P. J. Safarik University

Faculty of Science

STATE COMPLEXITY OF PARTIALLY NONDETERMINISTIC AUTOMATA - NONDETERMINISTIC CHOICE OF INITIAL STATES

MASTER'S THESIS

Field of Study:	**Computer Science**
Institute:	**Institute of Computer Science**
Tutor:	**RNDr. Juraj Šebej, PhD.**

Košice 2025

Bc. Šimon Huraj

Contents

List of Figures

List of Tables

Chapter 1

Definitions and Preliminaries

We assume that the reader has a basic understanding of automata theory as outlined in Hopcroft and Ullman's work [2].

1.1 Notations

A *non-deterministic finite-state automaton* (NFA) is a quintuple $N = (Q, \Sigma, I, F, \delta)$ that consists of a finite set of states Q, finite set of input symbols Σ, a set of initial states $I \subseteq Q$, a set of final states $F \subseteq Q$, and a transition function $\delta : Q \times \Sigma \to 2^Q$.

We call N a *deterministic finite-state automaton* (DFA) if $|I| = 1$ and $|\delta(q, a)| = 1$ for all $q \in Q$ and $a \in \Sigma$. Therefore instead of set of initial states I we denote single initial state by i. The transition function δ of an NFA and DFA is extended as usual to $\delta^* : Q \times \Sigma^* \to 2^Q$ by $\delta^*(q, \epsilon) = \{q\}$ and $\delta^*(q, aw) = \bigcup_{q' \in \delta(q,a)} \delta^*(q', w)$.

Then the language accepted by the DFA $M = (Q, \Sigma, \delta, i, F)$ is defined as

$$L(M) = \{w \in \Sigma^* | \text{ such that } \delta(i, w) \in F\}.$$

Analogously is defined language accepted by NFA.

State $q \in Q$ is *reachable* in automaton N, if there is a $q_i \in I$ and $\alpha \in \Sigma^*$ such that $\delta(q_i, \alpha) = q$. Two automata are *equivalent* if they accept the same language.

An NFA N $= (Q, \Sigma, \delta, I, F)$ can be converted to an *equivalent* DFA $(2^Q, \Sigma, \delta', i, F')$ by the subset construction [2]: The transition function δ' is defined by $\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$ for each state R in 2^Q and each symbol a in Σ, and a state R in 2^Q is in F' if $R \cap F \neq \emptyset$. We call the resulting DFA the subset automaton of the NFA N. The subset automaton may not be *minimal* since some of its states may be *unreachable* or *equivalent* to other states. The complete subset construction has been used in our work, as it enables examination of unreachable states.

A DFA M is *minimal* if every DFA that is *equivalent* to M has at least as many states as M. It is well-known that a DFA is *minimal* if all its states are *reachable* from the initial state, and no two different states are *equivalent*. Given a DFA $M = (Q, \Sigma, \delta, i, F)$, two states $q_1, q_2 \in Q$ are said to be *equivalent*, denoted $q_1 \approx q_2$, if for every $w \in \Sigma^*$, $\delta^*(q_1, w) \in F \iff \delta^*(q_2, w) \in F$. Two states that are not *equivalent* are called *distinguishable*. For every regular language, the *minimal* DFA is unique, up to the naming of the states. In our research, Hopcroft's algorithm for *minimization*, as described in [2], has been adopted with minor modifications. In particular, during the use of the complete subset construction methodology, no *unreachable* state is removed. However, unreachable states are removed later.

1.2 Preliminaries

Let $k \geq 1$. A k-entry deterministic finite automaton ($_k$DFA) is a quintuple $M = (Q, \Sigma, \delta, I, F)$, where Q is a finite set of states, Σ a finite set of input symbols, $\delta : Q \times \Sigma \to Q$ is the transition function, $I \subseteq Q$ is the set of initial states with $|I| = k$, and $F \subseteq Q$ the set of final states [1]. Notice that a $_k$DFA is an automaton with multiple initial states and deterministic transitions.

We can see that a $_k$DFA is defined as a special case of NFA. Note that the only point of non-determinism is in the choice of initial states. Therefore, we can work with it as a usual NFA.

Then the language accepted by the $_k$DFA $M = (Q, \Sigma, \delta, I, F)$ is defined as $L(M) = \{w \in \Sigma^* |$ there is an $i \in I$ such that $\delta(i, w) \in F\}$.

Let $k \geq 1$. A *family automaton* represents a group of $_k$DFAs where all the automata in the particular group have equivalent set of states Q, set of input symbols Σ, transition function δ and set of final states F. They only differ in the set of initial states I. We can see that, *family automaton* is a deterministic automaton with no initial states.

For one n-state *family automaton* there is $2^n - 1$ possibilities of choices of initial states. So, *family automaton* represents $2^n - 1$ different $_k$DFAs or languages. Moreover, notice that in general *family automaton* is not minimal, and such automaton does not have to be connected.

Although people generally prefer the graphical interpretation of an automaton for a better understanding of its structure, there are some occasions where alternative representations are more suitable. Therefore, we will provide an alternative repre-

sentation of every *family automaton* and $_k$DFA by three integers along with a brief explanation of how to convert an automaton to these three integers and vice versa.

Let $_k$DFA M be automaton with $Q = \{0, 1, \ldots, n-1\}$, so $|Q| = n$ and $\Sigma = \{0, 1, \ldots, m-1\}$, so $|\Sigma| = m$. Let us provide the representation of M by three integers denoted by (x, n, m), clearly n and m are already given. Next, we describe step-by-step how to get x. First, get $n.m$ digits number x_δ in the base-n. Let $i \in \{0, ..., n\}$ and $j \in \{0, ..., m\}$, then $(i+j)$-th digit represents destination state for q_i reading j-th letter of alphabet. Secondly, create x_F number, which is a binary number with n digits, the value of i-th is 1 when q_i is final, otherwise 0. Next, construct n digit number x_I. The value of i-th digit is 1 when q_i is initial, otherwise 0. Convert x_δ, x_F, x_I to base-10. Finally, add all the numbers together by the formula:

$$x = (x_\delta * 2^n + x_F) * 2^n + x_I.$$

This completes encoding of $_k$DFA by (x, n, m).

To transform x back to automaton we simply reverse the steps. Firstly, $x \mod 2^n$ converted to binary characterizes initial states. Then $\lfloor x/2^n \rfloor \mod 2^n$ converted to binary characterizes final states. And finally, $\lfloor \lfloor x/2^n \rfloor / 2^n \rfloor$ converted to base-n number characterizes transitions in automaton.

Example 1.1 (Conversion from automaton to number) Let's have an automaton M=(Q, Σ, δ, q_0, F), Q=$\{q_0, q_1, q_2\}$, Σ=$\{$a,b$\}$, F=$\{q_0\}$ and transitions are as shown on the Figure 1.1

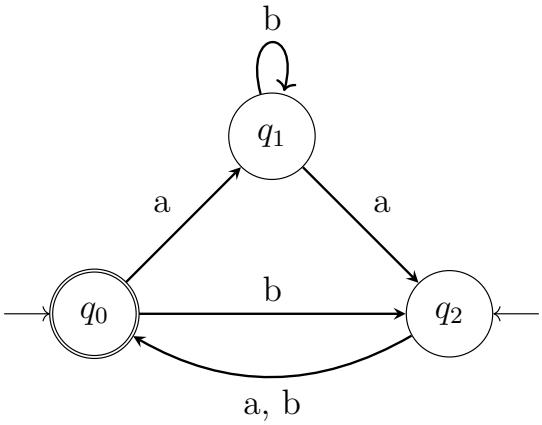


Figure 1.1: Three state automaton

Considering $|Q| = n = 3$ and $|\Sigma| = m = 2$, then $x_\delta = (122100)_3 = (468)_{10}, x_F = (100)_2 = (4)_{10}$ and $x_I = (101)_2 = (5)_{10}$. Putting it all together

$$(468 * 2^3 + 4) * 2^3 + 5 = 29989$$

and that is a corresponding number to automaton M. In conclusion, we can describe M with numbers (29989, 3, 2).

Example 1.2 (Conversion from number to automaton) Let's start with an automaton number $x = 29989$, number of states $n = 3$ and length of alphabet $m = 2$. Then we gradually extract numbers from x.

- $x_I = x \mod 2^n = 29989 \mod 8 = 5 = 101_2$
- $x_F = \lfloor x/2^n \rfloor \mod 2^n = \lfloor 29989/8 \rfloor \mod 8 = 4 = 100_2$
- $x_\delta = \lfloor \lfloor 29989/8 \rfloor /8 \rfloor = 468 = 122100_3$

And now we can construct an automaton. It will have n states and alphabet of length k. Without loss of generality Q $= \{q_0, q_1, q_2\}$ and $\Sigma = \{a, b\}$. From x_I we observe that q_0 and q_2 and initial, therefor $I = \{q_0, q_2\}$. From x_F we observe that only q_0 is final, therefore $F = \{q_0\}$. And from x_δ we acquire following: $q_0 \xrightarrow{a} q_1$, $q_0 \xrightarrow{b} q_2$, $q_1 \xrightarrow{a} q_2$, $q_1 \xrightarrow{b} q_1$, $q_2 \xrightarrow{a} q_0$, $q_2 \xrightarrow{b} q_0$. Based on the three numbers, we were able to obtain the automaton already showed on the Figure 1.1.

When working with *family automaton*, rather than $_k$DFA, we prefer a more convenient representation. Fortunately, the algorithm stated above can also be applied to *family automaton* with some customization. As noted in the definition, a *family automaton* is also an automaton but without initial states. The algorithm works in the same way as for individual automata, but in this case, it omits the part with initial states. Therefore, the formula for converting an automaton to a number changes to

$$x = x_\delta * 2^n + x_F.$$

Additionally, the reverse conversion does not take initial states into account.

Chapter 2

Generation of families and automata

In this thesis, we build upon our previous work on this subject. We had some initial generation results available for study. As generating more and more automata, especially with a higher number of states, becomes increasingly computationally intensive, we opted to explore the realm of parallel computing.

2.1 Parallel computing

Parallel computing is a computational approach where multiple processes or threads are executed simultaneously to solve a problem more efficiently. By dividing a task into smaller sub-tasks and running them concurrently on multiple processors, parallel computing can significantly reduce execution time and handle large datasets or complex computations effectively.

Advantages include faster processing, efficient utilization of multi-core systems, and the ability to solve problems that are computationally infeasible for sequential computing. It is particularly beneficial in fields like scientific simulations, big data analytics, and artificial intelligence.

However, disadvantages include the complexity of programming and debugging, the overhead of managing parallel tasks, and potential inefficiencies due to synchronization and communication between processes. Difficulties arise in ensuring load balancing, minimizing contention for shared resources, and debugging concurrency-related issues like race conditions and deadlocks. Additionally, not all problems are easily parallelizable, limiting the applicability of this approach.

2.2 Parallel generation

The problem of generating families and their subsequent processing is in fact excellent candidate for leveraging all the benefits of parallel computing. In our case, work can be cleverly divided between multiple processors to safely bypass all the problems when it comes to concurrent execution of a program. We know in advance what range of families are we going to generate, therefore we can split this range of numbers for families into disjoint sets, one for each available processor. Each processor then handles all of his families, creating his own statistics. The only problematic point could be joining the sub-results from all the sub-processes. However, as this is not a computationally heavy task we can simply gather the results from every single processor and join them sequentially in main thread of program.

Let us compare how the parallel version differ from the previous sequential approach. Algorithm 1 shows steps that were used for generating before the parallelization.

Algorithm 1 Generating on Single Thread

Require: Set of integers S

1: **for** i **from** S **do**
2: Create *family automaton* for i based on the conversion algorithm provided
3: Determinize
4: Minimize
5: **for every possible initial state do**
6: Traverse the construction from chosen initial state
7: Calculate desired statistics
8: **end for**
9: **end for**
10: **return** Result for automata from input range

Algorithm 2 then shows the parallel version making use of the single thread algorithm for the individual processors.

Algorithm 2 Generating on Multiple Threads

Require: Set of integers S, Number of Threads N

1: Divide S into sets $S_1, S_2, \ldots, S_N$ approximately the same size
2: Create a thread pool with N threads
3: **for** $i = 1$ to N **in parallel do**
4: Assign Algorithm 1 with input S_i to thread i
5: Execute thread i
6: **end for**
7: Wait for all threads to complete
8: Combine results from all threads
9: **return** Combined result

The result of this parallelization is straightforward. With N available processors we obtain linear speedup. So, $S_N \in \mathcal{O}(n)$.

2.3 Goals

In this thesis, we went in two directions of generating. The first one is in fact just a continuation of previous work. We were able to generate all 6 state families. However, going any further was not possible even with the optimization in the form of parallelism.

The other way was supposed to help us gather data for automata accepting languages over an alphabet of bigger size than 2. The goal behind this was to help us formulate a hypothesis about how the size of the alphabet influences state complexity and furthermore, the average state complexity of the class of automata we study. In this case, we managed to accumulate results for the alphabet of size 3 for families up to 4 states and for the alphabet of size 4 for families up to 3 states. As one can see, with the increasing size of the alphabet the maximum size of families we are able to analyze decreases. From the encoding provided in [**?**] it should be obvious that with increasing alphabet size the number of families to be analysed grows exponentially and so does the time needed for processing.

2.4 Results

In this section, we will provide data gathered through generation. Firstly let's look at results for 6 states family with binary alphabet. We will show charts with all the important results, for the precise values, please refer to the tables at the end of this section.

Figure 2.2 shows state complexity for individual 6 state ${}_k$DFAs. Every bar represents the number of automata with a certain state complexity. All the values for state complexities up to 63 are non-zero.

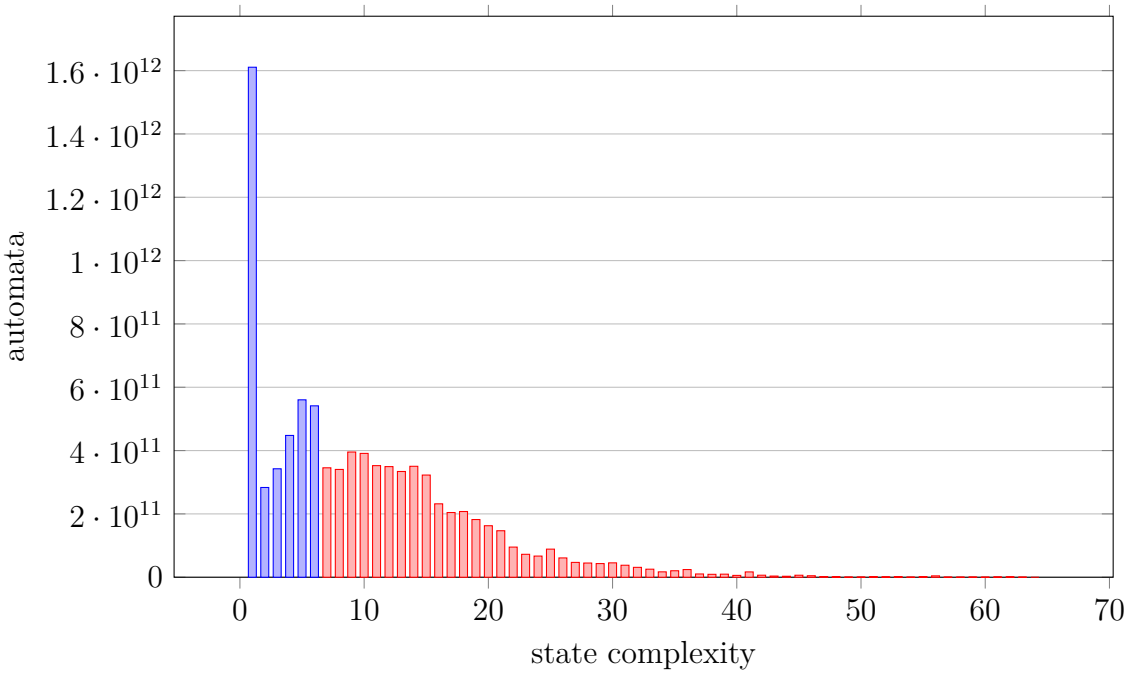


Figure 2.2: Six state ${}_kDFAs$ - state complexity

Figure 2.3 below shows non-equivalent languages per family. One bar of the chart represents a number of families that represent a certain number of non-equivalent languages.

A number of distinct state complexities of 6 state families can be seen in Figure 2.4 below. Every bar of this chart represents a number of families that represent automata with a particular number of distinct state complexities. 30 is the highest number of distinct state complexities that any 6-state family have.

All these figures look similarly as their counterparts we already examined for automata with fewer states.

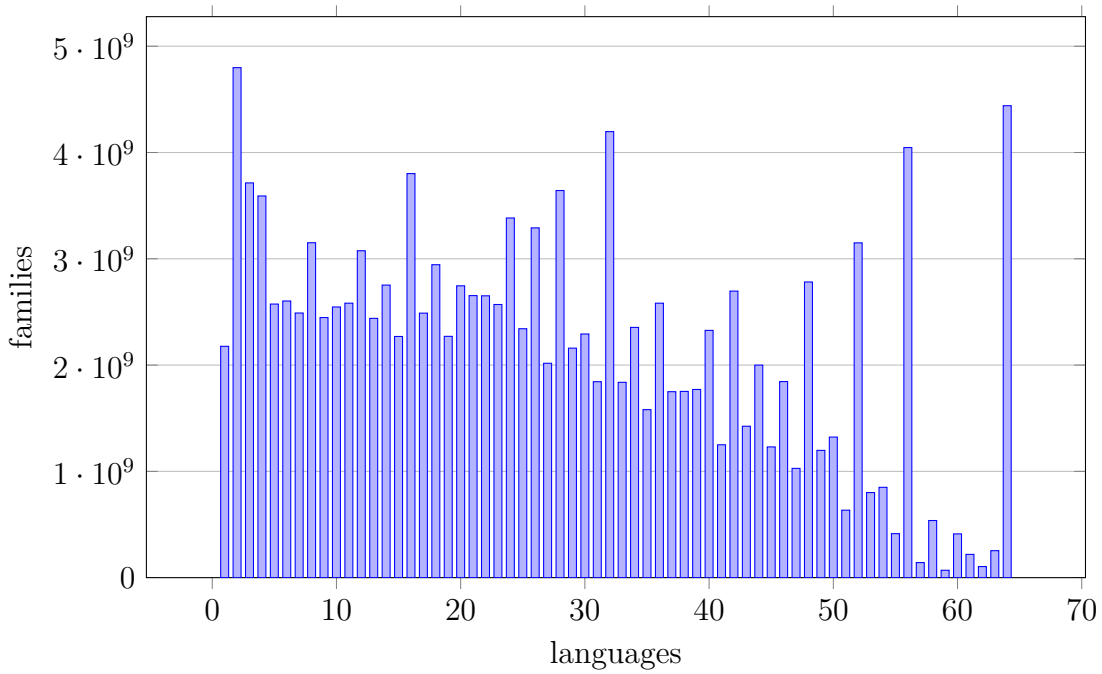


Figure 2.3: Six state ${}_kDFAs$ - number of families with number of non-equivalent languages

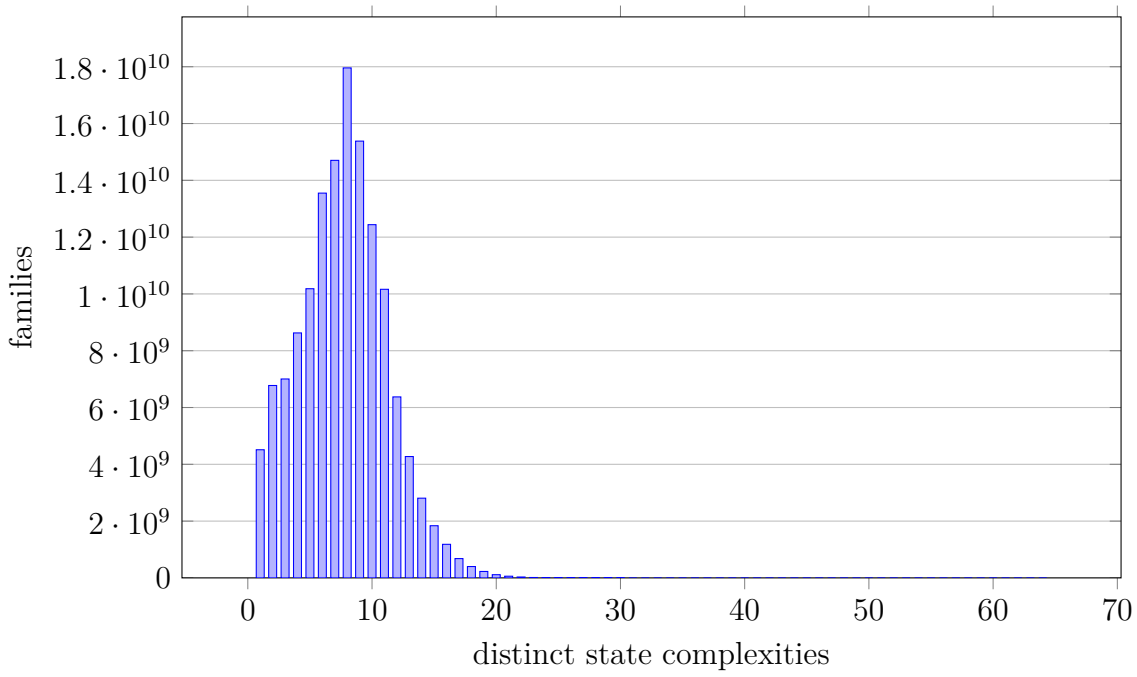


Figure 2.4: Six state ${}_kDFAs$ - number of families with particular number of distinct state complexities

2.5 Bigger alphabets

In Section 2.4 all the results as well as all the results in previous work were all based on the alphabet of size 2. Sometimes using a bigger alphabet could help to identify

State complexity	Number of Automata	Number of Non-equivalent Languages	Number of Families	Number of Distinct Number of States	Number of Families
1	446	1	64	1	130
2	262	2	80	2	66
3	60	3	38	3	60
4	0	4	74	4	0

Table 2.1: Results for 2 state automata with alphabet of size 3

some key features of the model, therefore we have decided to examine it. Obviously, increasing the size of the alphabet by any letters leads to an exponential increase in the number of families needed to analyze. Taking into account all the optimization performed on the program for generation, we still couldn't go much further with the number of states.

Here we will provide the results we were able to obtain for a number of states from 2 to 4. For 2 and 3 state automata we have a result for the alphabet of size up to 4. For 4-state automata only the results for the alphabet of size 3 were possible.

Table 2.1 shows results for 2 states automata with the alphabet size 3. Every row consists of 3 results. Let's take the second row. The left part states that there are 262 $_k$DFAs that have state complexity 2. The middle part states that there are 80 families that have 2 non-equivalent languages. The right side states that there are 66 families that have automata that have only 2 distinct values for state complexity.

State complexity	Number of Automata	Number of Non-equivalent Languages	Number of Families	Number of Distinct Number of States	Number of Families
1	1662	1	256	1	514
2	1090	2	288	2	190
3	320	3	130	3	320
4	0	4	350	4	0

Table 2.2: Results for 2 state automata with alphabet of size 3

Table 2.2 is similar to Table 2.1, the only difference is that it shows results for 2 state automata with alphabet size 4.

Next Tables 2.3, 2.4 and 2.5 show similar data for gradually 3 state automata

State Complexity	Number of Automata	Number of Non-equivalent Languages	Number of Families	Number of Distinct Number of States	Number of Families
1	361659	1	19683	1	39750
2	117801	2	25242	2	20172
3	320856	3	9081	3	28278
4	45216	4	17214	4	62604
5	72504	5	9264	5	6660
6	154800	6	6414	6	0
7	29412	7	26388	7	0
8	0	8	44178	8	0

Table 2.3: Results for 3 state automata with alphabet of size 3

with an alphabet of size 3, 3 state automata with an alphabet of size 4 and 4 state automata with an alphabet of size 3.

These results conclude our experiments with generation since increasing the number of states of the size of alphabet even further is so computationally heavy, that it requires a lot more time or computational power. However, neither of them we have.

2.6 Interesting observations

By studying generated data we came across some interesting observations. As much as we would like to prove them, due to time and capacity limitations, they only remain hypotheses for now.

The first observation concerns about *families*. From the generation results, we can connect data for a number of languages per family and the state complexity of these languages. What is interesting is that there are many *families* that have many distinct languages but these languages have small state complexities. This observation goes back to our original motivation with logical circuits. When we look at the ${}_k$DFA as equivalent to the logical circuit, we can see that this circuit without connected input pins can represent many simple logical functions. Simple in a way that the same function can be calculated by a circuit with few logical gates, which in our case represent states of automaton.

The second one is about automata themselves and their state complexity. When dealing with average state complexity, we have thought about what operation does

State Complexity	Number of Automata	Number of Non-equivalent Languages	Number of Families	Number of Distinct Number of States	Number of Families
1	8348727	1	531441	1	1064418
2	1730337	2	595524	2	254316
3	9288396	3	117549	3	618618
4	812496	4	257058	4	2129952
5	1909176	5	147552	5	184224
6	6325668	6	111738	6	0
7	1345896	7	674172	7	0
8	0	8	1816494	8	0

Table 2.4: Results for 3 state automata with alphabet of size 4

State Complexity	Number of Automata	Number of Non-equivalent Languages	Number of Families	Number of Distinct Number of States	Number of Families
1	740012880	1	16777216	1	33736472
2	138149580	2	22100480	2	14560760
3	300521124	3	6865060	3	16167816
4	757745784	4	7684692	4	43059648
5	139083768	5	5360892	5	74322192
6	225484128	6	5250012	6	62873352
7	223478640	7	9650808	7	14852736
8	245708784	8	14651088	8	7555104
9	404812368	9	6059496	9	1224144
10	462502368	10	6186816	10	78768
11	116237232	11	13782672	11	4464
12	83761488	12	17947248	12	0
13	69439392	13	16477848	13	0
14	103533696	14	34881840	14	0
15	16060608	15	17440704	15	0
16	0	16	67318584	16	0

Table 2.5: Results for 4 state automata with alphabet of size 3

more. Does determinization increase the number of states more, or on the other hand, does the minimization decrease the number of states more? Because then the state complexity of n-state automaton is decided on whether determinization or minimization "wins". We looked at the latter of the two operations more closely, but only experimentally. The result was that the minimization reduces the number of states by a constant factor somewhere right below 1,5. Meaning that if we have m states after determinization, after minimization we will have $\frac{m}{1,5}$.

Chapter 3

Magic Numbers

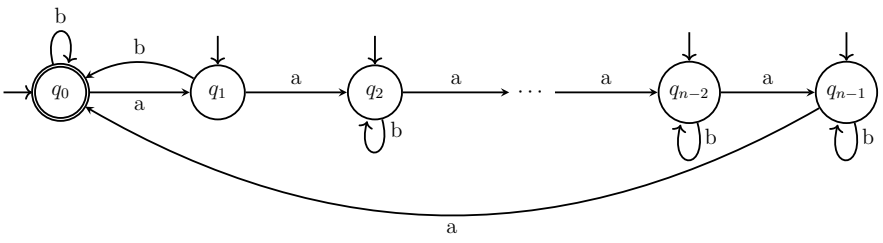
In this chapter, we will deal with so-called magic numbers. Let's say we have 3 state $_k$DFA. We will construct a minimal deterministic DFA and observe how many states it has. Clearly, it must be a number from the range 1 to $2^n - 1$. Now the question stands: Is there a number of states that can't be obtained from some $_k$DFA? If there is, the number is then called a magic number. The magic number doesn't have to be the only one, there can be multiple values that can't be obtained.

In our case, for this class of automata, the hypothesis is that there are no magic numbers. This hypothesis came from our experiments, where all the values of state complexity where reached for $_k$DFAs with 1 to 6 states. So, the hypothesis in other words says, that for every pair (n, m), $n \in \mathbf{N}$ and $m \in \mathbf{N}, m \leq 2^n - 1$, there exists a n-states $_k$DFA such that, minimal DFA recognizing the same language has exactly m states.

Firstly we will show that the upper bound for m is reachable.

Lemma 3.1 *For every $n \in \mathbf{N}$ there exists a n-state $_k$DFA such that equivalent minimal DFA has exactly $2^n - 1$ states.*

Proof.
We will construct a $_k$DFA:



Determinize this automaton while using complete subset construction to get DFA M with 2^n states, including a state that represents an empty subset of states. We will exclude this one as it can't be reached from the initial state because of the deterministic structure. This new automaton will have one initial state $\{q_0, q_1, \cdots, q_{n-1}\}$. Now we need to show that all $2^n - 1$ states are reachable from the initial state and distinct.

Firstly, let us show that all $2^n - 1$ states are distinct. Let F' be a set of final states in M. Take S and T, distinct subsets of Q. Without loss of generality $S \not\subseteq T$, if not we swap S and T. Next, take $q_s \in S$, $q_s \notin T$, $s = 0, \cdots, n-1$ and word a^{n-s}. Then $q_s \xrightarrow{a^{n-s}} q_0$, that means $S \in F'$. As s is not in T and no other state transitions to q_0 with this word, $T \notin F'$. As a result for every state q_i, $i = 0, \cdots, n-1$ a^{n-i} is a distinction word.

Let us show the reachability now. For this, we will use two features of how is the automaton constructed, contraction and rotation. Mark state as i-big when it is a subset of size i. Our initial state is n-big. Using contraction, $q_1 \xrightarrow{b} q_0$, we will reach $(n-1)$-big state. Here using rotation, we can visit all other $(n-1)$-big states. In every $(n-1)$-big state we can use contraction as well to get to $(n-2)$-big state. Again use rotation, and so on. Alternating contraction and rotation we will visit all $2^n - 1$ states.

□

Example 3.2 (Upper bound for state complexity) Let us take 3-state automaton from previous Lemma.

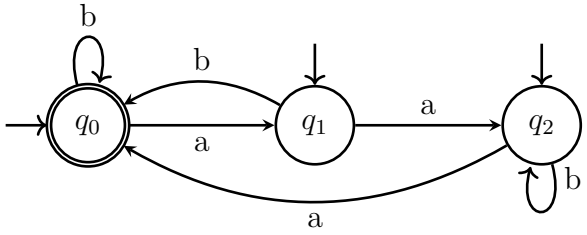


Figure 3.5: Three state automaton from Lemma 3.1.

And determinize it as can be seen in Figure 3.6.

It is clear that every state in determinized automaton, except $q_\emptyset$, is reachable, and every state can be distinct from all other states.

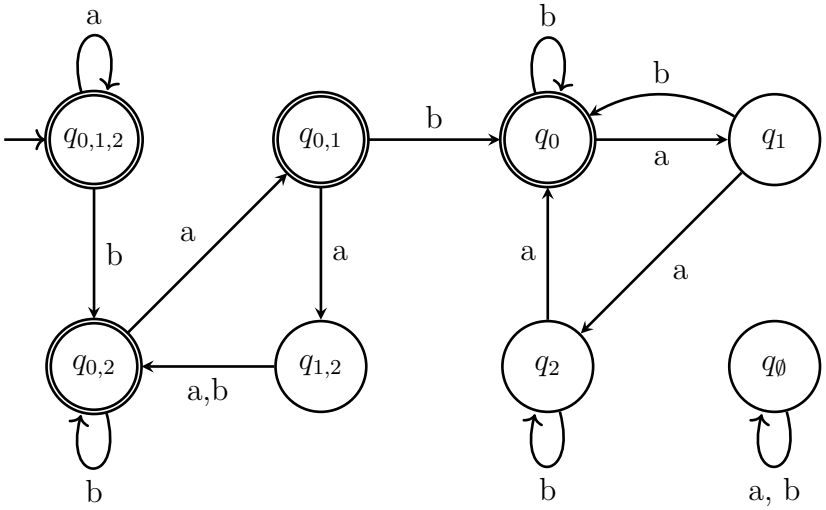


Figure 3.6: Minimal determinized three state automaton

Next we will show the stronger statement and so, that for every pair (n, m), $n \in \mathbf{N}$ and $m \in \mathbf{N}, m \leq 2^n - 1$, there exists a n-states ${}_k$DFA such that equivalent minimal DFA has exactly m states. However, in this case, the alphabet of linear size is required.

3.1 Linear alphabet

Firstly, let us explain the idea behind the construction of such automaton. We will have two special letters 'a' and 'b', 'a' for doubling resulting number of states and b for adding one additional state. Now, let's say we want 5-state ${}_k$DFA with 29 resulting state complexity. We can get that from 4-state ${}_k$DFA with 14 resulting state complexity using the letter 'a' for doubling, and letter 'b' to add exactly one more. And again the 4-state automaton can be obtained from 3-state ${}_k$DFA with 7 resulting state complexity using only the letter 'a' for doubling state complexity. We can look at it as a recursive function with input n - number of states of automaton and m - desired number of states of current level. Or, as in most cases when using recursive idea, we can reformulate this approach using ideas of dynamic programming where the complex problem is broken down into simpler subproblems. Then the final result is built from simple cases, from the bottom up. For better understanding see example Figure 3.7. Here we simple start with the only state and gradually, in phases, build our desired automaton adding letters a and b.

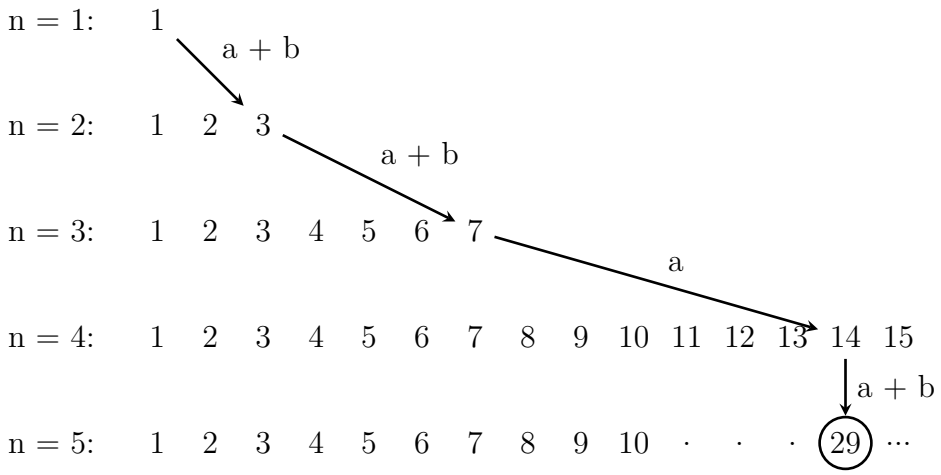


Figure 3.7: Magic numbers with linear alphabet

In general the build-up can be visualized as follows: Where blue arrow indicate

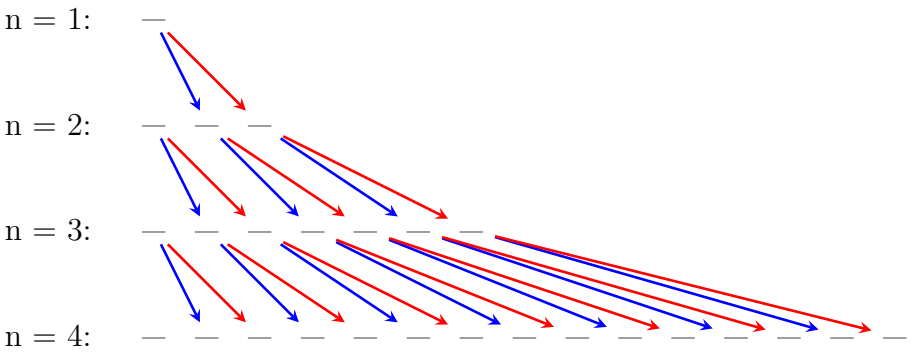


Figure 3.8: General build-up of state complexity

using only letter a and red one using both a and b for doubling plus one.

From the Figure 3.8 can be seen that it is not always necessary to start with $n = 1$. Let's say we want 4-state ${}_k$DFA with resulting state complexity 5. In this case we will start with 2-state ${}_k$DFA and build it up to 4-state.

Lemma 3.3 *For every pair (n, m), $n \in \mathbf{N}$ and $m \in \mathbf{N}, m \leq n$, there exists a n-states ${}_k$DFA such that equivalent minimal DFA has exactly m states.*

Proof.
We will construct a ${}_k$DFA $M = (Q, \Sigma, \delta, I, F)$, where $Q = \{q_0, \ldots, q_{n-1}\}$, $\Sigma = \{a\}$, $I = F = \{q_0\}$ and

$$\delta(q_i) = \begin{cases} q_{i+1} & i < m-1 \\ q_0 & i = m-1 \\ q_i & i \geq m \end{cases} \quad i \in \mathbf{N}$$

Such structure then looks like the one in Figure 3.9.

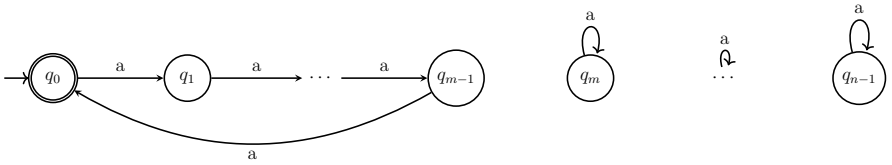


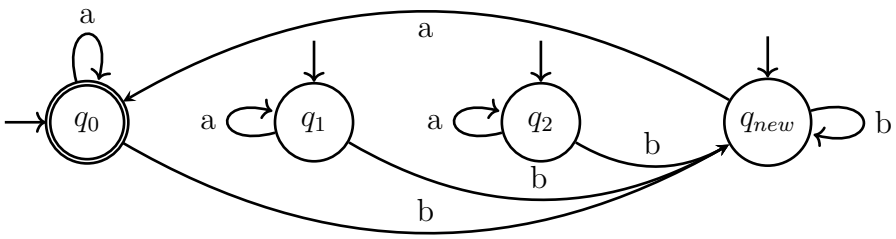
Figure 3.9: Example automaton

It is clear that m states will be visited. To show that states q_i and q_j, for $i > j$, are distinct, we can use word a^{m-i} which will transfer q_i to $q_{i+m-i} = q_0$ which is final and q_j to $q_{j+m-i} \neq q_0$ and that is non-final state.

□

Lemma 3.4 *For every pair (n, m), $n \in \mathbf{N}$ and $m \in \mathbf{N}, n \leq m \leq 2^n - 1$, there exists a n-states ${}_k$DFA such that equivalent minimal DFA has exactly m states.*

Proof.



□

To better understand the process of build-up, let us present a default case scenario on 2-state ${}_k$DFAs with the first and second phase of adding new letters with additional state. In Figure 3.10 there are two rows, as the first column indicates, first row includes 2-state ${}_k$DFA with its minimal DFA that has exactly 2 states. The second row includes 2-state ${}_k$DFA with its minimal DFA that has exactly 3 states.

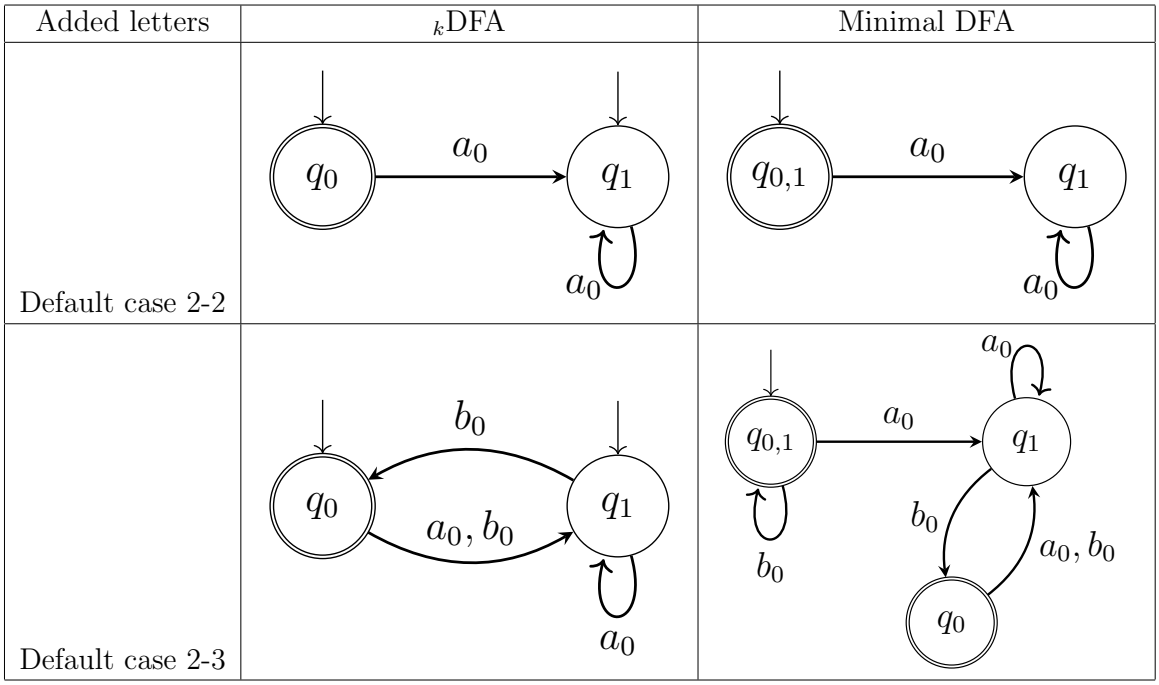


Figure 3.10: Default cases with 2-state $_k$DFAs

First phase of build-up consists of adding new state and letters. We will take default cases from Figure 3.10 and add new state to each of the $_k$DFAs. The next step is to correctly model new transitions. For every default case create a two new automata, one with added letter a_1 and the second one with added letters a_1 and b_1. The final result can be seen in Figure 3.11

This process continues similarly until we reach desired level. For the simplicity we will provide only the $_k$DFA without their minimal determinstic equivalent in second phase. The result can be seen in Figure 3.12

Lemma 3.5 *For every pair (n, m), $n \in \mathbf{N}$ and $m \in \mathbf{N}, m \leq 2^n - 1$, there exists a n-states $_k$DFA such that equivalent minimal DFA has exactly m states.*

Proof.
Proof is a result of Lemmas 3.3 and 3.4

□

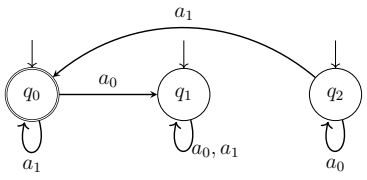
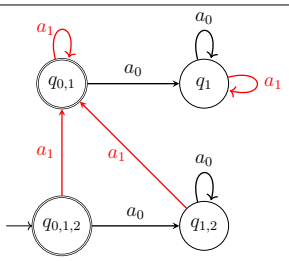
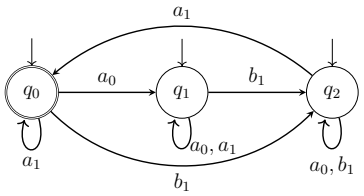
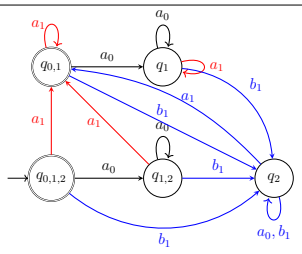
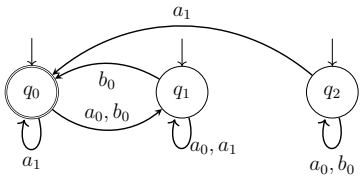
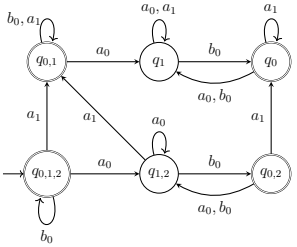
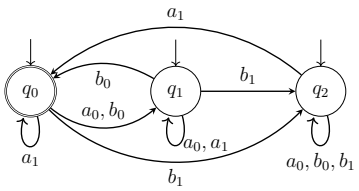
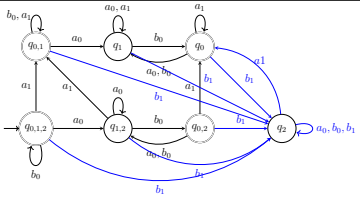
Added letters	$_k$DFA	Minimal DFA
2-2 + a = 3-4		
2-2 + $(a + b)$ = 3-5		 $q_{0,1}$ is equivalent to q_0
2-3 + a = 3-6		
2-3 + $(a + b)$ = 3-7		

Figure 3.11: First phase of build-up

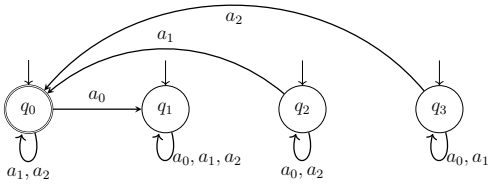
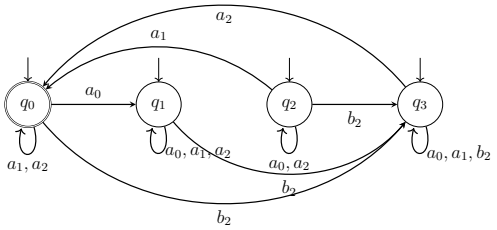
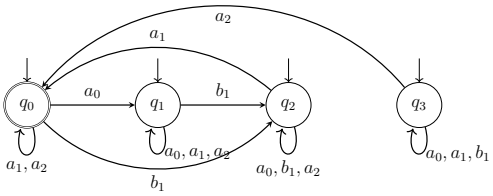
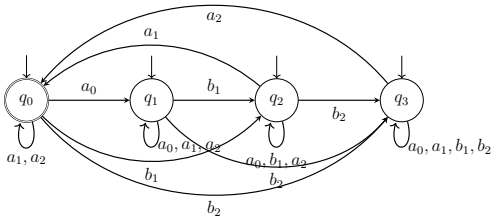
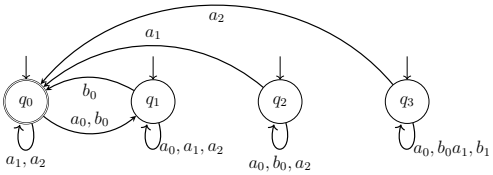
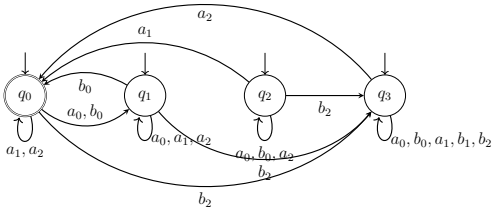
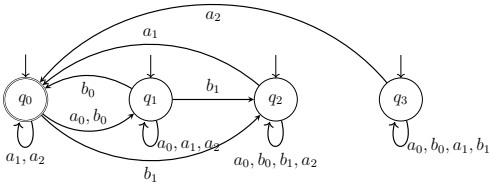
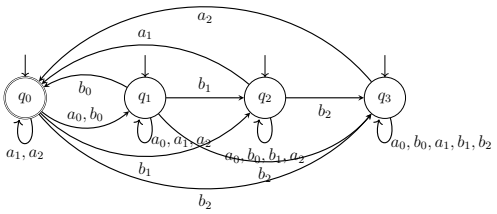
Base	${}_k$DFA with $+\ a$	${}_k$DFA with $+\ (a+b)$
3-4		
3-5		
3-6		
3-7		

Figure 3.12: Second phase of build-up

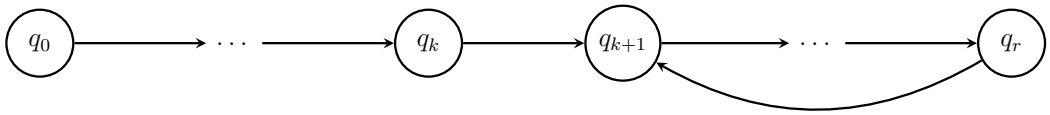
3.2 Unary alphabet

In this section, we will explore what is the achievable range of state complexities in unary $_k$DFAs. Usually, unary automata are special case when it comes to various problems. Same will hold in our case.

Lemma 3.6 *For pair $(n, m) \in \mathbf{N}^2$ there exists an unary n-state $_k$DFA such that equivalent minimal DFA has exactly m states if and only if $m \leq n$.*

So this Lemma 3.6 says that only rage from 1 to n of state complexity is reachable for unary $_k$DFAs.

Proof.
We will use the fact that every unary automaton can be replaced by equivalent automaton consisting of tail and loop.



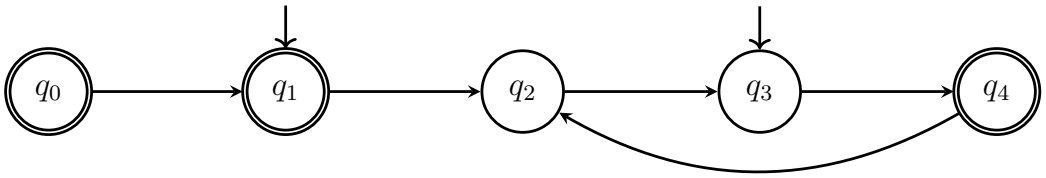
Final states can be assigned randomly. Assume there are f final states. And as it is $_k$DFA, by definition, every state can be also initial.

New, we will create a equivalent DFA. This DFA will have only one initial state, it will be q_0. For every initial state in $_k$DFA add all f final states to this newly created DFA, in a way that, if q_i is the initial state and q_j is currently added final state, assign q_{j-i} as final. Therefore if some string a^p ended in final states, thus is accepting in $_k$DFA, it will be accepting in DFA since from q_0 will end in final state.

As this construction is deterministic, it is clear that more than n states cannot be reached. Besides, it is not guaranteed that this automaton is minimal, so the final state complexity could potentially be even lower.

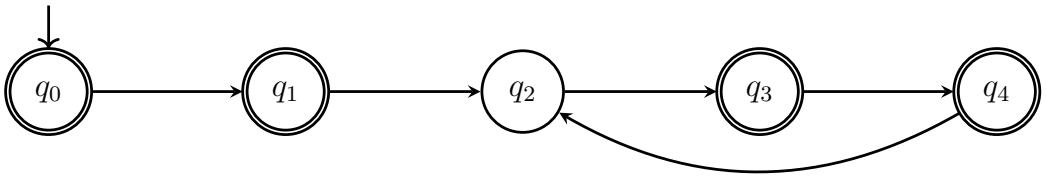
□

Example 3.7 Let's start with this $_k$DFA



Now we have 2 initial states: q_1 and q_3. Firstly, for q_1 states q_0 and q_3 we be final in constructed DFA. For the latter initial state states $q1$ and q_4 will be final.

So the resulting automaton will look like this:



In this case we can see that the q_0 was not reachable in $_k$DFA, so it does not play a role in the construction of the equivanelt DFA.

Chapter 4

Average state complexity and other results

4.1 Average state complexity

In our previous work, we have already ventured in to the area of average state complexity. There we proved exact formula for upper bound for average state complexity as well as more convenient form. This useful form was however proven only for odd $n \geq 5$ in Theorem:

Theorem 4.1 *Let $n \geq 5$ be odd number, then average state complexity of a language represented by an n-state ${}_kDFA$ is at most $5/8 \times 2^n$.*

In this work, we devoted a significant amount of time to proving the other counterpart, ensuring it holds for even n. But this turned up to be rather a difficult task. Then came the idea to start from the beginning and not divide the work into even and odd n. This was not easy as well, but we were able to formulate a stronger statement and also prove it.

Theorem 4.2 *Let $n \in \mathbb{N}$, $n \geq 1$, then average state complexity of a language represented by an n-state family is at most $5/8 \times 2^n$.*

Proof.
Basically, what we want to show is that:

$$\frac{\sum_{i=1}^{n} \sum_{j=1}^{i} \binom{n}{i}\binom{n}{j}}{2^n - 1} \leq \frac{5}{8} 2^n.$$

holds for every $n \in \mathbb{N}$.

Firstly, let's observe that the sum in the nominator is:

$$S = \sum_{i=1}^{n} \sum_{j=1}^{i} \binom{n}{i}\binom{n}{j} = \sum_{1 \le j \le i \le n} \binom{n}{i}\binom{n}{j} = \sum_{1 \le i \le j \le n} \binom{n}{i}\binom{n}{j}$$

Now, the idea is to combine the last two forms so that every summand corresponding to $j \le i$ or $i \le j$ appears once. This way we can free these indexes from each other. We just need to exclude those where $i = j$ as these are counted twice.

$$2S = \sum_{1 \le i \le j \le n} \binom{n}{i}\binom{n}{j} + \sum_{1 \le j \le i \le n} \binom{n}{i}\binom{n}{j} =$$

$$= \sum_{1 \le i \le j \le n} \binom{n}{i}\binom{n}{j} + \sum_{1 \le j < i \le n} \binom{n}{i}\binom{n}{j} + \sum_{1 \le i = j \le n} \binom{n}{i}\binom{n}{j} =$$

$$= \sum_{1 \le i,j \le n} \binom{n}{i}\binom{n}{j} + \sum_{i=1}^{n} \binom{n}{i}^2 =$$

$$= (2^n - 1)^2 + \binom{2n}{n} - 1.$$

Hence

$$S = \frac{(2^n - 1)^2 + \binom{2n}{n} - 1}{2}.$$

From this we can substitute S to inequality to obtain:

$$\frac{(2^n - 1)^2 + \binom{2n}{n} - 1}{2(2^n - 1)} \le \frac{5}{8} 2^n$$

$$2^{2n} - 2 \cdot 2^n + 1 + \binom{2n}{n} - 1 \le \frac{5}{4} 2^n (2^n - 1)$$

$$\binom{2n}{n} \le \frac{4^n}{4} + \frac{3 \cdot 2^n}{4}.$$

Using Stirling approximation can be shown that $\binom{2n}{n} \le \frac{4^n}{\sqrt{\pi n}}$ which is less than or equal to $\frac{4^n}{4} + \frac{3 \cdot 2^n}{4}$, for $n \ge 1$. This proves this theorem.

□

4.2 Different approach to average state complexity

As mentioned in the end of our previous work and also can be seen in Table 4.1. there is a quite big gap between calculated average state complexity and the real one. That is why other, possibly better, approaches are examined.

n	Actual average state complexity	Average state complexity obtained by Equation **??**	2^n
2	1.39	2.34	4
3	2.25	4.86	8
4	3.81	9.80	16
5	6.37	19.55	32
6	10.24	38.83	64
7	15.81	77.01	128

Table 4.1: Comparison of average state complexities

One of the viable approaches is to analyze the resulting state complexity of individual automata as was already presented in Figure 2.2. There we can see that there are many automata with state complexity 1, and also quite a lot with state complexity of less than or equal to n which is 6 and than the rest. Here the idea is to count automata for these 3 categories and that computes the average state complexity from the obtained numbers. To this day, we have only experimented with 2 categories: 1 and the rest, which did not provide better results than the already known results.

Another not really examined approach could be to look at the rate the state complexity grows between individual values of n. Then if we know that the average state complexity, consider the value computed from generation, and we also know that the state complexity from n-state ${}_k$DFAs to $(n+1)$-state ${}_k$DFAs grows let's say twice, then we can say that the average state complexity for the $(n+1)$-state ${}_k$DFAs is two times the calculated value. By induction, we can calculate the average state complexity for every n.

Bibliography

[1] M. Holzer, K. Salomaa, and S. Yu. On the state complexity of k-entry deterministic finite automata. *Journal of Automata, Languages and Combinatorics*, 6:453–466, 2001.

[2] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to automata theory, languages, and computation 2. edn.* 2. Addison-Wesley, 2003.