0.1 Introduction

Finite automata are a long established computation model that dates back to sources such as [10] and [13]. A known problem for finite automata is state space reduction, referring to the search of a language-equivalent automaton which uses fewer states than the original object. For deterministic finite automata (DFA), not just reduction but minimization was solved in [7]. Regarding nondeterministic finite automata (NFA), [8] proved the PSPACE-completeness of the minimization problem, which is why reduction algorithms such as [4] and [1] are a popular alternative.

In his prominent work [2], Büchi introduced the model of Büchi automata (BA) as an extension of finite automata to read words of one-sided infinite length. As these ω-automata tend to have higher levels of complexity in comparison to standard finite automata, the potential gain of state space reduction is even greater. Similar to NFAs, exact minimization for deterministic Büchi automata was shown to be NP-complete in [14] and spawned heuristic approaches such as [14], [9], or [5].

As [16] displays, deterministic Büchi automata are a strictly weaker model than nondeterministic Büchi automata. It is therefore interesting to consider different models of ω-automata in which determinism is possible while maintaining enough power to describe all ω-regular languages. Parity automata (PA) are one such model, a mixture of Büchi automata and Moore automata ([11]), that use a parity function rather than the usual acceptance set. [12] showed that deterministic parity automata are in fact sufficient to recognize all ω-regular languages. As for DBAs, the exact minimization problem for DPAs is NP-complete ([14]).

Our goal in this publication is to develop new algorithms for state space reduction of DPAs, partially adapted from existing algorithms for Büchi or Moore automata. We perform theoretical analysis of the algorithms in the form of proofs of correctness and analysis of run time complexity, as well as practical implementation of the algorithms in code to provide empirical data for or against their actual efficiency.

Chapter 1

Experiments

This chapter focuses on the analysis of the different algorithms in the form of implementation and empirical data.

All techniques for state space reduction were implemented in C++14. The source code can be found in []. The computer used to run the tests was an Arch Linux 4.19.4 64 bit machine powered by an AMD Ryzen 5 1600 processor and 16 GB DDR4-2400 RAM.

1.1 Test Automata

Several automata generated randomly using different parameters were used in the testing process. Three major different techniques of generation were used:

1. Use Spot ([]) to generate a random DPA. (called *gendet*)
2. Use Spot to generate a random non-deterministic Büchi automaton and use Spot again to convert it to a DPA. (called *detspot*)
3. Use Spot to generate a random non-deterministic Büchi automaton and use nbautils ([]) to convert it to a DPA. (called *detnbaut*)

Figures 2.1, 2.2, 2.3, and 2.4 present some information about the automata. Regarding the number of states we stopped the generation at about 100 states, as most algorithms become too slow at that point. For the detnbaut set, the path refinement procedure can be sped up which is why the upper limit is higher. We elaborate this point in the relevant section.

The number of priorities is rather small in general. For gendet, we intentionally limited the number to 4 to mimic real world behavior that can be found in the detnbautils set. In comparison to nbautils, Spot does not perform priority reduction on the determinization result which explains why the detspot automata use more priorities in general.

The number of SCCs is consistently small among all three sets. detspot and gendet contain a few examples of automata with up to 60 SCCs but these are extreme outliers. This low number of SCCs is important to consider, as multiple of the algorithms such as the skip merger perform better the less connected the automaton is.

Finally, the average size of equivalence classes $\mathfrak{C}(\equiv_L)$. Again, this is of relevance to some of the reduction algorithms such as LSF as only states which are language equivalent can be merged. We can observe that the gendet set almost entirely consists of trivial classes (i.e. classes of size 1) while detspot and detnbaut show more promise in that regard.

1.2 Skip Merger

1.3 Delayed Simulation

For reduction via delayed simulation we simply built a representative merge of the input DPA w.r.t. μ_{de}. Figure 2.5 shows the relative number of states that could be removed with this method. On gendet, the procedure barely achieved anything; on detspot, it worked poorly; on nbaut, there was a decent success with more than half of the automata being reduced by some percentage.

Figure 2.6 shows that, as one could expect, the reduction tends to improve as the automaton itself grows in size. The exception of course is the gendet set as there is hardly any reduction to be seen at all. On the other hand, figures 2.7 and 2.8 show no clear correlation between the reduction and the number of SCCs or priorities.

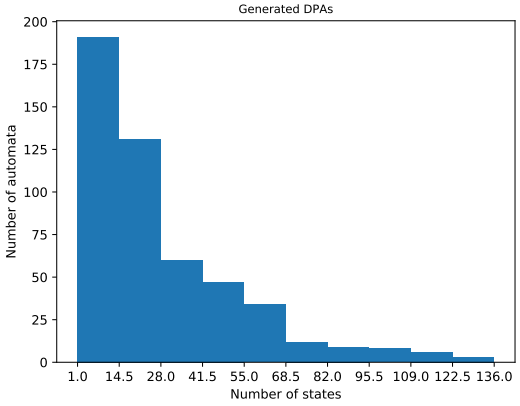
Finally, figure 2.9 displays the run time required to perform this reduction on the automaton as a function of the input size. In the gendet graph, the quadratic influence of $|Q|$ is clearly visible; in the other two less so. We conjecture that the number of removed states has a notable influence in our implementation as well, which would explain said variance.

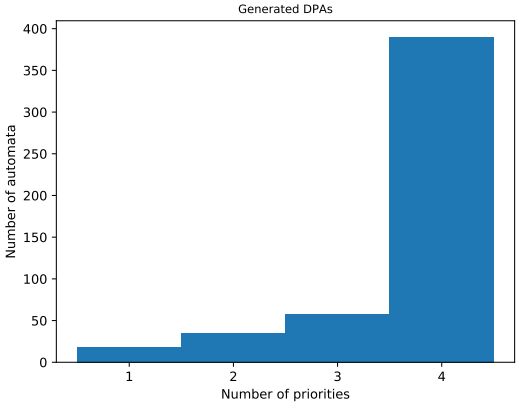
1.4 Iterated Moore

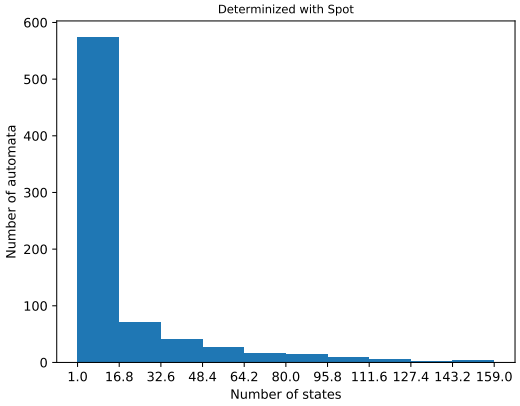
1.5 Path Refinement

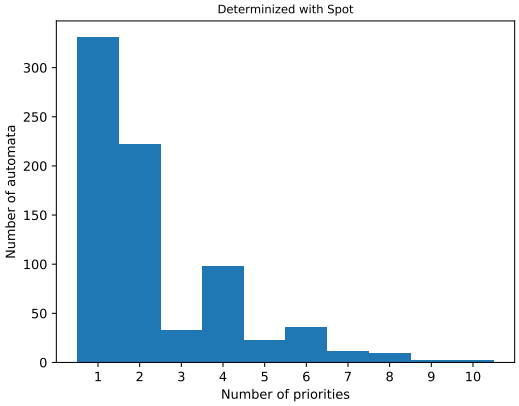
1.6 Threshold Moore

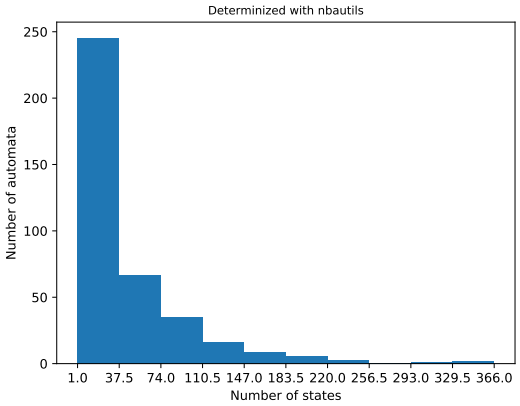
1.7 LSF











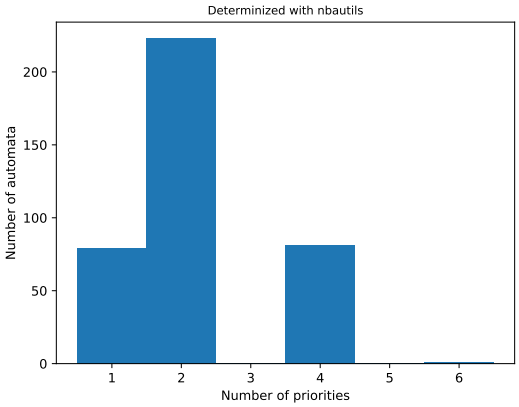
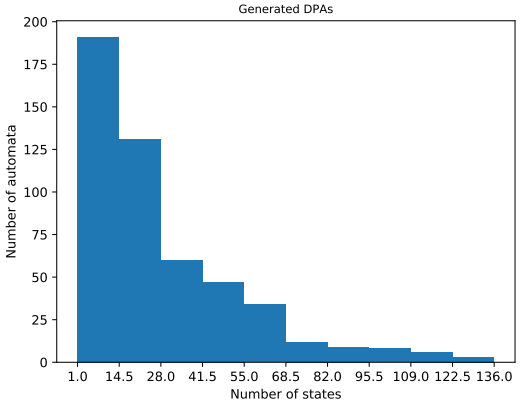
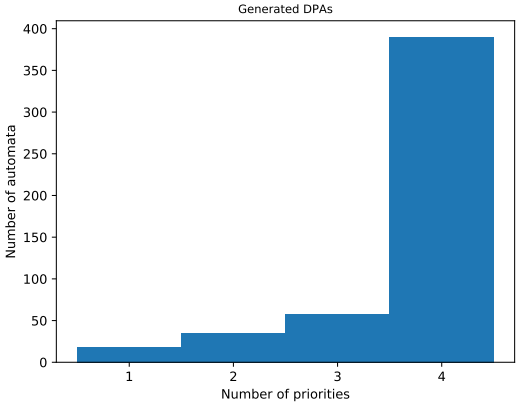
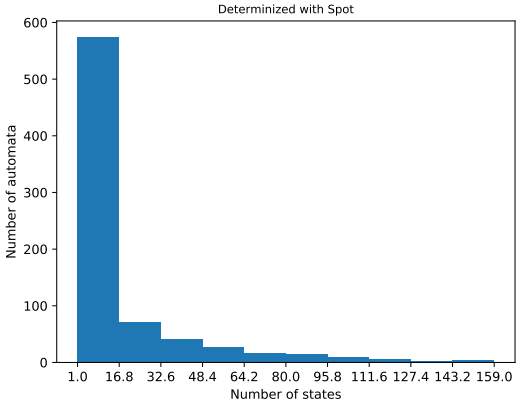


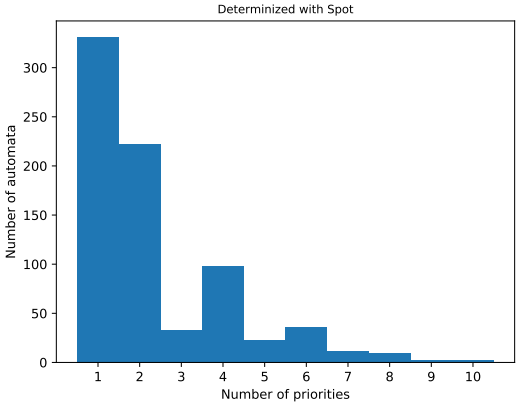
Figure 1.1: Sizes of the automata in the testing environment.

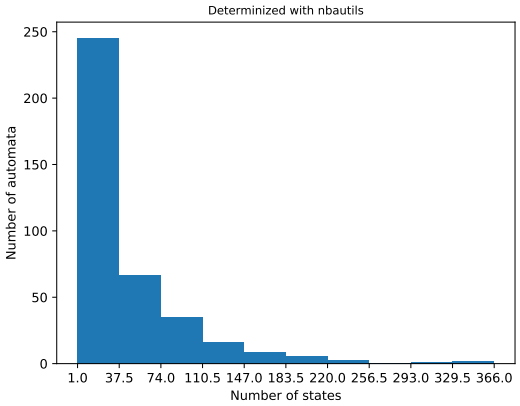
Figure 1.2: Number of priorities in the automata in the testing environment.











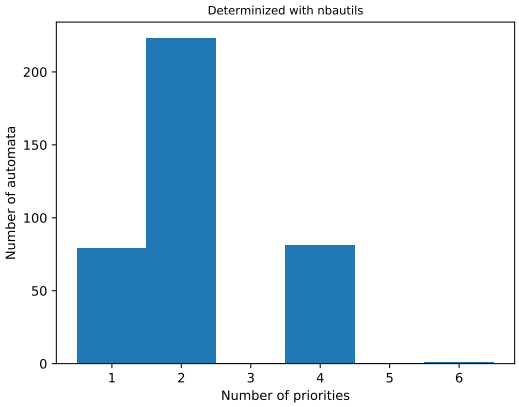
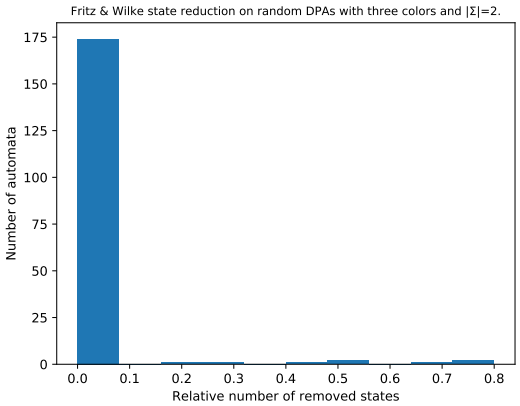
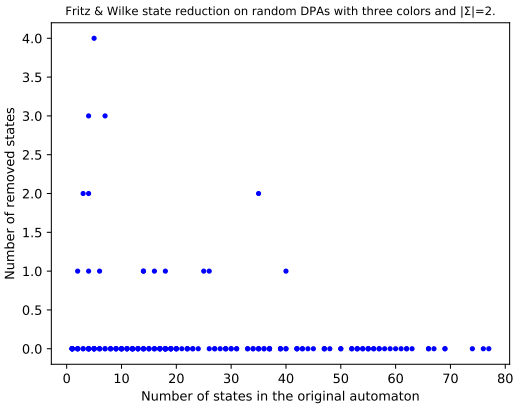
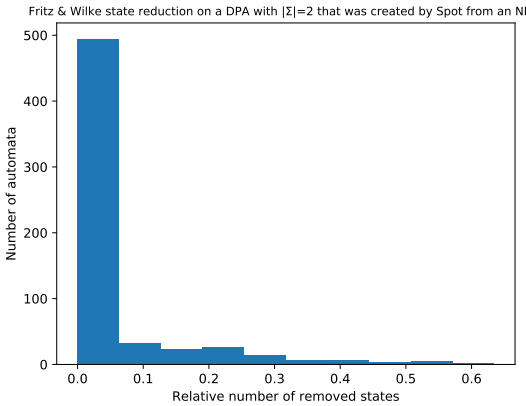


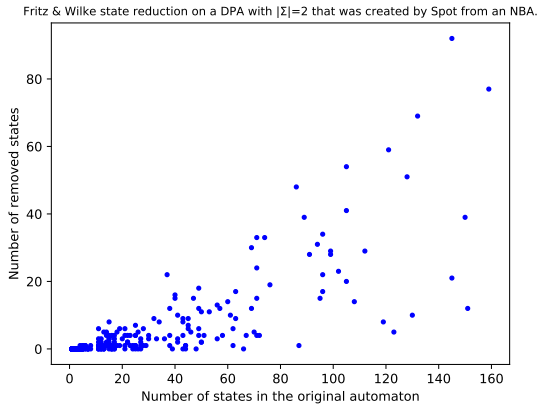
Figure 1.3: Number of SCCS in the automata in the testing environment.

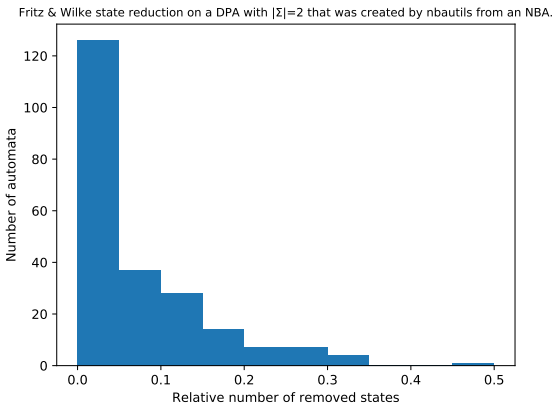
Figure 1.4: Average size of $\equiv_L$-classes of the automata in the testing environment.











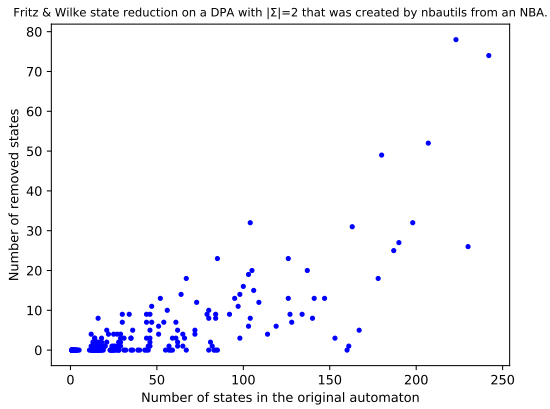
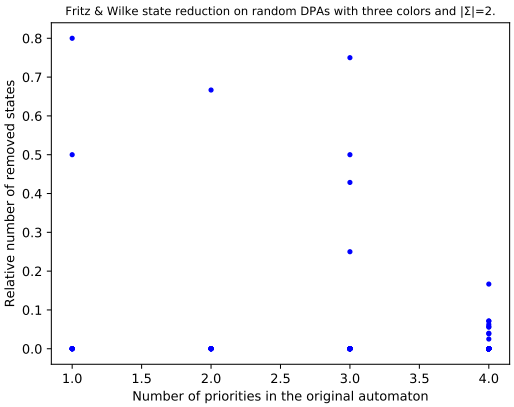
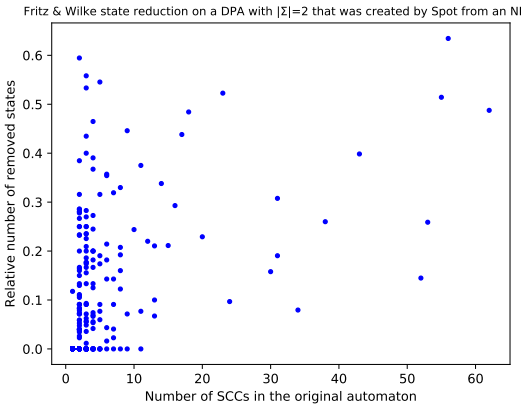
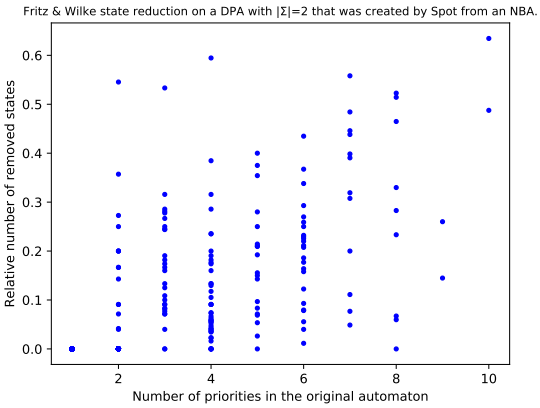


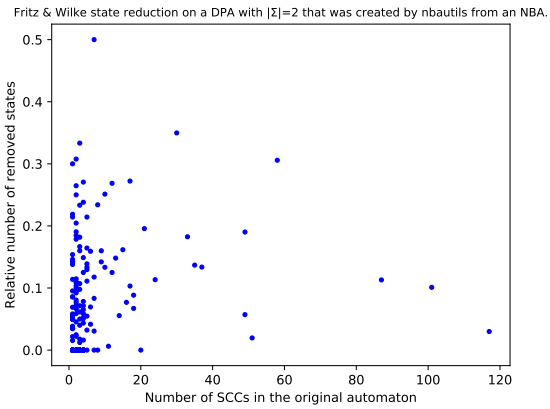
Figure 1.5: Relative state reduction of different automata using delayed simulation.

Figure 1.6: Relative state reduction of different automata using delayed simulation.









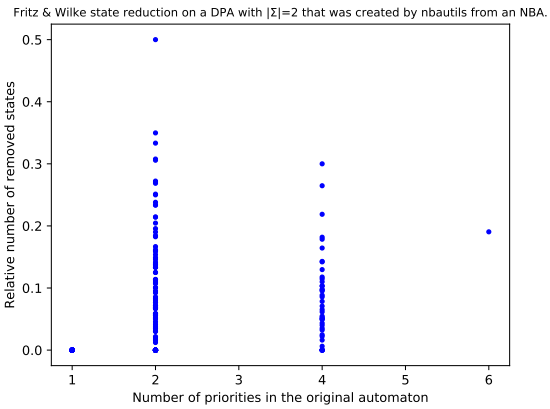


Figure 1.7: Relative state reduction of different automata using delayed simulation.

Figure 1.8: Relative state reduction of different automata using delayed simulation.

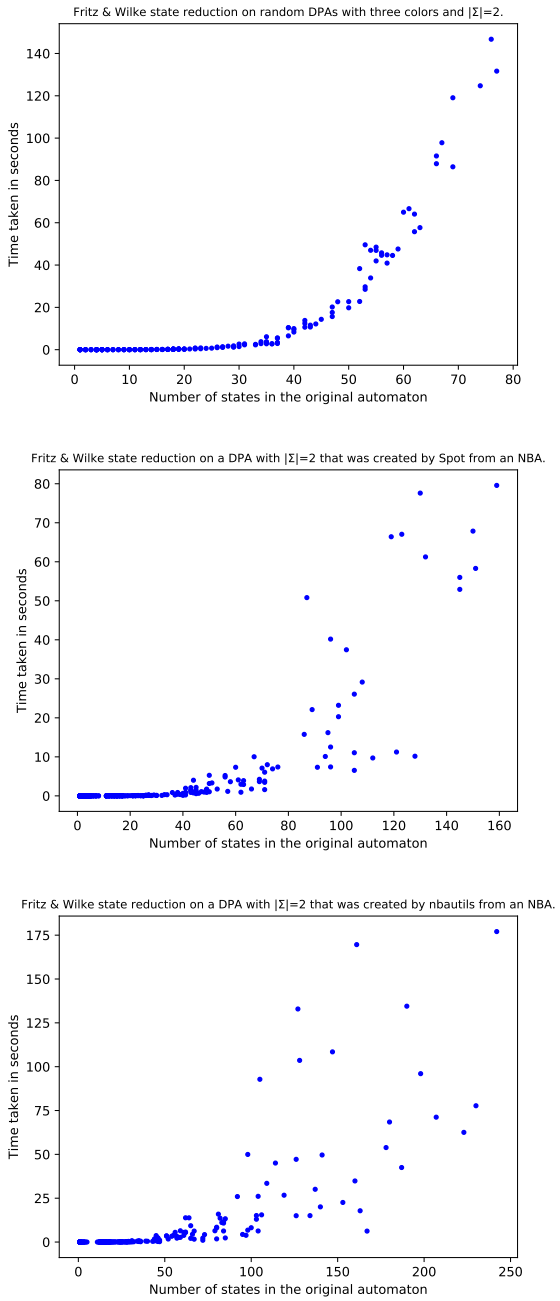


Figure 1.9: Run time of delayed simulation.

Bibliography

[1] Filippo Bonchi and Damien Pous. Checking nfa equivalence with bisimulations up to congruence. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '13, pages 457–468, New York, NY, USA, 2013. ACM.

[2] Julius Richard Büchi. On a decision method in restricted second order arithmetic. 1966.

[3] Olivier Carton and Ramón Maceiras. Computing the rabin index of a parity automaton. *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications*, 33(6):495–505, 1999.

[4] J.-M. Champarnaud and F. Coulon. Nfa reduction algorithms by means of regular inequalities. *Theoretical Computer Science*, 327(3):241 – 253, 2004. Developments in Language Theory.

[5] Kousha Etessami, Thomas Wilke, and Rebecca A. Schuller. Fair simulation relations, parity games, and state space reduction for büchi automata. In *Automata, Languages and Programming*, pages 694–707, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[6] Monika Rauch Henzinger and Jan Arne Telle. Faster algorithms for the nonemptiness of streett automata and for communication protocol pruning. In *Algorithm Theory — SWAT'96*, pages 16–27, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[7] John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. *An N Log N Algorithm for Minimizing States in A Finite Automaton*, page 15, 01 1971.

[8] Tao Jiang and B. Ravikumar. Minimal nfa problems are hard. In *Automata, Languages and Programming*, pages 629–640, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[9] Richard Mayr and Lorenzo Clemente. Advanced automata minimization. In *POPL 2013*, Oct 2012.

[10] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. 1943. *Bulletin of mathematical biology*, 52 1-2:99–115; discussion 73–97, 1990.

[11] Edward F. Moore. Gedanken-experiments on sequential machines. In Claude Shannon and John McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, Princeton, NJ, 1956.

[12] Nir Piterman. From nondeterministic büchi and streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3(3), 2007.

[13] Michael O Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–125, 04 1959.

[14] Sven Schewe. Beyond Hyper-Minimisation—Minimising DBAs and DPAs is NP-Complete. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 400–411, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[15] M. Sharir. A strong-connectivity algorithm and its applications in data flow analysis. *Computers & Mathematics with Applications*, 7(1):67 – 72, 1981.

[16] Wolfgang Thomas. Handbook of theoretical computer science (vol. b). chapter Automata on Infinite Objects, pages 133–191. MIT Press, Cambridge, MA, USA, 1990.