Polynomial-Time Equivalence Testing for Deterministic Fresh-Register Automata

Andrzej S. Murawski
University of Oxford, UK

Steven J. Ramsay
University of Bristol, UK

Nikos Tzevelekos
Queen Mary University of London, UK

Abstract

Register automata are one of the most studied automata models over infinite alphabets. The complexity of language equivalence for register automata is quite subtle. In general, the problem is undecidable but, in the deterministic case, it is known to be decidable and in NP. Here we propose a polynomial-time algorithm building upon automata- and group-theoretic techniques. The algorithm is applicable to standard register automata with a fixed number of registers as well as their variants with a variable number of registers and ability to generate fresh data values (fresh-register automata). To complement our findings, we also investigate the associated inclusion problem and show that it is PSPACE-complete.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases automata over infinite alphabets, language equivalence, bisimilarity, computational group theory

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.72

Funding Supported by EPSRC grants EP/J019577, EP/P004172.

1 Introduction

Register automata [9, 15] are one of the simplest models of computation over infinite alphabets. They operate on an infinite domain of data by storing data values in a finite number of registers, where the values are available for future comparisons or updates. The automata can also recognise when a data value does not appear in any of the registers. Fresh-register automata [20] are an extension of register automata that can, in addition, generate data values not seen so far.

In recent years, register-based automata have appeared in a variety of contexts, ranging from database query languages [18] and programming language semantics [14] to run-time verification [7]. Since the very beginning, there has been great interest in extending learning algorithms to register automata [16, 4, 1, 5, 12], driven by applications in verification [11] and system modelling [21].

Register automata are closely related to nominal automata [3], which constitute a nominal counterpart of finite-state machines. Their closure properties and associated decision problems have first been studied in [9, 15]. One of the most fundamental and applicable decision problems is that of language equivalence, not least due to connections with query equivalence,

43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018).
Editors: Igor Potapov, Paul Spirakis, and James Worrell; Article No. 72; pp. 72:1–72:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

program equivalence and learning. Unfortunately, it turns out that the equivalence problem for register automata is in general undecidable [15]. Fortunately, it is decidable in the *deterministic* case (by reduction to emptiness using closure properties [9]).

Our paper presents the first polynomial-time algorithm for the problem. The algorithm is actually applicable to a wider class of automata, namely fresh-register automata with a variable number of registers.

To begin with, we exploit the observation that in the deterministic setting, language equivalence and bisimilarity are closely related. Secondly, because in our setting only different values can be stored in different registers [9], we take advantage of symbolic representations of bisimulation relations based on partial permutations. The proposed algorithm attempts to build such a bisimulation relation incrementally. To avoid potential exponential blow-ups, the candidate relations are stored in a concise fashion through generators of symmetric groups. Thanks to the fact that group membership testing works in polynomial-time [6] and subgroup chains can only have linear length [2], we can prove that the process of refining the candidate will terminate in polynomial time. Consequently, the equivalence problem for our variant of fresh-register automata is in P, which improves upon the best upper bound known so far, namely, NP [13].

A natural question is whether the polynomial-time bound could have been obtained via the associated inclusion problem. We give a negative answer to this question by showing that the inclusion problem in our setting is PSPACE-complete.

2 Automata

Let $\mathcal{D}$ be an infinite set (alphabet). Its elements will be called *data values* (in process algebra, the term *names* is used instead). We shall work with a deterministic model of register automata over $\mathcal{D}$. As in [9], we require that different registers contain different data values. To allow for more flexible use of registers, the number of available registers will be allowed to vary according to the current state. Register content can be both erased and created. Creation can be local (new element is guaranteed not to occur in any register) or global (new element is guaranteed not to have been encountered in the whole run). We give the formal definition below. In Remark 5 we discuss the motivation behind various restrictions and their relevance to polynomial-time complexity.

▶ **Definition 1.** Given a natural number r, we write $[1,r]$ for the set $\{i \in \mathbb{N} \mid 1 \leq i \leq r\}$. An r-register assignment is an *injective* function from a subset of $[1,r]$ to $\mathcal{D}$. An r-**deterministic fresh-register automaton** (r-DFRA) is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \mu, \delta, F \rangle$, where:

- Σ is a finite alphabet of *tags*;
- Q is a finite set of states, $q_0 \in Q$ is *initial* and $F \subseteq Q$ contains *final* states;
- $\mu : Q \to \mathcal{P}([1,r])$ is the *availability* function indicating which registers are filled at each state, we require $\mu(q_0) = \emptyset$;
- $\delta = \delta_{old} + \delta_{fresh}$ is the transition function, where $\delta_{old} : Q \times \Sigma \times [1,r] \rightharpoonup Q$ controls the use of existing register values and $\delta_{fresh} : Q \times \Sigma \rightharpoonup Q \times [1,r] \times \{\bullet, \circledast\}$ indicates when fresh values are created and how fresh they are.

To preserve the meaning of μ, we insist that $\delta_{old}(q,t,i) = q'$ implies $i \in \mu(q)$ and $\mu(q) \supseteq \mu(q')$ and $\delta_{fresh}(q,t) = (q', i, x)$ implies $\mu(q) \cup \{i\} \supseteq \mu(q')$. Note the use of $\supseteq$ instead of $=$. This allows for register erasures during computation. We shall write $q \xrightarrow{t,i} q'$ for $\delta_{old}(q,t,i) = q'$ and $q \xrightarrow{t,i^x} q'$ for $\delta_{fresh}(q,t) = (q', i, x)$.

Next we formalise how to obtain a labelled transition system for a given r-DFRA.

▶ **Definition 2.** A **labelled transition system** (LTS) over $\mathcal{A}ct$ is a tuple $\mathcal{S} = (\mathcal{A}ct, \mathbb{C}, \to)$, where $\mathbb{C}$ is a set of *configurations*, $\mathcal{A}ct$ is a set of *action labels*, and $\to \subseteq \mathbb{C} \times \mathcal{A}ct \times \mathbb{C}$. We write $\kappa \xrightarrow{\ell} \kappa'$ for $(\kappa, \ell, \kappa') \in \to$. $\mathcal{S}$ is *deterministic* if $\kappa \xrightarrow{\ell} \kappa_1$ and $\kappa \xrightarrow{\ell} \kappa_2$ imply $\kappa_1 = \kappa_2$.

An r-DFRA induces a deterministic LTS as follows.

▶ **Definition 3.** Given an r-DFRA $\mathcal{A} = \langle \Sigma, Q, q_0, \mu, \delta, F\rangle$, we define its set of configurations:

$$\mathbb{C}_{\mathcal{A}} = \{(q, \rho, H) \,|\, q \in Q,\ \rho : \mu(q) \to \mathcal{D} \text{ is injective, } \mathsf{rng}(\rho) \subseteq H \subseteq_{fin} \mathcal{D}\}$$

We refer to H as *history*. Let $\mathcal{S}(\mathcal{A})$ be the LTS $\langle \Sigma \times \mathcal{D}, \mathbb{C}_{\mathcal{A}}, \to_{\mathcal{A}}\rangle$, where $\to_{\mathcal{A}}$ is defined in the following way: a configuration (q_1, ρ_1, H_1) can make a transition to a configuration (q_2, ρ_2, H_2) reading input (t, d), written $(q_1, \rho_1, H_1) \xrightarrow{(t,d)} (q_2, \rho_2, H_2)$, if one of the conditions listed below is satisfied (the last two cases never overlap, because δ_{fresh} is a partial function).

- $d = \rho_1(i)$, $\delta_{\mathit{old}}(q_1, t, i) = q_2$, $\rho_2 = (\rho_1 \restriction \mu(q_2))$ and $H_2 = H_1$
- $d \notin \mathsf{rng}(\rho_1)$, $\delta_{\mathit{fresh}}(q_1, t) = (q_2, i, \bullet)$, $\rho_2 = (\rho_1[i \mapsto d] \restriction \mu(q_2))$ and $H_2 = H_1 \cup \{d\}$
- $d \notin H_1$, $\delta_{\mathit{fresh}}(q_1, t) = (q_2, i, \circledast)$, $\rho_2 = (\rho_1[i \mapsto d] \restriction \mu(q_2))$ and $H_2 = H_1 \cup \{d\}$

Note that $\mathcal{S}(\mathcal{A})$ does not depend on the initial and final parameters q_0 and F.

▶ **Definition 4.** The configuration $\kappa_{\mathcal{A}}^{init} = (q_0, \emptyset, \emptyset)$ will be called *initial*. A sequence of configurations $\kappa_0, \cdots, \kappa_n$ such that $\kappa_0 = \kappa_{\mathcal{A}}^{init}$ and $\kappa_i \xrightarrow{t_i, d_i} \kappa_{i+1}$ $(i = 0, \cdots, n-1)$ is called a run on the data word $(t_0, d_0) \cdots (t_{n-1}, d_{n-1})$. A run is *accepting* if $\kappa_n = (q_n, \rho_n, H_n)$ and $q_n \in F$. We write $\mathcal{L}(\mathcal{A})$ for the set of words from $(\Sigma \times \mathcal{D})^*$ with accepting runs, and call it *the language of* $\mathcal{A}$.

▶ Remark 5. Our definition allows for a variable number of available registers, i.e. it is more permissive than that in [15, 19]. This flexible register regime makes it possible to express certain common computational scenarios more directly: in particular, data values can be discarded ("forgotten") as soon as they are no longer needed (cf. garbage collection). Our result shows that poly-time equivalence testing is still possible with this added flexibility. At the same time, the flexible number of registers simplifies the technical development: one can combine an r_1-DFRA and a r_2-DFRA into a single $\max(r_1, r_2)$-DFRA (see Remark 7) by taking the disjoint union of states and transitions.

We rely on injective register assignments, as in the original definition of Francez and Kaminski [9]. This restriction is important for poly-time complexity, as the presence of multiple copies of the same value in registers could be used to model binary memory content (e.g. 1 is represented by the same value in two registers and 0 by different values). Consequently, this would imply a PSPACE lower bound. The appeal of injectivity lies in the fact no expressivity is lost but the transition function has a particularly simple shape and one can define the deterministic variant without introducing any additional comparisons between registers. While the injective discipline may seem restrictive, it has proved a good match for several prominent formalisms that arise in programming language semantics, and does not limit expressivity (e.g. [1]). For example, one can show that the automata support elegant translations from the pi-calculus [19]. They are also a natural target when it comes to investigating the semantics of programs with unbounded data – this is one of the original motivations mentioned in [9], which was also exploited in our work on the ML programming language [14].

The explicit availability function μ guarantees that whenever a transition refers to existing register content, the relevant value will be available. Allowing for transitions that may block on unavailable values is known to lead to NP-hardness [17], already for emptiness in the deterministic case. Our variant of automata makes it possible for the automaton to drop multiple data values from registers. Conversely, values can also be created but only one at a time. Of course, such single value creations can be combined to create multiple new values. However, the new values must also occur in labels. One can imagine adding a facility for spontaneous value creation, where locally or globally fresh values would be added to the registers without being present in labels. However, the resultant non-determinism could then be used to prove universality undecidable in the same way as for nondeterministic automata, e.g. the argument from [15] could be repeated by employing spontaneous value creation to guess the location of errors. Like in [1, 12], we assume that the registers are not filled at the beginning and are initialised through transitions.

▶ **Definition 6.** A relation $R \subseteq \mathbb{C} \times \mathbb{C}$ is called a *simulation* if, for all $(\kappa_1, \kappa_2) \in R$, if $\kappa_1 \xrightarrow{t,a} \kappa_1'$ then there is $\kappa_2 \xrightarrow{t,a} \kappa_2'$ such that $(\kappa_1', \kappa_2') \in R$. R is called a **bisimulation** if both R and R^{-1} are simulations. The union of all bisimulations is denoted $\sim$. Two configurations κ_1, κ_2 are bisimilar just if $\kappa_1 \sim \kappa_2$, i.e. there is some bisimulation R containing them.

In this paper we are concerned with the *language equivalence* problem for DFRA, i.e. the question whether, given r_1-DFRA $\mathcal{A}_1$ and r_2-DFRA $\mathcal{A}_2$, we have $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$. Our approach to the problem is bisimulation-oriented: language equivalence testing of $\mathcal{A}_1$ and $\mathcal{A}_2$ can be viewed as a bisimilarity problem for a single r-DFRA with $r = \max(r_1, r_2)$.

▶ Remark 7. We explain this reduction in a little more detail. First we transform $\mathcal{A}_i$ into $\mathcal{A}_i'$ as follows:

- remove all transitions leading to states from which it is impossible to reach a final state,
- add a new state f_i and designate it as the only final state,
- add transitions from former final states to the new final state on a new tag $t^{\$}$.

Suppose $\mathcal{S}(\mathcal{A}_i') = \langle \Sigma \times \mathcal{D}, \mathbb{C}_{\mathcal{A}_i'}, \to_{\mathcal{A}_i'} \rangle$ $(i = 1, 2)$ and consider the LTS $\mathcal{S}_{\mathcal{A}_1,\mathcal{A}_2} = \langle \Sigma \times \mathcal{D}, \mathbb{C}_{\mathcal{A}_1'} + \mathbb{C}_{\mathcal{A}_2'}, \to_{\mathcal{A}_1'} + \to_{\mathcal{A}_2'} \rangle$. Now language equivalence of the original automata is equivalent to checking whether $\kappa_{\mathcal{A}_1}^{init}$ and $\kappa_{\mathcal{A}_2}^{init}$ are bisimilar in $\mathcal{S}_{\mathcal{A}_1,\mathcal{A}_2}$. If $\mathcal{A}_i' = \langle \Sigma, Q_i, q_0^i, \mu_i, \delta_i, \{f_i\} \rangle$, then let $\mathcal{S}_{\mathcal{A}_1,\mathcal{A}_2} = \mathcal{S}_{\mathcal{A}'}$, where $\mathcal{A}'$ is the $\max(r_1, r_2)$-DFRA defined by $\langle \Sigma, Q_1 + Q_2, q', \mu_1 + \mu_2, \delta_1 + \delta_2, F' \rangle$ for any $q' \in Q_1 + Q_2$ and $F' \subseteq Q_1 + Q_2$. Note, the components q', F' can be chosen arbitrarily, because they do not contribute to the definition of bisimilarity over (the configuration graph of) $\mathcal{S}_{\mathcal{A}'}$.

3 Symbolic bisimulations

In this section we introduce symbolic representations of bisimulation relations, for configuration pairs with common history,[1] based on partial permutations. A *partial permutation* over $[1, n]$ is a bijection between two (possibly different) subsets of $[1, n]$. Let $\mathcal{IS}_n$ stand for the set of partial permutations over $[1, n]$ and $\mathcal{S}_X$ for the group of permutations over X. Let us consider the kind of possible scenarios that may arise in simulating transitions of a DFRA.

- A transition on a value already stored in a register can be matched by a transition on a stored value or a locally fresh transition, but never a globally fresh one.
- A globally fresh transition can be matched by a globally fresh transition or a locally fresh one, but never a transition on a stored value.
- A locally fresh transition can be matched by a transition on a stored value, a locally fresh transition or a globally fresh one.

[1] By Remark 7, it suffices to consider configuration pairs with common history.

The use of partial bijections will help us specify which cases may occur. Although we work with automata over r registers, we shall use partial permutations over $[1, n]$, where $n = 2r$. They will be used to express not only a matching between data values occurring in two sets of r registers (corresponding to two configurations that we examine for bisimilarity) but also to indicate which values forgotten by one set are still remembered by the other.

The number $2r$ may be surprising but it is needed to provide an accurate account of scenarios in which local freshness can be simulated by global freshness. Note that this is possible if the registers of the second configuration contain all the data values that have been forgotten by the first one (i.e. do not appear in its registers any more). Once the size of the history exceeds $2r$, this is no longer possible: because the first configuration has only r registers it will have forgotten more than r data values and, because the second configuration has only r registers, it cannot remember them all. Consequently, we only need to track matches between forgotten values and register content of the other configuration as long as the size of the history does not exceed $2r$. To keep track of such scenarios, it is convenient to imagine that there are $2r$ registers available and use partial permutations to match values in registers with values that were possibly forgotten until the size of the history is at most $2r$. Once that is exceeded, matchings between the real r registers suffice.

Given $\sigma \in \mathcal{IS}_r$ and $q_1, q_2 \in Q$, we write $\sigma \upharpoonright (q_1, q_2)$ for $\sigma \cap (\mu(q_1) \times \mu(q_2))$. Next, in accordance with the use of $2r$ registers discussed above, we introduce notions that will allow us to represent configurations in which only a subset $S \subseteq [1, 2r]$ of the registers is available along with certain values that are not stored any longer. The data values occurring in registers S will occupy the same positions (as specified by S), for other values we impose the convention that they should reside in the leftmost register positions that are unoccupied.

▶ **Definition 8** (Notation)**.** Given $S \subseteq T \subseteq [1, 2r]$, let $S \triangleleft T \in \mathcal{S}_{2r}$ be the permutation that shifts all elements in $T \setminus S$ to the left (inside the interval $[1, 2r]$) without interfering with S. Formally, if $T \setminus S$ is ordered as $[i_1, \cdots, i_k]$ then:

$$S \triangleleft T = (i_1\, i'_1); \cdots ; (i_k\, i'_k), \quad \text{where } i'_j \text{ is the } j\text{th smallest element in } [1, 2r] \setminus S.$$

Each $(i\, i')$ denotes a transposition and ; is the composition of permutations. For example, taking $S = \{3, 6\}$ and $T = \{1, 3, 4, 6, 7\}$, the permutation would be $S \triangleleft T = (1\,1); (4\,2); (7\,4)$ and, therefore, $(S \triangleleft T)(T) = \{1, 2, 3, 4, 6\}$.
Given $S \subseteq [1, 2r]$ and $h \leq 2r$ with $|S| \leq h$, we define $S^{\triangleleft h}$ to be the unique T satisfying $S \subseteq T \subseteq [1, 2r]$, $|T| = h$ and $T = (S \triangleleft T)(T)$. In other words, $S^{\triangleleft h}$ is obtained by adding $h - |S|$ smallest numbers from $[1, 2r] \setminus S$ to S. For instance, for $S = \{3, 6\}$: $S^{\triangleleft 2} = S$, $S^{\triangleleft 3} = \{1, 3, 6\}$, $S^{\triangleleft 4} = \{1, 2, 3, 6\}$, etc. Finally, given $\sigma \in \mathcal{IS}_{2r}$ and $S_1 \subseteq \mathsf{dom}(\sigma)$, $S_2 \subseteq \mathsf{rng}(\sigma)$:

- we write: $\sigma^{(S_1, S_2)\triangleleft} = (S_1 \triangleleft \mathsf{dom}(\sigma))^{-1}; \sigma; (S_2 \triangleleft \mathsf{rng}(\sigma))$,
- and extend the notation to $q_1, q_2 \in Q$ by: $\sigma^{(q_1, q_2)\triangleleft} = \sigma^{(\mu(q_1), \mu(q_2))\triangleleft}$.

Next we shall introduce a symbolic notion of simulation. Pairs of configurations will be represented by elements of $\mathcal{U}_0 = Q \times \mathcal{IS}_{2r} \times Q \times ([0, 2r] \cup \{\infty\})$: each pair is represented by the states it contains and a partial permutation representing the two register assignments (a matching between their common data values). In order to handle the interaction between the two kinds of fresh transitions we also introduce an additional element storing the size of the common history (∞ stands for "bigger than $2r$"). Below we define a subset $\mathcal{U}$ of $\mathcal{U}_0$ that characterises the elements compatible with availability information. Moreover, once the history becomes larger than $2r$, we reduce the matchings to r registers only (see above).

▶ **Definition 9.** Let $\mathcal{U}_0 = Q\times \mathcal{IS}_{2r}\times Q\times ([0,2r]\cup\{\infty\})$ and:

$$\begin{aligned}\mathcal{U} = \{ \quad (q_1,\sigma,q_2,h)\in\mathcal{U}_0 \mid \quad & h\leq 2r \implies (\mathsf{dom}(\sigma)=\mu(q_1)^{\lhd h}\wedge \mathsf{rng}(\sigma)=\mu(q_2)^{\lhd h})\\ \wedge \quad & h=\infty \implies (\sigma\in\mathcal{IS}_r\wedge\sigma\subseteq\mu(q_1)\times\mu(q_2))\,\}\end{aligned}$$

Given configurations κ_1,κ_2, with $\kappa_i=(q_i,\rho_i,H)$ for some common H, we define the set of symbolic representations of (κ_1,κ_2) by:

$$\mathsf{symb}(\kappa_1,\kappa_2)=\begin{cases}\{(q_1,\rho_1;\rho_2^{-1},q_2,\infty)\} & |H|>2r\\ \{(q_1,(\hat\rho_1;\hat\rho_2^{-1})^{\lhd(q_1,q_2)},q_2,|H|)\mid \rho_i\subseteq\hat\rho_i\wedge\mathsf{rng}(\hat\rho_i)=H\} & |H|\leq 2r\end{cases}$$

The essence of the above representation is the abstracting away from the register assignments ρ_1,ρ_2 to a partial permutation $\sigma\in\mathcal{IS}_{2r}$. If the history is large, then σ is simply a matching between the common values of ρ_1 and ρ_2. If, on the other hand, H contains at most $2r$ elements then σ is obtained by extending each ρ_i to some $\hat\rho_i$ that stores the full history H, and these pairs $(\hat\rho_1,\hat\rho_2)$ are then represented by recording their indices containing matching values.

We proceed with defining symbolic bisimulations. The clauses (a)-(f) in the definition below cover all possible kinds of simulation scenarios. Partial bijections help to capture the conditions under which simulation is possible.

▶ **Definition 10.** Let $\mathcal{A}=\langle\Sigma,Q,q_0,\mu,\delta,F\rangle$ be an r-DFRA. A *symbolic simulation* on $\mathcal{A}$ is a relation $R\subseteq\mathcal{U}$, with elements $(q_1,\sigma,q_2,h)\in R$ written $q_1\,R^h_\sigma\,q_2$, such that all $(q_1,\sigma,q_2,h)\in R$ satisfy the (FSYS) conditions in R. We say that a tuple (q_1,σ,q_2,h) satisfies the *fresh symbolic simulation conditions* (FSYS) *in* R if the following conditions hold, where (a-c) apply to $h\leq 2r$, and (d-e) to $h=\infty$:

(a) for all $q_1\xrightarrow{t,i}q_1'$,
  1. if $\sigma(i)\in\mu(q_2)$ then there is some $q_2\xrightarrow{t,\sigma(i)}q_2'$ with $q_1'\,R^h_{\sigma'}\,q_2'$ and $\sigma'=\sigma^{(q_1',q_2')\lhd}$,
  2. if $\sigma(i)=j'\in[1,2r]\setminus\mu(q_2)$ then there is some $q_2\xrightarrow{t,j^\bullet}q_2'$ with $q_1'\,R^h_{\sigma'}\,q_2'$ and $\sigma'=(\sigma;(j\,j'))^{(q_1',q_2')\lhd}$;

(b) for all $q_1\xrightarrow{t,i^\bullet}q_1'$ and $i'\in\mathsf{dom}(\sigma)\setminus\mu(q_1)$,
  1. if $\sigma(i')\in\mu(q_2)$ then there is some $q_2\xrightarrow{t,\sigma(i')}q_2'$ with $q_1'\,R^h_{\sigma'}\,q_2'$ and $\sigma'=((i\,i');\sigma)^{(q_1',q_2')\lhd}$,
  2. if $\sigma(i')=j'\in[1,2r]\setminus\mu(q_2)$ then there is some $q_2\xrightarrow{t,j^\bullet}q_2'$ with $q_1'\,R^h_{\sigma'}\,q_2'$ and $\sigma'=((i\,i');\sigma;(j\,j'))^{(q_1',q_2')\lhd}$;

(c) for all $q_1\xrightarrow{t,\ell_i}q_1'$ with $\ell_i\in\{i^\bullet,i^\circledast\}$ there is some $q_2\xrightarrow{t,\ell_j}q_2'$ with $\ell_j\in\{j^\bullet,j^\circledast\}$ and,
  1. if $h<2r$ then $q_1'\,R^{h+1}_{\sigma'}\,q_2'$ with $\sigma'=((i\,2r);\sigma[2r\mapsto 2r];(j\,2r))^{(q_1',q_2')\lhd}$,
  2. if $h=2r$ then $q_1'\,R^\infty_{\sigma'}\,q_2'$ with $\sigma'=\sigma[i\mapsto j]\restriction(q_1',q_2')$;

(d) for all $q_1\xrightarrow{t,i}q_1'$,
  1. if $\sigma(i)\in\mu(q_2)$ then there is some $q_2\xrightarrow{t,\sigma(i)}q_2'$ with $q_1'\,R^\infty_{\sigma'}\,q_2'$ and $\sigma'=\sigma\restriction(q_1',q_2')$,
  2. if $i\in\mu(q_1)\setminus\mathsf{dom}(\sigma)$ then there is some $q_2\xrightarrow{t,j^\bullet}q_2'$ with $q_1'\,R^\infty_{\sigma'}\,q_2'$ and $\sigma'=\sigma[i\mapsto j]\restriction(q_1',q_2')$;

(e) for all $q_1\xrightarrow{t,i^\bullet}q_1'$ and $j\in\mu(q_2)\setminus\mathsf{rng}(\sigma)$, there exists $q_2\xrightarrow{t,j}q_2'$ with $q_1\,R^\infty_{\sigma'}\,q_2'$ and $\sigma'=\sigma[i\mapsto j]\restriction(q_1',q_2')$;

(f) for all $q_1\xrightarrow{t,\ell_i}q_1'$ with $\ell_i\in\{i^\bullet,i^\circledast\}$ there is some $q_2\xrightarrow{t,\ell_j}q_2'$ with $\ell_j\in\{j^\bullet,j^\circledast\}$, $q_1'\,R^\infty_{\sigma'}\,q_2'$ and $\sigma'=\sigma[i\mapsto j]\restriction(q_1',q_2')$, and $\ell_i=i^\bullet\implies\ell_j=j^\bullet$.

Define now the inverse of R by $R^{-1} = \{ (q_2, \sigma^{-1}, q_1, h) \mid (q_1, \sigma, q_2, h) \in R \}$ and call R a **symbolic bisimulation** if both R and R^{-1} are symbolic simulations. We let *s-bisimilarity*, denoted $\overset{\mathsf{s}}{\sim}$, be the union of all symbolic bisimulations.

In the rest of the paper, given $R \subseteq \mathcal{U}$ and $h \in [1, 2r] \cup \{\infty\}$, we shall write R^h for the projection of R on h: $R^h = \{(q, \sigma, q') \mid (q, \sigma, q', h) \in R\}$.

▶ Remark 11. To gain some further intuition about the definition above, let us consider the 2-DFRA configurations $\kappa_i = (q_i, \rho_i, H)$, $i = 1, 2$, where: $\mu(q_1) = \mu(q_2) = \{1, 2\}$, $\rho_1 = \{(1, a), (2, b)\}$, $\rho_2 = \{(1, a), (2, c)\}$ and $H = \{a, b, c\}$.
The pair (κ_1, κ_2) can be represented symbolically by $(q_1, \sigma, q_2, 3) \in \mathsf{symb}(\kappa_1, \kappa_2) \subseteq \mathcal{U}$, where $\sigma = \{(1, 1), (2, 3), (3, 2)\}$. This represents the fact that ρ_1, ρ_2 share the data value a in their first register and each have a private value in their second register.[2] The (FSYS) conditions express symbolically what it takes for κ_2 to simulate κ_1, i.e. what is needed for $(q_1, \sigma, q_2, 3)$ to belong to a (symbolic) simulation R. Let us look at two sample cases.

- Suppose $q_1 \xrightarrow{t,1} q_1'$. Then, $\kappa_1 \xrightarrow{(t,a)} \kappa_1'$ and, in order for κ_2 to match this, it must be the case that $q_2 \xrightarrow{t,1} q_2'$. This is imposed by Condition (a)1 of (FSYS).
- If $q_1 \xrightarrow{t,2} q_1'$ then $\kappa_1 \xrightarrow{(t,b)} \kappa_1'$. Then, κ_2 can only match a transition on b using a locally fresh transition (Condition (a)2), so we must have e.g. $q_2 \xrightarrow{t,1^\bullet} q_2'$, yielding some $\kappa_2 \xrightarrow{(t,b)} \kappa_2'$.

In each of the cases above, the (FSYS) conditions also stipulate that the resulting representation of (κ_1', κ_2') must also be in R. In the second case, assuming $\kappa_i' = (q_i', \rho_i', H)$ and $\mu(q_i') = \{1, 2\}$, we have that $\rho_1' = \{(1, a), (2, b)\}$ and $\rho_2' = \{(1, b), (2, c)\}$, and the pair (κ_1', κ_2') is represented by $(q_1', \sigma', q_2', 3)$ with $\sigma' = \{(1, 3), (2, 1), (3, 2)\}$. Because $\sigma' = \sigma; (3\ 1) = (\sigma; (3\ 1))^{(q_1', q_2')\lhd}$, the (FSYS) conditions require $(q_1', \sigma', q_2', 3) \in R$.

The importance of symbolic bisimulations lies in that they precisely represent actual bisimulations in a finite way. Below, we first show that the symbolic representations of pairs of configurations are well defined (the choice of extensions $\hat{\rho}_i$ for the case of $|H| \leq 2r$ does not matter for $\overset{\mathsf{s}}{\sim}$), and then prove the representation property.

▶ **Lemma 12.** *For any κ_1, κ_2 with $\kappa_i = (q_i, \rho_i, H)$ and $|H| \leq 2r$, either $\mathsf{symb}(\kappa_1, \kappa_2) \subseteq \overset{\mathsf{s}}{\sim}$ or $\mathsf{symb}(\kappa_1, \kappa_2) \cap \overset{\mathsf{s}}{\sim} = \emptyset$.*

▶ **Proposition 13.** For any κ_1, κ_2 with common history, $\kappa_1 \sim \kappa_2$ iff $\mathsf{symb}(\kappa_1, \kappa_2) \subseteq \overset{\mathsf{s}}{\sim}$.

Although finite, symbolic bisimulations are of exponential size in the worst case (with respect to the automaton size) because of including the partial bijections σ. Our equivalence-testing algorithm for r-DFRA will rely on representations of candidate symbolic bisimulations in a succinct way. In order to spell out in what sense these representations will capture subsets of $\mathcal{U}$ we need to introduce the following closure operations.

▶ **Definition 14.** Let $R \subseteq \mathcal{U}$. Then $Cl(R)$ is defined to be the smallest subset X of $\mathcal{U}$ such that $R \subseteq X$ and X is closed under the following rules.

$$\frac{S = \mu(q)^{\lhd h} \quad h \leq 2r}{(q, \mathsf{id}_S, q) \in X^h} \qquad \frac{}{(q, \mathsf{id}_{\mu(q)}, q) \in X^\infty} \qquad \frac{(q_1, \sigma, q_2) \in X^h}{(q_2, \sigma^{-1}, q_1) \in X^h}$$

$$\frac{(q_1, \sigma, q_2) \in X^\infty \quad \sigma \subseteq \sigma'}{(q_1, \sigma', q_2) \in X^\infty} \qquad \frac{(q_1, \sigma_1, q_2) \in X^h \quad (q_2, \sigma_2, q_3) \in X^h}{(q_1, \sigma_1; \sigma_2, q_3) \in X^h}$$

[2] E.g. the value b is in register 2 of ρ_1 but is not present in ρ_2. Seeing $\hat{\rho}_2$ as an expansion of ρ_2 to 3 registers (with register 3 containing forgotten values), we set $\hat{\rho}_2(3) = b$ and therefore $\sigma(2) = 3$.

The next lemma provides a handle on proving that closures $\mathit{Cl}(R)$ satisfy (FSʏS) conditions.

▶ **Lemma 15.** *Let $R, P \subseteq \mathcal{U}$ with $R = R^{-1}$. If all $g \in R$ satisfy the* (FSʏS) *conditions in P then all $g' \in \mathit{Cl}(R)$ satisfy the* (FSʏS) *conditions in $\mathit{Cl}(P)$.*

▶ **Corollary 16.** $\mathit{Cl}(\overset{\mathsf{s}}{\sim}) = \overset{\mathsf{s}}{\sim}$.

Proof. It suffices to show the left-to-right inclusion. All elements in $\overset{\mathsf{s}}{\sim}$ satisfy the (FSʏS) conditions in $\overset{\mathsf{s}}{\sim}$. Hence, by the previous lemma, all elements of $\mathit{Cl}(\overset{\mathsf{s}}{\sim})$ satisfy the (FSʏS) conditions in $\mathit{Cl}(\overset{\mathsf{s}}{\sim})$. This implies that $\mathit{Cl}(\overset{\mathsf{s}}{\sim})$ is a symbolic bisimulation. Thus, $\mathit{Cl}(\overset{\mathsf{s}}{\sim}) \subseteq \overset{\mathsf{s}}{\sim}$. ◀

▶ Remark 17. One may wonder to what extent our techniques apply to simulation rather than bisimulation. Although symbolic simulation can be related to simulation, our methods crucially exploit the fact that bisimilarity is symmetric. This is reflected in the top right rule of Definition 14, which introduces inverses, and enables us to develop a group-theoretic representation scheme in the next section.

4 Representation

Our algorithm for DFRA equivalence will rely on manipulating sets $\mathcal{H} \subseteq \mathcal{U}$ that, for positive instances, will ultimately converge to a symbolic bisimulation relation. We shall handle them through succinct representations based on group theory, whose shape is inspired by the structure of bisimulation relations [13]. The backbone of a generating system, to be defined next, is an equivalence relation $\diamond^h$ on states. As explained in Definition 19, the relation specifies which pairs of states may actually feature in tuples of the represented subset of $\mathcal{U}$.

▶ **Definition 18.** A *generating system* $\mathcal{R}$ consists of a set $\{\mathcal{R}^h \mid h \in [0, 2r] \cup \{\infty\}\}$, where each $\mathcal{R}^h = \langle \diamond^h, \{(q^h_C, X^h_C, G^h_C) \mid C \in Q/\diamond^h\}, \{\sigma^h_q \mid q \in Q\}\rangle$ satisfies the following constraints.

- $\diamond^h \subseteq Q \times Q$ is an equivalence relation.
- For any $\diamond^h$-equivalence class C:
  - q^h_C is a state from C (class representative);
  - $X^h_C = \mu(q^h_C)^{\lhd h}$ for $h \in [0, 2r]$ and $X^\infty_C \subseteq \mu(q^\infty_C)$;
  - $\emptyset \neq G^h_C \subseteq \mathcal{S}_{X^h_C}$.
- For any $q \in Q$, $C = [q]_{\diamond^h}$ and $h \in [0, 2r]$, we have $\sigma^h_q \in \mathcal{IS}_{2r}$ with $\mathsf{dom}(\sigma^h_q) = \mu(q^h_C)^{\lhd h}$ and $\mathsf{rng}(\sigma^h_q) = \mu(q)^{\lhd h}$. Moreover, $\sigma^\infty_q \in \mathcal{IS}_r$ and $\mathsf{dom}(\sigma^\infty_q) = X^\infty_C$. Finally, $\sigma^h_{q^h_C} = \mathsf{id}_{X^h_C}$.

Thus, at each level h, a generating system partitions the set of states into equivalence classes according to $\diamond^h$ and each class has a representative q^h_C, which is "connected" to each element of the class via σ^h_q. Each representative q^h_C is also equipped with a subset $X^h_C \subseteq [0, 2r]$ and a set G^h_C of permutations (generators) from $\mathcal{S}_{X^h_C}$.

▶ **Definition 19.** Let $\mathcal{R}$ be a generating system. The subset of $\mathcal{U}$ represented by $\mathcal{R}$, written $\mathsf{Gen}(\mathcal{R})$, is defined to be $\mathit{Cl}(\mathcal{H}_\mathcal{R})$, where $\mathcal{H}_\mathcal{R} = \bigcup_{h=0}^{2r} \mathcal{H}^h_\mathcal{R} \cup \mathcal{H}^\infty_\mathcal{R}$ and, for any $h \in [0, 2r] \cup \{\infty\}$, we take $\mathcal{H}^h_\mathcal{R} = \{\, (q^h_C, g^h_C, q^h_C, h) \mid C \in Q/\diamond^h,\ g^h_C \in G^h_C \,\} \cup \{\, (q^h_C, \sigma^h_q, q, h) \mid q \in Q, C = [q]_{\diamond^h} \,\}$.

▶ **Example 20.** The representation system $\mathcal{R}_{init}$ is defined by the following components.

- $\diamond^h = \{(q, q) \mid q \in Q\}$. Note that $[q]_{\diamond^h} = \{q\}$.
- For any equivalence class $C = \{q\}$ we have: $q^h_C = q$, $X^h_C = \mu(q)^{\lhd h}$ $(h \in [0, 2r])$, $X^\infty_C = \mu(q)$, $G^h_C = \{\mathsf{id}_{X^h_C}\}$.
- For any q, $\sigma^h_q = \mathsf{id}_{X^h_C}$.

Note that $\mathsf{Gen}(\mathcal{R}_{init}) = \mathit{Cl}(\emptyset)$.

Next we examine how generating systems can be employed in algorithms. We are particularly interested in membership testing and a special kind of updates.

4.1 Membership

The next lemma reduces testing for membership in $\mathsf{Gen}(\mathcal{R})$ to the classic problem of group membership testing [6]. Given $G \subseteq \mathcal{S}_X$, we let $\mathsf{Sub}(G)$ be the subgroup of $\mathcal{S}_X$ spanned by G.

▶ **Lemma 21.** *Let $\mathcal{R}$ be a generating system, $u = (q_1, \sigma, q_2, h) \in \mathcal{U}$ and $\overline{\sigma} = \sigma^h_{q_1}; \sigma; (\sigma^h_{q_2})^{-1}$. Then $u \in \mathsf{Gen}(\mathcal{R})$ if and only if $q_1 \diamond^h q_2$ and $\overline{\sigma} \in \mathsf{Sub}(G^h_C)$, where $C = [q_1]_{\diamond^h} = [q_2]_{\diamond^h}$.*

4.2 Update

Suppose $\mathsf{Gen}(\mathcal{R}) = \mathit{Cl}(\mathcal{H})$. We explain how, given $u = (q_1, \sigma, q_2, h) \in \mathcal{U}$, one can update $\mathcal{R}$ to $\mathcal{R}'$ so that $\mathsf{Gen}(\mathcal{R}') = \mathit{Cl}(\mathcal{H} \cup \{u\})$. Of course, if $u \in \mathsf{Gen}(\mathcal{R})$ then it suffices to take $\mathcal{R}' = \mathcal{R}$. Thus, let us assume $u \notin \mathsf{Gen}(\mathcal{R})$. By Lemma 21, this corresponds to the following cases, where $\overline{\sigma} = \sigma^h_{q_1}; \sigma; (\sigma^h_{q_2})^{-1}$.

1. $q_1 \diamond^h q_2$ and either (a) or (b) holds, where $C = [q_1]_{\diamond^h} = [q_2]_{\diamond^h}$:
   (a) $\overline{\sigma} \in \mathcal{S}_{X^h_C} \setminus \mathsf{Sub}(G^h_C)$,
   (b) $\overline{\sigma} \notin \mathcal{S}_{X^h_C}$, i.e. $\mathsf{dom}(\overline{\sigma}) \subsetneq X^h_C$.
2. $q_1 \diamond^h q_2$ does not hold.

Observe that 1.(b) will never arise for $h \neq \infty$ due to the definitions of $\mathcal{U}$ and $\mathcal{R}$. Note also that, for $h \neq \infty$, X^h_C is uniquely determined by q^h_C. However, this is not the case for X^∞_C.

Before we explain how to tackle each case, we introduce several technical lemmas that examine how partial permutations interact. They will inform the performance of updates based on modifying X^∞_C.

▶ **Lemma 22.** *Given $I \subseteq \mathcal{IS}_r$, let $\chi_I = \{\sigma \mid \sigma = \sigma_1^{\epsilon_1}; \cdots; \sigma_k^{\epsilon_k},\ k > 0,\ \sigma_i \in I,\ \epsilon_i \in \{1, -1\}\}$ and $\mathcal{D}_I = \{\mathsf{dom}(\sigma) \mid \sigma \in \chi_I\}$. Then χ_I is closed under composition and inversion, and $\mathcal{D}_I$ is closed under intersection.*

▶ **Lemma 23.** *Given $I \subseteq \mathcal{IS}_r$, let $B_I = \bigcap_{X \in \mathcal{D}_I} X$ be called the* base *of I. Then we have:*

1. *$B_I \in \mathcal{D}_I$ and $\mathsf{id}_{B_I} \in \chi_I$.*
2. *Given $\sigma \in \mathcal{IS}_r$ and $X \subseteq [1, r]$, let us write $\sigma \restriction X$ for $\mathsf{id}_X; \sigma$. Then, for any $\sigma \in I$, $\sigma \restriction B_I \in \chi_I$ and $\sigma \restriction B_I$ is a permutation of B_I.*

Next we show that, given I, the base B_I can be calculated via graph reachability.

▶ **Lemma 24.** *Let $I \subseteq \mathcal{IS}_r$. Consider the undirected graph $G_I = (V, E)$ with $V = [1, r]$, where $\{j_1, j_2\} \in E$ iff there exists $\sigma \in I$ such that $\sigma(j_1) = j_2$ or $\sigma(j_2) = j_1$. We shall call $v \in [1, r]$* endangered *if there exists $\sigma \in I$ such that $v \notin \mathsf{dom}(\sigma)$ or $v \notin \mathsf{rng}(\sigma)$. For any $i \in [1, r]$, $i \in B_I$ if and only if no endangered vertex is reachable from i in G_I.*

4.3 Update implementation

Finally, we are ready to return to the main issue of representation update. We discuss the three cases (1.(a), 1.(b) and 2.) in turn. Recall that $u = (q_1, \sigma, q_2, h) \in \mathcal{U}$ and $\overline{\sigma} = \sigma^h_{q_1}; \sigma; (\sigma^h_{q_2})^{-1}$.

1. **(a)** Here we have $\overline{\sigma} \in \mathcal{S}_{X^h_C} \setminus \mathsf{Sub}(G^h_C)$. To update the system in order to represent σ, it suffices to add $\overline{\sigma}$ to G^h_C without changing anything else.
1. **(b)** Here we have $\mathsf{dom}(\overline{\sigma}) \subsetneq X^h_C$ and $h = \infty$. In order to capture $\overline{\sigma}$, we replace X^∞_C with B_I, where $I = G^\infty_C \cup \{\overline{\sigma}\}$, and set $G^\infty_C = \{\sigma \restriction B_I \mid \sigma \in I\}$. Note that, by Lemma 23, all the elements are permutations, as required. Similarly to G^∞_C, we replace σ^∞_q with $\sigma^\infty_q \restriction B_I$ for each $q \in C$. Other elements of the system remain the same.

```
$i$=0; $\mathcal{R}_0$ = $\mathcal{R}_{init}$; $\Delta$ = $\{u_0\}$; $\Delta_0$ = $\emptyset$;
while ($\Delta$ is not empty) do {
    $u$ = $\Delta$.get();
    if $u \notin \mathsf{Gen}(\mathcal{R}_i)$ {
        if one-step test fails for $u$ return NO;
        $\Delta$.add(succ-set($u$));
        $\Delta_{i+1}$ = $\Delta_i$.add($\{u\}$);
        $\mathcal{R}_{i+1}$ = $\mathcal{R}_i$ updated with $u$;
        $i$=$i$+1;
    }
}
return YES
```

Figure 1 Bisimilarity checking algorithm.

2. This case is the hardest as we need to merge two different equivalence classes, namely, $C_1 = [q_1]_{\diamond^h}$ and $C_2 = [q_2]_{\diamond^h}$ into a single one $C = C_1 \cup C_2$ (formally, this is a change to $\diamond^h$). For the new class C, we take $q_C^h = q_{C_1}^h$.
   Next we discuss $X_{q_C}^h$. Given $\tau \in G_{q_{C_2}}^h$, let $\widehat{\tau} = \overline{\sigma};\tau;(\overline{\sigma})^{-1}$ and consider $I = G_{q_{C_1}}^h \cup \{\widehat{\tau} \mid \tau \in G_{q_{C_2}}^h\}$. We shall set $X_{q_C}^h$ to B_I. Note that, if $h \neq \infty$, all elements of I will have the same domains, so in this case $X_{q_C}^h$ will not change. As before, we set $G_C^h = \{\sigma \restriction B_I \mid \sigma \in I\}$. We also modify σ_q^h, but only for $q \in C_1 \cup C_2$. If $q \in C_1$, we take $\sigma_q^h \restriction B_I$ instead of σ_q^h. For $q \in C_2$, we need to take the change of representative into account and take $(\overline{\sigma};\sigma_q^h) \restriction B_I$ instead of σ_q^h.
   (For this to be a correct choice, we need to show that $\mathsf{dom}(\overline{\sigma};\sigma_q^h) \supseteq B_I$. This is indeed so, because $\mathsf{dom}(\overline{\sigma};\sigma_q^h) = \mathsf{dom}(\overline{\sigma};\mathsf{id}_{X_{q_{C_2}}^h})$, by $\mathsf{dom}(\sigma_q^h) = X_{q_{C_2}}^h$, and $\mathsf{dom}(\overline{\sigma};\mathsf{id}_{X_{q_{C_2}}^h}) = \mathsf{dom}(\overline{\sigma};\tau) \supseteq \mathsf{dom}(\overline{\sigma};\tau;\overline{\sigma}^{-1}) = \mathsf{dom}(\widehat{\tau}) \supseteq B_I$ for any $\tau \in G_{q_{C_2}}^h$.)

Recall that we work under the assumption that $\mathsf{Gen}(\mathcal{R}) = \mathit{Cl}(\mathcal{H})$ and let us write $\mathcal{R}'$ for the updated representation system. In each of the above cases, the modifications contribute to $\mathcal{H}_{\mathcal{R}'}$ only elements from $\mathit{Cl}(\mathcal{H} \cup \{u\})$. This is completely clear for 1.(a). For 1.(b) and 2., we need to appeal to Lemma 23 ($\sigma \restriction B_I \in \chi_I$) and the use of composition/inversion during construction. Consequently, $\mathsf{Gen}(\mathcal{R}') \subseteq \mathit{Cl}(\mathcal{H} \cup \{u\})$.

Conversely, $\mathit{Cl}(\mathcal{H} \cup \{u\}) \subseteq \mathsf{Gen}(\mathcal{R}')$, because all elements of $\mathcal{R}$ as well as u have been integrated into $\mathcal{R}'$, either directly or through composition and reductions to X_C^∞. Thanks to the defining rules for Cl (notably, closure under composition and inclusion), such changes preserve representability.

5 Algorithm

Finally, we present the algorithm for deciding whether two configurations $\kappa_i = (q_i, \rho_i, H)$ are bisimilar. Let $u_0 = (q_1, \sigma, q_2, h)$ be an arbitrary element of $\mathsf{symb}(\kappa_1, \kappa_2)$. By Lemma 12 and Proposition 13, bisimilarity of κ_1, κ_2 amounts to checking whether u_0 belongs to a symbolic bisimulation. Our algorithm will determine whether or not this is the case.

The algorithm is presented in Figure 1. It is similar in flavour to the classic Hopcroft-Karp algorithm for DFA [8], which maintains *sets of pairs of states*. In contrast, we work with *sets of elements from the set* $\mathcal{U}$, i.e. four-tuples (q_1, σ, q_2, h). As subsets of $\mathcal{U}$ may have exponential size, we do not store them explicitly. Instead we take advantage of the representation systems developed in the previous section.

Starting from u_0, the algorithm maintains generating systems $\mathcal{R}_i$, beginning with $\mathcal{R}_{init}$. We assume the availability of a data structure Δ for storing multisets of elements of $\mathcal{U}$ (e.g. a queue), equipped with emptiness testing, a *get* method that removes an occurrence of an element u from Δ and returns it as a result, and an *add* method that extends Δ with the elements listed as its argument .

▶ Remark 25. Each of the conditions for (FSys) relies on finding a matching transition satisfying an extra constraint spelt out in terms of R^h. If (FSys) fails for u or u^{-1} because no potential transition exists, we shall say that the *one-step* test fails for $u \in \mathcal{U}$. Note that we are not concerned whether the extra constraint is satisfied – we only check if a transition with the specified source and label exists.
Because we work with deterministic automata, the availability of a transition implies uniqueness. Consequently, if u passes the one-step test, the (FSys) rules for u and u^{-1} deliver a unique set of conditions that need to be checked in order for (FSys) to be satisfied (for u and u^{-1}). Formally, these conditions can be captured as a subset of $\mathcal{U}$ and we shall call them the *successor set* of u, written $\mathsf{succ\text{-}set}(u)$. In the code above the membership test ($u \notin \mathsf{Gen}(\mathcal{R}_i)$) is performed as specified in Section 4.1, while the extension of $\mathcal{R}_i$ with u follows Section 4.2.

The correctness arguments rely on the following invariants.

▶ **Lemma 26.** *The loop satisfies the following invariants.*

(a) *For any $i \geq 0$, $\mathsf{Gen}(\mathcal{R}_i) = Cl(\Delta_i)$ and, for all $v \in \Delta_i$, v, v^{-1} satisfy the* (FSys) *conditions in $Cl(\Delta_i \cup \Delta)$.*

(b) *For any symbolic bisimulation relation R, if $u_0 \in R$ then $\Delta \subseteq R$.*

▶ **Theorem 27** (Partial Correctness). *When the Algorithm returns YES, there exists a symbolic bisimulation containing u_0. When the Algorithm returns NO, no symbolic bisimulation can contain u_0.*

Proof. When the Algorithm returns YES, Δ is empty. Consequently, Lemma 26 (a) implies that each element of $\Delta_i \cup \Delta_i^{-1}$ satisfies the (FSys) conditions in $Cl(\Delta_i)$, so $Cl(\Delta_i)$ is a symbolic bisimulation relation by Lemma 15.

- If $u_0 \notin \mathsf{Gen}(\mathcal{R}_{init})$ then $i > 0$ and $u_0 \in \Delta_0 \subseteq \Delta_i$. Thus, $u_0 \in Cl(\Delta_i)$.
- If $u_0 \in \mathsf{Gen}(\mathcal{R}_{init})$ then the Theorem is also true, because $\mathsf{Gen}(\mathcal{R}_{init})$ is a symbolic bisimulation.

Thus, in each case, there exists a symbolic bisimulation containing u_0. The NO case follows immediately from Lemma 26 (b). ◀

Next we argue why the algorithm terminates and its complexity is polynomial. To that end, it will be useful to introduce the following measure on representation systems.

▶ **Definition 28.** Given $\mathcal{R}$, let $m_{\mathcal{R}} : ([0, 2r] \cup \{\infty\}) \times Q \to \mathbb{N} \times \mathcal{P}(\mathcal{IS}_{2r})$ be defined as follows.

$$m_{\mathcal{R}}(h, q) = (|Q/\diamond^h| + |X_{[q]_{\diamond^h}}|, \mathsf{Sub}(G^h_{[q]_{\diamond^h}}))$$

Given $(n_1, H_1), (n_2, H_2) \in \mathbb{N} \times \mathcal{P}(\mathcal{IS}_{2r})$, let $(n_1, H_1) \leq (n_2, H_2)$ stand for $n_1 < n_2$ or ($n_1 = n_2$ and $H_1 \supseteq H_2$). For $\mathcal{R}_1, \mathcal{R}_2$, we then write $m_{\mathcal{R}_1} \leq m_{\mathcal{R}_2}$ iff for all (h, q), $m_{\mathcal{R}_1}(h, q) \leq m_{\mathcal{R}_2}(h, q)$.

▶ **Lemma 29.** *Given a representation system $\mathcal{R}$ and $u \in \mathcal{U}$, let $\mathcal{R}'$ be its extension by u constructed in Section 4.2. Then $m_{\mathcal{R}'} \lneq m_{\mathcal{R}}$.*

▶ **Theorem 30.** *The Algorithm terminates.*

Proof. We argue by contradiction. Observe that, if the Algorithm does not terminate, there can be no bound on the number of times that elements are added to the queue. This will generate an infinite sequence of generating systems $\mathcal{R}_0, \mathcal{R}_1, \cdots, \mathcal{R}_i, \mathcal{R}_{i+1}, \cdots$, where each $\mathcal{R}_{i+1}$ extends $\mathcal{R}_i$ according to Section 4.2. By Lemma 29, $m_{\mathcal{R}_0} \gneqq m_{\mathcal{R}_1} \gneqq \cdots \gneqq m_{\mathcal{R}_i} \gneqq \cdots$. Given that the first components (numbers) in $m_{\mathcal{R}i}(h, q)$ are bounded by $|Q| + 2r$, for this to happen, we would need to have an infinite chain of subgroups of $\mathcal{S}_X$ for some $X \subseteq [1, 2r]$. This contradicts the bound from [2]. ◀

Following a similar pattern of reasoning, we can establish a bound on the number of generating systems that can be produced by the Algorithm, which happens to correspond to the value of i. We have already observed that the integers in the first component of $m_{\mathcal{R}i}(h, q)$ are bounded by $|Q| + 2r$. Consequently, that particular component can decrease $|Q| + 2r$ times for $h \in [0, 2r]$ and $|Q|$ times for $h = \infty$ (the sets X_C^h are not modified in this case). As for the second component, the bound on the number of times it can change is $2r + O(1)$ [2]. Because the decreases may occur for any q, h, the overall bound on i is $\underbrace{|Q|(2r+1)(|Q|+2r)2r}_{h\in[0,2r]} + \underbrace{|Q||Q|2r}_{h=\infty} = O(|Q|^2r^2 + |Q|r^3) + O(|Q|^2r)$. Each increase of i is accompanied by the addition of one-step successors to the queue. There are $O(r)$ such successors and their generation can take $O(r)$ steps due to rearrangements on permutations. Consequently, the handling of each element of u may require $O(r^2)$ steps ($O(r)$ steps for $h = \infty$). This does not take group membership tests into account, for which there exist polynomial-time algorithms [6]. Thus, the complexity can be conservatively bounded by $O(|Q|^2r^5p(r))$ steps, where $p(r)$ refers to the complexity of membership testing for $\mathcal{S}_{2r}$ (which bounds those for $\mathcal{S}_X$, where $X \subseteq [1, 2r]$). Note that for $h = \infty$, the complexity is $O(|Q|^2r^2p(r))$. Knuth [10] reports on an algorithm for which $p(r) = O(r^5 + mr^2)$, where m is the number of membership queries, adding that it runs considerably faster in practice.

▶ **Theorem 31.** *The language equivalence problem for r-DFRA is in PTIME.*

A natural question for further study is whether the problem is PTIME-complete. It is certainly NL-hard, by reduction from DFA.

Implementation

An implementation of our algorithm is available from `http://github.com/stersay/deq`. Although we leave a full analysis of our empirical results to a future publication, it is worth mentioning that initial case studies indicate that the high-degree of r in the worst case is not a hindrance in practice. For example, in comparing two encodings of automata simulating finite stack machines (considered previously by [12]), bisimulations for automata with $r \leq 1500$ can be computed in less than one minute.

6 Inclusion

Equivalence can often be attacked by reduction to the associated inclusion problem. As we explain next, for DFRA this route would not yield a PTIME bound.

▶ **Theorem 32.** *The inclusion problem for r-DFRA is in PSPACE-complete.*

Proof. For membership in PSPACE, we first note that inclusion can be reduced to simulation. Now observe that if there is a winning strategy for Attacker over the infinite alphabet then there will be one if $2r + 1$ letters are used. This is because $2r + 1$ letters are sufficient to simulate the effect of attacks that rely on global freshness: with $2r+1$ letters available it is always possible to choose a letter that is not stored in either set of the r-registers and, thus, attacks based on global freshness can be simulated. Consequently, failures of inclusion can be detected by guessing the relevant word using $2r+1$ letters on the understanding that for globally fresh transitions we need to choose a letter not occurring in any of the $2r$ registers. To this end, polynomial space is needed to keep track of the current content of both sets of registers.

We can show PSPACE-hardness already for DFRA without global freshness, which we refer to as DRA. Because DRA can be complemented easily, we actually show that the equivalent problem of DRA intersection emptiness is PSPACE-hard. This is done by reduction from non-emptiness of deterministic linear-bounded Turing machines. The main difficulty in the argument is to represent the tape through registers. This seems impossible at first given that a register assignment must contain different data values. We overcome this by constructing two $(n+1)$-DRA $\mathcal{A}_1$, $\mathcal{A}_2$ such that whenever they synchronise on a data word, their register assignments ρ_1, ρ_2 represent the content of n tape cells as follows: 0 in the ith cell is represented by $\rho_1(i) = \rho_2(i)$, and 1 by $\rho_1(i) \notin \mathsf{rng}(\rho_2)$. The $(n+1)$th register plays a technical role that helps us to maintain the representation. The position of the head and state of the machine are maintained in the state of the automata. ◀

References

1 F. Aarts, P. Fiterau-Brostean, H. Kuppens, and F. W. Vaandrager. Learning register automata with fresh value generation. In *Proceedings of ICTAC*, volume 9399 of *Lecture Notes in Computer Science*, pages 165–183. Springer, 2015.

2 L. Babai. On the length of subgroup chains in the symmetric group. *Communications in Algebra*, 14(9):1729–1736, 1986.

3 M. Bojańczyk, B. Klin, and S. Lasota. Automata theory in nominal sets. *LMCS*, 10(3), 2014.

4 B. Bollig, P. Habermehl, M. Leucker, and B. Monmege. A robust class of data languages and an application to learning. *Logical Methods in Computer Science*, 10(4), 2014.

5 S. Cassel, F. Howar, B. Jonsson, and B. Steffen. Active learning for extended finite state machines. *Formal Asp. Comput.*, 28(2):233–263, 2016.

6 M. L. Furst, J. E. Hopcroft, and E. M. Luks. Polynomial-time algorithms for permutation groups. In *Proceedings of FOCS*, pages 36–41. IEEE Computer Society, 1980.

7 R. Grigore, D. Distefano, R. L. Petersen, and N. Tzevelekos. Runtime verification based on register automata. In *Proceedings of TACAS*, LNCS. Springer, 2013.

8 J. E. Hopcroft and R. M. Karp. A linear algorithm for testing equivalence of finite automata. Technical Report 114, Cornell University, 1971.

9 M. Kaminski and N. Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

10 D. E. Knuth. Efficient representation of perm groups. *Combinatorica*, 11(1):33–43, 1991.

11 M. Leucker. Learning meets verification. In *Proceedings of FMCO*, volume 4709 of *Lecture Notes in Computer Science*, pages 127–151, 2007.

12 J. Moerman, M. Sammartino, A. Silva, B. Klin, and M. Szynwelski. Learning nominal automata. In *Proceedings of POPL*, pages 613–625. ACM, 2017.

13 A. S. Murawski, S. J. Ramsay, and N. Tzevelekos. Bisimilarity in fresh-register automata. In *Proceedings of LICS*, pages 156–167, 2015.

14 A. S. Murawski and N. Tzevelekos. Algorithmic nominal game semantics. In *Proceedings of ESOP*, volume 6602 of *Lecture Notes in Computer Science*, pages 419–438. Springer-Verlag, 2011.

15 F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

16 H. Sakamoto. *Studies on the Learnability of Formal Languages via Queries.* PhD thesis, Kyushu University, 1998.

17 H. Sakamoto and D. Ikeda. Intractability of decision problems for finite-memory automata. *Theor. Comput. Sci.*, 231(2):297–308, 2000.

18 T. Schwentick. Automata for XML - A survey. *J. Comput. Syst. Sci.*, 73(3):289–315, 2007.

19 N. Tzevelekos. Full abstraction for nominal general references. *Logical Methods in Computer Science*, 5(3), 2009.

20 N. Tzevelekos. Fresh-register automata. In *Proceedings of POPL*, pages 295–306. ACM Press, 2011.

21 F. W. Vaandrager. Model learning. *Commun. ACM*, 60(2):86–95, 2017.