Universal Disjunctive Concatenation and Star

Nelma Moreira[1], Giovanni Pighizzini[2], Rogério Reis[1]

[1] Centro de Matemática e Faculdade de Ciências da Universidade do Porto, Portugal
{nam,rvr}@dcc.fc.up.pt*
[2] Dipartimento di Informatica, Università degli Studi di Milano, Italy
pighizzini@di.unimi.it**

Abstract. Two language operations that can be expressed by suitably combining complement with concatenation and star, respectively, are introduced. The state complexity of those operations on regular languages is investigated. In the deterministic case, optimal exponential state gaps are proved for both operations. In the nondeterministic case, for one operation an optimal exponential gap is also proved, while for the other operation an exponential upper bound is obtained.

1 Introduction

In a recent paper, we investigated automata with partially specified behaviors, shortly called *don't care automata* (dcFA) [**?**]. These devices are defined as standard nondeterministic finite state automata, with the only difference that they have two sets of final states: the set of *accepting states* and the set of *rejecting states*. In this way, a dcFA A defines two languages: the accepted language $\mathcal{L}^{\oplus}(A)$ and the rejected language $\mathcal{L}^{\ominus}(A)$. It is required that these two languages are disjoint.

Don't care automata can be interesting in situations where it is not necessary to fix the behavior on each possible string, because, for instance, some strings will never be received by the automaton (e.g., when the input of the automaton is generated by a source which produces strings in a certain format), or because for other reasons the answer of the automaton on some strings is not interesting. In the same paper, we studied the optimal reductions, in terms of states, of such devices to compatible deterministic automata; namely to standard deterministic automata which "agree" with the behavior of the given don't care automata.

Triggered by a paper published in 1994 [**?**], a lot of work has been conducted in the last 20 years to study the state complexity of operations on finite automata. Inspired by this research, we started to investigate how standard operations (boolean, concatenation and Kleene star) could be extended to dcFAs. An obvious requirement is that an operation extended to dcFAs matches

* Authors partially funded by the European Regional Development Fund through the programme COMPETE and by the Portuguese Government through the FCT under project UID/MAT/00144/2013.

** Author partially supported by MIUR under the project PRIN "Automi e Linguaggi Formali: Aspetti Matematici e Applicativi", code H41J12000190001.

the original operation, if the behavior of the automaton is fully specified. For instance, considering union, a string should be accepted when it is accepted by at least one of the two given automata and should be rejected when it is rejected by both automata. Considering don't care values, it is quite obvious how to extend this behavior, as depicted in the table to the right, where *yes* and *no* represent acceptance and rejection, respectively, and *?* represents an unspecified behavior. Similar tables can be filled in for intersection and complement. Notice that this is related to *three-valued logic* [**?**].

	no	*?*	*yes*
no	no	?	yes
?	?	?	yes
yes	yes	yes	yes

For the other regular operations, concatenation and star, the situation is slightly more complicated and, probably, more interesting. Let us consider two dcFAs A and B on the same input alphabet Σ that have completely specified behaviors, i.e., $\mathcal{L}^{\oplus}(A) \cup \mathcal{L}^{\ominus}(A) = \mathcal{L}^{\oplus}(B) \cup \mathcal{L}^{\ominus}(B) = \Sigma^{\star}$. A dcFA C for concatenation should accept all the strings which can be obtained by concatenating strings accepted by A and B, i.e., $\mathcal{L}^{\oplus}(C) = \mathcal{L}^{\oplus}(A)\mathcal{L}^{\oplus}(B)$, and should reject all the strings which cannot be obtained in this way; namely, all the strings w such that for each factorization $w = uv$ either $u \notin \mathcal{L}^{\oplus}(A)$ or $v \notin \mathcal{L}^{\oplus}(B)$. Since in this case $\mathcal{L}^{\ominus}(A)$ and $\mathcal{L}^{\ominus}(B)$ are the complement of $\mathcal{L}^{\oplus}(A)$ and $\mathcal{L}^{\oplus}(B)$, respectively, this is equivalent to saying that for each factorization $w = uv$ either $u \in \mathcal{L}^{\ominus}(A)$ or $v \in \mathcal{L}^{\ominus}(B)$.

This leads to consider a new language operation, that we call *universal disjunctive concatenation* and, in a similar way, starting from the star, another new operation called *universal disjunctive star*. This paper is devoted to investigating the state complexity of these two operations in both deterministic and nondeterministic cases. Using the fact that these two operations can be expressed by combining complement with concatenation and star, respectively, we prove that in the deterministic case their state complexity is exponential.

We deepen this investigation by considering the nondeterministic case. We prove that given two nondeterministic automata (NFAs) A and B with m and n states, there exists an NFA accepting the universal disjunctive concatenation of the languages accepted by A and B, with at most 2^{m+n} states. Furthermore, the exponential gap cannot be reduced in the worst case.

We also prove that for each NFA A with n states there exists an NFA with no more than 2^n states accepting the universal disjunctive star of $\mathcal{L}(A)$.

In the final part of the paper, we shortly discuss the state complexity of operations on don't care automata.

2 Universal Disjunctive Concatenation

Given a language L over an alphabet Σ, let us denote by $\mathsf{pref}(L)$ the language consisting of *all prefixes* of strings in L. The set of all nonempty prefixes of L, i.e., $\mathsf{pref}(L) \setminus \{\varepsilon\}$, is denoted by $\mathsf{pref}_{+}(L)$. By $\mathsf{p}(L)$ we denote the *prefix-closed interior* of L; namely the largest subset of L which is prefix closed.

Similar definitions can be given by considering suffixes. Hence, we let $\mathsf{suff}(L)$ denote the set of all suffixes of strings in L, $\mathsf{suff}_{+}(L)$ denote the set $\mathsf{suff}(L) \setminus \{\varepsilon\}$,

and $\mathsf{s}(L)$ denote the *suffix-closed interior* of L, namely the largest subset of L which is suffix closed.

The complement of L will be denoted by $\overline{L}$.

Definition 1. *Let L_1 and L_2 be languages over an alphabet Σ. The* universal disjunctive concatenation *of L_1 and L_2 is the language $L_1 \circledcirc L_2$ defined as*

$$L_1 \circledcirc L_2 = \{w \in \Sigma^\star \,|\, \forall x_1, x_2,\ w = x_1 x_2 \ \Rightarrow\ (x_1 \in L_1 \vee x_2 \in L_2)\}.$$

Example 2. Given $\Sigma = \{a, b\}$, consider L_1 the set of all strings containing an even number of occurrences of a and $L_2 = a(a+b)^\star$. Then, $L_1 \circledcirc L_2 = (aa+b)^\star$. Given a string w, let us number the occurrences of the letter a in w, starting from 1. Each prefix of w containing an even number of a's belongs to L_1. If a prefix ends with an odd numbered a then it does not belong to L_1. Thus, in order to have $w \in L_1 \circledcirc L_2$, the remaining suffix should belong to L_2; hence, it should start with an a. Hence each odd numbered a should be immediately followed by another a.

We point out the role of the alphabet Σ we are considering in the definition of the operation $\circledcirc$. For instance, given $L = a^\star$, if $\Sigma = \{a\}$ then $L \circledcirc L = a^\star = L$. However, if the alphabet we are considering is $\Sigma = \{a, b\}$, then $L \circledcirc L = a^\star + a^\star b a^\star$; namely, L is the set of all strings which contain at most one occurrence of the letter b. This is due to the fact that this operation involves, in some sense, complementation, as we will explain below. We can observe that, for $\Sigma = \{a, b\}$, $\overline{L}$ is the set of strings that contain at least one occurrence of the letter b and, hence, $\overline{L}\,\overline{L}$ is the set of all strings that contain at least two occurrences of b. Thus, its complement coincides with $L \circledcirc L$.

Actually, this is a general property. In fact, just by using the definition of $\circledcirc$, we can observe that given two languages L_1 and L_2, $w \notin L_1 \circledcirc L_2$ if and only if there are two strings u, v such that $w = uv$, $u \notin L_1$, and $v \notin L_2$. Thus,

$$L_1 \circledcirc L_2 \ = \ \overline{\overline{L_1}\,\overline{L_2}}\,. \tag{1}$$

As a consequence, the operation $\circledcirc$ preserves regularity; namely, if L_1 and L_2 are regular, then $L_1 \circledcirc L_2$ is regular too. A special case of this operation when $L_1 = \emptyset$ was studied by Birget [**?**]. We now study some basic properties of the $\circledcirc$ operation.

Proposition 3. *The operation $\circledcirc$ is associative, i.e., $L_1 \circledcirc (L_2 \circledcirc L_3) = (L_1 \circledcirc L_2) \circledcirc L_3$, for all languages L_1, L_2, L_3.*

Because $\circledcirc$ is associative it makes sense to write $L_1 \circledcirc L_2 \circledcirc L_3$ and it is not difficult to realize that

$$L_1 \circledcirc L_2 \circledcirc L_3 = \{w \mid \forall x, y, z,\ w = xyz \ \Rightarrow\ x \in L_1 \ \vee\ y \in L_2 \ \vee\ z \in L_3\}\,.$$

Proposition 4. *The $\circledcirc$ operation has an identity element. For any language L, $\Sigma^+ \circledcirc L = L \circledcirc \Sigma^+ = L$.*

Proof. Immediate consequence of Equation (1), observing that $\overline{\Sigma^+} = \{\varepsilon\}$. □

We note there are no languages L_1, L_2, apart from Σ^+, such that $L_1 \circledcirc L_2 = \Sigma^+$, i.e. $\circledcirc$ has no nontrivial inverses. In general, it is easy to see that

Proposition 5. *Let L_1, L_2 be languages. If $\varepsilon \notin L_1$, then $L_1 \circledcirc L_2 \subseteq L_2$ and $L_2 \circledcirc L_1 \subseteq L_2$.*

Proposition 6. *Let L be a language. Then $\emptyset \circledcirc L = \mathsf{s}(L)$ and $L \circledcirc \emptyset = \mathsf{p}(L)$.*

Proof. From the definition of $\circledcirc$, it easily follows that a string w belongs to $\emptyset \circledcirc L$ only if each suffix of w belongs to L. Hence, $\emptyset \circledcirc L \subseteq \mathsf{s}(L)$. Conversely, if $w \in \mathsf{s}(L)$ then each suffix of w belongs to L, which would imply that $w \in \emptyset \circledcirc L$. In a similar way, it can be proved that $L \circledcirc \emptyset = \mathsf{p}(L)$. □

Proposition 7. *Let $L, X \subseteq \Sigma^\star$ be languages.*

(a) If L is prefix closed then $L \subseteq L \circledcirc X$.
(b) $\mathsf{pref}(L) \subseteq \mathsf{pref}(L) \circledcirc L$.
(c) If $\varepsilon \notin L$ then $\mathsf{pref}(L) \circledcirc L = \mathsf{pref}(L)$.
(d) $\mathsf{pref}_+(L) \circledcirc L = L$.

Proof. (a) is trivial.

(b) immediately follows from (a).

(c) Since $w = w\varepsilon$ and $\varepsilon \notin L$, from $w \in \mathsf{pref}(L) \circledcirc L$, we obtain $w \in \mathsf{pref}(L)$. Hence $\mathsf{pref}(L) \circledcirc L \subseteq \mathsf{pref}(L)$. The converse inclusion is given in (b).

(d) Let $w \in L$, then for every x, y such that $xy = w$, $x \in \mathsf{pref}_+(L)$ unless $x = \varepsilon$, but in that case $y = w \in L$. On the other hand, if $w \in \mathsf{pref}_+(L) \circledcirc L$ then, as $\varepsilon w = w$, necessarily $w \in L$. □

Notice that if $\varepsilon \in L$ then $\mathsf{pref}(L) \circledcirc L$ could differ from both L and $\mathsf{pref}(L)$. For instance, given $\Sigma = \{a\}$, for $L = \{\varepsilon, aa\}$, we have $\mathsf{pref}(L) \circledcirc L = \{\varepsilon, a, aa, aaa\}$.

We can prove properties similar to those in Proposition 7, considering suffixes:

Proposition 8. *Let $L, X \subseteq \Sigma^* $ be languages.*

(a) If L is suffix closed then $L \subseteq X \circledcirc L$.
(b) $\mathsf{suff}(L) \subseteq L \circledcirc \mathsf{suff}(L)$.
(c) If $\varepsilon \notin L$ then $L \circledcirc \mathsf{suff}(L) = \mathsf{suff}(L)$.
(d) $L \circledcirc \mathsf{suff}_+(L) = L$.

Proposition 9. *If L_1 is prefix closed and L_2 is suffix closed, then $L_1 L_2 \subseteq L_1 \circledcirc L_2$.*

Proof. Suppose $w \in L_1 L_2$. Let $x \in L_1$ and $y \in L_2$ be such that $w = xy$. Then for all strings u, v verifying $w = uv$ either u is a prefix of x, thus implying $u \in L_1$, or v is a suffix of y, thus implying $v \in L_2$. This allows to conclude that $w \in L_1 \circledcirc L_2$. □

Our previous example with $\Sigma = \{a, b\}$ and $L_1 = L_2 = a^\star$ shows that the inclusion proved in Proposition 9 can be proper.

Now we are going to study the state complexity of the operation $\circledcirc$. First of all, by using results on the state complexity of concatenation [**?**], we can obtain the following bound:

Theorem 10. *For all integers $m, n \geq 2$, let A' be an m-state DFA and A'' an n-state DFA. Then any DFA that accepts $\mathcal{L}(A') \circledcirc \mathcal{L}(A'')$ needs at most $m2^n - (m - f)2^{n-1}$ states, where f is the number of* final *states of A'. Futhermore, this bound is tight.*

Proof. It follows immediately from Equation (1) that the state complexity of $\circledcirc$ coincides with the state complexity of concatenation because the state complexity of the complement of a language L coincides with the state complexity of L. Hence, the upper bound follows from Theorem 2.3 in [**?**], after switching the role of final and nonfinal states in the automaton A', due to the complementation. The lower bound also derives from a result in the same paper (Theorem 2.1). □

The investigation of the state complexity of $\circledcirc$ is now deepened by proving that the bound in Theorem 10 cannot be reduced if we allow the resulting automaton to be nondeterministic. To this aim, for each integer $n \geq 1$, let us consider the following language over $\Sigma = \{a, b\}$:

$$L_n = (a(a+b)^{n-1})^\star(\varepsilon + a(a+b)^{<n}) + (b(a+b)^{n-1})^\star(\varepsilon + b(a+b)^{<n}), \quad (2)$$

where $(a+b)^{<n}$ denotes *less than* n repetitions of $a+b$. In other words, a string w belongs to L_n if and only if the same symbol occurs in all positions $in+1$ of w, with $i \geq 0$ and $in + 1 \leq |w|$.

Theorem 11. *Let L_n be the language defined in (2). Then:*

(a) The minimum DFA accepting L_n has $2n + 2$ states.
(b) $\mathsf{s}(L_n) = \{x^k y \mid$ for some $x \in \{a, b\}^n, k \geq 0, y \in \mathsf{pref}(x)\}$.
(c) Each NFA accepting $\mathsf{s}(L_n)$ requires at least 2^n states.

Proof. (a) A DFA accepting L_n is depicted in Fig. 1. By a standard distinguishability argument, it can be proved that it is minimal.

(b) From the definition of L_n, we can observe that a string $w \in \mathsf{s}(L_n)$ if and only if each two symbols of w at distance n, i.e., with $n - 1$ symbols in between, are equal. This implies that w consists of a prefix x of length n which is repeated a certain number of times and a suffix y of length $< n$, which is a prefix of x.

(c) Consider the set $S = \{(x, x) \mid x \in \{a, b\}^n\}$. From (b), it follows that for $x, y \in \{a, b\}^n$, $xx \in \mathsf{s}(L_n)$ and $xy \notin \mathsf{s}(L_n)$. Hence, S is a *fooling set* for $\mathsf{s}(L_n)$ and each NFA accepting it requires at least $\#S$ states [**?**]. □

As a consequence of Theorem 11 we can now conclude that the exponential upper bound given in Theorem 10 cannot be reduced, even if the resulting automaton is nondeterministic.

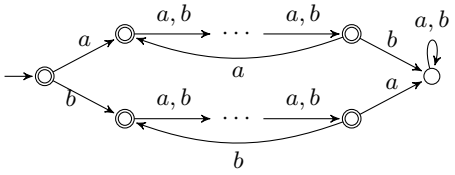

Fig. 1. The minimum DFA accepting L_n.

Theorem 12. *Let A be the 1-state automaton accepting the empty language. For each integer n there exists an n-state DFA B accepting a language defined over a binary alphabet such that each NFA accepting $\mathcal{L}(A) \circledcirc \mathcal{L}(B)$ requires at least $2^{\lfloor (n-2)/2 \rfloor}$ states.*

Proof. Given $k = \lfloor (n-2)/2 \rfloor$, consider the language L_k, defined according to (2). If n is even, we choose as DFA B the minimum automaton accepting L_k. If n is odd, we obtain B by splitting the trap state of the minimum automaton accepting L_k in two states (this can be done with a simple change in the transition graph in Figure 1). This adds one extra state, without changing the accepted language.

In both cases, the DFA B has n states and recognizes $L_{\lfloor (n-2)/2 \rfloor} = L_k$. From Proposition 6, it turns out that $\emptyset \circledcirc L_k = \mathsf{s}(L_k)$. Hence, by Theorem 11, each NFA accepting $\emptyset \circledcirc L_k$ requires 2^k states. □

We point out that Theorem 12 improves a similar gap proved in [**?**], by considering a three-letter alphabet.

Now, we further investigate the nondeterministic case, by studying the state complexity of $\circledcirc$ in the case of nondeterministic automata. The upper bound in Theorem 10 is derived from (1) which uses a double complementation. This leads to a double exponential upper bound on the state complexity, when the given automata are nondeterministic. However, it produces a deterministic automaton. It could be interesting to see if the state complexity can be reduced, if we want to derive a *nondeterministic* automaton. This could be done using *alternating automata* [**?**]. However, in the next result we present a direct construction.

Theorem 13. *For each integers $m, n \geq 1$, let A' be an m-state NFA and A'' an n-state NFA, then there is an NFA that accepts $\mathcal{L}(A') \circledcirc \mathcal{L}(A'')$ with at most 2^{m+n} states.*

Proof. Let $A' = \langle Q', \Sigma, \delta', I', F' \rangle$, $A'' = \langle Q'', \Sigma, \delta'', I'', F'' \rangle$, $L' = \mathcal{L}(A')$, and $L'' = \mathcal{L}(A'')$. We define an NFA $A = \langle Q, \Sigma, \delta, i, F \rangle$ which accepts the language $L' \circledcirc L''$. First we informally explain how A works, in the case the two given automata A' and A'' are deterministic. Let i' and i'' be their initial states. Given a string $w \in \Sigma^\star$, in order to test if $w \in L' \circledcirc L''$, the automaton A has to check that for each prefix of w which is rejected by A' the corresponding suffix is accepted by A''. To this aim, while reading w, A simulates the deterministic

control of A'. Each time that in the simulation A' reaches a nonfinal state — namely the prefix read so far does not belong to L' — the automaton A starts the simulation of a computation of A'' to check if the remaining suffix belongs to L''. In this way, A works by simulating in parallel a computation of A' on the given input and, for each suffix corresponding to a prefix not in L', one computation of A''. At the end, A must verify that all the computations are accepting.

The automaton A is implemented using states (q, α) where $q \in Q'$ and $\alpha \subseteq Q''$. The first component is used for the simulation of A', the second one keeps track of the states reached by A'' on the suffixes under examination. So, if the initial state i' of A' is final, then A starts its computation in $(i', \emptyset)$. Otherwise, since $\varepsilon \notin L'$, A needs to verify that all the input belongs to L'' and hence it starts the computation in $(i', \{i''\})$. When in the state (q, α) the automaton A reads a symbol σ, it moves to the state (p, β) where $p = \delta'(q, \sigma)$ and β contains all the states that are reached by states in α reading σ. In this way, the second component continues the inspection of input suffixes. However, if $p \notin F'$ then A needs to simulate A'' on the incoming input suffix. To this aim, in this case, β also contains the state i''. At the end of the computation, A has to verify that all the suffixes under examination are accepted by A''. Hence, if (q, α) is the state reached at the end of the computation, all states in α should belong to F''. However, since we also have to verify that either the input w belongs to L' or $\varepsilon \in L''$, we should additionally ask if either $q \in F'$ or $i'' \in F''$. Notice that the resulting automaton A is deterministic. The resulting DFA A is formally defined with $Q = Q' \times 2^{Q''}$; $i = (i', \emptyset)$ if $i' \in F'$ and $i = (i', \{i''\})$ otherwise; $F = \{(q, \alpha) \mid \alpha \subseteq F'' \wedge (q \in F' \vee i'' \in F'')\}$ and

$$\delta((q, \alpha), \sigma) = \begin{cases} (\delta'(q, \sigma), \delta''(\alpha, \sigma)), & \text{if } \delta'(q, \sigma) \in F'; \\ (\delta'(q, \sigma), \delta''(\alpha, \sigma) \cup \{i''\}), & \text{otherwise.} \end{cases}$$

Furthermore, in the construction above, we can observe that all the states (q, α) with $q \notin F'$ and $i'' \notin \alpha$ are not reachable in A. Then the total number of states of A is at most $m2^n - (m - f)2^{n-1}$, where f is the number of final states of A', which is exactly the same number derived in Theorem 10.

When A'' is nondeterministic, the construction is slightly more complicated. In fact, on each suffix we could have different computations. We need to verify that at least one of them is accepting. To do that, we simply use nondeterministic choices. Hence, when in the state (q, α) the automaton reads a symbol σ, each possible next state (p, β) is obtained by taking $p = \delta'(q, \sigma)$ and by nondeterministically choosing a state $s \in \delta''(r, \sigma)$ to be in β for each state $r \in \alpha$. When $p \notin F'$, the automaton A needs to start a computation of A'' on the incoming suffix. Hence, a nondeterministically chosen state $i'' \in I''$ is added to β. The formal definition of A, in the case of A' deterministic and A'' nondeterministic is the following:

- $Q = Q' \times 2^{Q''}$,
- $I = \begin{cases} \{(i', \emptyset)\}, & \text{if } i' \in F'; \\ \{(i', \{i''\}) \mid i'' \in I''\}, & \text{otherwise;} \end{cases}$

– for $(q,\alpha) \in Q' \times 2^{Q''}$, $\sigma \in \Sigma$, let us consider the following set

$$\mathsf{next}(\alpha,\sigma) = \{\gamma \in 2^{Q''} \mid \exists f : \alpha \to \gamma \text{ s.t. } f \text{ is surjective and } f(r) = s \Rightarrow\ s \in \delta''(r,\sigma)\},$$

the set $\delta((q,\alpha),\sigma)$ contains all the states $(p,\beta) \in Q' \times 2^{Q''}$ such that $p = \delta'(q,\sigma)$ and $\beta = \begin{cases} \gamma, & \text{if } p \in F'; \\ \gamma \cup \{i''\}, & \text{otherwise;} \end{cases}$ for some $\gamma \in \mathsf{next}(\alpha,\sigma)$, $i'' \in I''$,

– $F = \begin{cases} \{(q,\alpha) \mid q \in F', \alpha \subseteq F''\}, \text{ if } I'' \cap F'' = \emptyset; \\ \{(q,\alpha) \mid q \in Q', \alpha \subseteq F''\}, \text{ otherwise.} \end{cases}$

Finally, when even A' is nondeterministic, we can preliminary convert it into an equivalent DFA applying the subset construction, and then proceed as above described. In this case, the set of states of the resulting automaton is a subset of $2^{Q'} \times 2^{Q''}$. Hence, its cardinality is bounded by 2^{m+n}. □

From Theorem 12, it follows that the exponential upper bound in Theorem 13 cannot be reduced.

3 Universal Disjunctive Star

In this section we study the other operation we are interested in, the universal disjunctive star, defined in the following way:

Definition 14. *Let $L \subseteq \Sigma^\star$ be a language. Let $L^{\circledcirc 0} = \Sigma^+$ and $L^{\circledcirc k} = L^{\circledcirc k-1} \circledcirc L$, for each integer $k > 0$. Then we define the* universal disjunctive star *as*

$$L^{\circledast} = \bigcap_{k \geq 0} L^{\circledcirc k}.$$

Notice that by this definition, it turns out that a string $w \in L^{\circledast}$ if and only if for each factorization of w as $w = x_1 x_2 \cdots x_k$, with $k \geq 1$, at least one factor x_i belongs to the language L. We now show that we can restrict our attention to the nonempty factors. Due to space limitations, we omit the proof of the following propositions.

Proposition 15. *For each integer $i \geq 0$:*

(a) If $\varepsilon \in L$, then $\Sigma^{<i} \subseteq L^{\circledcirc i}$.
(b) If $w \in \Sigma^\star$, $|w| = i$, and $w \in L^{\circledcirc i}$, then for each $j > i$, $w \in L^{\circledcirc j}$.
(c) If $\varepsilon \notin L$ and $|w| = i$, $w \in L^{\circledcirc i}$ if and only if for each $j > i$, $w \in L^{\circledcirc j}$.

As a consequence we obtain:

Proposition 16. *Given $L \subseteq \Sigma^\star$ and $w \in \Sigma^\star$, $w \in L^{\circledast}$ if and only if, for each $0 \leq i \leq |w|$, $w \in L^{\circledcirc i}$.*

As a consequence of the previous proposition, we get that a string $w \in L^{\circledast}$ if and only if for each decomposition of w in at most $|w|$ factors, at least one of them belongs to L. Hence, we can express $L^{\circledast}$ as:

$$L^{\circledast} = \{w \in \Sigma^{\star} \mid \forall k \leq |w| \, \forall x_1, \ldots, x_k \in \Sigma^{+}, w = x_1 \cdots x_k, \exists i \leq k \;\; x_i \in L\}. \quad (3)$$

We can also observe that

$$L^{\circledast} = \overline{(\overline{L})^{\star}} \quad (4)$$

that implies that the class of regular languages is closed under of this operation. Considering Equation (4), $^{\circledast}$ is exactly the Kleene interior studied by Brzozowski et al. [**?**] when characterising the number of different languages that can occur by successive application of star and complement to a given regular language. Furthermore, using the results about the state complexity of the star [**?**, Cor. 3.2, Th. 3.3], we immediately obtain the following result:

Theorem 17. *For any n-state DFA A, $n \geq 1$, there exists a DFA A' of at most $2^{n-1} + 2^{n-2}$ states such that $\mathcal{L}(A') = (\mathcal{L}(A))^{\circledast}$. Furthermore, this bound cannot be reduced in the worst case.*

We now consider the state complexity of $^{\circledast}$ in the nondeterministic case. We prove that the upper bound remains exponential.

Theorem 18. *For any n-state NFA A, $n \geq 1$, there exists an NFA A' with at most 2^n states such that $\mathcal{L}(A') = (\mathcal{L}(A))^{\circledast}$.*

Proof. To make clearer the main argument used to define A' from A, first we discuss the construction for the deterministic case. Subsequently, we will describe the generalization to the nondeterministic case.

Let us start by supposing $A = \langle Q, \Sigma, \delta, I, F\rangle$, with $I = \{i\}$, is deterministic. We also suppose that $\varepsilon \notin L$, i.e., $i \notin F$. We describe a DFA $A' = \langle Q', \Sigma, \delta', I', F'\rangle$ which accepts the language $L^{\circledast}$.

First of all, we remind the reader that, by definition, $\varepsilon \notin L^{\circledast}$. So let us consider an input $w \in \Sigma^{+}$. The automaton A' has to verify that for each factorization of w in $k \geq 1$ nonempty factors, at least one of them belongs to the language L. In the following, a factorization satisfying such a property will be said to be *accepted*.

A' works by exploring all input factorizations in parallel computation branches that are generated while reading the input in the following way. Suppose that a string u has been read and consider a computation branch corresponding to a factorization $u = u_1 u_2 \cdots u_h$ of the input in $h \geq 1$ nonempty strings. Before reading the next input symbol γ, the computation branch is split into two branches according to the following possible factorizations of $w\gamma$:

(a) $u_1, \ldots, u_h\gamma$; namely, γ will be considered as a further symbol of the hth factor,
(b) $u_1, \ldots, u_h, \gamma$; namely, γ will be considered as a new factor.

Now suppose that the string u is the prefix of the input w that has been read so far, namely, $w = uv$, for some $v \in \Sigma^\star$. Let $w = w_1 \cdots w_k$ be a factorization of w in a computation branch which is obtained, after inspecting the suffix v, from the computation branch on the factorization $u = u_1 \cdots u_h$. Then, $w_j = u_j$ for $j = 1, \ldots, h-1$ (however u_h could be a *proper* prefix of w_h).

Suppose $u_j \in L$, for some $j < h$. In this case, the factorization of w is accepted regardless the suffix v and all factors $u_{j'}$, with $j' > j$, namely, *each* input factorization which begins by $u_1, \ldots, u_j$ is accepted and, thus, the input symbols after the factor u_j do not need to be inspected.

On the other hand, if $u_j \notin L$ for each $j < h$, then the computation branch has to test the membership to L of the factor u_h. To do this, it remembers the state $q = \delta(i, u_h)$. We observed that before reading the next input symbol, the computation branch is split in two. In the computation branch corresponding to (a), the simulation of A is continued from the state q. For (b) there are two possibilities. If $q \in F$, i.e., $u_h \in L$, then all factorizations beginning by $u_1, \ldots, u_h$ are accepted and A' does not need to consider the remaining part of the input. Thus, the second computation branch is not needed. Otherwise a computation branch corresponding to (b) is generated in order to inspect, from the initial state, a factor which begins with the next input symbol.

The automaton has to accept when each computation branch discovers that its corresponding factorization is accepted. This can happen either by testing during the computation that a factor is in L or at the end of the input by reaching a final state, so proving that the last factor is in L.

For each computation branch, the automaton A' needs only to remember the current state. This allows to implement A' by keeping in its finite state control the set of states which are reached by computation branches. More precisely, the formal definition of $A' = \langle Q', \Sigma, \delta', i', F' \rangle$ is as follows

- $Q' = 2^Q$,
- $i' = \{i\}$,
- for $\alpha \in Q'$, $\sigma \in \Sigma$, $\delta'(\alpha, \sigma) = \begin{cases} \delta(\alpha, \sigma), & \text{if } \delta(\alpha, \sigma) \subseteq F; \\ \delta(\alpha, \sigma) \cup \{i\}, & \text{otherwise;} \end{cases}$
- $F' = \{\alpha \in Q' \mid \alpha \subseteq F\}$.

In particular, in the definition of $\delta'(\alpha, \sigma)$, the part $\delta(\alpha, \sigma)$ corresponds to the computation branch (a), while the part $\{i\}$ corresponds to (b) and it is not added when after reading a symbol σ all the states are final, namely, all factors that end in σ (after the already inspected input prefix) are in L. See Figure 2.

Now, we switch to the nondeterministic case supposing that $A = \langle Q, \Sigma, \delta, I, F \rangle$ is nondeterministic, where $I = \{i\}$ with $i \notin F$. We build an NFA A' which accepts the language $L^{\circledast}$. The general working strategy of A' is similar to that described for the deterministic case: in parallel computation branches, A' inspects all different factorizations of the input. However, in this case A' needs also to simulate the nondeterministic choices of A.

In the construction for the operation $\circledcirc$, the purpose of the nondeterministic simulation was to check the membership of an input suffix to the language L''. In this construction the situation is more delicate, because we have to check

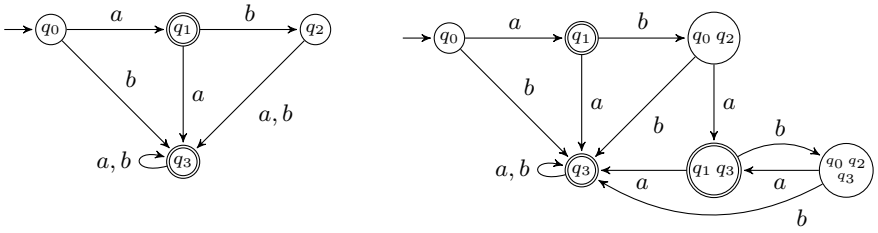

Fig. 2. Let $\Sigma = \{a, b\}$ and $L = \Sigma^\star \setminus \{ab\}$. The DFA depicted on the left recognizes $L \setminus \{\varepsilon\}$. Applying to it the construction presented in the proof of Thm. 18, the DFA depicted on the right is obtained, which accepts $L^{\circledast} = \Sigma^\star \setminus \{(ab)^\star\}$.

input factors instead of suffixes. For the initial state there is only one choice. However, from a state r we can have a nondeterministic choice which leads to the acceptance of a certain factor x and to the rejection of another factor y, and a different choice which leads to reject x and to accept y. Since A' has to inspect all different factorizations, both choices have to be considered. Thus, each time A' needs to simulate a transition from a state r on a symbol σ, a nonempty set of transitions from r on σ is nondeterministically selected, guessing that this set will lead to test that each factorization inspected in computation branches that visit the state r at that point is accepted.

The formal definition of $A' = \langle Q', \Sigma, \delta', I', F' \rangle$, obtained along these lines, is the following:

- $Q' = 2^Q$,
- $I' = I = \{i\}$,
- for $\alpha \in Q$, $\sigma \in \Sigma$, let us consider the following set:

$$\mathsf{next}(\alpha, \sigma) = \{\gamma \in 2^Q \mid \exists f : \alpha \to 2^\gamma \text{ s.t. } \gamma = \bigcup_{r \in \alpha} f(r) \wedge \\ \wedge\, \forall r \in \alpha (\emptyset \neq f(r) \subseteq \delta(r, \sigma))\}; \quad (5)$$

  the set $\delta'(\alpha, \sigma)$ contains all $\beta \in 2^Q$ such that $\beta = \begin{cases} \gamma, & \text{if } \gamma \subseteq F; \\ \gamma \cup \{i\}, & \text{otherwise;} \end{cases}$ for some $\gamma \in \mathsf{next}(\alpha, \sigma)$,
- $F' = \{\alpha \in Q' \mid \alpha \subseteq F\}$.

Finally, we study an upper bound on the state complexity. Given an NFA with n states accepting a language L, by adding one more state i, we can obtain an NFA A in the form required by our last construction, accepting the language $L \setminus \{\varepsilon\}$. Notice that $L^{\circledast} = (L \setminus \{\varepsilon\})^{\circledast}$. Thus, the number of states of the resulting automaton A' is at most 2^{n+1}. However, inspecting the construction, we can see that only at most one-half of them can be reached. In fact, from the definition of δ' we observe that each state β in the range of δ' is defined by a subset $\gamma \subseteq Q \setminus \{i\}$, taking $\beta = \gamma$ when all the states in γ are final, and $\beta = \gamma \cup \{i\}$ otherwise. This gives 2^n possibilities. However, $\gamma = \emptyset$ should not be considered because it is not reachable (see the definition of $\mathsf{next}(\alpha, \sigma)$) and, on the other hand, the

state $\{i\}$ does not belong the range of δ' but it is used at the beginning of the computation. Hence, we conclude that the total number of reachable states is at most 2^n. □

4 Conclusion

As mentioned in the Introduction, the investigation of universal disjunctive concatenation and star was initially motivated by the study of state costs of boolean and regular operations on dcFAs. To this respect, we now sketch some results that can be easily derived. All automata we consider in this discussion are nondeterministic.

First of all, the *complementation* of a dcFA can be trivially done, without increasing the number of its states, just switching the set of accepting states with the set of rejecting states. For the other operations, we can proceed as follows. From each dcFA A, we can get two NFAs $A^{\oplus}$ and $A^{\ominus}$ with at most the same number of states as A, recognizing the languages $\mathcal{L}^{\oplus}(A)$ and $\mathcal{L}^{\ominus}(A)$. Hence, given two dcFAs A and B we can easily build a dcFA C, corresponding to the *union*, by combining $A^{\oplus}$ and $B^{\oplus}$, according to the standard construction for the union, and $A^{\ominus}$ and $B^{\ominus}$, according to the standard construction for the intersection. The number of states of C is polynomial with respect to those of A and B. We can proceed in a similar way for the *intersection*.

In the case of the *concatenation*, the accepting part of C is obtained by combining $A^{\oplus}$ and $B^{\oplus}$, as in the standard construction for the concatenation. Since we are interested in a nondeterministic automaton, this uses a number of states which is bounded by the sum of the states in A and B. The rejecting part of C should recognize the language $A^{\ominus} \circledcirc B^{\ominus}$, so according to Theorems 12 and 13, it uses an exponential number of states. In a similar way, for the *star* we have a polynomial number of states for the accepting part, but an exponential upper bound for the rejecting part, according to Theorem 18.