P. J. Safarik University

Faculty of Science

ANALYSIS AND DESIGN OF THE SOLUTION

MATERIALS FOR SEMINAR

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

Košice 2023

Šimon Huraj

Contents

List of Figures

List of Tables

Chapter 1

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.

1.1 Parallel generation

The problem of generating families and their subsequent processing is in fact excellent candidate for leveraging all the benefits of parallel computing. If the work is cleverly divided between multiple processors we can safely bypass all the problems when it comes to concurrent execution of a program. In our case, this is easily done. 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. In the end, the only thing left to do is to correctly merge all of the gathered statistics, which is rather an easy problem as it is done sequentially in the main thread of the program.

1.2 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.

1.3 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 1.1 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 1.2 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 1.3 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.

1.4 Bigger alphabets

In Section 1.3 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 can help to identify 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

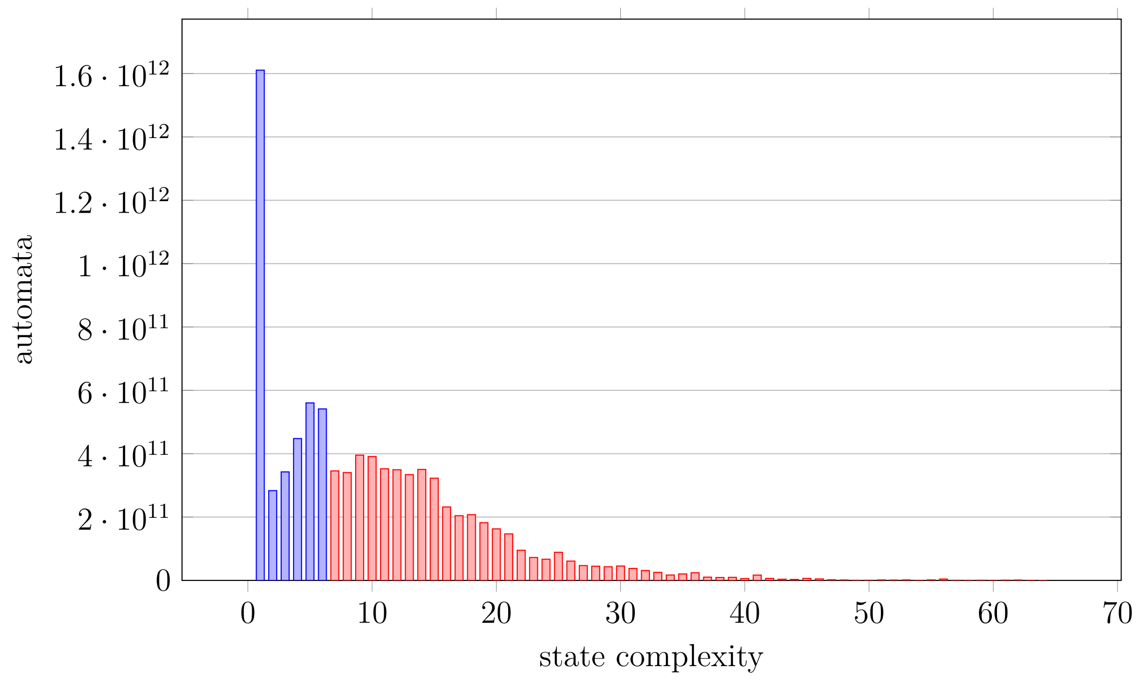


Figure 1.1: Six state ${}_kDFAs$ - state complexity

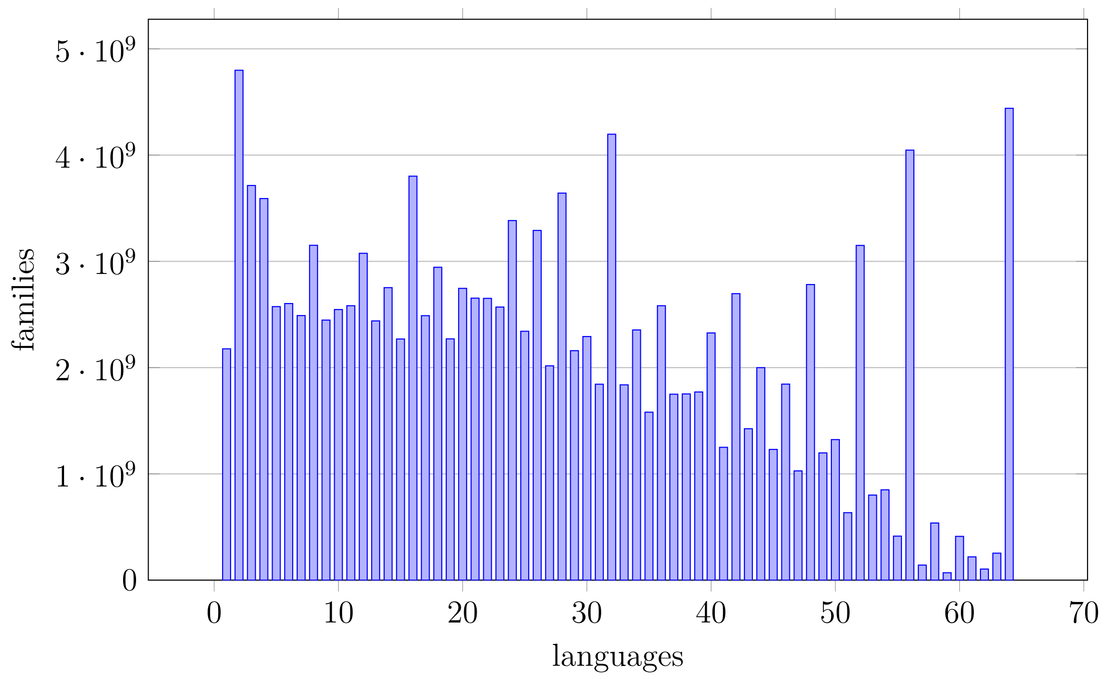


Figure 1.2: Six state ${}_kDFAs$ - number of families with number of non-equivalent languages

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

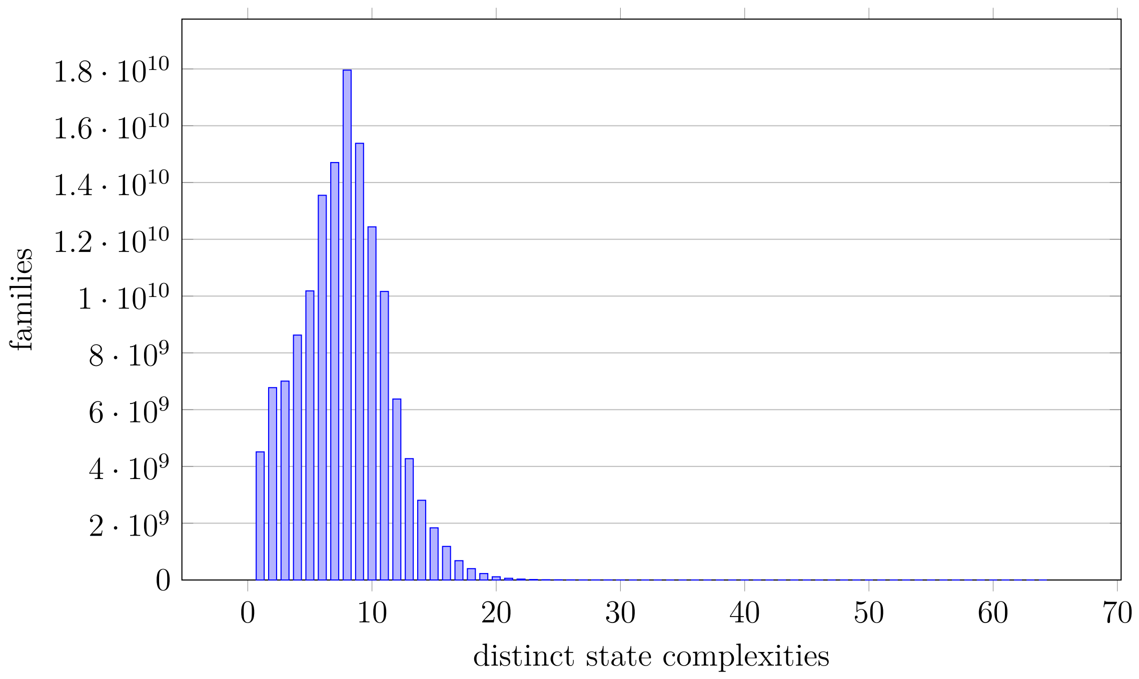


Figure 1.3: Six state ${}_kDFAs$ - number of families with particular number of distinct state complexities

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 1.1: Results for 2 state automata with alphabet of size 3

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 1.1 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.

Table 1.2 is similar to Table 1.1, the only difference is that it shows results for 2 state automata with alphabet size 4.

Next Tables 1.3, 1.4 and 1.5 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	1662	1	256	1	514
2	1090	2	288	2	190
3	320	3	130	3	320
4	0	4	350	4	0

Table 1.2: Results for 2 state automata with alphabet of size 3

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 1.3: 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.

1.5 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

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 1.4: 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 1.5: Results for 4 state automata with alphabet of size 3

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 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 of minimization "wins". We looked at the latter of the two operations more closely. 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 2

Magic Numbers

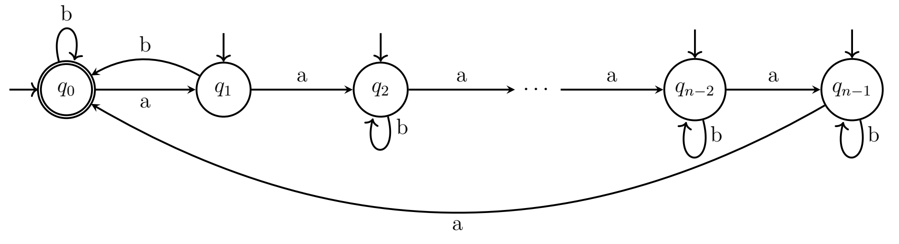
In this chapter, we will deal with so-called magic numbers. Let's say we have 3 state ${}_k$DFA. We will construct a minimal deterministic DFA and observe how many states it has. Clearly, it must be a number from the range 1 to $2^n - 1$ Now the question stands: Is there a number of states that can't 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. In other words, 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.

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

Lemma 2.1 *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 $S \nsubseteq T$, if not we swap S and T. Next, take $s = 0, \cdots, n-1$, $q_s \in S$, $q_s \notin T$ 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 2.2 (Upper bound for state complexity) Let us take 3-state automaton from previous Lemma.

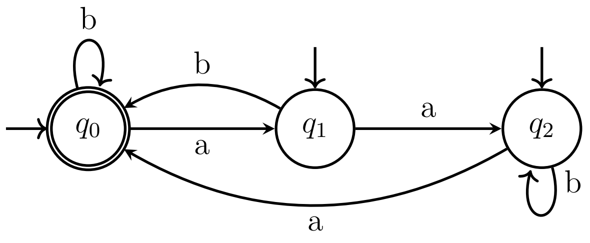


Figure 2.4: Three state automaton from Lemma 2.1.

And determinize it

It can be seen that that every state in determinized automaton is reachable, and every state can be distinct from all other states.

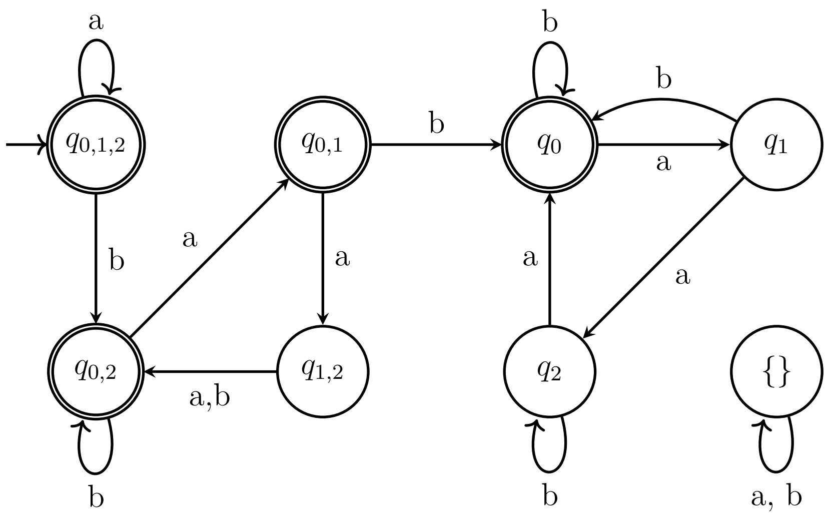


Figure 2.5: Three state automaton determinized

Now we will show the stronger statement. However, in this case, the alphabet of exponential size is required.

Lemma 2.3 *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.*

2.1 Linear alphabet

The first approach that we tried to prove this Lemma included a linearly big alphabet considering the number of states n. The approach is quite simple, take some small n, in our case $n = 3$, and find an automata that will determinize and minimize to gradually 1, 2, ..., $2^n - 1$ state DFA. Next, find a special letter or letters that will help build the whole range for every n. Let us consider a n-state ${}_k$DFA M that after determinization and minimization will have m states. Now, add a state with these special letters to M to obtain a $n+1$-state ${}_k$DFA D. One set of letters should be defined in a way that D after minimization and determinization will have $2m$ states. The second set of letters should get us a $2m + 1$ state automaton. For better understanding see Figure 2.6. There is depicted a case where we start with a 3-state ${}_k$DFA that has 2 states after determinization and minimization. Then the first set of special letters with one extra state is used to form a 4-state ${}_k$DFA that has 4 stats after determinization and minimization. The second set of letters is used to get 5 state DFA. We can continue this way to build automata as if we filled in a table for n and m.

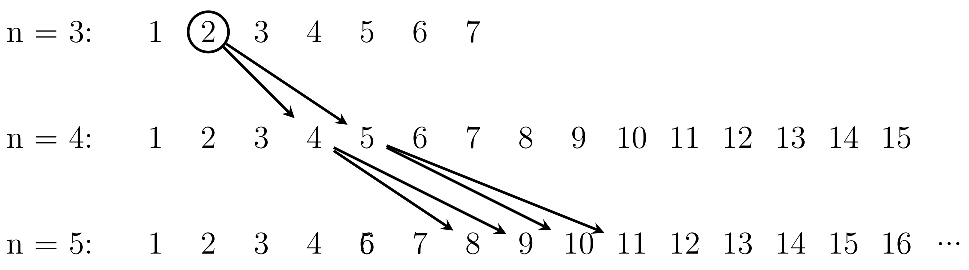

n = 3: 1 2 3 4 5 6 7
n = 4: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
n = 5: 1 2 3 4 6 7 8 9 10 11 12 13 14 15 16 ···

Figure 2.6: Magic numbers with linear alphabet

This approach seems really straightforward. The only hard thing about it is to find the special letters. In our case, this turned out to be very difficult. That is why we opted for another solution. In this case, however, a larger alphabet is needed, and that is of the exponential size. So the proof is not optimal. On the other hand, we do not provide a statement that it is not possible to do better.

2.2 Exponential alphabet

The second approach is based on simply creating a special automaton for every pair (n, m). We always start with n states but with the empty alphabet and every state is initial. And now we can start adding letters, two for every additional state that we want to obtain after determinization and minimization. As one can see m is from range $1, \cdots, 2^n - 1$ to at the worst case we would need 2^n letters in our alphabet. The meaning of the 2 letters is as follows. One letter is required to ensure reachability from the initial state to the state representing some subset of all states. The second one is used for the distinction between already reached states. The only tricky thing in this approach is that we need to add those letters in a specific order. Firstly we need to add letters that will ensure the reachability of single-element subsets. Then, double element subsets and so on.

Proof.
Show how to get from n state kDFA to DFA with m states simply by adding letters to alphabet, one for reachability and one for distinguishability. ...

□

Chapter 3

Average state complexity and other results

3.1 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 3.1 *Let $n \geq 5$ be odd number, then average state complexity of a language represented by an n-state ${}_k$DFA 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 3.2 *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.

□

3.2 Different approach to average state complexity

As mentioned in the end of our previous work and also can be seen in Table 3.1. 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 3.1: 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 1.1. 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.