International Journal of Foundations of Computer Science
Vol. 19, No. 3 (2008) 581–595

World Scientific
www.worldscientific.com

STATE COMPLEXITY OF UNION AND INTERSECTION OF FINITE LANGUAGES*

YO-SUB HAN*
Intelligence and Interaction Research Center,
Korea Institute of Science and Technology,
P.O.BOX 131, Cheongryang, Seoul, Korea
emmous@kist.re.kr

and

KAI SALOMAA†
School of Computing, Queen's University,
Kingston, Ontario K7L 3N6, Canada
ksalomaa@cs.queensu.ca

Received 27 September 2007
Accepted 18 January 2008
Communicated by Tero Harju and Juhani Karhumäki

We investigate the state complexity of union and intersection for finite languages. Note that the problem of obtaining the tight bounds for both operations was open. First we compute upper bounds using structural properties of minimal deterministic finite-state automata for finite languages. Then, we show that the upper bounds are tight if we have a variable sized alphabet that can depend on the size of input deterministic finite-state automata. In addition, we prove that the upper bounds are unreachable for any fixed sized alphabet.

Keywords: State complexity; finite-state automata; finite languages.

1. Introduction

Regular languages are one of the most important and well-studied topics in computer science. They are often used in various practical applications such as `vi`, `emacs` and `Perl`. Furthermore, researchers have developed a number of software libraries for manipulating formal language objects with an emphasis on regular languages; examples include Grail [16] and Vaucanson [2].

*A preliminary version of this paper appeared in Proceedings of 11th International Conference Developments in Language Theory, DLT 2007, Lect. Notes Comput. Sci. **4588**, Springer-Verlag, 2007, pp. 217–228.

†Han was supported by the KIST research grant and the KRCF research grant.
†Salomaa was supported by the Natural Sciences and Engineering Research Council of Canada Grant OGP0147224.

The applications and implementations of regular languages motivate the study of descriptional complexity of regular languages. The descriptional complexity of regular languages can be defined in various ways since regular languages can be characterized in different ways. For example, the family of languages accepted by deterministic finite-state automata (DFAs) or by nondeterministic finite-state automata (NFAs) or described by regular expressions consists of exactly the regular languages. Yu and his co-authors [1, 18, 19] regarded the number of states in the complete minimal DFA for L as the complexity of L and studied the state complexity of basic operations on regular languages and finite languages. Similar results where state complexity is defined using incomplete DFAs already appeared in the work by Maslov [13]. Holzer and Kutrib [8, 9] investigated the state complexity of NFAs. Recently, Ellul et al. [5] examined the size of the shortest regular expression for a given regular language. There are many other results on state complexity with different viewpoints [3, 4, 6, 7, 11, 12, 14, 15]. We focus on the measure of Yu [18]: The *state complexity* of a regular language is the number of states of its minimal DFA. The state complexity of an operation on regular languages is a function that associates to the state complexities of the operand languages the worst-case state complexity of the language resulting from the operation. For instance, we say that the state complexity of the intersection of $L(A)$ and $L(B)$ is mn, where A and B are minimal DFAs and the numbers of states in A and B are m and n, respectively. This means that mn is the worst-case number of states of the minimal DFA for $L(A) \cap L(B)$.

Yu et al. [19] gave a systematic study of state complexity of regular language operations. Câmpeanu et al. [1] investigated the state complexity of finite languages. The known results [1, 19] are summarized in Table 1. In the figure, $m, n \geq 1$ denote the state complexity of L_1 and L_2, respectively.

Table 1. The state complexity of basic operations on finite languages and regular languages. Note that $\diamond$ refers to results using a two-character alphabet.

operation	finite languages	regular languages
$L_1 \cup L_2$	$O(mn)$	mn
$L_1 \cap L_2$	$O(mn)$	mn
$\Sigma^* \setminus L_1$	m	m
$L_1 \cdot L_2$	$(m-n+3)2^{n-2}-1^{\diamond}$	$(2m-1)2^{n-1}$
L_1^*	$2^{m-3}+2^{m-4}$, for $m \geq 4^{\diamond}$	$2^{m-1}+2^{m-2}$
L_1^R	$3 \cdot 2^{p-1}-1$ if $m=2p$ 2^p-1 if $m=2p-1$	2^m

All complexity bounds, except for union and intersection of finite languages, in Table 1 are tight; namely, there exist worst-case examples that reach the given bounds. For union and intersection, clearly mn is an upper bound since finite languages are a proper subfamily of regular languages. We also note that Yu [18]

briefly mentioned a rough upper bound $mn - (m + n - 2)$ for both operations. Therefore, it is natural to investigate the tight bounds for union and intersection of finite languages.

To conclude the introduction, we summarize the contents of the following sections. We define some basic notions in Section 2. In Section 3, we obtain an upper bound $mn-(m+n)$ for the union of two finite languages L_1 and L_2 that is based on the structural properties of the DFAs for L_1 and L_2. Then, we prove that the bound is tight if the alphabet size can depend on m and n. We also examine the intersection of L_1 and L_2 in Section 4 and obtain an upper bound $mn - 3(m + n) + 12$. We again demonstrate that the upper bound is reachable using a variable sized alphabet. Some conclusions and open problems are given in Section 5.

2. Preliminaries

Here we recall some definitions needed in the later sections. For all unexplained notions related to formal languages and finite automata we refer the reader to Hopcroft and Ullman [10] or Yu [17].

In the following Σ always denotes a finite alphabet of characters and Σ^* is the set of all strings over Σ. The number of characters in Σ is denoted by $|\Sigma|$. A language over Σ is any subset of Σ^*. The symbol $\emptyset$ denotes the empty language and the symbol λ denotes the null string.

A finite-state automaton A (FA) is specified by a tuple $(Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, Σ is an input alphabet, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $s \in Q$ is the start state and $F \subseteq Q$ is a set of final states. The automaton is *deterministic* if δ is a total function $Q \times \Sigma \to Q$. The number of states of an FA A is denoted also by $|A|$.

A string x over Σ is accepted by A if there is a labeled path from s to a final state in F such that this path spells out the string x. Thus, the language $L(A)$ recognized by A is the set of all strings that are spelled out by paths from s to a final state in F.

Assume that A has a transition $\delta(p, a) = q$. In this case, we say that p has an *out-transition* and q has an *in-transition.* Furthermore, p is a *source state* of q and q is a *target state* of p. We say that A is *non-returning* if the start state of A does not have any in-transitions. A state $q \in Q$ is *non-exiting* if all out-transitions from q go to the sink state.

Note that we require a DFA to be *complete;* namely, each state will have $|\Sigma|$ out-transitions. In particular, a DFA accepting a finite language will have a sink state (or dead state). Since all sink states are always equivalent, a minimal DFA for a finite language has a unique sink state. Furthermore, a minimal DFA accepting a finite language has a unique non-exiting final state. These observations are used without further mention in the following.

Given an FA $A = (Q, \Sigma, \delta, s, F)$, we define the *right language* L_q of a state $q \in Q$ to be the set of strings that are spelled out by some path from q to a final state in A. Namely, L_q is the language accepted by the FA obtained from A by changing the start state to q. We say that two states p and q are *equivalent* if $L_p = L_q$.

Let $L \subseteq \Sigma^*$. The right invariant congruence of L, $\equiv_L \subseteq \Sigma^* \times \Sigma^*$, is defined by setting $u \equiv_L v$ if

$$(\forall x \in \Sigma^*)\ ux \in L \text{ iff } vx \in L.$$

3. Union of Finite Languages

Given two DFAs A and B for languages L_1 and L_2, we can in the well-known way obtain a DFA for the union of $L(A)$ and $L(B)$ using the Cartesian product construction.

Proposition 1 *Given two DFAs $A = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $B = (Q_2, \Sigma, \delta_2, s_2, F_2)$, let $M_\cup = (Q_1 \times Q_2, \Sigma, \delta, (s_1, s_2), F)$, where for all $p \in Q_1$ and $q \in Q_2$ and $a \in \Sigma$, $\delta((p,q),a) = (\delta(p,a), \delta(q,a))$ and $F = \{(p, f_2) \mid p \in Q_1 \text{ and } f_2 \in F_2\} \cup \{(f_1, q) \mid f_1 \in F_1 \text{ and } q \in Q_2\}$. Then, $L(M_\cup) = L(A) \cup L(B)$ and $M_\cup$ is deterministic.*

In order to improve the upper bound for the state complexity of union of finite languages, a crucial observation is that both A and B must be non-returning. Therefore, as Yu [18] observed, if we apply the Cartesian product for union, any state (s_1, q) where $q \neq s_2$ and any state (p, s_2) where $p \neq s_1$ is not reachable from the start state (s_1, s_2) in $M_\cup$. Thus, the the DFA $M_\cup$ has at least $(m+n)-2$ useless states.

Consider the right language of a state (i,j) in $M_\cup$. We recall the following result from Han et al. [7].

Proposition 2 (Han et al. [7]) *For a state (i,j) in $M_\cup$, the right language $L_{(i,j)}$ of (i,j) is the union of the right language L_i of i (as a state of A) and the right language L_j of j (as a state of B).*

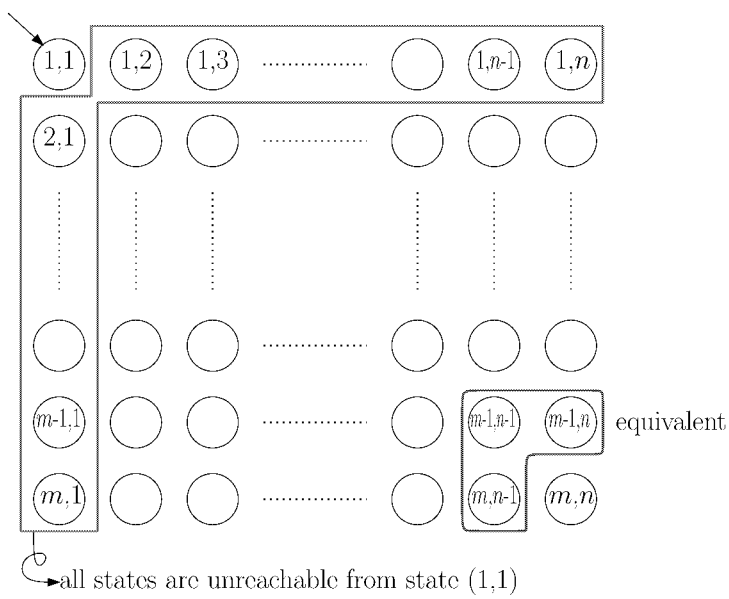

Fig. 1. A DFA constructed for the union of two minimal DFAs of finite languages based on the Cartesian product of states. The figure omits all transitions.

Since A recognizes a finite language, A must have a final state f such that all of f's out-transitions go to the sink state. Let d_1 and d_2 be the sink states of A and B and f_1 and f_2 be the non-exiting final states of A and B, respectively. Then, by Proposition 2, (f_1, f_2), (d_1, f_2) and (f_1, d_2) are equivalent and, thus, can be merged into a single state. Fig. 1 illustrates this step; note that $(f_1, f_2) = (m{-}1, n{-}1)$, $(d_1, f_2) = (m, n{-}1)$ and $(f_1, d_2) = (m{-}1, n)$ in the figure. This shows that we can reduce two more states from $M_\cup$. Therefore, we obtain the following result.

Lemma 1 *Given two minimal DFAs A and B for finite languages, $mn - (m + n)$ states are sufficient for the union of $L(A)$ and $L(B)$, where $m = |A|$ and $n = |B|$.*

We next examine whether or not the upper bound of Lemma 1 can be reached in the worst-case. We first consider the case where the size of the alphabet may depend on the state complexity of the component languages.

Lemma 2 *The upper bound $mn - (m + n)$ for union is reachable if the size of the alphabet can depend on m and n.*

Proof. Let m and n be positive numbers and

$$\Sigma = \{b, c\} \cup \{a_{i,j} \mid 1 \le i \le m-2,\ 1 \le j \le n-2 \text{ and } (i,j) \ne (m-2, n-2)\}.$$

Let $A = (Q_1, \Sigma, \delta_1, p_0, \{p_{m-2}\})$, where $Q_1 = \{p_0, p_1, \ldots, p_{m-1}\}$ and δ_1 is defined as follows:

- $\delta_1(p_i, b) = p_{i+1}$, for $0 \le i \le m-2$.
- $\delta_1(p_0, a_{i,j}) = p_i$, for $1 \le i \le m-2$ and $1 \le j \le n-2$, $(i,j) \ne (m-2, n-2)$.

For all cases not covered above, δ_1 takes the source state to the state p_{m-1}, and consequently p_{m-1} is the sink state of A.

Next, let $B = (Q_2, \Sigma, \delta_2, q_0, \{q_{n-2}\})$, where $Q_2 = \{q_0, q_1, \ldots, q_{n-1}\}$ and δ_2 is defined as follows:

- $\delta_2(q_i, c) = q_{i+1}$, for $0 \le i \le n-2$.
- $\delta_2(q_0, a_{i,j}) = q_j$, for $1 \le j \le n-2$ and $1 \le i \le m-2$, $(i,j) \ne (m-2, n-2)$.

Again, for all cases not covered above, the target state is the sink state q_{n-1}. Fig. 2 illustrates the definition of the DFAs A and B.

Let $L = L(A_1) \cup L(A_2)$. We claim that the minimal DFA for L needs $mn - (m + n)$ states. To prove the claim, it is sufficient to show that there exists a set R consisting of $mn - (m+n)$ strings over Σ that are pairwise inequivalent modulo the right invariant congruence of L, $\equiv_L$.

We choose $R = R_1 \cup R_2 \cup R_3$, where

$R_1 = \{b^i \mid 0 \le i \le m-1\}$.

$R_2 = \{c^j \mid 1 \le j \le n-3\}$. (Note that R_2 does not include c^0, c^{n-2} and c^{n-1}.)

$R_3 = \{a_{i,j} \mid 1 \le i \le m-2 \text{ and } 1 \le j \le n-2 \text{ and } (i,j) \ne (m-2, n-2)\}$.

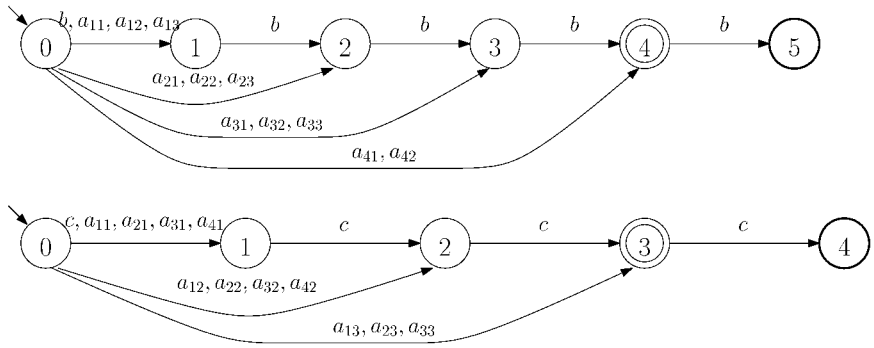

Fig. 2. The DFAs A and B in the case where $m = 6$ and $n = 5$. State 5 of A (above) and state 4 of B (below) are sink states. Except for the b-transition to state 5 in A and the c-transition to state 4 in B, the figure omits all other in-transitions of the sink states.

Any string b^i from R_1 cannot be equivalent with a string c^j from R_2 since $c^j \cdot c^{n-2-j} \in L$ but $b^i \cdot c^{n-2-j} \notin L$. Note that $j \geq 1$ and hence also $b^0 \cdot c^{n-2-j} \notin L$.

Next consider a string b^i from R_1 and a string $a_{k,j}$ from R_3. There are four possibilities.

1. $k \neq i$ and $0 \leq i \leq m-3$: Now b^i and $a_{k,j}$ are inequivalent since $b^i \cdot b^{m-2-i} \in L$ but $a_{k,j} \cdot b^{m-2-i} \notin L$.
2. $k \neq i$ and $i = m-2$: This implies that $k < m-2$ and, thus, b^i and $a_{k,j}$ are inequivalent since $a_{k,j} \cdot b^{m-2-k} \in L$ but $b^i \cdot b^{m-2-k} \notin L$.
3. $k \neq i$ and $i = m-1$: The path for $b^i = b^{m-1}$ must end at the sink state for the minimal DFA for L since b^{m-1} is not a prefix of any string in L. On the other hand, $a_{k,j}$ can be completed to a string of L by appending zero or more symbols c.
4. $k = i$: Now the strings under consideration are b^i and $a_{i,j}$.
   (a) $j < n-2$: We note that $a_{i,j} \cdot c^{n-2-j} \in L$ but $b^i \cdot c^{n-2-j} \notin L$ since no string of L can have both b's and c's. Note that $k = i$ implies that $i \geq 1$.
   (b) $j = n-2$: Since $j = n-2$, $i < m-2$ by the definition of R_3. Now $b^i \cdot \lambda \notin L$ but $a_{i,j} = a_{i,n-2} \cdot \lambda \in L(B) \subseteq L$.

   Therefore, b^i and $a_{i,j}$ are inequivalent.

Symmetrically, we see that any string from R_2 cannot be equivalent with a string from R_3. This case is, in fact, simpler than the previous case since R_2 is more restrictive than R_1.

Finally, we show that all strings from R_1 (respectively, from R_2 and from R_3) are pairwise inequivalent with each other.

1. R_1: By appending a suitable number of b's, we can always distinguish two distinct strings from R_1.
2. R_2: By appending a suitable number of c's, we can always distinguish two distinct strings from R_2.

3. R_3: Consider two distinct strings $a_{i,j}$ and $a_{x,y}$ from R_3. Without loss of generality, we assume that $i < x$. The other possibility, where j and y differ, is completely symmetric. Since $a_{i,j} \cdot b^{m-2-i} \in L$ but $a_{x,y} \cdot b^{m-2-i} \notin L$, $a_{i,j}$ and $a_{x,y}$ are inequivalent. Note that $m-2-i > 0$ and, thus, the inequivalence holds even in the case where $y = n-2$.

This concludes the proof. □

In the construction of the proof of Lemma 2, $|\Sigma|$ is $mn-2m-2n+5$. By using a more complicated construction, it would be possible to reduce $|\Sigma|$. However, below we see that the alphabet Σ has to depend on m and n.

Lemma 3 *The upper bound $mn-(m+n)$ for union cannot be reached with a fixed alphabet when m and n are arbitrarily large.*

Proof. Assume that Σ is a fixed alphabet with $|\Sigma| = t$. We show that it is not possible to reach the upper bound $mn-(m+n)$ for arbitrarily large m and n.

Let A have state set $\{p_0, p_1, \ldots, p_{m-1}\}$ and B have state set $\{q_0, q_1, \ldots, q_{n-1}\}$, where p_0 and q_0 are start states. We assume that $m, n \geq 2$. The only finite language with a one-state DFA is $\emptyset$.

Without loss of generality we can assume that A and B are minimal DFAs. Since A recognizes a finite language, we can order the states such that if p_j is reachable from p_i, then $i < j$. In particular, p_{m-1} is the sink state and p_{m-2} the non-exiting final state. The states of B are ordered in a similar way.

Let C be the DFA with $mn-(m+n)$ states recognizing $L(A) \cup L(B)$ that is constructed as in the proof of Lemma 1. The DFA C is obtained from $M_\cup$ by identifying some of the pair-states.

Let $i \in \{1, \ldots, m-2\}$. Any string that reaches p_i from p_0 can go through only the states $p_1, \ldots, p_{i-1}$ in between and cannot visit the same state twice. Hence, there are at most

$$t + t^2 + \cdots + t^i = \frac{t(t^i-1)}{t-1} =_{\text{def}} D(i)$$

strings that can reach p_i from p_0.

Since C is deterministic, this implies that for any fixed i, $1 \leq i \leq m-2$, at most $D(i)$ of the pair-states (p_i, q_j), $j \in \{1, \ldots, n-2\}$, are reachable from (p_0, q_0) in C. Thus, if $n-2 > D(i)$, then some pair-states of C with p_i as the first component are not reachable. From the proof of Lemma 1, we know that C recognizes $L(A) \cup L(B)$ and, hence, the minimal DFA for $L(A) \cup L(B)$ needs fewer than $mn-(m+n)$ states. □

We establish the following statement from Lemmas 1 and 2.

Theorem 1 *Given two minimal DFAs A and B for finite languages, $mn-(m+n)$ states are necessary and sufficient in the worst-case for the minimal DFA of $L(A) \cup L(B)$, where $m = |A|$ and $n = |B|$.*

Lemma 3 shows that the upper bound in Lemma 1 is unreachable if $|\Sigma|$ is fixed and m and n are arbitrarily large whereas Lemma 2 shows that the upper bound is reachable if $|\Sigma|$ depends on m and n. These results naturally lead us to examine the state complexity of union with a fixed sized alphabet.

Below, for ease of presentation, we first give the lower bound result using a four character alphabet and afterward explain how the construction can be modified for a binary alphabet.

Lemma 4 *Let Σ be an alphabet with four characters. There exists a constant α such that the following holds for infinitely many $m, n \geq 1$, where $\min\{m, n\}$ is unbounded. There exist DFAs A and B, with m and n states respectively, that recognize finite languages over Σ such that the minimal DFA for the union $L(A) \cup L(B)$ requires $\alpha(\min\{m, n\})^2$ states.*

The same result holds for a binary alphabet.

Proof. Let $\Sigma = \{a, b, c, d\}$. We introduce some notations for the proof. Given an even length string $w \in \Sigma^*$, $\mathsf{odd}(w)$ denotes the subsequence of characters that occur in odd positions in w and, thus, the length of $\mathsf{odd}(w)$ is half the length of w. For example, if $w = adacbcbc$, then $\mathsf{odd}(w) = aabb$. Similarly, $\mathsf{even}(w)$ denotes the subsequence of characters that occur in even positions in w. With the same example as above, $\mathsf{even}(w) = dccc$.

Let $s \geq 1$ be arbitrary and $r = \lceil \log s \rceil$. We define the finite language

$$L_1 = \{w_1 w_2 \mid |w_1| = 2r,\ w_2 = \mathsf{odd}(w_1) \in \{a, b\}^*,\ \mathsf{even}(w_1) \in \{c, d\}^*\}.$$

The language L_1 can be recognized by a DFA A with at most $10s$ states. For reading a prefix of length $2r$ of an input string, the start state of A has two out-transitions with labels a and b and the two corresponding target states are different. Then, each target state has two out-transitions with labels c and d where the target states are the same. This repeats in A until we finish reading a prefix of length $2r$. All other transitions go to the sink state. Fig. 3 illustrates the construction of A with $r = 3$.

The computations of A, which do not go to the sink state, on inputs of length $2r$ form a tree-like structure that branches into 2^r different states. Each of the 2^r states represents a unique string $\mathsf{odd}(u) \in \{a, b\}^*$, where u is the (prefix of the) input of length $2r$. Then, the computation from each of these 2^r states verifies whether or not the remaining suffix is identical to the string $\mathsf{odd}(u)$. This can be accomplished using a tree that merges all the computations into a single final state. (See the right part of Fig. 3 for an example.) From each state, there is only one out-transition (either with symbol a or b), if we ignore transitions into the sink state. (The structure looks like a tree when we ignore transitions into the sink state.)

The first "expanding" tree of A that corresponds to computations on strings of length $2r$ (for instance, the left part of Fig. 3) uses less than $4 \cdot 2^r < 8s$ states[a] since we repeat each level with the c, d transitions in the tree and $s \leq 2^r < 2s$.

Finally, consider the number of states in the "merging" tree. (This is the right part of Fig. 3.) The merging tree has 2^r leaves and, therefore, the tree needs at most $2 \cdot 2^r < 4s$ states. However, we observe that the the 2^r leaves of the expanding tree are identical to the 2^r leaves of the merging tree, as states of A. Therefore, we only need $2s$ "new" states for the merging tree.

[a]Note that a balanced tree with 2^r leaves has less than $2 \cdot 2^r$ nodes.

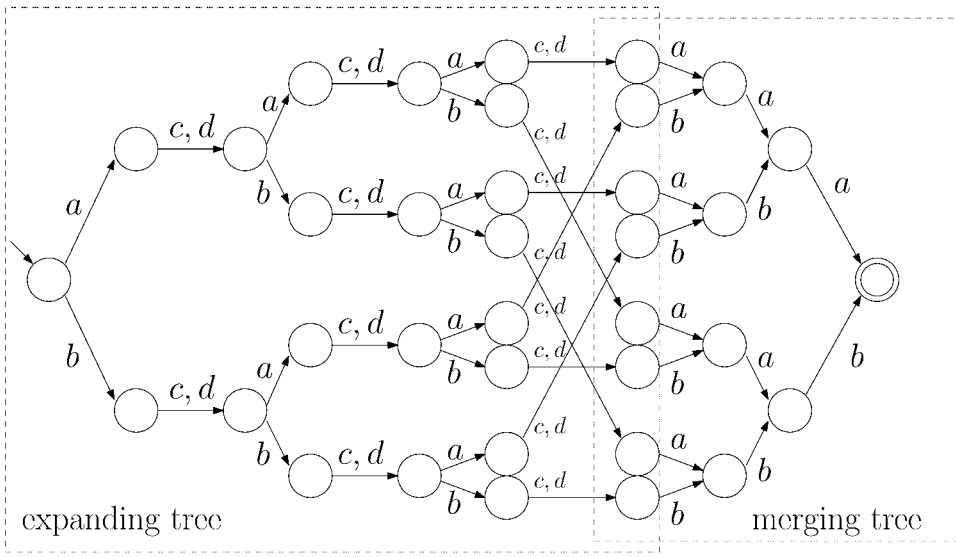

Fig. 3. A DFA A that recognizes L_1 when $r = 3$. We omit the sink state and its in-transitions.

The total number of states in A is less than $10s$. (1)

Symmetrically, we define

$$L_2 = \{w_1 w_2 \mid |w_1| = 2r,\ \mathsf{odd}(w_1) \in \{a,b\}^*,\ w_2 = \mathsf{even}(w_1) \in \{c,d\}^*\}.$$

The language L_2 consists of strings uv, where $|u| = 2r$, odd characters of u are in $\{a,b\}$, even characters of u are in $\{c,d\}$ and $\mathsf{even}(u)$ coincides with v. Using an argument similar to that for equation (1), we establish that

L_2 can be recognized by a DFA with less than $10s$ states. (2)

Now let $L = L_1 \cup L_2$. Let u_1 and u_2 be arbitrary distinct strings of length $2r$ such that $\mathsf{odd}(u_i) \in \{a,b\}^*$ and $\mathsf{even}(u_i) \in \{c,d\}^*$, for $i = 1, 2$.

If $\mathsf{odd}(u_1) \neq \mathsf{odd}(u_2)$, then $u_1 \cdot \mathsf{odd}(u_1) \in L_1 \subseteq L$ but $u_2 \cdot \mathsf{odd}(u_1) \notin L$. Hence, u_1 and u_2 are not equivalent modulo the right invariant congruence of L. Similarly, if $\mathsf{even}(u_1) \neq \mathsf{even}(u_2)$, then, $u_1 \cdot \mathsf{even}(u_1) \in L_2 \subseteq L$ but $u_2 \cdot \mathsf{even}(u_1) \notin L$.

The above implies that the right invariant congruence of L has at least $2^r \cdot 2^r \geq s^2$ different classes. Therefore, if $m = n = 10s$ is the size of the minimal DFAs for the finite languages L_1 and L_2, then from equations (1) and (2) we know that the minimal DFA for $L = L_1 \cup L_2$ needs at least

$$\frac{1}{100} n^2 \text{ states.} \tag{3}$$

Notice that we have used an alphabet Σ with four characters. If we encode the languages L_1 and L_2 over a binary alphabet, then we get a similar lower bound

for the state complexity of $L_1 \cup L_2$ with the only change that the constant $\frac{1}{100}$ in equation (3) will become smaller. □

4. Intersection of Finite Languages

Next we examine the state complexity of intersection of finite languages. Our approach is again based on the structural properties of minimal DFAs recognizing finite languages. Using the well-known Cartesian product construction, we obtain a DFA for the intersection of two regular languages. For details, see Hopcroft and Ullman [10].

Proposition 3 *Given two DFAs* $A = (Q_1, \Sigma, \delta_1, s_1, F_1)$ *and* $B = (Q_2, \Sigma, \delta_2, s_2, F_2)$, *let* $M_\cap = (Q_1 \times Q_2, \Sigma, \delta, (s_1, s_2), F_1 \times F_2)$, *where for all* $p \in Q_1$ *and* $q \in Q_2$ *and* $a \in \Sigma$, $\delta((p,q),a) = (\delta_1(p,a), \delta_2(q,a))$. *Then,* $L(M_\cap) = L(A) \cap L(B)$.

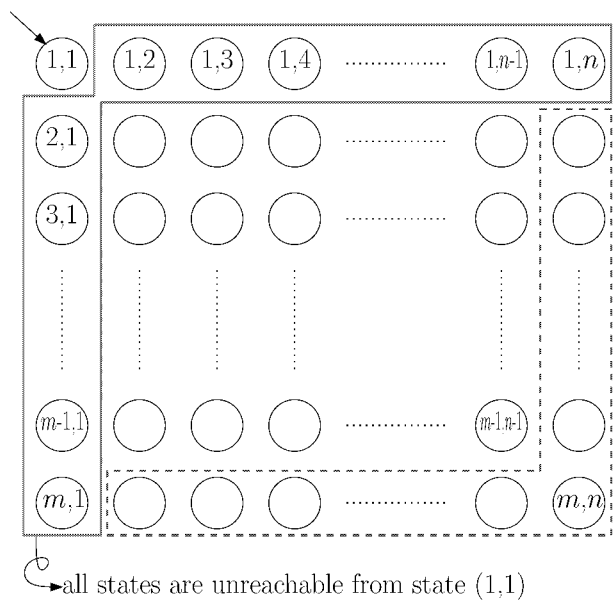

Fig. 4. The DFA recognizing the intersection of two finite languages. We omit all transitions in the figure.

Fig. 4 illustrates the construction of $M_\cap$ that recognizes the intersection of languages recognized by DFAs A and B. In the figure, m and n denote the sink states of A and B and $m-1$ and $n-1$ denote non-exiting final states of A and B, respectively. All states of $M_\cap$ in the first row and in the first column are unreachable from $(1,1)$ since A and B are non-returning and, thus, these states are useless in $M_\cap$. Moreover, by the construction, all remaining states in the last row and in the last column are equivalent to the sink state and, therefore, can be merged. Let us examine the remaining states in the second-to-last row and in the second-to-last column except for $(m-1, n-1)$.

Claim 1 *A state* $(i, n-1)$ *in the second-to-last column, for* $2 \leq i \leq m-1$, *is either*

equivalent to $(m-1, n-1)$ *if state* i *is a final state in* A*, or,*

equivalent to (m, n) *if state* i *is not a final state in* A*.*

Proof. By the construction, for a state (i, j) of $M_\cap$, the right language $L_{(i,j)}$ of (i, j) is $L_i \cap L_j$. Thus, if state i is a final state in A, then $L_{(i,n-1)} = \{\lambda\}$ and, therefore, $(i, n-1)$ and $(m-1, n-1)$ are equivalent. Otherwise, $L_{(i,n-1)} = \emptyset$ and, thus, $(i, n-1)$ and (m, n) are equivalent. □

A property completely analogous to Claim 1 holds for the states in the second-to-last row in $M_\cap$. Therefore, all the remaining states in the second-to-last row and in the second-to-last column except for $(m-1, n-1)$ can be merged with either $(m-1, n-1)$ or (m, n). Thus, the number of remaining pairwise inequivalent states is

$$mn - \{(m-1) + (n-1)\} - \{(m-2) + (n-2)\} - \{(m-3) + (n-3)\}$$

$$= mn - 3(m+n) + 12,$$

where $\{(m-1) + (n-1)\}$ represents the number of states merged in the first row and the first column, $\{(m-2) + (n-2)\}$ is from the last row and the last column and $\{(m-3)+(n-3)\}$ is from the second-to-last row and the second-to-last column. We establish the following lemma from the calculation.

Lemma 5 *Given two minimal DFAs* A *and* B *recognizing finite languages, where* $m = |A|$ *and* $n = |B|$*,* $mn - 3(m+n) + 12$ *states are sufficient for the intersection of* $L(A)$ *and* $L(B)$*.*

We now show that $mn - 3(m+n) + 12$ states are necessary and, therefore, the bound is tight.

Let m and n be positive numbers and choose

$$\Sigma = \{a_{i,j} \mid 1 \le i \le m-2 \text{ and } 1 \le j \le n-2\} \cup \{a_{m-1,n-1}\}.$$

Let $A = (Q_1, \Sigma, \delta_1, p_0, \{p_{m-2}\})$, where $Q_1 = \{p_0, p_1, \ldots, p_{m-1}\}$ and δ_1 is defined by setting:

- $\delta_1(p_x, a_{i,j}) = p_{x+i}$, for $0 \le x \le m-2$, $1 \le i \le m-2$ and $1 \le j \le n-2$.

If the sum $x + i$ is larger than $m-1$, then p_{x+i} is the sink state $(= p_{m-1})$. In all other cases not covered above, the relation δ_1 takes the source state to the sink state p_{m-1}. In particular, this means that $a_{m-1,n-1}$ takes all states of A to the sink state.

Next, let $B = (Q_2, \Sigma, \delta_2, q_0, \{q_{m-2}\})$, where $Q_2 = \{q_0, q_1, \ldots, q_{n-1}\}$ and δ_2 is defined by:

- $\delta_2(q_x, a_{i,j}) = q_{x+j}$, for $0 \le x \le m-2$, $1 \le j \le n-2$ and $1 \le i \le m-2$.

Similarly, if the sum $x+j$ is larger than $n-1$, then q_{x+j} is the sink state $(= q_{n-1})$. Again for all cases not covered above, the target state of the transition is the sink state q_{n-1}. Fig. 5 illustrates the construction of DFAs A and B.

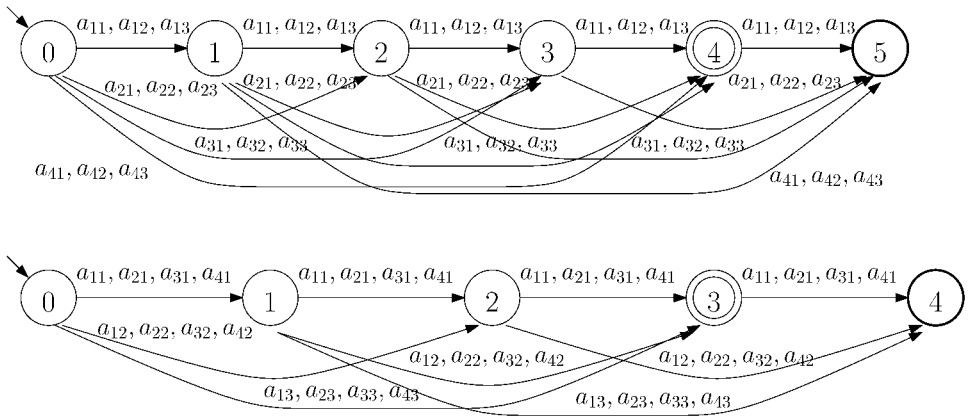

Fig. 5. The construction of DFAs A and B with $m = 6$ and $n = 5$. Here the state 5 of A (above) and state 4 of B (below) are the sink states. We omit a large number of in-transitions into the sink states.

Lemma 6 *Let A and B be as above and $L = L(A) \cap L(B)$. The minimal DFA for L needs $mn - 3(m+n) + 12$ states.*

Proof. We prove the statement by showing that there exists a set R of $mn - 3(m+n)+12$ strings over Σ that are pairwise inequivalent modulo the right invariant congruence of L.

We choose $R = R_1 \cup R_2 \cup R_3 \cup R_4$, where

$R_1 = \{\lambda\}$.

$R_2 = \{a_{m-2,n-2}\}$.

$R_3 = \{a_{m-1,n-1}\}$.

$R_4 = \{a_{i,j} \mid \text{for } 1 \leq i \leq m-3 \text{ and } 1 \leq j \leq n-3\}$.

Any string x from $R_2 \cup R_3 \cup R_4$ cannot be equivalent with λ from R_1 since $\lambda \cdot a_{m-2,n-2} \in L$ but $x \cdot a_{m-2,n-2} \notin L$. Similarly, any string x from $R_1 \cup R_3 \cup R_4$ cannot be equivalent with $a_{m-2,n-2}$ from R_2 since $a_{m-2,n-2} \cdot \lambda \in L$ but $x \cdot \lambda \notin L$. Note that the string $a_{m-1,n-1}$ from R_3 is not a prefix of any string in L whereas any string x from $R_1 \cup R_2 \cup R_4$ can be completed to a string in L by appending a suitable suffix. Therefore, R_1, R_2 and R_3 are inequivalent with each other including R_4.

Finally, we consider two strings $a_{i,j}$ and $a_{x,y}$ in R_4. Note that $a_{i,j} \cdot a_{m-2-i,n-2-j} \in L$ but $a_{x,y} \cdot a_{m-2-i,n-2-j} \notin L$ when $(i,j) \neq (x,y)$. Therefore, any two strings from R_4 are not equivalent.

Now we count the number of strings in R. We note that $|R_1| = |R_2| = |R_3| = 1$ and $|R_4| = (m-3)(n-3)$. Therefore, $|R| = mn - 3(m+n) + 12$. This implies that there are at least $mn - 3(m+n) + 12$ states in the minimal DFA for L. □

We obtain the following result from Lemmas 5 and 6.

Theorem 2 *Given two minimal DFAs A and B recognizing finite languages, where $m = |A|$ and $n = |B|$, $mn - 3(m+n) + 12$ states are necessary and sufficient in the worst-case for the intersection of $L(A)$ and $L(B)$.*

Note that the upper bound $mn - 3(m+n) + 12$ is reachable when $|\Sigma|$ depends on m and n as shown in Lemma 6. On the other hand, using the same argument as in Lemma 3, we can prove that the upper bound cannot be reached for large values of m and n when the alphabet Σ is fixed.

Finally, we establish a lower bound for the state complexity of intersection of finite languages over a fixed alphabet that is within a multiplicative constant from the upper bound.

Lemma 7 *Let Σ be an alphabet with four characters. There exists a constant α such that the following holds for infinitely many $m, n \geq 1$, where $\min\{m, n\}$ is unbounded. There exist minimal DFAs A and B that recognize finite languages over Σ such that the minimal DFA for the intersection $L(A) \cap L(B)$ requires $\alpha(\min\{m, n\})^2$ states, where $|A| = m$ and $|B| = n$.*

The same result holds for a binary alphabet.

Proof. The construction is a modification of the construction that we used in Lemma 4. For ease of presentation, we first define the languages over a four character alphabet and, afterward, observe how this can be done with a binary alphabet.

Let $\Sigma = \{a, b, c, d\}$, and let s be an arbitrary integer and $r = \lceil \log s \rceil$. We define

$$\begin{aligned} L_3 \quad = \quad & \{w_1 w_2 \mid |w_1| = |w_2| = 2r, \ \ \mathsf{odd}(w_1) = \mathsf{odd}(w_2), \\ & \mathsf{odd}(w_1), \mathsf{odd}(w_2) \in \{a, b\}^*, \ \ \mathsf{even}(w_1), \mathsf{even}(w_2) \in \{c, d\}^*\}. \end{aligned}$$

The functions $\mathsf{odd}(w)$ and $\mathsf{even}(w)$ have been defined in Section 3.

It is easy to verify that L_3 can be recognized by a DFA with $14s$ states. The construction of the DFA is similar to that given for the language L_1 in Lemma 4 except that the new construction may need (close to) $4 \cdot 2s - 2s = 6s$ states for the "merging tree".

Similarly, we define

$$\begin{aligned} L_4 \quad = \quad & \{w_1 w_2 \mid |w_1| = |w_2| = 2r, \ \ \mathsf{even}(w_1) = \mathsf{even}(w_2), \\ & \mathsf{odd}(w_1), \mathsf{odd}(w_2) \in \{a, b\}^*, \ \ \mathsf{even}(w_1), \mathsf{even}(w_2) \in \{c, d\}^*\}. \end{aligned}$$

We, then, observe that L_4 can be recognized by a DFA with at most $14s$ states by a similar construction as was used for L_3.

Let $L = L_3 \cap L_4$ and u_1 and u_2 be distinct strings in $(\{a, b\}\{c, d\})^r$. Then, $u_1 u_1 \in L$ but $u_2 u_1 \notin L$ and, hence, u_1 and u_2 are not equivalent modulo the right invariant congruence of L. We have seen that the right invariant congruence of L has at least $2^{2r} \geq s^2$ classes. (Recall that $2^r \geq s$.) Thus, if $m = n = 14s$ is the size of the minimal DFAs for L_3 and L_4, then the minimal DFA for $L = L_3 \cap L_4$ needs at least

$$\frac{1}{196} n^2 \text{ states.} \tag{4}$$

This construction uses an alphabet of size four. The same construction works if we encode the characters over a binary alphabet and the modification only changes the constant $\frac{1}{196}$ in equation (4). Therefore, a state complexity lower bound $\alpha \cdot n^2$, where α is a constant, holds also for the intersection of finite languages over a binary alphabet. □

5. Conclusions

We have considered the state complexity of union and intersection of finite languages. Recall that the precise state complexity of union and intersection has been open although rough upper bounds were given by Yu [18].

Based on the structural properties of two minimal DFAs recognizing finite languages (having m and n states, respectively), we have proved that

1. For union, $mn - (m + n)$ states are necessary and sufficient.
2. For intersection, $mn - 3(m + n) + 12$ states are necessary and sufficient.

We have noted that the bounds can be reached in the worst-case if $|\Sigma|$ is allowed to depend on the sizes of the minimal DFAs for the two finite languages. If $|\Sigma|$ is fixed and m and n are arbitrarily large, then we have shown that the upper bounds for either case are not reachable.

The main open problem remaining would be to calculate the precise worst-case state complexity of union and intersection of finite languages as a function of the alphabet size. That is, what is the worst-case state complexity, as a function of m, n and k, for the union (intersection) of two finite languages over a k-letter alphabet, $k \geq 2$, where the component languages require m and n states, respectively.

References

1. C. Câmpeanu, K. Culik II, K. Salomaa, and S. Yu. State complexity of basic operations on finite languages. In *Proceedings of WIA'99*, Lecture Notes in Computer Science 2214, 60–70, 2001.
2. T. Claveirole, S. Lombardy, S. O'Connor, L.-N. Pouchet, and J. Sakarovitch. Inside Vaucanson. In *Proceedings of CIAA'05*, Lecture Notes in Computer Science 3845, 116–128, 2006.
3. M. Domaratzki. State complexity of proportional removals. *Journal of Automata, Languages and Combinatorics*, 7(4):455–468, 2002.
4. M. Domaratzki and K. Salomaa. State complexity of shuffle on trajectories. *Journal of Automata, Languages and Combinatorics*, 9(2-3):217–232, 2004.
5. K. Ellul, B. Krawetz, J. Shallit, and M.-W. Wang. Regular expressions: New results and open problems. *Journal of Automata, Languages and Combinatorics*, 9:233–256, 2004.
6. Y.-S. Han and K. Salomaa. State complexity of basic operations on suffix-free regular languages. In *Proceedings of MFCS'07*, Lecture Notes in Computer Science 4708, 501–512, 2007.
7. Y.-S. Han, K. Salomaa, and D. Wood. State complexity of prefix-free regular languages. In *Proceedings of DCFS'06*, 165–176, 2006. *Full version is submitted for publication.*

8. M. Holzer and M. Kutrib. Unary language operations and their nondeterministic state complexity. In *Proceedings of DLT'02*, Lecture Notes in Computer Science 2450, 162–172, 2002.
9. M. Holzer and M. Kutrib. Nondeterministic descriptional complexity of regular languages. *International Journal of Foundations of Computer Science*, 14(6):1087–1102, 2003.
10. J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 2 edition, 1979.
11. M. Hricko, G. Jirásková, and A. Szabari. Union and intersection of regular languages and descriptional complexity. In *Proceedings of DCFS'05*, 170–181, 2005.
12. J. Jirásek, G. Jirásková, and A. Szabari. State complexity of concatenation and complementation. *International Journal of Foundations of Computer Science*, 16(3):511–529, 2005.
13. A. Maslov. Estimates of the number of states of finite automata. *Soviet Mathematics Doklady*, 11:1373–1375, 1970.
14. C. Nicaud. Average state complexity of operations on unary automata. In *Proceedings of MFCS'99*, Lecture Notes in Computer Science 1672, 231–240, 1999.
15. G. Pighizzini and J. Shallit. Unary language operations, state complexity and Jacobsthal's function. *International Journal of Foundations of Computer Science*, 13(1):145–159, 2002.
16. D. R. Raymond and D. Wood. Grail: A C++ library for automata and expressions. *Journal of Symbolic Computation*, 17:341–350, 1994.
17. S. Yu. Regular languages. In G. Rozenberg and A. Salomaa, editors, *Word, Language, Grammar*, volume 1 of *Handbook of Formal Languages*, 41–110. Springer-Verlag, 1997.
18. S. Yu. State complexity of regular languages. *Journal of Automata, Languages and Combinatorics*, 6(2):221–234, 2001.
19. S. Yu, Q. Zhuang, and K. Salomaa. The state complexities of some basic operations on regular languages. *Theoretical Computer Science*, 125(2):315–328, 1994.