Abstract

What is a basic automata-theoretic model of computation with names and fresh-name generation? We introduce Fresh-Register Automata (FRA), a new class of automata which operate on an infinite alphabet of names and use a finite number of registers to store fresh names, and to compare incoming names with previously stored ones. These finite machines extend Kaminski and Francez’s Finite-Memory Automata by being able to recognise globally fresh inputs, that is, names fresh in the whole current run. We examine the expressivity of FRA’s both from the aspect of accepted languages and of bisimulation equivalence. We establish primary properties and connections between automata of this kind, and answer key decidability questions. As a demonstrating example, we express the theory of the pi-calculus in FRA’s and characterise bisimulation equivalence by an appropriate, and decisive in the finitary case, notion in these automata.

Categories and Subject Descriptors F.1.1 [Computation by Abstract Devices]: Models of Computation; D.3.1 [Programming Languages]: Formal Definitions and Theory—Semantics

General Terms Theory, Languages, Verification

1. Introduction

One of the most common and useful abstractions in programming is the assumption that entities of specific kinds can be created at will and, moreover, in such a manner that newly created entities are always fresh — distinct from any other such created thus far. This is, for example, the case with mutable reference cells, exceptions user-declared datatypes, etc. in languages like Standard ML [15]. Following a long tradition in computer science [20], we call these entities names and specify them as follows.

Names can be created fresh dynamically and locally, compared for equality and communicated between agents or subroutines. Apart from the uses mentioned above, names form the basis of calculi of mobile processes (e.g. the π-calculus [14]); appear in network protocols and secure transactions; and are general essential in programming for identifying variables, channels, threads, objects, codes, and many other sorts of name in disguise. To our knowledge, there has not been in the literature a proposal of a basic automata-theoretic model of names, providing abstract machines underlying all these paradigms. We propose just such a model here.

Our model is based on the successful paradigm of Finite-Memory Automata (FMA), introduced by Kaminski and Francez in the early 90’s [11]. Motivated by real-world problems (where codes, addresses, identifiers, etc. may have unbounded domains), those automata address a demand for a “natural” finite-state machine model over infinite alphabets. An FMA A is an automaton attached with a finite number of name-storing registers. Its structure looks identical to that of an ordinary finite-state automaton over a finite set of labels generated by indices in the range 1, . . . , n, where n is the number of registers. However, A truly operates on the infinite set of inputs H (the set of names), with indices i referring to the names stored in the i-th register of A. This simple idea lifts the automaton from finite to infinite alphabet.

There are two ways in which an FMA can access its registers: either by comparing an input name to a stored one, or by storing an input name in one of its registers but only in case it is locally fresh, that is, it does not already appear in any of them. Thus, FMA’s are history-free: their computational steps rely solely on their current registers. Here we introduce Fresh-Register Automata (FRA), a finite-register automaton model which extends FMA’s by global freshness recognition: an automaton can now accept (and store) an input name just in case it is fresh in the whole run. For example, a transition

\[ q \xrightarrow{a} q' \]

means that if A is at state q and the set of names that have appeared in its registers so far is H, then A can accept any name \( a \notin H \), store it in its i-th register and proceed to q’. This history-sensitive feature precisely captures fresh-name creation.1 Thus, e.g. the following language (not recognised by FMA’s [11]) is recognised by a single-state FRA with one register.

\[ L_1 = \{ a_1 \cdots a_k \in H^* \mid \forall i \neq j, a_i \neq a_j \} \]

An intuitive way to view \( L_1 \) is as the trace of a fresh-name generator; one which returns reference cells in SML, objects in Java, memory addresses in C, etc.

Research in FMA’s and their formal languages has been extensive [2, 6, 11, 21, 25, 27]. It has been shown [11, 21] that FMA-recognisable languages are closed under union, intersection, concatenation and Kleene star; they are not closed under complement; emptiness of FMA’s is decidable; and universality is undecidable. Our first contribution is to answer this series of questions for FRA’s. We show that for emptiness and universality the situation remains the same as in FMA’s. On the other hand, FRA-recognisable languages are still closed under union and intersection, but history-sensitivity prohibits this for concatenation and Kleene star. Moreover, they are not closed under complement and, in fact, there is an FMA-recognisable language whose complement is not recognised by FRA’s.

\[ 1 \text{Note that, although history-sensitive, the automaton does not have full access to the history } H. \text{ In automata-theoretic jargon, the situation can be described as consulting an oracle who can decide the freshness of names.} \]

Nikos Tzevelekos
Oxford University Computing Laboratory
nikt@comlab.ox.ac.uk
Our main vehicle for studying equivalence between FRA’s is bisimulation equivalence (also called bisimilarity). The notion is very relevant from the point of view of programming, and process calculi in particular, and in the case of FRA’s it implies language equivalence. More importantly, we show that by examining FRA’s at the symbolic level, i.e. as ordinary finite-state automata on the set of index-generated labels, it is possible to capture bisimilarity by an appropriate symbolic notion: we thus prove that FRA-bisimilarity is decidable. A symbolic bisimulation relates states of two automata in specific environments, the latter specifying how the names which appear in their registers related.

As a demonstrating example, we express the \( \pi \)-calculus in the context of fresh-register automata. We introduce the \( \pi \)-calculus system: a presentation of the \( \pi \)-calculus with early transition semantics [14, 26], in which processes are states of an infinite FRA. Transitions are given by FRA-transitions and the system is finitely branching. More specifically, bound outputs are modelled by globally fresh transitions, while each input is decomposed into finitely many cases: either the incoming name is locally fresh or it already appears in the registers. This clean treatment of fresh and bound names is the main advantage of the \( \pi \)-calculus and allows for the finite representation, as ordinary FRA’s, of finitary processes. Moreover, we characterise strong bisimilarity by an appropriate symbolic notion in \( \pi \). This gives an alternative proof of decidability of bisimilarity for finitary processes.

Motivation and related work

Programming languages  The idea of studying names in higher-order languages and in isolation of other effects was first pursued by Pitts and Stark [24]. They introduced the \( \nu \)-calculus, an extension of the simply-typed \( \lambda \)-calculus with references of unit type. Investigations on the \( \nu \)-calculus were meticulously carried on by Stark in his PhD thesis [28], which exposed a rather unexpected complexity hidden behind names. It became evident that better models for languages with names were needed. To address this, new directions in denotational [1, 12, 13, 18] and operational [3, 10] models were explored, significantly advancing our understanding of computation with names but, at the same time, leaving basic questions unanswered. In particular, those works examined computation at the higher level, that of programs and program equivalence, leaving open the question of a basic, lower-level model.

Interestingly, in their initial paper on FMA’s [11], Kaminski and Francez motivate their construction (also) by briefly presenting an idealised procedural language with names. There, names cannot be freshly created, but they can be read from the environment as inputs and stored in a finite memory. Moreover, stored names can flow inside the memory from one register to another and can also be compared for equality and thus trigger goto’s. The authors explain that FMA’s operate like acceptors for that simple imperative language with names. By analogy, FRA’s describe the extension of the language with fresh-name generation.

Process calculi  For mobile systems like the \( \pi \)-calculus [14], where processes can create locally, receive or send names, the use of ordinary labelled transition systems for its semantics is in many ways unsatisfactory: for example, infinite branching arises even in the case of very simple processes that receive a (locally fresh) name, or output a locally created (globally fresh) one. Such shortcomings naturally led to solutions involving representations of processes by formalisms which incorporate name-reasoning of some sort [4, 5, 16]. The most notable paradigm in this direction is that of History-Dependent Automata (HD-Automata) [16, 22], which are structures defined in a universe of named sets and named functions. HD-automata can succinctly represent the \( \pi \)-calculus, as HD-transitions match ‘on-the-fly’ names between the source, target, and label of \( \pi \)-calculus transitions, allowing thus for the use of representatives of processes and transitions, rather than all possible ones under e.g. permutation of fresh names. The stream of research on HD-automata has focussed both on foundational issues [17, 22] and on pragmatic applications [7]. The work presented here shares objectives with HD-automata, and to some extent can be viewed as a complementary attempt to the same question, albeit based on basic machines of ‘first principles’.

Outline

In the next section we give the basic definitions on FRA’s. Section 3 provides some useful bisimilar constructions. In Section 4 we recall FMA’s and establish their connection to FRA’s. We examine WFRA’s, a weaker notion of FRA’s focussing on global freshness, in Section 5. In Section 6 we prove some technical results regarding closure properties for FRA’s, and in Section 7 we show that emptiness and bisimilarity are decidable using symbolic methods. Section 8 examines the \( \pi \)-calculus in the setting of FRA’s.

2. Definitions

We distinguish between two sets of input symbols:

- an infinite set of names, \( \mathcal{N} \), and
- a finite set of constants, \( \mathcal{C} \).

Constants have an auxiliary role and are non-storable.\(^3\) We let \( a, b, \) etc. range over names. We write \( \mathcal{A}^{*} \) for the set of finite strings of names, and \( \mathcal{A}^{\infty} \) for its restriction to those containing pairwise distinct names. Strings \( a_{1}, \ldots, a_{n} \) will be typically represented by vectors \( \vec{a} \), in which case \( \text{img}(\vec{a}) = \{ a_{1}, \ldots, a_{n} \} \).

For each \( n \in \omega \), we write \([n]\) for the set \( \{1, \ldots, n\} \), and let

\[ L_{n} = \mathcal{C} \cup \{ i, i^{*}, i^{\infty} | i \in [n] \} . \]

be the set of labels generated by \([n]\). Moreover, we define

\[ \text{Reg}_{n} = \{ \sigma : [n] \rightarrow \mathcal{A} \cup \{ \sharp \} | \forall i \neq j. \sigma(i) = \sigma(j) \quad \rightarrow \quad (\sigma(i) = \sharp) \} \]

to be the set of register assignments of size \( n \). We write \( \text{img}(\sigma) \) for the name-range of \( \sigma \), i.e. \( \text{img}(\sigma) = \{ a \in \mathcal{N} | \exists i. \sigma(i) = a \} \), and let \( \text{dom}(\sigma) = \{ i \in [n] | \sigma(i) \in \mathcal{C} \} \).

Whenever \( a \notin \text{img}(\sigma) \),

\[ \sigma(i) \mapsto a = \{ \langle i, a \rangle \} \cup \{ \langle j, \sigma(j) \rangle | \ j \in [n] \ \ \& \ \ i \} \]

is an update of \( \sigma \), for any \( i \in [n] \).

Definition 1. A fresh-register automaton (FRA) of \( n \) registers is a quintuple \( A = (Q, q_{0}, \sigma_{0}, \delta, F) \) where:

- \( Q \) is a finite set of states,
- \( q_{0} \) is the initial state,
- \( \sigma_{0} \in \text{Reg}_{n} \) is the initial register assignment,
- \( \delta \subseteq Q \times L_{n} \times Q \) is the transition relation,
- \( F \subseteq Q \) is the set of final states.

\( A \) is called a register automaton (RA) if there are no \( q, q^{*}, q^{\prime} \) such that \( (q, i^{0}, q^{\prime}) \in \delta \).

Transitions containing labels of the form \( i \) are called known transitions; those of the form \( i^{*} \) are locally fresh ones; and globally fresh transitions involve \( i^{\infty} \). Thus, an RA is an FRA with no globally fresh transitions.\(^4\)

Here is an informal reading of \( \delta \). Suppose \( A \) is at state \( q_{1} \) with current register assignment \( \sigma \). If input \( i \in \mathcal{C} \cup \mathcal{A} \) arrives then:\(^5\)

\[ \delta \subseteq Q \times L_{n} \times Q \]

is the transition relation, where:

\[ \{ i, a \} \]

is a set of updates, for any \( i \in [n] \).

\( A \) is a register automaton (RA) if there are no \( q, q^{*}, q^{\prime} \) such that \( (q, i^{0}, q^{\prime}) \in \delta \).

Transitions containing labels of the form \( i \) are called known transitions; those of the form \( i^{*} \) are locally fresh ones; and globally fresh transitions involve \( i^{\infty} \). Thus, an RA is an FRA with no globally fresh transitions.

\(^{3}\) In other presentations [11, 21] there is no such distinction, but symbols that appear in the initial register assignment can play the role of constants.

\(^{4}\) This yields the same notion of register automaton as that of [21].

\(^{5}\) Note that the same symbol, \( i \), is later used to range over elements of \( L_{n} \).

\(^{2}\) A process is finitary if its it does not grow unboundedly in parallelism.
The above is formally defined by means of configurations representing the intended current state of the automaton, which apart from states contains information on the current register assignment and the set of names having appeared thus far (the history). The latter component is necessary for globally fresh transitions.

**Definition 2.** A configuration of $A$ is a triple $(q, \sigma, H) \in \hat{Q}$, with $\hat{Q} = Q \times \mathbb{R}_{\omega} \times \mathbb{N}(A)$ and $\mathbb{N}(A)$ being the set of finite subsets of $A$. From $\delta$ define a transition relation on configurations $\rightarrow_s$ as follows. For all $(q, \sigma, H) \in \hat{Q}$ and $(q, \ell, q') \in \delta$:

- If $\ell \in \mathbb{C}$ then $(q, \sigma, H) \overset{\ell}{\rightarrow} (q', \sigma, H)$.
- If $\ell = i$ and $\sigma(i) = a$ then $(q, \sigma, H) \overset{\ell}{\rightarrow} (q', \sigma, H \cup \{a\})$.
- If $\ell = \hat{s}$ and $a \notin \text{img}(\sigma)$ then $(q, \sigma, H) \overset{\ell}{\rightarrow} (q', \sigma', H')$ with $\sigma' = \sigma \cup \{a\}$ and $H' = H \cup \{a\}$.
- If $\ell = \hat{b}$ and $a \notin \text{img}(\sigma)$ then $(q, \sigma, H) \overset{\ell}{\rightarrow} (q', \sigma', H')$ with $\sigma' = \sigma \cup \{a\}$ and $H' = H \cup \{a\}$.

We write $\rightarrow^r$ for the reflexive transitive closure of $\rightarrow_s$.

We say that configuration $q$ is reachable if $(q_0, \sigma_0, \emptyset) \overset{\ell}{\rightarrow}^r q$ for some $\ell \in \mathbb{A}$.

### Example 4. The reader can check that the language $L_1$ is recognized by the following DFA:

$$A_0 = \{(q_0), q_0, \{(1, \sharp)\}, \{q_0, 1^\omega, q_0\}, \{q_0\}\}$$

Note that the FRA $B = \{(q_0), q_0, \{(1, \sharp)\}, \{q_0, 1^\omega, q_0\}, \{q_0\}\}$ recognizes the language:

$$L_2 = \{a_1 \cdots a_k \in A^* \mid k \in \omega \text{ and } \forall i, a_i \neq a_{i+1}\}$$

This is equivalent to $A$. A more elaborate example is the following. Let $A$ be the FRA:

![Diagram of FRA](image)

with initial assignment $\{(1, \sharp)\}$. The automaton works as follows. It receives a name $a$ and then keeps receiving $a$ until some $b \neq a$ arrives; then it keeps receiving $b$ until a globally fresh $c$ arrives; it then repeats from start. Thus, members of $L(A)$ are of the form

$$a_{0^n} b_{c_n} a_{1^n} b_{c_1} a_{2^n} b_{c_2} \cdots a_{m^n} b_{c_m}$$

where, for all $i$, we have $j_i, k_i > 0$, $a_i \neq b_i$ and $c_i$ differs from all symbols preceding it. Formally, setting

$$L'(H) = \{a_1 b c \mid \lor_n \forall a \neq b \land c \notin H \cup \{a, b\}\}$$

we have that $L(A) = \bigcup_{i} L_i$, where we set $L_0 = L'(\emptyset)$ and $L_{i+1} = \{a b \mid \exists \sigma \in L_i \land b \in L'(\text{img}(\sigma))\}$.

### Some basic results

The languages of FMA's [11] are regular once constrained to a finite number of symbols. Moreover, the language accepted by an FMA is impervious to name-mentioned that do not affect its initial register. These properties carry over to FRA's, and are proved as in [11].

**Proposition 5.** Let $A = \langle Q, q_0, \sigma_0, \delta, F \rangle$ be an FRA of $n$ registers and $S \subseteq \mathbb{K}$ be finite. Then, $L(A) \cap S^*$ is a regular language.

**Proposition 6.** For $A$ as above, if $\bar{a} \in L(A)$ and $\pi : A \to \bar{a}$ is such that $\pi(a) = a$ for all $a \in \text{img}(\sigma_0)$ then $\pi(\bar{a}) \in L(A)$.

**Bisimulation** Bisimulation equivalence turns out to be a great tool for relating automata, even from different paradigms. It implies language equivalence and, in all our cases of interest, it is not too strict in this aspect. We choose it here as our main vehicle of study.

**Definition 7.** Let $A_i = \langle Q_i, q_{0i}, \sigma_{0i}, \delta_i, F_i \rangle$ be FRA's with $n_i$ registers, for $i = 1, 2$. A relation $R \subseteq \hat{Q}_1 \times \hat{Q}_2$ is called a simulation on $A_1$ and $A_2$ if, for all $(q_1, q_2) \in R$:

- if $\pi_1(q_1) \in F_1$ then $\pi_1(q_2) \in F_2$,
- if $q_1 \overset{\ell}{\rightarrow}_{\delta_1} q'_1$ then $q_2 \overset{\ell}{\rightarrow}_{\delta_2} q'_2$ for some $(q'_1, q'_2) \in R$.

$R$ is called a bisimulation if both $R$ and $R^{-1}$ are simulations. We say that $A_1$ and $A_2$ are bisimilar, written $A_1 \sim A_2$, if there is a bisimulation $R$ such that $\langle (q_{01}, 0, \emptyset) \rangle, \langle (q_{02}, \sigma_0, 0) \rangle \in R$.

**Lemma 8.** If $A_1 \sim A_2$ then $L(A_1) = L(A_2)$.

The above is proved using standard methods. Bisimilarity is also called bisimulation equivalence. For instance, the automaton $A_0$ of example 4 is bisimilar to $B = \{(q_0, q_0), \{(1, \sharp)\}, \{(q_0, 1^\omega, q_0), \{(q_0, 1^\omega, q_0)\}\}, \{(q_0, q_0)\}, \{(q_0, q_0)\}\}$, with a bisimulation witnessing this being the following:

$$\{(q_0, \sigma_0, 0) \} \cup \{(q_0, 1^\omega, H_1), (q_0, \sigma_0, H_2) \mid H_1 = H_2\}$$

where $\sigma_0 = \{(1, \sharp)\}$.

### Bisimilar constructions

In this section we demonstrate some bisimilar constructions which will be useful in the sequel. Starting from a fresh-register automaton $A = \langle Q, q_0, \sigma_0, \delta, F \rangle$ of $n$ registers, we effectively construct the following bisimilar automata.

- The closed FRA $A$ called the closure of $A$.
- For any $\bar{a} \in A^*$, the FRA $A \upharpoonright \bar{a}$.

This is called the extension of $A$ by $\bar{a}$, and its initial assignment is $\sigma_0 + \bar{a} = \sigma_0 \cup \{(i + n, a_i) \mid 1 \leq i \leq |\bar{a}|\}$.

Our presentation will focus on constructing the bisimilar automata and explaining the candidate bisimulation relation $R$, omitting the actual proof that $R$ is a bisimulation, as these proofs are not difficult (but tedious) and follow directly from the constructions.
Closures For \( A \) as above with \( n \) registers we define its closure to be the \( n \)-register FRA \( A = (Q', q_0', \sigma_0', \delta', F') \) given as follows. We set \( Q' = Q \times P(n) \), \( q_0' = (q_0, \text{dom}(\sigma_0)) \), \( \sigma_0' = \sigma_0 \) and \( F' = \{ \langle q, S \rangle \mid q \in F \} \). Recall we want to construct an automaton which is closed, that is, whenever a configuration with state \( q \) and assignment \( \sigma \) is reached and \( \langle q, i, q' \rangle \) is a transition, then \( \sigma(i) \in A \) and therefore the transition is allowed. The extra component added in \( Q' \) monitors the registers that have been assigned a name (note that once a register has been assigned a name it cannot return to the \( \emptyset \) state). Consequently, \( \delta' \) will be designed in such a way so that this monitoring carries through and, moreover, the known transitions included in \( \delta' \) are always allowed:

\[
\delta' = \{ \langle (q, S), \ell, (q', S') \rangle \mid (q, \ell, q') \in \delta \land \ell \in \mathbb{C} \} \\
\cup \{ \langle (q, S), i, (q', S') \rangle \mid (q, i, q') \in \delta \land i \in \mathbb{C} \} \\
\cup \{ \langle (q, S), \ell, \delta(i)^{\prime}, (q', S') \rangle \mid (q, \ell, q') \in \delta \land S = S' \} 
\]

Now, we can check that the following relation is a bisimulation

\[
R = \{ \langle (q, \sigma, H), (q, S), \sigma, H) \rangle \mid \text{dom}(\sigma) = S \}
\]

and therefore that \( A \sim A' \). Moreover, the reachable configurations of \( A \) are of the form \( \langle (q, S), \sigma, H) \rangle \) with \( \text{dom}(\sigma) = S \), and therefore the automaton is closed.

Remark 9. If \( A = (Q, q_0, \sigma_0, \delta, F) \) is a closed FRA then each path \( q_0, q_1, \ldots, q_m, q_n \) in \( A \) (where arrow notation \( \sigma \) denotes a path configuration

\[
(q_0, \sigma_0, \emptyset) \xrightarrow{\ell_1} \xrightarrow{\ell_2} \ldots \xrightarrow{\ell_m} (q_m, \sigma_m, H_m)
\]

given as follows. We set

\[
Q' = Q \times \{ 0, 1, \ldots, n \} \\
q_0' = (q_0, \ell_1) \\
F' = \{ (q, f, S) \mid q \in Q \} \\
\sigma_0' = \sigma_0 + \ell
\]

Finally:

\[
\delta' = \{ \langle (q, f, S), f(\ell) \rangle \mid \ell \in \mathbb{C} \} \\
\cup \{ \langle (q, S), j, (q', f', S') \rangle \mid (q, j, q') \in \delta \land f' \notin \text{img}(f) \} \\
\cup \{ \langle (q, S), j, \delta(i)^{\prime}, (q', f', S') \rangle \mid (q, j, q') \in \delta \land j \notin S \}
\]

where \( f(i)^{\prime} = f(i)^{\prime}, f(i)^{\prime} = f(i)^{\prime}, f(\ell) = \ell \) for all \( \ell \in \mathbb{C}, f' = f[\ell \mapsto \emptyset] \), and \( S = S' \setminus \{ j \} \).

The transition relation in \( A \sim A' \) proceeds as in \( A \) with the exception of locally/global fresh transitions, where some extra care is needed. Since the registers of the new automaton contain more names than those of the initial one, fresh transitions in \( A \sim A' \) can now capture fewer names. For example, if \( a \) is one of the added names then a fresh transition from the initial configuration could capture it before, but this is no more the case as \( a \) appears in \( \sigma_0' \); instead, we need an explicit \( j \) transition for this purpose. This is what the second clause of the definition \( \delta' \) addresses. For this to work we need to introduce the component \( f \) to keep track of the correspondences between old and new registers that arise in the way just described. For globally fresh transitions a similar situation arises, only that this time we need only remember which of the names in the initial \( \delta' \) have not appeared in the history thus far, which is what the component \( S \) achieves. Thus, the following is a bisimulation

\[
R = \{ \langle (q, \sigma, H), (q, f, S, \sigma', H) \rangle \mid \sigma = \sigma' \cap \text{img}(\ell) \subset H \cap \sigma'(S) \}
\]

and therefore \( A \sim A' \).}

4. Finite-memory automata

We now present FMA’s and examine their properties in relation to FRA’s and RAs. In fact, RA’s are equivalent to FMA’s and in the literature they have been used as synonyms (e.g. compare [11] with [21]). The precise correspondence is stated in proposition 11, which is a folklore result.

Let us recall the original definition from [11]. A finite-memory automaton (FMA) of \( n \) registers is a sextuple \( A = (Q, q_0, \sigma_0, \rho, \delta, F) \) where:

- \( Q \) is a finite set of states, with \( q_0 \in Q \) initial, and \( F \subseteq Q \) final.
- \( \sigma_0 \in \text{Reg}_n \) is the initial register assignment.
- \( \rho : Q \rightarrow [n] \) is the reassignment (partial) function.
- \( \delta \subseteq Q \times [n] \times Q \) is the transition relation.

The intuitive reading of \( \delta \) is the following. Suppose \( A \) is state \( q_1 \) with register assignment \( \sigma \) and let \( (q_1, i, q_2) \in \delta \). If input \( a \in \mathbb{A} \) arrives then:

- If \( \sigma(i) = a \) then \( A \) accepts and moves to state \( q_2 \).
- If \( a \notin \text{img}(\sigma) \) and \( \rho(q_1) = i \) then \( A \) accepts \( a \), it sets \( \sigma(i) = a \) and moves to state \( q_2 \).

Formally, a configuration is now a pair \( (q, \sigma) \in Q \), where

\[
\hat{Q} = Q \times \text{Reg}_n, \\
\text{and the transition relation } \rightarrow \subseteq \hat{Q} \times A \times \hat{Q} \text{ is defined as follows. For all } (q, \sigma) \in \hat{Q} \text{ and } (q, i, q') \in \delta:
\]

- If \( \sigma(i) = a \) then \( (q, \sigma) \xrightarrow{a} (q', \sigma) \).
- If \( \rho(q_1) = i \) then, for all \( a \notin \text{img}(\sigma) \), \( (q, \sigma) \xrightarrow{a} (q', \sigma[i \mapsto a]) \).

The notions of reachable configurations and accepted strings and languages are defined just as in the case of FRA’s.

Example 10. Recall the language \( L_2 \) of example 4:

\[
L_2 = \{ a_1 \cdots a_n \in \mathbb{A}^* \mid \forall i. a_i \neq a_{i+1} \}
\]

which is RA-recognisable. \( L_2 \) is recognised by the FMA:

\[
B = (Q, q_0, \sigma_0, \{ (q_0, 1), (q_1, 2) \}, \{ (q_1, 1), (q_2, 2, q_0) \}, Q)
\]

where \( Q = \{ q_0, q_1 \} \) and \( \sigma_0 = \{ (1, \emptyset), (2, \hat{\emptyset}) \} \). Comparing this to \( B \) of example 4, the reader can observe how the differences between RA’s and FMA’s in reassignment have been addressed here by use of the extra register.

The main properties of FMA’s and FMA-recognisable languages have been established as follows.

(a). Emptiness is decidable for FMA’s [11] (i.e. is \( L(A) = \emptyset \) ?), and in particular it is NP-complete [25].

(b). The languages accepted by FMA’s are closed under union, intersection, concatenation and Kleene star; they are not closed under complement [11].

(c). Universality is undecidable [21] (i.e. is \( L(A) = \mathbb{A}^* \) ?). Hence, the equivalence and containment problems are undecidable too (i.e. is \( L(A) \subseteq L(B) ? \)).

We shall see that the emptiness problem is also decidable for FRA’s (proposition 24). Clearly, FRA’s being extensions of FMA’s implies that universality of the former is undecidable, and hence the same holds for equivalence and containment. In section 6 we will examine closure properties of FRA’s and show that closure under concatenation and Kleene star are lost, closure under complement still fails, but closure under union and intersection prevail.

We now relate FMA’s to the kind of automata we have introduced previously: in essence, FMA’s are the same as RA’s. The
of states; we match each transition (proposition 17).

Corollary 12. The universality, equivalence and containment problems are undecidable for RA’s and FRA’s.

5. Weak fresh-register automata

In this section we examine a weaker version of FRA’s by concentrating on the aspect of global freshness while relaxing that of local freshness. Even though this restriction leads us to machines that do not extend FMA’s, we show that universality remains undecidable (proposition 17).

The machines we introduce operate on sets of labels
\[ L_n^m = C \cup \{ i, i^? , i^@ \mid i \in [n] \} , \]
where \( i^? \) stands for “accept any name” transitions. Moreover, their registers are now taken from the sets \( \text{Reg}_n^m = [n] \rightarrow A \cup \{ \emptyset \} \).

Definition 13. A weak fresh-register automaton (WFRA) of \( n \) registers is a quintuple \( A = (Q, q_0, \sigma_0, \delta, F) \) where:

- \( Q \) is a finite set of states, with \( q_0 \in Q \) initial, and \( F \subseteq Q \) final.
- \( \sigma_0 \in \text{Reg}_n^m \) is the initial register assignment.
- \( \delta \subseteq Q \times L_n^m \times Q \) is the transition relation.

The transition relation has the same intuitive meaning as in the case of FRA’s, with the exception that in transitions of the form \((q_1, i^?, q_2)\in \delta\) the automaton accepts any name \( a \) stores it at its \( i \)-th field and moves to state \( q_2 \). Formally, a configuration is now given as a triple \((q, \sigma, H)\in Q\) where
\[ Q = Q \times ([n] \rightarrow (A \cup \{ \emptyset \})) \times \text{Reg}_n^m , \]
and the transition relation \( \rightarrow \subseteq \bar{Q} \times (C \cup A) \times Q \) on configurations is defined as follows. For all \((q, \sigma, H)\in Q\) and \((q, \ell, q')\in \delta\):

- if \( \ell \in C \) then \((q, \sigma, H) \rightarrow a_{a} (q', \sigma, H)\);
- if \( \ell = i \) and \( \sigma(i) = a \) then \((q, \sigma, H) \rightarrow a_{\sigma} (q', \sigma', H')\);
- if \( \ell = i^? \) then \((q, \sigma, H) \rightarrow a_{\sigma} (q', \sigma', H')\);
- if \( \ell = i^@ \) and \( a \notin H \cup \text{img}(\sigma_0) \) then \((q, \sigma, H) \rightarrow a_{\sigma} (q', \sigma', H')\);
with \( \sigma' = \sigma[i \mapsto a] \) and \( H' = H \cup \{ a \} \).

Reachable configurations and accepted strings/languages are defined exactly as in FRA’s.

Example 14. Consider the following language,
\[ L_3 = \{ a_1 \cdots a_k b_1 \cdots b_l \in A^* \mid \forall i \neq j, a_i \neq a_j \land b_i \neq b_j \} \]
which is in fact the concatenation of \( A^* \) with itself, and the WFRA:

with 2 registers, both of them initially empty. Call the above \( A \). We claim that \( L(A) = A^* \cup L_3 \), that is, \( s \in L(A) \iff s \notin L_3 \) for all \( s \in A^* \).

The forward implication is clear: if \( s \in L(A) \) then the same name \( a \) appears three times in \( s \) (via the path \( q_0 q_1 q_2 q_3 q_4 q_5 \)), or names \( a_1 \) and \( a_2 \) appear each twice in \( s \) without interleaving (via the path \( q_0 q_1 q_2 q_3 q_4 q_5 \)). In both cases, \( s \notin L_3 \).

For the opposite direction, let \( s \notin L_3 \) and feed it to \( A \). Since \( s \notin A^* \), we can write \( s = s_1 a_1 s_2 a_2 s_3 \) with \( s_1, s_2, s_3 \in A^* \). In \( A \), \( s_1 a_1 s_2 a_2 s_3 \) leads control to \( q_2 \). Now, \( s \notin L_3 \) implies that \( a_1 s_2 \notin A^* \) so there is some \( a_3 \) in \( a_1 s_2 \) such that \( a_1 s' \neq a_1 s_2 a_3 \) and \( a_2 \) appears in \( a_2 s' \). If \( a_2 = a_1 \) then \( s' \) leads \( A \) directly to \( q_4 \). Otherwise, it leads to \( q_4 \) via \( q_3 \).

The reader may want to verify that changing the labels of the loops at \( q_0 \) and \( q_1 \) above to \( 1^* \), and the label from \( q_0 \) to \( q_1 \) to \( 2^* \), leads to a WFRA \( A' \) that still satisfies \( L(A') = A^* \cup L_3 \).

We show that any WFRA has a bisimilar FRA of the same number of registers. The idea is to simulate the non-linear memory (i.e. a set of registers that may contain names in common) of the WFRA by a linear memory plus a reordering function on the FRA part. For example, here is such a simulation:

\[ ((1, a), (2, b), (3, c)) \rightarrow ((1, a), (2, b), (3, c)) \]
\[ \text{plus (} 1 \rightarrow 2, 2 \rightarrow 3, 3 \rightarrow 2 \) \]

The reordering functions will be attached to the states of the FRA. Moreover, we shall simulate any transitions (i.e. of the form \( i^? \)) of the WFRA by means of locally-fresh-transitions \( i^* \) and known-transitions \( j \), for all \( i \) in. In the end, defining the new transition relation gets a bit involved as one has to bear reorderings in mind, which need to be accounted for before making a transition and updated afterwards.

Lemma 15. For any WFRA \( A \) of \( n \) registers there is an effectively constructible FRA \( B \) of \( m \) registers such that \((A \sim B) \).

Proof. Let \( A = (Q, q_0, \sigma_0, \delta, F) \); construct \( B = (Q', q'_0, \sigma'_0, \delta', F') \) as follows. We set \( Q' = Q \times \{ [n] \rightarrow [n] \} \) and write elements of \( Q' \) as \((q, f)\). Simulation of non-linear memory \( \sigma \) by linear memory \( \sigma' \) and reordering \( f \) is defined in the obvious manner: \( \sigma = \sigma' \circ f \).

Moreover, for each \( i \in [n] \), the multiplicity of \( \sigma(i) \), i.e. the number of times it appears in \( \sigma \), is given by the size of \( f^{-1}(\{ f(i) \}) \); we denote this by \( \mu(i) \). We let \( \sigma'_0(q_0, f_0) \) be a simulation of \( \sigma_0 \) such that \( \sigma'_0 \) contains no more names than \( \sigma_0 \) and set \( q'_0 = (q_0, f_0) \) and \( F' = \{ (q, f) \mid q \in Q \} \). We now define \( \delta' \):

\[ \delta' = \{ ((q, f), \ell, (q', f')) \mid (q, \ell, q') \in \delta \land \ell \in C \} \]
\[ \cup \{ ((q, f), f(i), (q', f')) \mid (i, f(i), q') \in \delta \land \mu(i) = 1 \} \]
\[ \cup \{ ((q, f), f(i), (q', f')) \mid (i, f(i), q') \in \delta \land \mu(i) > 1 \land j \notin \text{img}(f) \} \]
\[ \cup \{ ((q, f), f(i), (q', f')) \mid (i, f(i), q') \in \delta \land \mu(i) = 1 \} \]
\[ \cup \{ ((q, f), j^*, (q', f')) \mid (q, j^*, q') \in \delta \land \mu(i) > 1 \land j \notin \text{img}(f) \} \]
\[ \cup \{ ((q, f), j, (q', f')) \mid (i, f(i), q') \in \delta \} \]
where \( j' = f[i \mapsto j] \). The first line is straightforward.

The second line says that receiving the name of the \( i \)-th register in \( A \) is simulated by receiving the \( f(i) \)-th name in \( B \). The same rationale is repeated in the third line, only that now we have to do a memory update and therefore we need to be careful with reorderings. In particular, storing the new name, say \( a \), in the \( f(i) \)-th register should not be allowed when \( \mu(i) > 1 \); if this is the case and we set \( \sigma'(f(i)) = a \) then \( a \) still appears in \( \sigma \) but no longer appears in \( \sigma' \), breaking thus the simulation. Nonetheless, if \( \mu(i) > 1 \) then there must be some \( j \) which is free in \( \sigma' \) (i.e. \( j \notin \text{img}(f) \)) and we can safely store the new name in there, updating the reordering function.
Accordingly, the last three lines of δ′ implement the idea that receiving any name can be matched by receiving either a locally fresh name or one of the stored ones. Thus,

\[ R = \{(q, \sigma, H), ((q, f), \sigma'), (H) ) | \sigma = \sigma' \circ f \} \]

is a bisimulation and therefore \( A \sim B \).

We next show that the absence of locally fresh transitions in WFRA’s renders them incapable of recognising FMA-recognisable languages. Combining this with the previous result we obtain that WFRA’s are indeed strictly weaker than FRA’s.

**Lemma 16.** The language \( L_2 = \{a_1 \cdots a_k | \forall i, a_i \neq a_{i+1} \} \) of examples 4 and 10 is not WFRA-recognisable.

**Proof.** Suppose \( L_2 = L(A) \), for a WFRA \( A \) with \( n \) registers. Then, for any \( s \in \Sigma^m \) of length \( m > 1 \), we have \( ss \in L(A) \).

Let the following be the transition path in \( A \) accepting it,

\[ q_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_m} q_m \]

with the subpath from \( q_0 \) to \( q_m \) accepting the second copy of \( s \). Then, none of the \( \alpha_i \)’s can be of the form \( i^0 \) as their names have appeared before. Moreover, if \( \alpha_i = j^0 \) then \( \alpha_i \) can also accept the preceding symbol, contradicting the fact that \( L(A) = L_2 \). Hence, all \( \alpha_i \)’s are in \( \{n\} \). Choosing \( m > n \) we arrive to a contradiction.

Emptiness is decidable for WFRA’s, by inheritance. More interestingly, the universality problem remains undecidable, and hence the same happens for equivalence and containment.

**Proposition 17.** Universality is undecidable for WFRA’s.

**Proof.** The proof is by reduction from the Post Correspondence Problem, and follows the track of the analogous proof in [21]. In particular, we show that the locally fresh transitions of the RA’s constructed in that proof can be replaced by WFRA-transitions. Unlike [21], here it is necessary to use the set \( C \).

6. Closure properties

In order to establish closure properties of FRA’s, and following the approach on FMA’s in [11], it is useful to introduce a version of FRA’s with *multiple assignment*, that is, automata that can store an input name at several of their registers at one step. In particular, assignments will now be taken from the sets \( \operatorname{Reg}_n^\ast \). The set of labels we shall use is the following:

\[ L_n = C \cup \left( \mathcal{P}(\{n\}) \times \mathcal{P}(\{n\}) \right) \]

Labels of the form \((S, T, \perp)\) are written simply \((S, T)\), and when we write \((S, T, A)\) we assume \( A \neq \perp \). If we want to allow for \( \perp \), we write \((S, T, \perp)\).

**Definition 18.** An *MFA* \( A \) of \( n \) registers is a quintuple \( A = (Q, \sigma_0, \delta, F, \cdot) \) where:

- \( Q \) is a finite set of states, \( \sigma_0 \in Q \) is initial and \( F \subseteq Q \) are final.
- \( \sigma_0 \in \operatorname{Reg}_n^\ast \) is the initial register assignment.
- \( \delta \subseteq Q \times L_n \times Q \) is the transition relation.

The intuitive reading of \( \delta \) is the following. If \( A \) at state \( q \) with register assignment \( \sigma \) and input \( \ell \in C \cup A \) arrives then:

- if \( \ell \in C \) and \((q, \ell, q_2) \in \delta \) then \( A \) accepts \( \ell \) and moves to \( q_2 \).
- if \( \ell \in A \) and \((q, (S, T), q_2) \in \delta \) then \( (\sigma(S) \mapsto \ell)^{-1} = T = T \) and \( \ell \) has not appeared in the history nor does it appear in \( \sigma_0(A) \) then \( A \) accepts \( \ell \), it sets \( \sigma(S) = \{\ell\} \) and moves to state \( q_2 \).

- if \( \ell \in A \) and \((q, (S, T), q_2) \in \delta, (\sigma(S) \mapsto \ell)^{-1} \neq T \) and \( \ell \) has not appeared in the history nor does it appear in \( \sigma_0(A) \) then \( A \) accepts \( \ell \), it sets \( \sigma(S) = \{\ell\} \) and moves to state \( q_2 \).

Thus, labels of the form \((S, T)\) work in the same way as in \( M \)-automata [11], and the main novelty here is the inclusion of \((S, T, A)\): in order for the transition to be allowed, the input name \( a \) must be fresh in the history and in the part of \( \sigma_0 \) specified by \( A \). This addition allows us to model globally fresh transitions and also to combine automata unifying their initial assignments.

Formally, let \( \hat{Q} = Q \times \operatorname{Reg}_n^\ast \times \mathcal{P}(\{n\}) \) be the set of configurations and define \( \langle S, T, A \rangle \subseteq Q \times (C \cup A) \times Q \) as follows.

For all \((q, \sigma, H) \in Q \):

- \( \ell \in A \) and \((q, (S, T), q_2) \in \delta, (\sigma(S) \mapsto \ell)^{-1} = T \) and \( \ell \) has not appeared in the history nor does it appear in \( \sigma_0(A) \) then \( A \) accepts \( \ell \), it sets \( \sigma(S) = \{\ell\} \) and moves to state \( q_2 \).

**Lemma 19.** For any FRA \( A \) of \( n \) registers there is an effectively constructible MFA \( B \) of \( n+1 \) registers such that \( A \sim B \).

The other direction is a bit more elaborate and we achieve it in two steps. Let us say that an MFA \( A \) is pure if, for all transitions \((q, (S, T), a, q') \) of \( A, S = T \) and \( A = \{n\} \).

**Lemma 20.** For any MFA \( A \) of \( n \) registers there is an effectively constructible pure MFA \( B \) of \( 2n \) registers such that \( A \sim B \).

**Lemma 21.** For any pure MFA \( A \) of \( n \) registers there is an effectively constructible FRA \( B \) of \( n \) registers such that \( A \sim B \).

We can now establish the following closure properties. Closure under union and intersection is answered positively, while closure under concatenation, Kleene star or complement fails.

**Proposition 22.** For \( A \) and \( B \), the languages \( L(A) \cup \{L(B) \} \) and \( L(A) \cap \{L(B) \} \) are FRA-recognisable.

**Proof.** Assume MFA’s \( A' = (Q_1, q_0, \sigma_0, \delta_1, F_1) \sim A \) and \( B' = (Q_2, q_0, \sigma_2, \delta_2, F_2) \sim B \) of \( n \) and \( m \) registers respectively. For the union, construct an MFRA \( C = (Q, q_0, \sigma_0, \delta, F) \) of \( n+m \) registers, where

\[ Q = \{q_0\} + Q_1 \cup Q_2 \]

\[ \sigma_0 = \sigma_0 + \sigma_2 \]

\[ F = F_1 \cup F_2 \]

with \( \phi : Q_1 \cup Q_2 \to Q \) mapping \( q_0 \) and \( q_0 \) to \( q_0 \), and being elsewhere the identity. Finally:

\[ \delta = \{(q'', q', \ell) | q'' \in C \cup \{q, q'\} \in \delta_1 \}
\]

\[ \cup \{ (q'', (S \cup [m]^{+}), T \cup [m]^{+}, a), q'') | (q, (S, T, A), a) \in \delta_1 \}
\]

\[ \cup \{ (q'', [n]^{m+n}, [n]^{m+n}, a, q') | (q, (S, T, A), a) \in \delta_2 \}
\]

where \( q'' \in \{q, \phi(q)\} \) and \( S^{m+n} = \{ i+n | i \in S \} \), for each

\[ S \subseteq \omega \] and \( \cup^{m+n} = \{ \} \).

It follows that \( L(C) \subseteq L(A) \cup L(B) \).

For the intersection, construct an MFRA \( C = (Q, q_0, \sigma_0, \delta, F) \) of \( n \) and \( m \) registers where \( Q = Q_1 \times Q_2, \sigma_0 = (q_0, q_0), \sigma_0 = \sigma_0 + \sigma_2, F = F_1 \times F_2 \). And, assuming \( \cup A = \emptyset, A = A_2 \):\n
\[ \delta = \{(q, q', \ell) | q \in C \cup \emptyset \in \emptyset, (\pi(q), (S, T, A_1)) \in \delta_1 \}
\]

\[ \cup \{ (q, (S_1 \cup S_2, T_1, T_2, A_1, A_2, \{l\}^2) | \forall \ell \in [2]. (\pi(q), (S, T, A_1, A_2, \{l\}^2)) \in \delta_1 \}
\]
It follows that $\mathcal{L}(C) = \mathcal{L}(A) \cap \mathcal{L}(B)$. □

**Proposition 23.** There are FRA's $A$ and $B$ such that the language $\mathcal{L}(A) \cup \mathcal{L}(B)$ is not FRA-recognisable. Moreover, there is an FRA $A$ such that the language $\mathcal{L}(A')$ is not FRA-recognisable. Finally, there is an RA $B$ such that the language $\mathcal{L}(B)$ is not FRA-recognisable.

**Proof.** For the first part we show that the language $L' = L_1 \cup L_2$ is not FRA-recognisable, where $L_1 = \mathcal{L}(A)$. Suppose $L'$ were recognised by an FRA $C$ of $n$ registers, as $s \in \mathcal{L}(C)$ with $s$ a string of $m$ distinct names. Letting the following be the transition path in $C$ accepting it,

$$q_0 \xrightarrow{\alpha_1} q_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_m} q_0$$

with the subpath from $q_0$ to $q_0'$, call it $p$, accepting the second copy of $s$. As all the symbols of $s$ have already appeared before, none of the $\alpha_i$'s is of the form $\cdot \otimes \cdot$. Moreover, as all the symbols in $s$ are distinct, there cannot be $i \in [n]$ and $j < j'$ such that $\alpha_i \in \{i, i^*\}$ and $\alpha_{j'} = i$, as $\alpha_j$ would then repeat a name already present in the subpath $p$. Moreover, there cannot be $i, i' \in [n]$ and $j < j' < j''$ such that $\alpha_j \in \{i, i^*\}$, $\alpha_{i'} = i^*$ and $\alpha_{j''} = i^*$. For suppose this were the case, and suppose that all $\alpha$'s between $j$ and $j'$ are not in $\{i, i^*\}$, and that all $\alpha$'s between $j'$ and $j''$ are not in $\{i^*\}$. Then, $s = s_1a_1s_2a_2s_3a_3s_4$ with $a_1, a_2, a_3$ corresponding to $\alpha_j, \alpha_{i'}, \alpha_{j''}$ respectively. But $i^*$ is also allowed to accept $\alpha_1$, hence there is $s_4$ such that $s_1s_2a_1s_2a_2s_3a_3s_4 \in \mathcal{L}(C)$, contradicting $\mathcal{L}(C) = L'$. But now taking $m > n + 1$ we obtain a contradiction.

The second part is shown in a similar manner, taking $A$ the automaton accepting the language

$$\mathcal{L}_{A,a_0} = \{ a_0a_1 \cdots a_k \in \mathcal{L} \mid k \neq j, a_i \neq a_j \}$$

for some chosen $a_0$. A similar argument to the above applies, that is, we assume $\mathcal{L}_{A,a_0} = \mathcal{L}(C)$ for some FRA $C$ and select $a_0a_1 \cdots a_n \in \mathcal{L}_{A,a_0}$ of size big enough to yield a contradiction. Finally, it suffices to show $\mathcal{L}(C) = \mathcal{L}(A) \cap \mathcal{L}(B)$ for an RA $B$.

Example 14 we have that $L' = \mathcal{L}(A)$ for a WERA $A$ with no fresh transitions. From that, we obtain $B$ by applying lemma 15. □

### 7. Symbolic methods

The automata we have introduced can be viewed in two different manners: either as ordinary finite-state automata operating on constant symbols and the symbols $1, 1^*, 1^{\otimes 0}, \ldots, 1^{\otimes n}$ (for machines with $n$ registers), or as machines which recognise languages from an alphabet comprising a finite set of constants and an infinite set of names. We use the term **semantic level** for the latter interpretation, and **symbolic level** for the former one. The semantic is of course the intended interpretation but, on the other hand, viewing our automata as operating on the finite alphabet $L_n$ is much more convenient. In this section we examine methods from the symbolic level which characterise semantic notions. More specifically, we start by giving a simple proof of decidability of FRA-emptiness by reducing the problem to FSA-emptiness. Then we proceed to our main point of focus, which is the definition of an appropriate notion of symbolic bisimilarity that is equivalent to the notion of bisimilarity we have been using thus far. As a corollary we prove that bisimilarity is decidable for FRA's.

**Proposition 24.** The emptiness problem is decidable for FRA's.

**Proof.** Given an FRA $A$ of $n$ registers, construct its closure $\overline{A}$, and take $A'$ to be the ordinary FSA with the same set of states, initial state, transition relation and final states as $\overline{A}$, and operating on the set of labels $L_n$. We claim that $\mathcal{L}(\overline{A}) = \emptyset \iff \mathcal{L}(A') = \emptyset$.

Indeed, if $A'$ accepts a string $s \in \mathcal{L} \cup A'$ then, the accepting path in $A'$ yields a string $s' \in L_n^*$ and $s' \in \mathcal{L}(A')$. Conversely, if $A'$ accepts a string $s'$ then the accepting path in $A'$ is also a path in $A'$ ending in an accepting state. From remark 9, we have that the latter yields a string $s \in \mathcal{L}(A')$.

In order to define a symbolic notion of bisimulation equivalence which captures its semantical analogue, we introduce auxiliary structures which record the way in which two register assignments are related. In particular, they record the domains of the assignments and those indices on which the two assignments coincide. A symbolic bisimulation between two automata relates states of the automata in specific record environments. At each bisimulation step the records are updated according to the specific symbolic transitions taking place. This symbolic description is shown to accurately capture what happens at the semantical level.

We adapt Stark’s notion of span [28]. We call

$$S_1, \rho, S_2 \in \mathcal{P}(n_1) \times \mathcal{P}(n_2) \times \mathcal{P}(n_2)$$

typically a span on $(n_1, n_2)$ if:

- $(i, j), (i', j') \in \rho$ implies $i = i' \iff j = j'$;
- $\|\rho\| \subseteq S_2$, where $\|\rho\| = \{ i \in [n] \mid \exists j, (i, j) \in \rho \}$;
- $\rho_1 \subseteq S_1$, where $\rho_1 = \{ i \in [n] \mid \exists j, (i, j) \in \rho \}$.

We write $[n_1] \equiv [n_2]$ for the set of typed spans on $(n_1, n_2)$. A perhaps more intuitive way to view a typed span $(S_1, \rho, S_2)$ is as a triple of relations,

$$(S_1 \hookleftarrow \rho \Rightarrow S_2)$$

By abuse of notation, we write $\rho$ for the whole of $(S_1, \rho, S_2)$, in which case we also use the notation $S_1(\rho) = S_1$ and $S_2(\rho) = S_2$. If $\rho : [n_1] \equiv [n_2]$ and $(i, j) \in \rho$ then $\rho[i \leftrightarrow j] : [n_1] \equiv [n_2]$ is the typed span:

$$(S_1(\rho)) \cup \{ (i', j') \mid i = i' \land j = j' \} \cup \{ (i, j) \}$$

A typed span $(S_1, \rho, S_2)$ relates register assignments $\sigma_1$ and $\sigma_2$ just in case $\rho$ is a bijection between the parts of $[n_1]$ and $[n_2]$ that have common images under $\sigma_1$ and $\sigma_2$, while $S_1$ keeps track of (the indices of) all names in $\sigma_1$. Formally, $\rho = \sigma_1 \leftrightarrow \sigma_2$ if:

$$\text{dom}(\sigma_1) = S_1(\rho) \cap \text{dom}(\sigma_2) = S_2(\rho) \cap \text{dom}(\sigma_2) \cup \{ (i, j) \mid \sigma_1(i) = \sigma_2(j) \}$$

In this case, $\|\rho\| = |S_1(\rho)| + |S_2(\rho)| - |\text{dom}(\rho)|$ gives the total number of names in $\sigma_1$ and $\sigma_2$.

Suppose, for example, that we have related state $q_1$ of automaton $A_1$ to state $q_2$ of $A_2$ with respect to $\rho$. If $(q_1, i, q_1')$ is a transition in $A_1$ and $i \in \text{dom}(\rho)$ then the name in register $i$ of $A_1$ (in the semantical scenario captured by the symbolic description) resides in register $\rho(i)$ of $A_2$. Consequently, $A_2$ can only simulate the transition by some $(q_2, \rho(i), q_2')$. On the other hand, if $(q_1, i^*, q_1')$ is a transition in $A_1$ then there are several factors to consider:

- Any private name of $A_1$ can be captured by $i^*$. Hence, $A_2$ needs a simulating transition $(q_2, j, q_2')$ for every $j \in S_2(\rho) \cap \text{img}(\rho)$.
- Moreover, $A_2$ needs a transition for all names locally fresh to both $A_1$ and $A_2$. This can be some $(q_2, j^\ast, q_2')$ but, under circumstances, it may also be some $(q_2, j^\ast, q_2')$.
We can therefore resolve the latter case by adding a component which counts the names in the history, up to $n_1 + n_2$. In the following we write $n$ for $n_1 + n_2$, and set $h^+ = [h + 1]^n = (h + 1)$ if $h < n$, and $n$ otherwise.

**Definition 25.** Let $A_i = (Q_i, q_{0i}, \sigma_0, \delta_i, F_i)$ be FRA's of $n_i$ registers, for $i = 1, 2$, such that $\text{img}(\rho_{0i}) = \text{img}(\rho_{02}) = H_0$. A **symbolic simulation** on $A_1$ and $A_2$ is a relation

$$R \subseteq Q_1 \times (\{n\} \cup \{0\}) \times (\{n_1\} \cup \{n_2\}) \times Q_2$$

such that, whenever $(q_1, h, \rho, q_2) \in R$, if $q_1 \in F_1$ then $q_2 \in F_2$ and if $(q_1, \ell, q'_2) \in \delta_1$ then:

1. If $\ell \in \mathbb{C}$ then $(q_2, \ell, q'_2) \in \delta_2$ for some $(q'_1, h, \rho, q_2') \in R$.
2. If $\ell = i$ and $i \in \text{dom}(\rho)$ then $(q_2, \rho(i), q'_2) \in \delta_2$ for some $(q'_1, h, \rho, q_2') \in R$.
3. If $\ell = i$ and $i \in S_1(\rho) \setminus \text{dom}(\rho)$ then $(q_2, j^*, q'_2) \in \delta_2$ for some $(q'_1, h, \rho[i \to j], q_2') \in R$.
4. If $\ell = i^*$ then, for any $j \in S_2(\rho) \setminus \text{img}(\rho)$, $(q_2, j, q'_2) \in \delta_2$ for some $(q'_1, h, \rho[i \to j], q_2') \in R$.
5. If $\ell = i^*$ and $h = n$ or $|\rho| < h$ then $(q_2, j, q'_2) \in \delta_2$ for some $(q'_1, h, \rho[i \to j], q_2') \in R$.
6. If $\ell \in \{i^*, i^*\}$ then $(q_2, j^*, q'_2) \in \delta_2$, or $(q_2, j^*, \delta) \in \delta_2$, for some $(q'_1, h, \rho[i \to j], q_2') \in R$.

Setting $(S_1, \rho, S_2)^{-1} = (S_2, \rho^{-1}, S_1)$, the inverse of $R$ is:

$$R^{-1} = \{(q_2, h, \rho, q_1) \mid (q_1, h, \rho^{-1}, q_2) \in R\}.$$

We say that $R$ is a **symbolic bisimulation** if both $R$ and $R^{-1}$ are symbolic simulations. We say that $A_1$ and $A_2$ are **symbolic bisimilar**, written $A_1 \sim A_2$, if there is a symbolic bisimulation $R$ on $A_1$ and $A_2$ such that $(q_0_1, h_0, \rho_0, q_0_2) \in R$ with $h_0 = [H_0]$ and $\rho_0 = \sigma_0 \sim \sigma_0$.

In the following propositions let us assume the hypotheses of Definition 25. Let us also write $\bar{H}$ for $H \cup H_0$, and $n$ for $n_1 + n_2$.

**Proposition 26.** If $R$ is a symbolic simulation on $A_1$ and $A_2$ then

$$R' = \{(q_1, \sigma_1, H), (q_2, \sigma_2, H) \mid (q_1, h, \rho, q_2) \in R \land \rho = \sigma_1 \sim \sigma_2 \land h = [\bar{H}]^n \land \text{img}(\sigma_i) \subseteq \bar{H}\}$$

is a simulation. Moreover, if $R$ is a symbolic bisimulation then $R'$ is a bisimulation.

**Proposition 27.** If $A_1$ and $A_2$ are closed FRA's and $R$ is a simulation on $A_1$ and $A_2$ then

$$R' = \{(q_1, h, \rho, q_2) \mid (q_1, \sigma_1, H), (q_2, \sigma_2, H) \in R \land \rho = \sigma_1 \sim \sigma_2 \land h = [\bar{H}]^n \land \text{img}(\sigma_i) \subseteq \bar{H}\}$$

is a symbolic simulation. Moreover, if $R$ is a bisimulation then $R'$ is a symbolic bisimulation.

**Corollary 28.** Bisimilarity is decidable for FRA's is directly representable by FRA's. Since transitions are multi-symbol, and our automata can recognise one symbol at a time, they will be decomposed into atomic ones. We add sets of input and output processes which cater for the intermediate stages in these decompositions. For example,

$$\overline{a \circ P} \overset{b}{\rightarrow} P \overset{\rho}{\rightarrow} P'$$

where $b \circ P$ is an output process. Output [resp. input] processes are in the middle of sending [receiving] a name on a chosen channel.

**Definition 29.** The $\pi$-calculus syntax is given by the sets $\Pi$, $\Pi_{\text{out}}$, and $\Pi_{\text{inp}}$, with elements:

$$P, Q ::= 0 | \overline{ab} P \mid a(b).P \mid [a = b]P \mid [a = b]P \mid [a = b]P \mid [a = b]P \mid [a = b]P \mid P + Q | P | Q | p(\overline{a})$$

$P_{\text{out}} ::= b.P | [a = b]P_{\text{out}} | P | P_{\text{out}} | [a = b]P_{\text{out}} | P_{\text{out}}$

$P_{\text{inp}} ::= \overline{b}.P | [a = b]P_{\text{inp}} | P | P_{\text{inp}} | P_{\text{inp}} | P_{\text{inp}} | P_{\text{inp}} | P_{\text{inp}} | P_{\text{inp}} | P_{\text{inp}} | P_{\text{inp}}$
where \( a, b \in \Lambda \) and \( \sigma \in \Lambda^* \). We write \( \Pi \) for \( \Pi \cup \Pi_{out} \cup \Pi_{inp} \), and let \( \tilde{P}, \tilde{Q}, \ldots \) range over its elements, which we equate up to \( \alpha \)-equivalence. Name binding is defined as expected: \( b \) is bound in \( \nu b.P, a(b), P \) and \( b(b) \).

It is handy to introduce here some very basic notions from the theory of nominal sets [8, 23]. We call nominal structure any set which may contain names (i.e. elements of \( \Lambda \)), and we denote by \( \text{Perm}(\Lambda) \) the set of finite permutations on \( \Lambda \) (i.e. bijections \( \pi : \Lambda \rightarrow \Lambda \) such that \( \pi(a) \neq a \) for finitely many \( a \in \Lambda \)). For example, if \( \text{id} = \{(a, a) \mid a \in \Lambda\} \in \text{Perm}(\Lambda) \). We shall define for each set \( X \) of nominal structures of interest a function

\[
\cdot : \text{Perm}(\Lambda) \times X \rightarrow X
\]

such that \( \cdot(\pi : x) = (\pi \circ \pi') \cdot x \) and \( \cdot x = x \), for all \( x \in X \) and \( \pi, \pi' \in \text{Perm}(\Lambda) \). \( X \) will be called a nominal set if all its elements involve finitely many names, that is, for all \( x \in X \) there is a finite set \( \Lambda \subseteq X \) such that \( \pi \cdot x = x \) whenever \( \forall a \in \Lambda. \pi(a) = a \). For example, \( \Pi \in \text{Perm}(\Lambda) \) with action \( \pi \cdot S = \{ \pi(a) \mid a \in S \} \). Also, any set of non-nominal structures is a nominal set with trivial action \( \pi : x = x \). More interestingly, if \( X \) is a nominal set then so is \( X^\pi \) with action \( \pi : x = x \). Also, if \( X \) is a nominal set then so is the set \( \bigcup_{\pi \in \Lambda^*}[\pi] \rightarrow X \) with action \( \pi : f = \{(i, x) : (i, x) \in f\} \). Thus, \( \Pi, \Pi_{out}, \Pi_{inp}, \Pi \) are all nominal sets. For example,

\[
\pi \cdot (a(b), b(c), 0) = (a(b'), b(c'), 0)
\]

where \( a' = \pi(a), b' = \pi(b), c' = \pi(c) \) (note that permutations equally affect and bound free and names). Similarly to \( X^\pi \), we have that \( X \times Y \) is a nominal set whenever \( X \) and \( Y \) are. Note that if \( X \) is a nominal set and \( X' \subseteq X \) is such that \( \pi \cdot x \in X' \), for all \( x \in X' \) and \( \pi \in \text{Perm}(\Lambda) \), then \( X' \) is also a nominal set with the inherited action. Hence, the following set is a nominal set:

\[
\tilde{K} = \{(\sigma, \tilde{P}) \mid \sigma \in \bigcup_{\pi \in \Lambda^*} \text{Reg}_n \land \tilde{P} \in \Pi \land \text{fn}(\tilde{P}) \subseteq \text{img}(\sigma)\}
\]

We write \( K \) for the restriction of \( \tilde{K} \) to elements \( (\sigma, \tilde{P}) \) with \( \tilde{P} \in \Pi \). Finally, from a nominal set \( X \) we can derive its set of \( \alpha \)-orbits:

\[
O(X) = \{O(x) \mid x \in X\} \text{ where } O(x) = \{\pi \cdot x \mid \pi \in \text{Perm}(\Lambda)\}.
\]

Note that each \( O(x) \) is a nominal subset of \( X \).

The technology of the previous paragraph is used for defining the transition system of the extended calculus. In contrast to the ordinary \( \pi \)-calculus, the transition relation we define is finitely branching, and this is achieved by considering processes-in-context and specifying channels by their context indices instead of their names. More specifically, we let \( O(\tilde{K}) \) be the set of processes-in-context. Each such \( O(\tilde{K}) \) is written \( \sigma \rightarrow P \).

Since \( \sigma \rightarrow P = \pi \sigma \rightarrow \pi P \), for any permutation \( \pi \), what matters in \( \sigma \rightarrow P \) is not the specific names occurring in \( \sigma \) or \( P \), but only their index in \( \sigma \). For example,

\[
\{(1, a), (2, c)\} \rightarrow (a(b), b(c), 0) = \{(1, a'), (2, c')\} \rightarrow (a'(b'), b(c'), 0)
\]

and in essence both of these are specified by an expression e.g. like \( \{(1, o), (2, o)\}, (1(b), 0). \) Borrowing notation from FRA’s, we build up on the indices idea and use transition labels of the form \( i^\bullet \) for fresh inputs/outputs.

**Definition 30.** The semantics of the \( \pi \)-calculus is given via a labelled transition system with set of states \( O(\tilde{K}) \) and labels:

\[
\alpha ::= i^\bullet \mid i^\circ \mid \tau \mid i^\bullet j \mid i^\circ j \mid ij^\bullet \mid ij^\circ
\]

where \( i, j \in \omega \). The transition relation is given by the rules in Table 1.

Note that \( \sigma \rightarrow \tilde{P} \overset{\cdot}{\rightarrow} \sigma' \rightarrow \tilde{P}' \) implies \( |\sigma| = |\sigma'| \). Some further remarks on reduction:

- Transitions restricted to \( \Pi \) use only \( \tau \) and double labels, i.e. from \( \{ij, ij^\circ, ij^\bullet, ij\} \mid i, j \in \omega \} \).
- Inputs are decomposed as known inputs (INP2a) and locally fresh ones (INP2b), and are therefore finitely branching. The side-conditions impose that, whenever \( \sigma \rightarrow \Pi_{out} \overset{\cdot}{\rightarrow} \sigma' \rightarrow P \), then \( \sigma' = \sigma[\pi \mapsto a], a \notin \text{img}(\sigma) \) and \( \pi \) is the least index such that \( \sigma(i) \notin \text{fn}(P) \). Similar finiteness and minimisation apply to both outputs (OPEN).
- Note that the \( \text{CLOSE} \) rule involves bound outputs, hence globally fresh transitions on the output side. On the input side, it is then necessary to have a matching locally fresh transition: global freshness implies local freshness.

**Example 31.** For each \( a \in \Lambda \), let \( \sigma_a = \{(1, a)\} \) and \( \Pi_a = \nu b. p(ab) \) with definition \( p(ab) = ab \cdot vc. p(bc) \).

In the \( \pi \)-calculus, \( \Pi_a \) induces an infinitely-branching, infinite-path transition graph:

\[
P_a \overset{\cdot}{\rightarrow} P \overset{\cdot}{\rightarrow} P \overset{\cdot}{\rightarrow} P \overset{\cdot}{\rightarrow} \cdots
\]

In the extended calculus, \( \Pi_a \) induces the following transition graph,

\[
\sigma_a \rightarrow P_a \overset{\cdot}{\rightarrow} \sigma_a \rightarrow \nu b. b. vc. p(bc) \overset{\cdot}{\rightarrow} \sigma_b \rightarrow P_b \overset{\cdot}{\rightarrow} \cdots
\]

which is economic by branching once at each step. In fact, setting \( \Pi_{out} = \nu b. b. vc. p(bc) \), and since \( \sigma_a \rightarrow P_a = \sigma_b \rightarrow P_b \), all \( a, b \in \Lambda \), the graph above contains just two nodes:

\[
\sigma_a \rightarrow P_a \overset{\cdot}{\rightarrow} \sigma_a \rightarrow P_{out}
\]

and using double labels we get simply \( \sigma_a \rightarrow P_a \overset{\cdot}{\rightarrow} \sigma_a \rightarrow P_{out} \).

The way in which the two transition relations are related is given by the following lemma, which verifies the intuitions of Table 1.

**Lemma 32.** Let \( \sigma, \sigma' \) be registers, and \( \alpha, \bar{a} \) be labels of \( \tau \) and \( \pi \) respectively. For all \( P, P' \in \Pi \) with \( \text{fn}(P) \subseteq \text{img}(\sigma) \):

- if \( \sigma \rightarrow P \overset{\cdot}{\rightarrow} \sigma' \rightarrow P' \) then \( P \overset{\cdot}{\rightarrow} P' \),
- if \( P \overset{\cdot}{\rightarrow} P' \) then \( P \overset{\cdot}{\rightarrow} \sigma \overset{\cdot}{\rightarrow} P' \),

where either \( \bar{a} = \alpha = \tau \) and \( \sigma = \sigma' \), or \( \bar{a} = i^\bullet j/i^\circ j \).

**Definition 33.** An \( n \)-simulation is a relation

\[
R \subseteq O(\tilde{K}) \times \{[i] \rightarrow [i]\} \times O(\tilde{K})
\]

where either \( \bar{a} = \alpha = \tau \) and \( \sigma = \sigma' \), or \( \bar{a} = i^\bullet j/i^\circ j \).

Note that this translation typically yields infinite FRA’s—but we shall examine classes of processes where the resulting FRA’s are finite in the end of this section.
and, conversely, if 

\[ \sigma \vdash (b). P \xrightarrow{i} \sigma \vdash P \{a/b\} \]

\[ \sigma' \vdash \alpha. P \xrightarrow{\nu a} \sigma' \vdash \nu a. P' \]

\[ \sigma' \vdash \alpha. P \xrightarrow{i} \sigma' \vdash\]

\Large{\textbf{Table 1.}} The transition relation for the \( xx \)-calculus (symmetric counterparts of SUM, PAR, COMM, CLOSE omitted).
those graphs. As those bisimulations live in a space bounded relatively to the sizes of P₁ and P₂, we can search it exhaustively for such relations.

Equating processes up to structural congruence [14], the above results can be further strengthened to processes with finite degree of parallelism, in a similar manner to [4].

9. Further directions
We have introduced an abstract computational paradigm and established its key properties, laying the ground for further research. The next logical step is to examine concrete applications of FRA’s to the description of computation with names, either in the direction of mobile calculi or that of programming languages, relating this approach to existing higher-level approaches. A first such advance of mobile calculi or that of programming languages, relating this to existing approaches to, e.g., the lambda calculus or the pi-calculus, has been recently accomplished in [19] by constructing a model representing the fully abstract game semantics of the language [18].

A potential criticism towards FRA’s concerns the fact that they lack a notion of polarity inherent in computation with names. As those bisimulations live in a space bounded relatively to the sizes of P₁ and P₂, we can search it exhaustively for such relations.

10. Acknowledgements
Thanks to Andrzej Murawski, Samson Abramsky, Vassilis Koutavas and Ulrich Schöpp for discussions and suggestions. Thanks also to the anonymous reviewers for their comments. Section 7 in particular is now much simpler due to a reviewer’s suggestion.

A. Proofs from section 6
Proof of Lemma 19. Let A = ⟨Q₁, q₀, σ₀, δ, F⟩. The construction of B = ⟨Q′, q₀′, σ₀′, δ′, F′⟩ follows closely [11]. In particular, each transition of A involving a name induces an assignment of that name in the extra register of B. If the transition were a fresh assignment then this would result in the name occurring in B just once after assignment, otherwise it would occur twice. As the actual extra register of B changes during this process we can add an extra component in states to remember it.

We set Q′ = Q × [n + 1] and write elements of Q′ as (q, π). Moreover, q₀′ = (q₀, id), σ₀′ = σ₀[n + 1 ↦ 1] and F′ = {(q, π) | q ∈ F}. Finally:

\[ \delta = \{(q_1, π_1, δ_1, (q_2, π_2) | (q_1, q_2) ∈ Δ \} \]
\[ \cup \{ (q_1, \{π(n+1), π(i), π(n+1)\}, (q_1, π_1, δ_1) \} | (q_1, i, q_2) ∈ Δ \} \]
\[ \cup \{ (q_1, \{π(n+1)\}, (q_1, δ_1') | (q_1, i, q_2, δ_1') ∈ Δ \} \]
\[ \cup \{ (q_1, \{π(n+1), \{π(n+1), \{n\}, \{i, π_1, π_2\} | (q_1, i, q_2) ∈ Δ \} \}

where q₀′ = (q₀, π) and σ′ = δ(q₁ ↦ δ(q₁+1)) o π (we write (k ↦ j) for the permutation that swaps k and j). We can show that the following relation is a bisimulation and therefore that A ∼ B.

R = \{(q, σ, H), (q′, σ′, H′) | \forall i ∈ [n], σ(i) = σ′(π(i)) \}

Proof of Lemma 20. Let A = ⟨Q₁, q₀, σ₀, δ, F⟩ and construct B = ⟨Q′, q₀′, σ₀′, δ′, F′⟩ as follows. The idea is to keep in the extra memory registers of B a copy of the initial configuration σ₀ which is never touched by assignments. Thus, whenever A wants to make a transition with label (S, T, A), B will simulate it by a transition (S, σ₀[n]) and transitions of the form (T, S U Tₐ) where Tₐ U Tₐ = \{n+1, ..., 2n\} a ∈ σ₀[n] \ A and a is not in the history. In order to accomplish this we need to enrich states with information regarding whether the names in img(σ₀) appear in the history. Therefore, we set Q′ = Q × T′(img(σ₀)), q₀′ = (q₀, 0), σ₀′ = σ₀ + σ₀, F′ = {(q, I) | q ∈ F} and:

\[ \delta = \{(q, I), (q, π, I) | (q, I) ∈ C \} \cup \{ (q_1, \{σ\}, q_1, δ_1) \} | (q_1, q_2) ∈ Δ \} \]
\[ \cup \{ (q_1, \{σ\}, (q_1, δ_1) | (q_1, q_2) ∈ Δ \} \]
\[ \cup \{ (q_1, \{σ\}, (q_1, δ_1) | (q_1, q_2) ∈ Δ \} \]
\[ \cup \{ (q_1, \{σ\}, (q_1, δ_1) | (q_1, q_2) ∈ Δ \} \]

where a ∈ img(σ₀), Tₐ = \{n+1, ..., 2n\} a ∈ σ₀[n] \ A and Tₐ = Tₐ. We can check that R = \{(q, σ, H), (q′, σ′, H′) | I = H o img(σ₀) \ A = σ₀ + σ₀ = σ₀ \ A \}

is a bisimulation and therefore that A ∼ B.

Proof of Lemma 21. Let A = ⟨Q₁, q₀, σ₀, δ, F⟩ and construct B = ⟨Q′, q₀′, σ₀′, δ′, F′⟩ by setting Q′ = Q × \{n\} and selecting f₀, σ₀ such that img(σ₀) = img(σ₀) and σ₀ = σ₀ \ o f₀. Moreover, set q₀′ = (q₀, f₀), F′ = {{q, f} | q ∈ F} and:

\[ \delta = \{(q, f), (q, π, f') | (q, f) ∈ C \} \cup \{ (q_1, \{σ\}, q_1, δ_1) \} | (q_1, q_2) ∈ Δ \} \]
\[ \cup \{ (q_1, \{σ\}, (q_1, δ_1) | (q_1, q_2) ∈ Δ \} \]
\[ \cup \{ (q_1, \{σ\}, (q_1, δ_1) | (q_1, q_2) ∈ Δ \} \]
\[ \cup \{ (q_1, \{σ\}, (q_1, δ_1) | (q_1, q_2) ∈ Δ \} \]

where f’ = f[S ↦ i]. Now, the following is a bisimulation.

R = \{(q, σ, H), (q, f, σ′, H′) | σ = σ′ ∘ f \} and hence A ∼ B.
B. Proofs from section 7

Proof of Proposition 26. It will suffice to check only non-constant transitions. So let \((q_1, \sigma_1, H), (q_2, \sigma_2, H) \in R\) due to some \((q_1, h, \rho, q_2) \in R\) and suppose that \((q_1, \sigma_1, H) \rightarrow_{\sigma_1} (q_1, \sigma_1', H')\) with \(H' = H \cup \{a\}\). We do case analysis on \(a\). Below we write \(\rho[p_i \mapsto j]\).

- \(a \in \text{img}(\sigma_1) \cap \text{img}(\sigma_2)\), say \(a = \sigma_1(i) = \sigma_2(j)\). Then, it is necessary that \((q_1, i, q_1') \in \delta_1\) and \(\sigma_1' = \sigma_1\). Also, \(\rho = \sigma_1 \mapsto R\) implies \((i, j) \in \rho\), so \((q_1, h, \rho, q_2) \in R\). Thus, \((q_2, \sigma_2, H) \rightarrow_{\sigma_2} (q_2, \sigma_2, H')\) and, noting that \(H' = H\) so \(h = |H'|^{\ast}\), we can see that \(((q_1, \sigma_1, H'), (q_2, \sigma_2, H')) \in R'\).

- \(a \in \text{img}(\sigma_1) \setminus \text{img}(\sigma_2)\), say \(a = \sigma_1(i)\). Then, again \((q_1, i, q_1') \in \delta_1\) and \(\sigma_1' = \sigma_1\), but \(i \in S_1(\rho) \setminus \text{dom}(\rho)\). Thus, \((q_2, j, \rho, q_2) \in R\) for some \((q_1, h, \rho, q_2) \in R\). Thus, \((q_2, \sigma_2, H) \rightarrow_{\sigma_2} (q_2, \sigma_2, H')\), \(\sigma_2' = \sigma_2\) \(\rightarrow a\). Noting that \(\rho' = \sigma_1 \mapsto \sigma_2'\) and \(H' = H\), we have that \(((q_1, \sigma_1, H'), (q_2, \sigma_2, H')) \in R'\).

- \(a \in \text{img}(\sigma_2) \setminus \text{img}(\sigma_1)\), say \(a = \sigma_2(j)\). Since \(a \in H \setminus \text{img}(\sigma_1),\) we have some \((q_1, i', q_1') \in \delta_1,\) and \(\sigma_1' = \sigma_1[i \mapsto a]\). Moreover, \(j \in S_2(\rho) \setminus \text{dom}(\rho)\) and therefore \((q_2, j, q_2) \in R\) for some \((q_1, h, \rho, q_2) \in R\). Thus, \((q_2, \sigma_2, H) \rightarrow_{\sigma_2} (q_2, \sigma_2, H')\), \(\sigma_2' = \sigma_2\) and \(h = |H'|^{\ast}\). We have \(\rho = \sigma_1' \mapsto \sigma_2'\) and \(H' = H\) so \(((q_1, \sigma_1, H'), (q_2, j, \rho, q_2) \in R,\) and \((q_2, \sigma_2, H) \rightarrow_{\sigma_2} (q_2, \sigma_2, H')\), \(\sigma_2' = \sigma_2\) \(\rightarrow a\). We have that \(h' = |H'|^{\ast}\), so \(((q_1, \sigma_1', H'), (q_2, \sigma_2, H')) \in R'\).

Thus, \(R'\) is a simulation. If \(R\) is a symbolic bisimulation then, by symmetry, \(R'\) is a bisimulation. Finally, if \((q_0, h_0, \sigma_0) \in R\) then \(((q_0, \sigma_0, \emptyset), (q_0, \sigma_0, \emptyset)) \in R'\).

References


