State Complexity of Deletion

Yo-Sub Han[1,⋆], Sang-Ki Ko[1,⋆], and Kai Salomaa[2,⋆⋆]

[1] Department of Computer Science, Yonsei University, 50, Yonsei-Ro, Seodaemun-Gu, Seoul 120-749, Korea
{emmous,narame7}@cs.yonsei.ac.kr
[2] School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada
ksalomaa@cs.queensu.ca

Abstract. It is well known that the language obtained by deleting an arbitrary language from a regular language is regular. We give an upper bound for the state complexity of deleting an arbitrary language from a regular language and a matching lower bound. We show that the state complexity of deletion is $n \cdot 2^{n-1}$ (respectively, $(n + \frac{1}{2}) \cdot 2^n - 2$) when using complete (respectively, incomplete) deterministic finite automata.

1 Introduction

The descriptional complexity of finite automata has been studied for over half a century [21–24] and there has been much renewed interest since the early 90's [11, 13, 20, 29]. Operational state complexity investigates the size of a DFA (deterministic finite automaton) needed to recognize the language obtained by applying a regularity preserving operation to given DFAs. The precise worst case state complexity of many basic language operations has been established; see [1, 4, 5, 7, 12, 14, 22, 25, 30] where further references can be found. Also there has been much work on the state complexity of combinations of basic language operations [2, 6, 8, 9, 15, 26].

Deletion is one of the basic operations in formal language theory [17, 18]. The deletion of a string v from a string u consists of erasing a contiguous substring v from u. We denote the result of deleting a language L_2 from a language L_1 by $L_1 \rightsquigarrow L_2$.[1]

Deletion is the simplest and most natural generalization of the left/right quotient [18]. It is known that for L_1 recognized by a DFA with n states and an arbitrary language L_2, the worst case state complexity of the left-quotient $L_2 \backslash L_1$ is $2^n - 1$ and the state complexity of the right-quotient L_1/L_2 is n [29]. Recently, the state complexity of insertion which, using the terminology of [16], is the left inverse of deletion, was investigated in [10].

⋆ Han and Ko were supported by the Basic Science Research Program through NRF funded by MEST (2012R1A1A2044562).
⋆⋆ Salomaa was supported by the Natural Sciences and Engineering Research Council of Canada Grant OGP0147224.
[1] See section 2 for a formal definition.

A.M. Shur and M.V. Volkov (Eds.): DLT 2014, LNCS 8633, pp. 37–48, 2014.

It is well known that $L_1 \rightsquigarrow L_2$ is always regular for a regular language L_1 and an arbitrary language L_2 [18]. However, in spite of deletion being a fundamental language operation its precise state complexity has not been studied in the literature. When L_1 is recognized by a DFA with n states, the proof of Theorem 1 of [18] yields an upper bound 2^{2n} for the size of the DFA needed to recognize $L_1 \rightsquigarrow L_2$. The proof works for an arbitrary language L_2 and is not, in general, effective.

More general types of deletion operations, called *deletion along trajectories* have been considered by Domaratzki [3] and Kari and Sosik [19]. In this context a set of trajectories is a set $T \subseteq \{i, d\}^*$ where i stands for "insert" and d stands for "delete". The result of deleting a language L_2 from a language L_1 along a set of trajectories T is denoted $L_1 \rightsquigarrow_T L_2$. Deletion along a regular set of trajectories preserves regularity [3], that is, for regular languages L_1, L_2 and T, also $L_1 \rightsquigarrow_T L_2$ is regular. The ordinary deletion operation we consider here is defined by the set of trajectories $i^*d^*i^*$, and the construction used in the proof of Lemma 3.1 of [3] would yield an upper bound 2^{3mn} for the state complexity of $L_1 \rightsquigarrow_{i^*d^*i^*} L_2$ when L_1 (respectively, L_2) is recognized by a DFA of size m (respectively, n). Naturally, Lemma 3.1 of [3] deals with deletion along arbitrary regular sets of trajectories and the result cannot be expected to yield a good bound in the very special case we are considering here.

Most of the literature uses complete DFAs to measure the state complexity of a regular language, but state complexity based on incomplete DFAs also has been considered. Câmpeanu et al. [1] give the state complexity of shuffle in terms of incomplete DFAs while the precise state complexity of shuffle in terms of complete DFAs remains still open. For a given regular language the sizes of the minimal complete and the minimal incomplete DFA differ by at most one state, however, there can be a more significant difference in the state complexity functions when the measure is based on complete and incomplete DFAs, respectively.

In this paper we give a tight state complexity bound for the language obtained from a regular language by deleting an arbitrary language. We show that if L_1 is recognized by a complete DFA with n states and L_2 is an arbitrary language, the complete DFA for the language $L_1 \rightsquigarrow L_2$ needs $n \cdot 2^{n-1}$ states in the worst case. The corresponding state complexity function based on incomplete DFAs is shown to be $(n+\frac{1}{2}) \cdot 2^n - 2$. While the upper bounds hold for arbitrary languages L_2 (that need not be even recursively enumerable) we show that matching lower bound constructions can be found where L_2 consists of a single string of length one. We give conditions based on L_2 and the DFA for L_1 that are necessary for the state complexity of deletion to reach the worst case bound.

2 Preliminaries

We assume that the reader is familiar with the basics of finite automata and formal languages and recall here just some definitions and notation. For more information on the topic the reader may consult the monographs [27, 28] or the survey [29].

In the following Σ always stands for a finite alphabet and the set of strings over Σ is Σ^*. A language is a subset of Σ^*. The cardinality of a finite set S is denoted $|S|$.

The set of strings obtained from $u \in \Sigma^*$ by deleting a string $v \in \Sigma^*$ is

$$u \rightsquigarrow v = \{w \in \Sigma^* \mid (\exists u_1, u_2 \in \Sigma^*)\ w = u_1 u_2 \text{ and } u = u_1 v u_2\}.$$

For example, $bababa \rightsquigarrow aba = \{bba, bab\}$. The *deletion operation* is extended in the natural way for languages $L_1, L_2 \subseteq \Sigma^*$ by setting

$$L_1 \rightsquigarrow L_2 = \bigcup_{u \in L_1, v \in L_2} u \rightsquigarrow v.$$

An *incomplete deterministic finite automaton* (incomplete DFA) is a five-tuple $A = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ is an alphabet, δ is a partial function $Q \times \Sigma \to Q$, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is a set of final (or accepting) states.

The transition function δ is in the usual way extended as a partial function $Q \times \Sigma^* \to Q$ and the language recognized by A is $L(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}$. A language is regular if it is recognized by some DFA.

For $q \in Q$, $P \subseteq Q$, $b \in \Sigma$ and $L \subseteq \Sigma^*$ we also denote

$$\delta(P, b) = \{\delta(p, b) \mid p \in P\} \quad \text{and} \quad \delta(q, L) = \{\delta(q, w) \mid w \in L\}.$$

A DFA $A = (Q, \Sigma, \delta, q_0, F)$ is said to be *complete* if δ is a total function $Q \times \Sigma \to Q$. We will use both complete and incomplete DFAs and, when not separately mentioned, by a DFA we mean an incomplete DFA.[2]

A DFA $A = (Q, \Sigma, \delta, q_0, F)$ is *minimal* if each state $q \in Q$ is reachable from the initial state q_0 (that is, $\delta(q_0, w) = q$ for some string w) and no two states $q_1, q_2 \in Q$, $q_1 \neq q_2$, are equivalent. States $q_1, q_2 \in Q$ are said to be equivalent if

$$(\forall w \in \Sigma^*)\ \delta(q_1, w) \in F \text{ iff } \delta(q_2, w) \in F.$$

The minimal (complete or incomplete) DFA for a given regular language L is unique and the sizes of the minimal complete and incomplete DFAs for L differ by at most one state. The minimal complete DFA may have a dead state (or sink state). In the minimal incomplete DFA the dead state can always be omitted.

The *state complexity* of L, $\mathrm{sc}(L)$, is the size of the minimal complete DFA recognizing L. Similarly, the *incomplete state complexity* of L, $\mathrm{isc}(L)$, is the size of the minimal incomplete DFA recognizing L. For each regular language L either $\mathrm{sc}(L) = \mathrm{isc}(L) + 1$ or $\mathrm{sc}(L) = \mathrm{isc}(L)$.

3 Upper Bound for Deletion

It is known that the result of deleting an arbitrary language from a regular language is regular [18]. Hence in the lemmas establishing the upper bound for

[2] Naturally, a complete DFA is just a special case of an incomplete DFA.

deletion (for complete or incomplete DFAs) we do not need to assume that the deleted language L_2 is regular. However, in the case of an arbitrary L_2 finding a DFA for the language $L_1 \rightsquigarrow L_2$ is not, in general, effective.

First we give an upper bound construction for complete DFAs.

Lemma 1. *Consider $L_1, L_2 \subseteq \Sigma^*$ where L_1 is recognized by a complete DFA with n states. Then*

$$\mathrm{sc}(L_1 \rightsquigarrow L_2) \leq n \cdot 2^{n-1}.$$

Proof. Let $A = (Q, \Sigma, \delta, q_0, F_A)$ be a complete DFA for L_1 where $|Q| = n$. To recognize the language $L_1 \rightsquigarrow L_2$ we define a DFA

$$B = (P, \Sigma, \gamma, p_0, F_B),$$

where $P = \{(r, R) \mid r \in Q, R \subseteq Q, \delta(r, L_2) \subseteq R\}$, $p_0 = (q_0, \delta(q_0, L_2))$ and

$$F_B = \{(r, R) \mid r \in Q, R \subseteq Q, \delta(r, L_2) \subseteq R \text{ and } R \cap F_A \neq \emptyset\}.$$

It remains to define the transitions of γ. For $(r, R) \in P$ and $b \in \Sigma$ we set

$$\gamma((r, R), b) = (\delta(r, b), \;\; \delta(R, b) \cup \delta(\delta(r, b), L_2)). \tag{1}$$

The transition relation always adds the elements of $\delta(\delta(r, b), L_2)$ to the second component and, consequently, the state $\gamma((r, R), b)$, as defined above, is an element of P.

The intuitive idea of the construction is as follows. In order to recognize the language $L_1 \rightsquigarrow L_2$, the DFA B must check that the input string w can be completed to a string of L_1 by inserting a string $u \in L_2$ in some position, that is, for some decomposition $w = w_1 w_2$ we have $w_1 u w_2 \in L_1$. Since we do not know at which position the string $u \in L_2$ is to be inserted and B has to be deterministic, roughly speaking, B has to keep track of all computations of A on strings where a string of L_2 was deleted from some earlier position.

The first component of the states of B simply simulates the computation of A, i.e., it keeps track of the state of A, assuming that up to the current position in the input a string of L_2 was not yet deleted. The second component of the states of B keeps track of all states that A could be in assuming that at some point in the preceding computation a string of L_2 was deleted from the input.

We need to verify that the transitions of B (as defined in (1)) preserve these properties. For the first component it is clear that the simulation works as claimed. To verify the claim for the second component, assume that the input is ubv, $u, v \in \Sigma^*$, $b \in \Sigma$ and after reading the prefix u the DFA B has reached a state (r, R). In the following discussion b refers to the particular symbol occurrence just after the prefix u. After reading the symbol b, the second component of the state of B will be $\delta(R, b) \cup \delta(\delta(r, b), L_2))$, where R consists of states that A could be in, assuming a string of L_2 was deleted somewhere before the symbol b and the states of $\delta(R, b)$ are then the states A could be in after reading b assuming a string of L_2 was deleted before symbol occurrence b. On the other hand, $r = \delta(q_0, u)$, i.e., r is the state A reaches after reading the prefix

u where no deletion has occurred, and hence $\delta(\delta(r,b),L_2)$ consists of exactly all states A can be in the simulated computation, assuming a string of L_2 was deleted directly after the symbol occurrence b. This means that the transition relation γ correctly preserves the property that the second component of the state of B consists of all states that A could be in assuming a string of L_2 was deleted some time previously.

The choice of final states guarantees that B accepts exactly the strings obtained from strings of $L(A)$ by deleting a string of L_2 at some position.

We still need to verify that the number of states of B is as claimed. If $L_2 = \emptyset$, then $L_1 \rightsquigarrow L_2 = \emptyset$ and $L_1 \rightsquigarrow L_2$ has a DFA of size one. Hence in what follows we can assume that $L_2 \neq \emptyset$.

Since A is a complete DFA and $L_2 \neq \emptyset$, for each $r \in Q$ we have $|\delta(r,L_2)| \geq 1$. This means that for a given $r \in Q$, there exist at most $2^{|Q|-1}$ sets R such that (r,R) is a state of B. Thus, the number of states of B is at most $|Q| \cdot 2^{|Q|-1}$. □

Next we consider the case of incomplete DFAs. The upper bound construction uses similar ideas as the above proof of Lemma 1, and we just need to modify the construction to allow the possibility of undefined transitions.

Lemma 2. *Let $L_1, L_2 \subseteq \Sigma^*$ where L_1 is recognized by an incomplete DFA A with n states. Then*

$$\mathrm{isc}(L_1 \rightsquigarrow L_2) \leq (n+1) \cdot 2^n - (2^{n-1}+2).$$

Proof. Let $A = (Q, \Sigma, \delta, q_0, F_A)$ be an incomplete DFA for L_1, $|Q| = n$. We define the completion of δ as a function $\delta' : (Q \cup \{\mathrm{dead}\}) \times \Sigma \to Q \cup \{\mathrm{dead}\}$ by setting for $r \in Q \cup \{\mathrm{dead}\}$ and $b \in \Sigma$,

$$\delta'(r,b) = \begin{cases} \delta(r,b), & \text{if } r \in Q \text{ and } \delta(r,b) \text{ is defined;} \\ \mathrm{dead}, & \text{otherwise.} \end{cases}$$

To recognize the language $L_1 \rightsquigarrow L_2$ we define a DFA

$$B = (P, \Sigma, \gamma, p_0, F_B),$$

where $P = (Q \cup \{\mathrm{dead}\}) \times 2^Q - \{(\mathrm{dead},\emptyset), (\mathrm{dead},Q)\}$, $p_0 = (q_0, \delta(q_0,L_2))$ and

$$F_B = \{(r,R) \mid r \in Q \cup \{\mathrm{dead}\}, R \subseteq Q \text{ and } R \cap F_A \neq \emptyset\}.$$

(Note that $|P| = (n+1) \cdot 2^n - 2$. However, as will be seen below at least 2^{n-1} elements of P will be unreachable as states of B.)

The transitions of γ are defined by setting, for $(r,R) \in P$ and $b \in \Sigma$,

$$\gamma((r,R),b) = \begin{cases} (\delta'(r,b),\ \delta(R,b) \cup \delta(\delta(r,b),L_2)), & \text{if } r \in Q \text{ and } [\delta'(r,b) \neq \mathrm{dead} \\ & \quad \text{or } \delta(R,b) \cup \delta(\delta(r,b),L_2) \neq \emptyset]; \\ (\mathrm{dead}, \delta(R,b)), & \text{if } r = \mathrm{dead} \text{ and } \delta(R,b) \neq \emptyset; \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

As in the proof of the previous lemma, the idea is that the first component simulates the computation of the original DFA A, assuming a string of L_2 has

so far not been deleted (and using the new state "dead" to indicate that the simulated computation of A has failed), and the second component keeps track of the set of all possible states that A can be in, assuming a string of L_2 was deleted somewhere previously (this set can now be empty because A is incomplete). We leave the details of verifying that B recognizes $L(A) \rightsquigarrow L_2$ to the reader.

Below we explain why the states $(\text{dead}, \emptyset)$ and (dead, Q) can be omitted from the state set of B, and that, furthermore at least 2^{n-1} elements of P must be unreachable as states of B.

The state $(\text{dead}, \emptyset)$ would be a sink state of the DFA B and, according to the definition of γ, it is never entered. Also, we note that when the computation, for the first time, reaches a state where the first component is "dead" this has to occur on an alphabet symbol that has at least one undefined transition in A. This means that when the computation initially reaches a state with first component "dead", the cardinality of the second component is at most $|Q| - 1$, and after that point the transitions of γ do not add new states to the second component because if the deletion of the string of L_2 did not occur previously, the computation has already failed. Thus, the state (dead, Q) is always unreachable.

To verify the unreachability of 2^{n-1} further states, without loss of generality, we can assume that for some $q_1 \in Q$ and $w_1 \in L_2$, $\delta(q_1, w_1)$ is defined. Note that in the opposite case, no string of L_2 can occur as a substring of a string of $L(A) = L_1$ and, hence, $L_1 \rightsquigarrow L_2 = \emptyset$. Now all transitions of B that enter a state with the first component q_1 add the element $\delta(q_1, w_1)$ to the second components. This means that all elements (q_1, R), $\delta(q_1, w_1) \notin R$, are unreachable. □

In the above proof we noted that 2^{n-1} states of the constructed DFA B are unreachable for each state q of A such that $\delta(q, w)$ is defined for some $w \in L_2$. Thus, the worst case state complexity blow-up can occur only when transitions spelling out a string in L_2 originate only from one state of A and, slightly more precisely, we get the following necessary condition for languages that can reach the worst case state complexity of the deletion operation.

Corollary 1. *Let $A = (Q, \Sigma, \delta, q_0, F)$ be an incomplete DFA with n states, $L_1 = L(A)$ and L_2 is an arbitrary language. Then a necessary condition for* $\text{isc}(L_1 \rightsquigarrow L_2)$ *to reach the upper bound $(n+1) \cdot 2^n - (2^{n-1} + 2)$ given by Lemma 2 is that*

$$(\exists q \in Q)\ [\ |\delta(q, L_2)| = 1 \text{ and } (\forall p \in Q, p \neq q)\ \delta(p, L_2) = \emptyset\ ].$$

4 Lower Bound Constructions

As our main result we show here that the bounds given in the previous section are optimal. We begin with the case of complete DFAs where the construction is somewhat simpler.

4.1 Lower Bound for Complete DFAs

From the construction of the complete DFA B for $L_1 \rightsquigarrow L_2$ in Lemma 1 we know that the possible states of B are pairs (r, R) where r (respectively, R) is a

state (respectively, a set of states) of the DFA A recognizing L_1 and R has the property that it contains all states that are reachable from r on a string of L_2 (and possibly other states). Hence a possible worst case construction should use a singleton language L_2 or a language L_2 such that for any given state r of A, all strings of L_2 take r to the same state. In the proof of Lemma 3 we choose L_2 to be a singleton language consisting of a string of length one.

Lemma 3. *Let $\Sigma = \{a, b, c, d, e\}$. For every $n \geq 3$ there exists a complete DFA A over Σ with n states such that*

$$\mathrm{sc}(L(A) \rightsquigarrow \{c\}) = n \cdot 2^{n-1}.$$

Proof. Choose $A = (Q, \Sigma, \delta, 0, \{0\})$ where $Q = \{0, 1, \ldots, n-1\}$ and the transitions of δ are defined by setting

- $\delta(i, a) = i + 1$ for $0 \leq i \leq n-2$, $\delta(n-1, a) = 0$;
- $\delta(0, b) = 0$, $\delta(i, b) = i + 1$ for $1 \leq i \leq n-2$, $\delta(n-1, b) = 0$;
- $\delta(0, c) = 1$, $\delta(i, c) = i$ for $1 \leq i \leq n-1$;
- $\delta(0, d) = 0$, $\delta(1, d) = 1$, $\delta(i, d) = i + 1$ for $2 \leq i \leq n-2$, $\delta(n-1, d) = 0$;
- $\delta(0, e) = \delta(1, e) = 1$, $\delta(i, e) = i + 1$ for $2 \leq i \leq n-2$, $\delta(n-1, e) = 0$.

The DFA A is depicted in Figure 1.

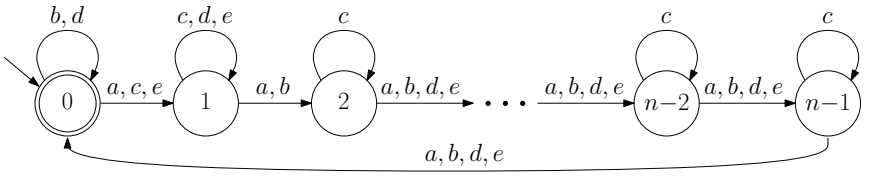

Fig. 1. The complete DFA A used in the proof of Lemma 3

Let $B = (P, \Sigma, \gamma, p_0, F_B)$ be the complete DFA recognizing the language $L(A) \rightsquigarrow \{c\}$ that is constructed as in the proof of Lemma 1. Since the deleted language $\{c\}$ consists of only one string, B has $n \cdot 2^{n-1}$ states and in order to prove the lemma it is sufficient to show that all states of B are reachable from the initial state $p_0 = (0, \{1\})$ and all states of B are pairwise inequivalent.
Claim 1. All states $(0, R) \in P$ where $R \supseteq \delta(0, \{c\}) = \{1\}$ are reachable from $(0, \{1\})$.

We prove the claim by induction on the cardinality of R. In the base case $|R| = 1$ there is nothing to prove because $R = \{1\}$ is the only set satisfying the required condition on R.

Inductively, now assume that the claim holds for all sets R of cardinality $1 \leq k < n$, that is, all states $(0, R)$ where $1 \in R$ and $|R| \leq k$ are reachable. Now consider a state of P,

$$\mathbf{u} = (0, \{1, i_1, \ldots i_k\}), \;\; 1 < i_1 < i_2 < \cdots < i_{k-1} \text{ and } (i_{k-1} < i_k \text{ or } i_k = 0).$$

Above the elements $i_1, \ldots, i_k$ are listed in increasing order, except that 0 is considered the largest element. By the inductive assumption the state

$$\mathbf{u}' = (0, \{1, i_2 - i_1 + 1, i_3 - i_1 + 1, \ldots, i_k - i_1 + 1\})$$

is reachable. Here the arithmetic operations are done modulo n. If $i_k = 0$, above $i_k - i_1 + 1$ stands for $n - i_1 + 1$. Now

$$\gamma(\mathbf{u}', b) = (0, \{1, 2, i_2 - i_1 + 2, i_3 - i_1 + 2, \ldots, i_k - i_1 + 2\}),$$

because the transition where the first component enters 0 adds 1 to the second component and otherwise the transition on b cycles "upwards" the states of $\{1, i_2 - i_1 + 1, i_3 - i_1 + 1, \ldots, i_k - i_1 + 1\}$. (Note that this set does not contain the element 0.) Next applying to the state $\gamma(\mathbf{u}', b)$ $i_1 - 2$ times a transition on d we reach $\mathbf{u}$. Note that transitions on d "cycle upwards" the states in $\{2, 3, \ldots, n-1\}$, and keep the states 0 and 1 stationary, and do not add (in the transition relation of B), new elements to the second component of the state.

Next we show that all states (r, R), $R \supseteq \delta(r, \{c\})$, are reachable. We need to consider only cases where $r \neq 0$ (because the case $r = 0$ was handled in Claim 1 above) and, hence, $\delta(r, \{c\}) = \{r\}$. Consider an arbitrary state

$$\mathbf{v} = (i_j, \{i_1, i_2, \ldots, i_{j-1}, i_j, i_{j+1}, \ldots, i_k\}), \quad 0 \leq i_1 < i_2 < \cdots < i_k \leq n - 1.$$

For technical reasons we need to use a slightly different argument depending on whether the difference between i_{j+1} and i_j, as well as, between i_j and i_{j-1} is exactly one. We divide the following argument into three cases.

(i) Case where $i_j \neq i_{j-1} + 1$: This is the case where the set in the second component of $\mathbf{v}$ does not contain the element preceding i_j. By Claim 1 the state

$$\mathbf{v}' = (0, \{1, i_1 - i_j + 1, i_2 - i_j + 1, \ldots, i_{j-1} - i_j + 1, i_{j+1} - i_j + 1, \ldots, i_k - i_j + 1\})$$

is reachable. In the preceding line all quantities are computed modulo n. Now

$$\gamma(\mathbf{v}', c) = (1, \{1, i_1 - i_j + 1, i_2 - i_j + 1, \ldots, i_{j-1} - i_j + 1, i_{j+1} - i_j + 1, \ldots, i_k - i_j + 1\}).$$

Note that, because $i_j \neq i_{j-1} + 1$, the sequence $i_x - i_j + 1$, $x = 1, \ldots, j - 1, j + 1, \ldots, k$, does not contain the element 0, and hence a transition on c is the identity on these elements. In the DFA A the a-transitions just cycle through the states and hence applying $i_j - 1$ times the a-transition to state $\gamma(\mathbf{v}', c)$ we get the state $\mathbf{v}$.

(ii) Case where $i_j = i_{j-1} + 1$ and $i_{j+1} = i_j + 1$: This is the case where the set in the second component of $\mathbf{v}$ contains both the element preceding i_j and the element following i_j. By Claim 1 the state

$$\mathbf{v_1} = (0, \{i_1 - i_j, i_2 - i_j, \ldots, i_k - i_j\})$$

is reachable. Note that the second component contains the element 1 and hence $\mathbf{v_1}$ is a legal state of B. Now $\gamma(\mathbf{v_1}, a^{i_j}) = \mathbf{v}$.

(iii) Case where $i_j = i_{j-1} + 1$ and $i_{j+1} \neq i_j + 1$: This corresponds to the situation where the second component of $\mathbf{v}$ contains the element preceding i_j but does not contain the element following i_j. Here we cannot use a-transitions alone, because a state where the first component is 0 must contain 1 in the the second component. By Claim 1 the state

$$\mathbf{v_2} = (0, \{1, i_1 - i_j, i_2 - i_j, \ldots, i_{j-1} - i_j, i_{j+1} - i_j, \ldots, i_k - i_j\})$$

is reachable and

$$\gamma(\mathbf{v_2}, e) = (1, \{1, i_1 - i_j + 1, i_2 - i_j + 1, \ldots, i_k - i_j + 1\}).$$

Here we need the fact that $i_{j+1} \neq i_j + 1$ and hence an e-transition adds one to the state $i_{j+1} - i_j$. (Note also that $i_{j-1} - i_j = n - 1$ and, consequently, beginning with a c-transition as in case (i) above would not work, because the second component at the end would not contain i_{j-1}.)
Applying $i_j - 1$ a-transitions to the state $\gamma(\mathbf{v_2}, e)$ we get $\mathbf{v}$.

We have shown that all states of B are reachable and it remains to show that they are all pairwise inequivalent.

First consider two states $(r_1, R_1), (r_2, R_2) \in P$ where $R_1 \neq R_2$. Without loss of generality we can find $s \in R_1 - R_2$ since the other possibility is completely symmetric. If $s = 0$ then (r_1, R_1) is a final state and (r_2, R_2) is a nonfinal state of B. Thus it is sufficient to consider cases $s \in \{1, 2, \ldots, n-1\}$. Now $\gamma((r_1, R_1), a^{n-s}) \in F_B$ because the string a^{n-s} takes the state $s \in R_1$ to the element 0. We show that $\gamma((r_2, R_2), a^{n-s}) \notin F_B$. Since $s \notin R_2$, the string a^{n-s} does not take any element of R_2 to the element 0 (which is the only final state of A). We note that $r_2 \neq s$ because from the definition of legal states of B we know that $\delta(r_2, c)$ must be an element of R_2 (and $\delta(s, c) = s$ when $1 \leq s \leq n-1$). Also, if during the computation on a^{n-s} starting from (r_2, R_2), the transitions of γ add the element 1 to the second component when the first component becomes 0, then the added element 1 cannot cycle through all states to reach the final state 0 because after adding the element 1 there remains at most $n - s - 1$ input symbols and $s \geq 1$.

Second, consider two states $(r_1, R_1), (r_2, R_2) \in P$, where $r_1 \neq r_2$. Due to symmetry between r_1 and r_2 we can assume that $r_2 \neq 0$.

(i) Case where $r_1 = 0$ and $r_2 \neq n - 1$: Since $r_2 \neq r_1$, we have $r_2 \neq 0$ and we note that $\gamma((0, R_1), b) = (0, R_1')$ where $1 \in R_1'$ (because the self-loop on state 0 adds the element 1 to the second component) and $\gamma((r_2, R_2), b) = (r_2 + 1, R_2')$. Here $1 \notin R_2'$ because no transition of A labeled by b reaches the state 1 and also since $r_2 + 1 \neq 0$ the transition cannot add the element 1 to the second component. Since the second components are distinct sets we know that the states $(0, R_1')$ and $(r_2 + 1, R_2')$ are distinguishable.

(ii) Case where $r_1 = 0$ and $r_2 = n - 1$: We note that $\gamma((0, R_1), a) = (1, R_1')$ and $\gamma((n-1, R_2), a) = (0, R_2')$. The states $(1, R_1')$ and $(0, R_2')$ are inequivalent by case (i) above.

(iii) Case where $r_1 \neq 0$ and $r_2 \neq 0$: By cycling with $n - r_1$ a-transitions we get states $(0, R'_1)$ and $(n - r_1 + r_2, R'_2)$. If $r_1 - r_2 \neq 1$, this case was covered in (i) above and, if $r_1 - r_2 = 1$, this case was covered in (ii) above.

Above (i)–(iii) cover all cases where $r_1 \neq r_2$ and $r_2 \neq 0$. This concludes the proof of the lemma. □

Now by combining Lemmas 1 and 3 we get a tight state complexity bound for deletion.

Theorem 1. *For languages $L_1, L_2 \subseteq \Sigma^*$ where L_1 is regular,*

$$\mathrm{sc}(L_1 \rightsquigarrow L_2) \leq \mathrm{sc}(L_1) \cdot 2^{\mathrm{sc}(L_1)-1}.$$

For every $n \geq 3$ there exists a regular language L_1 over a five-letter alphabet with $\mathrm{sc}(L_1) = n$ and a singleton language L_2 such that in the above inequality we have equality.

4.2 Lower Bound for Incomplete DFAs

We show that the state complexity upper bound for incomplete DFAs from Lemma 2 can be reached by DFAs defined over a five-letter alphabet. Based on the observations made in Corollary 1, as the language of deleted strings, we use a singleton set $\{c\}$ where, furthermore, a c-transition is defined only for one state of the DFA recognizing L_1. The conditions of Corollary 1 do not force L_2 to be a singleton set, however, the conditions indicate that a construction may be simpler to achieve using a singleton set. (The proof of Lemma 4 is omitted due to the limitation on the number of pages.)

Lemma 4. *Let $\Sigma = \{a, b, c, d, e\}$. For every $n \geq 4$ there exists a regular language $L_1 \subseteq \Sigma^*$ recognized by an incomplete DFA with n states such that*

$$\mathrm{isc}(L_1 \rightsquigarrow \{c\}) = (n + 1) \cdot 2^n - (2^{n-1} + 2).$$

As a result of Lemma 4 we conclude that also the upper bound for the size of an incomplete DFA for the language $L_1 \rightsquigarrow L_2$ given in Lemma 2 is tight.

Theorem 2. *For languages $L_1, L_2 \subseteq \Sigma^*$ where L_1 is regular,*

$$\mathrm{isc}(L_1 \rightsquigarrow L_2) \leq (\mathrm{isc}(L_1) + 1) \cdot 2^{\mathrm{isc}(L_1)} - (2^{\mathrm{isc}(L_1)-1} + 2).$$

For every $n \geq 4$ there exists a language L_1 over a five-letter alphabet recognized by an incomplete DFA with n states and a singleton language L_2 such that in the above inequality we have an equality.

5 Conclusion and Further Work

We have established tight state complexity bounds for the deletion of an arbitrary language L_2 from a regular language L_1 both in the case where L_1 and $L_1 \rightsquigarrow L_2$ are represented by complete DFAs and when they are represented by incomplete DFAs. Furthermore, in the lower bound construction the deleted language can be chosen to be a singleton set consisting of a string of length one. This result is in some sense the strongest possible because deleting the empty string from L_1 yields just L_1.

Roughly speaking, in the upper bound constructions given in Section 3, the DFA B for $L_1 \rightsquigarrow L_2$ is based only on the DFA A for L_1 and B depends on L_2 only by way of the transitions the strings of L_2 define on A. This causes that the transitions in parts of B that, respectively, simulate the original DFA A and the computation of A after a string was deleted are closely related and, perhaps partly because of this reason, the lower bound constructions that match the upper bound (respectively, for complete and for incomplete DFAs) are fairly involved. For the constructions we used a five-letter alphabet. The alphabet size could likely be reduced, but this would lead to considerably more complicated proofs of correctness. Furthermore, it does not seem clear whether the general upper bound can be reached using a binary alphabet. Note that for a unary alphabet, deletion coincides with right-quotient and the state complexity is known to be n [29].

More general types of deletion operations have been considered within the context of deletion along trajectories [3, 19]. Our "ordinary" deletion operation is defined by the set of trajectories $i^*d^*i^*$ and the left-quotient and right-quotient operations are defined, respectively, by the sets of trajectories d^*i^* and i^*d^*. The set of trajectories $d^*i^*d^*$ defines the *bipolar deletion* operation [3, 16, 18]. The language $L_1 \rightsquigarrow_{d^*i^*d^*} L_2$ consists of all strings v such that for some string $u = u_1u_2 \in L_2$, the string u_1vu_2 is in L_1. From [3, 18] it is known that bipolar deletion preserves regularity but the state complexity bound given by these results is not optimal. Differing from deletion, the state complexity of bipolar deletion would need to depend on the size of DFAs for both of the argument languages.

References

1. Câmpeanu, C., Salomaa, K., Yu, S.: Tight lower bound for the state complexity of shuffle of regular languages. J. Autom. Lang. Comb. 7, 303–310 (2002)
2. Cui, B., Gao, Y., Kari, L., Yu, S.: State complexity of combined operations with two basic operations. Theoret. Comput. Sci. 437, 98–107 (2012)
3. Domaratzki, M.: Deletion along trajectories. Theoret. Comput. Sci. 320, 293–313 (2004)
4. Domaratzki, M., Okhotin, A.: State complexity of power. Theoret. Comput. Sci. 410, 2377–2392 (2009)
5. Domaratzki, M., Salomaa, K.: State complexity of shuffle on trajectories. J. Automata, Languages and Combinatorics 9, 217–232 (2004)
6. Eom, H.-S., Han, Y.-S.: State complexity of combined oeprations for suffix-free regular languages. Theoret. Comput. Sci. 510, 87–93 (2013)

7. Eom, H.-S., Han, Y.-S., Jirásková, G.: State complexity of basic operations on non-returning regular languages. In: Jurgensen, H., Reis, R. (eds.) DCFS 2013. LNCS, vol. 8031, pp. 54–65. Springer, Heidelberg (2013)
8. Eom, H.-S., Han, Y.-S., Salomaa, K.: State complexity of k-union and k-intersection for prefix-free regular languages. In: Jurgensen, H., Reis, R. (eds.) DCFS 2013. LNCS, vol. 8031, pp. 78–89. Springer, Heidelberg (2013)
9. Gao, Y., Kari, L.: State complexity of star of union and square of union on k regular languages. Theoret. Comput. Sci. 499, 38–50 (2013)
10. Gao, Y., Piao, X.: State complexity of insertion, manuscript in preparation (2014)
11. Gao, Y., Yu, S.: State complexity and approximation. Int. J. Found. Comput. Sci. 23(5), 1085–1098 (2012)
12. Han, Y.-S., Salomaa, K.: State complexity of basic operations on suffix-free regular languages. Theoret. Comput. Sci. 410, 2537–2548 (2009)
13. Holzer, M., Kutrib, M.: Descriptional and computational complexity of finite automata — A survey. Inf. Comput. 209, 456–470 (2011)
14. Jirásek, J., Jiráskova, G., Szabari, A.: State complexity of concatenation and complementation. Internat. J. Foundations Comput. Sci. 16, 511–529 (2005)
15. Jirásková, G., Shallit, J.: The state complexity of star-complement-star. In: Yen, H.-C., Ibarra, O.H. (eds.) DLT 2012. LNCS, vol. 7410, pp. 380–391. Springer, Heidelberg (2012)
16. Kari, L.: On language equations with invertible operations. Theoret. Comput. Sci. 132, 129–150 (1994)
17. Kari, L.: On insertion and deletion in formal languages. PhD thesis. University of Turku (1991)
18. Kari, L.: Deletion operations: Closure properties. Internat. J. Comput. Math. 52, 23–42 (1994)
19. Kari, L., Sosik, P.: Aspects of shuffle and deletion on trajectories. Theoret. Comput. Sci. 332, 47–61 (2005)
20. Kutrib, M., Pighizzini, G.: Recent trends in descriptional complexity of formal languages. Bulletin of the EATCS 111, 70–86 (2013)
21. Lupanov, O.B.: A comparison of two types of finite sources. Problemy Kibernetiki 9, 328–335 (1963)
22. Maslov, A.N.: Estimates on the number of states of finite automata. Soviet Math. Dokl. 11, 1373–1375 (1970)
23. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars and formal systems. In: Proc. SWAT (FOCS), pp. 188–191. IEEE Computer Society (1971)
24. Moore, F.R.: On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. IEEE Transactions on Computers C-20, 1211–1214 (1971)
25. Rampersad, N.: The state complexity of L^2 and L^k. Inform. Proc. Letters 98, 231–234 (2006)
26. Salomaa, A., Salomaa, K., Yu, S.: State complexity of combined operations. Theoret. Comput. Sci. 383, 140–152 (2007)
27. Shallit, J.: A Second Course in Formal Languages and Automata Theory. Cambridge University Press (2009)
28. Wood, D.: Theory of Computation. John Wiley & Sons, Inc., New York (1987)
29. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. I, pp. 41–110. Springer (1997)
30. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theoret. Comput. Sci. 125, 315–328 (1994)