Properties and Complexity of Deterministic State-Partition Automata[☆]

Galina Jirásková[a,1], Tomáš Masopust[b,2,*]

[a] Mathematical Institute, Slovak Academy of Sciences, Grešákova 6, 040 01 Košice, Slovak Republic
[b] Institute of Mathematics, Academy of Sciences of the Czech Republic, Žižkova 22, 616 62 Brno, Czech Republic

Abstract

A deterministic finite automaton accepting a regular language L is a state-partition automaton with respect to a projection P if the state set of the deterministic finite automaton accepting the projected language $P(L)$, obtained by the standard subset construction, forms a partition of the state set of the automaton. In this paper, we study fundamental properties of state-partition automata. We provide a linear algorithm to decide whether an automaton is a state-partition automaton with respect to a given projection, a construction of the minimal state-partition automaton, discuss closure properties of state-partition automata under the standard constructions of deterministic finite automata for regular operations, and show that almost all of them fail to preserve the property of being a state-partition automaton. Finally, we define the notion of state-partition complexity, and prove the tight bound on state-partition complexity of regular languages represented by incomplete deterministic finite automata.

Keywords: Regular languages, finite automata, descriptional complexity, projections, state-partition automata.

1. Introduction

A deterministic finite automaton G accepting a regular language L is a *state-partition automaton* with respect to a projection P if the state set of the deterministic automaton accepting the projected language $P(L)$, obtained by the standard subset construction [5, 24], forms a partition of the state set of the automaton G. This means that, unlike the general case, the projection of a string uniquely specifies the state of the projected automaton. Therefore, all projected strings of a language with the same observation, that is, with the same projections, lead to the same state of the projected automaton. This property immediately implies that the size of the state set of the minimal state-partition automaton accepting the language L is not smaller than the size of the minimal deterministic finite automaton accepting the projected language $P(L)$.

From the practical point of view, state-partition automata are of interest in engineering and computer science, especially in applications where the user, supervisor, or controller has only a partial observation of the whole behavior of a system, which is modeled by a projection. From the theoretical point of view, state-partition automata have found applications as a proof formalism for systems with partial observations. Namely, they have been successfully used to simplify constructions and proofs, and are useful in applications of natural projections to obtain or describe an abstraction of a system. Note that projections are sometimes generalized to so-called *causal reporter maps*, see [22, 25]. We refer the reader to [3, 4, 12, 13] for applications of state-partition automata in supervisory control of discrete-event systems. Note that state-partition automata are related to the Schützenberger covering. More specifically, the construction of a state-partition automaton is close to the Schützenberger construct [16].

A system represented by a state-partition automaton with respect to a projection that describes an abstraction or a partial observation has a projected automaton that is not larger than the original automaton. This is the most

[☆]This is an extended version of the work presented at IFIP Theoretical Computer Science 2012, Amsterdam, The Netherlands [7].
[*]Corresponding author.
Email addresses: jiraskov@saske.sk (Galina Jirásková), masopust@math.cas.cz (Tomáš Masopust)
[1]Research supported by the Slovak Research and Development Agency under contract APVV-0035-10.
[2]Research supported by GAČR grant no. P202/11/P028 and by RVO: 67985840.

important property from the application point of view. Notice that, up to now, there is only one well-known condition ensuring that a projected automaton is smaller than the original automaton, an *observer property*, cf. [21], although there is still no efficient algorithm to compute the projected automaton. The study of state-partition automata is thus a further step to the understanding and characterization of the class of automata useful for practical applications in, e.g., coordination or hierarchical supervisory control of discrete-event systems [1, 9, 10, 11, 18, 19].

In this paper, we discuss fundamental properties of state-partition automata. In Section 3, we provide a linear algorithm to decide whether a deterministic finite automaton is a state-partition automaton with respect to a given projection. In Section 4, we recall the known result proving that every regular language has a state-partition automaton with respect to a given projection. A procedure to construct this automaton is known, see [3]. We repeat the construction here and use it to obtain the minimal state-partition automaton. The last result of this section describes a regular language and two projections with respect to which the language has no state-partition automaton. This negative result indicates that state-partition automata are useful for systems with either a partial observation or abstraction, but not with the combination of both. Then, in Section 5, we study the closure properties of state-partition automata under the standard constructions of deterministic automata for the operations of complement, union, intersection, concatenation, Kleene star, reversal, cyclic shift, and left and right quotients. We show that almost all of them fail to preserve the property of being a state-partition automaton. Only two of the considered operations preserve this property, namely, the construction of a deterministic automaton for the right quotient of two regular languages, and the construction of a deterministic automaton for the complement of regular languages represented by complete deterministic automata. Finally, in the last section, we introduce and study the notion of *state-partition complexity* of regular languages with respect to a projection, defined as the smallest number of states in any state-partition automaton (with respect to the projection) accepting the language. The first result of this section shows that a language represented by a minimal incomplete deterministic automaton with n states has state-partition complexity at most $3n \cdot 2^{n-3}$. The second result then proves the tightness of this bound using a language defined over a three-letter alphabet and a projection on binary strings.

2. Preliminaries and Definitions

In this paper, we assume that the reader is familiar with the basic notions and concepts of formal languages and automata theory, and we refer the reader to [5, 15, 17] for all details and unexplained notions. For a finite non-empty set Σ, called an alphabet, the set Σ^* represents the free monoid generated by Σ. A string over Σ is any element of Σ^*, and the unit of Σ^* is the empty string denoted by ε. A language over Σ is any subset of Σ^*. For a string w in Σ^*, let $|w|$ denote the length of w, and for a symbol a in Σ, let $|w|_a$ denote the number of occurrences of the symbol a in w. If $w = xyz$, for strings x, y, z, w in Σ^*, then x is a prefix of w, and y is a factor of w.

A *deterministic finite automaton* (DFA) is a quintuple $G = (Q, \Sigma, \delta, s, F)$, where Q is a finite non-empty set of states, Σ is an input alphabet, $\delta : Q \times \Sigma \to Q$ is a partial transition function, $s \in Q$ is the initial (or start) state, and $F \subseteq Q$ is the set of final states. Note that we consider *incomplete* deterministic finite automata that are also called *generators* in the literature, cf. [2, 23]. That is why we prefer to use G to denote an incomplete deterministic automaton. The transition function can be naturally extended to the domain $Q \times \Sigma^*$ by induction. The language *accepted* by the automaton G is the set of strings $L(G) = \{w \in \Sigma^* \mid \delta(s, w) \in F\}$. A state q of G is called *reachable* if $q = \delta(s, w)$ for a string w in Σ^*, and it is called *useful*, or *co-reachable*, if $\delta(q, w) \in F$ for a string w.

A *nondeterministic finite automaton* (NFA) is a quintuple $N = (Q, \Sigma, \delta, S, F)$, where Q, Σ, and F are as in a DFA, $S \subseteq Q$ is the set of initial states, and $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ is the nondeterministic transition function that can be extended to the domain $2^Q \times \Sigma^*$ by induction. The language *accepted* by the NFA N is defined as the set $L(N) = \{w \in \Sigma^* \mid \delta(S, w) \cap F \neq \emptyset\}$. Notice that our NFAs may have ε-transitions and multiple initial states. However, ε-transitions and multiple initial states can be eliminated by a standard technique [5].

Two automata are equivalent if they accept the same language. Every NFA $N = (Q, \Sigma, \delta, S, F)$ without ε-transitions can be converted to an equivalent DFA $\det(N) = (2^Q, \Sigma, \delta_d, s_d, F_d)$ by an algorithm known as the "subset construction" [14], where we have

$$\begin{aligned}\delta_d(R,a) &= \delta(R,a) \text{ for each } R \text{ in } 2^Q \text{ and } a \text{ in } \Sigma,\\ s_d &= S, \text{ and}\\ F_d &= \{R \in 2^Q \mid R \cap F \neq \emptyset\}.\end{aligned}$$

We call the deterministic automaton $\det(N)$ the *subset automaton* corresponding to the automaton N. Notice that the state set of the subset automaton is the set of all subsets of Q, even though some of them may be unreachable from the initial state s_d.

Let Σ be an alphabet and $\Sigma_o \subseteq \Sigma$. A homomorphism P from Σ^* to Σ_o^* is called a *(natural) projection* if it is defined by $P(a) = a$ for each a in Σ_o and $P(a) = \varepsilon$ for each a in $\Sigma \setminus \Sigma_o$. The *inverse image* of P is a mapping P^{-1} from Σ_o^* to 2^{Σ^*} defined by $P^{-1}(w) = \{u \in \Sigma^* \mid P(u) = w\}$.

Let $G = (Q, \Sigma, \delta, s, F)$ be a DFA accepting a language L and P be the projection from Σ^* to Σ_o^* with $\Sigma_o \subseteq \Sigma$. From the DFA G, we construct an NFA N_G accepting the language $P(L)$ by replacing all transitions labeled by symbols from $\Sigma \setminus \Sigma_o$ with ε-transitions, and by eliminating these ε-transitions. Then the *projected automaton* for the language $P(L)$ is the deterministic automaton

$$P(G) = (Q', \Sigma_o, \delta', s', F')$$

that forms the reachable part of the subset automaton $\det(N_G)$. Thus, Q' is the set of all states of 2^Q reachable from the initial state s'. Notice that we do not eliminate states from which no final state is reachable. This is due to applications in supervisory control, where this problem is known as the problem of *nonblockingness* [2].

A DFA $G = (Q, \Sigma, \delta, s, F)$ is a *state-partition automaton* (SPA) with respect to a projection P from Σ^* to Σ_o^* with $\Sigma_o \subseteq \Sigma$ if the states of the projected automaton $P(G) = (Q', \Sigma_o, \delta', s', F')$ are pairwise disjoint as sets. Note that if all states of G are reachable, then the state set of the projected automaton $P(G)$ defines a partition of the state set of G.

For an automaton G (deterministic or nondeterministic), let $\mathrm{sc}(G)$ denote the number of states of the automaton G. We immediately have the following result.

Lemma 1. *Let G be a DFA over an alphabet Σ that has no unreachable states. Let P be a projection from Σ^* to Σ_o^* with $\Sigma_o \subseteq \Sigma$. If G is a state-partition automaton with respect to P, then* $\mathrm{sc}(P(G)) \leq \mathrm{sc}(G)$.

Now we define a parallel composition of two incomplete deterministic automata, which is basically the intersection of their languages defined over two different alphabets. Therefore, it is first necessary to unify their alphabets by adding the missing symbols.

For two deterministic finite automata $G_1 = (Q_1, \Sigma_1, \delta_1, s_1, F_1)$ and $G_2 = (Q_2, \Sigma_2, \delta_2, s_2, F_2)$, we define the *parallel composition* of G_1 and G_2, denoted by $G_1 \parallel G_2$, as the reachable part of the DFA $(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (s_1, s_2), F_1 \times F_2)$, where

$$\delta((p,q),a) = \begin{cases} (\delta_1(p,a), \delta_2(q,a)), & \text{if } \delta_1(p,a) \text{ is defined in } G_1 \text{ and}\\ & \quad \delta_2(q,a) \text{ is defined in } G_2;\\ (\delta_1(p,a), q), & \text{if } \delta_1(p,a) \text{ is defined in } G_1 \text{ and } a \notin \Sigma_2;\\ (p, \delta_2(q,a)), & \text{if } a \notin \Sigma_1 \text{ and } \delta_2(q,a) \text{ is defined in } G_2;\\ \text{undefined}, & \text{otherwise.}\end{cases}$$

From the language point of view, it can be shown that

$$L(G_1 \parallel G_2) = P_1^{-1}(L(G_1)) \cap P_2^{-1}(L(G_2)),$$

where P_i is the projection from $(\Sigma_1 \cup \Sigma_2)^*$ to Σ_i^* for $i = 1, 2$.

Let us briefly recall definitions of the operations of reversal, cyclic shift, and left and right quotients for languages over an alphabet Σ. The *reversal of a string* w over Σ is defined by $\varepsilon^R = \varepsilon$ and $(va)^R = av^R$ for a symbol a in Σ and a string v in Σ^*. The *reversal of a language* L is the language $L^R = \{w^R \in \Sigma^* \mid w \in L\}$. The *cyclic shift* of a language L is defined as the language $L^{shift} = \{uv \in \Sigma^* \mid vu \in L\}$. The *left and right quotients* of a language L by a language K are the languages $K \backslash L = \{x \in \Sigma^* \mid \text{there exists } w \in K \text{ such that } wx \in L\}$ and $L/K = \{x \in \Sigma^* \mid \text{there exists } w \in K \text{ such that } xw \in L\}$, respectively. By L^c we denote the complement of a language L, that is, the language $\Sigma^* \setminus L$.

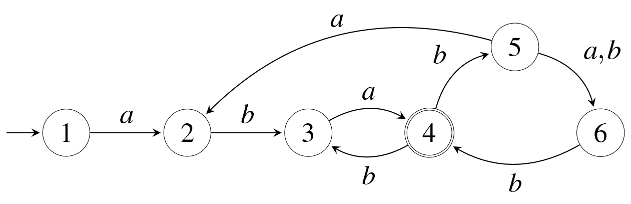

Figure 1: A DFA demonstrating the proof technique; $P : \{a, b\}^* \to \{a\}^*$.

3. A linear algorithm for checking the SPA property

In this section, we present a linear algorithm with respect to the number of states and transitions to check whether, given a DFA G and a projection P, G is a state-partition automaton with respect to projection P.

Theorem 2. *Let G be a DFA over an alphabet Σ with all states reachable, and let P be a projection from $\Sigma^* \to \Sigma_o^*$. There exists a linear algorithm with respect to the number of states plus the number of transitions of the DFA deciding whether DFA G is a state-partition automaton with respect to projection P.*

Proof. Let $G = (Q, \Sigma, \delta, q_0, F)$ be a DFA, and let $P(G) = (Q', \Sigma_o, \delta', Q_0, F')$ denote the projected automaton corresponding to DFA G and projection P. For a in Σ, an a-transition is said to be an ε-transition if $P(a) = \varepsilon$.

By definition, if G is an SPA, then each state of G belongs to exactly one state of the projected automaton $P(G)$, which is the property we need to check. To do this, we will use a slight modification of the standard algorithm to construct the DFA $P(G)$ from DFA G and projection P. The problem with the standard algorithm is that to compare two sets in each step can make the algorithm non-linear, thus we need to take care of these checks. It is explained below.

First, note that by definition we can use an array, A, indexed by states of G to remember a state of $P(G)$ to which the state belongs, that is, for a state q of G, $A[q]$ denotes the state of $P(G)$ containing q. Moreover, notice that all states of any strongly connected component with respect to ε-transitions[3] are contained either all or none in the state of $P(G)$, hence the first step of our algorithm is to eliminate cycles under ε-transitions.

Step 1: Compute the graph of strongly connected components (SCC) with respect to ε-transitions.

In this step, all bs with $P(b) = \varepsilon$ are replaced with ε, and all other transitions are ignored in the computation of SCC, but not removed and still present in the resulting graph, see Figures 1 and 2. As the resulting graph has no cycles under ε-transitions, we can topologically order it with respect to ε-transitions. This ordering is not necessary for the algorithm, it only allow us to speak about maximal and minimal SCC components with respect to a given component. We say that a component is *maximal* with respect to a given component C if it is ε-reachable from C and there is no ε-transition leaving this component, and it is *minimal* with respect to C if C is ε-reachable from it and there is no ε-transition going to it.

In Figure 2, component 3 is maximal with respect to 3, and 2 and 4 are minimal with respect to 3.

Note that the graph of SCC components is an NFA accepting the language $P(L(G))$. We now use this NFA to compute the automaton $P(G)$. We first initialize the array A.

Step 2: Set $A[i] = -$, for $1 \le i \le n$, where $n \le |Q|$ is the number of SCC components.

[3]This basically denotes cycles under ε-transitions.

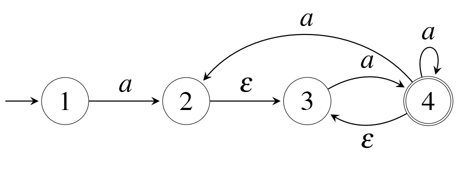

Figure 2: The graph of SCC components with respect to ε-transitions. State 4 is a representative of the SCC component $\{4, 5, 6\}$.

We now need to compute the initial state Q_0 of the automaton $P(G)$.

Step 3: Construct Q_0 as the union of all SCC components that are ε-reachable from the component containing state q_0. For each such a component c, set $A[c] = Q_0$. Moreover, set $MAX[Q_0]$ to be the list of all maximal components reached during this step.

For the NFA of Figure 2, $Q_0 = \{1\}$, $A = [Q_0, -, -, -]$, and $MAX[Q_0] = \{1\}$.

Step 3.1: Use the list $MAX[Q_0]$ to compute the list $MIN[Q_0]$ of minimal components reachable from elements of $MAX[Q_0]$. If there exists a minimal component $c \in MIN[Q_0]$ such that $A[c] \neq Q_0$, then stop because G is not SPA as explained below.

Step 3.2: Put Q_0 to a queue.

The minimal components can be computed in linear time using a copy of the NFA with reversed transitions. In our example, $MIN[Q_0] = \{1\}$. If there is a component $c \in MIN[Q_0]$ with $A[c] = -$, let Q_c denote the state of $P(G)$ containing the component c. Such a state must exist in $P(G)$ because c is reachable in G. However, $MAX[Q_0] \cap Q_c \neq \emptyset$ because c has been reached from a maximal element of Q_0 by reversed ε-transitions. Then the states $Q_0 \neq Q_c$ have nonempty intersection, which violates the SPA property.

Note that, so far, only components of Q_0 and ε-transitions connecting them have been investigated twice.

Now, assume that we have computed states $Q_0, Q_1, \ldots, Q_k$, for some $0 \leq k \leq n-1$, and compute the set $\delta'(Q_i, a)$, for some $0 \leq i \leq k$ and $a \in \Sigma_o$. By definition, if $\delta'(Q_i, a) \cap Q_j \neq \emptyset$, for some $0 \leq j \leq k$, then $\delta'(Q_i, a) = Q_j$. Thus, the next step of the algorithm is to check this property.

Step 4: Pop a state, X, of $P(G)$ from the queue to be examined next, and compute $\delta'(X, a)$, $a \in \Sigma_o$, as follows:

Step 4.1: Investigating the first a-transition from X, going to a component c, set the variable **ReachedComponent** := $A[c]$, the reached state of $P(G)$. Remove or mark each used a-transition so that no a-transition is investigated more than once.

Step 4.2: For each a-transition going from X to a component, say c', check that **ReachedComponent** = $A[c']$. This checks that all a-transitions go from X to the same state of $P(G)$. If this is not satisfied, then stop because G is not an SPA as it violates the definition (see the paragraph above Step 4).

The following step verifies that it goes exactly to that state and not only to a substate.

Step 4.3: If **ReachedComponent** = Q_j, for some previously constructed state Q_j of $P(G)$, check that $\delta'(X, a)$ visits all minimal components of Q_j, i.e., $MIN[Q_j]$. If not, then stop because G is not an SPA: If G is an SPA, then there must be an a-transition from X to each component of $MIN[Q_j]$ (all minimal components of Q_j), otherwise we get two states with nonempty intersection as above in Step 3.

Note that the number of steps is linear with respect to the number of a-transitions even though we need to go through the whole list $MIN[Q_j]$. This is because if G is an SPA, then there is an edge to each component of $MIN[Q_j]$, hence to go through the list $MIN[Q_j]$ only doubles this number.

Step 4.4: If **ReachedComponent** = $-$, let m be a unique marker. In this step, mark every reached unmarked component, c', with m, including those components that are unmarked and ε-reached from c'. Set $A[c'] = \delta'(X, a)$, and update $MAX[\delta'(X, a)]$. Due to the marking, any component of $\delta'(X, a)$ is investigated only once. Now, go through the list of maximal elements of $MAX[\delta'(X, a)]$ and investigate the ε-transitions from them in the inverse direction to compute $MIN[\delta'(X, a)]$. If a new component is reached, then stop because G is not an SPA as explained in Step 3 above.

The complexity of this step is linear: a-transitions are removed/marked as soon as they are used, and ε-transitions are used only twice (once when computing the maximal components, and once when computing the minimal components).

The overall complexity of this algorithm is then $O(n + m)$, where n is the number of states and m the number of transitions.

To finish the demonstration of the algorithm, recall that we have computed the set $Q_0 = \{1\}$ and $A = [Q_0, -, -, -]$. To compute $Q_1 = \delta'(Q_0, a)$, note that **ReachedComponent** $= -$ because $\delta(1, a) = 2$ and $A[2] = -$. As state 3 is ε-reachable from state 2, we have to check that $A[2] = A[3]$, which is satisfied. Hence, we set $A = [Q_0, Q_1, Q_1, -]$, and $MAX[Q_1] = \{3\}$. However, when we compute $MIN[Q_1] = \{2, 4\}$, we obtain component 4 that does not belong to Q_1. Thus, the algorithm stops and rejects. Obviously, the states $\delta(\{1\}, a) = \{2, 3\}$ and $\delta(\{1\}, a^2) = \{3, 4\}$ violate the SPA property. $\square$

4. Minimal State-Partition Automata

The fundamental question is whether every regular language can be accepted by a state-partition automaton with respect to a given projection. If this is the case, can we construct such a state-partition automaton efficiently? The answer to this question is known, and we repeat it in the following theorem. Although a proof has been given in [3], we prefer to recall it here since some fundamental observations play a role later in the paper.

Theorem 3 ([3, 4])**.** *Let P be a projection from Σ^* to Σ_o^* with $\Sigma_o \subseteq \Sigma$. Let L be a language over the alphabet Σ, and let G be a DFA accepting the language L. Then the automaton $P(G) \parallel G$ is a state-partition automaton with respect to the projection P that accepts the language L.*

Proof. Let $G = (Q, \Sigma, \delta, s, F)$ be a DFA accepting the language L, and let $P(G) = (Q', \Sigma_o, \delta', s', F')$ be the corresponding projected automaton. By definition of the parallel composition and the comment below the definition, we have that

$$L(P(G) \parallel G) = P^{-1}(P(L(G))) \cap L(G) = L(G)\,.$$

Hence, the automaton $P(G) \parallel G$ accepts the language L.

Let w be a string over the alphabet Σ_o. Then the state of the projected automaton $P(P(G) \parallel G)$ reached from the initial state by the string w is

$$\left\{(\delta'(s', w), q) \mid q \in \delta(s, P^{-1}(w))\right\}\,.$$

Since $\delta(s, P^{-1}(w)) = \delta'(s', w)$, by definition of the transition function of the automaton $P(G)$, the state reachable from its initial state by the string w in the DFA $P(P(G) \parallel G)$ is, in fact,

$$\{(\delta'(s', w), q) \mid q \in \delta'(s', w)\}\,.$$

It then follows that the states of the projected automaton $P(P(G) \parallel G)$ reachable by two different strings are either the same or disjoint. $\square$

Next we prove that the state-partition automaton constructed from a minimal DFA using the construction of the previous theorem is the minimal state-partition automaton with respect to the number of states. To prove this, we need the notion of isomorphic automata, and the result proved in the following lemma.

Let $G_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $G_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ be two DFAs. Let f be a mapping from Q_1 to Q_2 such that

- $f(\delta_1(q, a)) = \delta_2(f(q), a)$ for each q in Q_1 and a in Σ,
- $f(s_1) = s_2$, and
- $q \in F_1$ if and only if $f(q) \in F_2$.

The mapping f is called a *homomorphism* from G_1 to G_2. If f is a bijection, then it is called an *isomorphism*, and G_1 and G_2 are said to be isomorphic.

The next lemma shows that the parallel composition of automata $P(G)$ and G is isomorphic to G for a state-partition automaton G.

Lemma 4. *Let G be an SPA with respect to a projection P from Σ^* to Σ_o^* in which all states are reachable. Then the DFA $P(G) \parallel G$ is isomorphic to G.*

Proof. Let $G = (Q, \Sigma, \delta, s, F)$ be a state-partition automaton with respect to the projection P, and let the automaton $P(G) = (Q', \Sigma_o, \delta', s', F')$ be the corresponding projected automaton. Let (X, q) be a state of $P(G) \parallel G$ reachable by a string w. By definition, we have $X = \delta(s, P^{-1}(P(w)))$ and $\delta(s, w) = q$. Thus, $q \in X$.

Define a mapping $f : Q' \times Q \to Q$ by $f(X, q) = q$. Then it holds that $\delta(q, a) = \delta(f(X, q), a)$. Since each state q is reachable in G by a string w, $q = \delta(s, w)$, and the state $(\delta(s, P^{-1}(P(w))), \delta(s, w))$ of $P(G) \parallel G$ is mapped to the state q. Hence, f is surjective. On the other hand, let (X, q) and (X', q') be two states of the automaton $P(G) \parallel G$ with $f(X, q) = f(X', q')$. Then $q = q'$, and $q \in X \cap X'$ implies that $X = X'$ since the automaton G is a state-partition automaton. Hence, f is also injective, thus it is bijective. Finally, let δ'' denote the transition function of the automaton $P(G) \parallel G$. By definition of the mapping f, we have $f(\delta''((X, q), a)) = f(\delta'(X, P(a)), \delta(q, a)) = \delta(q, a)$, $f(s', s) = s$, and (X, q) is a final state of the automaton $P(G) \parallel G$ if and only if $f(X, q)$ is a final state of the automaton G. This shows that f is an isomorphism from $P(G) \parallel G$ to G. □

The following result constructs the minimal state-partition automaton for a given regular language and a projection.

Theorem 5. *Let L be a regular language over an alphabet Σ, and let G be the minimal DFA accepting the language L. Let P be a projection from Σ^* to Σ_o^*. Then the DFA $P(G) \parallel G$ is the minimal state-partition automaton with respect to the projection P that accepts the language L.*

Proof. Let $G = (Q, \Sigma, \delta, s, F)$ be the minimal DFA accepting the language L, and let $G_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ be a state-partition automaton with respect to the projection P that also accepts the language L. We may assume that all states of the DFA G_2 are reachable and useful; otherwise, we can remove unreachable and useless states from G_2 and obtain a smaller state-partition automaton.

Define a mapping $f : Q_2 \to Q$ as follows. For a state q in Q_2 that is reachable in the automaton G_2 from the initial state s_2 by a string w, set $f(q) = \delta(s, w)$, that is, $f(q)$ is a state in Q that is reachable in the automaton G from the initial state s by the string w. Notice that f is well-defined since if a state in Q_2 is reached by two different strings u and v, then states $\delta(s, u)$ and $\delta(s, v)$ must be equivalent in the automaton G, and since G is minimal, we must have $\delta(s, u) = \delta(s, v)$.

Next, we have $f(\delta_2(q, a)) = \delta(f(q), a)$ for each state q in Q_2 and symbol a in Σ, $f(s_2) = s$, and $q \in F_2$ if and only if $f(q) \in F$. Hence f is a homomorphism from G_2 to G.

Now, extend the mapping f to a mapping from the state set of the automaton $P(G_2) \parallel G_2$ to the state set of the automaton $P(G) \parallel G$ by setting
$$f(X, q) = (f(X), f(q)) .$$
This is well-defined because if (X, q) is a state of the automaton $P(G_2) \parallel G_2$, then there exists a string w over Σ such that $X = \delta_2(s_2, P^{-1}(P(w)))$ and $q = \delta_2(s_2, w)$, and, therefore,
$$(f(X), f(q)) = (\delta(s, P^{-1}(P(w))), \delta(s, w)) ,$$
which means that $(f(X), f(q))$ is reachable by the string w in the automaton $P(G) \parallel G$.

To prove that f is surjective, consider a state (Y, p) of the automaton $P(G) \parallel G$. Then, there exists a string w such that
$$(Y, p) = (\delta(s, P^{-1}(P(w))),\ \delta(s, w)) .$$
Let $(X, q) = (\delta_2(s_2, P^{-1}(P(w))),\ \delta_2(s_2, w))$. Then (X, q) is a state of the automaton $P(G_2) \parallel G_2$ reachable by the string w, and $f(X, q) = (f(X), f(q)) = (Y, p)$. Since f is surjective and the automaton G_2 is a state-partition automaton with respect to the projection P, we have, using Lemma 4, that
$$\mathrm{sc}(P(G) \parallel G) \le \mathrm{sc}(P(G_2) \parallel G_2) = \mathrm{sc}(G_2) .$$
This completes the proof. □

Corollary 6. *Let L be a regular language over an alphabet Σ, and let P be a projection from Σ^*. Then the minimal state-partition automaton accepting the language L is unique up to isomorphism.*

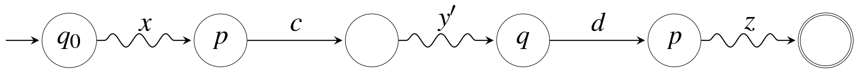

Figure 3: The computation of G on the string $w = xcy'dz$.

Proof. Let G_2 be a minimal state-partition automaton with respect to the projection P that accepts the language L. By Lemma 4, the automaton G_2 is isomorphic to the automaton $P(G_2) \parallel G_2$. Consider the extended mapping f from the automaton $P(G_2) \parallel G_2$ to a minimal state-partition automaton $P(G) \parallel G$, where G is the minimal automaton accepting the language L, constructed in the proof of Theorem 5. It can be proved that the mapping f is a homomorphism. Moreover, since the automaton G_2 is a minimal state-partition automaton, the homomorphism f is injective. Hence, it is an isomorphism. □

It is natural to ask whether an automaton can be a state-partition automaton with respect to more than one projection. This property would be useful in applications, where both an abstraction and a partial observation are combined, cf. [1]. Unfortunately, the following result shows that this does not hold true in general.[4]

Lemma 7. *There exist a language L and projections P and $\tilde{P}$ such that no DFA accepting the language L is a state-partition automaton with respect to both projections P and $\tilde{P}$.*

Proof. Let $\Sigma = \{a, b\}$. Let P and $\tilde{P}$ be projections from Σ^* onto $\{a\}^*$ and $\{b\}^*$, respectively. Consider the language $L = (ab)^*$. Assume that $G = (Q, \Sigma, \delta, s, F)$ is a state-partition automaton for both projections P and $\tilde{P}$ accepting the language L. Notice that the DFA G does not have any loop, that is, no state of G goes to itself on any symbol, because otherwise the automaton G would accept a string that does not belong to the language L.

Let w be a string of the language L of length at least $|Q|$. Then at least one state appears twice in the computation of the automaton G on the string w. Let p be the first such state. Then $w = xyz$, where x is the shortest prefix of w such that the initial state s goes to state p by x, and y is the shortest non-empty factor of w by which p goes to itself. Since the automaton G has no loops, the length of y is at least two. Therefore, $y = cy'd$, where $c, d \in \{a, b\}$. In addition, $c \neq d$ because $xyyz = xcy'dcy'dz$ belongs to the language L. Let q be the state of the automaton G that is reached from the state p on reading the string cy'. Figure 3 illustrates the computation of G on the string w. Since x is the shortest prefix of w that moves G to state p, and y is the shortest non-empty factor of w by which p goes to itself, we have $p \neq q$.

In case $d = b$, we consider the projected automaton

$$P(G) = (Q', \{a\}, \delta', s', F') .$$

Let $X = \delta'(s', P(x))$ and $Y = \delta'(X, P(ay'))$ be two states of the automaton $P(G)$. Then $p \in X$ and $p, q \in Y$. Notice that $X = \delta(s, P^{-1}(P(x)))$. Since $c = a$ and $w \in L$, we have $x = (ab)^k$ for a non-negative integer k. Therefore, $P^{-1}(P(x)) = P^{-1}(a^k)$.

Assume that there exists a string u in $P^{-1}(a^k)$ that moves the automaton G from the initial state s to the state q. Then the string udz is accepted by the automaton G. Since $d = b$, we must have $u = (ab)^{k-1}a$. However, then the state q would be the first state in the computation on the string w that appears at least twice in it, which contradicts the choice of the state p. It follows that $q \notin X$, and, therefore, $X \neq Y$. Hence, the automaton G is not a state-partition automaton with respect to the projection P.

In case $d = a$, we consider the projected automaton

$$\tilde{P}(G) = (Q'', \{b\}, \delta'', s'', F'') ,$$

and two of its states $X = \delta''(s'', \tilde{P}(x))$ and $Y = \delta''(X, \tilde{P}(cy'))$. Then $p \in X$ and $p, q \in Y$. Now, since $c = b$, we have $x = (ab)^k a$ and $\tilde{P}^{-1}(\tilde{P}(x)) = \tilde{P}^{-1}(b^k)$. Assume that there exists a string v in $\tilde{P}^{-1}(a^k)$ with $\delta(s, v) = q$. Then $vdz \in L$ implies that $v = (ab)^k$, and we obtain a contradiction as above. Thus, we again have that $q \notin X$, so $X \neq Y$. Therefore, the automaton G is not a state-partition automaton with respect to the projection $\tilde{P}$. □

[4]Personal communication with J. Komenda, K. Schmidt, and J. H. van Schuppen.

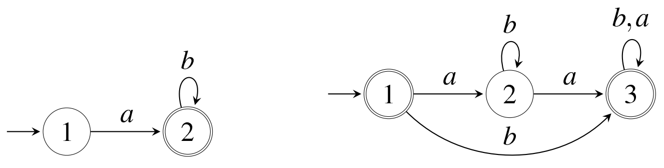

Figure 4: SPA G (left), and DFA G^c for the complement of the language $L(G)$ (right); projection $P : \{a,b\}^* \to \{a\}^*$.

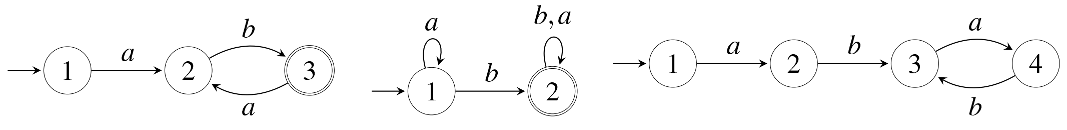

Figure 5: SPAs G_1 (left) and G_2 (middle), and their cross-product $G_1 \times G_2$ (right); projection $P : \{a,b\}^* \to \{a\}^*$.

5. Closure Properties

Since every regular language has a state-partition automaton with respect to a given projection, the class of languages accepted by state-partition automata is closed under all regular operations. In the following, we consider the closure properties of state-partition automata under the standard *constructions* of deterministic automata for regular operations as described in the literature [5, 17, 20, 24]. Hence, we investigate the following question: Given state-partition automata with respect to a projection, is the deterministic automaton resulting from the standard construction for a regular operation a state-partition automaton with respect to the same projection?

We prove that almost all standard constructions, except for the complement of complete state-partition automata and right quotient, fail to preserve the property of being a state-partition automaton.

Theorem 8. *State-partition automata are not closed under the operations of complement, intersection, union, concatenation, star, reversal, cyclic shift, and left quotient.*

Proof. We briefly recall the standard construction of a deterministic automaton for each operation under consideration. Let us emphasize that we do not minimize the resulting deterministic automata.

Complement: To get a deterministic automaton for complement from a possibly incomplete DFA G, add the dead state, if necessary, and interchange the final and non-final states. We prove that state-partition automata are not closed under this operation.

Consider the two-state DFA G in Figure 4 (left). The DFA accepts the language ab^*. Let P be the projection from $\{a,b\}^*$ to $\{a\}^*$. Then G is a state-partition automaton with respect to the projection P since the projected automaton $P(G)$ is deterministic. However, the complement of G, the DFA G^c shown in Figure 4 (right), is not a state-partition automaton with respect to the projection P because we have to add the dead state, 3, which then appears in two different reachable sets of the projected automaton $P(G^c)$, namely, in $\{1,3\}$ reached by ε and in $\{2,3\}$ reached by a. However, as the next theorem shows, the resulting DFA is a state-partition automaton if the given DFA is complete.

Intersection and union: To get the deterministic automaton for intersection and union, we apply the standard cross-product construction.

Consider two automata G_1 and G_2 shown in Figure 5, and their cross-product automaton $G_1 \times G_2$ depicted in Figure 5. In the case of intersection, the only final state is state 3, while in the case of union, the final states are states 3 and 4. Let P be the projection from $\{a,b\}^*$ to $\{a\}^*$. Both G_1 and G_2 are state-partition automata with respect to the projection P. However, the automaton $G_1 \times G_2$ is not since the sets $\{2,3\}$ and $\{3,4\}$ are reachable in the projected automaton $P(G_1 \times G_2)$ by strings a and aa, respectively.

Concatenation: Recall that an NFA for concatenation of two DFAs G_1 and G_2 is obtained from G_1 and G_2 by adding ε-transitions from final states of G_1 to the initial state of G_2, and by setting the initial state to be the initial state of G_1, and final states to be final states of G_2. The corresponding subset automaton restricted to its reachable states provides the resulting DFA for concatenation.

Now, let G be the DFA shown in Figure 6 (left). Let P be the projection from $\{a,b\}^*$ to $\{b\}^*$. The projected automaton $P(G)$ is a one-state automaton and, therefore, the DFA G is a state-partition automaton with respect to the

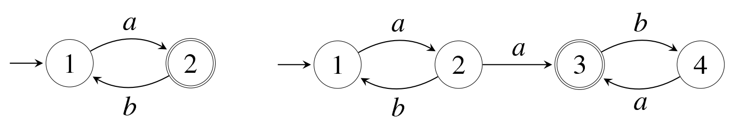

Figure 6: SPA G (left) and DFA $G \cdot G$ for concatenation of the languages $L(G) \cdot L(G)$ (right); projection $P : \{a,b\}^* \to \{b\}^*$.

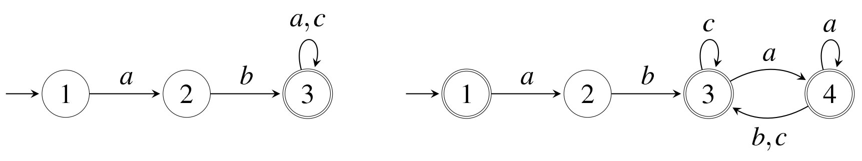

Figure 7: SPA G (left), and DFA G^* for the star of the language $L(G)$ (right); projection $P : \{a,b,c\}^* \to \{a,b\}^*$.

projection P. The DFA $G \cdot G$ for concatenation is depicted in Figure 6 (right), and states $\{1,2,3\}$ and $\{1,2,3,4\}$ are reachable in the projected automaton $P(G \cdot G)$ by strings ε and b, respectively. Hence, the DFA $G \cdot G$ for concatenation is not a state-partition automaton for the projection P.

Star: To construct an NFA for star of a DFA G, add a new initial and final state and ε-transitions from all final states, including the new one, to the original initial state of the automaton G. The subset construction results in a DFA for star.

Consider the DFA G in Figure 7 (left), and the projection P from $\{a,b,c\}^*$ to $\{a,b\}^*$. The automaton G is a state-partition automaton with respect to the projection P since the projected automaton $P(G)$ is deterministic. However, the deterministic automaton G^* for star, shown in Figure 7 (right), is not a state-partition automaton with respect to the projection P because the sets $\{3\}$ and $\{3,4\}$ are reachable in the projected automaton $P(G^*)$ by strings ab and aba, respectively.

Reversal: We can get an NFA for reversal from a DFA G by swapping the roles of initial and final states, and by reversing all transitions. After the application of the subset construction, we obtain a DFA for reversal.

Consider the DFA G in Figure 8 (left), and the projection P from $\{a,b,c\}^*$ to $\{a,c\}^*$. The DFA G is a state-partition automaton with respect to P since the states of the projected automaton $P(G)$ are $\{2,3\}$ and $\{1\}$. On the other hand, the DFA G^R in Figure 8 (right) is not a state-partition automaton with respect to the projection P because the sets $\{2\}$ and $\{2,3\}$ are reachable in the projected automaton $P(G^R)$ by strings a and ac, respectively.

Cyclic shift: For the construction of an NFA for cyclic shift, we refer to [8]. Figure 9 (middle) shows an NFA for the cyclic shift of the language accepted by the DFA G of Figure 9 (left). Let P be the projection from $\{a,b\}^*$ to $\{b\}^*$. Then G is a state-partition automaton with respect to the projection P since the projected automaton $P(G)$ has just one state $\{1,2\}$. However, the automaton G^{shift} in Figure 9 (right) is not a state-partition automaton with respect to the projection P since states $\{1,2,3\}$ and $\{2,3,4,5,6,7,8\}$ are reachable by strings ε and b, respectively.

Left quotient: Construct a DFA for left quotient by a string w from a DFA G by making the state reached after reading the string w initial.

Consider the DFA G shown in Figure 10 (left) and the projection P from $\{a,b\}^*$ to $\{b\}^*$. The automaton G is a state-partition automaton with respect to the projection P as in the case of cyclic shift. The automaton $a\backslash G$ for the left quotient by the string a is shown in Figure 10 (right). It is not a state-partition automaton with respect to the projection P since the sets $\{2\}$ and $\{1,2\}$ are reachable in the projected automaton by strings ε and b, respectively. □

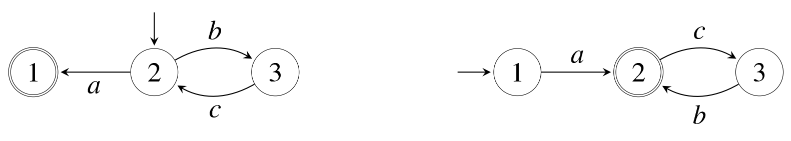

Figure 8: SPA G (left), and DFA G^R for the reversal of the language $L(G)$ (right); projection $P : \{a,b,c\}^* \to \{a,c\}^*$.

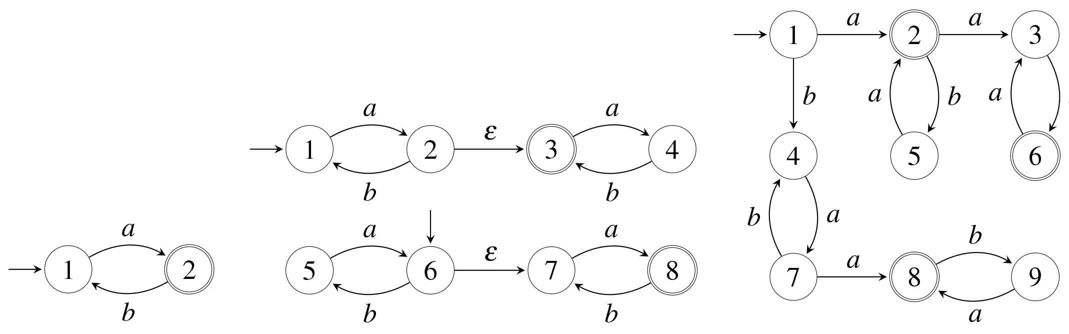

Figure 9: SPA G (left), NFA for shift($L(G)$) (middle), and DFA G^{shift} (right); projection $P : \{a,b\}^* \to \{b\}^*$.

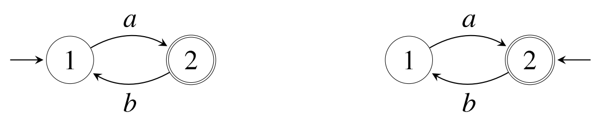

Figure 10: SPA G (left) and DFA $a\backslash G$ for the left quotient by the string a (right); projection $P : \{a,b\}^* \to \{b\}^*$.

The following theorem demonstrates that if the structure of the automaton is not changed after an operation, then the automaton remains state-partition with respect to the same projection.

Theorem 9. *State-partition automata are closed under the operations of right quotient and complement of complete state-partition automata.*

Proof. Let G be a complete state-partition automaton. Construct a deterministic automaton G^c for the complement of $L(G)$ from the DFA G by interchanging final and non-final states. The result now follows from the fact that the states of the projected automaton $P(G^c)$ are the same as the states of the projected automaton $P(G)$ since the structure of the automaton G^c is the same as the structure of the automaton G.

Now, consider the right quotient of a language $L(G)$ by a language K; here, the DFA G may be incomplete. Construct an automaton for the right quotient $L(G)/K$ from the automaton G by replacing the set of final states with the set of states of G from which a string of the language K is accepted. Again, the structure of the automaton remains the same; we only change the set of final states. □

6. State-Partition Complexity

Let L be a regular language over an alphabet Σ, and let P be a projection from Σ^* to Σ_o^*. We define the *state-partition complexity* of the language L, denoted by $\mathrm{spc}(L)$, as the smallest number of states in any automaton accepting the language L that is a state-partition automaton with respect to the projection P. By Theorem 5, the state-partition complexity of the language L is the number of states of the DFA $P(G) \parallel G$, where G is the minimal incomplete DFA accepting the language L.

Now, we give the upper bound on the state-partition complexity of regular languages, and prove that this bound is tight.

Theorem 10. *Let L be a language over an alphabet Σ accepted by the minimal incomplete DFA G with n states. Let P be a projection from Σ^* to Σ_o^*. Then* $\mathrm{spc}(L) \leq 3n \cdot 2^{n-3}$.

Proof. Let $G = (Q, \Sigma, \delta, s, F)$ be the minimal incomplete DFA accepting the language L, and let the automaton $P(G) = (Q', \Sigma_o, \delta', s', F')$ be the corresponding projected automaton. Consider the automaton $P(G) \parallel G$. By definition of the transition function of the DFA $P(G) \parallel G$, a pair (X, q) in $2^Q \times Q$ is a state of the automaton $P(G) \parallel G$ if and only if $q \in X$ and X is a state of the automaton $P(G)$; see also proofs of Theorems 3 and 5. This implies that the sum

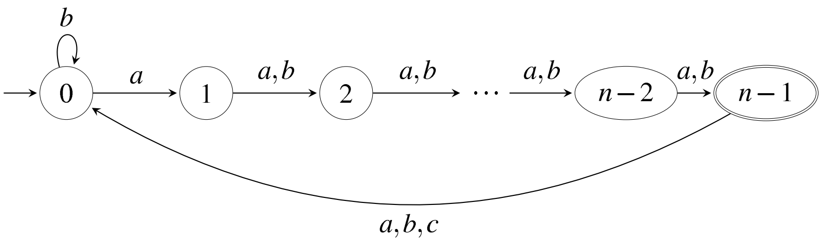

Figure 11: The minimal incomplete DFA G meeting the upper bound $3n \cdot 2^{n-3}$.

of the cardinalities of all states of the automaton $P(G)$ gives the upper bound on the state-partition complexity of the language L, that is,

$$\mathrm{spc}(L) \leq \sum_{X \in Q'} |X| \leq \sum_{i=0}^{n} \binom{n}{i} i \,, \tag{1}$$

where $\sum_{i=0}^{n} \binom{n}{i} i$ counts the cardinalities of all i-element subsets. However, it is known that not all subsets of the state set Q are states of the automaton $P(G)$. It was shown in [6], see also [21], that $|Q'| \leq 3 \cdot 2^{n-2} - 1$ if the projection is not an identity mapping, and that this bound is met only if the automaton has only one transition labeled by a symbol removed by the projection. Let $(q, a, p) \in Q \times \Sigma \times Q$ be a transition of the automaton G such that $P(a) = \varepsilon$. Then, every state of the automaton $P(G)$ containing state q must also contain state p. Therefore, we have to subtract the number of subsets containing state q but not state p from the number of subsets included in the last element of (1). There are 2^{n-2} such subsets because we are interested in all subsets $Y \cup \{q\}$, where Y is a subset of $Q \setminus \{p, q\}$. After the multiplication of these subsets by their cardinalities and subtracting the resulting value from the right-hand side in (1), we get the following upper bound

$$\mathrm{spc}(L) \leq \sum_{i=0}^{n} \binom{n}{i} i - \sum_{i=0}^{n-2} \binom{n-2}{i} (i+1).$$

Using the equality

$$\sum_{i=0}^{k} \binom{k}{i} i = k 2^{k-1} \,,$$

we have

$$\mathrm{spc}(L) \leq n \cdot 2^{n-1} - ((n-2) \cdot 2^{n-3} + 2^{n-2}) = 3n \cdot 2^{n-3} \,,$$

which proves the theorem. □

Finally, we prove that the bound proved in the previous theorem is tight.

Theorem 11. *For every integer $n \geq 3$, there exists a regular language L accepted by the minimal incomplete DFA G with n states such that* $\mathrm{spc}(L) = 3n \cdot 2^{n-3}$.

Proof. Consider the language L accepted by the DFA G depicted in Figure 11 and the projection P from $\{a, b, c\}^*$ to $\{a, b\}^*$. We need to prove that all subsets of the state set $\{0, 1, \ldots, n-1\}$, except for the sets that contain $n-1$ and do not contain 0, are states of the automaton $P(G)$. Notice that if X is reachable in $P(G)$ by a string u over $\{a, b\}$ and $q \in X$, then state q is reachable in the automaton G by a string w in $P^{-1}(u)$. This means that (X, q) is a reachable state in the automaton $P(G) \parallel G$ since $(X, q) = (\delta(s, P^{-1}(P(w)), \ \delta(s, w))$. First, we construct an NFA accepting the language $P(L)$ as shown in Figure 12. Let us show that all subsets of the state set $\{0, 1, \ldots, n-1\}$ containing state 0, as well as all non-empty subsets of the set $\{1, 2, \ldots, n-2\}$ are reachable.

The proof is by induction on the size of subsets. Each set $\{i\}$, where $i \leq n-2$, is reached from $\{0\}$ by the string a^i. Let $2 \leq k \leq n$. Assume that each subset of size $k-1$, satisfying the above mentioned conditions, is reachable. Let $X = \{i_1, i_2, \ldots, i_k\}$, where $0 \leq i_1 < i_2 < \cdots < i_k \leq n-1$, be a subset of size k. Consider two cases:

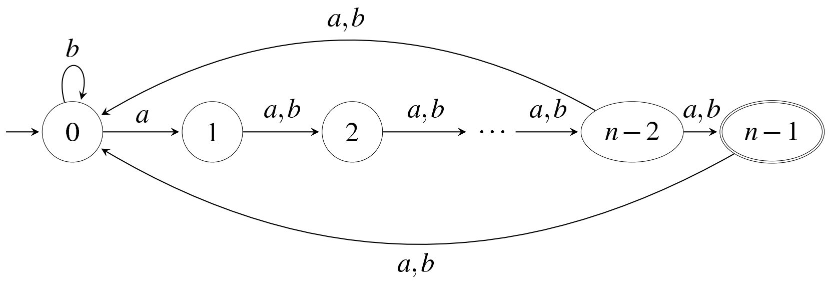

Figure 12: An NFA for language $P(L(G))$, where G is shown in Figure 11.

(i) $i_1 = 0$. Take $Y = \{i_j - i_2 - 1 \mid 3 \leq j \leq k\} \cup \{n - 2\}$. Then Y is of size $k - 1$ and it does not contain state $n - 1$. Therefore, it is reachable by the induction hypothesis. The subset Y goes to X on the string aab^{i_2-1} since we have

$$\begin{aligned} Y &\xrightarrow{a} \{0, n-1\} \cup \{i_j - i_2 \mid 3 \leq j \leq k\} \\ &\xrightarrow{a} \{0, 1\} \cup \{i_j - i_2 + 1 \mid 3 \leq j \leq k\} \\ &\xrightarrow{b^{i_2-1}} X. \end{aligned}$$

(ii) $i_1 \geq 1$. Then $i_k \leq n - 2$. Take $Y = \{0\} \cup \{i_j - i_1 \mid 2 \leq j \leq k\}$. Then the subset Y is of size k and contains state 0. Therefore, it is reachable as shown in case (i). The subset Y goes to X on the string a^{i_1}.

This proves the reachability of all $3 \cdot 2^{n-2} - 1$ subsets discussed in the proof of the previous theorem. Using the arguments of the proof of Theorem 10 then completes the proof. □

7. Conclusions and Discussion

We investigated deterministic state-partition automata with respect to a given projection. The state set of such an automaton is partitioned into disjoint subsets that are reachable in the projected automaton. Using a result from the literature that every regular language has a state-partition automaton with respect to a given projection, we provided the construction of the minimal state-partition automaton to a regular language and a projection. We also described a regular language and two projections such that no automaton accepting this language is a state-partition automaton with respect to both projections.

Next, we studied closure properties of state-partition automata under the standard constructions of deterministic automata for the operations of complement, union, intersection, concatenation, star, reversal, cyclic shift, and left and right quotients. We showed that except for the right quotient and complement of complete deterministic automata, all other constructions fail to preserve the property of being a state-partition automaton.

Finally, we defined the notion of state-partition complexity of a regular language as the smallest number of states of any state-partition automaton with respect to a given projection accepting the language. We proved that the tight bound on state-partition complexity of a language represented by an incomplete deterministic automaton with n states is $3n \cdot 2^{n-3}$. To prove the tightness of this bound, we used a language defined over the ternary alphabet $\{a, b, c\}$ and the projection from $\{a, b, c\}^*$ to $\{a, b\}^*$. Note that it follows from the results of [6] that this bound cannot be reached using a smaller alphabet or a projection to a singleton.

State-partition complexity of regular operations may be investigated in the future. We only know that state-partition complexity of a language and its complement differs by one in the case of complete deterministic automata, and by $3n$ if the automata are incomplete. Defining nondeterministic state-partition automata and investigating their properties may also be of interest.

Acknowledgements.

We wish to thank Jan Komenda, Klaus Schmidt, and Jan H. van Schuppen for a very fruitful discussion on state-partition automata.

References

[1] O. Boutin, J. Komenda, T. Masopust, K. Schmidt, J.H. van Schuppen, Hierarchical control with partial observations: Sufficient conditions, in: Proc. of IEEE Conference on Decision and Control and European Control Conference (CDC-ECC 2011), Orlando, Florida, USA, pp. 1817–1822.

[2] C.G. Cassandras, S. Lafortune, Introduction to discrete event systems, Springer, second edition, 2008.

[3] H. Cho, S.I. Marcus, On supremal languages of classes of sublanguages that arise in supervisor synthesis problems with partial observation, Mathematics of Control, Signals, and Systems 2 (1989) 47–69.

[4] H. Cho, S.I. Marcus, Supremal and maximal sublanguages arising in supervisor synthesis problems with partial observations, Theory of Computing Systems 22 (1989) 177–211.

[5] J.E. Hopcroft, R. Motwani, J.D. Ullman, Introduction to automata theory, languages, and computation, Addison-Wesley, Boston, 2003.

[6] G. Jirásková, T. Masopust, On a structural property in the state complexity of projected regular languages, Theoretical Computer Science 449 (2012) 93–105.

[7] G. Jirásková, T. Masopust, On properties and state complexity of deterministic state-partition automata, in: Prof. of IFIP Theoretical Computer Science (TCS 2012), volume 7604 of *LNCS*, IFIP International Federation for Information Processing, 2012, pp. 164–178.

[8] G. Jirásková, A. Okhotin, State complexity of cyclic shift, RAIRO – Theoretical Informatics and Applications 42 (2008) 335–360.

[9] J. Komenda, T. Masopust, J.H. van Schuppen, Synthesis of controllable and normal sublanguages for discrete-event systems using a coordinator, Systems & Control Letters 60 (2011) 492–502.

[10] J. Komenda, T. Masopust, J.H. van Schuppen, On algorithms and extensions of coordination control of discrete-event systems, in: Proc. of Workshop on Discrete Event Systems (WODES 2012), Guadalajara, Mexico, pp. 245–250. [Online.]
Available at `http://www.ifac-papersonline.net/`.

[11] J. Komenda, T. Masopust, J.H. van Schuppen, Supervisory control synthesis of discrete-event systems using a coordination scheme, Automatica 48 (2012) 247–254.

[12] J. Komenda, J.H. van Schuppen, Supremal normal sublanguages of large distributed discrete-event systems, in: Proc. of International Workshop on Discrete Event Systems (WODES 2004), Reims, France, pp. 73–78.

[13] J. Komenda, J.H. van Schuppen, Modular control of discrete-event systems with coalgebra, IEEE Transactions on Automatic Control 53 (2008) 447–460.

[14] M.O. Rabin, D. Scott, Finite automata and their decision problems, IBM Journal of Research and Development 3 (1959) 114–125.

[15] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, volume 1–3, Springer, 1997.

[16] J. Sakarovitch, A construction on finite automata that has remained hidden, Theoretical Computer Science 204 (1998) 205–231.

[17] A. Salomaa, Formal languages, Academic Press, New York, 1973.

[18] K. Schmidt, C. Breindl, Maximally permissive hierarchical control of decentralized discrete event systems, IEEE Transactions on Automatic Control 56 (2011) 723–737.

[19] K. Schmidt, T. Moor, S. Perk, Nonblocking hierarchical control of decentralized discrete event systems, IEEE Transactions on Automatic Control 53 (2008) 2252–2265.

[20] M. Sipser, Introduction to the theory of computation, PWS Publishing Company, Boston, 1997.

[21] K. Wong, On the complexity of projections of discrete-event systems, in: Proc. of Workshop on Discrete Event Systems (WODES 1998), Cagliari, Italy, pp. 201–206.

[22] K. Wong, W. Wonham, Hierarchical control of discrete-event systems, Discrete Event Dynamic Systems: Theory and Applications 6 (1996) 241–273.

[23] W.M. Wonham, Supervisory control of discrete-event systems, 2011. Lecture notes, University of Toronto.

[24] S. Yu, Regular languages, in: Handbook of Formal Languages, volume I, Springer, 1997, pp. 41–110.

[25] H. Zhong, W.M. Wonham, On the consistency of hierarchical supervision in discrete-event systems, IEEE Transactions on Automatic Control 35 (1990) 1125–1134.