Algorithmic nominal game semantics

A. S. Murawski* and N. Tzevelekos**

[1] Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, UK
[2] Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD, UK

Abstract. We employ automata over infinite alphabets to capture the semantics of a finitary fragment of ML with ground-type references. Our approach is founded on game semantics, which allows us to translate programs into automata in such a way that contextual equivalence is characterized by a finitary notion of bisimilarity. As a corollary, we derive a decidability result for a class of first-order programs, including open ones that contain unspecified first-order procedures.

1 Introduction

Recent years have seen a surge of interest in automata-theoretic models over infinite alphabets. It stemmed from the realization that finite automata, while immensely successful, do not lend satisfactory representations of a variety of interesting phenomena. In program verification, for instance, one might want to consider the interaction of unboundedly many agents, each of which issues requests that have to be traceable. In database theory, in turn, integrity constraints are often expressed in terms of data *values* possibly drawn from an infinite set (as opposed to data labels, which come from a finite one). Since giving automata too much power in manipulating values from an infinite domain quickly results in undecidability, decidable models over infinite alphabets have to be restricted so that the values can only be tested for equality. A number of such formalisms have been proposed in recent years: register automata [7], pebble automata [14] and data automata [3], to name a few.

The general goal of our paper is to draw techniques from these developments, adapt them to use in programming language semantics so that, ultimately, they can be applied to program verification: in our case, to automated equivalence checking.

Our results will concern Reduced ML [16], which is a subset of ML with ground-type references only (neither higher-order functions nor reference names can be stored in memory). This is a simple fundamental language that combines functional and imperative programming in a minimal fashion and in the style of ML. Despite its simplicity, it gives rise to a subtle theory of contextual program

* Supported by an EPSRC Advanced Research Fellowship (EP/C539753/1).
** Supported by EPSRC (EP/F067607/1).

equivalence, which comprises elements of secrecy, freshness, locality and object identity. Here are several sample (in)equivalences that can be (dis)proved in an automated fashion using our results.

– Dynamically generated reference names are private[3].

$$\vdash \mathsf{let}\, n = \mathsf{ref}(0)\, \mathsf{in}\ \lambda x^{\mathsf{int\,ref}}.(x =_{\mathsf{int\,ref}} n) \quad \cong \quad \lambda x^{\mathsf{int\,ref}}.\mathit{false} : \mathsf{int\,ref} \to \mathsf{bool}$$

– Intermediate states of computation are invisible.

$$x : \mathsf{int\,ref} \vdash x := 0; x := 1 \quad \cong \quad x := 1 : \mathsf{unit}$$

– Local declarations and function abstraction do not commute in general.

$$\vdash \lambda x^{\mathsf{unit}}.\mathsf{let}\, n = \mathsf{ref}(0)\, \mathsf{in}\, n \quad \ncong \quad \mathsf{let}\, n = \mathsf{ref}(0)\, \mathsf{in}\, (\lambda x^{\mathsf{unit}}.n) : \mathsf{unit} \to \mathsf{int\,ref}$$

– But they do sometimes, at the cost of explicit initialization.

$$\begin{array}{l} f : \mathsf{int\,ref} \to \mathsf{int} \vdash \ \lambda x^{\mathsf{unit}}.\mathsf{let}\, n = \mathsf{ref}(0)\, \mathsf{in}\, f(n) \\ \qquad\qquad\qquad \cong \quad \mathsf{let}\, n = \mathsf{ref}(0)\, \mathsf{in}\, \lambda x^{\mathsf{unit}}.n := 0; f(n) : \mathsf{unit} \to \mathsf{int} \end{array}$$

– In a similar fashion local variables can be globalized.

$$\begin{array}{l} \mathit{guard} : \mathsf{unit} \to \mathsf{int},\ \ \mathit{body} : \mathsf{int\,ref} \to \mathsf{unit} \vdash \\ \qquad \mathsf{while}\ \mathit{guard}()\ \mathsf{do}\ (\mathsf{let}\, n = \mathsf{ref}(0)\, \mathsf{in}\, \mathit{body}(n)) \\ \qquad \cong \quad \mathsf{let}\, n = \mathsf{ref}(0)\, \mathsf{in}\, (\mathsf{while}\ \mathit{guard}()\ \mathsf{do}\ (n := 0; \mathit{body}(n))) : \mathsf{unit} \end{array}$$

– Not all differences between names can be picked up by the environment, as reference names are not storable.

$$\begin{array}{l} f : \mathsf{int\,ref} \to \mathsf{unit} \vdash \ \mathsf{let}\, n_1 = \mathsf{ref}(0)\, \mathsf{in}\, \mathsf{let}\, n_2 = \mathsf{ref}\ (0)\, \mathsf{in}\, f(n_1); (n_2 := !n_1); n_2 \\ \qquad\qquad\qquad \cong \quad \mathsf{let}\, n = \mathsf{ref}\ 0\, \mathsf{in}\, f(n); n : \mathsf{int\,ref} \end{array}$$

In order to derive decidability results for program equivalence, we shall consider a finitary fragment of Reduced ML: with finite datatypes, no recursion and restricted higher-order types. To be exact, our approach will be applicable to terms $\Gamma \vdash M : \theta$, where θ as well as the type of each identifier in Γ is of the form β or $\beta \to \beta$, and β stands for any base type (unit, int or $\mathsf{int\,ref}$), like in all of the examples given above.

To enable a systematic and computer-aided verification of equivalence between such terms, we translate them into a special class of automata over infinite alphabets in such a way that languages accepted by the automata will be faithful representations of their fully abstract game semantics [12].

Game semantics views interaction as an exchange of moves (play) between two players representing the environment (Opponent) and the program (Proponent) respectively. In our particular game model the moves will contain (reference) names drawn from a countable set of locations. Each move will also be equipped with a carefully selected fragment of the current heap, represented as

[3] $x =_{\mathsf{int\,ref}} r$ denotes reference equality test.

a set of (*name*,*value*) pairs. The involvement of an infinite set of names makes it very natural to view such plays as words over an infinite alphabet.

The automata we employ are deterministic variants of fresh-register automata [17], which themselves build upon register automata [7]: each automaton will be equipped with a finite set of registers in which names can be stored for future reference. The automata are designed in such a way that, in the spirit of register automata, they can classify each name coming from the environment as known (currently stored in one of the registers) or as *locally* fresh – not present in current memory. On the other hand, the names that a program can send to its environment will be either those already in its memory, or *globally* fresh ones, i.e. names that have not been encountered thus far, but can be obtained on demand by invoking a fresh-name generator, in the style of fresh-register automata. We therefore see that local freshness is inherently a property of Opponent, while global freshness is specific to Proponent.

Our decision procedure comprises three stages.

- First we construct automata that represent term behaviour (in the sense of game semantics) under the liberal assumption that the environment is capable of distinguishing all names created by the program and modifying the corresponding values at any time.
- Subsequently we refine the automata so that they capture exactly the interactions with contexts of Reduced ML.
- Finally, we introduce a finite notion of bisimilarity on the automata to ascertain that they represent equivalent interactions. Because the underlying game model is fully abstract, this relates contextual equivalence to decidable bisimilarity.

On the whole, our work combines automata-theoretic and semantic insights to develop a new verification routine.

Related work. We make a notable step towards a full classification of decidable fragments of Reduced ML. Our results apply to reference types, while all earlier work [6, 10, 13] on that topic was based on the game model of RML [1], a variant of Reduced ML with "bad variables" (terms of type int ref with designated methods for reading and writing). Consequently, when reference types were present in a typing judgment the induced notion of equivalence was strictly stronger than in Reduced ML proper. For example, $x := !x$ and $()$ could be distinguished by a bad variable that crashes on dereferencing. Another drawback of RML was that reference equality could not be studied, as it did not make sense. RML had a definite advantage though, as the associated game model was based on a finite set of moves. Equivalence in Reduced ML turns out much more subtle and the corresponding fully abstract game model [12] is unsuited to finite-alphabet representations. It so happens that the presence of bad variables does not change the induced observational equivalence in the call-by-name case of Idealized Algol [8], where a complete map of decidable fragments already exists [11] and increases in complexity (of deciding equivalence) are tightly linked to type-theoretic order.

$$\theta ::= \quad \mathsf{unit} \quad | \quad \mathsf{int} \quad | \quad \mathsf{int\,ref} \quad | \quad \theta \to \theta$$

$$\frac{}{\Gamma \vdash () : \mathsf{unit}} \qquad \frac{i \in \{0, \cdots, max\}}{\Gamma \vdash i : \mathsf{int}} \qquad \frac{\Gamma \vdash M : \theta}{\pi(\Gamma) \vdash M : \theta} \quad \pi \in Perm(\Gamma)$$

$$\frac{\Gamma \vdash M : \mathsf{int} \qquad \Gamma \vdash M_0 : \theta \qquad \cdots \qquad \Gamma \vdash M_{max} : \theta}{\Gamma \vdash \mathsf{case}(M)[M_0, \cdots, M_{max}] : \theta} \qquad \frac{\Gamma \vdash M : \mathsf{int} \quad \Gamma \vdash N : \mathsf{unit}}{\Gamma \vdash \mathsf{while}\, M\, \mathsf{do}\, N : \mathsf{unit}}$$

$$\frac{\Gamma \vdash M : \mathsf{int\,ref}}{\Gamma \vdash !M : \mathsf{int}} \qquad \frac{\Gamma \vdash M : \mathsf{int\,ref} \quad \Gamma \vdash N : \mathsf{int}}{\Gamma \vdash M := N : \mathsf{unit}} \qquad \frac{\Gamma \vdash M : \mathsf{int}}{\Gamma \vdash \mathsf{ref}\, M : \mathsf{int\,ref}}$$

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

Fig. 1. Syntax of $\mathsf{RedML}_{\mathrm{fin}}$.

Contextual equivalence in ML-like languages, also those richer than Reduced ML, has also been studied extensively using relational techniques [15, 2, 4], albeit without decidability results.

2 Finitary Reduced ML

Finitary Reduced ML ($\mathsf{RedML}_{\mathrm{fin}}$) is the (call-by-value) λ-calculus over the ground types $\mathsf{unit}, \mathsf{int}, \mathsf{int\,ref}$ augmented with (*finitely* many) integer constants $0, \cdots, max$, branching, looping and reference manipulation. Its typing rules are given in Figure 1. Note that we have not included reference equality testing, as it is expressible [15] (assuming $max > 0$). For instance, one can define $\mathsf{eq}_{\mathsf{int\,ref}} : \mathsf{int\,ref} \to \mathsf{int\,ref} \to \mathsf{int}$ to be

$$\begin{array}{l}\lambda x^{\mathsf{int\,ref}}.\lambda y^{\mathsf{int\,ref}}.\ \mathsf{let}\, v_x = \mathsf{ref}\, !x\, \mathsf{in} \\ \qquad\qquad \mathsf{let}\, v_y = \mathsf{ref}\, !y\, \mathsf{in} \\ \qquad\qquad \mathsf{let}\, b = \mathsf{ref}\, 0\, \mathsf{in} \\ \qquad\qquad\quad (x := 0; y := 1; (\mathsf{if}\, !x = 1\, \mathsf{then}\, b := 1\, \mathsf{else}\, ()); x := !v_x; y := !v_y; !b).\end{array}$$

In the above and in what follows, we write

- $\mathsf{let}\, x = M\, \mathsf{in}\, N$ for the term $(\lambda x^{\theta}.N)M$;
- $M; N$ for the term $(\lambda x^{\theta}.N)M$, if x is not free in N;
- $\mathsf{if}\, M\, \mathsf{then}\, M_1\, \mathsf{else}\, M_0$ for $\mathsf{case}(M)[M_0, M_1, \cdots, M_1]$;
- and $M =_{\mathsf{int\,ref}} N$ for $\mathsf{eq}_{\mathsf{int\,ref}}\, M\, N$.

We refer the reader to [16] for a detailed exposition of the operational semantics.

Definition 1. *Two terms-in-context $\Gamma \vdash M : \theta$ and $\Gamma \vdash N : \theta$ are contextually* equivalent *if, and only if, for any $\mathsf{RedML}_{\mathrm{fin}}$-context $C[-]$ such that $\vdash C[M], C[N] : \mathsf{unit}$, $C[M]$ evaluates to $()$ iff $C[N]$ does. Then we write $\Gamma \vdash M \cong N : \theta$.*

In this paper we show that contextual equivalence is decidable for a fragment of $\mathsf{RedML}_{\mathrm{fin}}$, called $\mathsf{RedML}_{\mathrm{fin}}^{\beta\to\beta}$, to be defined next.

Definition 2. *Suppose $\Gamma = [x_1 : \theta_1, \cdots, x_m : \theta_m]$. The term-in-context $\Gamma \vdash M : \theta$ belongs to $\mathsf{RedML}_{\mathrm{fin}}^{\beta\to\beta}$ provided each of $\theta_1, \cdots, \theta_m, \theta$ is generated by the grammar*

$$\theta ::= \quad \beta \quad | \quad \beta \to \beta$$

in which β stands for any base type (unit, int *or* $\mathsf{int\,ref}$*).*

In Section 4 we shall define a class of automata over infinite alphabets to which terms of $\mathsf{RedML}_{\mathrm{fin}}^{\beta\to\beta}$ will be translated in Sections 5 and 6. In order to make the translation more concise, we are going to focus on translating terms in canonical form only. The canonical shapes are defined as follows.

$$\begin{array}{l}\mathbb{C} ::= () \mid i \mid x^{\mathsf{int\,ref}} \mid \mathsf{case}(x^{\mathsf{int}})[\mathbb{C}, \cdots, \mathbb{C}] \mid (\mathsf{while}\,(!x^{\mathsf{int\,ref}})\,\mathsf{do}\,\mathbb{C}); \mathbb{C} \mid \\ \quad (x^{\mathsf{int\,ref}} := i); \mathbb{C} \mid \mathsf{let}\, y^{\mathsf{int}} = !x^{\mathsf{int\,ref}}\,\mathsf{in}\,\mathbb{C} \mid \mathsf{let}\, x^{\mathsf{int\,ref}} = \mathsf{ref}\,(0)\,\mathsf{in}\,\mathbb{C} \mid \\ \quad \lambda x^{\beta}.\mathbb{C} \mid \mathsf{let}\, y^{\beta} = z()\,\mathsf{in}\,\mathbb{C} \mid \mathsf{let}\, y^{\beta} = z\, i\,\mathsf{in}\,\mathbb{C} \mid \mathsf{let}\, y^{\beta} = z\, x^{\mathsf{int\,ref}}\,\mathsf{in}\,\mathbb{C}\end{array}$$

Lemma 3. *Let $\Gamma \vdash M : \theta$ be an $\mathsf{RedML}_{\mathrm{fin}}^{\beta\to\beta}$-term. There exists a $\mathsf{RedML}_{\mathrm{fin}}^{\beta\to\beta}$-term $\Gamma \vdash \mathbb{C}_M : \theta$ in canonical form, effectively constructible from M, such that $\Gamma \vdash M \cong \mathbb{C}_M$.*

Proof. $\mathbb{C}_M$ can be obtained via a series of η-expansions, β-reductions and commuting conversions involving let and case. □

3 Game Semantics

In this section we briefly recapitulate the game semantics of Reduced ML [12], insofar as it concerns modelling $\mathsf{RedML}_{\mathrm{fin}}^{\beta\to\beta}$. We present it in a more concrete way, specialized to the types of $\mathsf{RedML}_{\mathrm{fin}}^{\beta\to\beta}$, along with examples that motivate the respective technical conditions.

Game semantics views computation as a dialogue between the environment (Opponent, O) and the program (Proponent, P). The game model we are going to sketch falls into the realm of *nominal game semantics*: moves may involve *names* drawn from an infinite set $\mathbb{A}$. Consequently, we can apply name-permutations to moves, plays and strategies. Put otherwise, they form *nominal sets* [5]. We begin with some auxiliary definitions before specifying what it means to play our games.

Definition 4.
- *For every type θ let us define the associated set of labels $\mathcal{L}_\theta$ as follows: $\mathcal{L}_{\mathsf{unit}} = \{\star\}$, $\mathcal{L}_{\mathsf{int}} = \{0, \cdots, max\}$, $\mathcal{L}_{\mathsf{int\,ref}} = \mathbb{A}$, $\mathcal{L}_{\beta\to\beta'} = \{\star\}$. We shall write $\mathcal{L}$ for the set of all labels.*
- *Given a $\mathsf{RedML}_{\mathrm{fin}}^{\beta\to\beta}$ typing judgment $\Gamma \vdash M : \theta$ we write $\mathcal{T}_{\Gamma\vdash\theta}$ for the set of associated* tags*, defined to be*

$$\{\mathsf{c}_x, \mathsf{r}_x \,|\, (x : \theta_x) \in \Gamma,\, \theta_x \not\equiv \beta\} \cup \{\mathsf{r}_\downarrow\} \cup \{\mathsf{c}, \mathsf{r} \,|\, \theta \not\equiv \beta\}$$

.

Thus, for each function-type identifier x in Γ, we have introduced tags c_x and r_x. They can be viewed as calls and returns related to that identifier. Similarly, $\mathsf{r}_\downarrow$ can be taken to correspond to the fact that M was successfully evaluated, and, if θ is a function type, c and r refer respectively to calling the corresponding value and obtaining a result.

Given $\Gamma \vdash M : \theta$, $\ell \in \mathcal{L}$ and $t \in \mathcal{T}_{\Gamma\vdash\theta}$, we shall say that the pair (ℓ, t) is *consistent* if the following conditions are satisfied.

- If $t = \mathsf{r}_\downarrow$ then $\ell \in \mathcal{L}_\theta$.
- If $t = \mathsf{c}_x$ then $\theta_x \equiv \beta \to \beta'$ and $\ell \in \mathcal{L}_\beta$.
- If $t = \mathsf{r}_x$ then $\theta_x \equiv \beta \to \beta'$ and $\ell \in \mathcal{L}_{\beta'}$.
- If $t = \mathsf{c}$ then $\theta \equiv \beta \to \beta'$ and $\ell \in \mathcal{L}_\beta$.
- If $t = \mathsf{r}$ then $\theta \equiv \beta \to \beta'$ and $\ell \in \mathcal{L}_{\beta'}$.

Suppose $\Gamma = [x_1 : \theta_1, \cdots, x_m : \theta_m]$. The set of *initial moves* I_Γ is defined to be $\{(\ell_1, \cdots, \ell_m) \mid \ell_i \in \mathcal{L}_{\theta_i},\ 1 \le i \le m\}$.

Definition 5. *A* play[4] *over $\Gamma \vdash \theta$ is a (possibly empty) sequence of the form $\iota(\ell_1, t_1)\cdots(\ell_k, t_k)$ such that $\iota \in I_\Gamma$, all pairs (ℓ_i, t_i) are consistent and $t_1 \cdots t_k$ is a prefix of a word matching $X\mathsf{r}_\downarrow(\mathsf{c}X\mathsf{r})^*$, where $X = (\sum_{\substack{(x:\theta_x)\in\Gamma \\ \theta_x \not\equiv \beta}} (\mathsf{c}_x\, \mathsf{r}_x))^*$. We assume that $X\mathsf{r}_\downarrow(\mathsf{c}X\mathsf{r})^*$ degenerates to $X\mathsf{r}_\downarrow$ if c, r are not available (i.e. θ is a base type). A play is* complete *whenever $t_1 \cdots t_k$ matches a word from $X\mathsf{r}_\downarrow(\mathsf{c}X\mathsf{r})^*$.*

The shape of plays can be thought of as a record of computation. First, calls are being made to the free identifiers of function type (expression X), then a value is reached ($\mathsf{r}_\downarrow$) and, if the type of the value is a function type, we have a series of calls and returns with external calls in-between ($(\mathsf{c}X\mathsf{r})^*$).

We shall refer to ι and (ℓ_i, t_i) as *moves*. Moves are assigned ownership as follows: ι and those with tags r_x, c belong to O (environment) and the rest (tags $\mathsf{c}_x, \mathsf{r}_\downarrow, \mathsf{r}$) belong to P (program). We shall write o and p to range over O- and P-moves respectively. We shall say that $\ell \in \mathbb{A}$ is an O-name (resp. P-name) in a given play s, provided the first occurrence of ℓ was in an O-move (resp. P-move). The set of O- and P-names in s will be denoted by $O(s)$ and $P(s)$ respectively.

The fully abstract game model of Reduced ML [12] is based on a more complicated notion of plays, in which each move can contain a store. Since we are considering a language with ground-type references only, i.e. references names themselves cannot be stored, programs will not be able, in general, to remember all names obtained from the environment. Accordingly, we shall not insist that the stores contain values for all names introduced by O, but only those that are potentially available to the program. Intuitively, these are environment names that the program has managed to bind. The notion of P-view helps to capture this concept.

[4] Readers familiar with game semantics will notice that we omit justification pointers in plays. This is because they are uniquely recoverable with the help of tags.

Definition 6. *Given a play s, we define its P-view $\ulcorner s \urcorner$ as follows.*

$$\begin{array}{rl} \ulcorner \iota \urcorner = \iota & \\ \ulcorner s\,(\ell_{\mathsf{r}}, \mathsf{r}_x) \urcorner = \ulcorner s \urcorner\,(\ell_{\mathsf{r}}, \mathsf{r}_x) & \\ \ulcorner s\,(\ell_{\mathsf{c}}, \mathsf{c}) \urcorner = \ulcorner s' \urcorner\,(\ell_{\mathsf{r}_\downarrow}, \mathsf{r}_\downarrow)\,(\ell_{\mathsf{c}}, \mathsf{c}) & s = s'(\ell_{\mathsf{r}_\downarrow}, \mathsf{r}_\downarrow)s'' \\ \ulcorner s\,p \urcorner = \ulcorner s \urcorner\,p & \end{array}$$

It can be checked that the P-view of a play is also a play. Given a play s, the set $\mathsf{Av}_{\mathrm{P}}(s)$ of P-available names is defined as $P(s) \cup O(\ulcorner s \urcorner)$. We can now define a new notion of play, in which players can play moves equipped with stores (moves-with-store, for short).

Definition 7. *A* play-with-store *over $\Gamma \vdash \theta$ is a sequence $m_1^{\Sigma_1} \cdots m_k^{\Sigma_k}$ of moves-with-store satisfying the conditions below.*

- *$m_1 \cdots m_k$ is a play over $\Gamma \vdash \theta$.*
- *For any P-move $m_{2i} = (\ell_{2i}, t_{2i})$, if $\ell_{2i} \in O(m_1 \cdots m_{2i})$ then $\ell_{2i} \in \ulcorner m_1 \cdots m_{2i-1} \urcorner$.*
- *For any $1 \leq i \leq k$, Σ_i is a partial function from $\mathbb{A}$ to $\{0, \cdots, max\}$ such that $\mathsf{dom}(\Sigma_i) = \mathsf{Av}_{\mathrm{P}}(m_1 \cdots m_i)$.*

Using the richer notion of play we define strategies.

Definition 8. *A* strategy *σ is a non-empty, even-prefix closed set of plays-with-store closed under name-permutation. Given a play s, let us write $[s]$ for its equivalence class with respect to name-permutation. A* deterministic *strategy is also required to satisfy the following condition: whenever $s_1 o_1^{\Sigma_1} p_1^{\Sigma_1'}, s_2 o_2^{\Sigma_2} p_2^{\Sigma_2'} \in \sigma$, $[s_1 o_1^{\Sigma_1}] = [s_2 o_2^{\Sigma_2}]$, then $[s_1 o_1^{\Sigma_1} p_1^{\Sigma_1'}] = [s_2 o_2^{\Sigma_2} p_2^{\Sigma_2'}]$.*

We have shown how to assign deterministic strategies to programs of Reduced ML in [12]. Let us write $[\![\Gamma \vdash M : \theta]\!]_0$ for the deterministic strategy corresponding to the $\mathsf{RedML}_{\mathrm{fin}}^{\beta\to\beta}$-term $\Gamma \vdash M : \theta$.

Example 9. – $[\![\vdash \lambda x^{\mathsf{unit}}.\mathsf{let}\, n = \mathsf{ref}(0)\,\mathsf{in}\, n : \mathsf{unit} \to \mathsf{int\,ref}]\!]_0$ consists of plays of the following shape

$$\begin{array}{ccccccccccccc} \iota^{\emptyset} & (\star, \mathsf{r}_\downarrow)^{\emptyset} & (\star, \mathsf{c})^{\Sigma_0'} & (n_1, \mathsf{r})^{\Sigma_1} & (\star, \mathsf{c})^{\Sigma_1'} & (n_2, \mathsf{r})^{\Sigma_2} & \cdots & (\star, \mathsf{c})^{\Sigma_{i-1}'} & (n_i, \mathsf{r})^{\Sigma_i} & (\star, \mathsf{c})^{\Sigma_i'} & \cdots \\ O & P & O & P & O & P & & O & P & O & \end{array}$$

where $\Sigma_0' = \emptyset$ and, for all $i > 0$, $\Sigma_i = \Sigma_{i-1}' \cup \{(n_i, 0)\}$, $\mathsf{dom}(\Sigma_i') = \mathsf{dom}(\Sigma_i)$. Moreover, for any $i \neq j$ we have $n_i \neq n_j$. Note that Σ_i' can be different from Σ_i, i.e. the environment is free to change the values stored at all of the locations that have been revealed to it. Here the stores keep on growing as the names are being generated by the program.

– $[\![\vdash \lambda x^{\mathsf{int\,ref}}.\mathsf{case}(!x)[1, 0, 0, \cdots, 0] : \mathsf{int\,ref} \to \mathsf{int}]\!]_0$ generates, among others, the play

$$\iota^{\emptyset}\ (\star, \mathsf{r}_\downarrow)^{\emptyset}\ (n, \mathsf{c})^{(n,2)}\ (0, \mathsf{r})^{(n,2)}\ (n', \mathsf{c})^{(n',0)}\ (1, \mathsf{r})^{(n',0)}\ (n, \mathsf{c})^{(n,0)}\ (1, \mathsf{r})^{(n,0)}.$$

Note that in the play above the store does not grow as the names are being played by the environment and they disappear from P-view after each call.

By comparing strategies corresponding to terms we cannot yet prove all equivalences. This is because strategies do not take into account the fact that the environment (O) must also be subjected to restrictions concerning recognizability and visibility of names.

Example 10. The following equivalences hold. Yet, strategies corresponding to the terms on the left contain plays-with-store, given below, that seem to contradict this.

1. $$f : \mathsf{int\,ref} \to \mathsf{unit} \vdash \mathsf{let}\, n = \mathsf{ref}(0)\, \mathsf{in}\, fn; (\lambda x^{\mathsf{int\,ref}}.\mathsf{eq}_{\mathsf{int\,ref}}\, x\, n) \cong \mathsf{let}\, n = \mathsf{ref}(0)\, \mathsf{in}\, fn; (\lambda x^{\mathsf{int\,ref}}.0) : \mathsf{int\,ref} \to \mathsf{int}$$

$$\begin{array}{ccccccccc} \star^{\emptyset} & & (n, c_f)^{(n,0)} & & (\star, \mathsf{r}_f)^{(n,2)} & & (\star, \mathsf{r}_\downarrow)^{(n,2)} & (n, \mathsf{c})^{(n,2)} & (1, \mathsf{r})^{(n,2)} \\ O & & P & & O & & P & O & P \end{array}$$

2. $$f : \mathsf{int\,ref} \to \mathsf{unit} \vdash \mathsf{let}\, n = \mathsf{ref}(0)\, \mathsf{in}\, fn; n := 0; (\lambda x^{\mathsf{unit}}.\mathsf{case}(!n)[1, 0, 0, \cdots]) \cong \mathsf{let}\, n = \mathsf{ref}(0)\, \mathsf{in}\, fn; (\lambda x^{\mathsf{unit}}.1) : \mathsf{unit} \to \mathsf{int}$$

$$\star\ (n, \mathsf{c}_f)^{(n,0)}\ (\star, \mathsf{r}_f)^{(n,k)}\ (\star, \mathsf{r}_\downarrow)^{(n,0)}\ (\star, \mathsf{c})^{(n,1)}\ (0, \mathsf{r})^{(n,1)}$$

In 1. O played a P-name that could not possibly be remembered by a context with ground-type references. In 2. O changes the value stored at such a location. This mismatch motivates further restrictions on the shape of strategies that are dual to those already imposed on P.

Definition 11. *Given a play s, we define its O-view ($\llcorner s \lrcorner$) as follows.*

$$\begin{array}{rcll} \llcorner \epsilon \lrcorner & = & \epsilon & \\ \llcorner \iota\, s\, (\ell_{\mathsf{c}_x}, \mathsf{c}_x) \lrcorner & = & \iota\, (\ell_{\mathsf{c}_x}, \mathsf{c}_x) & \\ \llcorner \iota\, s\, (\ell_{\mathsf{r}_\downarrow}, \mathsf{r}_\downarrow) \lrcorner & = & \iota\, (\ell_{\mathsf{r}_\downarrow}, \mathsf{r}_\downarrow) & \\ \llcorner s\, (\ell_{\mathsf{r}}, \mathsf{r}) \lrcorner & = & \llcorner s' \lrcorner\, (\ell_{\mathsf{c}}, \mathsf{c})\, (\ell_{\mathsf{r}}, \mathsf{r}) & s = s'\, (\ell_{\mathsf{c}}, \mathsf{c}) s'' \text{ and } \mathsf{c} \notin s'' \\ \llcorner s\, o \lrcorner & = & \llcorner s \lrcorner\, o & \end{array}$$

The side condition in the last but one case stipulates that $(\ell_{\mathsf{c}}, \mathsf{c})$ be the last move in s with tag c (i.e. c matches r).

Returning to our examples, we can now see that in the fifth move O was playing/modifying a location from outside the current O-view.

Definition 12. – *A play-with-store $m_1^{\Sigma_1} \cdots m_k^{\Sigma_k}$ is* relevant *if, for all O-moves $(\ell_{2i+1}, t_{2i+1})^{\Sigma_{2i+1}}$ $(1 < 2i+1 \leq k)$ the following conditions hold.*
- *If $\ell_{2i+1} \in P(m_1 \cdots m_{2i+1})$ then $\ell_{2i+1} \in \llcorner m_1 \cdots m_{2i} \lrcorner$.*
- *For any $n \in P(m_1 \cdots m_{2i})$, if $n \notin \llcorner m_1 \cdots m_{2i} \lrcorner$ then $\Sigma_{2i+1}(n) = \Sigma_{2i}(n)$.*

– *A* protoplay *is a sequence of moves-with-stores $m_1^{\Sigma_1} \cdots m_k^{\Sigma_k}$ such that $m_1 \cdots m_k$ is a play and, for any $1 \leq i \leq k$,*
- *$n \in \mathsf{dom}(\Sigma_i) \cap P(m_1 \cdots m_i)$ iff $n \in \llcorner m_1 \cdots m_i \lrcorner$;*
- *$n \in \mathsf{dom}(\Sigma_i) \cap O(m_1 \cdots m_i)$ iff $n \in \ulcorner m_1 \cdots m_i \urcorner$.*

Let us write $[\![\Gamma \vdash M : \theta]\!]_1$ for the set of protoplays obtained by restricting $[\![\Gamma \vdash M : \theta]\!]_0$ to relevant plays and subsequently constraining stores to contain only P-names occurring in the suitable O-view. Protoplays are still not sufficient to prove all equivalences, because they do not convey the idea that the environment might not be able to recognize all different P-names.

Example 13. The following terms are equivalent, yet they induce different protoplays.

$$f : \mathsf{int\,ref} \to \mathsf{unit} \vdash \begin{array}{l} \mathsf{let}\, n_1 = \mathsf{ref}(0)\, \mathsf{in}\, \mathsf{let}\, n_2 = \mathsf{ref}\,(0)\, \mathsf{in}\, f(n_1); (n_2 := !n_1); n_2 \\ \cong \quad \mathsf{let}\, n = \mathsf{ref}\,(0)\, \mathsf{in}\, f(n); n : \mathsf{int\,ref} \end{array}$$

$$\star^{\emptyset}\,(n_1, \mathsf{c}_f)^{(n_1,0)}\,(\star, \mathsf{r}_f)^{(n_1,k)}\,(n_2, \mathsf{r}_\downarrow)^{(n_2,k)} \qquad \star^{\emptyset}\,(n, \mathsf{c}_f)^{(n,0)}\,(\star, \mathsf{r}_f)^{(n,k)}\,(n, \mathsf{r}_\downarrow)^{(n,k)}$$

In an interaction the environment will only be able to detect a difference between two P-names if they occur in the same O-view. Consequently, if P repeats a name introduced by himself, but none of the previous occurrences are present in the O-view, the name should present itself to the environment as if it were fresh. This motivates the last definitions.

Definition 14. *Given a protoplay s, let us subject it to the following refreshment routine: as long as $s = s_1(\ell, t)^{\Sigma} s_2$, where (ℓ, t) is a P-move, $\ell \in P(s)$, ℓ occurs in s_1, but its only occurrence in $\llcorner s_1(\ell, t) \lrcorner$ is in the final move, apply the following to s*

- *if $t = \mathsf{r}_\downarrow$ then replace ℓ with a fresh name.*
- *if $t = \mathsf{c}_x$ then replace ℓ with a fresh name and, provided ℓ occurs there, also in the following O-move;*
- *if $t = \mathsf{r}$ then replace ℓ with a fresh name and, provided ℓ occurs there, in all the following moves with tags c and r.*

Definition 15. *Let $[\![\Gamma \vdash M : \theta]\!]_2$ consist of protoplays from $[\![\Gamma \vdash M : \theta]\!]_1$ refreshed according to Definition 14.*

A play-with-store or a protoplay will be called complete, if the underlying play is complete. Given a set S of plays-with-store or protoplays, let us write $\mathsf{comp}(S)$ for the subset of S consisting of complete elements only.

Theorem 16 (Lemma 17 [12]). *For any $\mathsf{RedML}^{\beta\to\beta}_{\mathrm{fin}}$-terms $\Gamma \vdash M_1, M_2 : \theta$, $\Gamma \vdash M_1 \cong M_2 : \theta$ if, and only if, $\mathsf{comp}([\![\Gamma \vdash M_1 : \theta]\!]_2) = \mathsf{comp}([\![\Gamma \vdash M_2 : \theta]\!]_2)$.*

In Section 5, for a given term $\Gamma \vdash M : \theta$ of $\mathsf{RedML}^{\beta\to\beta}_{\mathrm{fin}}$ in canonical form, we shall construct a family of automata representing $[\![\Gamma \vdash M : \theta]\!]_0$. It will be refined in Section 6 to represent $[\![\Gamma \vdash M : \theta]\!]_2$. Section 7 will be devoted to crafting a bisimulation relation that will enable us to implement the equivalence test from Theorem 16.

4 Automata

As Example 9 demonstrates, the stores present in plays can grow indefinitely and, even though we shall work with infinite alphabets, we cannot afford to represent them literally, as this would amount to being able to memorize an unbounded supply of names. Instead we shall skip store information in O-moves on the understanding that, whenever this is done, the omitted value could be arbitrary. Similarly, if store values are omitted for P-moves, it will be the case that P does not change them, i.e. they remain the same as in the previous O-move. According to this convention the first play from Example 9 can be faithfully represented by $\iota^{\emptyset}\,(\star,\mathsf{r}_{\downarrow})^{\emptyset}\,(\star,c)^{\emptyset}\,(n_1,r)^{(n_1,0)}\,(\star,c)^{\emptyset}\,(n_2,r)^{(n_2,0)}\,\cdots$.

Next we introduce the kind of automata that will be used as acceptors of (representatives of) plays. In a single transition step they will be able to read a (representation of a) single move-with-store $(\ell,t)^{\Sigma}$ (subject to the condition that Σ is a subset of the actual store). On the technical level, the automata are a variant of fresh-register automata [17], adapted to process plays-with-stores. Their sets of states will be partitioned into O- and P-states, which correspond to the stages of play when O and P respectively are about to make a move. The machines will be equipped with a finite number of registers for storing names. At O-states they will be able to recognize whether the currently read name is present in one of the registers. At P states they will be able to process a currently stored name or a fresh one (one that has not been processed so far).

To enable a finite specification of the automata and to describe their semantics we introduce the following definitions. Recall that $\mathbb{A}$ is the set of names. Let $\mathbb{C} = \{\star, 0, \cdots, max\}$ be the set of *constants*. Let us also fix a finite set $\mathbb{T}$ of *tags* and a positive integer n.

Definition 17.
- $\mathbb{L} = \mathbb{C} \cup \{\, \mathsf{fr}(i), \mathsf{kn}(i) \mid 1 \leq i \leq n \,\}$ *is the set of* symbolic labels. *We use ℓ to range over its elements.*
- Reg *is the set of all* $\rho : \{1, \cdots, n\} \to \mathbb{A} \cup \{\sharp\}$ *such that* $\rho(i) = \rho(j) \in \mathbb{A}$ *implies* $i = j$. *Its elements will be called* register assignments *and, from now on, we shall use ρ to range over them.*
- $\mathsf{Sto}^{\mathsf{s}} = \{1, \ldots, n\} \to \{\sharp, 0, \ldots, max\}$ *is the set of* symbolic stores*, which will be ranged over by S.*
- Sto *is the set of partial functions* $\Sigma : \mathbb{A} \rightharpoonup \{0, \ldots, max\}$ *such that* $\mathsf{dom}(\Sigma)$ *contains at most n elements. Its elements will be referred to as* stores *and ranged over by Σ.*
- $\mathsf{LT} = \mathbb{L} \times \mathbb{T} \times \mathsf{Sto}^{\mathsf{s}}$ *is the set of* transition labels*, ranged over by* $(\ell,t)^{S}$.

We shall abuse notation somewhat and write $\mathsf{dom}(\rho)$ for the set $\rho^{-1}(\mathbb{A})$, and similarly for $\mathsf{dom}(S)$. Given a pair $(\rho, S) \in \mathsf{Reg} \times \mathsf{Sto}^{\mathsf{s}}$ such that $\mathsf{dom}(\rho) = \mathsf{dom}(S)$, we can derive the store $\mathsf{Sto}(\rho, S) = \{\, (\rho(i), S(i)) \mid i \in \mathsf{dom}(\rho) \,\}$.

We can now define (n_{r}, n)-automata, which will be used for representing game semantics. An (n_{r}, n)-automaton is equipped with n registers, the first n_{r} of which will be read-only.

Definition 18. *An* **(n_{r}, n)-automaton** *of type θ is given as a quintuple* $\mathcal{A} = \langle Q, q_0, \rho_0, \delta, F \rangle$ *where:*

– Q *is a finite set of states, partitioned into* Q_O *(O-states) and* Q_P *(P-states);*
– $q_0 \in Q_P$ *is the initial state;*
– $\rho_0 \in \mathsf{Reg}$ *is the initial register assignment such that* $\mathsf{dom}(\rho_0) = \{1, \cdots, n_\mathsf{r}\}$*;*
– $\delta \subseteq (Q_O \times \mathsf{LT} \times Q_P) \cup (Q_P \times \mathsf{LT} \times Q_O) \cup (Q_O \times \mathcal{P}(\{n_\mathsf{r}+1, \ldots, n\}) \times Q_O) \cup (Q_P \times \mathcal{P}(\{n_\mathsf{r}+1, \ldots, n\}) \times Q_P)$ *is the transition relation;*
– $F \subseteq Q_O$ *is the set of final states.*

Additionally, the following properties must hold.

– *if* $(q, (\ell, t)^S, q') \in \delta$ *and* $\ell = \mathsf{fr}(i)$ *then* $i > n_\mathsf{r}$ *and* $i \in \mathsf{dom}(S)$*.*
– *if* θ *is a base type then there is a unique final state* q_F*, and* $\delta \upharpoonright \{q_F\} = \emptyset$ *(no outgoing transition).*

Our automata operate on words over the infinite alphabet $(\mathbb{C} \cup \mathbb{A}) \times \mathbb{T} \times \mathsf{Sto}$. We shall write $(\ell, t)^\Sigma$ to refer to its elements. We first explain the meaning of the transition function informally. Suppose $\mathcal{A}$ is at state q_1 and ρ is the current register assignment.

– If $(q_1, (\ell', t)^S, q_2) \in \delta$, $\mathcal{A}$ can move to state q_2 on the input symbol $(\ell, t)^\Sigma$ if one of the following conditions is satisfied.
  - $\ell \in \mathbb{C}$, $\ell' = \ell$, $\mathsf{dom}(S) \subseteq \mathsf{dom}(\rho)$ and $\Sigma = \mathsf{Sto}(\rho, S)$.
  - $\ell \in \mathbb{A}$, $\ell' = \mathsf{kn}(i)$, $\rho(i) = \ell$, $\mathsf{dom}(S) \subseteq \mathsf{dom}(\rho)$ and $\Sigma = \mathsf{Sto}(\rho, S)$.
  - $\ell \in \mathbb{A}$, $\ell' = \mathsf{fr}(i)$, $\mathsf{dom}(S) \subseteq \mathsf{dom}(\rho) \cup \{i\}$, $\Sigma = \mathsf{Sto}(\rho[i \mapsto \ell], S)$ and
    * either $q_1 \in Q_O$ and ℓ does not belong to $\rho(\{1, \cdots, n\})$ (*locally fresh*),
    * or $q_1 \in Q_P$ and ℓ has not appeared in the current run of $\mathcal{A}$ (*globally fresh*).

  In this case the automaton also sets $\rho(i)$ to ℓ.
– If $(q_1, N, q_2) \in \delta$, where N is a subset of writable register indices, $\mathcal{A}$ can clear all registers in N (i.e. set $\rho(i) = \sharp$ for all $i \in N$) and move to q_2 without reading any input symbol (ϵ-transition).

The above is formalized next. A *configuration* of $\mathcal{A}$ is a triple $(q, \rho, H) \in \hat{Q}$, where $\hat{Q} = Q \times \mathsf{Reg} \times \mathcal{P}_\mathsf{fn}(\mathbb{A})$ and $\mathcal{P}_\mathsf{fn}(\mathbb{A})$ is the set of finite subsets of $\mathbb{A}$.

Definition 19. *Let* $\mathcal{A} = \langle Q, q_0, \rho_0, \delta, F \rangle$ *be an* (n_r, n)*-automaton. The configuration graph* $(\hat{Q}, \longrightarrow_\delta)$ *of* $\mathcal{A}$ *is defined as follows (transitions are labelled by* ϵ *or elements of* $(\mathbb{C} \cup \mathbb{A}) \times \mathbb{T} \times \mathsf{Sto}$*). For all* $(q, \rho, H) \in \hat{Q}$ *and* $(q, (\ell, t)^S, q') \in \delta$*:*

– *if* $\ell \in \mathbb{C}$ *and* $\mathsf{dom}(S) \subseteq \mathsf{dom}(\rho)$ *then* $(q, \rho, H) \xrightarrow{(\ell,t)^\Sigma}_\delta (q', \rho, H)$ *where* $\Sigma = \mathsf{Sto}(\rho, S)$*,*
– *if* $\ell \notin \mathbb{C}$ *and* $\mathsf{dom}(S) \subseteq \mathsf{dom}(\rho')$ *then* $(q, \rho, H) \xrightarrow{(\ell,t)^\Sigma}_\delta (q', \rho', H')$ *where* $\Sigma = \mathsf{Sto}(\rho', S)$*,* $H' = H \cup \{\ell\}$*, and*
  - *if* $\ell = \mathsf{kn}(i)$ *then* $\ell = \rho(i)$ *and* $\rho' = \rho$*,*
  - *if* $\ell = \mathsf{fr}(i)$ *and* $q \in Q_O$ *then* $\ell \notin \rho(\{1, \cdots, n\})$ *and* $\rho' = \rho[i \mapsto \ell]$*,*
  - *if* $\ell = \mathsf{fr}(i)$ *and* $q \in Q_P$ *then* $\ell \notin \rho(\{1, \cdots, n_\mathsf{r}\}) \cup H$ *and* $\rho' = \rho[i \mapsto \ell]$*.*

Moreover, for all $(q, \rho, H) \in \hat{Q}$ *and* $(q, N, q') \in \delta$ *we have* $(q, \rho, H) \xrightarrow{\epsilon}_\delta (q', \rho', H)$*, where* $\rho' = \rho[N \mapsto \sharp]$*. The set of strings* accepted *by* $\mathcal{A}$ *is defined to be*

$$\mathcal{L}(\mathcal{A}) = \{\, \vec{\ell} \in ((\mathbb{C} \cup \mathbb{A}) \times \mathbb{T} \times \mathsf{Sto})^* \quad | \quad (q_0, \rho_0, \emptyset) \xrightarrow{\vec{\ell}}_\delta (q, \rho, H), \quad q \in F \,\}.$$

Definition 20. *We say that $\mathcal{A}$ is* deterministic *if, for any reachable configuration $\hat{q}$ and any $\hat{q} \xrightarrow{\ell_1}_\delta \hat{q}_1, \hat{q} \xrightarrow{\ell_2}_\delta \hat{q}_2$, if $\ell_1 = \ell_2$ then $\hat{q}_1 = \hat{q}_2$.*

Here is a structural constraint that guarantees determinacy.

Definition 21. *$\mathcal{A}$ is* strongly deterministic *if:*

- *for each $q \in Q_P$ there exists* at most *one transition out of q: $|\, \delta \upharpoonright \{q\}| \leq 1$;*
- *for each $q \in Q_O$ and $(q, (\ell_1, t)^{S_1}, q_1), (q, (\ell_2, t)^{S_2}, q_2) \in \delta$:*
  - *if $\ell_1 = \mathsf{fr}(i_1)$ and $\ell_2 = \mathsf{fr}(i_2)$ then $i_1 = i_2$,*
  - *if $\ell_1 = \ell_2$ and $S_1 = S_2$ then $q_1 = q_2$,*
  - *$\mathsf{dom}(S_1) \setminus \{\, i \mid \ell_1 = \mathsf{fr}(i)\} \;=\; \mathsf{dom}(S_2) \setminus \{\, i \mid \ell_2 = \mathsf{fr}(i)\}$.*
- *for any $q_1 \in Q_O$, if there exists $q_2 \in Q_O$ such that $(q_1, N, q_2) \in \delta$, then this is the only outgoing transition from q_1: $|\, \delta \upharpoonright \{q_1\}| = 1$.*

Definition 22. *Let $\mathcal{A} = \langle\, Q, q_0, \rho_0, \delta, q_F \,\rangle$ be a strongly deterministic automaton of base type. We define the set of* quasi-final *states E to be the set of states that reach q_F in one step. Then E is canonically partitioned as $E = \biguplus_{(\ell,t)^S} E_{(\ell,t)^S}$ where $E_{(\ell,t)^S} = \{\, q \in Q \mid (q, (\ell, t)^S, q_F) \in \delta \,\}$ and $\mathcal{A}$ is uniquely determined by the structure $\mathcal{A}^- = \langle Q, q_0, \rho_0, \delta, E\rangle$.*

The following remark sheds some light on the formal nominal setting underlying our constructions. It can be safely skipped by readers not familiar with nominal sets [5].

Remark 23. Note that the initial register assignments of our automata contain names. One can view the automata as elements of nominal sets where name-permutation works as follows: for any name-permutation π, $\pi \cdot \langle Q, q_0, \rho_0, \delta, F\rangle = \langle Q, q_0, \pi \cdot \rho_0, \delta, F\rangle$, where $\pi \cdot \rho = \pi \circ \rho$. Note that then $\mathcal{L}(\pi \cdot \mathcal{A}) = \pi \cdot \mathcal{L}(\mathcal{A})$.

Moreover, the indexed families of automata to be used in the next definition are of nominal nature. Let X be a nominal set. By an X-indexed family of automata of type θ we mean a set $\{\, \mathcal{A}_x \mid x \in X \,\}$ such that each $\mathcal{A}_x$ is an (n^x_{r}, n^x)-automaton of type θ and, moreover, for any name-permutation π, $\mathcal{A}_{\pi \cdot x} = \pi \cdot \mathcal{A}_x$.

5 From terms to plays-with-stores

Let $\Gamma = [x_1 : \theta_1, \cdots, x_m : \theta_m]$ and $\Gamma \vdash \mathbb{C} : \theta$ be a $\mathsf{RedML}^{\beta\to\beta}_{\mathrm{fin}}$-term in canonical form. Let us write $I^+_{\Gamma\vdash\theta}$ for the set of plays-with-store of length 1 over $\Gamma \vdash \theta$. Recall that each of them will have the form ι^{Σ_0}, where $\iota \in I_\Gamma$, i.e. $\iota = (\ell_1, \cdots, \ell_m)$, where $\ell_i \in \mathcal{L}_{\theta_i}$. Let $\mathcal{L}_\iota = \{\ell_i \,|\, \theta_i \equiv \mathsf{int\,ref}\}$ and $n_\iota = |\mathcal{L}_\iota|$. Then $\mathsf{dom}(\Sigma_0) = \mathcal{L}_\iota$. Let $\mathsf{idx} : \{1, \cdots, n_\iota\} \to \{1, \cdots, m\}$ be defined by $\mathsf{idx}(i) = j$ if $\ell_j \in \mathcal{L}_\iota$, there are $i-1$ different names in ι to the left of ℓ_j ($|\{\ell_1, \cdots, \ell_{j-1}\} \cap \mathcal{L}_\iota| = i-1$) and ℓ_j is not among them ($\ell_j \notin \{\ell_1, \cdots, \ell_{j-1}\}$).

We now instantiate the automata defined in the previous section by using the finite set of tags $\mathbb{T} = \mathcal{T}_{\Gamma\vdash\theta}$. A canonical form of $\mathsf{RedML}^{\beta\to\beta}_{\mathrm{fin}}$ will be translated into a family of automata indexed by $I^+_{\Gamma\vdash\theta}$. For each $\iota^{\Sigma_0} \in I^+_{\Gamma\vdash\theta}$, the corresponding automaton will accept exactly the words w such that $\iota^{\Sigma_0} w$ is a complete play induced by the canonical form. The family will be infinite, but *finite* when considered up to name-permutability.

Definition 24. *For any* $\mathsf{RedML}^{\beta\to\beta}_{\mathrm{fin}}$*-term* $\Gamma \vdash \mathbb{C} : \theta$ *in canonical form we define an* $I^+_{\Gamma\vdash\theta}$*-indexed family of automata* $(\!|\mathbb{C}|\!) = \{\, (\!|\mathbb{C}|\!)_{\iota^{\Sigma_0}} \mid \iota^{\Sigma_0} \in I^+_{\Gamma\vdash\theta} \,\}$ *by induction on the shape of* $\mathbb{C}$*. In all cases* $(\!|\mathbb{C}|\!)_{\iota^{\Sigma_0}}$ *will have* $n_{\mathsf{r}}^{\iota^{\Sigma_0}} = n_\iota$ *read-only registers and the initial assignment will be* $\rho_0^{\iota^{\Sigma_0}}(i) = \ell_{\mathsf{idx}(i)}$*. The precise number of registers can be calculated easily by reference to the constituent automata. Let us write* S_0 *for the function* $S_0 : \{1, \cdots, n\} \to \{\sharp, 0, \cdots, max\}$ *defined by* $S_0(i) = \Sigma_0(\ell_{\mathsf{idx}(i)})$ $(1 \le i \le n_\iota)$ *and* $S_0(i) = \sharp$ $(i > n_\iota)$*. The base and inductive cases are as follows.*[5]

- $(\!|\,()\,|\!)_{\iota^{\Sigma_0}} = \quad q_0 \xrightarrow{(\star,\mathsf{r}_\downarrow)^{S_0}} q_F$
- $(\!|\,j\,|\!)_{\iota^{\Sigma_0}} = \quad q_0 \xrightarrow{(j,\mathsf{r}_\downarrow)^{S_0}} q_F$
- $(\!|\,x^{\mathsf{int\,ref}}\,|\!)_{\iota^{\Sigma_0}} = \quad q_0 \xrightarrow{(\mathsf{kn}(j),\mathsf{r}_\downarrow)^{S_0}} q_F$ *, where* $x \equiv x_k$ *and* $\ell_{\mathsf{idx}(j)} = \ell_k$
- $(\!|\,\mathsf{case}(x)[\mathbb{C}_0, \cdots, \mathbb{C}_{max}]\,|\!)_{\iota^{\Sigma_0}} = (\!|\,\mathbb{C}_j\,|\!)_{\iota^{\Sigma_0}}$*, where* $x \equiv x_k$ *and* $\ell_k = j$
- $(\!|\,(x := i); \mathbb{C}\,|\!)_{\iota^{\Sigma_0}} = (\!|\,\mathbb{C}\,|\!)_{\iota^{\Sigma_0'}}$*, where* $x \equiv x_k$ *and* $\Sigma_0' = \Sigma_0[\ell_k \mapsto i]$
- $(\!|\,\mathsf{let}\, y = !x \,\mathsf{in}\, \mathbb{C}\,|\!)_{\iota^{\Sigma_0}} = (\!|\,\mathbb{C}\,|\!)_{(\iota\,\Sigma_0(\ell_k))^{\Sigma_0}}$*, where* $x \equiv x_k$
- $(\!|\,\mathsf{let}\, y^{\mathsf{unit}} = z() \,\mathsf{in}\, \mathbb{C}\,|\!)_{\iota^{\Sigma_0}}$ *is given by* $q_0 \xrightarrow{(\star,\mathsf{c}_z)^{S_0}} q_1 \xrightarrow{(\star,\mathsf{r}_z)^{S}} (\!|\,\mathbb{C}\,|\!)_{(\iota\,\star)^{\Sigma_S}}$ *, where* S *ranges over all symbolic stores with domain* $\{1, \ldots, n_\iota\}$ *and* $\Sigma_S(\ell_{\mathsf{idx}(i)}) = S(i)$*.*
- $(\!|\,\mathsf{let}\, y^{\mathsf{int}} = z() \,\mathsf{in}\, \mathbb{C}\,|\!)_{\iota^{\Sigma_0}} = \quad q_0 \xrightarrow{(\star,\mathsf{c}_z)^{S_0}} q_1 \xrightarrow{(j,\mathsf{r}_z)^{S}} (\!|\,\mathbb{C}\,|\!)_{(\iota\,j)^{\Sigma_S}}$ *with* S *as above and* $0 \le j \le max$*.*
- $(\!|\,\mathsf{let}\, y^{\mathsf{int\,ref}} = z() \,\mathsf{in}\, \mathbb{C}\,|\!)_{\iota^{\Sigma_0}} = \quad q_0 \xrightarrow{(\star,\mathsf{c}_z)^{S_0}} q_1 \xrightarrow{(\mathsf{kn}(j),\mathsf{r}_z)^{S}} (\!|\,\mathbb{C}\,|\!)_{(\iota\,\rho_0(j))^{\Sigma_S}}$ *where*

  $q_1 \xrightarrow{(\mathsf{fr}(n_\iota+1),\mathsf{r}_z)^{S'}} (\!|\,\mathbb{C}\,|\!)_{(\iota\,a)^{\Sigma_{S'}}}$

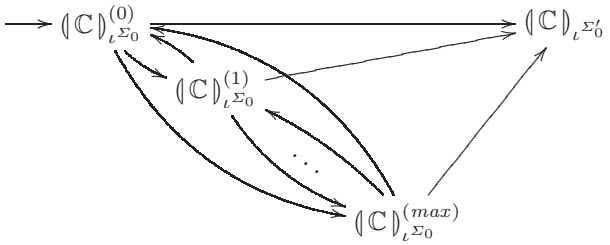
  $1 \le j \le n_\iota$*,* S *is as above,* a *is a name that does not occur in* ι*,*[6] S' *ranges over symbolic stores with domain* $\{1, \cdots, n_\iota + 1\}$*,* $\Sigma_{S'}(\ell_{\mathsf{idx}(i)}) = S'(i)$ $(1 \le i \le n_\iota)$ *and* $\Sigma_{S'}(a) = S'(n_\iota + 1)$
- $(\!|\,\mathsf{let}\, y^{\beta} = z\, i \,\mathsf{in}\, \mathbb{C}\,|\!)_{\iota^{\Sigma_0}}$ *and* $(\!|\,\mathsf{let}\, y^{\beta} = z\, x^{\mathsf{int\,ref}} \,\mathsf{in}\, \mathbb{C}\,|\!)_{\iota^{\Sigma_0}}$ *are defined similarly to the above.*
- $(\!|\,\lambda x^{\mathsf{unit}}.\mathbb{C}\,|\!)_{\iota^{\Sigma_0}} = \quad q_0 \xrightarrow{(\star,\mathsf{r}_\downarrow)^{S_0}} q_1 \xrightarrow{(\star,\mathsf{c})^{S}} ((\!|\,\mathbb{C}\,|\!)_{(\iota\,\star)^{\Sigma_S}}[\mathsf{r}/\mathsf{r}_\downarrow])$ *, where the loopback connects the final state of* $(\!|\,\mathbb{C}\,|\!)_{(\iota\,\star)^{\Sigma_S}}[\mathsf{r}/\mathsf{r}_\downarrow]$ *to* q_1*. Note that* S *ranges over all symbolic stores with domain* $\{1, \ldots, n_\iota\}$ *and* $\Sigma_S(\ell_{\mathsf{idx}(i)}) = S(i)$ $(1 \le i \le n_\iota)$*. The other cases of* λ*-abstraction are dealt with in a similar way.*
- *Case of* $\mathsf{let}\, x^{\mathsf{int\,ref}} = \mathsf{ref}\,(0) \,\mathsf{in}\, \mathbb{C}$*. Let* a *be a name not in* ι*, let* $\Sigma_0' = \Sigma_0[a \mapsto 0]$ *and consider the* $(n_\iota + 1, n)$*-automaton* $(\!|\mathbb{C}|\!)_{(\iota\,a)^{\Sigma_0'}}$*. For each* $0 \le j \le max$ *define an* (n_ι, n)*-automaton* $(\!|\mathbb{C}|\!)^{(j)}_{\iota^{\Sigma_0}}$ *to be a copy of* $(\!|\mathbb{C}|\!)_{(\iota\,a)^{\Sigma_0'}}$ *in which*

[5] Note that we ignore initial register assignments of automata that do not appear in starting positions in the diagrams.

[6] Any such choice of a yields the same automaton $(\!|\mathbb{C}|\!)_{(\iota\,a)^{\Sigma_{S'}}}$ after its initial register assignment is removed.

- *all transitions with label* $\mathsf{kn}(n_\iota + 1)$ *are removed,*
- *all transitions with symbolic store* S *such that* $S(n_\iota{+}1) \neq j$ *are removed,*
- *and* $n_\iota + 1$ *is removed from all symbolic stores in remaining transitions.*

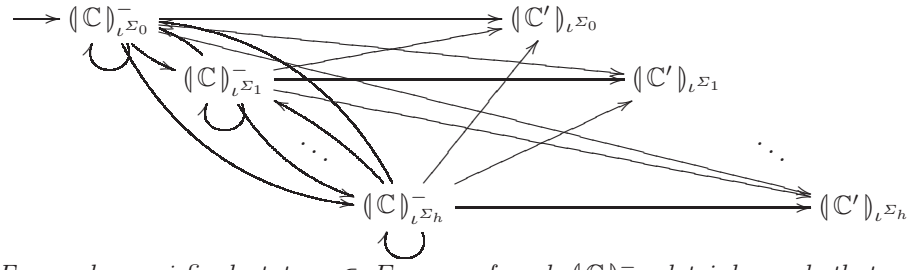
We define $(\!|\, \mathsf{let}\, x^{\mathsf{int\,ref}} = \mathsf{ref}\,(0)\,\mathsf{in}\,\mathbb{C} \,|\!)$ *as an* (n_ι, n)*-automaton obtained by interconnecting* $(\!|\,\mathbb{C}\,|\!)^{(0)}_{\iota^{\Sigma_0}}, \cdots, (\!|\,\mathbb{C}\,|\!)^{(max)}_{\iota^{\Sigma_0}}$ *and* $(\!|\,\mathbb{C}\,|\!)_{(\iota\, a)^{\Sigma'_0}}$.



The transitions between the copies (arrows in the figure) are as follows. For each $q_1 \xrightarrow{(\ell,t)^S} q_2$ *in* $(\!|\,\mathbb{C}\,|\!)_{\iota^{\Sigma'_0}}$*, where* q_1 *is a P-state:*

- *if* $\ell \neq \mathsf{kn}(n_\iota{+}1)$ *then, for each* $j \neq S(n_\iota{+}1)$*, we add a transition from the state* q_1 *of* $(\!|\,\mathbb{C}\,|\!)^{(j)}_{\iota^{\Sigma_0}}$ *to state* q_2 *of* $(\!|\,\mathbb{C}\,|\!)^{(S(n_\iota+1))}_{\iota^{\Sigma_0}}$*, with label* $(\ell, t)^{S\restriction(\mathsf{dom}(S)\setminus\{n_\iota+1\})}$*;*
- *if* $\ell = \mathsf{kn}(n_\iota + 1)$ *then, for each* $0 \leq j \leq max$*, we add a transition from the state* q_1 *of* $(\!|\,\mathbb{C}\,|\!)^{(j)}_{\iota^{\Sigma_0}}$ *to state* q_2 *of* $(\!|\,\mathbb{C}\,|\!)_{\iota^{\Sigma'_0}}$*, with label* $(\mathsf{fr}(n_\mathsf{r} + 1), t)^S$*.*

– *For* $(\mathsf{while}\,(!x)\,\mathsf{do}\,\mathbb{C}); \mathbb{C}'$*, let* $\Sigma_0, \ldots, \Sigma_h$ *be all the stores with the same domain as* Σ_0*. Assume* $x \equiv x_k$*. Recall the presentation of an automaton given in Definition 22. We define* $(\!|\,(\mathsf{while}\,(!x)\,\mathsf{do}\,\mathbb{C}); \mathbb{C}'\,|\!)_{\iota^{\Sigma_0}}$ *to be* $(\!|\,\mathbb{C}'\,|\!)_{\iota^{\Sigma_0}}$ *if* $\Sigma_0(\ell_k) = 0$*. Otherwise it is defined to be a combination of* $(\!|\,\mathbb{C}\,|\!)^-_{\iota^{\Sigma_0}}, \cdots, (\!|\,\mathbb{C}\,|\!)^-_{\iota^{\Sigma_h}}$ *and* $(\!|\,\mathbb{C}'\,|\!)_{\iota^{\Sigma_0}}, \cdots, (\!|\,\mathbb{C}'\,|\!)_{\iota^{\Sigma_h}}$ *connected together as explained below.*



For each quasi-final state $q \in E_{(\star,\mathsf{r}_\perp)^S}$ *of each* $(\!|\,\mathbb{C}\,|\!)^-_{\iota^{\Sigma_i}}$ *let* i *be such that* $\ell_{\mathsf{idx}(i)} = \ell_k$*. Add transitions labelled* $\{n_\iota + 1, \ldots, n\}$ *in the following cases.*

- *If* $S(i) = 0$*, add one from* q *to the initial state of* $(\!|\,\mathbb{C}'\,|\!)_{\iota^{\Sigma_S}}$*,*
- *if* $S(i) \neq 0$ *add one from* q *to the initial state of* $(\!|\,\mathbb{C}\,|\!)^-_{\iota^{\Sigma_S}}$*.*

We shall now formalize in what sense the automata defined above can be taken to represent strategies.

Definition 25. *Let* $s = m_1^{\Sigma_1} \cdots m_k^{\Sigma_k}$ *be a play over* $\Gamma \vdash \theta$ *and* $t = m_1^{\Theta_1} \cdots m_k^{\Theta_k}$ *be a sequence of moves-with-store. We say that* s *is an* extension *of* t *if the following conditions are satisfied.*

– $\Theta_i \subseteq \Sigma_i$ $(1 \leq i \leq k)$
– *For any* $1 \leq i \leq \lfloor k/2 \rfloor$, *if* $a \in \mathsf{dom}(\Sigma_{2i}) \setminus \mathsf{dom}(\Theta_{2i})$ *then* $\Sigma_{2i-1}(a)$ *is defined and* $\Sigma_{2i}(a) = \Sigma_{2i-1}(a)$.

Note that, because s is a play, the clause about $\Sigma_{2i-1}(a)$ being defined amounts to stipulating that $\Theta_{2i}(a)$ be defined if the first occurrence of a in $m_1 \cdots m_k$ is in m_{2i}. Observe that this is always the case for words accepted by automata from Definition 24, as the store values are always printed out for fresh labels. Let us write $\mathsf{ext}(t)$ for the set of all extensions of t.

Lemma 26. *For any* $\iota^{\Sigma_0} \in I^+_{\Gamma \vdash \theta}$, *the automaton* $(\!|\, \mathbb{C} \,|\!)_{\iota^{\Sigma_0}}$ *is strongly deterministic and* $\bigcup_{t \in \mathcal{L}((\!|\, \mathbb{C} \,|\!)_{\iota^{\Sigma_0}})} \mathsf{ext}(\iota^{\Sigma_0} t) = \{\, \iota^{\Sigma_0} t \mid \iota^{\Sigma_0} t \in \mathsf{comp}([\![\Gamma \vdash \mathbb{C} : \theta]\!]_0) \,\}$.

Proof. Strong determinacy follows from the shape of the automata. For the latter part, the non-trivial cases are:

$\mathsf{let}\, x = \mathsf{ref}\,(0)\,\mathsf{in}\,\mathbb{C}$: The role of $(\!|\, \mathbb{C} \,|\!)^{(j)}_{\iota^{\Sigma_0}}$ $(0 \leq j \leq max)$ is to mimic the strategy corresponding to $\mathsf{let}\, x = \mathsf{ref}\,(0)\,\mathsf{in}\,\mathbb{C}$ before the name a (corresponding to the reference name x) is revealed. Accordingly, O is not allowed to play the name for the first time (as in the definition of composition [12]) or change store-values associated with a. P can still change them, though, and the current value at a is represented by the superscript (j). If P reveals the name for the first time, a fresh name is generated, written to register $n_\iota + 1$, and computation proceeds to $(\!|\, \mathbb{C} \,|\!)_{\iota^{\Sigma'_0}}$.

$\lambda x^\beta.\mathbb{C}$: Correctness follows from the fact that the strategy corresponding to $\lambda x^\beta.\mathbb{C}$ is single-threaded, i.e., following $(\star, \mathsf{r}_\downarrow)$, it is an interleaving of independent copies of strategies for $\mathbb{C}$.

$(\mathsf{while}\,(!x)\,\mathsf{do}\,\mathbb{C}); \mathbb{C}'$: Observe that the strategy for $(\mathsf{while}\,(!x)\,\mathsf{do}\,\mathbb{C})$ can be viewed as a subset of the single-threaded strategy for $\lambda y^{\mathsf{unit}}.\mathsf{if}\, !x\, \mathsf{then}\, \mathbb{C}\, \mathsf{else}\,()$, where the final values of free variables are communicated to the next thread and on reaching 0 by x control is transferred to the strategy for $\mathbb{C}'$. This is precisely what the construction achieves. □

6 Automata for protoplays

In this section we shall transform the automata representing $[\![\Gamma \vdash \mathbb{C}]\!]_0$ to represent $[\![\Gamma \vdash \mathbb{C}]\!]_2$. To that end, we need to focus on protoplays that have been refreshed according to Definition 14. Such protoplays adhere to a specific pattern with respect to P-names: any P-name that appears in the O-view for the first time has not appeared earlier. This implies that in each such protoplay $s = s_1(\ell, t)^\Sigma$ with $\ell \in P(s)$:

– if $t = \mathsf{c}_x$ or $t = \mathsf{r}_\downarrow$ then ℓ is fresh for s_1,
– if $t = \mathsf{r}$ then either ℓ is fresh for s_1 or it has been introduced by some move with tag r.

The moral is that P-names introduced with tag c_x or $\mathsf{r}_\downarrow$ are played only once. In the former case, O may play them in his next move, but then they will not

reappear in the protoplay. In the latter case, no moves can be made after the P-name ($\theta \equiv \mathsf{int\,ref}$). Hence, we can safely replace P-names introduced with tags c_x and $\mathsf{r}_\downarrow$ by dummy labels ($\circledast$) and still have a faithful representation, provided their values are remembered. We shall accommodate them in special labels so that the representation is still a protoplay, albeit in the following extended syntax.

Remark 27. Using $\circledast$ will also let us see that for terms $\Gamma \vdash \mathbb{C} : \beta$ one does not need globally fresh transitions. Hence the corresponding automata will then be variants of register automata [7] rather than fresh-register automata [17].

Definition 28. – *For every base type β we define the set of extended labels* $\mathcal{L}^+_\beta = \mathcal{L}_\beta \cup \{\, \ell\langle i\rangle \mid \ell \in \mathcal{L}_\beta \cup \{\circledast\},\ 0 \le i \le max \,\}$.
– *We define the set of transition labels* $\mathbb{L}^+ = \mathbb{L} \cup \{\, \ell\langle i\rangle \mid \ell \in \mathbb{L} \cup \{\circledast\},\ 0 \le i \le max \,\}$.

We proceed to define a translation from an automaton $\mathcal{A}$ in the original syntax to an automaton $\overline{\mathcal{A}^+}$ in the extended one, following the intuitions described above. As a first step, we are going to enrich $\mathcal{A}$ with information about ownership of the names that are currently stored in the registers. This is concretely achieved as follows.

Definition 29. *For each automaton $\mathcal{A} = \langle Q, q_0, \rho_0, \delta, F\rangle$ construct the automaton $\mathcal{A}^+ = \langle Q', q_0', \rho_0', \delta', F'\rangle$ by setting*

$$Q' = \{\, (q, N_O, N_P) \in Q \times \mathcal{P}(\{1,\dots,n\})^2 \mid N_O \cap N_P = \emptyset,\ \{1,\dots,n_\mathsf{r}\} \subseteq N_O \,\},$$

$q_0' = (q_0, \{1,\dots,n_\mathsf{r}\}, \emptyset)$, $\rho_0' = \rho_0$, $F' = Q' \restriction F$, *and defining δ' as follows.*

– *If $q \xrightarrow{(\ell,t)^S} q'$ in $\mathcal{A}$ then $(q, N_O, N_P) \xrightarrow{(\ell,t)^S} (q', N_O, N_P)$, provided $\mathsf{dom}(S) = N_O \cup N_P$ and $\ell \in \mathbb{C} \cup \{\mathsf{kn}(i) \mid i \in N_O \cup N_P\}$.*
– *If $q_O \xrightarrow{(\mathsf{fr}(i),t)^S} q_P$ in $\mathcal{A}$ then $(q_O, N_O, N_P) \xrightarrow{(\mathsf{fr}(i),t)^S} (q_P, N_O \cup \{i\}, N_P \setminus \{i\})$, provided $\mathsf{dom}(S) = N_O \cup N_P$. If the transition is from q_P to q_O, the dual holds.*
– *If $q \xrightarrow{N} q'$ in $\mathcal{A}$ then $(q, N_O, N_P) \xrightarrow{N} (q', N_O \setminus N, N_P \setminus N)$.*

For each $q \in Q'$ we shall write $O(q)$ and $P(q)$ for its second and third components respectively. We say $\mathcal{A}^+$ is *non-overwriting* if, for any $q \xrightarrow{(\mathsf{fr}(i),t)^S} q'$, we have that $i \notin O(q) \cup P(q)$. Observe that the constructions presented in Definition 24 always yield non-overwriting automata. We can show that the new automaton reaches state q only if its non-empty registers are those in $N_O(q) \cup N_P(q)$ and, moreover, each register in $N_O(q)$ (resp. $N_P(q)$) is filled with an O-name (a P-name). Note also that $\mathcal{A}^+$ is strongly deterministic, if $\mathcal{A}$ was.

For the next step, recall that we are using a finite set of tags $\mathbb{T} = \mathcal{T}_{\Gamma\vdash\theta}$, for some Γ, θ. The new automaton will feature states augmented with an extra component N to record those P-names that were originally introduced by c_x-moves but have been replaced by $\circledast$. For O-states we need an extra component

S which records the symbolic store prior to hiding, and also an index $i \in N \cup \{\text{-}\}$ reporting whether the preceding move was a name replaced now by $\circledast$ (by convention, if the preceding move was not such then the index is set to '-').

Definition 30. *Let $\mathcal{A}^+ = \langle Q, q_0, \rho_0, \delta, F\rangle$ be non-overwriting. We define an automaton $\overline{\mathcal{A}^+} = \langle Q', q_0', \rho_0', \delta', F'\rangle$ with labels from $\mathbb{L}^+$ by setting*

$$Q' = \{\, (q, N) \mid q \in Q_P,\ N \subseteq P(q) \,\} \quad \cup \\ \{\, (q, N, S, i) \mid q \in Q_O,\ N \subseteq P(q),\ S \in \mathsf{Sto}^{\mathsf{s}},\ i \in N \cup \{\text{-}\} \,\}$$

$q_0' = (q_0, \emptyset)$, $\rho_0' = \rho_0$, $F' = Q' \upharpoonright F$ and by defining δ' as follows.

$$\frac{q_P \xrightarrow{(\ell,\mathsf{c}_x)^S} q \quad \ell \in \mathbb{C} \cup \{\mathsf{kn}(j) \mid j \in O(q)\}}{(q_P, N) \xrightarrow{(\ell,\mathsf{c}_x)^{S\upharpoonright O(q)}} (q, N, S, \text{-})} \qquad \frac{q_O \xrightarrow{(\ell,\mathsf{r}_x)^S} q \quad \ell \notin \{\mathsf{kn}(j) \mid j \in P(q)\}}{(q_O, N, S', i) \xrightarrow{(\ell\langle S(i)\rangle,\mathsf{r}_x)^{S\upharpoonright O(q)}} (q, N)}\ {\scriptstyle S\upharpoonright(N\setminus\{i\}) = S'\upharpoonright(N\setminus\{i\}),\ i\in N}$$

$$\frac{q_P \xrightarrow{(\mathsf{kn}(j),\mathsf{c}_x)^S} q \quad j \in P(q)}{(q_P, N) \xrightarrow{(\circledast\langle S(j)\rangle,\mathsf{c}_x)^{S\upharpoonright O(q)}} (q, N, S, j)} \qquad \frac{q_O \xrightarrow{(\mathsf{kn}(j),\mathsf{r}_x)^S} q \quad j \in P(q)}{(q_O, N, S', j) \xrightarrow{(\circledast\langle S(j)\rangle,\mathsf{r}_x)^{S\upharpoonright O(q)}} (q, N)}\ {\scriptstyle S\upharpoonright(N\setminus\{j\}) = S'\upharpoonright(N\setminus\{j\})}$$

$$\frac{q_P \xrightarrow{(\mathsf{fr}(j),\mathsf{c}_x)^S} q}{(q_P, N) \xrightarrow{(\circledast\langle S(j)\rangle,\mathsf{c}_x)^{S\upharpoonright O(q)}} (q, N \cup \{j\}, S, j)} \qquad \frac{q_O \xrightarrow{(\ell,\mathsf{r}_x)^S} q \quad \ell \notin \{\mathsf{kn}(j) \mid j \in P(q)\}}{(q_O, N, S', \text{-}) \xrightarrow{(\ell,\mathsf{r}_x)^{S\upharpoonright O(q)}} (q, N)}\ {\scriptstyle S\upharpoonright N = S'\upharpoonright N}$$

$$\frac{q_P \xrightarrow{(\ell,\mathsf{r})^S} q \quad \ell \notin \{\mathsf{kn}(j) \mid j \in N\}}{(q_P, N) \xrightarrow{(\ell,\mathsf{r})^{S\upharpoonright O(q)\cup(P(q)\setminus N)}} (q, N, S, \text{-})} \qquad \frac{q_O \xrightarrow{(\ell,\mathsf{c})^S} q \quad \ell \notin \{\mathsf{kn}(j) \mid j \in N\}}{(q_O, N, S', \text{-}) \xrightarrow{(\ell,\mathsf{c})^{S\upharpoonright O(q)\cup(P(q)\setminus N)}} (q, N)}\ {\scriptstyle S\upharpoonright N = S'\upharpoonright N}$$

$$\frac{q_P \xrightarrow{(\mathsf{kn}(j),\mathsf{r})^S} q \quad j \in N}{(q_P, N) \xrightarrow{(\mathsf{fr}(j),\mathsf{r})^{S\upharpoonright O(q)\cup\{j\}\cup(P(q)\setminus N)}} (q, N \setminus \{j\}, S, \text{-})} \qquad \frac{q \xrightarrow{N'} q'}{(q, N, \dots) \xrightarrow{N'} (q', N \setminus N', \dots)}$$

$$\frac{q_P \xrightarrow{(\mathsf{kn}(j)/\mathsf{fr}(j),\mathsf{r}_\downarrow)^S} q}{(q_P, N) \xrightarrow{(\circledast\langle S(j)\rangle,\mathsf{r}_\downarrow)^{S\upharpoonright(O(q)\setminus\{j\})}} (q, N, S, j)} \qquad \frac{q_P \xrightarrow{(\ell,\mathsf{r}_\downarrow)^S} q \quad \ell \in \mathbb{C}}{(q_P, N) \xrightarrow{(\ell,\mathsf{r}_\downarrow)^{S\upharpoonright O(q)}} (q, N, S, \text{-})}$$

Let us write $[\![\Gamma \vdash \mathbb{C} : \theta]\!]_2^{\iota^{\Sigma_0}}$ for all the protoplays from $[\![\Gamma \vdash \mathbb{C} : \theta]\!]_2$ that begin with ι^{Σ_0}.

Lemma 31. *For any $\iota^{\Sigma_0} \in I^+_{\Gamma\vdash\theta}$, the automaton $\overline{(\!|\mathbb{C}|\!)^+_{\iota^{\Sigma_0}}}$ is non-blocking, strongly deterministic, non-overwriting and*[7] $\bigcup_{t\in\mathcal{L}(\overline{(\!|\mathbb{C}|\!)^+_{\iota^{\Sigma_0}}})} \mathsf{ext}(\iota^{\Sigma_0} t) = \mathsf{comp}([\![\Gamma \vdash \mathbb{C} : \theta]\!]_2^{\iota^{\Sigma_0}})$.

Proof. Note that the computation histories of the new automata have been obtained by restricting the automata obtained from Lemma 26 so that they trace out relevant plays only:

- O never uses P-names invisible to him thanks to rules[8] 4 (j in $\mathsf{kn}(j)$ has to match (q_O, N, S', j)) and 6 ($\ell \notin \{\mathsf{kn}(j) \mid j \in N\}$),

[7] Here $\mathsf{ext}(\cdots)$ will stand for the set of extensions with respect to *protoplays*.
[8] Counting from left to right, top to bottom.

– O will not change values referred to by P-names not available to him, because restrictions of the form $S \restriction \cdots = S' \restriction \cdots$ forbid that.

The labels generate protoplays, because of the $S \restriction \cdots$ restrictions on symbolic stores. The refreshments of Definition 14 are performed via rules 9 and 11 (as well as introducing $\circledast$ in rules 3 and 5, if the P-names were introduced with c_x tags). Consequently, the lemma follows from Lemma 26. □

From now on we shall assume that from each non-initial state of $\overline{(\!|\mathbb{C}|\!)^{+}_{\iota^{\Sigma_0}}}$ it is possible to reach a final state (if this is not the case, states violating this reachability requirement can be removed without affecting the above lemma). Note that because of strong determinacy, this implies that the automata will not have ϵ-cycles. This technical assumption will allow us to relate language equivalence to bisimulation in the next section.

7 Bisimulation

Here we define a notion of (weak) bisimilarity that will allow us to carry out the test from Theorem 16. Note that, given a term, our second translation to automata yields representatives for each complete protoplay in $[\![\cdots]\!]_2$. These representatives are by no means canonical, as can be seen below.

Example 32. The following terms are equivalent.

$$\begin{array}{l} f : \mathsf{unit} \to \mathsf{int}\,\mathsf{ref} \vdash\ f();f() \quad \cong \\ \qquad \mathsf{let}\ z = \mathsf{ref}\ (2)\ \mathsf{in}\ \mathsf{while}\ (!z)\ \mathsf{do}\ (f(); z := \mathsf{case}(!z)[0,0,1,\cdots]) : \mathsf{unit} \end{array}$$

The corresponding automata for $[\![\cdots]\!]_0$ (and $[\![\cdots]\!]_2$, which coincides with $[\![\cdots]\!]_0$ in this case) accept respectively the words given below.

$$\begin{array}{c} (\star,\mathsf{c}_f)^{\emptyset}(n_1,\mathsf{r}_f)^{(n_1,k_1)}(\star,\mathsf{c}_f)^{(n_1,k_1)}(n_2,\mathsf{r}_f)^{\{(n_1,k_1'),(n_2,k_2)\}}(\star,\mathsf{r}_\downarrow)^{\{(n_1,k_1'),(n_2,k_2)\}} \\ (\star,\mathsf{c}_f)^{\emptyset}(n_1,\mathsf{r}_f)^{(n_1,k_1)}(\star,\mathsf{c}_f)(n_2,\mathsf{r}_f)^{(n_2,k_2)}(\star,\mathsf{r}_\downarrow)^{\emptyset} \end{array}$$

The notion of bisimulation to be introduced aims to identify different representatives of identical protoplays by checking that they represent consistent store histories. First we define it on configuration graphs of automata. Let us say that that stores Σ_1, Σ_2 are *compatible*, written $\Sigma_1 \asymp \Sigma_2$, if $\Sigma_1 \cup \Sigma_2$ is a valid store (i.e. for all $a \in \mathsf{dom}(\Sigma_1) \cap \mathsf{dom}(\Sigma_2)$, $\Sigma_1(a) = \Sigma_2(a)$).

Definition 33. *Let $\mathcal{A}_i = \langle Q_i, q_{0i}, \rho_{0i}, \delta_i, F_i\rangle$ be automata of type θ, for $i = 1, 2$, and let us write F_i^ϵ for the set of states that can reach some final state by means of empty transitions. We call a relation $R \subseteq \hat{Q}_1 \times \mathsf{Sto}_{n_1+n_2} \times \hat{Q}_2$ a* simulation *on $\mathcal{A}_1$ and $\mathcal{A}_2$ if, for all $(\hat{q}_1, \Sigma, \hat{q}_2) \in R$,*

- *if $\pi_1(\hat{q}_1) \in F_1$ then $\pi_1(\hat{q}_2) \in F_2^\epsilon$, ($\pi_1$ the first-projection function);*
- *if $\hat{q}_1 \xrightarrow{(\ell,t)^{\Sigma_1}}_{\delta_1} \hat{q}_1'$ and $\pi_1(\hat{q}_1) \in Q_{1O}$ then either $\hat{q}_2 \xrightarrow{\epsilon}_{\delta_2} \hat{q}_2'$ and $(\hat{q}_1, \Sigma, \hat{q}_2') \in R$, or there is a finite $D \subseteq \mathbb{A}$ such that, for all $\Sigma_2 \asymp \Sigma_1$ with $\mathsf{dom}(\Sigma_2) = D$, there is some $\hat{q}_2 \xrightarrow{(\ell,t)^{\Sigma_2}}_{\delta_2} \hat{q}_2'$ with $(\hat{q}_1', \Sigma_1 \cup \Sigma_2, \hat{q}_2') \in R$;*

– *if* $\hat{q}_1 \xrightarrow{(\ell,t)^{\Sigma_1}}_{\delta_1} \hat{q}_1'$ *and* $\pi(\hat{q}_1) \in Q_{1P}$ *then either* $\hat{q}_2 \xrightarrow{\epsilon}_{\delta_2} \hat{q}_2'$ *and* $(\hat{q}_1, \Sigma, \hat{q}_2') \in R$*, or there is some* $\hat{q}_2 \xrightarrow{(\ell,t)^{\Sigma_2}}_{\delta_2} \hat{q}_2'$ *with* $(\hat{q}_1', \Sigma_1 \cup \Sigma_2, \hat{q}_2') \in R$ *and* $\Sigma_2 \asymp \Sigma_1 \cup (\Sigma \restriction (\mathsf{dom}(\Sigma) \setminus \mathsf{dom}(\Sigma_1)))$*;*
– *if* $\hat{q}_1 \xrightarrow{\epsilon}_{\delta_1} \hat{q}_1'$ *then* $(\hat{q}_1', \Sigma, \hat{q}_2) \in R$*.*

We call R *a* bisimulation *if both* R *and* R^{-1} *are simulations. We say that* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *are* bisimilar*, written* $\mathcal{A}_1 \sim \mathcal{A}_2$*, if there is a bisimulation* R *such that* $((q_{01}, \rho_{01}, \emptyset), \emptyset, (q_{02}, \rho_{02}, \emptyset)) \in R$*.*

Although bisimilarity is an infinite notion, we can capture it with a finite (and, hence, decidable) notion of *symbolic bisimilarity*, which relates states of the automata augmented with auxiliary finite structure, rather than configurations. This can be achieved by keeping a log of how the registers of the two automata are dynamically related, that is, which of their registers contain common names and which contain private ones. Transitions can then be simulated by referring to that log and updating it. For example, at the symbolic level, a transition of the form $q_1 \xrightarrow{(\mathsf{kn}(i),t)^{S_1}} q_1'$ of automaton $\mathcal{A}_1$ can be matched by $\mathcal{A}_2$ with $q_2 \xrightarrow{(\mathsf{kn}(j),t)^{S_2}} q_2'$ if we know that registers i and j are related, and symbolic stores S_1 and S_2 are equal at their related registers. If, however, our log tells us that i is private to $\mathcal{A}_1$, then $\mathcal{A}_2$ can only simulate it by $q_2 \xrightarrow{(\mathsf{fr}(j),t)^{S_2}} q_2'$ if q_1, q_2 are O-states.

Lemma 34. $\overline{(\!|\, \mathbb{C}_1 \,|\!)^+_{\iota^{\Sigma_0}}} \sim \overline{(\!|\, \mathbb{C}_2 \,|\!)^+_{\iota^{\Sigma_0}}}$ *iff* $\mathsf{comp}([\![\Gamma \vdash \mathbb{C}_1]\!]_2^{\iota^{\Sigma_0}}) = \mathsf{comp}([\![\Gamma \vdash \mathbb{C}_2]\!]_2^{\iota^{\Sigma_0}})$.

Proof. L2R: Take $s \in \mathsf{comp}([\![\Gamma \vdash \mathbb{C}_1]\!]_2^{\iota^{\Sigma_0}})$. By Lemma 31 there exists $t_1 \in \mathcal{L}(\overline{(\!|\, \mathbb{C}_1 \,|\!)^+_{\iota^{\Sigma_0}}})$ such that $s \in \mathsf{ext}(\iota^{\Sigma_0} t_1)$. Because $\overline{(\!|\, \mathbb{C}_1 \,|\!)^+_{\iota^{\Sigma_0}}} \sim \overline{(\!|\, \mathbb{C}_2 \,|\!)^+_{\iota^{\Sigma_0}}}$ we can find $t_2 \in \mathcal{L}(\overline{(\!|\, \mathbb{C}_2 \,|\!)^+_{\iota^{\Sigma_0}}})$ such that $s \in \mathsf{ext}(\iota^{\Sigma_0} t_2)$ by using s to resolve choices of missing store values in the corresponding bisimulation game. Hence $s \in \mathsf{comp}([\![\Gamma \vdash \mathbb{C}_2]\!]_2^{\iota^{\Sigma_0}})$. The other inclusion is symmetric.

R2L: Suppose $\overline{(\!|\, \mathbb{C}_1 \,|\!)^+_{\iota^{\Sigma_0}}} \not\sim \overline{(\!|\, \mathbb{C}_2 \,|\!)^+_{\iota^{\Sigma_0}}}$. Because the automata are deterministic, accept the same sets of extensions with respect to protoplays and final states can always be reached, there must exist $s \in \mathsf{comp}([\![\Gamma \vdash \mathbb{C}_1]\!]_2^{\iota^{\Sigma_0}}) = \mathsf{comp}([\![\Gamma \vdash \mathbb{C}_2]\!]_2^{\Sigma_0})$ and $j \in \{1, 2\}$ such that $s \in \mathsf{ext}(\iota^{\Sigma_0} t_j)$, where $t_j \in \mathcal{L}(\overline{(\!|\, \mathbb{C}_j \,|\!)^+_{\iota^{\Sigma_0}}})$, but $s \notin \mathsf{ext}(\iota^{\Sigma_0} t_{3-j})$ for any $t_{3-j} \in \mathcal{L}(\overline{(\!|\, \mathbb{C}_{3-j} \,|\!)^+_{\iota^{\Sigma_0}}})$, which contradicts Lemma 31. □

Theorem 35. *Program equivalence for* $\mathsf{RedML}^{\beta\to\beta}_{\mathrm{fin}}$ *terms is decidable.*

Proof. Let k be the number of equivalence classes of initial moves I_Γ with regard to name-permutability and let $\iota_0, \cdots, \iota_k$ be their representatives. Then it suffices to verify $\overline{(\!|\, \mathbb{C}_1 \,|\!)^+_{\iota^{\Sigma_0}}} \sim \overline{(\!|\, \mathbb{C}_2 \,|\!)^+_{\iota^{\Sigma_0}}}$ for all $\iota = \iota_0, \cdots, \iota_k$ and all possible stores Σ_0 with domain $\mathcal{L}_\iota$. Altogether only finitely many bisimulation queries need to be made. □

8 Further work

It would be desirable to understand what other models over infinite alphabets are suitable for representing $\mathsf{RedML}_{\mathrm{fin}}$-terms featuring more complicated types. For instance, it seems that a variant of pushdown automata would be needed to capture terms of type $\vdash (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}$. Another interesting avenue for future work concerns investigating relationships between automata over infinite alphabets and history-dependent automata [9].

References

1. S. Abramsky and G. McCusker. Call-by-value games. In *Proc. of CSL*, LNCS 1414, pp 1–17. Springer-Verlag, 1997.
2. A. Ahmed, D. Dreyer, and A. Rossberg. State-dependent representation independence. In *Proc. of POPL*, pp 340–353. ACM, 2009.
3. M. Bojańczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *Proceedings of LICS*, pp 7–16, 2006.
4. D. Dreyer, G. Neis, and L. Birkedal. The impact of higher-order state and control effects on local relational reasoning. In *Proc. of ICFP*, pp 143–156. ACM, 2010.
5. M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2002.
6. D. R. Ghica. Regular-language semantics for a call-by-value programming language. In *Proc. of MFPS*, ENTCS 45. Elsevier, 2001.
7. M. Kaminski and N. Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.
8. G. McCusker. On the semantics of Idealized Algol without the bad-variable constructor. In *Proc. of MFPS*, ENTCS 83. Elsevier, 2003.
9. U. Montanari and M. Pistore. An introduction to history dependent automata. ENTCS 10, 1997.
10. A. S. Murawski. Functions with local state: regularity and undecidability. *Theor. Comput. Sci.*, 338(1/3):315–349, 2005.
11. A. S. Murawski, C.-H. L. Ong, and I. Walukiewicz. Idealized Algol with ground recursion and DPDA equivalence. In *Proc. of ICALP*, LNCS 3580, pp. 917–929. Springer, 2005.
12. A. S. Murawski and N. Tzevelekos. Full abstraction for Reduced ML. In *Proc. of FOSSACS*, LNCS 5504, pp 32–47. Springer-Verlag, 2009.
13. A. S. Murawski and N. Tzevelekos. Block structure vs scope extrusion: between innocence and omniscience. In *Proc. of FOSSACS*, LNCS 6014, pp 33–47. Springer-Verlag, 2010.
14. F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.
15. A. M. Pitts and I. D. B. Stark. Operational reasoning for functions with local state. In A. D. Gordon and A. M. Pitts, editors, *Higher-Order Operational Techniques in Semantics*, pages 227–273. Cambridge University Press, 1998.
16. I. D. B. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge Computing Laboratory, 1995. Technical Report No. 363.
17. N. Tzevelekos. Fresh-register automata. In *Proceedings of POPL*. ACM, 2011.