

# Polynomial time algorithms for inclusion and equivalence of deterministic parity acceptors<sup>\*</sup>

Dana Angluin<sup>1</sup> and Dana Fisman<sup>2</sup>

<sup>1</sup> Yale University

<sup>2</sup> Ben-Gurion University

**Abstract.** The class of omega languages recognized by deterministic parity acceptors (DPAs) is exactly the regular omega languages. The inclusion problem is the following: given two DPAs  $\mathcal{P}_1$  and  $\mathcal{P}_2$ , determine whether the language recognized by  $\mathcal{P}_1$  is a subset of the language recognized by  $\mathcal{P}_2$ , and if not, return an ultimately periodic omega word accepted by  $\mathcal{P}_1$  but not  $\mathcal{P}_2$ . We describe a polynomial time algorithm to solve this problem. Corollaries include polynomial time algorithms to solve the equivalence problem for DPAs and also the inclusion and equivalence problems for deterministic Büchi and coBüchi acceptors.

## 1 Preliminaries

### 1.1 Words and Automata

If  $\Sigma$  is a finite alphabet of symbols, then  $\Sigma^*$  denotes the set of all finite words over  $\Sigma$ ,  $\varepsilon$  denotes the empty word,  $|u|$  denotes the length of the finite word  $u$ , and  $\Sigma^+$  denotes the set of all nonempty finite words over  $\Sigma$ . Also,  $\Sigma^\omega$  denotes the set of all infinite words over  $\Sigma$ , termed  $\omega$ -words. For a finite or infinite word  $w$ , the successive symbols are indexed by positive integers, and  $w[i]$  denotes the symbol with index  $i$ . For words  $u \in \Sigma^*$  and  $v \in \Sigma^+$ ,  $u(v)^\omega$  denotes the ultimately periodic  $\omega$ -word consisting of  $u$  followed by infinitely many copies of  $v$ .

A *complete deterministic automaton* is a tuple  $\mathcal{M} = \langle \Sigma, Q, q_\iota, \delta \rangle$  consisting of a finite alphabet  $\Sigma$  of symbols, a finite set  $Q$  of states, an initial state  $q_\iota \in Q$ , and a transition function  $\delta : Q \times \Sigma \rightarrow Q$ . In this paper, *automaton* will mean a complete deterministic automaton. We extend  $\delta$  to the domain  $Q \times \Sigma^*$  in the usual manner, and for  $u \in \Sigma^*$ , define  $\mathcal{M}(u) = \delta(q_\iota, u)$ , the state of  $\mathcal{M}$  reached on input  $u$ .

The *run* of an automaton  $\mathcal{M}$  on an input  $u \in \Sigma^*$  is the sequence of states  $q_0, q_1, \dots, q_k$ , where  $k = |u|$ ,  $q_0 = q_\iota$ , and for each  $i = 1, 2, \dots, k$ ,  $q_i = \delta(q_{i-1}, u[i])$ . The *run* of  $\mathcal{M}$  on an input  $w \in \Sigma^\omega$  is the infinite sequence of states  $q_0, q_1, q_2, \dots$ , where  $q_0 = q_\iota$ , and for each positive integer  $i$ ,  $q_i = \delta(q_{i-1}, w[i])$ . For an infinite word  $w \in \Sigma^\omega$ , let  $\text{Inf}_{\mathcal{M}}(w)$  denote the set of states of  $\mathcal{M}$  that appear infinitely often in the run of  $\mathcal{M}$  on input  $w$ .

<sup>\*</sup> This research was supported by grant 2016239 from the United States – Israel Binational Science Foundation (BSF).

A state  $q$  of an automaton  $\mathcal{M}$  is *reachable* if and only if there exists a finite word  $u \in \Sigma^*$  such that  $\mathcal{M}(u) = q$ . We may restrict  $\mathcal{M}$  to contain only its reachable states without affecting its finite or infinite runs.

For any automaton  $\mathcal{M} = \langle \Sigma, Q, q_i, \delta \rangle$  we may construct a related directed graph  $G(\mathcal{M}) = (V, E)$  as follows. The set  $V$  of vertices is just the set of states  $Q$ , and there is a directed edge  $(q_1, q_2) \in E$  if and only if for some symbol  $\sigma \in \Sigma$  we have  $\delta(q_1, \sigma) = q_2$ . There are some differences in the terminology related to strong connectivity between graph theory and omega automata, which we resolve as follows.

In graph theory, a *path* of length  $k$  from  $u$  to  $v$  in a directed graph  $(V, E)$  is a finite sequence of vertices  $v_0, v_1, \dots, v_k$  such that  $u = v_0$ ,  $v = v_k$  and for each  $i$  with  $0 < i \leq k$ ,  $(v_{i-1}, v_i) \in E$ . Thus, for every vertex  $v$ , there is a path of length 0 from  $v$  to  $v$ . A set of vertices  $S$  is *strongly connected* if and only if for all  $u, v \in S$ , there is a path of some nonnegative length from  $u$  to  $v$ . Thus, for every vertex  $v$ , the singleton set  $\{v\}$  is a strongly connected set of vertices. A *strongly connected component* of a directed graph is a maximal strongly connected set of vertices. There is a linear time algorithm to find the set of strong components of a directed graph [3].

In the theory of omega automata, a *strongly connected component* (SCC) of  $A$  is a nonempty set  $C \subseteq Q$  of states such that for any  $q_1, q_2 \in C$ , there exists a nonempty word  $u$  such that  $\delta(q_1, u) = q_2$ . Note that a SCC of  $A$  need not be maximal, and that a single state  $q$  of  $A$  is not a SCC of  $A$  unless for some symbol  $\sigma \in \Sigma$  we have  $\delta(q, \sigma) = q$ .

In this paper, we use the terminology *SCC* and *maximal SCC* to refer to the definitions from the theory of omega automata, and the terminology *graph-theoretic strongly connected components* to refer to the definitions from graph theory. Additionally, we use the term *trivial strong component* to refer to a graph-theoretic strongly connected component that is a singleton vertex  $\{v\}$  such that there is no edge  $(v, v)$ . Then if  $\mathcal{M}$  is an automaton, the maximal SCCs of  $\mathcal{M}$  are the graph-theoretic strongly connected components of  $G(\mathcal{M})$  with the exception of the trivial strong components. We also have the following.

**Lemma 1.** *For any automaton  $\mathcal{M}$  and any  $w \in \Sigma^\omega$ ,  $\text{Inf}_{\mathcal{M}}(w)$  is a SCC of  $\mathcal{M}$ .*

*The Product of Two Automata.* Suppose  $\mathcal{M}_1$  and  $\mathcal{M}_2$  are automata with the same alphabet  $\Sigma$ , where for  $i = 1, 2$ ,  $\mathcal{M}_i = \langle \Sigma, Q_i, (q_i)_i, \delta_i \rangle$ . Their product automaton, denoted  $\mathcal{M}_1 \times \mathcal{M}_2$ , is the deterministic automaton  $\mathcal{M} = \langle \Sigma, Q, q_i, \delta \rangle$  such that  $Q = Q_1 \times Q_2$ , the set of ordered pairs of states of  $\mathcal{M}_1$  and  $\mathcal{M}_2$ ,  $q_i = ((q_i)_1, (q_i)_2)$ , the pair of initial states of the two automata, and for all  $(q_1, q_2) \in Q$  and  $\sigma \in \Sigma$ ,  $\delta((q_1, q_2), \sigma) = (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma))$ . For  $i = 1, 2$ , let  $\pi_i$  be projection onto the  $i$ th coordinate, so that for a subset  $S$  of  $Q$ ,  $\pi_1(S) = \{q_1 \mid \exists q_2 (q_1, q_2) \in S\}$ , and analogously for  $\pi_2$ .

## 1.2 Acceptors

If  $\mathcal{M} = \langle \Sigma, Q, q_i, \delta \rangle$  is an automaton, we may augment it with an acceptance condition  $\alpha$  to get a *complete deterministic acceptor*  $\mathcal{A} = \langle \Sigma, Q, q_i, \delta, \alpha \rangle$ , which is

a machine that accepts some words and rejects others. In this paper, an *acceptor* will mean a complete deterministic acceptor. An acceptor accepts a word if the run on that word is accepting, as defined below for the types of acceptors we consider.

For finite words the acceptance condition is a set  $F \subseteq Q$  and the run on a word  $v \in \Sigma^*$  is accepting iff it ends in an accepting state, that is,  $\mathcal{M}(v) \in F$ . We use  $\text{DFA}$  to denote the class of acceptors of finite words, and  $\text{DFA}$  for the languages they accept, which is the class of regular languages.

For  $\omega$ -words, there are various acceptance conditions in the literature; we consider three of them: Büchi, coBüchi, and parity. The Büchi and coBüchi acceptance conditions are also specified by a set  $F \subseteq Q$ . The run of a Büchi acceptor on an input word  $w \in \Sigma^\omega$  is accepting iff it visits at least one state in  $F$  infinitely often, that is,  $\text{Inf}_{\mathcal{M}}(w) \cap F \neq \emptyset$ . The run of a coBüchi acceptor on an input word  $w \in \Sigma^\omega$  is accepting iff it visits  $F$  only finitely many times, that is,  $\text{Inf}_{\mathcal{M}}(w) \cap F = \emptyset$ .

A parity acceptance condition is a map  $\kappa : Q \rightarrow \mathbb{N}$  assigning to each state a natural number termed a color (or priority). We extend  $\kappa$  to sets of states in the natural way, that is, for  $S \subseteq Q$ ,  $\kappa(S) = \{\kappa(q) \mid q \in S\}$ . For a parity acceptor  $\mathcal{P}$  and an  $\omega$ -word  $w$ , we denote by  $\mathcal{P}(w)$  the minimum color of all states visited infinitely often by  $\mathcal{P}$  on input  $w$ , that is,

$$\mathcal{P}(w) = \min(\kappa(\text{Inf}_{\mathcal{M}}(w))).$$

The run of a parity acceptor  $\mathcal{P}$  on an input word  $w \in \Sigma^\omega$  is accepting iff the minimum color visited infinitely often is odd, that is,  $\mathcal{P}(w)$  is odd.

We use  $\llbracket \mathcal{A} \rrbracket$  to denote the set of words accepted by a given acceptor  $\mathcal{A}$ . Two acceptors  $\mathcal{A}_1$  and  $\mathcal{A}_2$  are *equivalent* iff  $\llbracket \mathcal{A}_1 \rrbracket = \llbracket \mathcal{A}_2 \rrbracket$ . We use DBA, DCA, and DPA for the classes of (deterministic) Büchi, coBüchi, and parity acceptors. We use  $\text{DBA}$ ,  $\text{DCA}$ , and  $\text{DPA}$  for the classes of languages they recognize.  $\text{DPA}$  is the full class of regular  $\omega$ -languages, while  $\text{DBA}$  and  $\text{DCA}$  are proper subclasses.

We observe the following facts about DBAs, DCAs and DPAs.

**Claim 1.** *Let  $\mathcal{B} = \mathcal{C} = \langle \Sigma, Q, q_i, \delta, F \rangle$ , where  $\mathcal{B}$  is a DBA and  $\mathcal{C}$  is a DCA. Then the languages recognized by  $\mathcal{B}$  and  $\mathcal{C}$  are complements of each other, that is,  $\llbracket \mathcal{B} \rrbracket = \Sigma^\omega \setminus \llbracket \mathcal{C} \rrbracket$ .*

**Claim 2.** *Let the DBA  $\mathcal{B} = \langle \Sigma, Q, q_i, \delta, F \rangle$ . Define the coloring  $\kappa_{\mathcal{B}}(q) = 1$  for all  $q \in F$  and  $\kappa_{\mathcal{B}} = 2$  for all  $q \in (Q \setminus F)$ . Define the DPA  $\mathcal{P} = \langle \Sigma, Q, q_i, \delta, \kappa_{\mathcal{B}} \rangle$ . Then  $\mathcal{B}$  and  $\mathcal{P}$  accept the same language, that is,  $\llbracket \mathcal{B} \rrbracket = \llbracket \mathcal{P} \rrbracket$ .*

Analogously, for DCAs we have the following.

**Claim 3.** *Let the DCA  $\mathcal{B} = \langle \Sigma, Q, q_i, \delta, F \rangle$ . Define the coloring  $\kappa_{\mathcal{C}}(q) = 0$  for all  $q \in F$  and  $\kappa_{\mathcal{C}} = 1$  for all  $q \in (Q \setminus F)$ . Define the DPA  $\mathcal{P} = \langle \Sigma, Q, q_i, \delta, \kappa_{\mathcal{C}} \rangle$ . Then  $\mathcal{C}$  and  $\mathcal{P}$  accept the same language, that is,  $\llbracket \mathcal{C} \rrbracket = \llbracket \mathcal{P} \rrbracket$ .*

### 1.3 Right congruences

An equivalence relation  $\sim$  on  $\Sigma^*$  is a *right congruence* if  $x \sim y$  implies  $xv \sim yv$  for every  $x, y, v \in \Sigma^*$ . The *index* of  $\sim$ , denoted  $|\sim|$  is the number of equivalence classes of  $\sim$ .

Given an automaton  $\mathcal{M} = \langle \Sigma, Q, q_i, \delta \rangle$ , we can associate with it a right congruence as follows:  $x \sim_{\mathcal{M}} y$  iff  $\mathcal{M}$  reaches the same state when reading  $x$  or  $y$ , that is,  $\mathcal{M}(x) = \mathcal{M}(y)$ . If all the states of  $\mathcal{M}$  are reachable then the index of  $\sim_{\mathcal{M}}$  is exactly the number of states of  $\mathcal{M}$ .

Given a language  $L \subseteq \Sigma^*$ , its *canonical right congruence*  $\sim_L$  is defined as follows:  $x \sim_L y$  iff  $\forall z \in \Sigma^*$  we have  $xz \in L \iff yz \in L$ . For a word  $v \in \Sigma^*$ , the notation  $[v]$  is used for the equivalence class of  $\sim$  in which  $v$  resides.

With a right congruence  $\sim$  of finite index one can naturally associate an automaton  $\mathcal{M}_{\sim} = \langle \Sigma, Q, q_i, \delta \rangle$  as follows. The set of states  $Q$  consists of the equivalence classes of  $\sim$ . The initial state  $q_i$  is the equivalence class  $[\epsilon]$ . The transition function  $\delta$  is defined by  $\delta([u], a) = [ua]$ .

The Myhill-Nerode Theorem states that a language  $L \subseteq \Sigma^*$  is regular iff  $\sim_L$  is of finite index. Moreover, if  $L$  is accepted by a DFA  $\mathcal{A}$  with automaton  $\mathcal{M}$ , then  $\sim_{\mathcal{M}}$  refines  $\sim_L$ . Finally, the index of  $\sim_L$  gives the number of states of the minimal DFA for  $L$ .

For an  $\omega$ -language  $L \subseteq \Sigma^\omega$ , the right congruence  $\sim_L$  is defined analogously, by quantifying over  $\omega$ -words. That is,  $x \sim_L y$  iff  $\forall z \in \Sigma^\omega$  we have  $xz \in L \iff yz \in L$ .

For a regular  $\omega$ -language  $L$ , the right congruence relation  $\sim_L$  is always of finite index, and we may define the *right congruence automaton* for  $L$  to be the automaton  $\mathcal{M}_{\sim_L}$ . However, unlike in the case of the Myhill-Nerode Theorem, the right congruence automaton for  $L$  may not be sufficiently informative to support an acceptor for  $L$ . As an example consider the language  $L = (a + b)^*(bba)^\omega$ . We have that  $\sim_L$  consists of just one equivalence class, since for any  $x \in \Sigma^*$  and  $w \in \Sigma^\omega$  we have that  $xw \in L$  iff  $w$  has  $(bba)^\omega$  as a suffix. Clearly, a DPA recognizing  $L$  needs more than a single state.

The classes  $\mathbb{IB}$ ,  $\mathbb{IC}$ ,  $\mathbb{IP}$  of omega languages are defined as those for which the right congruence automaton will support an acceptor of the corresponding type. A language  $L$  is in  $\mathbb{IB}$  (resp.,  $\mathbb{IC}$ ,  $\mathbb{IP}$ ) if there exists a DBA (resp., DCA, DPA) acceptor  $\mathcal{A}$  such that  $L = \llbracket \mathcal{A} \rrbracket$  and the automaton part of  $\mathcal{A}$  is isomorphic to the right congruence automaton of  $L$ . These classes are more expressive than one might conjecture; it was shown in [1] that in every class of the infinite Wagner hierarchy [4] there are languages in  $\mathbb{IP}$ .

## 2 Inclusion and equivalence for DPAs

The *inclusion problem for DPAs* is the following. Given as input two DPAs  $\mathcal{P}_1$  and  $\mathcal{P}_2$  over the same alphabet, determine whether the language accepted by  $\mathcal{P}_1$  is a subset of the language accepted by  $\mathcal{P}_2$ , that is, whether  $\llbracket \mathcal{P}_1 \rrbracket \subseteq \llbracket \mathcal{P}_2 \rrbracket$ . If so, the answer should be “yes”; if not, the answer should be “no” and a *witness*, that is, an ultimately periodic  $\omega$ -word  $u(v)^\omega$  accepted by  $\mathcal{P}_1$  but rejected by  $\mathcal{P}_2$ .

The *equivalence problem for DPAs* is similar: the input is two DPAs  $\mathcal{P}_1$  and  $\mathcal{P}_2$  over the same alphabet, and the problem is to determine whether they are equivalent, that is, whether  $\llbracket \mathcal{P}_1 \rrbracket = \llbracket \mathcal{P}_2 \rrbracket$ . If so, the answer should be “yes”; if not, the answer should be “no” and a witness, that is, an ultimately periodic  $\omega$ -word  $u(v)^\omega$  that is accepted by one of the DPAs and rejected by the other.

The inclusion and equivalence problems for DBAs or DCAs are defined analogously, where the inputs are two DBAs or DCAs. By Claim 1, the inclusion and equivalence problems for DCAs are efficiently reducible to those for DBAs, and vice versa. Also, by Claims 2 and 3, the inclusion and equivalence problems for DBAs and DCAs are efficiently reducible to those for DPAs. Note that if we have a procedure to solve the inclusion problem, at most two calls to it will solve the equivalence problem. Thus, we focus on the inclusion problem for DPAs and describe a polynomial time algorithm for it.

Note that while polynomial algorithm for testing inclusion of DFAs can be obtained using polynomial algorithms for complementation, intersection and emptiness (since for any two languages  $L_1 \subseteq L_2$  if and only if  $L_1 \cap \overline{L_2} = \emptyset$ ) a similar approach cannot be taken in the case of DPAs since while complementation and emptiness can be computed in polynomial time, intersection cannot [2, Theorem 9].

### 2.1 Searching for $w$ with $\mathcal{P}_1(w) = k_1$ and $\mathcal{P}_2(w) = k_2$

We first describe a polynomial time algorithm that takes as input two DPAs over the same alphabet, say for  $i = 1, 2$ ,  $\mathcal{P}_i = \langle \Sigma, Q_i, (q_i)_i, \delta_i, \kappa_i \rangle$  and two nonnegative integers  $k_1$  and  $k_2$ , and answers the question of whether there exists an  $\omega$ -word  $w$  such that for  $i = 1, 2$ ,  $\mathcal{P}_i(w) = k_i$ , that is, the minimum color of the states visited infinitely often on input  $w$  in  $\mathcal{P}_1$  is  $k_1$  and in  $\mathcal{P}_2$  is  $k_2$ . If there is no such  $w$ , the return value will be “no”, but if there is such a  $w$ , the return value will be “yes” and a witness  $u(v)^\omega$  such that  $\mathcal{P}_1(u(v)^\omega) = k_1$  and  $\mathcal{P}_2(u(v)^\omega) = k_2$ .

For  $i = 1, 2$  let  $(\mathcal{M})_i = \langle \Sigma, Q_i, (q_i)_i, \delta_i \rangle$ , the automaton part of  $\mathcal{P}_i$ . We construct the product automaton  $\mathcal{M} = \mathcal{M}_1 \times \mathcal{M}_2$ , which we may assume is restricted to contain only reachable states. Then we construct the related directed graph  $G(\mathcal{M})$ . Next, we construct  $G'$  by removing from  $G(\mathcal{M})$  all vertices  $(q_1, q_2)$  such that  $\kappa_1(q_1) < k_1$  or  $\kappa_2(q_2) < k_2$ .

In  $G'$  we compute the graph-theoretic strongly connected components, which may be done in linear time. We then eliminate any trivial strong components. In the remaining graph-theoretic strong components of  $G'$ , if there is no component  $C$  such that  $\min(\kappa_1(\pi_1(C))) = k_1$  and  $\min(\kappa_2(\pi_2(C))) = k_2$ , then return the answer “no”.

Otherwise, the algorithm computes a witness  $u(v)^\omega$  as follows, and returns the answer “yes” together with this witness. Choose a graph-theoretic strongly connected component  $C$  of  $G'$  such that for  $i = 1, 2$ ,  $\min(\kappa_i(\pi_i(C))) = k_i$ . Because  $C$  is a non-trivial graph theoretic strongly connected component of  $G'$ , there exists a cycle of length at least 1 that visits every vertex of  $G'$  at least once, which may be found in polynomial time. This cycle may be used to find a state  $(q_1, q_2)$  of  $\mathcal{M}$  and a nonempty word  $v$  such that  $\mathcal{M}$  on input  $v$  starting from

state  $(q_1, q_2)$  visits every element of  $C$  and loops back to state  $(q_1, q_2)$ . Because all the states of  $\mathcal{M}$  are reachable, a breadth-first search can be used to find a word  $u$  such that  $\mathcal{M}(u) = (q_1, q_2)$ . Then on input  $u(v)^\omega$ ,  $\text{Inf}_{\mathcal{M}}(u(v)^\omega) = C$  and for  $i = 1, 2$ ,  $\pi_i(C)$  is the set of states visited infinitely often by  $\mathcal{M}_i$  on input  $u(v)^\omega$ , which has minimum color  $k_i$ . That is, for  $i = 1, 2$ ,  $\mathcal{P}_i(u(v)^\omega) = k_i$ , and  $u(v)^\omega$  is a correct witness. Thus, if the algorithm answers “yes” with a witness, the returned values are correct.

To see that the algorithm does not incorrectly answer “no”, we argue as follows. Suppose  $w$  is an  $\omega$ -word such that for  $i = 1, 2$ ,  $\mathcal{P}_i(w) = k_i$ , that is, if  $C_i = \text{Inf}_{\mathcal{M}_i}(w)$  then  $\min(\kappa_i(C_i)) = k_i$ . Clearly, no state in  $C_i$  has a color less than  $k_i$ , so if  $C = \text{Inf}_{\mathcal{M}}(w)$  is the set of states visited infinitely often in  $\mathcal{M}$  on input  $w$ , all the elements of  $C$  will be in  $G'$ . Because  $C$  is a SCC of  $\mathcal{M}$ ,  $C$  is a nontrivial graph theoretic strongly connected set of vertices of  $G'$ . Then  $C$  is contained in a graph theoretic nontrivial (maximal) strong component  $C'$  of  $G'$ , and because there are no vertices  $(q_1, q_2)$  in  $G'$  with  $\kappa_1(q_1) < k_1$  or  $\kappa_2(q_2) < k_2$ , we must have  $\min(\kappa_i(\pi_i(C')) = k_i$ . Thus, the algorithm will find at least one such graph theoretic strong component  $C'$  of  $G'$  and return “yes” and a correct witness. Thus we have the following.

**Theorem 4.** *There is a polynomial time algorithm that takes as input two arbitrary DPAs  $\mathcal{P}_1$  and  $\mathcal{P}_2$  over the same alphabet and two nonnegative integers  $k_1$  and  $k_2$ , and determines whether there exists an  $\omega$ -word  $w$  such that for  $i = 1, 2$ ,  $\mathcal{P}_i(w) = k_i$ , returning the answer “no” if not, and returning the answer “yes” and a witness word  $w = u(v)^\omega$  if so.*

## 2.2 Inclusion and equivalence algorithms

Then, given DPAs  $\mathcal{P}_1$  and  $\mathcal{P}_2$  over the same alphabet, the inclusion question can be answered by asking, for all odd  $k_1$  in the range of  $\kappa_1$  and all even  $k_2$  in the range of  $\kappa_2$ , whether there exists an  $\omega$ -word  $w$  such that  $\mathcal{P}_i(w) = k_i$  for  $i = 1, 2$ . If no such  $w$  exists, then  $L(\mathcal{P}_1) \subseteq L(\mathcal{P}_2)$ , and otherwise, the algorithm of Theorem 4 yields an ultimately periodic  $\omega$ -word  $u(v)^\omega \in \llbracket \mathcal{P}_1 \rrbracket \setminus \llbracket \mathcal{P}_2 \rrbracket$ . Note that the range of  $\kappa_i$  has at most  $|Q_i|$  distinct elements. Using this and the reduction of equivalence to inclusion, we have the following.

**Corollary 1.** *There are polynomial time algorithms for the inclusion and equivalence problems for DPAs.*

From Claims 2 and 3, we have the following.

**Corollary 2.** *There are polynomial time algorithms for the inclusion and equivalence problems for DBAs and DCAs.*

These results also give us a way of testing whether two states of a DPA have the same right congruence class. If  $\mathcal{P} = \langle \Sigma, Q, q_\ell, \delta, \kappa \rangle$  is a DPA and  $q \in Q$  is a state, let  $\mathcal{P}_q$  denote the DPA  $\mathcal{P}$  with the initial state changed from  $q_\ell$  to  $q$ . If  $\mathcal{P}$  is a DPA with states  $q_1$  and  $q_2$ , then these two states have the same right

congruence class iff  $\llbracket \mathcal{P}_{q_1} \rrbracket = \llbracket \mathcal{P}_{q_2} \rrbracket$ , so we may construct  $\mathcal{P}_{q_1}$  and  $\mathcal{P}_{q_2}$  and test them for equivalence using Corollary 1. Note that this test will return a witness  $u(v)^\omega$  if their right congruence classes are not equal.

**Corollary 3.** *There is a polynomial time algorithm to test whether two states of an arbitrary DPA have the same right congruence class, returning a witness  $u(v)^\omega$  if not.*

## References

1. Dana Angluin and Dana Fisman. Regular omega-languages with an informative right congruence. In *GandALF*, volume 277 of *EPTCS*, pages 265–279, 2018.
2. Udi Boker. Why these automata types? In *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Awassa, Ethiopia, 16-21 November 2018*, pages 143–163, 2018.
3. Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.
4. K. W. Wagner. A hierarchy of regular sequence sets. In *4th Symposium on Mathematical Foundations of Computer (MFCS)*, pages 445–449, 1975.