Open access • Book Chapter • DOI:10.1007/978-3-642-31585-5_30

Algorithmic games for full ground references — Source link

Andrzej S. Murawski, Nikos Tzevelekos

Institutions: University of Leicester, Queen Mary University of London

Published on: 09 Jul 2012 - International Colloquium on Automata, Languages and Programming

Topics: Decidability, Undecidable problem, Pushdown automaton, Finitary and Game semantics

Form Methods Syst Des
DOI 10.1007/s10703-017-0292-9

Algorithmic games for full ground references

Andrzej S. Murawski[1] · Nikos Tzevelekos[2]

Abstract We present a full classification of decidable and undecidable cases for contextual equivalence in a finitary ML-like language equipped with full ground storage (both integers and reference names can be stored). The simplest undecidable type is $\mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}$. At the technical level, our results marry game semantics with automata-theoretic techniques developed to handle infinite alphabets. On the automata-theoretic front, we show decidability of the emptiness problem for register pushdown automata extended with fresh-symbol generation.

Keywords Program equivalence · Game semantics · Full abstraction · Automata over infinite alphabets

1 Introduction

Mutable variables in which numerical values can be stored for future access and update are the pillar of imperative programming. The memory in which the values are deposited can be allocated statically, typically to coincide with the lifetime of the defining block, or dynamically, on demand, with the potential to persist forever. In order to support memory management, modern programming languages feature mechanisms such as *pointers* or *references*, which allow programmers to access memory via addresses. Languages like C (through `int*`) or ML (via `int ref ref`) make it possible to store the addresses themselves, which creates the need for storing references to references etc. We refer to this scenario as *full ground storage*. In this paper we study an ML-like language GRef with full ground storage, which permits the creation of references to integers as well as references to integer references, and so on.

✉ Andrzej S. Murawski
a.murawski@warwick.ac.uk

[1] University of Warwick, Coventry, UK

[2] Queen Mary University of London, London, UK

Published online: 09 August 2017

We concentrate on contextual equivalence[1] in that setting. Reasoning about program equivalence has been a central topic in programming language semantics since its inception. This is in no small part due to important applications, such as verification problems (equivalence between a given implementation and a model implementation) and compiler optimization (equivalence between the original program and its transform). Specifically, we attack the problem of automated reasoning about our language in a finitary setting, with finite datatypes and with looping instead of recursion, where decidability questions become interesting and the decidability/undecidability frontier can be identified. In particular, it is possible to quantify the impact of higher-order types on decidability, which goes unnoticed in Turing-complete frameworks.

The paper presents a complete classification of cases in which GRef program equivalence is decidable. The result is phrased in terms of the syntactic shape of types. We write $\theta_1, \cdots, \theta_k \vdash \theta$ to refer to the problem of deciding contextual equivalence between two terms M_1, M_2 such that $x_1 : \theta_1, \cdots, x_m : \theta_m \vdash M_i : \theta$ $(i = 1, 2)$. We investigate the problem using a fully abstract game model of GRef.[2] Such a model can be easily obtained by modifying existing models of more general languages, e.g. by either adding type information to Laird's model of untyped references [19] or trimming down our own model for general references [24]. The models are *nominal* in that moves may involve elements from an infinite set of *names* to account for reference names. Additionally, each move is equipped with a store whose domain consists of all names that have been revealed (played) thus far and the corresponding values. Note that values of reference types also become part of the domain of the store. This representation grows as the play unfolds and new names are encountered. We shall rely on the model both for decidability and undecidability results. Our work identifies the following undecidable cases as minimal.

$$\vdash \mathsf{unit} \to \mathsf{unit} \to \mathsf{unit} \qquad (\mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit} \vdash \mathsf{unit}$$
$$\vdash ((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit} \qquad (((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit} \vdash \mathsf{unit}$$

Obviously, undecidability extends to typing judgments featuring syntactic supertypes of those listed above (for instance, when fourth-order types appear on the left-hand side of the turnstile or types of the shape $\theta_1 \to \theta_2 \to \theta_3$ occur on the right). The remaining cases are summarized by typing judgements in which each of $\theta_1, \cdots, \theta_m$ is generated by the grammar given on the left below, and θ by the grammar on the right,

$$\Theta_L ::= \beta \mid \Theta_R \to \Theta_L \qquad \Theta_R ::= \beta \mid \Theta_1 \to \beta$$

where β stands any ground type and Θ_1 is a first-order type, i.e. $\beta ::= \mathsf{unit} \mid \mathsf{int} \mid \mathsf{ref}^i\ \mathsf{int}$ and $\Theta_1 ::= \beta \mid \beta \to \Theta_1$. We shall show that all these cases are in fact decidable. In order to arrive at a decision procedure we rely on effective reducibility to a canonical (β-normal) form. These forms are then inductively translated into a class of automata over infinite alphabets that represent the associated game semantics. Finally, we show that the representations can be effectively compared for equivalence.

The automata we use are especially designed to read moves-with-stores in a single computational step. They are equipped with a finite set of registers for storing elements from

[1] Two program phrases are regarded as *contextually equivalent*, or simply *equivalent*, if they can be used interchangeably in any context without affecting the observable outcome.

[2] A model is *fully abstract* if it captures contextual equivalence denotationally, i.e. equivalence can be confirmed/disproved by reference to the interpretations of terms.

the infinite alphabet (names). Moreover, in a single transition step, the content of a subset of registers can be pushed onto the stack (along with a symbol from the stack alphabet), to be popped back at a later stage. We use visibly pushdown stacks [3], i.e. the alphabet can be partitioned into letters that consistently trigger the same stack actions (push, pop or no-op). Conceptually, the automata extend register pushdown automata [7] with the ability to generate fresh names, as opposed to their existing capability to generate names not currently present in registers. Crucially, we can show that the emptiness problem for the extended machine model remains decidable.

Because the stores used in game-semantic plays can grow unboundedly, one cannot hope to construct the automata in such a way that they will accept the full game semantics of terms. Instead we construct automata that, without loss of generality, will accept plays in which the domains of stores are bounded in size. Each such restricted play can be taken to represent a *set* of real plays compatible with the representation. Compatibility means that values of names omitted in environment-moves (O-moves) can be filled in arbitrarily, but values of names omitted in program-moves (P-moves) must be the same as in preceding O-moves. That is to say, the omissions leading to bounded representation correspond to copy-cat behaviour.

Because we work with representations of plays, we cannot simply use off-the-shelf procedures for checking program equivalence, as the same plays can be represented in different ways: copy-cat behaviour can be modelled explicitly or implicitly via the convention. However, taking advantage of the fact that stacks of two visibly pushdown automata over the same partitioning of the alphabet can be synchronized, we show how to devise another automaton that can run automata corresponding to two terms in parallel and detect inconsistencies in the representations of plays. Exploiting decidability of the associated emptiness problem, we can conclude that GRef program equivalence in the above-mentioned cases is decidable.

This article is the journal version of [25], with full proofs and rearrangement of the material. Also, we relate our work to what has been done after the conference version was published. We start by introducing the language GRef in Sect. 2 along with a canonical form result. Then, in Sect. 3 we introduce the game model of GRef (we refer the interested reader to [28] for a detailed account of the model, which is in effect a restricted version of game models for larger languages presented in [19,24]). The first main results are obtained in Sect. 4 and concern undecidability of equivalence in specific fragments of GRef via reductions to queue machine problems. Finally, in Sect. 5 we present a decidability procedure for equivalence in the remaining fragment GRef☺. The argument implements a reduction to checking non-emptiness in a specific kind of automata over infinite alphabets, the decidability of which is proved in Sect. 5.4.

2 GRef

We work with a finitary ML-like language GRef whose types θ are generated according to the following grammar.

$$\theta ::= \beta \mid \theta \to \theta \qquad \beta ::= \mathsf{unit} \mid \gamma \qquad \gamma ::= \mathsf{int} \mid \mathsf{ref}\,\gamma$$

Note that reference types are available for each type of the shape γ (full ground storage). The language is best described as the call-by-value λ-calculus over the ground types β augmented with finitely many constants, do-nothing command, case distinction, looping, and reference manipulation (allocation, dereferencing, assignment). The typing rules are:

$$\frac{}{\Gamma \vdash () : \mathsf{unit}} \quad \frac{i \in \{0, \cdots, max\}}{\Gamma \vdash i : \mathsf{int}} \quad \frac{(x:\theta) \in \Gamma}{\Gamma \vdash x : \theta} \quad \frac{\Gamma \vdash M : \mathsf{int} \quad \Gamma \vdash N : \mathsf{unit}}{\Gamma \vdash \mathsf{while}\, M\, \mathsf{do}\, N : \mathsf{unit}}$$

$$\frac{\Gamma \vdash M : \mathsf{int} \quad \Gamma \vdash N_0 : \theta \quad \cdots \quad \Gamma \vdash N_{max} : \theta}{\Gamma \vdash \mathsf{case}(M)[N_0, \cdots, N_{max}] : \theta} \quad \frac{\Gamma \vdash M : \theta \to \theta' \quad \Gamma \vdash N : \theta}{\Gamma \vdash MN : \theta'}$$

$$\frac{\Gamma \cup \{x:\theta\} \vdash M : \theta'}{\Gamma \vdash \lambda x^{\theta}.M : \theta \to \theta'} \quad \frac{\Gamma \vdash M : \gamma}{\Gamma \vdash \mathsf{ref}_{\gamma}(M) : \mathsf{ref}\,\gamma} \quad \frac{\Gamma \vdash M : \mathsf{ref}\,\gamma}{\Gamma \vdash !M : \gamma} \quad \frac{\Gamma \vdash M : \mathsf{ref}\,\gamma \quad \Gamma \vdash N : \gamma}{\Gamma \vdash M := N : \mathsf{unit}}$$

In what follows, we write $M; N$ for the term $(\lambda z^{\theta}.N)M$, where z does not occur in N and θ matches the type of M. $\mathsf{let}\, x = M\, \mathsf{in}\, N$ will stand for $(\lambda x^{\theta}.N)M$ in general. The operational semantics of the language can be found in [24,28]. Note that, assuming $max > 0$, reference equality is expressible in the above syntax [29].

Definition 1 Given $\vdash M : \mathsf{unit}$, we write $M\Downarrow$ if M evaluates to $()$. We say that the term-in-context $\Gamma \vdash M_1 : \theta$ ***approximates*** $\Gamma \vdash M_2 : \theta$ (written $\Gamma \vdash M_1 \lesssim M_2$) if $C[M_1]\Downarrow$ implies $C[M_2]\Downarrow$ for any context $C[-]$ such that $\vdash C[M_1], C[M_2] : \mathsf{unit}$. Two terms-in-context are ***equivalent*** if one approximates the other (written $\Gamma \vdash M_1 \cong M_2$).

We next present a canonical form result for GRef terms. Its use will become apparent when demonstrating decidability of equivalence in the fragment GRef☺ of GRef, in Sect. 5. The grammar below defines a notion of canonical form for GRef terms.

$$\begin{aligned}\mathbb{C} ::= {} & () \mid i \mid x^{\mathsf{ref}\,\gamma} \mid \lambda x^{\theta}.\mathbb{C} \mid \mathsf{case}(x^{\mathsf{int}})[\mathbb{C}, \cdots, \mathbb{C}] \mid (\mathsf{while}\,(!x^{\mathsf{ref\,int}})\,\mathsf{do}\,\mathbb{C}); \mathbb{C} \\ & \mid \mathsf{let}\, y^{\gamma} = !x^{\mathsf{ref}\,\gamma}\, \mathsf{in}\, \mathbb{C} \mid (x^{\mathsf{ref\,int}} := i); \mathbb{C} \mid (x^{\mathsf{ref}^2\gamma} := y^{\mathsf{ref}\,\gamma}); \mathbb{C} \\ & \mid \mathsf{let}\, x^{\mathsf{ref\,int}} = \mathsf{ref}(0)\, \mathsf{in}\, \mathbb{C} \mid \mathsf{let}\, x^{\mathsf{ref}^2\gamma} = \mathsf{ref}(y^{\mathsf{ref}\,\gamma})\, \mathsf{in}\, \mathbb{C} \mid \mathsf{let}\, y = z\,()\, \mathsf{in}\, \mathbb{C} \\ & \mid \mathsf{let}\, y = z\, i\, \mathsf{in}\, \mathbb{C} \mid \mathsf{let}\, y = z\, x^{\mathsf{ref}\,\gamma}\, \mathsf{in}\, \mathbb{C} \mid \mathsf{let}\, y = z\,(\lambda x^{\theta}.\mathbb{C})\, \mathsf{in}\, \mathbb{C}\end{aligned}$$

Lemma 2 *Let $\Gamma \vdash M : \theta$ be an* GRef*-term. There exists a* GRef*-term $\Gamma \vdash \mathbb{C}_M : \theta$ in canonical form, effectively constructible from M, such that $\Gamma \vdash M \cong \mathbb{C}_M$.*

3 Game semantics

Game semantics views computation as a dialogue between the environment (Opponent, O) and the program (Proponent, P). We give an overview of the fully abstract game model of GRef [28]. Let $\mathbb{A} = \biguplus_{\gamma} \mathbb{A}_{\gamma}$ be a collection of countably infinite sets of *reference names*, or just *names*. The model is constructed using mathematical objects (moves, plays, strategies) that will feature names drawn from $\mathbb{A}$. Although names underpin various elements of our model, their precise nature is irrelevant. Hence, all of our definitions preserve name-invariance, i.e. our objects are (strong) *nominal sets* [11,32]. Note that we do not need the full power of the theory but mainly the basic notion of name-permutation. For an element x belonging to a (nominal) set X, we write $\nu(x)$ for its name-support, i.e. the set of names occurring in x. Moreover, for any $x, y \in X$, we write $x \sim y$ if x and y are the same up to a permutation of $\mathbb{A}$. Our model is couched in the Honda-Yoshida style of modelling call-by-value computation [13]. Before we define what it means to play our games, let us introduce the auxiliary concept of an arena. Q and A are used to distinguish question- and answer-moves respectively.

Definition 3 An *arena* $A = \langle M_A, I_A, \lambda_A, \vdash_A \rangle$ is given by a set M_A of moves, its subset I_A of initial ones, a labelling function $\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$ and a justification relation $\vdash_A \subseteq M_A \times (M_A \backslash I_A)$. These satisfy, for each $m, m' \in M_A$, the conditions:

- $m \in I_A \implies \lambda_A(m) = (P, A)$,
- $m \vdash_A m' \wedge \lambda_A^{QA}(m) = A \implies \lambda_A^{QA}(m') = Q$,
- and $m \vdash_A m' \implies \lambda_A^{OP}(m) \neq \lambda_A^{OP}(m')$.

where we write λ_A^{OP} and λ_A^{QA} for λ_A post-composed with the first and second projections respectively.

We shall use i to refer to initial moves. Let $\bar{\lambda}_A$ be the OP-complement of λ_A. Given arenas A, B, the arenas $A \otimes B$ and $A \Rightarrow B$ are constructed as below, where $\bar{I}_A = M_A \backslash I_A$, $\bar{\vdash}_A = (\vdash_A \restriction \bar{I}_A \times \bar{I}_A)$ (and similarly for B).

$$\begin{aligned}
M_{A \Rightarrow B} &= \{\star\} \uplus M_A \uplus M_B & \lambda_{A \Rightarrow B} &= \big[\star \mapsto PA,\, \bar{\lambda}_A[\mathrm{i}_A \mapsto OQ],\, \lambda_B\big] \\
I_{A \Rightarrow B} &= \{\star\} & \vdash_{A \Rightarrow B} &= \{(\star, \mathrm{i}_A), (\mathrm{i}_A, \mathrm{i}_B)\} \cup \vdash_A \cup \vdash_B \\
M_{A \otimes B} &= (I_A \times I_B) \uplus \bar{I}_A \uplus \bar{I}_B & \lambda_{A \otimes B} &= \big[(\mathrm{i}_A, \mathrm{i}_B) \mapsto PA,\, \lambda_A \restriction \bar{I}_A,\, \lambda_B \restriction \bar{I}_B\big] \\
I_{A \otimes B} &= I_A \times I_B & \vdash_{A \otimes B} &= \{((\mathrm{i}_A, \mathrm{i}_B), m) \mid \mathrm{i}_A \vdash_A m \vee \mathrm{i}_B \vdash_B m\} \cup \bar{\vdash}_A \cup \bar{\vdash}_B
\end{aligned}$$

Let us write $[i, j]$ for the set $\{i, i+1, \cdots, j\}$. For each type θ we can define the corresponding arena $[\![\theta]\!]$.

$$\begin{aligned}
[\![\mathsf{unit}]\!] &= \langle \{\star\}, \{\star\}, \emptyset, \emptyset \rangle & [\![\mathsf{int}]\!] &= \langle [0, max], [0, max], \emptyset, \emptyset \rangle \\
[\![\mathsf{ref}\, \gamma]\!] &= \langle \mathbb{A}_\gamma, \mathbb{A}_\gamma, \emptyset, \emptyset \rangle & [\![\theta \to \theta']\!] &= [\![\theta]\!] \Rightarrow [\![\theta']\!]
\end{aligned}$$

Although types are interpreted by arenas, the actual games will be played in *prearenas*, which are defined in the same way as arenas with the exception that initial moves are O-questions. Given arenas A, B we define the prearena $A \to B$ as follows.

$$\begin{aligned}
M_{A \to B} &= M_A \uplus M_B & \lambda_{A \to B} &= [\bar{\lambda}_A[\mathrm{i}_A \mapsto OQ], \lambda_B] \\
I_{A \to B} &= I_A & \vdash_{A \to B} &= \{(\mathrm{i}_A, \mathrm{i}_B)\} \cup \vdash_A \cup \vdash_B
\end{aligned}$$

A ***store*** is a type-sensitive finite partial function $\Sigma : \mathbb{A} \rightharpoonup [0, max] \cup \mathbb{A}$ such that $a \in \mathsf{dom}(\Sigma) \cap \mathbb{A}_{\mathsf{int}}$ implies $\Sigma(a) \in [0, max]$, and $a \in \mathsf{dom}(\Sigma) \cap \mathbb{A}_{\mathsf{ref}\,\gamma}$ implies $\Sigma(a) \in \mathsf{dom}(\Sigma) \cap \mathbb{A}_\gamma$. We write Sto for the set of all stores. A move-with-store on a (pre)arena A is a pair m^Σ with $m \in M_A$ and $\Sigma \in \mathsf{Sto}$.

Definition 4 A *justified sequence* on a prearena A is a sequence of moves-with-store on A such that, apart from the first move, which must be of the form i^Σ with $\mathrm{i} \in I_A$, every move $n^{\Sigma'}$ in s is equipped with a pointer to an earlier move m^Σ such that $m \vdash_A n$. m is then called the justifier of n, which is represented as $\cdots \overset{\frown}{m^\Sigma \cdots n} \cdots$ in drawings.

For each $S \subseteq \mathbb{A}$ and Σ we define $\Sigma^0(S) = S$ and $\Sigma^{i+1}(S) = \Sigma(\Sigma^i(S)) \cap \mathbb{A}$ $(i \geq 0)$. Let $\Sigma^*(S) = \bigcup_i \Sigma^i(S)$. The set of *available names* of a justified sequence is defined inductively by $\mathsf{Av}(\epsilon) = \emptyset$ and $\mathsf{Av}(sm^\Sigma) = \Sigma^*(\mathsf{Av}(s) \cup \nu(m))$. The *view* of a justified sequence is defined by:

$$view(\epsilon) = \epsilon$$

$$view(m^\Sigma) = m^\Sigma$$

$$view(s\, \overset{\frown}{m^\Sigma\, t\, n}{}^{\Sigma'}) = view(s)\, m^\Sigma n^{\Sigma'}$$

We shall write $s \sqsubseteq s'$ to mean that s is a prefix of s'.

Definition 5 Let A be a prearena. A justified sequence s on A is called a ***play***, if it satisfies the conditions below.

- No adjacent moves belong to the same player (*Alternation*).
- The justifier of each answer is the most recent unanswered question (*Bracketing*).
- For any $s'm^{\Sigma} \sqsubseteq s$ with non-empty s', the justifier of m occurs in $\mathit{view}(s')$ (*Visibility*).
- For any $s'm^{\Sigma} \sqsubseteq s$, $\mathsf{dom}(\Sigma) = \mathsf{Av}(s'm^{\Sigma})$ (*Frugality*).

Definition 6 A ***strategy*** σ on a prearena A, written $\sigma : A$, is a set of even-length plays of A satisfying:

- If $so^{\Sigma}p^{\Sigma'} \in \sigma$ then $s \in \sigma$ (*Even-prefix closure*).
- If $s \in \sigma$ and $s \sim t$ then $t \in \sigma$ (*Equivariance*).
- If $s_1 p_1^{\Sigma_1}, s_2 p_2^{\Sigma_2} \in \sigma$ and $s_1 \sim s_2$ then $s_1 p_1^{\Sigma_1} \sim s_2 p_2^{\Sigma_2}$ (*Nominal determinacy*).

GRef-terms $\Gamma \vdash M : \theta$, where $\Gamma = \{x_1 : \theta_1, \cdots, x_n : \theta_n\}$, are interpreted by strategies for the prearena $[\![\theta_1]\!] \otimes \cdots \otimes [\![\theta_n]\!] \to [\![\theta]\!]$, which we shall denote by $[\![\Gamma \vdash \theta]\!]$. Given a set of plays X, let us write $\mathsf{comp}(X)$ for the set of complete plays in X, i.e. those in which each occurrence of a question justifies an answer. The interpretation is then fully abstract in the following sense.

Proposition 7 *([19,24,28]) Let $\Gamma \vdash M_1, M_2 : \theta$ be GRef-terms. $\Gamma \vdash M_1 \lesssim M_2$ if, and only if, $\mathsf{comp}([\![\Gamma \vdash M_1 : \theta]\!]) \subseteq \mathsf{comp}([\![\Gamma \vdash M_2 : \theta]\!])$. Hence, $\Gamma \vdash M_1 \cong M_2$ if, and only if, $\mathsf{comp}([\![\Gamma \vdash M_1 : \theta]\!]) = \mathsf{comp}([\![\Gamma \vdash M_2 : \theta]\!])$.*

We shall rely on the result for proving both undecidability and decidability results, by referring to complete plays generated by terms.

Example 8 The name-generating term $\vdash \lambda x^{\mathsf{unit}}.\mathsf{ref}(0) : \mathsf{unit} \to \mathsf{ref\,int}$ yields complete plays of the shape given below (the corresponding prearena is given on the right).

$$\begin{tikzcd}
q & O \\
\star \arrow[u, dash] & P \\
q_0 \arrow[u, dash] & O \\
a \arrow[u, dash] & P
\end{tikzcd}$$

$$\begin{tikzcd}
q^{\emptyset} & \star^{\emptyset} \arrow[l, bend right] & q_0^{\Sigma'_0} \arrow[l, bend right] & a_0^{\Sigma_1} \arrow[l, bend right] & \cdots & q_0^{\Sigma'_{i-1}} \arrow[llll, bend right] & a_i^{\Sigma_i} \arrow[l, bend right] & q_0^{\Sigma'_i} \arrow[llllll, bend right] & \cdots
\end{tikzcd}$$

where $\Sigma'_0 = \emptyset$ and, for all $i > 0$, $\Sigma_i = \Sigma'_{i-1} \cup \{(a_i, 0)\}$, $\mathsf{dom}(\Sigma'_i) = \mathsf{dom}(\Sigma_i)$. Moreover, for any $i \neq j$ we have $a_i \neq a_j$. 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.

Note that in the above example the sizes of stores keep on growing indefinitely. However, the essence of the strategy is already captured by plays of the shape $q \star q_0\, a_0^{(a_0,0)} \cdots q_0\, a_i^{(a_i,0)} q_0 \cdots$ under the assumption that, whenever a value is missing from the store of an O-move, it is arbitrary and, for P-moves, it is the same as in the preceding O-move. Next we spell out how a sequence of moves-with-store, not containing enough information to qualify as a play, can be taken to represent proper plays.

Definition 9 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 $\Theta_i \subseteq \Sigma_i$ $(1 \le i \le k)$ and, for any $1 \le i \le \lfloor k/2 \rfloor$, if $a \in \mathsf{dom}(\Sigma_{2i}) \backslash \mathsf{dom}(\Theta_{2i})$ then $\Sigma_{2i}(a) = \Sigma_{2i-1}(a)$. We write $\mathsf{ext}(t)$ for the set of all extensions of t.

Because we cannot hope to encode plays with unbounded stores through automata, our decidability results will be based on representations of plays that capture strategies via extensions.

4 Undecidability arguments

We begin with undecidable cases. Our argument will rely on queue machines, which are finite-state devices equipped with a queue.

Definition 10 Let $\mathcal{A}$ be a finite alphabet. A queue machine over $\mathcal{A}$ is specified by $\langle Q, Q_E, Q_D, init, \delta_E, \delta_D \rangle$, where Q is a finite set of states such that $Q = Q_E \uplus Q_D$, $init \in Q_E$ is the initial state, $\delta_E : Q_E \to Q \times \mathcal{A}$ is the enqueuing function, whereas $\delta_D : Q_D \times \mathcal{A} \to Q$ is the dequeuing function.

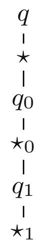
A queue machine starts at state *init* with an empty queue. Whenever it reaches a state $q \in Q_E$, it will progress to the state $\pi_1(\delta_E(q))$ and $\pi_2(\delta_E(q))$ will be added to the associated queue, where π_1, π_2 are the first and second projections respectively. If the machine reaches a state $q \in Q_D$ and its queue is empty, the machine is said to *halt*. Otherwise, it moves to the state $\delta_D(q, x)$, where x is the symbol at the head of the associated queue, which is then removed from the queue. The halting problem for queue machines is well known to be undecidable (e.g. [17]). By encoding computation histories of queue machines as plays generated by GRef terms we next show that the equivalence problem for GRef terms must be undecidable. Note that this entails undecidability of the associated notion of term approximation.

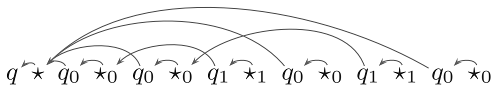
Theorem 11 *The contextual equivalence problem is undecidable in the following cases (even in absence of looping).*

- $\vdash M_1 \cong M_2 : \mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}$
- $f : (\mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit} \vdash M_1 \cong M_2 : \mathsf{unit}$
- $f : (((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit} \vdash M_1 \cong M_2 : \mathsf{unit}$
- $\vdash M_1 \cong M_2 : ((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit}$

In the following we prove undecidability in each of the cases of Theorem 11.
$\vdash \mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}$: We first sketch the argument. The arena used to interpret closed terms of type $\mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}$ has the shape given on the right.

We are going to use plays from the arena to represent sequences of queue operations. Enqueuing will be represented by segments of the form $q_0 \star_0$, whereas $q_1 \star_1$ will be used to represent dequeuing. Additionally, in the latter case q_1 will be justified by $\star_0$ belonging to the segment representing the enqueuing of the element that is now being dequeued. For instance, the sequence $EEDEDE$, in which E, D stand for enqueing and dequeing respectively, will be represented as follows.





Note that all such plays are complete. Given a queue machine $\mathbb{Q}$, let us write $\mathsf{hist}(\mathbb{Q})$ for the (prefix-closed) subset of $(E \uplus D)^*$ corresponding to all sequences of queue operations performed by $\mathbb{Q}$. Note that $\mathsf{hist}(\mathbb{Q})$ is finite if and only if $\mathbb{Q}$ halts. Additionally, define $\mathsf{hist}^-(\mathbb{Q})$ to be $\mathsf{hist}(\mathbb{Q})$ from which the longest sequence is removed (if $\mathsf{hist}(\mathbb{Q})$ is infinite and the sequence in question does not exist we set $\mathsf{hist}^-(\mathbb{Q}) = \mathsf{hist}(\mathbb{Q})$). Note that the sequence corresponds to a terminating run and necessarily ends in D.

Lemma 12 *Let* $\mathbb{Q}$ *be a queue machine. There exist* GRef *terms* $\vdash M, M^- : \mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}$ *such that* $\mathsf{comp}([\![M]\!])$, $\mathsf{comp}([\![M^-]\!])$ *represent* $\mathsf{hist}(\mathbb{Q})$, $\mathsf{hist}^-(\mathbb{Q})$ *respectively.*

Proof WLOG we shall assume that Q can be fitted into int (otherwise, we could use a fixed number of variables to achieve the desired storage capacity). Let $D[-] \equiv C[\lambda x.C_0[\lambda y.C_1[-]]]$, where $C[-]$, $C_0[-]$, $C_1[-]$ are given in Fig. 1 ($*$ is a special symbol not in the queue alphabet and Ω is a canonical divergent term). $C_0[-]$ and $C_1[-]$ handle enqueuing and dequeuing respectively. We take

$$M^- \equiv D[\mathsf{if}\ (!STATE \in Q_D \wedge !!LAST = *)\ \mathsf{then}\ \Omega]$$

and $M \equiv D[()]$.

Note that there are only three moves that O can play: q, q_0 and q_1. After the initial q, P must follow with $\star$ thanks to $C[-]$, which will not cause divergence. Note that it declares the variable $STATE$ (initialized to $init$), whose scope spans over the whole term and which will be updated at each step to mimic the state of $\mathbb{Q}$. After q is played, it can never be played again, but O can still play q_0 or q_1. These are handled by $C_0[-]$ and $C_1[-]$ respectively.

$$M^- \equiv D[\mathsf{if}\ (!STATE \in Q_D \wedge !!LAST = *)\ \mathsf{then}\ \Omega]$$

$$\begin{aligned} C[-] = {} & \mathsf{let}\ STATE = \mathsf{ref}(init)\ \mathsf{in} \\ & \mathsf{let}\ LAST = \mathsf{ref}(\mathsf{ref}(*))\ \mathsf{in}\ [-] \end{aligned}$$

$$\begin{aligned} C_0[-] = {} & \mathsf{if}\ (!STATE \notin Q_E)\ \mathsf{then}\ \Omega; \\ & STATE := \pi_1\delta_E(!STATE); \\ & \mathsf{let}\ SYM = \mathsf{ref}(\pi_2\delta_E(!STATE))\ \mathsf{in} \\ & \mathsf{let}\ PREV = \mathsf{ref}(!LAST)\ \mathsf{in} \\ & \quad LAST := SYM;\ [-] \end{aligned}$$

$$\begin{aligned} C_1[-] = {} & \mathsf{if}\ (!STATE \notin Q_D)\ \mathsf{then}\ \Omega; \\ & \mathsf{if}\ (!!PREV \neq * \vee !SYM = *)\ \mathsf{then}\ \Omega; \\ & STATE := \delta_D(!STATE, !SYM); \\ & SYM := *;\ [-] \end{aligned}$$

Fig. 1 Simulating a queue machine in $\vdash \mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}$. The variable $STATE : \mathsf{ref\,int}$ contains the current state of the machine. The queue is encoded as a backwards-connected list with elements $(PREV, SYM) : \mathsf{ref}^2\,\mathsf{int} \times \mathsf{ref\,int}$, with last-element pointer $LAST : \mathsf{ref}^2\,\mathsf{int}$. Enqueuing adds a new last element while dequeuing sets the first non-$*$ symbol of the list to $*$

- If O plays q_0 when $\mathbb{Q}$ is not able to enqueue, P will not respond. This is caused by the condition $!\mathit{STATE} \notin Q_E$ in $C_0[-]$. However, if $\mathbb{Q}$ is in enqueuing mode, *local* references *SYM* and *PREV* will be created. *SYM* is initialized to the symbol that $\mathbb{Q}$ will add to the queue. *PREV* contains the name of the reference cell in which the previously enqueued symbol was stored (as soon as the symbol is dequeued, the value stored in that cell will be set to $*$).
  The "global" reference *LAST* (of type $\mathsf{ref}(\mathsf{ref\,int})$) is used to pass the name from one $q_0\star_0$ segment to the next. Hence, the current value of *SYM* is written to *LAST* as soon as the previous value of *LAST* got recorded in *PREV*. The assignment is followed by the value $\lambda y.C_3[-]$, so P will respond with $\star_0$.
- If O plays q_1 in an enqueuing state, P will not respond due to the $!\mathit{STATE} \notin Q_D$ check in $C_1[-]$. Furthermore, P will not reply when
  - q_1 is justified by $\star_0$ from a block corresponding to an element that has already been taken off the queue ($!\mathit{SYM} = *$);
  - q_1 is justified by $\star_0$ from a block corresponding to elements that are still present in the queue, but do not occur at its head ($!!\mathit{PREV} \neq *$).

  Otherwise (i.e. if O plays q_1 and justifies it with $\star_0$ from the $q_0\star_0$ corresponding to the least recent symbol that has not been dequeued) *STATE* will be updated and *SYM* will be set to $*$ to record the access. The strategy corresponding to M will then reply with $\star_0$ (because of ()). The one associated with M^- will do the same, unless $\mathbb{Q}$ is about to halt. This is thanks to the $(!\mathit{STATE} \in Q_D \wedge !!\mathit{LAST} = *)$ condition, which checks whether $\mathbb{Q}$ is about to dequeue ($!\mathit{STATE} \in Q_D$) the empty queue ($!!\mathit{LAST} = *$).

Hence, M and M^- both represent the behaviour of $\mathbb{Q}$, except that, if $\mathbb{Q}$ halts, the strategy corresponding to M_1 will not generate the last $q_1\star_1$ segment corresponding to the last dequeuing operation. Consequently, M_0 and M_1 corresponds to $\mathsf{hist}(\mathbb{Q})$ and $\mathsf{hist}^-(\mathbb{Q})$ respectively. □

Observe that $\mathsf{hist}(\mathbb{Q}) = \mathsf{hist}^-(\mathbb{Q})$ exactly when $\mathbb{Q}$ does not halt. Consequently, the problem of deciding $\mathsf{hist}(\mathbb{Q}) = \mathsf{hist}^-(\mathbb{Q})$ is undecidable. Thus, via Proposition 7, we can conclude that program equivalence is undecidable for closed terms of type $\mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}$. The remaining cases are discussed below. $(\mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit} \vdash \mathsf{unit}$: The arena at hand has the following shape. As before, we use $q_0\star_0$ and $q_1\star_1$ to represent enqueuing and dequeuing respectively. They will be preceded by a single segment qq'. Note that this means that no complete plays will arise until q is answered. We shall arrange for this to happen only when the whole terminating run (if any) has been represented.

$$\begin{tikzcd}
q \arrow[d, dash] \arrow[dr, dash] & \\
q' \arrow[d, dash] \arrow[dr, dash] & \star \\
q_0 \arrow[d, dash] & \star' \\
\star_0 \arrow[d, dash] & \\
q_1 \arrow[d, dash] & \\
\star_1 &
\end{tikzcd}$$

Lemma 13 *Let $\mathbb{Q}$ be a queue machine. Then there exists a term*

$$f : (\mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit} \vdash M : \mathsf{unit}$$

such that $\mathsf{comp}(\llbracket M \rrbracket) = \{\epsilon\}$ if and only $\mathbb{Q}$ does not halt.

Proof Reusing $C[-]$, $C_0[-]$, $C_1[-]$ from the previous case, we take M to be $C[f(\lambda x.C_0[\lambda y. C_1[()]]); \mathit{test}]$, where *test* stands for $\mathsf{if}\ (!\mathit{STATE} \in Q_D \wedge !!\mathit{LAST} = *)\ \mathsf{then}\ ()\ \mathsf{else}\ \Omega$. The last condition $(!\mathit{STATE} \in Q_D \wedge !!\mathit{LAST} = *)$ means that whenever O plays $\star'$, P will not respond unless $\mathbb{Q}$ terminates and the terminating run has been wholly represented in the play. The argument showing that M represents $\mathbb{Q}$ is analogous to that for Lemma 12. □

Proposition 14 *It is undecidable whether a given term* $f : (\mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit} \vdash M : \mathsf{unit}$ *is equivalent to* $()$.

$(((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit} \vdash \mathsf{unit}$: The corresponding arena has the shape given on the right. Our representation scheme in this case will start off with qq', enqueuing will be interpreted by q_0q_1 and dequeuing by $q_2\star_2$, where q_2 is justified by q_1 corresponding to the element being dequeued. Note that sequences of this kind are not complete plays, because q, q', q_0, q_1 will remain unanswered. Hence, in the term construction it will not be possible to answer them until a terminating run has been fully simulated. Then O's $\star_1$ will trigger P's $\star_0$, and $\star'$ will trigger $\star$.

$$\begin{tikzcd}
q \arrow[d, dash] \arrow[dr, dash] & \\
q' \arrow[d, dash] \arrow[dr, dash] & \star \\
q_0 \arrow[d, dash] \arrow[dr, dash] & \star' \\
q_1 \arrow[d, dash] \arrow[dr, dash] & \star_0 \\
q_2 \arrow[d, dash] & \star_1 \\
\star_2 &
\end{tikzcd}$$

Lemma 15 *Let* $\mathbb{Q}$ *be a queue machine. Then there exists a term*

$$f : (((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit} \vdash M_{\mathbb{Q}} : \mathsf{unit}$$

such that $\mathsf{comp}([\![M_{\mathbb{Q}}]\!]) = \{\epsilon\}$ *if and only* $\mathbb{Q}$ *does not halt.*

Proof Take $M_{\mathbb{Q}}$ to be $C[f(\lambda g.C_0[g(\lambda h.C_1[()])]; \mathit{test})]; \mathit{test}$. The *test* phrases block P from answering $\star_1$ or $\star'$ prematurely. □

Proposition 16 *The problem of deciding whether a given term*

$$f : (((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit} \vdash M : \mathsf{unit}$$

is equivalent to $()$ *is undecidable.*

$\vdash ((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit}$: The corresponding arena has the shape given on the right. Our representation scheme in this case will start off with $q\star$, enqueuing will be interpreted by q_0q_1 and dequeuing by $q_2\star_2$, where q_2 is justified by q_1 corresponding to the element being dequeued. Note that, apart from $q\star$, sequences of this kind are not complete plays, because q_0, q_1 will remain unanswered. Hence, in the term construction it will not be possible to answer them until a terminating run has been fully simulated. Then O's $\star_1$ will trigger P's $\star_0$.

$$\begin{tikzcd}
q \arrow[d, dash] & \\
\star \arrow[d, dash] & \\
q_0 \arrow[d, dash] \arrow[dr, dash] & \\
q_1 \arrow[d, dash] \arrow[dr, dash] & \star_0 \\
q_2 \arrow[d, dash] & \star_1 \\
\star_2 &
\end{tikzcd}$$

Lemma 17 *Let $\mathbb{Q}$ be a queue machine. Then there exists a term* $\vdash M_{\mathbb{Q}} : ((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit}$ *such that* $\mathsf{comp}([\![M_{\mathbb{Q}}]\!]) = \{\epsilon, q\star\}$ *if and only* $\mathbb{Q}$ *does not halt.*

Proof Take $M_{\mathbb{Q}}$ to be $C[\lambda f^{(\mathsf{unit}\to\mathsf{unit})\to\mathsf{unit}}.C_0[f(\lambda g^{\mathsf{unit}}.C_1[()]); \mathit{test}]]$. The *test* phrases block P from answering $\star_1$ prematurely. □

Proposition 18 *The problem of deciding whether a given term* $\vdash M : ((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit}$ *is equivalent to* $\lambda f^{(\mathsf{unit}\to\mathsf{unit})\to\mathsf{unit}}.\Omega$ *is undecidable.*

5 Decidability

We now focus on a fragment of GRef, called GRef☺, that comprises all types that do *not* fall under the undecidable cases identified earlier.

Definition 19 Suppose $\Gamma = x_1 : \theta_1, \cdots, x_m : \theta_m$. The term-in-context $\Gamma \vdash M : \theta$ belongs to GRef☺ provided $\theta_1, \cdots, \theta_m$ can be generated from Θ_L and θ is generated from Θ_R, where $\Theta_L ::= \beta \mid \Theta_R \to \Theta_L$, $\Theta_R ::= \beta \mid \Theta_1 \to \beta$ and $\Theta_1 ::= \beta \mid \beta \to \Theta_1$.

Put otherwise, we focus on sequents of the form:

$$\Theta_R \to \cdots \to \Theta_R \to \beta \vdash \Theta_R$$

where $\Theta_R = (\beta \to \cdots \to \beta) \to \beta$.

Each type θ can be written in the form $\theta = \theta_n \to \ldots \to \theta_1 \to \beta$, for types $\theta_1, \ldots, \theta_n$ and base type β. For brevity, we shall write $\theta = (\theta_n, \ldots, \theta_1, \beta)$. We call n the *arity* of θ and denote it by $ar(\theta)$.

Definition 20 For every type θ let us define the associated set of labels $\mathcal{L}_\theta$ as follows:

$$\mathcal{L}_{\mathsf{unit}} = \{\star\} \qquad \mathcal{L}_{\mathsf{ref}\,\gamma} = \mathbb{A}_\gamma$$
$$\mathcal{L}_{\mathsf{int}} = \{0, \cdots, max\} \qquad \mathcal{L}_{\theta\to\theta'} = \{\star\}$$

We shall write $\mathcal{L}$ for the set of all labels.

Let us fix notation for referring to moves that are available in arenas corresponding to GRef☺ typing judgments: each move can be viewed as a pair (l, t) subject to consistency constraints induced by the subtypes which contribute them, e.g. the label corresponding to a tag related to int must be a number from $[0, max]$.

More precisely, given $\Gamma \vdash M : \theta, l \in \mathcal{L}$ and some symbol t (determined below), we shall say that the pair (l, t) is *consistent* if the following conditions are satisfied. Below we assume that $(x : \theta') \in \Gamma$ and $\theta' \equiv (\theta_m, \cdots, \theta_1, \beta)$.

– If $t = \mathsf{r}_\downarrow$ then $l \in \mathcal{L}_\theta$.

- If $t = \mathsf{c}_i^x$ then $l \in \mathcal{L}_{\theta_i}$.
- If $t = \mathsf{r}_i^x$ then $l \in \mathcal{L}_{(\theta_{i-1},\ldots,\theta_1,\beta)}$.
- If $t = \mathsf{c}_{j,i}^x$ then $l \in \mathcal{L}_{\theta_{j,i}}$, where $\theta_j \equiv \theta_{j,0} \to \beta$ and $\theta_{j,0} \equiv (\theta_{j,k}, \cdots, \theta_{j,1}, \beta')$.
- If $t = \mathsf{r}_{j,i}^x$ then $l \in \mathcal{L}_{\theta_{j,i}}$, where $\theta_j \equiv (\beta_k, \cdots, \beta_1, \beta) \to \beta_0$, $\theta_{j,0} \equiv \beta_0$ and $\theta_{j,i} \equiv (\beta_{i-1}, \cdots, \beta_1, \beta)$ for $i > 0$.
- If $t = \mathsf{c}_i$ then $l \in \mathcal{L}_{\theta_i}$, where $\theta \equiv \theta_0 \to \beta$ and $\theta_0 \equiv (\theta_n, \cdots, \theta_1, \beta')$.
- If $t = \mathsf{r}_i$ then $l \in \mathcal{L}_{\theta_i}$, where $\theta \equiv (\beta_n, \cdots, \beta_1, \beta) \to \beta_0$, $\theta_0 \equiv \beta_0$ and $\theta_i \equiv (\beta_{i-1}, \cdots, \beta_1, \beta)$ for $i > 0$.

Thus, consistent pairs (t, l) uniquely specify moves of $[\![\Gamma \vdash \theta]\!]$. The tag t specifies the position of the move within the arena, while l determines its actual value. We shall write $\mathbb{T}$ to refer to the set of tags.

In order to show decidability we first translate GRef☺ terms into automata that represent their game semantics. A corollary of Lemma 2 is that any GRef☺ term can be effectively converted to an equivalent term in canonical shape.

$$\begin{aligned}\mathbb{C} ::= {} & () \mid i \mid x^{\mathsf{ref}\,\gamma} \mid \lambda x^{\Theta_1}.\mathbb{C} \mid \mathsf{case}(x^{\mathsf{int}})[\mathbb{C}, \cdots, \mathbb{C}] \mid (\mathsf{while}\,(!x^{\mathsf{ref\,int}})\,\mathsf{do}\,\mathbb{C}); \mathbb{C} \\ & \mid \mathsf{let}\, y^{\gamma} = !x^{\mathsf{ref}\,\gamma}\,\mathsf{in}\,\mathbb{C} \mid (x^{\mathsf{ref\,int}} := i); \mathbb{C} \mid (x^{\mathsf{ref}^2\,\gamma} := y^{\mathsf{ref}\,\gamma}); \mathbb{C} \\ & \mid \mathsf{let}\, x^{\mathsf{ref\,int}} = \mathsf{ref}(0)\,\mathsf{in}\,\mathbb{C} \mid \mathsf{let}\, x^{\mathsf{ref}^2\,\gamma} = \mathsf{ref}(y^{\mathsf{ref}\,\gamma})\,\mathsf{in}\,\mathbb{C} \mid \mathsf{let}\, y^{\Theta_L} = z\,()\,\mathsf{in}\,\mathbb{C} \\ & \mid \mathsf{let}\, y^{\Theta_L} = z\,i\,\mathsf{in}\,\mathbb{C} \mid \mathsf{let}\, y^{\Theta_L} = z\,x^{\mathsf{ref}\,\gamma}\,\mathsf{in}\,\mathbb{C} \mid \mathsf{let}\, y^{\Theta_L} = z\,(\lambda x^{\Theta_1}.\mathbb{C})\,\mathsf{in}\,\mathbb{C}\end{aligned}$$

Consequently, it suffices to show that program equivalence between terms in canonical form is decidable. Accordingly, in what follows, we focus exclusively on translating terms in canonical shape.

We next introduce a class of automata (over an infinite input alphabet) which will be the target of our translation from canonical forms of GRef☺.

5.1 A class of automata

To enable a finite specification of our automata and to describe their semantics we introduce the following definitions. Recall that $\mathbb{A}$ is the set of names, partitioned as:

$$\mathbb{A} = \biguplus_{\gamma} \mathbb{A}_{\gamma}$$

Let $\mathbb{C} = \{\star, 0, \cdots, max\}$ be the set of *constants*. Let us also fix natural numbers n_{r}, n with $n_{\mathsf{r}} \le n$, a finite set $\mathbb{C}_{\mathsf{stack}}$ of *stack symbols* and a finite set $\mathbb{T}$ of *tags*, partitioned into push tags, pop tags and no-op tags:

$$\mathbb{T} = \mathbb{T}_{\mathsf{push}} \uplus \mathbb{T}_{\mathsf{pop}} \uplus \mathbb{T}_{\mathsf{noop}}$$

As general notation, given a partial function f, we write $\mathsf{dom}(f), \mathsf{cod}(f)$ for the sets $\{i \mid f(i) \text{ defined}\}$ and $\{j \mid \exists i.\, f(i) = j\}$ respectively. For each $1 \le i < j \le n$, $[i, j]$ is the set $\{i, i{+}1, \cdots, j\}$.

Definition 21 We introduce the following notions.

- $\mathbb{L} = \mathbb{C} \cup \{\,\mathrm{R}_i \mid 1 \le i \le n\,\}$ is the set of *symbolic labels*. We use ℓ to range over its elements.
- Reg is the set of injective partial functions $\rho : \{1, \cdots, n\} \rightharpoonup \mathbb{A}$. Its elements are called *register assignments* and we use ρ to range over them.
- Sto is the set of partial functions $\Sigma{:}\mathbb{A} \rightharpoonup [0, max] \cup \mathbb{A}$ such that $\mathsf{dom}(\Sigma)$ contains at most n elements and, moreover, if $\Sigma(a) = v$ then: if $a \in \mathbb{A}_{\mathsf{int}}$ then $v \in [0, max]$; if

$a \in \mathbb{A}_{\mathsf{ref}\,\gamma}$ then $v \in \mathbb{A}_\gamma \cap \mathsf{dom}(\Sigma)$ (i.e. Σ is closed and well-typed). Its elements will be called *stores* and ranged over by Σ.

– SSto is the set of partial functions $S{:}[1,n] \rightharpoonup [0, \mathit{max}] \cup \{\mathtt{R}_1, \cdots, \mathtt{R}_n\}$ such that $[1, n_\mathsf{r}] \subseteq \mathsf{dom}(S)$ and, for each $i \in \mathsf{dom}(S)$, $\mathsf{depth}(S,i)$ is well-defined (and finite). The *depth* and the *full value* of an index $i \in \mathsf{dom}(S)$ are given respectively by:

$$\mathsf{depth}(S,i) = \begin{cases} 1 \\ 1 + \mathsf{depth}(S,j) \end{cases} \qquad S^*(i) = \begin{cases} S(i) & \text{if } S(i) \in \{0, ..., \mathit{max}\} \\ (\mathtt{R}_j, S^*(j)) & \text{if } S(i) = \mathtt{R}_j \end{cases}$$

The elements of SSto will be called *symbolic stores* and ranged over by S.[3] The depth restriction ensures that symbolic stores are closed and acyclic.

– $\mathsf{Sta} = (\mathbb{C}_\mathsf{stack} \times \mathsf{Reg})^*$ is the set of *stacks*. We shall range over stacks by σ, and over elements of a stack σ by (s, ρ).

– Mix is the set of partial injections $\pi : [n_\mathsf{r}{+}1, n] \rightharpoonup [n_\mathsf{r}{+}1, n]$.[4] For each π, we write $\overline{\pi}$ for the extension of π on $[1,n]$: $\overline{\pi} = \pi \cup \{(i,i) \mid i \in [1, n_\mathsf{r}]\}$.

– TL is the set of *transition labels*, taken from the set:

$$\begin{gathered}(\mathcal{P}([n_\mathsf{r}{+}1, n]) \times \mathbb{L} \times \mathbb{T}_\mathsf{push} \times \mathbb{C}_\mathsf{stack} \times \mathsf{Mix} \times \mathsf{SSto}) \\ \cup(\mathcal{P}([n_\mathsf{r}{+}1, n]) \times \mathbb{L} \times \mathbb{T}_\mathsf{pop} \times \mathbb{C}_\mathsf{stack} \times \mathsf{Mix} \times \mathsf{SSto}) \\ \cup(\mathcal{P}([n_\mathsf{r}{+}1, n]) \times \mathbb{L} \times \mathbb{T}_\mathsf{noop} \times \mathsf{SSto})\end{gathered}$$

We range over TL by $\nu X.(\ell, t, \phi)^S$, where ϕ can either be:

- a push pair (s, π), in which case we may also write $\nu X.(\ell, t)^S/(s, \pi)$;
- a pop pair (s, π), in which case we may also write $\nu X.(\ell, t)^S, (s, \pi)$;
- or a no-op (), in which case we may simply write $\nu X.(\ell, t)^S$.

We stipulate that $S(j)$ be defined whenever $\nu X.(\ell, t, \phi)^S \in \mathsf{TL}$ and $j \in X$ or $\ell = \mathtt{R}_j$. Moreover, we partition $\mathsf{TL} = \mathsf{TL}_\mathsf{push} \uplus \mathsf{TL}_\mathsf{pop} \uplus \mathsf{TL}_\mathsf{noop}$ depending on the partitioning of tags (e.g. $\mathsf{TL}_\mathsf{push} = \{\nu X.(\ell, t, \phi)^S \mid t \in \mathbb{T}_\mathsf{push}\}$).

We write $\overline{\pi}(S)$ for $\{(\overline{\pi}(i), \mathtt{R}_{\overline{\pi}(j)}) \mid (i, \mathtt{R}_j) \in S\} \cup \{(\overline{\pi}(i), j) \mid (i,j) \in S\}$. Given a pair $(\rho, S) \in \mathsf{Reg} \times \mathsf{SSto}$ we say that ρ, S are *compatible* if $\mathsf{dom}(S) = \mathsf{dom}(\rho)$ and, for all $i \in \mathsf{dom}(S)$,

$$\rho(i) \in \mathbb{A}_\mathsf{int} \implies S(i) \in \{0, \ldots, \mathit{max}\}, \quad \rho(i) \in \mathbb{A}_{\mathsf{ref}\,\gamma} \implies S(i) = \mathtt{R}_j \wedge \rho(j) \in \mathbb{A}_\gamma.$$

In such a case, we can derive the store:

$$\mathsf{Sto}(\rho, S) = \{\, (\rho(i), S(i)) \mid S(i) \in \{0, ..., \mathit{max}\} \,\} \cup \{\, (\rho(i), \rho(j)) \mid S(i) = \mathtt{R}_j \,\}$$

Moreover, we shall be using the following notation for assignment updates,

$$\rho[(i_1, ..., i_m) \mapsto (z_1, ..., z_m)] = \{(i, \rho(i)) \mid i \in X\} \cup \{(i_j, z_j) \mid 1 \le j \le m, z_j \ne \sharp\}$$

with $X = \mathsf{dom}(\rho) \backslash \{i_1, \cdots, i_m\}$ and each $z_j \in \mathbb{A} \cup \{\sharp\}$. Note in particular that the symbol $\sharp$ is used for register deletions. Similar notations will be used for store and partial-injection updates. Furthermore, for a store Σ and a set of names B, we define $\Sigma \restriction B = \{(a, v) \in \Sigma \mid a \in B\}$ and $\Sigma \backslash B = \{(a, v) \in \Sigma \mid a \notin B\}$. Finally, we let $\mathsf{clo}(\Sigma, B)$ be the least set of names C such that $B \subseteq C$ and, for all $a \in C$, if $\Sigma(a) \in \mathbb{A}$ then $\Sigma(a) \in C$. For each

[3] Symbolic stores represent stores by use of indices instead of actual names. For example, $S(i) = \mathtt{R}_j$ means that in S the i-th name stores the j-th name.

[4] Explicitly, for all $i \ne j$, if $\pi(i), \pi(j)$ are both defined then $\pi(i) \ne \pi(j)$.

symbolic store S and set of indices X, we define $S \upharpoonright X$, $S \backslash X$ and $\mathsf{clo}(S, X)$ in an analogous manner.

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, and utilises a pushdown stack where it pushes stack symbols along with full register assignments.

Definition 22 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; $F \subseteq Q_O$ is the set of final states;
- $\rho_0 \in \mathsf{Reg}$ is the initial register assignment such that $[1, n_\mathsf{r}] \subseteq \mathsf{dom}(\rho_0)$;
- $\delta \subseteq (Q_P \times (\mathsf{TL}_\mathsf{push} \cup \mathsf{TL}_\mathsf{noop}) \times Q_O) \cup (Q_O \times (\mathsf{TL}_\mathsf{pop} \cup \mathsf{TL}_\mathsf{noop}) \times Q_P) \cup (Q_O \times \mathsf{Mix} \times Q_O) \cup (Q_P \times \mathsf{Mix} \times Q_P)$ is the transition relation.

Additionally, if θ is a base type then there is a unique final state q_F without outgoing transitions and reachable only via no-op transitions.

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

- If $(q_1, \nu X.(\ell, t, \phi)^S, q_2) \in \delta$, $\mathcal{A}$ will accept an input $(l, t)^\Sigma$ and move to state q_2 if the following steps are successful.
  - If $t \in \mathbb{T}_\mathsf{pop}$ and $\phi = (s, \pi)$, $\mathcal{A}$ will check whether the stack has the form $\sigma = (s, \rho') :: \sigma'$ with $\rho(i) = \rho'(i') \in \mathbb{A}$ iff $\pi(i) = i'$, for all i, i', and $\mathsf{dom}(\rho) \cap \mathsf{dom}(\rho') = \emptyset$. In such a case $\mathcal{A}$ will pop from the stack, that is, it will set $\sigma = \sigma'$ and $\rho = \rho[(i_1, ..., i_m) \mapsto (\rho'(i_1), ..., \rho'(n_m))]$, where $i_1, ..., i_m$ is an enlisting of $\mathsf{dom}(\rho') \backslash \mathsf{cod}(\pi)$.
  - $\mathcal{A}$ will update ρ with fresh names, that is, it will check whether $\mathsf{dom}(\rho) \cap X = \emptyset$ and, if so, it will set $\rho = \rho[(i_1, \cdots, i_m) \mapsto (a_1, \cdots, a_m)]$, where $i_1, \cdots, i_m$ is an enumeration of X and $a_1, \cdots, a_m$ are distinct names such that:
    - if $q_1 \in Q_O$ then $a_1, \cdots, a_m \notin \rho([1, n])$ (*locally fresh*),
    - if $q_1 \in Q_P$ then $a_1, \cdots, a_m$ have not appeared in the current run of $\mathcal{A}$ (*globally fresh*).
  - $\mathcal{A}$ will check if (l, Σ) corresponds to (ℓ, S) via ρ, that is, whether $\Sigma = \mathsf{Sto}(\rho, S)$ and either $\ell = l \in \mathbb{C}$, or $\ell = \mathrm{R}_i$ and $\rho(i) = l$.
  - If $t \in \mathbb{T}_\mathsf{push}$ and $\phi = (s, \pi)$, $\mathcal{A}$ will perform a push of the registers in $\mathsf{dom}(\pi)$, after rearranging them according to π, that is, it will set $\sigma = (s, \rho \circ \pi) :: \sigma$.
- If $(q_1, \pi, q_2) \in \delta$, for $\pi \in \mathsf{Mix}$, $\mathcal{A}$ will reorganize the contents of registers in $[n_\mathsf{r}{+}1, n]$ according to π, that is, set $\rho = \rho \circ \overline{\pi}$, and move to q_2 without reading any input symbol (ϵ-transition).

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

Definition 23 Let $\mathcal{A} = \langle Q, q_0, \rho_0, \delta, F\rangle$ be an (n_r, n)-automaton. The configuration graph $(\hat{Q}, \to_\delta)$ 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 $(q, \rho, \sigma, H) \in \hat{Q}$ and $(q, \nu X.(\ell, t, \phi)^S, q') \in \delta$ we have $(q, \rho, \sigma, H) \xrightarrow{(l,t)^\Sigma}_\delta (q', \rho', \sigma', H')$ where $\Sigma = \mathsf{Sto}(\rho', S)$ and:

- if $t \in \mathbb{T}_{\mathsf{pop}}$ and $\phi = (s, \pi)$ then $\sigma = (s, \rho_0) :: \sigma'$ and
  - for all i, i', $\rho(i) = \rho_0(i')$ iff $(i, i') \in \pi$, and $\mathsf{dom}(\rho_0) \cap \mathsf{dom}(\rho) = \emptyset$,
  - $\rho_1 = \rho[(i_1, ..., i_m) \mapsto (\rho_0(i_1), ..., \rho_0(i_m))]$ with $\{i_1, ..., i_m\} = \mathsf{dom}(\rho_0) \backslash \mathsf{cod}(\pi)$;

  otherwise $\rho_1 = \rho$;
- if $X = \{i_1, \cdots, i_m\}$ then $\mathsf{dom}(\rho_1) \cap X = \emptyset$, $H_1 = H \cup \{a_1, \cdots, a_m\}$ and $\rho' = \rho_1[(i_1, \cdots, i_m) \mapsto (a_1, \cdots, a_m)]$ where $a_1, \cdots, a_m$ are distinct names and:
  - if $q \in Q_O$ then $a_1, \cdots, a_m \notin \rho_1([1, n])$,
  - if $q \in Q_P$ then $a_1, \cdots, a_m \notin \rho_1([1, n]) \cup H$;
- if $\ell \in \mathbb{C}$ then $l = \ell$ and $H' = H_1$;
- if $\ell = \mathtt{R}_i$ then $l = \rho'(i)$ and $H' = H_1 \cup \{l\}$;
- if $t \in \mathbb{T}_{\mathsf{push}}$ and $\phi = (s, \pi)$ then $\sigma' = (s, \rho' \circ \pi) :: \sigma$.

Moreover, for all $(q, \rho, \sigma, H) \in \hat{Q}$ and $(q, \pi, q') \in \delta$ we have $(q, \rho, \sigma, H) \xrightarrow{\epsilon}_\delta (q', \rho', \sigma, H)$, where $\rho' = \rho \circ \overline{\pi}$.
The set of strings *accepted* by $\mathcal{A}$ is defined as below, where ϵ is the empty stack.

$$L(\mathcal{A}) = \{\mathbf{l} \in ((\mathbb{C} \cup \mathbb{A}) \times \mathbb{T} \times \mathsf{Sto})^* \ \mid \ (q_0, \rho_0, \epsilon, \emptyset) \xrightarrow{\mathbf{l}}\!\!\!\!\rightarrow_\delta (q, \rho, \sigma, H), \ \ q \in F\}.$$

We say that $\mathcal{A}$ is ***deterministic*** if, for any reachable configuration $\hat{q}$, any $x_1, x_2 \in \{\epsilon\} \cup (\mathbb{C} \cup \mathbb{A}) \times \mathbb{T} \times \mathsf{Sto}$, and any $\hat{q} \xrightarrow{x_1}_\delta \hat{q}_1, \hat{q} \xrightarrow{x_2}_\delta \hat{q}_2$, if $x_1 = x_2$ then $\hat{q}_1 = \hat{q}_2$. The automata specifically used for our constructions follow some stronger disciplines.

Definition 24 We say that $\mathcal{A}$ is *strongly deterministic* if:

- for each $q \in Q_P$ there is at most one transition out of q (i.e. $|\delta \restriction \{q\}| \leq 1$), and if $(q, \nu X.(\ell, t, \phi)^S, q') \in \delta$ then $|\delta \restriction \{q'\}| \leq 1$ and in particular q' may only have an outgoing transition of the form (q', π, q'') such that $\forall \pi', q'''.\ (q'', \pi', q''') \notin \delta$;
- for each $q \in Q_O$ and $(q, \nu X_i.(\ell_i, t, \phi)^{S_i}, q_i) \in \delta$, $i = 1, 2$, if $\nu X_1.(\ell_1, S_1)$ and $\nu X_2.(\ell_2, S_2)$ are equal up to permutation of indices[5] in X_1, X_2 then $\nu X_1.(\ell_1, t, \phi)^{S_1} = \nu X_2.(\ell_2, t, \phi)^{S_2}$ and $q_1 = q_2$;
- for each $(q, \nu X.(\ell, t, \phi)^S, q') \in \delta$, X is contained in $\mathsf{clo}(S, X_{\mathsf{Av}})$ where $X_{\mathsf{Av}} = (\mathsf{dom}(S) \backslash X) \cup \{j \mid \ell = \mathtt{R}_j\}$.

For such an $\mathcal{A}$, we may write $q_P \xrightarrow{\nu X.(\ell,t,\phi)^S;\pi} q_O$ for $q_P \xrightarrow{\nu X.(\ell,t,\phi)^S} q'_O \xrightarrow{\pi} q_O$, where $q_P \in Q_P, q_O \in Q_O$. The last condition above corresponds to *frugality* (cf. Definition 5): fresh names must be reachable from names that were already available or appear in the current transition label.

Lemma 25 *If $\mathcal{A}$ is strongly deterministic then it is deterministic.*

Proof The claim is obvious for configurations with P-states, as well as for (reachable) configurations with O-states and outgoing transitions of the form $q \xrightarrow{\pi} q'$, because of the first condition in the previous definition. For configurations with O-states and transitions of the form $q \xrightarrow{\nu X.(\ell,t,\phi)^S} q'$, the second condition above ensures that each label has at most one accepting edge, as long as in each configuration each top stack element can be popped uniquely (i.e. with at most one ϕ). The latter follows from the definition of configuration graphs. □

[5] Here, in fact, we refer to the alpha-equivalence relation induced by the ν binder: for example, $\nu\{1, 2\}.(\mathtt{R}_1, \{(1, \mathtt{R}_2), (2, 0), (3, 1)\}) \sim_\alpha \nu\{5, 6\}.(\mathtt{R}_6, \{(3, 1), (5, 0), (6, \mathtt{R}_5)\})$.

Lemma 26 *Given $\mathcal{A}$ strongly deterministic and $w_1, w_2 \in L(\mathcal{A})$, if* $\text{ext}(w_1) \cap \text{ext}(w_2) \neq \emptyset$ *then $w_1 = w_2$.*

Proof WLOG we assume that w_1, w_2 have the same underlying sequence of moves. Let $w_i' m^{\Sigma_i}$ be the prefix of w_i of length n ($i = 1, 2$). We show by induction on n that $\Sigma_1 = \Sigma_2$. By IH (if $n > 1$) or by definition (if $n = 1$) we have that $w_1' = w_2'$. Moreover, by the previous lemma, w_1', w_2' lead to some common configuration $\hat{q}$ of $\mathcal{A}$. If n is even then, by the first clause of Definition 24, we have that $\Sigma_1 = \Sigma_2$. If n is odd then by hypothesis we have that Σ_1, Σ_2 have a common extension and, thus, using the last clause of Definition 24, we obtain $\Sigma_1 = \Sigma_2$. □

Definition 27 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 $E = \biguplus_{(X,\ell,t,S)} E_{\nu X.(\ell,t)^S}$ where $E_{\nu X.(\ell,t)^S} = \{\, q \in Q \mid (q, \nu X.(\ell, t, ())^S, q_F) \in \delta \,\}$ and $\mathcal{A}$ is uniquely determined by the structure $\mathcal{A}^- = \langle Q, q_0, \rho_0, \delta, E \rangle$.

5.2 Automata for GRef☺

Recall we are only going to translate terms in canonical form.

Let $\Gamma = \{x_1 : \theta_1, \cdots, x_m : \theta_m\}$ and $\Gamma \vdash \mathbb{C} : \theta$ be a GRef☺-term in canonical form. Let us write $P^1_{\Gamma\vdash\theta}$ for the set of plays-with-store of length 1 over $\Gamma \vdash \theta$. Recall that each of them will have the form i^{Σ_0}, where $\mathrm{i} \in I_\Gamma$, i.e. $\mathrm{i} = (l_1, \cdots, l_m)$ with $l_i \in \mathcal{L}_{\theta_i}$. Moreover, the names in i^{Σ_0} coincide with those of $\mathsf{dom}(\Sigma_0) = \nu(\Sigma_0)$ and, by frugality, $\nu(\Sigma_0) = \mathsf{clo}(\Sigma_0, \nu(\mathrm{i}))$. These can be ordered by use of register assignments, of fixed size appropriate to contain all names in Σ_0 and names created while translating GRef☺-terms, leading to the following construction.

$$I^+_{\Gamma\vdash\theta} = \{(\mathrm{i}^{\Sigma_0}, \rho_0) \mid \mathrm{i}^{\Sigma_0} \in P^1_{\Gamma\vdash\theta},\ \nu(\rho_0) = \nu(\Sigma_0),\ \exists k.\, \rho_0([1,k]) = \nu(\mathrm{i})\}$$

For brevity, we shall write each element $(\mathrm{i}^{\Sigma_0}, \rho_0) \in I^+_{\Gamma\vdash\theta}$ as $\mathrm{i}^{\Sigma_0}_{\rho_0}$. We now instantiate the automata defined in the previous section by using the finite set of tags $\mathbb{T} = \mathbb{T}_{\mathsf{push}} \cup \mathbb{T}_{\mathsf{pop}} \cup \mathbb{T}_{\mathsf{noop}}$, where

$$\mathbb{T}_{\mathsf{push}} = \{\mathsf{c}_i \in \mathbb{T} \mid i > 0\} \cup \{\mathsf{c}^x_i \in \mathbb{T}\} \cup \{\mathsf{c}^x_{j,i} \in \mathbb{T} \mid i > 0\}$$
$$\mathbb{T}_{\mathsf{pop}} = \{\mathsf{r}_i \in \mathbb{T} \mid i > 0\} \cup \{\mathsf{r}^x_i \in \mathbb{T}\} \cup \{\mathsf{r}^x_{j,i} \in \mathbb{T} \mid i > 0\}$$
$$\mathbb{T}_{\mathsf{noop}} = \{\mathsf{r}_\downarrow, \mathsf{c}_0, \mathsf{r}_0, \mathsf{c}^x_{j,0}, \mathsf{r}^x_{j,0}\}.$$

Moreover, we will impose the following condition on our automata. All push/pops will not involve any registers (i.e. they will have $\phi = (s, \emptyset)$), except if the tag t in question satisfies:

$$t \in \{\mathsf{c}_i, \mathsf{r}_i \in \mathbb{T} \mid i > 0\} \cup \{\mathsf{c}^x_{j,i}, \mathsf{r}^x_{j,i} \in \mathbb{T} \mid i > 0\}$$

Remark 28 A canonical form of GRef☺ will be translated into a family of automata indexed by $I^+_{\Gamma\vdash\theta}$. For each $\mathrm{i}^{\Sigma_0}_{\rho_0} \in I^+_{\Gamma\vdash\theta}$, the corresponding automaton will accept exactly the words w such that $\mathrm{i}^{\Sigma_0} w$ is a representation of a complete play induced by the canonical form. The family will be infinite, but *finite* when considered up to name permutations.

For any GRef☺-term $\Gamma \vdash \mathbb{C} : \theta$ in canonical form we define an $I^+_{\Gamma\vdash\theta}$-indexed family of automata $(\!|\, \mathbb{C} \,|\!) = \{\, (\!|\, \mathbb{C} \,|\!)_{\mathrm{i}^{\Sigma_0}_{\rho_0}} \mid \mathrm{i}^{\Sigma_0}_{\rho_0} \in I^+_{\Gamma\vdash\theta} \,\}$ (each of type θ) by induction on the shape of $\mathbb{C}$. In all cases $(\!|\, \mathbb{C} \,|\!)_{\mathrm{i}^{\Sigma_0}_{\rho_0}}$ will have $n_0 = |\nu(\mathrm{i})|$ read-only registers and the initial assignment will be ρ_0. The precise number of registers can be calculated easily by reference to the constituent automata. Let us write S_0 for the symbolic store defined by $S_0(i) = \Sigma_0(\rho_0(i))$ if $\Sigma_0(\rho_0(i)) \in \{0, \ldots, max\}$, and $S_0(i) = \mathtt{R}_j$ if $\Sigma_0(\rho_0(i)) = \rho_0(j)$. The base and inductive cases are as follows.

- $\llparenthesis\, () \,\rrparenthesis_{\mathrm{i}^{\Sigma_0}_{\rho_0}} = \quad q_0 \xrightarrow{(\star,\mathsf{r}_\downarrow)^{S_0}} q_F\,.$
- $\llparenthesis\, i \,\rrparenthesis_{\mathrm{i}^{\Sigma_0}_{\rho_0}} = \quad q_0 \xrightarrow{(i,\mathsf{r}_\downarrow)^{S_0}} q_F\,.$
- $\llparenthesis\, x^{\mathsf{ref}\,\gamma} \,\rrparenthesis_{\mathrm{i}^{\Sigma_0}_{\rho_0}} = \quad q_0 \xrightarrow{(\mathtt{R}_j,\mathsf{r}_\downarrow)^{S_0}} q_F$, where $x \equiv x_k$ and $l_k = \rho_0(j)$.
- $\llparenthesis\, \mathsf{case}(x^{\mathsf{int}})[\mathbb{C}_0, \cdots, \mathbb{C}_{max}] \,\rrparenthesis_{\mathrm{i}^{\Sigma_0}_{\rho_0}} = \llparenthesis\, \mathbb{C}_j \,\rrparenthesis_{\mathrm{i}^{\Sigma_0}_{\rho_0}}$, where $x \equiv x_k$ and $l_k = j$.
- $\llparenthesis\, (x := y); \mathbb{C} \,\rrparenthesis_{\mathrm{i}^{\Sigma_0}_{\rho_0}} = \quad q_0 \xrightarrow{\pi} \llparenthesis\, \mathbb{C} \,\rrparenthesis_{\mathrm{i}^{\Sigma''_0}_{\rho'_0}}$ where $x \equiv x_k$, $y \equiv x_j$ and $\Sigma'_0 = \Sigma_0[l_k \mapsto l_j]$,

  and the initial transition deletes all names in ρ_0 which break frugality of $\mathrm{i}^{\Sigma'_0}$, that is, $\pi(i) = i$ just if $\rho_0(i) \in \mathsf{clo}(\Sigma'_0, \nu(\mathrm{i}))$. Moreover, $\rho'_0 = \rho_0 \circ \overline{\pi}$ and $\Sigma''_0 = \Sigma'_0 \restriction \mathsf{cod}(\rho'_0)$.
- $\llparenthesis\, \mathsf{let}\, y = !x \,\mathsf{in}\, \mathbb{C} \,\rrparenthesis_{\mathrm{i}^{\Sigma_0}_{\rho_0}} = \quad q_0 \xrightarrow{\pi} \llparenthesis\, \mathbb{C} \,\rrparenthesis_{(\mathrm{i}\ \Sigma_0(l_k))^{\Sigma_0}_{\rho'_0}}$ where $x \equiv x_k$ and π transfers $\Sigma_0(l_k)$,

  if it is a name, to the register in position $n_0 + 1$, and leaves all other names in ρ_0 untouched. Moreover, $\rho'_0 = \rho_0 \circ \overline{\pi}$.
- $\llparenthesis\, \mathsf{let}\, y^{\mathsf{unit}} = z() \,\mathsf{in}\, \mathbb{C} \,\rrparenthesis_{\mathrm{i}^{\Sigma_0}_{\rho_0}} = \quad q_0 \xrightarrow{(\star,\mathsf{c}^z_r)^{S_0}/(q_0,\emptyset)\,;\,\emptyset} q_1 \xrightarrow{\nu X.(\star,\mathsf{r}^z_r)^{S},(q_0,\emptyset)} \llparenthesis\, \mathbb{C} \,\rrparenthesis_{(\mathrm{i}\,\star)^{\Sigma}_{\rho'_0}}$

  where z has arity r and Σ ranges over stores with $\mathsf{dom}(\Sigma) = \mathsf{clo}(\Sigma, \nu(\mathrm{i}))$. These are infinitely many, but finitely many up to permutations of fresh names. In the transitions we pick X, S such that there is one transition for each of the (finitely many) equivalence classes. Moreover, ρ'_0 is specified by stipulating $\mathsf{dom}(\rho'_0) = [1, n_0] \cup X$ and $\Sigma = \mathsf{Sto}(\rho'_0, S)$.
- $\llparenthesis\, \mathsf{let}\, y^{\mathsf{int}} = z() \,\mathsf{in}\, \mathbb{C} \,\rrparenthesis_{\mathrm{i}^{\Sigma_0}_{\rho_0}} = \quad q_0 \xrightarrow{(\star,\mathsf{c}^z_r)^{S_0}/(q_0,\emptyset)\,;\,\emptyset} q_1 \xrightarrow{\nu X.(j,\mathsf{r}^z_r)^{S},(q_0,\emptyset)} \llparenthesis\, \mathbb{C} \,\rrparenthesis_{(\mathrm{i}\,j)^{\Sigma}_{\rho'_0}}$

  with r, Σ, S, X, ρ'_0 as above and j ranging over $[0, max]$.
- $\llparenthesis\, \mathsf{let}\, y^{\mathsf{ref}\,\gamma} = z() \,\mathsf{in}\, \mathbb{C} \,\rrparenthesis_{\mathrm{i}^{\Sigma_0}_{\rho_0}} =$

$$\begin{tikzcd} q_0 \arrow[r, "(\star{,}\mathsf{c}^z_r)^{S_0}/(q_0{,}\emptyset)\,;\,\emptyset"] & q_1 \arrow[r, "\nu X.(\mathtt{R}_j{,}\mathsf{r}^z_r)^{S}{,}(q_0{,}\emptyset)"] \arrow[dr, "\nu X'.(\mathtt{R}_{n_0+1}{,}\mathsf{r}^z_r)^{S'}{,}(q_0{,}\emptyset)"'] & \llparenthesis\, \mathbb{C} \,\rrparenthesis_{(\mathrm{i}\,\rho_0(j))^{\Sigma}_{\rho'_0}} \\ & & \llparenthesis\, \mathbb{C} \,\rrparenthesis_{(\mathrm{i}\,b)^{\Sigma'}_{\rho''_0}} \end{tikzcd}$$

  where r, Σ, S, X, ρ'_0 are as above, j ranges over elements of $\mathsf{dom}(S)$ such that $\mathsf{ref}\,\gamma = \mathsf{ref}^{\mathsf{depth}(S,j)}\,\mathsf{int}$, and $b \in \mathbb{A}_\gamma$ is a fresh name. We let Σ' range over all stores with $\mathsf{dom}(\Sigma') = \mathsf{clo}(\Sigma', \nu(\mathrm{i}) \cup \{b\})$. We pick X', S' such that there is one symbolic transition for each of the intended transitions $(b, \mathsf{r}^z_r)^{\Sigma'}$ and specify ρ''_0 accordingly, making sure that $\rho''_0(n_0 + 1) = b$.

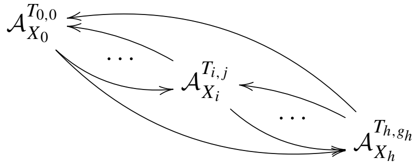
- $(\!|\,\mathsf{let}\, y^{\Theta_R \to \Theta_L} = z()\ \mathsf{in}\ \mathbb{C}\,|\!)_{\mathrm{i}_{\rho_0}}^{\Sigma_0} = \ q_0 \xrightarrow{(\star,\mathsf{c}_r^z)^{S_0}/(q_0,\emptyset)\,;\,\emptyset} q_1 \xrightarrow{\nu X.(\star,\mathsf{r}_r^z)^{S},(q_0,\emptyset)} (\!|\,\mathbb{C}\,|\!)_{(\mathrm{i}\,\star)_{\rho_0'}^{\Sigma}}[z/y]$

  where r, Σ, S, X, ρ_0' are as above and $(\!|\,\mathbb{C}\,|\!)_{(\mathrm{i}\,\star)_{\rho_0'}^{\Sigma}}[z/y]$ is $(\!|\,\mathbb{C}\,|\!)_{(\mathrm{i}\,\star)_{\rho_0'}^{\Sigma}}$ where we have replaced every tag superscript y with z.
- $(\!|\,\mathsf{let}\, y = z\, i\ \mathsf{in}\ \mathbb{C}\,|\!)_{\mathrm{i}_{\rho_0}}^{\Sigma_0}$ and $(\!|\,\mathsf{let}\, y = z\, x^{\mathsf{ref}\,\gamma}\ \mathsf{in}\ \mathbb{C}\,|\!)_{\mathrm{i}_{\rho_0}}^{\Sigma_0}$ are defined similarly to the above.
- Case of $\mathsf{let}\, x^{\mathsf{ref}^2\,\gamma} = \mathsf{ref}(y^{\mathsf{ref}\,\gamma})\ \mathsf{in}\ \mathbb{C}$. Here the inductive hypothesis gives us an automaton for $\Gamma, x : \mathsf{ref}^2\,\gamma \vdash \mathbb{C} : \theta$. In order to transform the latter into an automaton for our given term, we need to hide the name corresponding to x from the automaton, until the point where the name is eventually revealed in some move (it is also possible that the name remains private indefinitely). This hiding of x effectively has wider repercussions as we need also to hide any name that x exclusively points to, and so on, along with their stored values. It is therefore useful to define a more general hiding construction.

Let $\mathcal{A}$ be an (n_{r}, n)-automaton, let $X_0, \cdots, X_h$ be an enumeration of all subsets of $[n_{\mathsf{r}} + 1, n]$ and let $T_{i,0}, \cdots, T_{i,g_i}$ be an enumeration of all partial symbolic stores on X_i (i.e. of all $T = S \upharpoonright X_i$ for some symbolic store S). For each $0 \le i \le h$ and $0 \le j \le g_i$ we define an (n_{r}, n)-automaton $\mathcal{A}_{X_i}^{T_{i,j}}$ to be a copy of $\mathcal{A}$ in which we have hidden the names in registers X_i, while the stored restricted to the names in those registers is $T_{i,j}$. Concretely, $\mathcal{A}_{X_i}^{T_{i,j}}$ is a copy of $\mathcal{A}$ in which we have removed all transitions apart from those of the form $q_1 \xrightarrow{\nu X.(\ell,t,\phi)^S} q_2$ with q_1 an O-state and such that:

$$S \upharpoonright X_i = T_{i,j}\ ,\ \forall m \in X_i. \forall m' \notin X_i.\ \ell \neq \mathrm{R}_m,\ S(m') \neq \mathrm{R}_m$$

and in those remaining transitions we have removed X_i from the domains of all symbolic stores. We define $\nu\mathcal{A}$ as the (n_{r}, n)-automaton fragment (no initial state) obtained by interconnecting these automata as below.


$$\begin{tikzcd}
\mathcal{A}_{X_0}^{T_{0,0}} \arrow[dr, bend right] \arrow[ddrr, bend right] & & \\
 & \mathcal{A}_{X_i}^{T_{i,j}} \arrow[ul, bend right] \arrow[dr, bend right] & \\
 & & \mathcal{A}_{X_h}^{T_{h,g_h}} \arrow[uull, bend right] \arrow[ul, bend right]
\end{tikzcd}$$

The transitions are as follows. Let (X, T) be an element of the above enumeration.

- For each $q_1 \xrightarrow{\nu Y.(\ell,t,\phi)^S} q_2$ in $\mathcal{A}$ with q_1 a P-state, add a transition from q_1 in $\mathcal{A}_X^T$ to q_2 in $\mathcal{A}_{X'}^{T'}$ with label $\nu Y'.(\ell, t, \phi)^{S'}$, where $X' = (X \cup Y)\backslash Y'$, $S' = S\backslash X'$, $T' = S \upharpoonright X'$ and $Y' = \mathsf{clo}(S, (\mathsf{dom}(S)\backslash X) \cup \{j \mid \ell = \mathrm{R}_j\})$.
- For each $q_1 \xrightarrow{\pi} q_2$ in $\mathcal{A}$ add a transition from q_1 in $\mathcal{A}_X^T$ to q_2 in $\mathcal{A}_{X'}^{T'}$ with label π, where $X' = \pi^{-1}(X)$ and $T' = \pi^{-1}(T)$.

Let now $a \in \mathbb{A}_{\mathsf{ref}\,\gamma} \backslash \nu(\rho_0)$, suppose $y \equiv x_k$, $\rho_0(k') = l_k$, let $\Sigma_0' = \Sigma_0[a \mapsto l_k]$ and let n_0' be the first empty register in ρ_0. We take $\pi_0 \in \mathsf{Mix}$ to be such that it swaps $n_0 + 1$ and n_0', and fixes all other indices, and set $\rho_0' = \rho_0[n_0 + 1 \mapsto a, n_0' \mapsto \rho_0(n_0 + 1)]$. We define:

$(\!|\, \mathsf{let}\, x = \mathsf{ref}(y)\, \mathsf{in}\, \mathbb{C} \,|\!) = \; q_0 \xrightarrow{\pi_0} \nu(\!|\, \mathbb{C} \,|\!)_{(\mathrm{i}a)^{\Sigma'_0}_{\rho'_0}}$ where the transition π_0 points to the initial state of the $(\{n_0+1\}, \{(n_0+1, \mathrm{R}_{k'})\})$ component of $\nu(\!|\, \mathbb{C} \,|\!)_{(\mathrm{i}a)^{\Sigma'_0}_{\rho'_0}}$.[6]

- The case of $\mathsf{let}\, x^{\mathsf{ref\, int}} = \mathsf{ref}(0)\, \mathsf{in}\, \mathbb{C}$ is dealt with similarly to the above.
- For $(\mathsf{while}\, (!x)\, \mathsf{do}\, \mathbb{C}); \mathbb{C}'$, given the automata $(\!|\, \mathbb{C} \,|\!)$ and $(\!|\, \mathbb{C}' \,|\!)$, with appropriate initialisations, we construct a new automaton as follows. Suppose $x \equiv x_k$ and $\rho_0(k') = l_k$. If the initial value stored for x is 0 (i.e. if $\Sigma_0(l_k) = 0$) then we simply return $(\!|\, \mathbb{C}' \,|\!)_{\mathrm{i}^{\Sigma_0}_{\rho_0}}$. Otherwise, we need to combine the automata for $\mathbb{C}$ and $\mathbb{C}'$ in such a away so that $(\!|\, \mathbb{C} \,|\!)$ is involved repeatedly (with appropriate initialisation), until it reaches a final state with a final transition with a symbolic store assigning 0 to k'. At this point, the automaton would switch and start simulating $(\!|\, \mathbb{C}' \,|\!)$. An important point in this construction is that the final transitions of $(\!|\, \mathbb{C} \,|\!)$ are hidden in the new automaton, as the return values of the while guard are not revealed in the semantics. This hiding implies a potential hiding of names as well: any names created in final transitions of $(\!|\, \mathbb{C} \,|\!)$ need to be hidden as well. This latter kind of hiding is delegated to the ν-construction that we described two cases above.

Formally, let $\Sigma_0, \cdots, \Sigma_h$ be an enumeration (modulo permutation of fresh names) of all stores Σ with $\mathsf{dom}(\Sigma) = \mathsf{clo}(\Sigma, \nu(\mathrm{i}))$. Recall $x \equiv x_k$, $\rho_0(k') = l_k$, and recall the presentation of an automaton given in Definition 27. We define $(\!|\, (\mathsf{while}\, (!x)\, \mathsf{do}\, \mathbb{C}); \mathbb{C}' \,|\!)_{\mathrm{i}^{\Sigma_0}_{\rho_0}}$ to be $(\!|\, \mathbb{C}' \,|\!)_{\mathrm{i}^{\Sigma_0}_{\rho_0}}$ if $\Sigma_0(l_k) = 0$. Otherwise, we define it to be a combination of $\nu(\!|\, \mathbb{C} \,|\!)^{-}_{\mathrm{i}^{\Sigma_0}_{\rho_0}}$, $\cdots$, $\nu(\!|\, \mathbb{C} \,|\!)^{-}_{\mathrm{i}^{\Sigma_h}_{\rho_h}}$ and $\nu(\!|\, \mathbb{C}' \,|\!)_{\mathrm{i}^{\Sigma_0}_{\rho_0}}, \cdots, \nu(\!|\, \mathbb{C}' \,|\!)_{\mathrm{i}^{\Sigma_h}_{\rho_h}}$, with each ρ_i specified by Σ_i as above, connected together as below.

$$\begin{tikzcd}
\nu(\!|\, \mathbb{C} \,|\!)^{-}_{\mathrm{i}^{\Sigma_0}_{\rho_0}} \arrow[rr] \arrow[dr] \arrow[drrr] \arrow[loop below] \arrow[dr, phantom, bend left, "\ddots"] & & \nu(\!|\, \mathbb{C}' \,|\!)_{\mathrm{i}^{\Sigma_0}_{\rho_0}} & \ddots & \\
& \nu(\!|\, \mathbb{C} \,|\!)^{-}_{\mathrm{i}^{\Sigma_h}_{\rho_h}} \arrow[ul, bend right] \arrow[ur] \arrow[rr] \arrow[loop below] & & \nu(\!|\, \mathbb{C}' \,|\!)_{\mathrm{i}^{\Sigma_h}_{\rho_h}} &
\end{tikzcd}$$

The initial state is the one of the $(\emptyset, \emptyset)$ component of $\nu(\!|\, \mathbb{C} \,|\!)^{-}_{\mathrm{i}^{\Sigma_0}_{\rho_0}}$. Let S_i be specified by each Σ_i, ρ_i. For each quasi-final state $q \in E_{\nu X.(\star, \mathsf{r}_\downarrow)S}$ of each $\nu(\!|\, \mathbb{C} \,|\!)^{-}_{\mathrm{i}^{\Sigma_j}_{\rho_j}}$, there are unique m and π such that $S^*(i) = (\overline{\pi}(S_m))^*(i)$ for all $i \in [1, n_0]$. Let $X' = \pi^{-1}(X)$ and $T' = S_m \restriction X'$. We add a transition labelled with π:

– if $S(k') = 0$, from q to the initial state of the (X', T') component of $\nu(\!|\, \mathbb{C}' \,|\!)_{\mathrm{i}^{\Sigma_m}_{\rho_m}}$,
– if $S(k') \neq 0$, from q to the initial state of the (X', T') component of $\nu(\!|\, \mathbb{C} \,|\!)^{-}_{\mathrm{i}^{\Sigma_m}_{\rho_m}}$.

- Case of $\lambda x^{\mathsf{unit} \to \Theta_1}.\mathbb{C}$. We define $(\!|\, \lambda x.\mathbb{C} \,|\!)_{\mathrm{i}^{\Sigma_0}_{\rho_0}}$ as an automaton which combines states q_0, q_1, q_2 and two modified copies of $(\!|\, \mathbb{C} \,|\!)_{(\mathrm{i}\,\star)^{\Sigma}_{\rho'_0}}$, for each (of the finitely many relevant) Σ, in each of which we have replaced tags $\mathsf{r}_\downarrow$ by r_0 and removed all transitions with tags

[6] In fact, $(\!|\, \mathbb{C} \,|\!)_{(\mathrm{i}a)^{\Sigma'}_{\rho'_0}}$ is an (n_0+1, n)-automaton, but we render it into an (n_0, n) one by changing each $q_1 \xrightarrow{\pi} q_2$ to $q_1 \xrightarrow{\pi[n_0+1 \mapsto n_0+1]} q_2$.

c_i^x, r_i^x. We let X, S be derived from Σ and denote the two copies by $\widehat{(\!|\mathbb{C}|\!)}_S$, $\widetilde{(\!|\mathbb{C}|\!)}_S$. Each state q in $(\!|\mathbb{C}|\!)_{(\mathsf{i}\,\star)^{\Sigma}_{\rho_0'}}$ has copies $\hat{q}$, $\tilde{q}$ in $\widehat{(\!|\mathbb{C}|\!)}_S$, $\widetilde{(\!|\mathbb{C}|\!)}_S$ respectively.

$$\begin{tikzcd}
q_0 \arrow[r, "(\star,\mathsf{r}_\downarrow)^{S_0};\,\emptyset"] & q_1 \arrow[r, "\nu X.(\star,\mathsf{c}_0)^S"] & \widehat{(\!|\mathbb{C}|\!)}_S \arrow[l, bend left, "\texttt{done}"] \arrow[r, "\texttt{push}"] & q_2 \arrow[l, bend right, "\texttt{pop}"'] \arrow[r, "\nu X.(\star,\mathsf{c}_0)^S"] \arrow[r, bend left, "\texttt{pop}'"] & \widetilde{(\!|\mathbb{C}|\!)}_S \arrow[l, bend left, "\texttt{done}'"] \arrow[l, bend right=60, "\texttt{push}'"']
\end{tikzcd}$$

The unique final state is q_1. The transitions in typewriter font are defined as follows.

– We connect every final state of $\widehat{(\!|\mathbb{C}|\!)}_S$ with q_1 using a transition with label $\emptyset$ (done). Similarly for done′.

– For each sequence $q_A \xrightarrow{\nu X_A.(\ell_A,\mathsf{c}_i^x)^{S_A}/(s,\emptyset)\,;\,\pi} q_B \xrightarrow{\nu X_B.(\ell_B,\mathsf{r}_i^x)^{S_B},(s,\emptyset)} q_C$ in $(\!|\mathbb{C}|\!)_{(\mathsf{i}\,\star)^{\Sigma}_{\rho_0'}}$ we add $\hat{q}_A \xrightarrow{\nu X_A.(\ell_A,\mathsf{c}_i)^{S_A}/(\hat{q}_A,\pi)\,;\,\emptyset} q_2 \xrightarrow{\nu X_B.(\ell_B,\mathsf{r}_i)^{S_B},(\hat{q}_A,\emptyset)} \hat{q}_C$ (push, pop) and $\tilde{q}_A \xrightarrow{\nu X_A.(\ell_A,\mathsf{c}_i)^{S_A}/(\tilde{q}_A,\pi)\,;\,\emptyset} q_2 \xrightarrow{\nu X_B.(\ell_B,\mathsf{r}_i)^{S_B},(\tilde{q}_A,\emptyset)} \tilde{q}_C$ (push′, pop′).

The other cases of $\lambda x^{\beta\to\Theta_1}.\mathbb{C}$ are dealt with in a similar way. The case of $\lambda x^{\beta}.\mathbb{C}$ is treated as above, excluding q_2 and $\widetilde{(\!|\mathbb{C}|\!)}_S$ from the construction.

- For the case of $\mathsf{let}\, y = z(\lambda x^{\mathsf{unit}\to\Theta_1}.\mathbb{C})\,\mathsf{in}\,\mathbb{C}'$ it is useful to introduce a notion of automaton which operates by interleaving runs from two constituent automata. Since a similar construction will be of use in the next section, we give a general notion of automaton which can combine runs either by matching or by interleaving them. We define these generalised automata and give the construction of the one corresponding to $\mathsf{let}\, y = z(\lambda x^{\mathsf{unit}\to\Theta_1}.\mathbb{C})\,\mathsf{in}\,\mathbb{C}'$. Generalised automata can be reduced to equivalent ones.

Let $n_{\mathsf{r}} \le n_1, n_2$ and set $n' = n_1 + n_2 - n_{\mathsf{r}}$. For each $i \in [n_{\mathsf{r}}+1, n_2]$ we define its *shift* $i^+ = i + n_1 - n_{\mathsf{r}}$; note that $i^+ \in [n_1+1, n']$. We also fix a fresh label symbol $\checkmark$, which is used to indicate the automaton that will not advance in a given transition. For any set X, we write $X^{\checkmark}$ for $(X \uplus \{\checkmark\})^2 \backslash \{(\checkmark,\checkmark)\}$.

A *generalised* $(n_{\mathsf{r}}, n_1, n_2)$*-automaton* is given as a quintuple $\mathcal{A} = \langle Q, q_0, \rho_0, \delta, F\rangle$, where these components are defined as for an (n_{r}, n')-automaton except that now:

– $\delta \subseteq (Q_P \times (\mathsf{TL}_{\mathsf{push}} \cup \mathsf{TL}_{\mathsf{noop}})^{\checkmark} \times Q_O) \cup (Q_O \times (\mathsf{TL}_{\mathsf{pop}} \cup \mathsf{TL}_{\mathsf{noop}})^{\checkmark} \times Q_P) \cup (Q_O \times \mathsf{Mix}^{\checkmark} \times Q_O) \cup (Q_P \times \mathsf{Mix}^{\checkmark} \times Q_P)$,
– $\rho_0 \in \mathsf{Reg2}$ where $\mathsf{Reg2}$ contains all $\rho : [1,n'] \rightharpoonup \mathbb{A}$ such that $\rho \restriction [1,n_1]$ and $\rho \restriction ([1,n_{\mathsf{r}}] \cup [n_1{+}1, n'])$ are both injective.

Moreover, apart from the usual partitioning to O- and P-states, Q is partitioned to *normal* and *divergent* states: $Q = Q_N \uplus Q_D$. There is a map $\mathsf{div} : Q_N \cap Q_P \to Q_D \cap Q_P$, while $q_0 \in Q_N$.

The automaton operates on words over the alphabet $(\mathbb{C}\cup\mathbb{A}) \times \mathbb{T} \times \mathsf{Sto}$, with configurations given by tuples $(q, \rho, \sigma, H, \Sigma)$, where now $\rho \in \mathsf{Reg2}$, $\Sigma \in \mathsf{Sto}$ and the stack σ is an element of $\mathsf{Sta2} = (\mathbb{C}^{\checkmark}_{\mathsf{stack}} \times \mathsf{Reg2})^*$. We define projections and pairings for moving from the generalised setting to the ordinary one and vice versa:

– for each $\rho \in \mathsf{Reg2}$ let $\pi_1(\rho) = \rho \restriction [1,n_1]$ and $\pi_2(\rho) = \rho \restriction [1,n_{\mathsf{r}}] \cup \{(i, \rho(i^+)) \mid i \in [n_{\mathsf{r}}{+}1, n_2]\}$; moreover, for each $\rho_1, \rho_2 \in \mathsf{Reg}$ let $\langle \rho_1, \rho_2\rangle = \rho_1 \cup \{(i^+, \rho_2(i)) \mid i \in [n_{\mathsf{r}}+1, n_2]\}$;

– for each $(s_1, s_2, \rho) :: \sigma \in \mathsf{Sta2}$ we set $\pi_i((s_1, s_2, \rho) :: \sigma) = (s_i, \pi_i(\rho)) :: \pi_i(\sigma)$, and $\pi_i(\epsilon) = \epsilon$, for $i = 1, 2$; moreover, let $\langle (s_1, \rho_1) :: \sigma_1, (s_2, \rho_2) :: \sigma_2\rangle = (s_1, s_2, \langle \rho_1, \rho_2\rangle) :: \langle \sigma_1, \sigma_2\rangle$ and $\langle \epsilon, \epsilon\rangle = \epsilon$.

The automaton induces the following configuration graph. The initial configuration is $(q_0, \rho_0, \epsilon, \emptyset, \emptyset)$. For each $(q, \rho, \sigma, H, \Sigma)$ and $(q, \nu X_1.(\ell_1, t_1, \phi_1)^{S_1}, \nu X_2.(\ell_2, t_2, \phi_2)^{S_2}, q')$ $\in \delta$ if $q \xrightarrow{\nu X_i.(\ell_i,t_i,\phi_i)^{S_i}} q'$ induces $(q, \pi_i(\rho), \pi_i(\sigma), H) \xrightarrow{(l_i,t_i)^{\Sigma_i}} (q', \rho_i, \sigma_i, H_i)$ on an ordinary (n_{r}, n_i)-automaton where

1. $(l_1, t_1) = (l_2, t_2)$,
2. if $q \in Q_P$ then $\Sigma[\Sigma_1] \cup \Sigma[\Sigma_2]$ is well-defined,[7]
3. if $q \in Q_O$ then $\Sigma_1 \cup \Sigma_2$ is well-defined,

then $(q, \rho, \sigma, H, \Sigma) \xrightarrow{(l_1,t_1)^{\Sigma_1\cup\Sigma_2}} (q', \langle \rho_1, \rho_2\rangle, \langle \sigma_1, \sigma_2\rangle, H_1 \cup H_2, \Sigma_1 \cup \Sigma_2)$. Moreover, in case $q \in Q_P \cap Q_N$ and conditions 1,2 above cannot be satisfied by any combination of l_i, t_i, Σ_i then the automaton *diverges*, that is, $(q, \rho, \sigma, H, \Sigma) \xrightarrow{\epsilon} (\mathsf{div}(q), \rho, \sigma, H, \Sigma)$.

If $(q, \nu X_1.(\ell_1, t_1, \phi_1)^{S_1}, \checkmark, q') \in \delta$ with $(q, \pi_1(\rho), \pi_1(\sigma), H) \xrightarrow{(l_1,t_1)^{\Sigma_1}} (q', \rho_1, \sigma_1, H_1)$ in an (n_{r}, n_1)-automaton, we have $(q, \rho, \sigma, H, \Sigma) \xrightarrow{(l_1,t_1)^{\Sigma'}} (q', \langle \rho_1, \pi_2(\rho)\rangle, \langle \sigma_1, \sigma_2\rangle, H_1, \Sigma')$ where $\Sigma' = \Sigma[\Sigma_1] \upharpoonright \mathsf{cod}(\langle \rho_1, \pi_2(\rho)\rangle)$ and:

$$\sigma_2 = \begin{cases} (\checkmark, \emptyset) :: \pi_2(\sigma) & \text{if } t_1 \in \mathbb{T}_{\mathsf{push}} \\ \sigma' & \text{if } t_1 \in \mathbb{T}_{\mathsf{pop}} \text{ and } \pi_2(\sigma) = (s_2, \rho_2') :: \sigma' \\ \pi_2(\sigma) & \text{if } t_1 \in \mathbb{T}_{\mathsf{noop}} \end{cases}$$

For each $(q, \pi_1', \pi_2', q') \in \delta$ we have $(q, \rho, \sigma, H, \Sigma) \xrightarrow{\epsilon} (q', \langle \rho_1, \rho_2\rangle, \sigma, H, \Sigma')$ with $\rho_i = \pi_i(\rho) \circ \overline{\pi_i'}$ and $\Sigma' = \Sigma \upharpoonright \mathsf{cod}(\langle \rho_1, \rho_2\rangle)$. If $(q, \pi_1', \checkmark, q') \in \delta$ then $(q, \rho, \sigma, H, \Sigma) \xrightarrow{\epsilon} (q', \langle \rho_1, \pi_2(\rho)\rangle, \sigma, H, \Sigma \upharpoonright \mathsf{cod}(\langle \rho_1, \pi_2(\rho)\rangle))$. For each $(q, \checkmark, z, q') \in \delta$ we do the analogous of the symmetric case above.

Note in particular that if $\mathcal{A}$ only contains transitions which include $\checkmark$ then we can drop the component Σ in configurations and disregard divergence. In fact, any (n_{r}, n_1)-automaton can be rendered into an $(n_{\mathsf{r}}, n_1, n_2)$-automaton by simply changing each transition (q, z, q') to $(q, z, \checkmark, q')$. The dual applies to every (n_{r}, n_2)-automaton.

We now proceed with the case of $\mathsf{let}\, y = z(\lambda x^{\mathsf{unit}\to\Theta_1}.\mathbb{C})\,\mathsf{in}\,\mathbb{C}'$. The corresponding automaton will first read a move indicating that z is being called (tag c_r^z). After that, a detour to $\lambda x.\mathbb{C}$ will be an option, which O can initiate with a move tagged with $\mathsf{c}_r^{z,0}$. Modelling the detour is analogous to the interpretation of $\lambda x.\mathbb{C}$. Once the detour is completed or the possibility is not exercised, O can play a move tagged with r_r^z corresponding to the return by z (tagged r_r^z). This will trigger a transition to the automaton for $\mathbb{C}'$, in which we need to modify labels by replacing y with z and to allow for detours to $\lambda x.\mathbb{C}$ each time $\mathbb{C}'$ makes a transition on a P-move corresponding to y. Note that this is consistent with the behaviour of the corresponding strategy, due to the visibility and well-bracketing conditions.

Technically, for each Σ' and related S', X', ρ_0'', we consider two modified copies of $(\!|\, \mathbb{C}' \,|\!)_{\mathsf{i}\,\rho_0''}^{\Sigma'}$ in which:

– all O-states q for which there are $q_A \xrightarrow{\nu X_A.(\ell_A,t_A,\phi_A)^{S_A}\,;\,\pi} q \xrightarrow{\nu X_B.(\ell_B,t_B,\phi_B)^{S_B}} q_B$ in $(\!|\, \mathbb{C}' \,|\!)_{\mathsf{i}\,\rho_0''}^{\Sigma'}$,

[7] i.e. We stipulate that Σ_1, Σ_2 agree on common names, while values of private names remain unchanged. This stems from the fact that we do not compare plays but, rather, representations thereof.

with t_A having superscript y, are tagged as $q_{\mathbb{C}'}$;
- we replace tag superscripts y with z.

We set n_2 to be the maximum number of registers in these automata, and let $n = n_1 + n_2 - n_0$ with n_1 defined below. We denote the two copies by $\widehat{\llparenthesis \mathbb{C}' \rrparenthesis}_{S'}$ and $\widetilde{\llparenthesis \mathbb{C}' \rrparenthesis}_{S'}$ respectively and consider them to be (n_0, n_1, n_2)-automata. In the first one we write states as $\hat{q}$, while the other one has (the same) states in form $\tilde{q}$.

For each tagged state $\hat{q}_{\mathbb{C}'}$ of $\widehat{\llparenthesis \mathbb{C}' \rrparenthesis}_{S'}$, and each Σ and related X, S, ρ_0', consider the automaton $\llparenthesis \mathbb{C} \rrparenthesis_{(\mathrm{i}\star)^{\Sigma}_{\rho_0'}}$. We set n_1 to be the maximum number of registers in these automata. We define a modified copy of each $\llparenthesis \mathbb{C} \rrparenthesis_{(\mathrm{i}\star)^{\Sigma}_{\rho_0'}}$ by:

- removing all transitions of the form $q_A \xrightarrow{\nu X_A.(\ell_A, \mathsf{c}_i^x, \phi_A)^{S_A}\,;\,\pi} q_B \xrightarrow{\nu X_B.(\ell_B, \mathsf{r}_i^x, \phi_B)^{S_B}} q_C$;
- replacing all tags $\mathsf{r}_\downarrow$ by $\mathsf{r}^z_{r,0}$, where r is the arity of z.

For each q in $\llparenthesis \mathbb{C} \rrparenthesis_{(\mathrm{i}\star)^{\Sigma}_{\rho_0'}}$ we denote the resulting automaton by $\llparenthesis \mathbb{C} \rrparenthesis_S^{\hat{q}_{\mathbb{C}'}}$, considered as an (n_0, n_1, n_2)-automaton, and its states by $(q, \hat{q}_{\mathbb{C}'}, S)$. We construct another copy $\llparenthesis \mathbb{C} \rrparenthesis_S^{\tilde{q}_{\mathbb{C}'}}$ following the same routine, albeit for each state $\tilde{q}_{\mathbb{C}'}$ of $\widetilde{\llparenthesis \mathbb{C}' \rrparenthesis}_{S'}$.

We construct an (n_0, n_1, n_2)-automaton for $\llparenthesis \mathsf{let}\, y = z(\lambda x.\mathbb{C})\, \mathsf{in}\, \mathbb{C}' \rrparenthesis_{\mathrm{i}^{\Sigma_0}_{\rho_0}}$ as follows.

$$\begin{tikzcd}
q_0 \arrow[r, "(\star{,}\mathsf{c}_r^z)^{S_0}/(q_0{,}\emptyset)\,;\,\emptyset"] & q_1 \arrow[r, "\nu X.(\star{,}\mathsf{c}^z_{r,0})^S"] \arrow[dl, "\nu X'.(\star{,}\mathsf{r}_r^z)^{S'}{,}(q_0{,}\emptyset)"'] & \widehat{\llparenthesis \mathbb{C} \rrparenthesis}_S \arrow[l, bend left, "\texttt{done}"] \arrow[r, "\texttt{push}"] & q_2 \arrow[l, bend right=60, "\texttt{pop}"'] \arrow[r, "\nu X.(\star{,}\mathsf{c}^z_{r,0})^S"] \arrow[r, bend left=40, "\texttt{pop}'"] & \widetilde{\llparenthesis \mathbb{C} \rrparenthesis}_S \arrow[l, bend left, "\texttt{done}'"] \arrow[l, bend right=70, "\texttt{push}'"'] \\
\widehat{\llparenthesis \mathbb{C}' \rrparenthesis}_{S'} \arrow[r, "\nu X.(\star{,}\mathsf{c}^z_{r,0})^S"] & \llparenthesis \mathbb{C} \rrparenthesis_S^{\hat{q}_{\mathbb{C}'}} \arrow[l, bend left, "\texttt{done}"] \arrow[r, bend left=20, "\texttt{push}"] & \widetilde{\llparenthesis \mathbb{C}' \rrparenthesis}_{S'} \arrow[l, bend right=50, "\texttt{pop}"'] \arrow[r, "\nu X.(\star{,}\mathsf{c}^z_{r,0})^S"] \arrow[r, bend left=40, "\texttt{pop}'"] & \llparenthesis \mathbb{C} \rrparenthesis_S^{\tilde{q}_{\mathbb{C}'}} \arrow[l, bend left, "\texttt{done}'"] \arrow[l, bend right=70, "\texttt{push}'"'] &
\end{tikzcd}$$

All the arrows above represent transitions in the first partition of the automaton (but we have omitted the RHS ✓ for economy), and the same goes for all transitions inside subautomata involving $\mathbb{C}$. The transitions in subautomata coming from $\mathbb{C}'$ contribute to the second component. We connect each state $\hat{q}_{\mathbb{C}'}$ of $\widehat{\llparenthesis \mathbb{C}' \rrparenthesis}_{S'}$ with the initial state of $\llparenthesis \mathbb{C} \rrparenthesis_S^{\hat{q}_{\mathbb{C}'}}$ (for each relevant S, S'), using a transition with label $\nu X.(\star, \mathsf{c}^z_{r,0})^S$. Similarly for the transition of the same label between $\widetilde{\llparenthesis \mathbb{C}' \rrparenthesis}_{S'}$ and $\llparenthesis \mathbb{C} \rrparenthesis_S^{\tilde{q}_{\mathbb{C}'}}$. The transitions in typewriter font in the first line of the diagram are as in the previous case. Those of the second line are explained below.

- We connect every final state of each $\llparenthesis \mathbb{C} \rrparenthesis_S^{\hat{q}_{\mathbb{C}'}}$ with $\hat{q}_{\mathbb{C}'}$ using a transition with label $\emptyset$ (`done`). Similarly for `done`$'$.
- For each $q_{\mathbb{C}'}$, S and each $q_A \xrightarrow{\nu X_A.(\ell_A, \mathsf{c}_i^x)^{S_A}/(s,\emptyset)\,;\,\pi} q_B \xrightarrow{\nu X_B.(\ell_B, \mathsf{r}_i^x)^{S_B}, (s,\emptyset)} q_C$ in $\llparenthesis \mathbb{C} \rrparenthesis_{(\mathrm{i}\star)^{\Sigma}_{\rho_0'}}$ we add (`push`, `pop`)
$$(q_A, \hat{q}_{\mathbb{C}'}, S) \xrightarrow{\nu X_A.(\ell_A, \mathsf{c}^z_{r,i})^{S_A}/\phi_A\,;\,\emptyset} \hat{q}_{\mathbb{C}'} \xrightarrow{\nu X_B.(\ell_B, \mathsf{r}^z_{r,i})^{S_B}, \phi_B} (q_C, \hat{q}_{\mathbb{C}'}, S)$$
with $\phi_A = ((q_A, \hat{q}_{\mathbb{C}'}, S), \pi)$ and $\phi_B = ((q_B, \hat{q}_{\mathbb{C}'}, S), \emptyset)$.
We also add (`push`$'$, `pop`$'$) $(q_A, \tilde{q}_{\mathbb{C}'}, S) \xrightarrow{\nu X_A.(\ell_A, \mathsf{c}^z_{r,i})^{S_A}/\phi_A\,;\,\emptyset} \tilde{q}_{\mathbb{C}'} \xrightarrow{\nu X_B(\ell_B, \mathsf{r}^z_{r,i})^{S_B}, \phi_B} (q_C, \tilde{q}_{\mathbb{C}'}, S)$ setting $\phi_A = ((q_A, \tilde{q}_{\mathbb{C}'}, S), \pi)$ and $\phi_B = ((q_B, \tilde{q}_{\mathbb{C}'}, S), \emptyset)$.

The other cases of $\mathsf{let}\, y = z(\lambda x^{\beta\to\Theta_1}.\mathbb{C})\,\mathsf{in}\,\mathbb{C}'$ are dealt with in a similar way.

Observe that the basic cases in our construction yield strongly deterministic automata. Moreover, strong determinacy is preserved in the inductive cases. This is obvious in cases without interaction between different sub-automata. Moreover, in the cases of $\mathsf{let}\, x = \mathsf{ref}(y)\,\mathsf{in}\,\mathbb{C}$ and $(\mathsf{while}\,(!x)\,\mathsf{do}\,\mathbb{C}); \mathbb{C}'$, the connections between different components are made from states which end up having unique outgoing transitions. The same holds also for the cases of $\lambda x.\mathbb{C}$ and $\mathsf{let}\, y = z(\lambda x.\mathbb{C})\,\mathsf{in}\,\mathbb{C}'$, with the addition that now there are also new pop transitions to be taken into account which, however, operate on fresh stack symbols and therefore do not interfere with the constituent automata. In the latter case, the reduction from the generalised interleaving automaton to the ordinary one preserves strong determinacy.

Recall from Remark 28 that $(\!|\cdots|\!)$ and $[\![\cdots]\!]$ merely differ by the absence of initial moves in $(\!|\cdots|\!)$. Consequently, the constructions outlined above imply the following lemma.

Lemma 29 *Let $\Gamma \vdash \mathbb{C} : \theta$ be a* GRef☺*-term in canonical form. For each $j = \mathrm{i}^{\Sigma_0}_{\rho_0} \in I^{+}_{\Gamma\vdash\theta}$, there exists a deterministic $(|\nu(\mathrm{i})|, m_j)$-automaton $\mathcal{A}_j$ of type θ with initial register assignment ρ_0 such that $\bigcup_{w\in L(\mathcal{A}_j)} \mathsf{ext}(\mathrm{i}^{\Sigma_0} w) = \mathsf{comp}([\![\Gamma \vdash \mathbb{C} : \theta]\!]) \cap P^{\mathrm{i}^{\Sigma_0}}_{\Gamma\vdash\theta}$, where $P^{\mathrm{i}^{\Sigma_0}}_{\Gamma\vdash\theta}$ is the set of plays over $[\![\Gamma \vdash \theta]\!]$ that start from i^{Σ_0}.*

5.3 Reduction of inclusion into emptiness

The aim of this section is to establish the following result.

Lemma 30 *Let $\Gamma \vdash \mathbb{C}_1, \mathbb{C}_2 : \theta$ be* GRef☺*-terms in canonical form. For each $j = \mathrm{i}^{\Sigma_0}_{\rho_0} \in I^{+}_{\Gamma\vdash\theta}$, there exists a deterministic $(|\nu(\mathrm{i})|, n_j)$-automaton $\mathcal{B}_j$ with initial register assignment ρ_0 such that $L(\mathcal{B}_j) = \emptyset$ iff $\mathsf{comp}([\![\Gamma \vdash \mathbb{C}_1 : \theta]\!]) \cap P^{\mathrm{i}^{\Sigma_0}}_{\Gamma\vdash\theta} \subseteq [\![\Gamma \vdash \mathbb{C}_2 : \theta]\!]$.*

Suppose $\Gamma \vdash \mathbb{C}_1, \mathbb{C}_2 : \theta$ be GRef☺-terms in canonical form and let $\mathcal{A}_1$ and $\mathcal{A}_2$ be the automata representing their respective semantics for a given initial move i^{Σ_0}. We construct an automaton $\mathcal{A}'$ such that:

$$\Big(\mathsf{comp}([\![\Gamma \vdash \mathbb{C}_1 : \theta]\!]) \cap P^{\mathrm{i}^{\Sigma_0}}_{\Gamma\vdash\theta} \not\subseteq \mathsf{comp}([\![\Gamma \vdash \mathbb{C}_2 : \theta]\!]) \cap P^{\mathrm{i}^{\Sigma_0}}_{\Gamma\vdash\theta}\Big) \iff L(\mathcal{A}') \neq \emptyset$$

The idea behind the construction of $\mathcal{A}'$ is the following. We arrange so that the automaton behaves as a product automaton for $\mathcal{A}_1$ and $\mathcal{A}_2$, i.e. an automaton accepting their common strings (with possible extensions to stores on each side). The constructed automaton has the additional feature that, for each transition of $\mathcal{A}_1$, it checks whether the transition can be replicated by $\mathcal{A}_2$. If it can, then the automaton continues behaving as a product automaton. If the transition cannot be replicated then the automaton switches to *divergence mode* where it behaves as $\mathcal{A}_1$. The automaton accepts just if it reaches a final state in the divergence mode. The latter is justified by the fact that, in such a case, the automaton has detected a string in $\mathcal{A}_1$ which $\mathcal{A}_2$ cannot replicate and which leads to a complete play of $[\![\mathbb{C}_1]\!]$.

Suppose now each $\mathcal{A}_i$ is an (n_{r}, n_i)-automaton given by $\mathcal{A}_i = \langle Q_i, q_{0i}, \rho_{0i}, \delta_i, F_i\rangle$ and assume that $\rho_{01} = \rho_{02}$. We first construct a generalised $(n_{\mathsf{r}}, n_1, n_2)$-automaton $\mathcal{A} = \langle Q, q_0, \rho_0, \delta, F\rangle$ that operates equivalently to our target automaton $\mathcal{A}'$ described above. In particular, we set:

$$Q = Q_1 \cup (Q_{1O} \times Q_{2O}) \cup (Q_{1P} \times Q_{2P}),\quad q_0 = (q_{01}, q_{02}),\quad \rho_0 = \langle \rho_{01}, \rho_{02}\rangle,\quad F = F_1,$$

with $Q_N = (Q_{1O} \times Q_{2O}) \cup (Q_{1P} \times Q_{2P})$, $Q_D = Q_1$ and div the first projection. For each $(q_1, q_2) \in Q$ we include in δ precisely the following transitions.

– If $q_i \xrightarrow{\pi_i} q_i'$, for $i = 1, 2$, then $(q_1, q_2) \xrightarrow{\pi_1,\pi_2} (q_1', q_2')$. Otherwise, if $q_1 \xrightarrow{\pi} q_1'$ then $(q_1, q_2) \xrightarrow{\pi,\checkmark} (q_1', q_2)$, and if $q_2 \xrightarrow{\pi} q_2'$ then $(q_1, q_2) \xrightarrow{\checkmark,\pi} (q_1, q_2')$.
– If $q_i \xrightarrow{\nu X_i.(\ell_i,t_i,\phi_i)^{S_i}} q_i'$, $i = 1, 2$, then $(q_1, q_2) \xrightarrow{\nu X_1.(\ell_1,t_1,\phi_1)^{S_1},\nu X_2.(\ell_2,t_2,\phi_2)^{S_2}} (q_1', q_2')$.
– If $q_1 \xrightarrow{\nu X_1.(\ell_1,t_1,\phi_1)^{S_1}} q_1'$ then $q_1 \xrightarrow{\nu X_1.(\ell_1,t_1,\phi_1)^{S_1},\checkmark} q_1'$.
– If $q_1 \xrightarrow{\pi_1} q_1'$ then $q_1 \xrightarrow{\pi_1,\emptyset} q_1'$.

Lemma 31 *For $\mathbb{C}_1, \mathbb{C}_2, \mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{A}$ as above:*

$$(\mathsf{comp}([\![\Gamma \vdash \mathbb{C}_1 : \theta]\!]) \cap P^{\mathrm{i}^{\Sigma_0}}_{\Gamma\vdash\theta} \nsubseteq \mathsf{comp}([\![\Gamma \vdash \mathbb{C}_2 : \theta]\!]) \cap P^{\mathrm{i}^{\Sigma_0}}_{\Gamma\vdash\theta}) \iff L(\mathcal{A}) \neq \emptyset$$

Proof Suppose $w_{\mathcal{A}} \in L(\mathcal{A})$. By construction of $\mathcal{A}$, the accepting run for $w_{\mathcal{A}}$ yields an accepting run for w_1 in $\mathcal{A}_1$ via first projection. By Lemma 29, we have $\mathsf{ext}(\mathrm{i}^{\Sigma_0} w_1) \subseteq \mathsf{comp}([\![\Gamma \vdash \mathbb{C}_1 : \theta]\!])$. Consider $\mathrm{i}^{\Sigma_0} w \in P^{\mathrm{i}^{\Sigma_0}}_{\Gamma\vdash\theta}$ that extends $\mathrm{i}^{\Sigma_0} w_{\mathcal{A}}$. We have $\mathrm{i}^{\Sigma_0} w \in \mathsf{ext}(\mathrm{i}^{\Sigma_0} w_1)$ and thus $\mathrm{i}^{\Sigma_0} w \in \mathsf{comp}([\![\Gamma \vdash \mathbb{C}_1 : \theta]\!])$. On the other hand, via the second projection of the accepting run for $w_{\mathcal{A}}$ up to the point of switching to divergence mode, we obtain a run of $\mathcal{A}_2$ which, however, may only be resumed in $\mathcal{A}_2$ by a different (up to extension) P-move than that required by $w_{\mathcal{A}}$. Hence, $\mathrm{i}^{\Sigma_0} w \notin [\![\Gamma \vdash \mathbb{C}_2 : \theta]\!]$. Conversely, let $\mathrm{i}^{\Sigma_0} w \in \mathsf{comp}([\![\Gamma \vdash \mathbb{C}_1 : \theta]\!]) \backslash [\![\Gamma \vdash \mathbb{C}_2 : \theta]\!]$ and let $\mathrm{i}^{\Sigma_0} w'xy$ be its least prefix that appears in $[\![\Gamma \vdash \mathbb{C}_1 : \theta]\!] \backslash [\![\Gamma \vdash \mathbb{C}_2 : \theta]\!]$. By construction, our automata are closed under legal O- to P-transitions, so $\mathcal{A}_2$ must accept (a representation of) $\mathrm{i}^{\Sigma_0} w'x$ and fail to process y. On the other hand, $\mathcal{A}_1$ will be able to process a whole representation of $\mathrm{i}^{\Sigma_0} w$. Thus, $\mathcal{A}$ will operate as product automaton until $\mathrm{i}^{\Sigma_0} w'x$ and then enter divergence mode and continue as $\mathcal{A}_1$. Consequently, $\mathcal{A}$ will accept some $\mathrm{i}^{\Sigma_0} \tilde{w}$ such that $\mathrm{i}^{\Sigma_0} w \in \mathsf{ext}(\mathrm{i}^{\Sigma_0} \tilde{w})$, i.e. $L(\mathcal{A}) \neq \emptyset$. □

Determinacy extends to generalised automata in the obvious way: an automaton is deterministic if its configuration graph is. The notion of strong determinacy extends the following manner. By construction, the automaton $\mathcal{A}$ above is strongly deterministic.

Definition 32 Let $\mathcal{A} = \langle Q, q_0, \rho_0, \delta, F \rangle$ be a generalised $(n_{\mathsf{r}}, n_1, n_2)$-automaton. We say that $\mathcal{A}$ is *strongly deterministic* if:

– for each $q \in Q_P$ there is at most one transition out of q (i.e. $|\delta \upharpoonright \{q\}| \leq 1$), and if $(q, z_1, z_2, q') \in \delta$ with either of z_1, z_2 of the form $\nu X.(\ell, t, \phi)^S$ then $|\delta \upharpoonright \{q'\}| \leq 1$ and in particular q' may only have an outgoing transition (q', z_1', z_2', q'') such that $z_i' \in \mathsf{Mix}$ iff $z_i \neq \checkmark$ and, for all $(q'', z_1'', z_2'', q''') \in \delta$, $z_1'', z_2'' \notin \mathsf{Mix}$;
– for each $q \in Q_O$ and $(q, z_{i1}, z_{i2}, q_i) \in \delta$ with $z_{i1}, z_{i2} \notin \mathsf{Mix}$, $i = 1, 2$,
  – if $z_{11} \sim_\alpha z_{21}$ and $z_{12} \sim_\alpha z_{22}$ then $z_{11} = z_{21}$, $z_{12} = z_{22}$ and $q_1 = q_2$;[8]
  – if $z_{11}, z_{12} \neq \checkmark$ then $z_{21}, z_{22} \neq \checkmark$;
  – of $z_{11} = \checkmark \neq z_{12}$ and $z_{21} \neq \checkmark = z_{22}$ then z_{12} and z_{21} contain different tags;
– for each $(q, z_1, z_2, q') \in \delta$, if either of z_1, z_2 is of the form $\nu X.(\ell, t, \phi)^S$ then X is contained in $\mathsf{clo}(S, X_{\mathsf{Av}})$ where $X_{\mathsf{Av}} = (\mathsf{dom}(S) \backslash X) \cup \{j \mid \ell = \mathtt{R}_j\}$.

Lemma 33 *If $\mathcal{A}$ is strongly deterministic then it is deterministic.*

Proof Again, cases with P- to O-transitions and transitions with rearrangements are taken care of by the first condition above. The case of O- to P-transition in one component follows from

[8] Here we use the alpha-equivalence relation under the ν binder mentioned in Definition 24.

the second condition as in Lemma 25, using the first and last subcases of the second condition above. Finally, if such transitions happen in both components and induce, say, $\hat{q} \xrightarrow{(l,t)^{\Sigma}} \hat{q}_i$, $i = 1, 2$, where $\hat{q}$ has state q, let these be combination of transitions with labels $(l,t)^{\Sigma_{11}}$ and $(l,t)^{\Sigma_{12}}$, and of $(l,t)^{\Sigma_{21}}$ and $(l,t)^{\Sigma_{22}}$ respectively (here the second index specifies the component). We have $\Sigma = \Sigma_{11} \cup \Sigma_{12} = \Sigma_{21} \cup \Sigma_{22}$. Consider the associated $\nu X_{ij}.(\ell, t, \phi_{ij})^{S_{ij}}$, for $i, j = 1, 2$, and in particular the ordinary transitions $(q, \nu X_{i1}.(\ell, t, \phi_{i1})^{S_{i1}}, q_1)$. Since they are both accepting from $\hat{q}$, it must be that $\phi_{11} = \phi_{21}$. Moreover, as Σ_{11} and Σ_{21} agree on their common names, S_{11} and S_{21} may only disagree on X_{11}, X_{21}. But, by frugality, the latter are reachable from the indices of available registers and therefore $S_{11} = S_{21}$, modulo permutation of fresh indices. Thus, $\nu X_{11}.(\ell, t, \phi_{11})^{S_{11}} \sim_\alpha \nu X_{21}.(\ell, t, \phi_{21})^{S_{21}}$ and, similarly, $\nu X_{12}.(\ell, t, \phi_{12})^{S_{12}} \sim_\alpha \nu X_{22}.(\ell, t, \phi_{22})^{S_{22}}$. By strong determinacy we get $q_1 = q_2$ and thus $\hat{q}_1 = \hat{q}_2$. □

The operation of a generalised automaton can be faithfully simulated by a corresponding ordinary automaton. More precisely, we can show that from a given generalised automaton $\mathcal{A}$ we can effectively construct a *bisimilar* ordinary one.

Let G_1, G_2 be labelled directed graphs with nodes selected from sets of *configurations* $\hat{Q}_1, \hat{Q}_2$ respectively, and labels selected from the set $\{\epsilon\} \cup (\mathcal{L} \times \mathbb{T} \times \mathsf{Sto})$. Moreover, let each G_i have initial configuration $\hat{q}_{0i}$ and *final* configurations $\hat{Q}_{iF}$. We say that a relation $\mathcal{R} \subseteq \hat{Q}_1 \times \hat{Q}_2$ is a ***simulation*** if, for all $\hat{q}_1 \mathcal{R} \hat{q}_2$:

- if $\hat{q}_1 \in \hat{Q}_{1F}$ then $\hat{q}_2 \in \hat{Q}_{2F}$,
- if $\hat{q}_1 \xrightarrow{l}_{G_1} \hat{q}_1'$, some $l \in \{\epsilon\} \cup (\mathcal{L} \times \mathbb{T} \times \mathsf{Sto})$, then $\hat{q}_2 \xrightarrow{l}_{G_2} \hat{q}_2'$ for some $\hat{q}_1' \mathcal{R} \hat{q}_2'$.

We say that $\mathcal{R}$ is a ***bisimulation*** if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are simulations. Moreover, G_1 and G_2 are ***bisimilar***, written $G_1 \sim G_2$, if there is a bisimulation $\mathcal{R}$ such that $\hat{q}_{01} \mathcal{R} \hat{q}_{02}$.

In particular, we say that $\mathcal{A}$ and $\mathcal{A}'$ are bisimilar, written $\mathcal{A} \sim \mathcal{A}'$, if their configuration graphs are bisimilar.

Lemma 34 *Let $\mathcal{A}$ be a (n_r, n_1, n_2)-automaton and set $n' = n_1 + n_2 - n_\mathsf{r}$. We can effectively construct a (n_r, n')-automaton $\mathcal{A}'$ such that $\mathcal{A} \sim \mathcal{A}'$. Moreover, if $\mathcal{A}$ is strongly deterministic then so is $\mathcal{A}'$.*

The construction of $\mathcal{A}'$ is presented in Appendix B. Combining Lemmata 31 and 34, and using the fact that bisimilarity implies language equivalence, we obtain the following.

Lemma 35 *Let $\Gamma \vdash \mathbb{C}_1, \mathbb{C}_2 : \theta$ be $\mathsf{GRef}☺$-terms in canonical form. For each $j = \mathrm{i}_{\rho_0}^{\Sigma_0} \in I^+_{\Gamma\vdash\theta}$, there exists a deterministic $(|\nu(\mathrm{i})|, n_j)$-automaton $\mathcal{A}_j'$ with initial register assignment ρ_0 such that $L(\mathcal{A}_j') = \emptyset$ if and only if $\mathsf{comp}([\![\Gamma \vdash \mathbb{C}_1 : \theta]\!]) \cap P^{\mathrm{i}^{\Sigma_0}}_{\Gamma\vdash\theta} \subseteq \mathsf{comp}([\![\Gamma \vdash \mathbb{C}_2 : \theta]\!]) \cap P^{\mathrm{i}^{\Sigma_0}}_{\Gamma\vdash\theta}$.*

Lemma 30 then follows as a corollary.

5.4 Emptiness for fresh pushdown register automata

Returning to Lemma 30, note that, although $I^+_{\Gamma\vdash\theta}$ is an infinite set, there exists a finite subset $J \subseteq I^+_{\Gamma\vdash\theta}$ such that $\{\mathcal{A}_j\}_{j\in J}$ already captures $\mathsf{comp}([\![\Gamma \vdash \mathbb{C} : \theta]\!])$, because up to name-permutation there are only finitely many initial moves. Consequently, we only need finitely many of them to check whether $\Gamma \vdash \mathbb{C}_1 \lessapprox \mathbb{C}_2$. By Lemma 30, to achieve this we need to be able to decide the emptiness problem for (n_r, n)-automata.

To show decidability, we translate (n_{r}, n)-automata into an extended variant of pushdown register automata [7] (PDRA) over infinite alphabets. They are similar to (n_{r}, n)-automata in that they are equipped with registers and a stack. However, there are a few differences.

- PDRA can only process one name in a computational step, while (n_{r}, n)-automata read a label, a tag and a store in a single step. This can easily be overcome by decomposing transitions of our automata into a bounded number of steps (the existence of the bound follows from the fact that symbolic stores in our transition function are bounded).
- All registers in PDRA must be full, while (n_{r}, n)-automata admit empty registers. This difference can be compensated by populating registers with dummy names, while storing information about which register is deemed to be empty in the finite state.
- (n_{r}, n) allow for rearrangements of registers in a single ϵ-step. Again this can be decomposed into a sequence of ϵ-transitions of an PDRA by using the stack.
- (n_{r}, n) have the ability to create *globally* fresh names (guaranteed not to have been encountered in the whole computational run), while PDRA can only create *locally* fresh names (through the so-called reassignment), which are guaranteed not to occur in the present register assignment. This discrepancy cannot be dealt with easily and we provide a separate argument why the emptiness problems for the extension of PDRA with (globally) fresh-name generation remains decidable.

We start off with a generalisation of pushdown register automata [7] to data words. That is to say, in a non-epsilon step, the automaton reads a pair consisting of a tag (taken from a finite set of tags) and a value, which comes from the infinite alphabet. The latter will often be referred to as a *name*. Such pairs are also pushed on the stack. Decidability of emptiness in absence of fresh-name generation was already shown in [7] in the tag-free case, but the generalisation to tags is rather cosmetic, since they can be emulated with fixed names. Let Σ be an infinite alphabet and $\mathcal{T}$ a finite set of tags with a distinguished bottom-of-stack tag $\perp$.

Definition 36 An r-register pushdown automaton (PDRA) $\mathcal{A}$ over $(\Sigma, \mathcal{T})$ is a tuple $\langle Q, q^{in}, u, \rho, \mu, F\rangle$, where

- Q is a finite set of *states*;
- $q^{in} \in Q$ is the *initial state*;
- $u : \{1, \cdots, r\} \to \Sigma$ is an injection, called the *initial assignment*;
- $\rho : Q \to \{1, \cdots, r\}$ is a partial function called the *reassignment*;
- μ is the *transition relation*, which is a mapping from

$$Q \times ((\mathcal{T} \times \{1, \cdots, r\}) \cup \{\epsilon\}) \times (\mathcal{T} \times \{1, \cdots, r\})$$

  to finite subsets of $Q \times (\mathcal{T} \times \{1, \cdots, r\})^*$;
- $F \subseteq Q$ is the set of *final states*.

A *configuration* is a triple (q, R, S) such that $q \in Q$, $R : \{1, \cdots, r\} \to \Sigma$ is injective and $S \in (\mathcal{T} \times \Sigma)^*$. The last component represents the stack content (the leftmost symbol stands for the top of the stack). A configuration is *initial* if $q = q^{in}$, $R = u$ and $S = [(\perp, u(r))]$. A configuration is *final* if $q \in F$. A *generalised run* of $\mathcal{A}$ is a sequence $C_0 \xrightarrow{x_1} \ldots \xrightarrow{x_k} C_k$ such that each $C_j = (q_i, R_i, S_i)$ $(0 \leq j \leq k)$ is a configuration, $q_0 = q^{in}$, $S_0 = [(\perp, R_0(r))]$, each $x_j \in \{\epsilon\} \cup (\mathcal{T} \times \Sigma)$ $(1 \leq j \leq k)$, and each C_j $(0 < j \leq k)$ is obtained from C_{j-1} and x_j according to μ. Note that in generalised runs R_0 is left unspecified. This will make working with them slightly simpler, because they will be closed under bijective renamings. A *run* is simply a generalised run such that C_0 is initial. In an *accepting run* we also have $q_k \in F$.

We shall say that two generalised runs are *related* if they are of the same length and the corresponding steps rely on the same elements of μ. Consequently, related generalised runs differ only in the names they involve (respective states and tags must be the same). We will be interested in characterizing generalised runs that are related to each other. Below we introduce several definitions that will make this possible.

Suppose $\mathcal{R}$ is a generalised run of length k. Let us assign *timestamps* from the set $\{1, \cdots, r+k\}$ to all occurrences of names in $\mathcal{R}$. They will indicate at which step the names were introduced into the run. Note that this can happen only in two ways: via the initial assignment in C_0 or reassignment through ρ. In other cases timestamps will be inherited from previous steps. Thus we can associate timestamps with occurrences of names in generalised runs as follows:

– the occurrence of $R_0(i)$ in R_0 is timestamped with i for each $1 \leq i \leq r$, the occurrence of $R_0(r)$ in S_0 is timestamped with r;
– if a name is generated in step $1 \leq j \leq k$ through reassignment, its occurrence in R_j gets timestamp $r + j$, otherwise occurrences of names in registers inherit timestamps from preceding configurations;
– a name that was just pushed on the stack inherits the timestamp it had in the register before being pushed, the timestamps of other names on the stack are inherited from preceding configurations.

In what follows we shall refer to timestamps using notation such as $t_{R_j(i)}$ or $t_{top(S_j)}$.

Let us write $\mathbb{T} \subseteq \{1, \cdots, r+k\}$ for the set of all timestamps used to mark occurrences of names of $\mathcal{R}$.

Remark 37 Note that, due to the stack discipline, whenever the same names are present on the stack in a configuration of a run, the associated sequence of timestaps, from top to bottom, must be non-increasing. Additionally, if the same name occurs in a register, its timestamp will not be smaller than the timestamps of occurrences of the same name on the stack.

We can replace all occurrences of names in $\mathcal{R}$ with their timestamps to obtain what we shall call a *symbolic run*: $C_0^{sym} \xrightarrow{x_1^{sym}} \cdots \xrightarrow{x_k^{sym}} C_k^{sym}$. Recall that states and tags will remain the same as in $\mathcal{R}$. Observe the following fact.

Lemma 38 *Two generalised runs are related if and only if the corresponding symbolic runs are the same.*

Next we characterize assignments $\alpha : \mathbb{T} \to \Sigma$ (of names to timestamps), which can be used to convert a symbolic run generated from $\mathcal{R}$ into a generalised run related to $\mathcal{R}$.

– First, the initial register assignment must be injective: $\alpha(i) \neq \alpha(j)$ for $1 \leq i < j \leq r$.
– The second set of constraints comes from reassignment steps, where it must be ensured that the reassigned name is different from those currently present in registers: if $\rho(q_i)$ $(0 \leq i < k)$ is defined then we require that $\alpha(t_{R_{i+1}(\rho(q_i))}) \neq \alpha(t_{R_i(j)})$ for any $1 \leq j \leq r$.
– The third set of constraints is induced by popping during transitions: the name on top of the stack must match the content of a suitable register, as specified by μ: if the passage from q_i to q_{i+1} $(0 \leq i < k)$ relies on an element of $\mu(q_i, (t_1, i_1), (t_2, i_2))$ or $\mu(q_i, \epsilon, (t_2, i_2))$ then $\alpha(t_{top(S_i)}) = \alpha(t_{R_{i+1}(i_2)})$.

Altogether, above we have extracted from $\mathcal{R}$ a set of conditions characterizing runs related to $\mathcal{R}$, in terms of which names introduced into a run must be equal or unequal.

Lemma 39 *Generalised runs related to $\mathcal{R}$ are in 1–1 correspondence with α:$\mathbb{T} \to \Sigma$ satisfying the above constraints.*

Below we single out a special family of such maps. Intuitively, we shall focus on generalised runs in which as many names as possible are used.

Let us define $=_{\mathcal{R}}$ to be the smallest equivalence relation such that if '$\alpha(i) = \alpha(j)$' belongs to the third set of constraints we have $i =_{\mathcal{R}} j$. $\alpha' : \mathbb{T} \to \Sigma$ will be called *distinctive* if, for all $i, j \in \mathbb{T}$, we have $\alpha'(i) = \alpha'(j)$ if and only if $i =_{\mathcal{R}} j$.

Remark 40 Note that every distinctive α' satisfies the first two kinds of constraints and, hence, gives rise to a generalised run. By definition distinctive maps α' exist and are determined uniquely up to name-permutation. In particular, there exists a distinctive α' that is compatible with the initial register assignment in $\mathcal{R}$.

The following property of distinctive maps will play a role in a future argument.

Lemma 41 *Let α' be distinctive, $i \in \mathbb{T}$ and $k_i = \min\{j \mid i =_{\mathcal{R}} j\}$. Then for all $j < k_i$ we have $\alpha'(j) \neq \alpha'(k_i) = \alpha'(i)$. In other words, $\alpha'(k_i)$ is fresh.*

Proof Take $j < k_i$. By definition of k_i, it is not the case that $i =_{\mathcal{R}} j$. Because α' is distinctive, we must have $\alpha'(j) \neq \alpha'(k_i)$. On the other hand, $k_i =_{\mathcal{R}} i$, so by distinctiveness $\alpha'(k_i) = \alpha'(i)$. □

Definition 42 A fresh PDRA (FPDRA) $\mathcal{A}$ over $(\Sigma, \mathcal{T})$ is defined in the same way as a PDRA except that the (partial) reassignment function has the form $\rho : Q \to \{1, \cdots, r\} \times \{L, G\}$. Whenever $\pi_2(\rho(q)) = L$ (locally fresh), $\mathcal{A}$ must generate a name that is currently not present in registers. If $\pi_2(\rho(q)) = G$ (globally fresh) then the name must in addition not have occurred before in the present run (in particular, it will not occur on the stack).

We shall show that the emptiness problem for FPDRA is decidable by referring to the analogous result for PDRA [7]. Let $\mathcal{A} = \langle Q, q^{in}, u, \rho, \mu, F \rangle$ be a FPDRA over $(\Sigma, \mathcal{T})$. Next we are going to define an PDRA $\mathcal{A}' = \langle Q', q^{in'}, u', \rho', \mu', F' \rangle$ such that there exists an accepting run of $\mathcal{A}$ iff there exists one for $\mathcal{A}'$.

$\mathcal{A}'$ will mimic steps made by $\mathcal{A}$ except that it will not be able to generate globally fresh names, so it will use locally fresh ones instead. In addition, $\mathcal{A}'$ will maintain information about such names by flagging registers and stack elements.

– A flagged register signifies the fact that in $\mathcal{A}$ the generated name would be different from any name currently occurring on the stack.
– A flagged name on the stack means that in $\mathcal{A}$ all names underneath it would be different.

Register flags will be stored inside the extended state, flags for names on the stack will be assigned by appending F to the associated tag. The way the flags are managed is described below.

– Whenever a locally fresh name is generated instead of a globally fresh name, the corresponding register will remain flagged as long as its content is not pushed on the stack.
– When a sequence of names is pushed on the stack, for each name that comes from a flagged register, we flag the rightmost (deepest) of its occurrences in the sequence. The registers in question become untagged.
– A name can be popped only if it occurs in an unflagged register. If the name is flagged on the stack, we then add a flag to the register. Note that we do not allow $\mathcal{A}'$ to follow $\mathcal{A}$, if this would entail popping a symbol that occurs in a flagged register.

Next we present the construction formally. $\mathcal{A}'$ will be an PDRA over $(\Sigma, \mathcal{T}')$, where:

$$\mathcal{T}' = \mathcal{T} \cup (\mathcal{T} \times \{F\}) \qquad q^{in'} = (q^{in}, \emptyset) \qquad F' = F \times 2^{\{1,\cdots,r\}}$$
$$Q' = Q \times 2^{\{1,\cdots,r\}} \qquad u' = u \qquad \rho' = \rho \circ \pi_1$$

To define μ', we use t_1, t_2 to range over $\mathcal{T}$ and $t_2^f, c_1^f, \cdots, c_n^f$ for elements of $\mathcal{T}'$. Whenever we use t^f to refer to elements of $\mathcal{T}'$, by dropping the superscript and writing t we mean to refer to the underlying tag from $\mathcal{T}$.

$$((p, Y), (c_1^f, j_1) \cdots (c_n^f, j_n)) \in \mu'((q, X), (t_1^f, i_1), (t_2^f, i_2))$$

if $(p, (c_1, j_1) \cdots (c_n, j_n)) \in \mu(q, (t_1, i_1), (t_2, i_2))$ and $i_2 \notin X$, $Y = X' \backslash \{j_1, \cdots, j_n\}$ where

$$X' = X \cup \{\pi_1(\rho(q)) \mid \pi_2(\rho(q)) = G\} \cup \{i_2 \mid t_2^f \in \mathcal{T} \times \{F\}\},$$

and, for all $i = 1, \cdots, n$,

$$c_i^f = \begin{cases} (c_i, F) & j_i \in X', \ \forall_{i<l\leq n}\ j_l \neq j_i \\ c_i & \text{otherwise.} \end{cases}$$

Lemma 43 *$\mathcal{A}$ has an accepting run if and only if $\mathcal{A}'$ has one.*

Proof Suppose there exists an accepting run of $\mathcal{A}$. The same run is then accepting for $\mathcal{A}'$. This is because whenever a register is flagged in $\mathcal{A}'$, it will indeed contain a name not present on the stack, and if a name is flagged on the stack the name will not occur below. Consequently, whenever a step of $\mathcal{A}$ depends on matching a name on top of the stack with a register, the corresponding register in $\mathcal{A}'$ will not be flagged. Thus, every step of $\mathcal{A}$ can be simulated by $\mathcal{A}'$.

Suppose we have an accepting run $\mathcal{R}$ of $\mathcal{A}'$. As $\mathcal{A}'$-transitions are transitions of $\mathcal{A}$ enriched with some information, it suffices to show that the locally fresh names generated instead of globally fresh names are globally fresh. This need not be the case for $\mathcal{R}$, but will show how to rename $\mathcal{R}$ so that another $\mathcal{A}'$-run emerges, which does enjoy the property.

Let us consider a step of $\mathcal{R}$ in which a locally fresh name is used as a substitute for a globally fresh one. Let t be the timestamp assigned to that name. We show that $t' =_{\mathcal{R}} t$ implies $t' \geq t$. For a start, we note that the '$\alpha(i) = \alpha(j)$' constraints generated before time t concern i, j strictly smaller than t. We show that all the equational constraints that affect $[t]_{=_{\mathcal{R}}}$ later concern timestamps that are at least t. We analyze the history of the name timestamped t during the run.

- Unless the name is pushed immediately on the stack, it will stay flagged in registers for a number of steps. Thus no pops will rely on it. Hence no constraints using $\alpha(t)$ will be generated.
- After the name is pushed on the stack, its deepest occurrence in the push sequence will be flagged. The register will be unflagged but, as long as the flagged occurrence remains on the stack, all equational constraints that rely on the same name will involve timestamps greater than or equal to t (see Remark 37).
- When the flagged occurrence with timestap t is eventually popped (if at all), there must be a register containing the same name, but this occurrence will bear a timestamp t' such that $t' \geq t$ and will become flagged. From then on the reasoning can be repeated for t' and, since $t' \geq t$, we can conclude that future constraints affecting $[t']_{=_{\mathcal{R}}} = [t]_{=_{\mathcal{R}}}$ will not involve timestamps smaller than t'.

So, for each timestamp t corresponding to a globally fresh name in $\mathcal{A}$, we have $[t]_{=_{\mathcal{R}}} \subseteq [t, \infty)$. Consequently, by Remark 40, in every related distinctive run, names with that timestamp will indeed be different from all names used by the automaton earlier, i.e. they will be globally fresh. Moreover, there exists a distinctive run whose initial assignment is u. That run is thus also a run of $\mathcal{A}$ (after erasing the flag information). □

The above result reduces the emptiness problem for FPDRA to the analogous problem for PDRA. As the latter is decidable [7], we obtain the following.

Lemma 44 *The emptiness problem for FPDRA is decidable.*

Finally, summing up, we obtain the desired decidability result.

Theorem 45 *Program approximation (and thus program equivalence) is decidable for* GRef☺*-terms.*

Proof Let $\Gamma \vdash M_1, M_2 : \theta$ be GRef☺-terms. By Lemma 2, they can be converted into canonical forms $\Gamma \vdash \mathbb{C}_{M_1}, \mathbb{C}_{M_2} : \theta$ such that $\Gamma \vdash M_1 \lesssim M_2$ if and only if $\Gamma \vdash \mathbb{C}_{M_1} \lesssim \mathbb{C}_{M_2}$. By Lemma 30 and our observations at the beginning of Sect. 5.4, the problem of determining whether $\Gamma \vdash \mathbb{C}_{M_1} \lesssim \mathbb{C}_{M_2}$ holds can be reduced to the emptiness problem for a finite number of FPDRA, all effectively constructible from $\mathbb{C}_{M_1}, \mathbb{C}_{M_2}$. Because FPDRA emptiness is decidable by Lemma 44, program approximation is decidable for GRef☺. □

6 Related and further work

In this paper we achieved a full characterisation of decidability of program equivalence in GRef, a higher-order language with full ground storage. Moreover, for the decidable fragment that we identified, we devised a decidability procedure which builds on automata-based representations of terms.

The investigations into models and reasoning principles for storage have a long history. In this quest, storage of names was regarded by researchers as an indispensable intermediate step towards capturing realistic languages with dynamic-allocated storage, such as ML or Java. Relational methods and environmental bisimulations for reasoning about program equivalence in settings similar to ours were studied in [2,5,9,16,30,31], albeit without decidability results. More foundational work included labelled transition system semantics [15] and game semantics [19,24]. In both cases, it turned out that the addition of name storage simplified reasoning, be it bisimulation-based or game-semantic. In the former case, bisimulation was even unsound without full ground storage. In the latter case, the game model of integer storage [22] turned out more intricate (complicated store abstractions) than that for full ground or general storage [19,24].

As for decidability results, we studied finitary Reduced ML (integer storage only) in [23], yet only judgements of the form $\cdots, \beta \to \beta, \cdots \vdash \beta$ were tackled due to intricacies related to store abstractions (in absence of full ground storage, names cannot be remembered by programs). A closely related language, called RML [1] (integer storage but with bad references) was studied in [8,14,27], but no full classification has emerged yet. Interestingly, closed terms of first-order type become decidable in this setting [8], in contrast to $\mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}$ for GRef. Finally, the approach presented herein was pursued for Interface Middleweight Java [21] and implemented in the equivalence checker Conneqct [20]. We note that, in presence of higher-order references and boolean storage, even termination is undecidable [26].

Apart from the semantics and programming languages community, program equivalence has been extensively examined in the context of regression verification [4,6,10,12,18]. There, the focus is put on sound methods for proving that newer versions of the same code fragment do not introduce new behaviours, outside the code specification. In terminology used in this paper, the property that is being verified is program approximation. Regression verification is based on strong abstraction techniques which are sound for approximation, and is typically applied to ground languages and code (i.e. no higher-order functions). In the future, we would like to expand the use of game models for checking equivalence to larger language fragments, where equivalence in undecidable. In doing so, we would abandon decidability and restrict ourselves to sound verification routines, using similar abstraction techniques.

Acknowledgements Partially supported by the Royal Academy of Engineering (RF 10216/111) and the EPSRC (EP/P004172/1).

A Canonical forms for GRef

In this section we prove Lemma 2:

Let $\Gamma \vdash M : \theta$ be an GRef*-term. There exists a* GRef*-term $\Gamma \vdash \mathbb{C}_M : \theta$ in canonical form, effectively constructible from M, such that $\Gamma \vdash M \cong \mathbb{C}_M$.*

We prove two auxiliary results first, both of which are special cases of Lemma 2.

Lemma 46 *Any identifier x^θ satisfies Lemma 2. Moreover, when $\theta \equiv \theta_1 \rightarrow \theta_2$, the canonical form is of the form $\lambda y^{\theta_1}.\mathbb{C}$.*

Proof Induction with respect to type structure. If θ is a base type, we can take $\mathbb{C}_{x^{\mathsf{unit}}}$ to be $()$ and $\mathbb{C}_{x^{\mathsf{int}}}$ to be $\mathsf{case}(x^{\mathsf{int}})[0, \cdots, max]$. $x^{\mathsf{ref}\,\gamma}$ is already in canonical form. For $\theta \equiv \theta_1 \rightarrow \theta_2$ we use the η-expansion rule

$$x^{\theta_1 \rightarrow \theta_2} \cong \lambda z^{\theta_1}.\mathsf{let}\ y^{\theta_2} = x z^{\theta_1}\ \mathsf{in}\ y$$

and appeal to the inductive hypothesis for z^{θ_1} and y^{θ_2}. This suffices for $\theta_1 \equiv \mathsf{unit}, \mathsf{ref}\,, \theta_1' \rightarrow \theta_2'$. For $\theta_1 \equiv \mathsf{int}$ we additionally observe that

$$\begin{gathered}\mathsf{let}\ y = x(\mathsf{case}(z)[0, \cdots, max])\ \mathsf{in}\ y \\ \cong \\ \mathsf{case}(z)[\mathsf{let}\ y = x\,0\ \mathsf{in}\ y, \cdots, \mathsf{let}\ y = x\ max\ \mathsf{in}\ y].\end{gathered}$$

□

Lemma 47 *Suppose $\mathbb{C}_1, \mathbb{C}_2$ are canonical forms. Then* $\mathsf{let}\ y^\theta = \mathbb{C}_1\ \mathsf{in}\ \mathbb{C}_2$*, if typable, satisfies Lemma 2.*

Proof Induction with respect to the structure of θ.

– $\theta \equiv \mathsf{unit}$. Using the equivalences listed below, we reason by induction on the structure of $\mathbb{C}_1$, which can take one of the following shapes: $()$, $\mathsf{case}(x^{\mathsf{int}})[C_0, \cdots, C_{max}]$ or $\mathsf{let} \cdots \mathsf{in}\ C$[9]. Note that free identifiers of type unit never occur in canonical forms, hence the first equivalence.

$$\begin{aligned}\mathsf{let}\ y = ()\ \mathsf{in}\ \mathbb{C}_2 &\cong \mathbb{C}_2 \\ \mathsf{let}\ y = \mathsf{case}(x^{\mathsf{int}})[C_0, \cdots, C_{max}]\ \mathsf{in}\ \mathbb{C}_2 &\cong \mathsf{case}(x)[\mathsf{let}\ y = C_0\ \mathsf{in}\ \mathbb{C}_2, \cdots, \mathsf{let}\ y = C_{max}\ \mathsf{in}\ \mathbb{C}_2] \\ \mathsf{let}\ y = (\mathsf{let} \cdots \mathsf{in}\ C)\ \mathsf{in}\ \mathbb{C}_2 &\cong \mathsf{let} \cdots \mathsf{in}\ (\mathsf{let}\ y = C\ \mathsf{in}\ \mathbb{C}_2)\end{aligned}$$

[9] Cases of the form $(\cdots); \mathbb{C}$ are covered by the $\mathsf{let} \cdots \mathsf{in}\ \mathbb{C}$ clause.

- $\theta \equiv \mathsf{int}$. We reason by induction on the structure of $\mathbb{C}_1$, which can take one of the following shapes: i, $\mathsf{case}(x^{\mathsf{int}})[C_0, \cdots, C_{max}]$ or $\mathsf{let} \cdots \mathsf{in}\, C$. The last two cases are dealt with as before. For $\mathsf{let}\, y = i \,\mathsf{in}\, \mathbb{C}_2$, note that, whenever identifiers y^{int} occur freely in canonical forms, it is always as part of a subterm of the form $\mathsf{case}(y^{\mathsf{int}})[\mathbb{C}_0, \cdots, \mathbb{C}_{max}]$. Given a canonical form $\mathbb{C}$ we write $\mathbb{C}^{y,i}$ for the canonical term obtained from $\mathbb{C}$ by replacing each of its subterms of the form $\mathsf{case}(y^{\mathsf{int}})[\mathbb{C}_0, \cdots, \mathbb{C}_{max}]$ with $\mathbb{C}_i$ (i.e. by removing y and branches corresponding to $y \neq i$). Then
$$\mathsf{let}\, y = i \,\mathsf{in}\, \mathbb{C}_2 \cong \mathbb{C}_2^{y,i}.$$
- $\theta \equiv \mathsf{ref}\,\gamma$. We reason by induction on the structure of $\mathbb{C}_1$, which can take one of the following shapes: $x^{\mathsf{ref}\,\gamma}$, $\mathsf{case}(x)[C_0, \cdots, C_{max}]$ or $\mathsf{let} \cdots \mathsf{in}\, C$. The last two inductive cases are dealt with as before. For $\mathsf{let}\, y = x^{\mathsf{ref}\,\gamma} \,\mathsf{in}\, \mathbb{C}_2$ we observe that $\mathbb{C}_2[x/y]$ is in canonical form and
$$\mathsf{let}\, y = x^{\mathsf{ref}\,\gamma} \,\mathsf{in}\, \mathbb{C}_2 \cong \mathbb{C}_2[x/y].$$
- $\theta \equiv \theta_1 \to \theta_2$. As earlier, we reason by induction on the structure of $\mathbb{C}_1$, which can now take one of the following shapes: $\lambda x_1^{\theta}.C$, $\mathsf{case}(x^{\mathsf{int}})[C_0, \cdots, C_{max}]$ or $\mathsf{let} \cdots \mathsf{in}\, C$. We can deal with the last two cases as before. The first one requires a separate argument, though.
Suppose $\mathbb{C}_1 \equiv \lambda x_1^{\theta_1}.C_1$. Let us substitute $\mathbb{C}_1$ for the rightmost occurrence of y in $\mathbb{C}_2$. This will create a non-canonical subterm in $\mathbb{C}_2$ of the form $\mathsf{let}\, x^{\theta_2} = (\lambda x_1^{\theta_1}.C_1)C \,\mathsf{in}\, C_2 \equiv \mathsf{let}\, x^{\theta_2} = (\mathsf{let}\, x_1^{\theta_1} = C \,\mathsf{in}\, C_1) \,\mathsf{in}\, C_2$. By inductive hypothesis for θ_1, $\mathsf{let}\, x_1^{\theta_1} = C \,\mathsf{in}\, C_1$ can be converted to canonical form, say, C_3. Consequently, the non-canonical subterm $\mathsf{let}\, x^{\theta_2} = (\lambda x_1^{\theta_1}.C_1)C \,\mathsf{in}\, C_2$ can be converted to $\mathsf{let}\, x^{\theta_2} = C_3 \,\mathsf{in}\, C_2$, which — by inductive hypothesis for θ_2 — can also be converted to canonical form. Thus, we have shown how to recover canonical forms after substitution for the rightmost occurrence of y. Because of the choice of the rightmost occurrence, the transformation does not involve terms containing other occurrences of y, so it will also decrease their overall number in $\mathbb{C}_2$ by one. Consequently, by repeated substitution for rightmost occurrences one can eventually arrive at a canonical form for $\mathsf{let}\, y^{\theta} = (\lambda x_1^{\theta_1}.C_1) \,\mathsf{in}\, \mathbb{C}_2$. □

Now we are ready to prove Lemma 2.

Proof (Lemma 2) We proceed by induction on term structure.

- $()$, i are already in canonical form.
- For $\mathsf{case}(M)[N_0, \cdots, N_{max}]$ we simply appeal to Lemma 47 for
$$\mathsf{let}\, x^{\mathsf{int}} = \mathbb{C}_M \,\mathsf{in}\, \mathsf{case}(x)[\mathbb{C}_{N_0}, \cdots, \mathbb{C}_{N_{max}}].$$
- For $!M$ we apply Lemma 47 to
$$\mathsf{let}\, x^{\mathsf{ref}\,\gamma} = \mathbb{C}_M \,\mathsf{in}\, (\mathsf{let}\, y^{\gamma} = !x \,\mathsf{in}\, \mathbb{C}_{y^{\gamma}}).$$
- For $M := N$, depending on whether N's type is int or $\mathsf{ref}\,\gamma$, we appeal to Lemma 47 for
$$\mathsf{let}\, y^{\mathsf{int}} = \mathbb{C}_N \,\mathsf{in}\, \mathsf{case}(y)[(x^{\mathsf{ref\,int}} := 0);\, (), \cdots, (x^{\mathsf{ref\,int}} := max);\, ()]$$
or
$$\mathsf{let}\, y^{\mathsf{ref}\,\gamma} = \mathbb{C}_N \,\mathsf{in}\, ((x^{\mathsf{ref\,ref}\,\gamma} := y^{\mathsf{ref}\,\gamma});\, ())$$
to obtain $\mathbb{C}$ and then invoke the same lemma for $\mathsf{let}\, x = \mathbb{C}_M \,\mathsf{in}\, \mathbb{C}$.

- For $\mathsf{ref}(M)$, depending on whether M's type is int or $\mathsf{ref}\,\gamma$, we invoke Lemma 47 for

$$\mathsf{let}\, y^{\mathsf{int}} = \mathbb{C}_M \,\mathsf{in}\,\mathsf{let}\, x = \mathsf{ref}(0)\,\mathsf{in}\,\mathsf{case}(y)[(x := 0); x, \cdots, (x := max); x].$$

  or

$$\mathsf{let}\, y^{\mathsf{ref}\,\gamma} = \mathbb{C}_M \,\mathsf{in}\,\mathsf{let}\, x^{\mathsf{ref}\,\mathsf{ref}\,\gamma} = \mathsf{ref}(y^{\mathsf{ref}\,\gamma})\,\mathsf{in}\,\mathbb{C}_x.$$

- For $\mathsf{while}\, M\,\mathsf{do}\, N$, we note that $\mathsf{while}\, M\,\mathsf{do}\, N$ is equivalent to

$$\mathsf{let}\, z = \mathsf{ref}(0)\,\mathsf{in}\,(z := M; \mathsf{while}\,(!z)\,\mathsf{do}\,(N; z := M)),$$

  where z is fresh. We can find a canonical form for the latter term by appealing to the inductive hypothesis and previous lemmas.
  By the preceding case, we can obtain $\mathbb{C}_{z := M}$. Using Lemma 47 we find $\mathbb{C}_{N;z := M}$. Note that $\mathsf{while}\,(!z)\,\mathsf{do}\,\mathbb{C}_{N;z := M}$ is then a canonical form. Next we can invoke Lemma 47 again for $\mathbb{C}_{z := M}$ and $\mathsf{while}\,(!z)\,\mathsf{do}\,\mathbb{C}_{N;z := M}$ to obtain $\mathbb{C}$. $\mathsf{let}\, z = \mathsf{ref}(0)\,\mathsf{in}\,\mathbb{C}$ is then a suitable canonical form for $\mathsf{while}\, M\,\mathsf{do}\, N$.
- For $\lambda x^\theta.M$ we can use $\lambda x^\theta.\mathbb{C}_M$.
- Finally, we handle application MN. By inductive hypothesis we obtain $\mathbb{C}_M$ which must take one of the following shapes: $\lambda x^\theta.C$, $\mathsf{case}(x)[C_0, \cdots, C_{max}]$ or $\mathsf{let}\cdots\mathsf{in}\, C$. Using the equivalences below (and Lemma 47) we can then reason by induction on the possible structure of $\mathbb{C}_M$.

$$\begin{aligned}(\lambda x^\theta.C)N &\cong \mathsf{let}\, x = \mathbb{C}_N\,\mathsf{in}\, C\\ (\mathsf{case}(x)[C_0, \cdots, C_{max}])N &\cong \mathsf{case}(x)[C_0N, \cdots, C_{max}N]\\ (\mathsf{let}\cdots\mathsf{in}\, C)N &\cong \mathsf{let}\cdots\mathsf{in}\, CN\end{aligned}$$

□

B Reduction via spans

This section shows how to reduce generalised automata to ordinary ones. We shall achieve this via bisimulation equivalence, using a notion of correspondence between different name environments called *span*.

Let $\mathcal{A} = \langle Q, q_0, \rho_0, \delta, F\rangle$ be a $(n_{\mathsf{r}}, n_1, n_2)$-automaton and set $n' = n_1 + n_2 - n_{\mathsf{r}}$. We proceed to construct an (n_{r}, n')-automaton $\mathcal{A}'$ such that $\mathcal{A} \sim \mathcal{A}'$. The automaton will simulate $\mathcal{A}$ and in particular it will encode the pair of register assignments of the latter into a single one. For this, it will use an auxiliary environment component. Let $N_1 = [n_{\mathsf{r}} + 1, n_1]$, $N_2 = [n_{\mathsf{r}} + 1, n_2]$ and $N' = [n_{\mathsf{r}} + 1, n']$. We call

$$(X_1, R, X_2) \in \mathcal{P}(N_1) \times \mathcal{P}(N_1 \times N_2 \times \{1, 2\}) \times \mathcal{P}(N_2)$$

a *span* on N_1, N_2 if:

- $(i, j, z), (i', j', z') \in R$ implies that $i = i' \iff j = j'$, and $i = i' \implies z = z'$,
- $\mathsf{dom}(R) = \{i \mid \exists j, z.\,(i, j, z) \in R\} \subseteq X_1$ and $\mathsf{cod}(R) = \{j \mid \exists i, z.\,(i, j, z) \in R\} \subseteq X_2$.

We write $N_1 \rightleftharpoons N_2$ for the set of spans on N_1, N_2. By abuse of notation, we write R for the whole of (X_1, R, X_2), in which case we also use the notation $\mathsf{xdom}(R) = X_1$ and $\mathsf{xcod}(R) = X_2$. We also define a map $\overline{R} : [1, n'] \to [1, n']$:

$$\begin{aligned}\overline{R} = &\{(i, i), (j^+, i) \mid (i, j, 1) \in R\} \cup \{(i, i) \mid i \in (N_1 \backslash \mathsf{dom}(R)) \cup [1, n_{\mathsf{r}}]\}\\ &\cup \{(i, j^+), (j^+, j^+) \mid (i, j, 2) \in R\} \cup \{(j^+, j^+) \mid j \in N_2 \backslash \mathsf{cod}(R)\}\end{aligned}$$

The notation extends to other domains containing indices, e.g. for labels: $\overline{R}(\mathrm{R}_i) = \mathrm{R}_{\overline{R}(i)}$ and $\overline{R}(\ell) = \ell$ if ℓ is not a register.

The role of a span is to allow us to simulate a pair of register assignments in a single one. (X_1, R, X_2) represents a pair of assignments where the first one has domain X_1 and the second one X_2. Moreover, R relates the common names: e.g. if $(i, j, z) \in R$ then register i in the first assignment contains the same name, say a, as register j in the second one. Finally, the index z specifies in which part of the simulating assignment does a really occur. For example, the pair of register assignments (suppose $n_{\mathsf{r}} = 0$ for simplicity, and $n_1 = n_2 = 3$),

$$\rho_1 = \{(2, b), (3, a)\}, \quad \rho_2 = \{(1, d), (2, a), (3, e)\}$$

can be represented by the single assignment

$$\rho = \{(2, b), (3, a), (4, d), (6, e)\}$$

with related span $R = (\{2, 3\}, \{(3, 2, 1)\}, \{1, 2, 3\})$. This is because ρ_1 has domain $\{2, 3\}$, ρ_2 has domain $\{1, 2, 3\}$, and the only common name is $a = \rho_1(3) = \rho_2(2)$, and is stored in the first half of ρ. The map $\overline{R} = \{(1, 1), (2, 2), (3, 3), (4, 4), (5, 3), (6, 6)\}$ then allows us to reconstruct ρ_1, ρ_2 by:

$$\langle \rho_1, \rho_2 \rangle = \{(2, b), (3, a), (4, d), (5, a), (6, e)\} = \rho \circ \overline{R}$$

In effect, $\overline{R}$ maps each index of the extended register $\langle \rho_1, \rho_2 \rangle$ to its "real" position in ρ.

If ρ_1, ρ_2 are register assignments of size n_1, n_2 respectively and with common $[1, n_{\mathsf{r}}]$-part then we can obtain the span which accommodates all common names to the left by:

$$\rho_1 \leftrightarrow \rho_2 = (\mathsf{dom}(\rho_1) \cap N_1, \{(i, j, 1) \mid i \in N_1, j \in N_2, \rho_1(i) = \rho_2(j)\}, \mathsf{dom}(\rho_2) \cap N_2)$$

The single register assignment which simulates ρ_1 and ρ_2 via $\rho_1 \leftrightarrow \rho_2$ is then given by $\rho = \langle \rho_1, \rho_2 \rangle \circ \{(i, i) \mid \overline{(\rho_1 \leftrightarrow \rho_2)}(i) = i\}$.

If $R : N_1 \rightleftharpoons N_2$ and $\pi_1, \pi_2 \in \mathsf{Mix}$ then we define the update of R with respect to the two (componentwise) rearrangements π_1 and π_2:

$$\begin{aligned} R[\pi_1, \pi_2] &= (\{i \mid \pi_1(i) \in \mathsf{xdom}(R)\}, R', \{j \mid \pi_2(j) \in \mathsf{xcod}(R)\}) \\ R' &= \{(i, j, z) \mid (\pi_1(i), \pi_2(j), z) \in R\} \end{aligned}$$

Moreover, we extend the shift notation componentwise to other constructions, e.g. for symbolic stores: $S^+ = \{(i^+, \mathrm{R}_{j^+}) \mid (i, \mathrm{R}_j) \in S\} \cup \{(i^+, j) \mid (i, j) \in S\}$. Finally, if $\pi, \pi' \in \mathsf{Mix}$ then we let $\pi[\pi'] = \pi' \cup \{(i, \pi(i)) \mid i \notin \mathsf{dom}(\pi')\}$.

For $\mathcal{A}$ given as above, we define an (n_{r}, n')-automaton $\mathcal{A}' = \langle Q', q_0', \rho_0', \delta', F' \rangle$ by:

$$\begin{aligned} &Q' = Q \times (N_1 \rightleftharpoons N_2) \times \mathsf{SSto} \\ &q_0' = (q_0, R_0, \emptyset) && R_0 = \pi_1(\rho_0) \leftrightarrow \pi_2(\rho_0) \\ &\rho_0' = \rho_0 \circ \{(i, i) \mid R_0(i) = i\} && F' = \{(q, R, S) \in Q' \mid q \in F\} \end{aligned}$$

The crux of the construction is the definition of δ'. The automaton will simulate the generalised automaton using one copy of each common name. R will be used as a record of common/private names between the two register components of $\mathcal{A}$, and as a specifier of which part of the registers do the common names appear in. Moreover, the automaton will operate with a stack belonging to $\mathsf{Sta}' = (\mathbb{C}^{\checkmark}_{\mathsf{stack}} \times (N_1 \rightleftharpoons N_2) \times \mathsf{Reg})^*$. In order to simulate $\mathcal{A}$'s matching conditions for stores, $\mathcal{A}'$ will use, apart from R, a symbolic store S (which will simulate to the store Σ appearing in configurations of $\mathcal{A}$).

For each $(q, R, S) \in Q'$ we include in δ' precisely the following transitions.

If $q \xrightarrow{\pi_1,\pi_2} q'$ then $(q, R, S) \xrightarrow{(\pi_1 \cup \pi_2^+)[\pi]} (q', R', S')$, where $\pi = \{(j^+, i) \mid (i, \pi_2(j), 1) \in R, i \notin \mathsf{cod}(\pi_1)\} \cup \{(i, j^+) \mid (\pi_1(i), j, 2) \in R, j \notin \mathsf{cod}(\pi_2)\}$, $R' = R[\pi_1, \pi_2]$ and $S' = S \circ \overline{\pi}$.[10] If $q \xrightarrow{\pi_1,\checkmark} q'$ then do the same, with $\pi_2 = \mathsf{id}$ (the identity injection). Dually if $q \xrightarrow{\checkmark,\pi_2} q'$.

If $q \in Q_P$ and $q \xrightarrow{\nu X_1.(\ell_1,t_1,\phi_1)^{S_1},\nu X_2.(\ell_2,t_2,\phi_2)^{S_2}} q'$ then if $t_1 = t_2$ and for some $R' : N_1 \rightleftharpoons N_2$:[11]

– $R' = (\mathsf{xdom}(R) \uplus X_1, R \uplus R'', \mathsf{xcod}(R) \uplus X_2)$ where $\mathsf{dom}(R'') = X_1, \mathsf{cod}(R'') = X_2$ and for all $(i, j, z) \in R''$ we have $z = 1$,
– $\overline{R'}(\ell_1) = \overline{R'}(\ell_2^+)$ and, for all $(i, j, z) \in R'$, $\overline{R'}(S_1(i)) = \overline{R'}(S_2(j)^+)$,
– for all $i \in \mathsf{dom}(S_1) \backslash \mathsf{dom}(R')$, $S_1(i) = S(i)$,
– for all $j \in \mathsf{dom}(S_2) \backslash \mathsf{dom}(R')$, $S_2(j) = S(j)$,

then $(q, R, S) \xrightarrow{\nu X_1.(\ell,t_1,\phi)^{S'}} (q', R', S')$, where:

– $\ell = \overline{R'}(\ell_1)$ and $S' = \overline{R'}(S_1) \cup \overline{R'}(S_2^+)$,
– if $t_1 \in \mathbb{T}_{\mathsf{noop}}$ then $\phi = ()$, and if $t_1 \in \mathbb{T}_{\mathsf{push}}$ and, say, $\phi_i = (s_i, \pi_i)$ then $\phi = ((s_1, s_2, R'[\pi_1, \pi_2]), (\pi_1 \cup \pi_2^+)[\pi])$ with $\pi = \{(i, j^+) \mid (\pi_1(i), j, 2) \in R', j \notin \mathsf{cod}(\pi_2)\} \cup \{(j^+, i) \mid (i, \pi_2(j), 1) \in R', i \notin \mathsf{cod}(\pi_1)\}$;

otherwise, if $q \in Q_N$ then switch to divergence mode, i.e. $(q, R, S) \xrightarrow{\mathsf{id}} (\mathsf{div}(q), R, S)$.

If $q \in Q_P$ and $q \xrightarrow{\nu X_1.(\ell_1,t_1,\phi_1)^{S_1},\checkmark} q'$ then $(q, R, S) \xrightarrow{\nu X_1.(\ell,t_1,\phi)^{S'}} (q', S', R')$, where:

– $\ell = \overline{R'}(\ell_1)$ and $S' = \overline{R'}(S_1)$,
– if $t_1 \in \mathbb{T}_{\mathsf{noop}}$ then $\phi = ()$, and if $t_1 \in \mathbb{T}_{\mathsf{push}}$ and, say, $\phi_1 = (s_1, \pi_1)$ then $\phi = ((s_1, \checkmark, R'[\pi_1, \emptyset]), \pi_1[\pi])$ with $\pi = \{(i, j^+) \mid (\pi_1(i), j, 2) \in R'\}$.

Dually if $q \xrightarrow{\checkmark,\nu X_2.(\ell_2,t_2,\phi_2)^{S_2}} q'$.

If $q \in Q_O$ and $q \xrightarrow{\nu X_1.(\ell_1,t,\phi_1)^{S_1},\nu X_2.(\ell_2,t,\phi_2)^{S_2}} q'$ then for all $R', R_{\mathsf{p}}, \hat{R} : N_1 \rightleftharpoons N_2$ and $\pi \in \mathsf{Mix}$ such that:

– if $t \in \mathbb{T}_{\mathsf{noop}}$ then $\hat{R} = R$, and if $t \in \mathbb{T}_{\mathsf{pop}}$ and, say, $\phi_i = (s_i, \pi_i)$ then, assuming that the popped symbol will be $(s_1, s_2, R_{\mathsf{p}})$:

  – R, π_1, π_2 and R_{p} should be consistent, that is:[12]
    - for each $(i, j, z) \in R$, if $\pi_1(i), \pi_2(j)$ are both defined or $\pi_1(i) \in \mathsf{dom}(R_{\mathsf{p}})$ or $\pi_2(j) \in \mathsf{cod}(R_{\mathsf{p}})$ then $(\pi_1(i), \pi_2(j), z') \in R_{\mathsf{p}}$, for some z',
    - for each $(i, j, z) \in R_{\mathsf{p}}$ if $\pi_1^{-1}(i), \pi_2^{-1}(j)$ are both defined or $\pi_1^{-1}(i) \in \mathsf{dom}(R)$ or $\pi_2^{-1}(j) \in \mathsf{cod}(R)$ then $(\pi_1^{-1}(i), \pi_2^{-1}(j), z') \in R$, for some z';

[10] Note here that rearrangements π_1 and π_2 kept intact except for the case when a name which is supposed to be shared from the two parts of the current assignment is deleted in the part that actually contains it but is retained in the other one. In such a case, the name is not deleted but copied to the other part of the assignment (this is what π achieves).

[11] Here we match all freshly created names and store them in the first component of the assignment. Moreover, we match labels and symbolic stores (on related indices) according to the new R'. For private indices we stipulate that values remain unchanged (this corresponds to $\Sigma[\Sigma_2] \cup \Sigma[\Sigma_2]$ being consistent in the definition of generalised automata).

[12] i.e. if two registers in the stack are related (by R_{p}) and each is related to a register in the machine registers (by π_1, π_2) then the latter must also be related (by R); and viceversa. This condition ensures that we can correctly compute the span R_{p} pertaining to the register assignment after popping.

- the new pop injection π is of the form $\pi = \pi_{A1} \cup \pi_{A2} \cup \pi_{B1} \cup \pi_{B2}$ where:[13]
  - $\pi_{A1} = \{(\overline{R}(i), \overline{R_{\mathsf{p}}}(\pi_1(i))) \mid i \in \mathsf{dom}(\pi_1)\}$,
  - $\pi_{A2} = \{(\overline{R}(j^+), \overline{R_{\mathsf{p}}}(\pi_2(j)^+)) \mid j \in \mathsf{dom}(\pi_2)\}$,
  - $\pi_{B1} \subseteq (\mathsf{xdom}(R) \backslash \mathsf{dom}(R)) \times (\mathsf{xcod}(R_{\mathsf{p}}) \backslash \mathsf{cod}(R_{\mathsf{p}}))^+$,
  - $\pi_{B2} \subseteq (\mathsf{xcod}(R) \backslash \mathsf{cod}(R))^+ \times (\mathsf{xdom}(R_{\mathsf{p}}) \backslash \mathsf{dom}(R_{\mathsf{p}}))$;
- the span $\hat{R}$ obtained after popping is then computed by $\mathsf{xdom}(\hat{R}) = \mathsf{xdom}(R) \cup (\mathsf{xdom}(R_{\mathsf{p}}) \backslash \mathsf{cod}(\pi))$, $\mathsf{xcod}(\hat{R}) = \mathsf{xcod}(R) \cup (\mathsf{xcod}(R_{\mathsf{p}}) \backslash \{j \mid j^+ \in \mathsf{cod}(\pi)\})$ and $\hat{R} = R \cup R_{A1} \cup R_{A2} \cup R_{A3} \cup R_B$ where:
  - $R_{A1} = \{(i, j, z) \in R_{\mathsf{p}} \mid i \notin \mathsf{cod}(\pi_1), j \notin \mathsf{cod}(\pi_2)\}$,
  - $R_{A2} = \{(\pi_1^{-1}(i), j, 1) \mid (i, j, z) \in R_{\mathsf{p}}, i \in \mathsf{cod}(\pi_1), j \notin \mathsf{cod}(\pi_2)\}$,
  - $R_{A3} = \{(i, \pi_2^{-1}(j), 2) \mid (i, j, z) \in R_{\mathsf{p}}, i \notin \mathsf{cod}(\pi_1), j \in \mathsf{cod}(\pi_2)\}$,
  - $R_B = \{(i, j, 1) \mid (i, j^+) \in \pi_{B1}\} \cup \{(i, j, 2) \mid (j^+, i) \in \pi_{B2}\}$;

– the span R' obtained after the transition satisfies the conditions:

- $\mathsf{xdom}(R') = \mathsf{xdom}(\hat{R}) \uplus X_1$ and $\mathsf{xcod}(R') = \mathsf{xcod}(\hat{R}) \uplus X_2$,
- $\hat{R} \subseteq R'$ and $R' \backslash \hat{R} = R_C \cup R_{C1} \cup R_{C2}$ where:[14]
  - $R_C \subseteq X_1 \times X_2 \times \{1\}$,
  - $R_{C1} \subseteq (\mathsf{xdom}(\hat{R}) \backslash \mathsf{dom}(\hat{R})) \times X_2 \times \{1\}$,
  - $R_{C2} \subseteq X_1 \times (\mathsf{xcod}(\hat{R}) \backslash \mathsf{cod}(\hat{R})) \times \{2\}$,
- $\overline{R'}(\ell_1) = \overline{R'}(\ell_2)$ and, for all $(i, j, z) \in R'$, if $S_1(i)$, $S_2(j)$ are both defined then $\overline{R'}(S_1(i)) = \overline{R'}(S_2(j)^+)$;

include $(q, R, S) \xrightarrow{\nu X.(\ell,t,\phi)^{S'}} (q', R', S')$, where:

– $X = (X_1 \backslash \mathsf{dom}(R_{C2})) \cup (X_2 \backslash (\mathsf{cod}(R_C) \cup \mathsf{cod}(R_{C1})))^+$,
– $\ell = \overline{R'}(\ell_1)$ and $S' = \overline{R'}(S_1) \cup \overline{R'}(S_2^+)$,
– if $t \in \mathbb{T}_{\mathsf{noop}}$ then $\phi = ()$, and if $t \in \mathbb{T}_{\mathsf{pop}}$ then $\phi = ((s_1, s_2, R_{\mathsf{p}}), \pi)$.

If $q \in Q_O$ and $q \xrightarrow{\nu X_1.(\ell_1,t,\phi_1)^{S_1},\checkmark} q'$ then for all s_2 and all $R', R_{\mathsf{p}}, \hat{R} : N_1 \rightleftharpoons N_2$ and $\pi \in \mathsf{Mix}$ such that:[15]

– if $t \in \mathbb{T}_{\mathsf{noop}}$ then $\hat{R} = R$, and if $t \in \mathbb{T}_{\mathsf{pop}}$ and, say, $\phi_1 = (s_1, \pi_1)$ then, assuming that the popped symbol will be $(s_1, s_2, R_{\mathsf{p}})$:

- R, π_1, π_2 and R_{p} should be consistent, that is, there is no $(i, j, z) \in R$ such that $\pi_1(i) \in \mathsf{dom}(R_{\mathsf{p}})$;
- the new pop injection π is of the form $\pi = \pi_{A1} \cup \pi_{B1} \cup \pi_{B2}$ where:
  - $\pi_{A1} = \{(\overline{R}(i), \overline{R_{\mathsf{p}}}(\pi_1(i))) \mid i \in \mathsf{dom}(\pi_1)\}$,
  - $\pi_{B1} \subseteq (\mathsf{xdom}(R) \backslash \mathsf{dom}(R)) \times (\mathsf{xcod}(R_{\mathsf{p}}) \backslash \mathsf{cod}(R_{\mathsf{p}}))^+$,
  - $\pi_{B2} \subseteq (\mathsf{xcod}(R) \backslash \mathsf{cod}(R))^+ \times (\mathsf{xdom}(R_{\mathsf{p}}) \backslash \mathsf{dom}(R_{\mathsf{p}}))$;

[13] Here we take into account the fact that, when popping, new matchings may occur between private registers in the ith component of private machine registers and the $\bar{i}$th component of the stack registers (cf. π_{B1} and π_{B2}).

[14] Since X_1 and X_2 are filled with componentwise locally fresh names, we allow for new matchings between private registers in the ith component and indices in $X_{\bar{i}}$ (cf. R_{C1} and R_{C2}). Intuitively, when the ith component creates a locally fresh name, it may very well be that the created name is in fact one of the private names of the $\bar{i}$th component.

[15] Note here that, although the pop happens in the first component, it may be the case that the stack top contains a non-$\checkmark$ symbol in its second component (e.g. in the product construction this can happen because the automaton is allowed to move to divergence mode while the stack is non-empty), in which case we need to range over all possible such symbols.

  - the span $\hat{R}$ obtained after popping is given by $\mathsf{xdom}(\hat{R}) = \mathsf{xdom}(R) \cup (\mathsf{xdom}(R_\mathsf{p})\setminus\mathsf{cod}(\pi))$, $\mathsf{xcod}(\hat{R}) = \mathsf{xcod}(R) \cup (\mathsf{xcod}(R_\mathsf{p})\setminus\{j \mid j^+ \in \mathsf{cod}(\pi)\})$ and $\hat{R} = R \cup R_{A1} \cup R_{A2} \cup R_B$ where:
    - $R_{A1} = \{(i, j, z) \in R_\mathsf{p} \mid i \notin \mathsf{cod}(\pi_1)\}$,
    - $R_{A2} = \{(\pi_1^{-1}(i), j, 1) \mid (i, j, z) \in R_\mathsf{p}, i \in \mathsf{cod}(\pi_1)\}$,
    - $R_B = \{(i, j, 1) \mid (i, j^+) \in \pi_{B1}\} \cup \{(i, j, 2) \mid (j^+, i) \in \pi_{B2}\}$;
- the span R' obtained after the transition satisfies the conditions:[16]
  - $\mathsf{xdom}(R') = \mathsf{xdom}(\hat{R}) \uplus X_1$ and $\mathsf{xcod}(R') = \mathsf{xcod}(\hat{R})$,
  - $\hat{R} \subseteq R'$ and $R'\setminus\hat{R} \subseteq X_1 \times (\mathsf{xcod}(\hat{R})\setminus\mathsf{cod}(\hat{R})) \times \{2\}$;

include $(q, R, S) \xrightarrow{\nu X.(\ell,t,\phi)^{S'}} (q', R', S')$, where:

- $X = X_1\setminus\mathsf{dom}(R'\setminus\hat{R})$, $\ell = \overline{R'}(\ell_1)$ and $S' = S[\overline{R'}(S_1)]$,
- if $t \in \mathbb{T}_\mathsf{noop}$ then $\phi = ()$, and if $t \in \mathbb{T}_\mathsf{pop}$ then $\phi = ((s_1, s_2, R_\mathsf{p}), \pi)$.

Dually if $q \xrightarrow{\checkmark,\nu X_2.(\ell_2,t,\phi_2)^{S_2}} q'$.

Let us remark that reachable nodes $((q, R, S), \rho, \sigma, H)$ in the configuration graph of $\mathcal{A}'$, apart from the initial one, satisfy $\mathsf{dom}(S) = \mathsf{dom}(\rho) = \overline{R}(\mathsf{xdom}(R) \cup \mathsf{xcod}(R))$ while all elements (s_1, s_2, R, ρ) of σ also satisfy $\mathsf{dom}(\rho) = \overline{R}(\mathsf{xdom}(R) \cup \mathsf{xcod}(R))$. We arrive at the following.

Lemma 48 *For $\mathcal{A}$ and $\mathcal{A}'$ as above, $\mathcal{A} \sim \mathcal{A}'$.*

Proof Consider the relation $\mathcal{R}$ between configurations of $\mathcal{A}$ and $\mathcal{A}'$ defined by:

$$\begin{aligned}\mathcal{R} = \{&((q, \rho, \sigma, H, \Sigma), ((q, R, S), \rho', \sigma', H)) | \rho = \rho' \circ \overline{R},\ \Sigma =' \mathsf{Sto}(\rho', S),\ \sigma = \mathsf{ev}(\sigma'),\\ &(q, \rho, \sigma, H, \Sigma), ((q, R, S), \rho', \sigma', H) \text{ reachable}\}\end{aligned}$$

where ev is given by $\mathsf{ev}(\epsilon) = \epsilon$ and $\mathsf{ev}((s_1, s_2, R, \rho) :: \sigma) = (s_1, s_2, \rho \circ \overline{R}) :: \mathsf{ev}(\sigma)$, and by $\Sigma =' \mathsf{Sto}(\rho', S)$ we mean $\Sigma = S = \emptyset$ or $\Sigma = \mathsf{Sto}(\rho', S)$. By case analysis we can show that $\mathcal{R}$ is a bisimulation. Moreover, $((q_0, \rho_0, \epsilon, \emptyset, \emptyset), ((q_0, R_0, \emptyset), \rho_0', \epsilon, \emptyset)) \in \mathcal{R}$, and therefore $\mathcal{A} \sim \mathcal{A}'$. □

In moving from $\mathcal{A}$ to $\mathcal{A}'$, each transition of the former may yield several transitions of the latter, in the case of O- to P-transitions. From these transitions, though, those that have the same label (up to $\sim_\alpha$) are allowed to range solely because of the choice of the final R'. In case $\mathcal{A}$ is strongly deterministic, these choices are unique up to $\sim_\alpha$: we make a canonical choice to allocate all common fresh names on the first component of the assignment, while frugality ensures that all fresh names are reachable from those already available, which are uniquely specified from the given R and $\hat{R}$. Thus, if $\mathcal{A}$ is strongly deterministic then so is $\mathcal{A}'$.

References

1. Abramsky S, McCusker G (1997) Call-by-value games. In: Proceedings of CSL (Lecture notes in computer science), vol 1414. Springer, pp 1–17

[16] As in the previous case, new matchings may occur in this step because of a locally fresh name created in an index from X_1 being matched with a private name of the second component of the current assignment. Similar matchings may also occur above when popping.

2. Ahmed A, Dreyer D, Rossberg A (2009) State-dependent representation independence. In: Proceedings of POPL. ACM, pp 340–353
3. Alur R, Madhusudan P (2004) Visibly pushdown languages. In: Proceedings of STOC'04, pp 202–211
4. Barthe G, Crespo JM, Kunz C (2011) Relational verification using product programs. In: International symposium on formal methods. Springer, pp 200–214
5. Benton N, Leperchey B (2005) Relational reasoning in a nominal semantics for storage. In: Proceedings of TLCA (Lecture notes in computer science), vol 3461. Springer, pp 86–101
6. Chaki S, Gurfinkel A, Strichman O (2012) Regression verification for multi-threaded programs. In: Proceedings of VMCAI. Springer, pp 119–135
7. Cheng EYC, Kaminski M (1998) Context-free languages over infinite alphabets. Acta Inf. 35(3):245–267
8. Cotton-Barratt C, Hopkins D, Murawski AS, Ong CHL (2015) Fragments of ML decidable by nested data class memory automata. In: Proceedings of FOSSACS'15. Springer, pp 249–263
9. Dreyer D, Neis G, Birkedal L (2010) The impact of higher-order state and control effects on local relational reasoning. In: Proceedings of ICFP. ACM, pp 143–156
10. Fedyukovich G, Sery O, Sharygina N (2013) evolcheck: incremental upgrade checker for c. In: Proceedings of TACAS. Springer, pp 292–307
11. Gabbay MJ, Pitts AM (2002) A new approach to abstract syntax with variable binding. Formal Asp Comput 13:341–363
12. Godlin B, Strichman O (2009) Regression verification. In: Proceedings of the 46th annual design automation conference (DAC). ACM, pp 466–471
13. Honda K, Yoshida N (1999) Game-theoretic analysis of call-by-value computation. Theor Comput Sci 221(1–2):393–456
14. Hopkins D, Murawski AS, Ong CHL (2011) A fragment of ML decidable by visibly pushdown automata. In: Proceedings of ICALP. Springer, pp 149–161
15. Jeffrey A, Rathke J (1999) Towards a theory of bisimulation for local names. In: Proceedings of LICS, pp 56–66
16. Koutavas V, Wand M (2006) Small bisimulations for reasoning about higher-order imperative programs. In: Proceedings of POPL. ACM, pp 141–152
17. Kozen D (1997) Automata and computability. Springer, Berlin
18. Lahiri SK, Hawblitzel C, Kawaguchi M, Rebêlo H (2012) Symdiff: a language-agnostic semantic diff tool for imperative programs. In: Proceedings of CAV. Springer, pp 712–717
19. Laird J (2008) A game semantics of names and pointers. Ann Pure Appl Log 151:151–169
20. Murawski AS, Ramsay SJ, Tzevelekos N (2015) A contextual equivalence checker for IMJ*. In: Proceedings of ATVA (LNCS). Springer, pp 234–240. Tool available at: http://bitbucket.org/sjr/coneqct/wiki/Home
21. Murawski AS, Ramsay SJ, Tzevelekos N (2015) Game semantic analysis of equivalence in IMJ. In: Proceedings of ATVA (LNCS). Springer, pp 411–428
22. Murawski AS, Tzevelekos N (2009) Full abstraction for Reduced ML. In: Proceedings of FOSSACS (Lecture notes in computer science), vol 5504. Springer, pp 32–47
23. Murawski AS, Tzevelekos N (2011) Algorithmic nominal game semantics. In: Proceedings of ESOP (Lecture notes in computer science), vol 6602. Springer, pp 419–438
24. Murawski AS, Tzevelekos N (2011) Game semantics for good general references. In: Proceedings of LICS. IEEE computer society press, pp 75–84
25. Murawski AS, Tzevelekos N (2012) Algorithmic games for full ground references. In: Proceedings of ICALP (Lecture notes in computer science), vol 7392. Springer, pp 312–324
26. Murawski AS, Tzevelekos N (2013) Deconstructing general references via game semantics. In: Proceedings of FOSSACS (Lecture notes in computer science), vol 7794. Springer, pp 241–256
27. Murawski AS (2005) Functions with local state: regularity and undecidability. Theor Comput Sci 338(1/3):315–349
28. Murawski AS, Tzevelekos N (2016) Nominal game semantics. Found Trends Progr Lang 2(4):191–269
29. Pitts AM, Stark IDB (1998) Operational reasoning for functions with local state. In: Gordon AD, Pitts AM (eds) Higher-order operational techniques in semantics. CUP, Cambridge, pp 227–273
30. Reddy US, Yang H (2004) Correctness of data representations involving heap data structures. Sci Comput Program 50(1–3):129–160
31. Sangiorgi D, Kobayashi N, Sumii E (2011) Environmental bisimulations for higher-order languages. ACM Trans Program Lang Syst 33(1):5
32. Tzevelekos N (2009) Full abstraction for nominal general references. LMCS 5(3):1–69