004.
17. Andrew M. Pitts and Ian D. B. Stark. Operational reasoning for functions with local state. *Higher order operational techniques in semantics*, pages 227–273, 1998.