axioms

MDPI

Article

Two Extensions of Cover Automata

Cezar Câmpeanu

School of Mathematical and Computational Sciences, University of Prince Edward Island, 550 University Ave, Charlottetown, PE C1A 4P3, Canada; ccampeanu@upei.ca

Abstract: Deterministic Finite Cover Automata (DFCA) are compact representations of finite languages. Deterministic Finite Automata with "do not care" symbols and Multiple Entry Deterministic Finite Automata are both compact representations of regular languages. This paper studies the benefits of combining these representations to get even more compact representations of finite languages. DFCAs are extended by accepting either "do not care" symbols or considering multiple entry DFCAs. We study for each of the two models the existence of the minimization or simplification algorithms and their computational complexity, the state complexity of these representations compared with other representations of the same language, and the bounds for state complexity in case we perform a representation transformation. Minimization for both models proves to be NP-hard. A method is presented to transform minimization algorithms for deterministic automata into simplification algorithms applicable to these extended models. DFCAs with "do not care" symbols prove to have comparable state complexity as Nondeterministic Finite Cover Automata. Furthermore, for multiple entry DFCAs, we can have a tight estimate of the state complexity of the transformation into equivalent DFCA.

Keywords: finite languages; deterministic finite cover automata; multiple entry automata; automata with "do not care" symbols; similarity relations

Citation: Câmpeanu, C. Two Extensions of Cover Automata. *Axioms* **2021**, *10*, 338. https://doi.org/10.3390/axioms10040338

Academic Editor: Oscar Humberto Montiel Ross

Received: 30 September 2021
Accepted: 7 December 2021
Published: 10 December 2021

1. Introduction

The concept of Cover Automata was first presented at a conference paper of Câmpeanu et al. at the Workshop on Implementations and Applications of Automata (WIAA) in Rouen (1999) [1,2] when the authors introduced a formal definition of a Deterministic Finite Cover Automaton (DFCA) and a minimization algorithm. A cover language for a language L is a superset L' of L. If L is a finite non-empty language, then the length of the longest word in L exists, and we can denote it with a natural number, l. A DFCA for a finite language L is a deterministic finite automaton (DFA) accepting a cover language for L, such that the accepted words that are not in L have their length greater than l.

During the last two decades, several papers used DCFAs for compact representation of finite languages. Other efficient minimization algorithms were also published, for example [3–8]. The concept of DFCA was also generalized to the nondeterministic version in a paper presented at AFL 2014 in Szeged by Câmpeanu [9], followed by the journal version [10].

Using nondeterminism, we can reduce the size of the automata recognizing some languages, but minimizing such automata is known to be PSPACE-complete. Therefore, several other intermediate representations of languages that maintain deterministic transitions were proposed. That is why it is a must to study these extensions in case they are applied to cover automata, which we are doing in Section 2. The first extension considered here is to enhance DFCAs with *"do not care"* symbols, thus obtaining finite cover automata with "do not care" symbols, denoted by $\diamond$-DFCAs, in other words, finite cover automata accepting partial words. Fischer and Paterson introduced partial words in [11] in 1974, and the authors in [12,13] prove that the minimization of finite automata with "do not care" symbols is NP-hard. As emphasized by Professor Solomon Marcus in [14], many

researches from other areas studied the same concept, but in a different theoretical setup with different notations. An example of such a paper related to partial words is [15], where the authors show strong connections between graph problems and pattern matching with some of the symbols in the patterns not known. In that paper, the "do not care" symbol denoted here by $\diamond$ is denoted by ϕ. We will prefer the $\diamond$ notation because most references use this notation and using a symbol that is not part of any alphabet is easier to identify. Holzer et al. in [16,17] prove that "almost all problems related to partial word automata, such as equivalence and universality, are already PSPACE-complete". Some of their proofs link non-deterministic automata problems with graph theory problems in a similar fashion as it is done in [15]. As such, because the minimization of $\diamond$-DFAs is hard, only the simplification algorithms were developed for these types of finite machines, and an example is presented in [13]. A simplification algorithm will eventually produce an equivalent automaton with less states than the input automaton, but it is not guaranteed to be minimal. In Section 3, we show that the same difficulties found for NFCA's simplification are also present for $\diamond$-DFCAs, even though $\diamond$-DFCAs can be considered a particular simpler class of NFCAs. We show a simplification algorithm for $\diamond$-DFCAs that has a better time complexity than the one presented for $\diamond$-DFAs in [13].

In [12], the authors give an example of automaton having limited nondeterminism—there is only one transition with degree 2 for the same letter—which is hard to minimize. The same argument can be used to prove that finding the minimal finite cover automaton with *"do not care"* symbols is also a hard problem. We already know that NFCA minimization is NP-hard, and details of why the previous proofs work as well for $\diamond$-DFCAs are presented in Section 3. In the same paper [12], the example of an automaton that is hard to minimize accepts a finite language having all the accepted words of length at most 3. This example is used to show that minimizing multiple entry deterministic automata (MEFA) (When the number k of entries is known, we use the term k-DFA instead of MEFA) is hard. In Section 4, we show that the method with exactly the same construction will also work for Multiple Entry Finite Cover Automata (MEFCA), or k-entry DFCAs (k-DFCA), adding the results on k-entry FA to the previous ones obtained in [18–21]. In Section 4, we show that for binary alphabets, by transforming a k-DFCA into a minimal DFCA we can reach the upper bound for NFA to DFCA transformation. Moreover, we show that the general bound is reached for the state complexity of this transformation. Section 5 includes future work and a list of open problems, and the conclusions are drawn in Section 6.

2. Cover Automata Extensions

2.1. Notations

The number of elements of a set T is $\#T$, an alphabet is usually denoted by Σ, and the set of words over Σ is $\Sigma^\star$. The length of a word $w \in \Sigma^\star$ is the number of letters of w, and it is denoted by $|w|$. Thus, if $w = w_1 w_2 \cdots w_k$, where $w_i \in \Sigma$, for all $1 \leq i \leq k$, then $|w| = k$. In particular, when $k = 0$, we have a word with no letters, denoted by ε, and $|\varepsilon| = 0$. We also use the following notations: $\Sigma^{=l} = \{w \in \Sigma^\star \mid |w| = l\}$, $\Sigma^{\leq l} = \{w \in \Sigma^\star \mid |w| \leq l\}$, $\Sigma^{>l} = \{w \in \Sigma^\star \mid |w| > l\}$, $\Sigma^{<l} = \{w \in \Sigma^\star \mid |w| < l\}$, and $\Sigma^+ = \{w \in \Sigma^\star \mid |w| \neq \varepsilon\} = \bigcup_{i>0} \Sigma^i$.

A Deterministic Finite Automaton (DFA) is a quintuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite non-empty set, the set of states, Σ is the alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta : Q \times \Sigma \longrightarrow Q$ is the transition function. In case the transition function δ is a partial function, denoted as $\delta : Q \times \Sigma \xrightarrow{\circ} Q$, we have a partial DFA. In case δ is defined for all values of $s \in Q$ and $a \in \Sigma$, the DFA is complete. If we do not emphasize that the DFA is partial, then we understand that the DFA is complete. The transition function δ can be extended in a natural way to $Q \times \Sigma^*$ as follows: $\overline{\delta}(q, \varepsilon) = q$, $\overline{\delta}(q, wa) = \delta(\overline{\delta}(q, w), a)$. For the rest of the paper we denote the extension $\overline{\delta}$, by δ. If the transition function $\delta : Q \times \Sigma \xrightarrow{\circ} Q$ is a partial function, then the automaton is incomplete; otherwise, it is a complete one. A Nondeterministic Finite Automaton (NFA) is a quintuple

$A = (Q, \Sigma, \delta, Q_0, F)$, where all the elements are the same as for a DFA except, $Q_0 \subseteq Q$ is the set of initial states and the transition function δ, which is now defined as $\delta : Q \times \Sigma \longrightarrow 2^Q$. In case of an NFA, for the transition function we have that for $w \in \Sigma^\star$, $\delta(Q_0, w) \subseteq Q$, and $\overline{\delta}(q, wa) = \bigcup_{s \in \overline{\delta}(q,w)} \delta(s, a)$. In what follows, we will use the NFA's with only one initial state as it is defined in [22]. For any NFA $A = (Q, \Sigma, \delta, Q_0, F)$ there is an equivalent NFA $A = (Q \cup \{q_0\}, \Sigma, \delta', q_0, F)$, where $q_0 \notin Q$, $\delta'(s, a) = \delta(s, a)$, for all $s \in Q$ and $a \in \Sigma$, and $\delta'(q_0, a) = \bigcup_{s \in Q_0} \delta(s, a)$. Using the form for NFA with only one initial state, or initially connected NFAs, simplify most of the definitions and results and the state complexity will differ from the general case by just one state.

For multiple entry automata, we have a quintuple $A = (Q, \Sigma, \delta, Q_0, F)$, where $Q_0 \subseteq Q$. In some cases [23], all the states are considered initial states, thus $Q_0 = Q$, while in most other cases, we consider k-entry DFA so the transition function δ is deterministic and $\#Q_0 = k$ [19].

A state s in a finite automaton A is reachable if there is a word $w \in \Sigma^\star$ such that $\delta(q_0, w) = s$. In case of a k-entry DFA or an NFA, the state q_0 must be one of the initial states. A state s is useful if there exists $w \in \Sigma^\star$ such that $\delta(s, w) \cap F \neq \emptyset$. In case of a deterministic δ, we have that $\delta(s, w) \in F$. A sink state or a dead state is a reachable state with all its transitions being self-loops. All states that are not reachable and not useful can be eliminated without changing the language accepted by the automaton. A deterministic automaton with all states reachable and useful, except one sink state, is called a reduced automaton. In the case of nondeterministic automata, an automaton is considered reduced if all its states are both reachable and useful. In what follows, all automata are reduced automata, so they do not have unreachable or unuseful states.

For an alphabet Σ, we can consider a new symbol $\diamond$, called "do not care symbol", which can replace any letter of Σ. Thus, a word w over the alphabet $\Sigma_\diamond = \Sigma \cup \{\diamond\}$, will be a partial word if $|w|_\diamond > 0$. We say that the word $u \in \Sigma_\diamond^\star$ is weaker than $v \in \Sigma_\diamond^\star$, denoted $u \lessapprox v$, if $|u| = |v|$ and for all positions i, $1 \leq i \leq |u|$, if $u_i \in \Sigma$ then $u_i = v_i$.

Let $L_\diamond$ be a regular language over the alphabet $\Sigma \cup \{\diamond\}$, with $\sigma : \Sigma_\diamond \longrightarrow 2^\Sigma$ a substitution such that $\sigma(a) = \{a\}$ for all $a \in \Sigma$, and $\sigma(\diamond) \subseteq \Sigma$. The regular language $L_\diamond \subseteq \Sigma_\diamond^\star$ is recognized by $\diamond$-DFA, $A = (Q, \Sigma_\diamond, \delta, q_0, F)$, if $L_\diamond = L(A)$. Accordingly, a $\diamond$-DFA $A_\sigma = (Q, \Sigma_\diamond, \delta, q_0, F)$ associated with some substitution σ is defined as a DFA that recognizes a partial language $L_\diamond$, and it is also associated with the total language $\sigma(L_\diamond)$.

A cover automaton for a finite language L is a DFA recognizing a cover language L' such that $L = L' \cap \Sigma^{\leq l}$, for l being the length of the longest word in L. An l-NFCA A is a cover automaton for the language $L(A) \cap \Sigma^{\leq l}$, [10,24]. Any DFA A accepting a finite language is a DFCA for $L(A)$ with $l = \max\{|w| \mid w \in L(A)\}$.

Two words, x and y, are similar with respect to the finite language L, written $x \sim_L y$, if for every $w \in \Sigma^{\leq l - \max\{|x|,|y|\}}$, $xw \in L$, whenever $yw \in L$. In this definition, l is the length of the longest words in L. The similarity relation on words is not an equivalence relation, as it is only reflexive, symmetric, and semi-transitive.

If A is a DFCA for the finite language L, we can also define the level of a state as the length of the shortest path from the initial state to that state, that is $level_A(p) = \min\{|w| \mid \delta(q_0, w) = p\}$. In case of multiple entry DFCAs, a state will have k levels, i.e., $level_{A,i}(p) = \min\{|w| \mid \delta(q_{0,i}, w) = p\}$, for all $1 \leq i \leq k$, and $level_A(p) = (level_{A,1}(p), level_{A,2}(p), \ldots, level_{A,k}(p))$, where $Q_0 = \{q_{0,1}, \ldots, q_{0,k}\}$.

The following definition is in [10] (Definition 2):

Definition 1. *In a NFCA $A = (Q, \Sigma, \delta, q_0, F)$, two states $p, q \in Q$ are similar, written $s \sim_A q$, if $\delta(p, w) \cap F \neq \emptyset$ if $\delta(q, w) \cap F \neq \emptyset$, for all $w \in \Sigma^{\leq l - \max\{level(p), level(q)\}}$.*

In case the NFCA A is understood, we may omit the subscript A, i.e., we write $p \sim q$ instead of $p \sim_A q$, also we can write $level(p)$ instead of $level_A(p)$.

We consider only non-trivial NFCAs for L, i.e., NFCAs such that $level(p) \leq l$ for all states p, and with all the states useful and reachable.

We define deterministic and nondeterministic state complexity of a language as:

$$sc(L) = \min\{\#Q \mid A = (Q, \Sigma, \delta, q_0, F), \text{ is deterministic, complete, and } L = L(A)\}$$

and

$$nsc(L) = \min\{\#Q \mid A = (Q, \Sigma, \delta, q_0, F), \text{is non-deterministic and } L = L(A)\}.$$

In case of a finite language L, we can also define the cover complexity variants:

$$csc(L) = \min\{\#Q \mid A = (Q, \Sigma, \delta, q_0, F), \text{ deterministic, complete, and } L = L(A) \cap \Sigma^{\leq l}\}$$

and

$$ncsc(L) = \min\{\#Q \mid A = (Q, \Sigma, \delta, q_0, F), \text{ non-deterministic, and } L = L(A) \cap \Sigma^{\leq l}\}.$$

We have that $ncsc(L) \leq nsc(L) \leq sc(L)$, and $ncsc(L) \leq csc(L) \leq sc(L)$.

For an automaton A, we say that it is minimal if the number of states of A is equal to its corresponding complexity; therefore, we can have minimal DFAs, NFAs, DFCAs, NFCAs, $\diamond$-DFAs, MEFAs, and MEFCAs. An algorithm which takes an automaton of one of the above types as input and produces a minimal automaton of the same type as output is called a minimization algorithm. In some cases, minimization algorithms are exponential. Therefore, it is worth designing algorithms that will reduce the number of states, but they may not produce a minimal automaton. In that case, we have simplification algorithms that may reduce the number of states of the automaton used as input and produce an equivalent one with possibly fewer states. Simplification algorithms are preferred for cases when their computational complexity is significantly lower than the complexity of a minimization algorithm.

For either minimization or simplification algorithms, the method that is used the most is to merge two or more states into one state in such a way that the recognized language does not change. By merging state p into state q we redirect all incoming transitions to state p to incoming transitions to state q. For outgoing transitions in case of deterministic automata, outgoing transitions from p are lost, but outgoing transitions from q are preserved. In case of nondeterministic automata merging can be done in many different ways. For example, the following definition is in [10] (Definition 3):

Definition 2. *Let $A = (Q, \Sigma, \delta, q_0, F)$ be an NFCA for the finite language L.*

1. *We say that the state q is* weakly mergeable *in state p if the automaton $A' = (Q', \Sigma, \delta', q_0, F')$, where $Q' = Q - \{q\}$, $F' = F \cap Q'$, and*

$$\delta'(s,a) = \begin{cases} \delta(s,a), & \text{if } \delta(s,a) \subseteq Q' \text{ and } s \neq p, \\ (\delta(s,a) \setminus \{q\}) \cup \{p\}, & \text{if } q \in \delta(s,a) \text{ and } s \neq p, \\ (\delta(s,a) \cup \delta(q,a)) \setminus \{q\}, & \text{if } s = p \end{cases}$$

   is also an NFCA for L. In this case, we write $p \lessapprox q$.
2. *We say that the state q is* strongly mergeable *in state p, if the automaton $A' = (Q', \Sigma, \delta', q_0, F')$, where $Q' = Q - \{q\}$, $F' = F \cap Q'$, and*

$$\delta'(s,a) = \begin{cases} \delta(s,a), & \text{if } \delta(s,a) \subseteq Q' \\ (\delta(s,a) \setminus \{q\}) \cup \{p\}, & \text{if } q \in \delta(s,a), \end{cases}$$

   is also an NFCA for L. In this case, we write $p \precsim q$.

By Theorem 3 of [10], if two states are similar, they are also strongly mergeable; therefore, we can reduce the size of that automaton.

Next, we analyze two possible extensions of cover automata. One of them is to allow "do not care" symbols, while the other is to add multiple initial states. For these two types of automata, first, we give the new definitions, then we analyze which results hold and which ones need to be adapted to the new concepts.

2.2. Cover Automata for Partial Words

A DFCA with "do not care" symbols, written $\diamond$-DFCA, is a cover automaton for the finite language $L_\diamond \subseteq \Sigma_\diamond^{\leq l}$. Please note that in a $\diamond$-DFA or $\diamond$-DFCA, it is not required to have for every state transitions with "do not-care" symbol $\diamond$. Thus, partial automata are usually presented as incomplete automata, namely, the transitions of "do not care" symbol to a dead state are omitted.

The language recognized by a $\diamond$-DFCA, A, over the extended alphabet $\Sigma \cup \{\diamond\}$ is $L' = \{w \mid \Sigma^\star \cup \{\diamond\} \mid \delta(q_0, w) \in F \text{ and } |w| \leq l\}$, where l is the length of the longest accepted word. We need to find the language over the original alphabet Σ, thus we apply a substitution $\sigma : \Sigma^\star \cup \{\diamond\} \longrightarrow 2^\Sigma$ to get $\sigma(L)$ as the σ-language over $\Sigma^\star$, accepted by the $\diamond$-DFCA.

In [13], as well as in [25], for the substitution σ we can have $\sigma(\diamond) = \Gamma \subseteq \Sigma$. In this paper, we only consider the case where $\sigma(\diamond) = \Sigma$, although most results are valid even if $\sigma(\diamond) \subset \Sigma$.

By replacing the "do not care" symbols in a $\diamond$-DFCA with all letters in Σ, the $\diamond$-DFCA becomes a NFCA. Thus, if L is a language accepted by a minimal $\diamond$-DFCA with n states, then the "do not care" state complexity of L is $dnccsc(L) = n$. Since any DFCA can be also considered a $\diamond$-DFCA, we have that $ncsc(L) \leq dnccsc(L) \leq dncsc(L) \leq sc(L)$.

3. Cover Automata with "Do Not Care" Symbols

In our study, we only need to see how "do not care" symbols influence state similarity and mergeability of two states, because everything that would be valid for NFCAs would then apply to $\diamond$-DFCAs. For strong mergeability, we always obtain deterministic transitions because we remove some of the states' transitions.

For a transition $t = p \xrightarrow{\alpha} q$, $p, q \in Q$, $\alpha \in \Sigma_\diamond$, and a substitution σ, we consider $\sigma(t) = \{p \xrightarrow{a} q | a \in \sigma(\alpha)\}$, i.e., the set of all transitions that can be obtained by substituting the letter $\alpha \in \Sigma_\diamond$ with $\sigma(\alpha)$. If $A = (Q, \Sigma_\diamond, \delta, q_0, F)$ is a DFCA for L, we can denote by $\Delta = \{p \xrightarrow{\alpha} q \mid \alpha \in \Sigma_\diamond, \delta(p, \alpha) = q\}$, i.e., the set of all transitions in the automaton A and $\Gamma = \{p \xrightarrow{a} q \mid (p \xrightarrow{a} q) \in \sigma(p \xrightarrow{\alpha} q), (p \xrightarrow{\alpha} q) \in \Delta)\}$, the set of all transitions obtained from the original ones by applying the substitution σ.

We now define the compatibility of two states in a $\diamond$-DFCA.

Definition 3. *Let $A = (Q, \Sigma_\diamond, \delta, q_0, F)$ be a $\diamond$-DFCA. Two states $p, q \in Q$ are σ-compatible for the substitution σ, denoted by $p \uparrow q$, if the set $\{(a, s) \mid (p \xrightarrow{a} s) \in \Gamma\} = \{(a, s) \mid (q \xrightarrow{a} s) \in \Gamma\}$.*

Two states $p, q \in Q$ are σ-strongly compatible, denoted $p \uparrow\uparrow q$, if they are σ-compatible, $\#\{s \mid (p \xrightarrow{a} s) \in \Gamma$ or $(q \xrightarrow{a} s) \in \Gamma\} \leq \#\Sigma_\diamond$, and if there are $s, r \in Q$ and $a \in \Sigma$ such that $(p \xrightarrow{a} s), (p \xrightarrow{a} r) \in \Gamma$, we either have $r = s$, or $(p \xrightarrow{b} s) \in \Delta$, for all $b \in \Sigma$, or $(p \xrightarrow{b} r) \in \Delta$, for all $b \in \Sigma$.

When the substitution σ is understood or in case $\sigma(\diamond) = \Sigma^\star$, it can be omitted and we say that p and q are compatible, respectively, strongly compatible.

In other words, two states are compatible if by applying the substitution of "do not care" symbols for the $\diamond$-transitions, we obtain the same destination states from p and q using the same letters in Σ. A weak merge, in the sense of Definition 2 can be used in case p and q are compatible, but we need to check that this procedure won't change the language.

At the same time, two states are strongly compatible if we can take a destination state s and all transitions with all letters to s can be replaced by only one transition using a "do not care" symbol, and all other destinations can be reached by at most one symbol from Σ for all the other transitions originating in p and q.

Thus, by merging state p with state q and considering the set T of consolidated transitions, we can replace transitions in T with transitions T' such that

1. we will have only one transition for each symbol in $\Sigma_\diamond$ and
2. by applying the substitution σ, we get the same consolidating transitions, i.e., $\sigma(T) = \sigma(T')$.

This new procedure can be defined for partial DFAs and it corresponds to the strongly merging procedure in Definition 2.

Let us check the time complexity required to:

1. Decide if two states are strongly compatible, and
2. Define a method to merge two strongly compatible states.

To decide if two states are strongly compatible, we need to check the following:

1. Check if $\{(a,s) \mid (p \xrightarrow{a} s) \in \sigma(p \xrightarrow{\alpha} s), p \xrightarrow{\alpha} s \in \Delta\} = \{(a,s) \mid (q \xrightarrow{a} s) \in \sigma(q \xrightarrow{\alpha} s), q \xrightarrow{\alpha} s \in \Delta\}$. If no, then the states are not strongly compatible, so we do not attempt to strongly merge them (Consolidate the outgoing transitions and modify them to get deterministic transitions only).
2. The number of destinations $\#\{s \mid (p \xrightarrow{a} s) \in \sigma(p \xrightarrow{\alpha} s), p \xrightarrow{\alpha} s \in \Delta\}$ can be at most $\#\Sigma_\diamond$. If not, the states are not strongly compatible.
3. If for a letter $a \in \Sigma$, there are at least three distinct states s_1, s_2, s_3, such that $(a, s_i) \in \{(a,s) \mid (p \xrightarrow{a} s) \in \sigma(p \xrightarrow{\alpha} s), p \xrightarrow{\alpha} s \in \Delta\}$, for all $1 \leq i \leq 3$, then the states are not strongly compatible.
4. If there exists a letter $a \in \Sigma$, such that if there are two states s_1, s_2 with $\{(a,s_1),(a,s_2)\} \subseteq \{(a,s) \mid (p \xrightarrow{a} s) \in \sigma(p \xrightarrow{\alpha} s), p \xrightarrow{\alpha} s \in \Delta\}$, then we must have either for all $b \in \Sigma \setminus \{a\}$, $b \in \sigma(\diamond)$, $(b,s_1) \in \{(a,s) \mid (p \xrightarrow{a} s) \in \sigma(p \xrightarrow{\alpha} s), p \xrightarrow{\alpha} s \in \Delta\}$, or for all $b \in \Sigma \setminus \{a\}$, $b \in \sigma(\diamond)$, $(b,s_2) \in \{(a,s) \mid (p \xrightarrow{a} s) \in \sigma(p \xrightarrow{\alpha} s), p \xrightarrow{\alpha} s \in \Delta\}$, but not both.
   If this condition is not satisfied, we cannot replace all the transitions on $b \in \sigma(\diamond)$ to only one of the states s_1, or s_2 with the "do not care" symbol, and we do not obtain determinism for the transitions in the merged state.

Because $\#\{s \mid (p \xrightarrow{a} s) \in \sigma(p \xrightarrow{\alpha} s), p \xrightarrow{\alpha} s \in \Delta\} \leq \#\Sigma_\diamond$, all these steps, 1 to 4, take $O(1)$ time. Of course, for step 4, we may have two choices for the resulting automaton, but either one we choose, it takes constant time to do the merging. In step 4, if we have a transition from state p to state s with a letter $a \in \Sigma$ and a transition from state p to state s with "do not care" symbol $\diamond$, the transition from state p to state s with a letter $a \in \Sigma$ can be absorbed into transition from state p to state s with "do not care" symbol $\diamond$, as it is a redundant transition.

In Figure 1 are depicted all possible cases of merging two strongly compatible states, in case the alphabet is $\Sigma = \{a, b\}$.

Remark 1. *By strongly merging two states, we may obtain nondeterministic transitions. However, in the case of strongly compatible states, redundant transitions can be absorbed into the do not care symbol obtaining only deterministic ones.*

For defining the similarity relation in a $\diamond$-DFCA for two states p and q, we need the states to be similar in the corresponding NFCA, as in Definition 2 of [10].

Hence, we get the following:

Definition 4. *For a $\diamond$-DFCA $A = (Q, \Sigma_\diamond, \delta, q_0, F)$ two states p and q are similar, denoted $p \sim_A q$, if for all $w \in \Sigma^{\leq l - \max\{level_A(p), level_A(q)\}}$ and a partial word u such that $w \in \sigma(u)$, there is a partial word v, such that $w \in \sigma(v)$, and we also have that $\delta(p,u) \in F$ if $\delta(q,v) \in F$.*

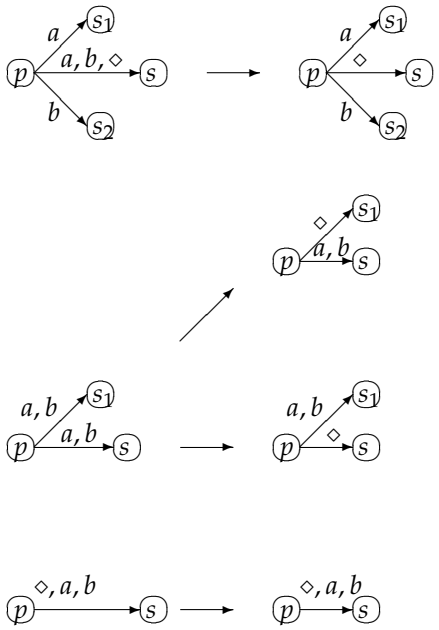

Figure 1. If the two states are strongly compatible and the result of strongly merging them is state p, the second $\diamond$ transition will be from p to s and it may overlap with an a or a b transition, or a diamond transition. In this case, we keep just one $\diamond$ transition and we drop the transitions on all letters $b \in \Sigma$ from p to s that are overlapping with it, to avoid nondeterminism. In all cases any non-deterministic choice can be avoided by "absorbing" a letter transition into the "do not-care" symbol transition.

Lemma 1. *Let $\diamond$-DFCA $A = (Q, \Sigma_\diamond, \delta, q_0, F)$ be a $\diamond$-DFCA. If two states p and q are similar, then they can be either strongly merged eliminating redundant transitions, if they are strongly compatible, or weakly merged otherwise.*

Proof. Let L' be the language of partial words accepted by A, $L = \sigma(L) \subseteq \Sigma^\star$ the associated finite language, and l the length of the longest words in L. Without any loss of generality, we may assume that $level_A(p) \leq level_A(q)$. Let $v \in \Sigma_\diamond^{\leq l}$ such that $\delta(q, v) \in F$, $v = \alpha v'$, $\alpha \in \Sigma_\diamond$. It follows that for every word $w \in \sigma(v)$ and any $x_q \in \sigma(y_q)$ such that $\delta(q_0, y_q) = q$ and $x_q w \leq l$, we have that $x_q w \in L$. We also have that $|x_q| \geq level(q) \geq level(p)$.

There is a partial word u, such that $w \in \sigma(u)$ and $\delta(p, u) \in F$ because $p \sim_A q$. Thus, by redirecting all transitions from q to p (the weakly merging method), we obtain a new automaton A' for which $x_q w$ is in the associated language of A'.

If we have a word w in the associated language of A', it means that there is a partial word z accepted by A' such that $w \in \sigma(z)$. If for every prefix of π of z, $\delta'(q_0, \pi) \neq p$, then $\delta(q_0, \pi) \neq p$, and z is accepted by A, therefore $w \in L$.

We have that $\delta'(p, v) \in F$ because $\delta'(q_0', z) \in F$ in case $\delta(q_0, \pi) = p$ for some π with $z = \pi v$. Since $p \sim_A q$, for $y \in \sigma(v)$, there is a partial word u, such that $y \in \sigma(u)$ and $\delta(q, u) \in F$. We have either $\delta(q_0, \pi) = p$, or $\delta(q_0, \pi) = q$ because A' is obtained from A by weakly merging q into state p. In both cases, $w \in \sigma(\pi u) \cup \sigma(\pi v)$, and either $\pi u \in L(A)$, or $\pi v \in L(A)$, so $w \in L$.

Hence, the language associated with the automaton A' does not change in case we do a weak merging of similar states.

If p and q are strongly compatible, let $w = aw'$, $a \in \sigma(\diamond)$. Thus, either $\delta'(p, a) = \delta(q, a)$, or $\delta'(p, \diamond) = \delta(q, a)$. Consequently, the word $x_q w$ is also in the language associated with the automaton A'.

If w is in the associated language of A', then there is a partial word z such that $w \in \sigma(z)$. In case $\delta(q_0, \pi) = p$ for some π with $z = \pi v$, because $\delta'(q_0', z) \in F$, then $\delta'(p, v) \in F$ and $w = xy$, $y \in \sigma(v)$.

Since $p \sim_A q$, for $y \in \sigma(v)$, there is a partial word u, such that $y \in \sigma(u)$, and $\delta(q, u) \in F$.

Because A' is obtained from A by strongly merging q into state p, we have either $\delta(q_0, \pi) = p$, or $\delta(q_0, \pi) = q$. In both cases, $w \in \sigma(\pi u) \cup \sigma(\pi v)$, and either $\pi u \in L(A)$, or $\pi v \in L(A)$, so $w \in L$. □

Let us see how we can use the above results to minimize $\diamond$-DFCAs.

In Section 2 of [12], the authors show that NFA minimization is NP-hard even in the case when the NFAs recognize finite languages, and they have limited non-determinism, i.e., the automata have at most one non-deterministic transition. Moreover, Corollary 7 on page 208 of [12], states that the minimization problem is NP-hard even if the input is given as a DFA. Their proof is based on the fact that the normal set cover problem is NP-complete [26,27]. Hence, if you consider these sets as paths, which corresponds to words, in an NFA, finding a minimal NFA is equivalent to finding a minimal set cover. For proving it in case of limited nondeterminism, they need a normal set cover B of a set C, i.e., for each $c \in C$, there is a subset B_c of B such that $c = \cup_{X \in B_c} X$ the elements in B_c are pairwise disjoint. The partition B is separable normal set basis for C if B can be written as a disjoint union of two other non-empty sets B_1 and B_2 such that for each $c \in C$, the subcollection B_c contains at most one element of B_1 and at most one element of B_2. To do that, they use a modified version of a known reduction from vertex cover to normal set basis (Lemma 4 in [28]), showing that the second problem is NP-hard. Using this result, they show that some instances of normal set basis sets in the partition will be pairwise disjoint and you can have just one state with two a-transitions. For (C, s), a separable normal set basis they consider, the language considered is $L = \{acb \mid c \in C, b \in c\}$, over a growing alphabet $\Sigma = \{a\} \cup \bigcup_{1 \leq i \leq n} \{c_i, b_{i_1}, \ldots, b_{i,n_i}\}$.

All accepted words are of length 3.

Therefore, for our case, we can use the same proof in two ways:

1. Either showing that $\diamond$-DFAs satisfy the conditions of Definition 1 page 201 of [12] and asking that the minimum length of the longest accepted string is at least 3, or
2. Use the same input as they use and replace the a symbol, that generates the nondeterministic transition, with a "do not care" symbol, so we get a $\diamond$-DFA. In this case, the only change would be that $L = \{\alpha cb \mid \alpha \in \Sigma, c \in C, b \in c\}$, and we would get several instances of the same problem, only with the first letter changed. Finding a minimal a normal set cover will only involve letters 2 and 3 for all paths from the start state to the final state, therefore, we can follow the same proof, but ignoring the part where they need to show that the minimal finite automaton is not ambiguous—in our case, that's not necessary. One can check that the proof works without any other change for $\diamond$-DFCAs, considering that the length of the longest accepted word is at least l, with $l \geq 3$.

It follows that:

Theorem 1. *Minimizing $\diamond$-DFCAs is NP-hard.*

Therefore, we need to seek simplification algorithms rather than minimization algorithms for $\diamond$-DFCAs. We already know [1,7,24,29] that all minimization algorithms for DFCAs are based on determining all similar states and merge them. For testing the similarity, one method [24,29], is to compute the gap function for two states p and q, where $gap(p, q)$ is the length of the shortest word that will distinguish between the states p and q. For $\diamond$-DFCAs, this means that we need to determine the length of the shortest word w such that:

1. if $w \in \sigma(u)$, $u \in \Sigma_\diamond^{\leq l - \max\{level(p), level(q)\}}$, and $\delta(q, u) \in F$, then for any partial word $v \neq u$, $v \in \Sigma_\diamond^{\leq l - \max\{level(p), level(q)\}}$ such that $w \in \sigma(v)$, we have that $\delta(p, v) \notin F$.

2. if $w \in \sigma(v)$, $v \in \Sigma_{\diamond}^{\leq l-\max\{level(p),level(q)\}}$, and $\delta(p,v) \in F$, then for any partial word $u \neq v$, $u \in \Sigma_{\diamond}^{\leq l-\max\{level(p),level(q)\}}$ such that $w \in \sigma(u)$, we have that $\delta(q,u) \notin F$.

It follows that:

1. if $\delta(p,a) = r$ and $\delta(q,a) = s$ and $gap(r,s) = k$, then $gap(p,q) \leq k+1$, and
2. if $\delta(p,a) = r$ and $\delta(q,\diamond) = s$, and $gap(r,s) = k$, then $gap(p,q) \leq k+1$.

Hence, we deduce that the gap function can be recursively computed as follows: $gap(p,q) = 1 + \min\{gap(r,s) \mid r = \delta(p,\alpha), s = \delta(q,\beta), \alpha,\beta \in \Sigma_{\diamond}$ and we have $\alpha = \beta$, or $\alpha \in \sigma(\beta)$, or $\beta \in \sigma(\alpha)\}$.

Because the number of transitions from p and q is bounded by $\#\Sigma_{\diamond}$, computing the gap function for a DFCA can be done in constant time for any pair p,q, if we know the gap function for all pairs r,s, such that $r = \delta(p,\alpha)$, $s = \delta(q,\beta)$, $\alpha,\beta \in \Sigma_{\diamond}$.

Two state would be then similar if $gap(p,q) \geq l - \max(level(p), level(q)$. With these observations and the fact that all known minimization algorithms for DFCAs are based on computing the similarity relation, we can modify any of the known minimizing algorithms for DFCAs [1,4,7,29] or l-DFCAs [24], to obtain a simplification algorithm for $\diamond$-DFCAs, without changing their computational complexity. Since the minimization algorithms for DFCAs are at most $O(n^4)$ [1,10], for all these simplification algorithms the time complexity will be at most $O(n^4)$ as well, which is better than the time complexity of the simplification algorithm proposed in [13]. Accordingly, we have obtained the following result:

Theorem 2. *For every DFCA minimization algorithm based on merging similar states having the run time complexity of $O(f(n))$, there is a simplification $\diamond$-DFCA algorithm having the same complexity of $O(f(n))$.*

A language L' is σ-minimal partial language for L if for any other language L'' such that $\sigma(L'') = L$, there is no word $w \in L'$ such we can find $x \in L''$ with x is weaker than w and $x \neq w$.

For example, $L' = \{\diamond a \diamond\}$ is σ-minimal partial language for $L = \{aaa, aab, baa, bab\}$. Indeed, if L'' is σ-partial language for L, $x \in L''$ and $w \in L'$ are such that $x \precapprox w$, then we either have $x = \diamond\diamond\diamond$ or $x = w$. In the first case, $aba \in \sigma(L'') \neq L$ and in the second case $x \neq w$ is false. Therefore, L' is a σ-minimal partial language for L.

It must be noted that the simplification algorithm proposed in [13] obtains an approximation of the σ-minimal partial language L' for the regular language L, and obtaining this σ-minimal partial language is NP-hard [13]. The cover language L', for the finite language L that we obtain by applying the simplification algorithm, may not be a cover language for the σ-minimal partial language L_σ, i.e., $L' \cap \Sigma_{\diamond}^{\leq l} \neq L_\sigma$, but it is a σ-partial language that may have a lower state complexity than the original DFCA for L and $\sigma(L')$ is a cover language for L, i.e., $\sigma(L') \cap \Sigma^{\leq l} = L$. Please note that L_σ is the weakest partial language such that $\sigma(L_\sigma)$.

In Figure 2 for the language $L = \{a, b, aa, ab, ba, aaa, aab, aba, abb, baa, bab\}$, $\max\{|w| \mid w \in L\} = l = 3$, we have the partial language $L_1 = \{\diamond, ba, a\diamond, \diamond a\diamond, a\diamond\diamond\}$, recognized by the $\diamond$-automaton $A_1 = (\{a,b\}, \{0,1,2,3,4\}, \delta_1, 0, \{2,4\})$, with useful transitions $\delta_1(0,a) = 1$, $\delta_1(0,\diamond) = 2$, $\delta_1(0,b) = 3$, $\delta_1(1,\diamond) = 4$, $\delta_1(2,a) = 0$, $\delta(3,a) = 2$, $\delta_1(4,\diamond) = 2$, that is a σ-minimal partial cover language for L, i.e., $\sigma(L_1) \cap \Sigma^{\leq l} = L$, and the words in L_1 are the weakest possible with this property. We have that the $csc(L_1) = 5$.

The language L_2 recognized by the $\diamond$-automaton $A_2 = (\{a,b\}, \{0,1,2\}, \delta_2, 0, \{2\})$, with useful transitions $\delta_2(0,a) = 2, \delta_2(1,a) = 2, \delta_2(0,b) = 1, \delta_2(2,\diamond) = 2$, is a σ-partial cover language for L, i.e., $\sigma(L_2) \cap \Sigma^{\leq l} = L$, and $csc(L_2) = 3$. $L_2 \cap \Sigma_{\diamond}^{\leq 3} = \{a, b, ba, a\diamond, a\diamond\diamond, ba\diamond\}$ and it contains 6 words, but $L_1 \cap \Sigma^{\leq 3} = \{\diamond, ba, a\diamond, a\diamond\diamond, \diamond a\diamond\}$ has only 5 words.

The example above shows that it is impossible to obtain a cover language for the σ-partial minimal language that has, at the same time, the minimal cover state complexity.

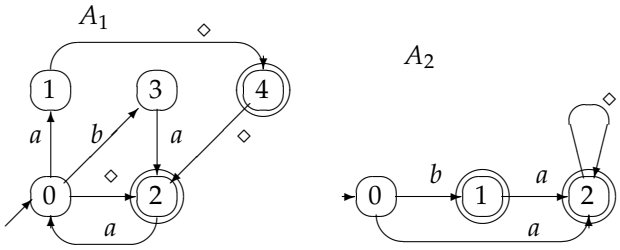

Figure 2. Automaton A_1, left, and automaton A_2, right. $L(A_1)$ is a cover language for the σ-minimal partial language for L, and $L(A_2)$ is a cover partial automaton for L, where $L = \{a, b, aa, ab, ba, aaa, aab, aba, abb, baa, bab\}$.

4. Multiple Entry DFCAs

For multiple entry DFCAs, we can have two possible flavors of extensions. The first one and the easiest to consider is the same maximum length for all words accepted by the m-DFCA. The second approach is to consider for each initial state a different maximum length. Therefore, we can use the following definition:

Definition 5. *A multiple entry DFCA with m initial states, i.e., an m-DFCA, is a structure $A = (Q, \Sigma, \delta, Q_0, F, \Lambda)$, such that Q, Σ, δ, and F are the same as for usual DFCAs, Q_0 is the set of initial states, $\#Q_0 = m$, and $\Lambda = (l_1, \ldots, l_m)$ is a sequence of m integers representing the maximum accepted length for each initial state. If $Q_0 = \{q_{0,1}, \ldots q_{0,m}\}$ and $\Lambda = (l_1, l_2, \ldots, l_m)$, the language accepted by the m-DFCA A is*

$$L(A) = \{x \in \Sigma^\star \mid \delta(q_{0,i}, x) \in F \text{ and } |x| \leq l_i, \text{ for some } 1 \leq i \leq m\}.$$

We have the condition $L(A) = \cup_{i=1}^{m} L_i$, where $L_i = L(A_i) \cap \Sigma^{\leq l_i}$, $A_i = (Q, \Sigma, \delta, q_{0,i}, F)$. The automata A_i are subautomata induced by the m-DFCA A.

We observe from the above definition that the set of initial states has the size m. Thus, by replacing the set Q_0 with an m-tuple $Q_0 = (q_{0,1}, \ldots q_{0,m})$, if we assign two possible lengths to an initial state, it does not change the accepted language. Assume that the initial states $q_{0,i}$ and $q_{0,j}$ are the same, i.e., $q_{0,i} = q_{0,j}$. The automaton will then accept all the words of length less than $\max\{l_i, l_j\}$, meaning that we can eliminate the one with the lowest maximum length, getting an $(m-1)$-DFCA.

In the example given by Björklund and Martens [12], they prove that minimizing m-DFAs is NP-hard, using a construction with a finite language. Because any m-DFA for a finite language is also an m-DFCA, by setting l_i to be the length of the longest accepting walk starting at $q_{0,i}$, it follows that

Theorem 3. *Minimizing m-DFCA is an NP-hard problem.*

We can reduce the size of m-DFCA efficiently in a similar way to the previous case for partial automata, obtaining a simplified m-DFCA by merging states. To avoid changing the language recognized by an m-DFCA A, the simplest solution is to merge similar states in all the subautomata A_i with the corresponding maximum length l_i. Any other merge of states will modify at least one of the languages involved, which will not guarantee that their union stays the same as before. Therefore, we can obtain the following definition for similarity in m-DFCAs:

Definition 6. *Let $A = (Q, \Sigma, \delta, Q_0, F, \Lambda)$ be an m-DFCA for the finite language L. Two states p and q are similar if p and q are similar in all cover automata $A_i = (Q, \Sigma, \delta, q_{0,i}, F, \lambda_i)$, $1 \leq i \leq m$.*

The simplification algorithms for m-DFCAs can be obtained as before by modifying existing DFA-minimization algorithms, therefore in what follows we will focus on state

complexity problem, namely, on constructing a minimal DFCA for the same language, and evaluating the state complexity of this transformation.

It is known that for NFA to DFA transformation for finite languages, [30], in case of a binary alphabet, the upperbound is $2^{\frac{n}{2}+1}-1$ if n is even, and $3\cdot 2^{\frac{n-1}{2}}-1$, if n is odd, n being the number of states of the NFA. Moreover, it is reached by the language $L_{m,n}=\{a,b\}^{\leq m}a\{a,b\}^n$ when $n=m$, or $n=m-1$, so $sc(L_{m-1,m-1})=2^{m+1}-1$, and $sc(L_{m,m-1})=3\cdot 2^m-1$.

Because the length of the longest word is $m+n+1$, the minimal nondeterministic finite automaton recognizing this language will have at least $m+n+2$ states, while the minimal DFA will have at least $m+n+3$ states, [30], Theorem 2. A minimal $(m+1)$-DFA for $L_{m,n}$ has the same number of states as the NFA, plus the sink state, therefore, is minimal as a m-DFA. A minimal $(m+1)$-DFCA for $L_{m,n}$ has the same number of states as the NFA minus one, because the sink state is similar with state 0, and that is the only possible similarity. We can see that the minimal nondeterministic cover automaton for $L_{m,n}$ has only $n+2$ states. For this NFCA for $L_{m,n}$, the initial state is obtained by merging all the $m+1$-entries into one, so a minimal $(m+1)$-DFCA must have m states more than the NFCA, that is $m+n+1$.

Starting from an m-DFA for a finite language, we can construct an equivalent NFA by observing that there is one initial state $q_{0,0}$ with no incoming transition, and for each initial state $q_{0,i}\in Q_0$, if $\delta(q_{0,i},a)=s_i$, we can add the transitions from $q_{0,0}$ with a in s_i, and we delete the sink state. This way, for the m-DFA to DFA transformation, we obtain the limits for NFA to DFA transformation, but we need to consider the extra sink state for m-DFAs.

Therefore, we just proved the following result:

Theorem 4. *In case of a binary alphabet, the upperbound for a n-state m-DFCA to DFCA transformation is* $2^{\frac{n-1}{2}+1}-1$ *if n is odd, and* $3\cdot 2^{\frac{n-2}{2}}-1$, *if n is even, and the bound is reached.*

Figure 3 shows a 5-DFA with 11 states for the language $L_{4,4}$. This 5-DFA is a minimal multi-entry DFA. Any nondeterministic finite automaton recognizing $L_{4,4}$ must have at least 11 states which is the length of the longest word in $L_{4,4}$ plus 1. A DFA, multi-entry or not must have a sink state, because the language is finite, therefore the automaton depicted in Figure 3a is minimal. The corresponding multi-entry DFCA, Figure 3b has the dead state d similar with state 0, so we can reduce the size by one state. The NFCA in Figure 3c recognizes $L_{4,4}$, and it has only 6 states. The general case for $L_{m,n-1}$ is depicted in Figure 4.

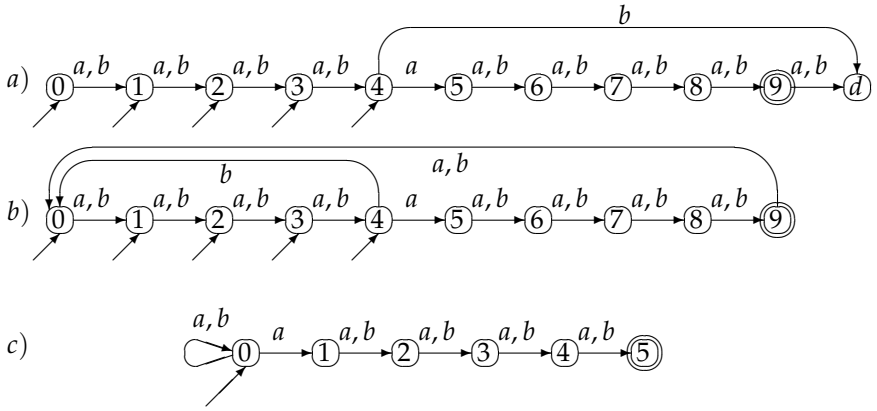

Figure 3. Example of 5-DFA having 11 states for $L_{4,4}$ (**a**), the corresponding 5-DFCA (**b**), and the corresponding NFCA (**c**). A corresponding equivalent minimal DFA for $L_{4,4}$ has 2^5-1 states and a minimal DFCA has 16 states.

The upperbound for m-DFA to DFCA transformation is the same as the NFA to DFCA transformation, but there is one difference. We have to consider that in the m-DFA, we

must have one more state as dead state because the language is finite, while in the NFA that state can be eliminated, as it is not useful.

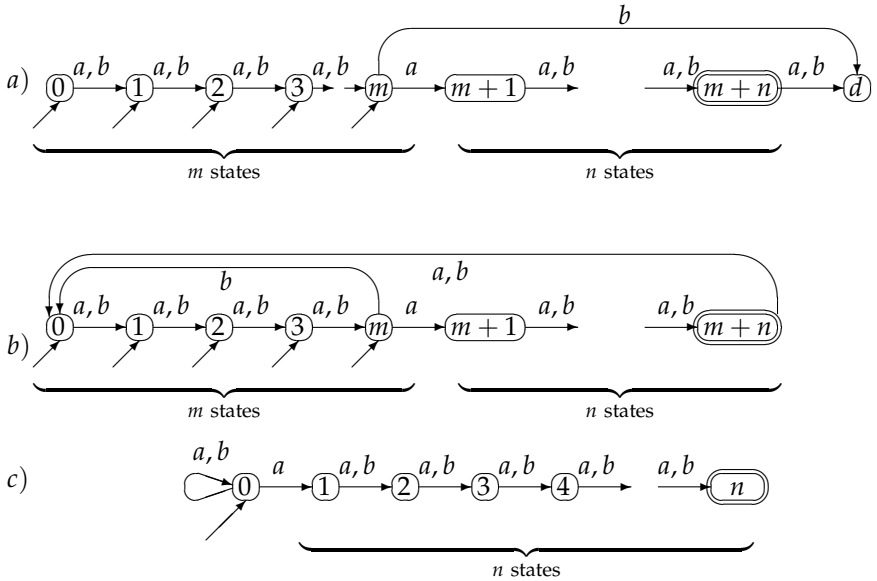

Figure 4. Example of m-DFA having $m + n + 2$ states for $L_{m,n-1}$ (**a**), the corresponding m-DFCA (**b**), and the corresponding NFCA (**c**).

5. Discussion

This paper investigates the feasibility of extending the definition of cover automata to include the cases when we allow multiple entries or "do not care" symbols. Because both operations induce a degree of nondeterminism, existing minimization algorithms working on DFAs may not give the smallest automaton in the new class, so we need to verify their run time complexity, and their correctness.

The previous automata constructions prove that minimization problems for certain nondeterministic automata are NP-hard. We checked that the same examples could also be used without any significant modification for cover automata. Hence, the results hold. The proof details for these results were omitted, as it can be found in [12] for NFAs and m-DFAs. For cover automata, we only needed to add the "do not care" symbols to substitute one letter in one transition for the first case and to add the maximum length for both cases.

In the previous studies [31,32] on state complexity of partial words DFAs we can find particular classes of languages where minimization bounds can be established. The general case is still open.

I proved that there are simplification algorithms with the same time complexity as existing minimizing DFCA algorithms. I have also computed the state complexity bound for m-DFCA to DFCA transformation.

In the case of DFCAs, the idea was floating around even in the 1960s [33,34], but no formal definition was given until 1998. That is the reason why until 2001, there was no result published on this topic, but several papers followed after the publication of [1]. In this paper, I give the required formal definitions for two DFCA extensions, and I also prove some essential results necessary to start any further investigation.

There are several questions that one may ask; for example, the following questions might be of interest:

1. Finding the state complexity of operations on $\diamond$-DFCAs.
2. Finding the state complexity of operations on with multiple entry DFCA.
3. Considering or exclusive nondeterministic finite cover automata, XNFCA.
4. Considering multiple entry XDFCA.
5. State complexity of XOR-star, XOR-concatenation for finite languages.
6. State complexity of XOR-reverse. Algebraic properties of finite languages and XOR acceptance—same length and different lengths.

7. Considering multiple lengths for multiple entry DFAs.

The study of state complexity of operations with finite languages represented by finite deterministic automata was started in [35] and later on in [36]. Later on, the state complexity of operations using nondeterministic automata were considered in [5,37–39]. It would be interesting to see where would the state complexity bounds for operations on these extensions that introduce a low level of nondeterminism would fit: closer to the deterministic results or closer to the non-deterministic models.

For each of these extensions, we need to study the following aspects:

1. Analyze the complexity of the membership problem.
2. Investigate the existence of complexity of minimization/simplification algorithms.
3. Find and evaluate the dynamic complexity and state complexity of transforming the new automata model into a known one.
4. Find bound for state complexity of operations done using the new representation model.

6. Conclusions

Two extensions are formally defined for the cover automata model of representing finite languages. Fundamental properties of these extensions were checked and proved, followed by the methodology at the end of Section 5. This article also proves that the minimization of $\diamond$-DFCAs and m-DFCAs is NP-hard, and it shows a process for obtaining simplification algorithms based on merging states in Theorem 2. The upper bound for m-DFCA to DFCA transformation is computed and proved in Theorem 4. The Discussion section also includes open problems and future research directions.

Funding: This research received no external funding.

Acknowledgments: I am very grateful to the organizers for this volume, In Memoriam of emeritus Professor Solomon Marcus. I was fortunate to participate in the celebration of his 90th birthday when we reconnected after many years, as he was one of the most influential professors in my career. I had the honor of having him as one of the professors during my time as a student at the University of Bucharest. Professor Solomon Marcus will be greatly missed for his friendly and enthusiastic personality by all of us.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Câmpeanu, C.; Sântean, N.; Yu, S. Minimal Cover Automata for Finite Languages. In Proceedings of the Third International Workshop on Implementing Automata (WIA 1998), Rouen, France, 17–19 September 1998; pp. 33–42.
2. Câmpeanu, C.; Santean, N.; Yu, S. Minimal Cover-Automata for Finite Languages. *Theor. Comput. Sci.* **2001**, *267*, 3–16. [CrossRef]
3. Câmpeanu, C.; Păun, A. Counting the Number of Minimal DFCA Obtained by Merging States. *IJFCS* **2003**, *14*, 995–1006. [CrossRef]
4. Câmpeanu, C.; Paun, A.; Smith, J.R. Incremental construction of minimal deterministic finite cover automata. *Theor. Comput. Sci.* **2006**, *363*, 135–148. [CrossRef]
5. Gao, Y.; Moreira, N.; Reis, R.; Yu, S. A Survey on Operational State Complexity. *arXiv* **2015**, arXiv:1509.03254.
6. Gruber, H.; Holzer, M.; Jakobi, S. More on deterministic and nondeterministic finite cover automata. *Theor. Comput. Sci.* **2017**, *679*, 18–30. [CrossRef]
7. Körner, H. A Time and Space Efficient Algorithm for Minimizing Cover Automata for Finite Languages. *Int. J. Found. Comput. Sci.* **2003**, *14*, 1071–1086. [CrossRef]
8. Wolfsteiner, S. Grammatical Complexity of Finite Languages. Ph.D. Thesis, TU Wien, Vienna, Austria, 2020.
9. Câmpeanu, C. Simplifying Nondeterministic Finite Cover Automata. In Proceedings of the 14th International Conference on Automata and Formal Languages, AFL 2014, Szeged, Hungary, 27–29 May 2014; Volume 151, pp. 162–173. [CrossRef]
10. Câmpeanu, C. Nondeterministic Finite Automata. In *Scientific Annals of Cuza University*; Alexandru Ioan Cuza University: Iaşi, Romania, 2015; pp. 1–25.
11. Fischer, M.J.; Paterson, M.S. String-matching and other products. *Complex. Comput.* **1974**, *7*, 113–126.
12. Björklund, H.; Martens, W. The tractability frontier for NFA minimization. *J. Comput. Syst. Sci.* **2012**, *78*, 198–210. [CrossRef]
13. Blanchet-Sadri, F.; Goldner, K.; Shackleton, A. Minimal partial languages and automata. In Proceedings of the 19th International Conference on Implementation and Application of Automata, Giessen, Germany, 30 July–2 August 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 110–123.

14. Marcus, S. Personal Communication with the Ocasion of His 90th Birthday, 2015.
15. Muthukrishnan, S.; Palem, K. Non-Standard Stringology: Algorithms and Complexity. In Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, Montreal, QC, Canada, 23–25 May 1994; Association for Computing Machinery: New York, NY, USA, 1994; pp. 770–779. [CrossRef]
16. Holzer, M.; Jakobi, S.; Wendlandt, M. On the computational complexity of partial word automata problems. In *IFIG Research Report 1404*; Institut für Informatik, Justus-Liebig-Universitäat Gießen: Gießen, Germany, 2014; pp. 1–27.
17. Holzer, M.; Jakobi, S.; Wendlandt, M. On the computational complexity of partial word automata problems. In Proceedings of the Sixth Workshop on Non-Classical Models for Automata and Applications—NCMA 2014, Kassel, Germany, 28–29 July 2014; Bensch, S., Freund, R., Otto, F., Eds.; Österreichische Computer Gesellschaft: Wien, Austria, 2014; Volume 304, pp. 131–146.
18. Galil, Z.; Simon, J. A note on multiple-entry finite automata. *J. Comput. Syst. Sci.* **1976**, *12*, 350–351. [CrossRef]
19. Holzer, M.; Salomaa, K.; Yu, S. On the State Complexity of k-Entry Deterministic Finite Automata. *J. Autom. Lang. Comb.* **2001**, *6*, 453–466.
20. Polák, L. Remarks on Multiple Entry Deterministic Finite Automata. *J. Autom. Lang. Comb.* **2007**, *12*, 279–288.
21. Veloso, P.A.; Gill, A. Some remarks on multiple-entry finite automata. *J. Comput. Syst. Sci.* **1979**, *18*, 304–306. [CrossRef]
22. Hopcroft, J.E.; Motwani, R.; Ullman, J.D. *Introduction to Automata Theory, Languages and Computation*; PearsonEd/AW: Boston, UK; San Francisco, CA, USA; New York, NY, USA; Toronto, Canada; Montreal, Canada, 2007.
23. Gill, A.; Kou, L.T. Multiple-entry finite automata. *J. Comput. Syst. Sci.* **1974**, *9*, 1–19. [CrossRef]
24. Jez, A.; Maletti, A. Computing All L-Cover Automata Fast. In Proceedings of the 16th International Conference on Implementation and Application of Automata, Blois, France, 13–16 July 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 203–214.
25. Dassow, J.; Manea, F.; Mercas, R. Connecting Partial Words and Regular Languages. In *How the World Computes—Turing Centenary Conference and 8th Conference on Computability in Europe, CiE 2012, Cambridge, UK, 18–23 June 2012*; Cooper, S.B., Dawar, A., Löwe, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7318, pp. 151–161. [CrossRef]
26. Karp, R.M., Reducibility among Combinatorial Problems. In *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, Held 20–22 March 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and Sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*; Springer: Boston, MA, USA, 1972; pp. 85–103. [CrossRef]
27. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*, 3rd ed.; MIT Press: Cambridge, MA, USA, 2009.
28. Jiang, T.; Ravikumar, B. NFA Minimization problems are Hard. *SIAM J. Comput.* **1993**, *22*, 117–141. [CrossRef]
29. Câmpeanu, C.; Paun, A.; Yu, S. An Efficient Algorithm for Constructing Minimal Cover Automata for Finite Languages. *Int. J. Found. Comput. Sci.* **2002**, *13*, 83–97. [CrossRef]
30. Salomaa, K.; Yu, S. NFA to DFA transformation for finite languages over arbitrary alphabets. *J. Aut. Lang. Comb.* **1997**, *2*, 177–186.
31. Blanchet-Sadri, F.; Goldner, K.; Shackleton, A. Minimal partial languages and automata. *RAIRO Theor. Inform. Appl.* **2017**, *51*, 99–119. [CrossRef]
32. Balkanski, E.; Blanchet-Sadri, F.; Kilgore, M.; Wyatt, B. On the state complexity of partial word DFAs. *Theor. Comput. Sci.* **2015**, *578*, 2–12. Implementation and Application of Automata, Revised Selected Papers. [CrossRef]
33. Gold, E.M. Language identification in the limit. *Inf. Control* **1967**, *10*, 447–474. [CrossRef]
34. Yu, S. Cover Automata for Finite Languages. *EATCS Bull.* **2007**, *92*, 65–74.
35. Maslov, A.N. Estimates of the number of states of finite automata. *Dokl. Akad. Nauk SSSR* **1970**, *194*, 1266–1268 (Russian). English Translation: *Sov. Math. Dokl.* **1970**, *11*, 1373–1375.
36. Yu, S.; Zhuang, Q.; Salomaa, K. The State Complexities of Some Basic Operations on Regular Languages. *Theor. Comput. Sci.* **1994**, *125*, 315–328. [CrossRef]
37. Holzer, M.; Kutrib, M. State Complexity of Basic Operations on Nondeterministic Finite Automata. *Lect. Notes Comput. Sci.* **2003**, *2608*, 148–157.
38. Holzer, M.; Kutrib, M. Unary Language Operations and Their Non-deterministic State Complexity. *Lect. Notes Comput. Sci.* **2003**, *2450*, 162–172.
39. Holzer, M.; Kutrib, M. Nondeterministic Finite Automata—Recent Results on the Descriptional and Computational Complexity. *Int. J. Found. Comput. Sci.* **2009**, *20*, 563–580. [CrossRef]