Regular Operations on Unambiguous Finite Automata: A Graph Theory Approach

3rd year project report

Candidate Number: 1035502

University of Oxford
Honour School of Computer Science Part B
Trinity 2021

Project Supervisor:
Stefan Kiefer

Word Count:
8154

Abstract

Various classes of automata have been a topic of research in computer science for decades. There has recently been interest in unambiguous finite automata (UFAs), a subset of nondeterministic finite automata (NFAs) and a superset of deterministic finite automata (DFAs). One of the main points of interest are the state complexities of various regular operations on UFAs and, in particular, the complement operation. The best known upper bound for the number of states needed for complementing a UFA with n states (i.e. the unambiguous state complexity of the complement operation) is $2^{0.7856n+\log(n)}$, shown by [Jirásek et al., 2018], who lowered it from the trivial upper bound of 2^n. We improve that upper bound to $(n+1)e^{n/e} \leq 2^{0.5308n+\log(n+1)}$. Our proof consists of showing a correspondence between a certain relevant property of UFAs and the sets of cliques and cocliques of undirected simple graphs, followed by obtaining an upper bound for the minimum of the sizes of those two sets, a problem in the domain of extremal graph theory, which has, to our knowledge, not been studied so far. We believe that both of these steps might be of independent interest. Furthermore, we use this new result to improve the upper bounds for the unambiguous state complexities of union and square (a language concatenated with itself) to $m + n \cdot 2^{0.5308 \cdot m+\log(m+1)}$ (n and m being the unambiguous state complexities of the two languages) and $2^{1.5308n+\log(n+1)}$, respectively. We also show a non-trivial lower bound for the worst-case behaviour (in the number of states produced) of the particular complement construction used ($2^{0.5n+0.5\log(n+1)-1}$), which is in $2^{0.5n+\omega(1)}$. Finally, we state a conjecture for a slightly tighter upper bound for unambiguous state complexity of the complement operation ($2^{0.5n+0.5\log(n+1)}$) based on data obtained from a brute-force search routine. This conjectured upper bound would match our proven lower bound (for the particular construction) up to a constant factor. Since we have obtained several novel results, a paper [Author and Kiefer, 2021][1] expanding on them has been uploaded to `arXiv.org` and submitted for publication.

[1]Project author name replaced with “Author” to preserve anonymity.

Acknowledgements

I would like to thank my supervisor, Stefan Kiefer, for his support and especially for proposing this topic to me, as I wholly enjoyed working on it. Furthermore, I would like to thank one of my tutors, Christoph Haase, for putting me in touch with Stefan after I expressed interest in working on a research project in Models of Computation.

Contents

List of Figures

Chapter 1

Introduction

For many types of language acceptors (automata, grammars, Turing machines, etc.) the relationship between deterministic and nondeterministic versions of the machine is of significant interest, in particular, the classes of languages they recognize and their succinctness. An intermediate notion between the two is an unambiguous machine. Such a machine can make nondeterministic choices, but it has at most one accepting computation on every input string.

This notion has been widely studied for context-free grammars and it is well known that the classes of languages accepted by deterministic, unambiguous and nondeterministic CFGs all have strict inclusions between them, in this order. On the other hand, it is known that both deterministic and nondeterministic finite automata accept the same class of languages, regular languages. However, NFAs can be exponentially more succinct than DFAs. Furthermore, UFAs can be exponentially separated from both, as shown by [Leung, 2004], i.e. there are families of languages for which UFAs are exponentially more succinct than DFAs and other such families for which NFAs are exponentially more succinct than UFAs. This has led to UFAs and other types of unambiguous automata being of interest for various other applications. One such example is [Baier et al., 2016], who used unambiguous Büchi automata in a polynomial-time algorithm for model checking discrete-time Markov chains against ω-regular specifications. This algorithm was later improved upon by [Kiefer and Widdershoven, 2019].

One question that arises naturally is about the minimum number of states needed to build a DFA, UFA, or NFA that accepts some regular language L. We call this quantity the deterministic, unambiguous or nondeterministic, respectively, state complexity of L. In particular, we are interested in the state complexities of various regular operations such as complement, union, intersection, reversal, star and others. Note that by the state complexity of an operation, say complement, we mean the worst-case state complexity of the complement of a language L in terms of the state complexity of L (usually denoted as n). All of these are known exactly for DFAs and NFAs, but not for UFAs. [Jirásek et al., 2018] managed to obtain exact results for most regular operations except complement, union and square (i.e. a language concatenated with itself). Furthermore, they got upper bounds for the unambiguous state complexities for those three operations, by first improving the upper bound for complement from the trivial bound of 2^n to $2^{0.7856n+\log(n)}$, and then using this result to show upper bounds for union ($m + n \cdot 2^{0.7856 \cdot m+\log(m)}$, where n and m are the unambiguous state complexities of the two languages) and square ($2^{1.7856n+\log(n)}$). On the other hand, [Raskin, 2018] showed a super-polynomial lower bound for the unambiguous state complexity of complement. This refuted a conjecture by [Colcombet, 2015] that it may be possible to complement UFAs with only a polynomial blow up in the number of states.

We build upon [Jirásek et al., 2018]'s work, starting with the same construction for complement, but performing a much more detailed analysis on it. Since the construction involves taking the smaller between the forward and the backward determinizations of the UFA, the relevant properties (corresponding to the determinizations' states) are the UFA's reachable sets of states (i.e. states which can be reached from an initial state by the same string) and co-reachable sets of states (i.e. states which can reach an accepting state by the same string). However, one characterization of UFAs is that no pair of reachable and co-reachable sets may contain more than one common state. This structure is what our proof is based on. We first define maximal cases for the pairs of sets of reachable and co-reachable sets and then show that each of them is equal to the pair of the set of cliques and the set of cocliques (independent sets) of some graph determined by the UFA. This transforms the original question about UFAs into an extremal graph theory problem. This idea is completely new and was in part inspired by observations on the results of computer-aided searches for worst-case witnesses for the UFA complement problem.

This graph theory problem, upper bounding the minimum of the number of cliques and the number of cocliques of a graph, does not appear to have been studied, which is surprising, since the result that, informally, a graph cannot have both many cliques and many cocliques, seems intuitive and useful. Another related problem, bounding the product of the number of cliques and the number of cocliques, would have also led to a useful result for the minimum of the two, but that has not been studied either. During this project, we developed an approach towards obtaining an upper bound for the problem in question – examine some constrained property of a graph and its cograph (e.g. the total number of edges of the two or the maximal cliques of each) and then use this constraint to analyze the graph and the cograph separately from each other. This first led us to a bound of $2^{0.7071n} + n$, based on a result by [Wood, 2007], but then we identified more appropriate work by [Zykov, 1949] and, after using our approach and a few calculus proofs, it led to an even better upper bound of $2^{0.5308n+\log(n+1)}$.

This bound then translates to an equivalent upper bound for the unambiguous state complexity of the complement operation. Finally, we apply this new result to get tighter upper bounds for the unambiguous state complexities of the union ($m + n \cdot 2^{0.5308 \cdot m+\log(m+1)}$) and square ($2^{1.5308n+\log(n+1)}$) operations, again using [Jirásek et al., 2018]'s constructions.

Since the complement operation is our main object of study and the construction we use for it (which is due to [Jirásek et al., 2018]) is likely the most natural one (and is the only one proposed so far), we also show a non-trivial lower bound for the maximum number of states it could produce. There is a not hard to show lower bound of $2^{0.5n}$ (where n is the number of states of the UFA which is being complemented). We improve this to $2^{0.5n+0.5\log(n+1)-1}$, which is asymptotically higher, since it is in $2^{0.5n+\omega(1)}$.

As already mentioned, before undertaking all of this, several optimized brute-force searches were conducted, in order to form concrete expectations for the amount of improvement possible to the already existing upper bound. Those showed promising results and provided possible paths to tackling the problem. The main search routine used will be described in chapter 4. Its results also suggest that a slightly better upper bound of $2^{0.5n+0.5\log(n+1)}$ than the one we show is still possible using the same construction. We make this explicit in Conjecture 3.4.2. Note that this conjectured upper bound is asymptotically equal to our lower bound for the maximum number of states the examined complement construction could produce. Therefore, if true, this upper bound would be asymptotically equal to the optimal one achievable with this approach.

Since we have obtained several novel results, contributing to both automata theory and extremal graph theory, which improve upon the published literature, we have decided that our results are worthy of publication as well. To that end, a paper [Author and Kiefer, 2021][2] expanding on them (see note below) has been written. It has been uploaded to `arXiv.org` and submitted to the journal "Information Processing Letters".

Note after completion of the project

After this report had been written, an insight by my supervisor, combined with further improvements by myself, led to a proof of Conjecture 3.4.1. Our joint paper [Author and Kiefer, 2021][2] now contains the reduction to the extremal graph theory problem from this project report, followed by a proof of Conjecture 3.4.1, thus leading to an upper bound of $2^{0.5n+0.5\log(n+1)}$ for the unambiguous state complexity of the complement operation. By Proposition 3.3.1, also included in the paper, this bound is tight up to a constant factor of 2 for this particular complement construction

[2]Project author name replaced with "Author" to preserve anonymity.

Chapter 2

Preliminaries

2.1 Finite automata and regular languages

Most of this section should be familiar to anyone who has studied the Models of Computation course, but there are some exceptions. Furthermore, for convenience, we use slightly different (but equivalent) definitions of NFAs and DFAs, mostly following the conventions of [Jirásek et al., 2018].

A familiarity with strings and languages is assumed of the reader. Readers not familiar with these concepts should refer to appendix B. Furthermore, since the results here are well known, we omit their proofs. Supplementary formal proofs of all results stated here can be found in appendix A.

Definition 2.1.1. A *nondeterministic finite automaton* (NFA) is 5-tuple $M = (Q, \Sigma, \delta, I, F)$ where:

- Q is a finite non-empty set of states
- Σ is a finite input alphabet
- $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation
- $I \subseteq Q$ is the set of initial states
- $F \subseteq Q$ is the set of final states

Each element $(p, a, q) \in \delta$ is called a *transition* of M.

Note that we allow multiple initial states and disallow ϵ-transitions (as those can be replaced without needing to add any extra states by having additional symbol transitions and accepting states).

For the rest of this section, whenever we introduce an NFA M, we implicitly define it as the 5-tuple $(Q, \Sigma, \delta, I, F)$. Therefore, whenever one of Q, Σ, δ, I or F is mentioned, we are referring to the respective component of the NFA.

Definition 2.1.2. The *size of an NFA M* is $|M| = |Q|$.

Definition 2.1.3. A *computation* of an NFA M on an input string $w \in \Sigma^*$ is a sequence of states $\bar{q} \in Q^*$ such that either $w = \epsilon$ and $\bar{q} = q$ where $q \in Q$ or $w = av$ and $\bar{q} = pq\bar{p}$ where $(p, a, q) \in \delta$ and $q\bar{p}$ is a computation on v. Furthermore, a computation is *accepting* if its first state is initial (in I) and its last state is final (in F).

We also write $p \xrightarrow{w}_M q$ to mean that M has a computation on w whose first and last states are p and q, respectively.

Note that, since we do not allow ϵ-transitions, we have no need to include the sequence of transitions (from δ) in the definition of a computation, as they are unambiguously defined by the sequence of states together with the input string.

Definition 2.1.4. The *language accepted* by an NFA M, $L(M) \subseteq \Sigma^*$, is the set of input strings for which there exists an accepting computation. We say that these strings are *accepted* by M.

Definition 2.1.5. A *regular language* is a language L which is accepted by some NFA M, i.e. $L = L(M)$.

Definition 2.1.6. The *reverse of an NFA* $M = (Q, \Sigma, \delta, I, F)$ is another NFA $M^R = (Q, \Sigma, \delta^R, F, I)$ where $\delta^R = \{\, (q, a, p) \mid (p, a, q) \in \delta \,\}$ is the *reverse of the transition relation* of M.

Lemma 2.1.7. *The language of the reverse of an NFA M is equal to the reverse of the language of M. Formally, $L(M^R) = (L(M))^R$.*

Definition 2.1.8. The set of states *reachable by* $w \in \Sigma^*$ in an NFA M is $\{\, q \in Q \mid \exists p \in I \cdot p \xrightarrow{w}_M q \,\}$. Similarly, the set of states *co-reachable by* w is $\{\, q \in Q \mid \exists p \in F \cdot q \xrightarrow{w}_M p \,\}$. A set is simply *reachable* if it is reachable by some string w and similarly for *co-reachable.* Finally, we define $\mathcal{R}_M$ to be the set of all reachable sets of M and similarly for $\mathcal{C}_M$ – the set of all co-reachable sets of M.

Lemma 2.1.9. *The set of reachable sets of the reverse of an NFA M is equal to the set of co-reachable sets of M and vice versa. Formally, $\mathcal{R}_{M^R} = \mathcal{C}_M$ and $\mathcal{C}_{M^R} = \mathcal{R}_M$.*

Definition 2.1.10. A *deterministic finite automaton* (DFA) is an NFA M where $|I| = 1$ and for each $p \in Q$ and $a \in \Sigma$ there is exactly one $q \in Q$ such that $(p, a, q) \in \delta$.

Note that unlike [Jirásek et al., 2018] we do not allow incomplete DFAs, meaning that pairs of states and input symbols with no transition associated with them cannot exist.

Definition 2.1.11. An *unambiguous finite automaton* (UFA) is an NFA M where for each $w \in \Sigma^*$ there is at most one accepting computation.

Lemma 2.1.12. *An NFA M is a UFA if and only if $|S \cap T| \leq 1$ for every reachable set S and every co-reachable set T.*

Then a consequence of the result above and Lemma 2.1.9 is the following corollary:

Corollary 2.1.13. *The reverse of a UFA is also a UFA.*

Lemma 2.1.14. *Every reachable set of a DFA M has size exactly* 1.

This lemma, together with Lemma 2.1.12, leads to the following corollary.

Corollary 2.1.15. *Every DFA is also a UFA.*

Definition 2.1.16. The *complement of a DFA* $M = (Q, \Sigma, \delta, I, F)$ is another DFA $\overline{M} = (Q, \Sigma, \delta, I, Q \setminus F)$.

Note that $\overline{M}$ is clearly a DFA, since its transition relation and initial states are the same as the ones of M, which is a DFA.

Lemma 2.1.17. *The language of the complement of a DFA M is equal to the complement of the language of M. Formally, $L(\overline{M}) = \overline{L(M)}$.*

Definition 2.1.18. The *subset construction of an NFA* $M = (Q, \Sigma, \delta, I, F)$ is a DFA $2^M = (\mathcal{R}_M, \Sigma, \delta_{2^M}, \{I\}, F_{2^M})$ where $\delta_{2^M} = \{\, (S, a, \{\, q \mid \exists p \in S \cdot (p, a, q) \in \delta \,\}) \mid S \in \mathcal{R}_M \wedge a \in \Sigma \,\}$ and $F_{2^M} = \{\, S \mid S \in \mathcal{R}_M \wedge S \cap F \neq \emptyset \,\}$.

Note that 2^M is clearly a DFA, since it has only one initial state and its transition relation has exactly one transition per pair of state and input symbol. We also need to show that this transition relation is valid (i.e. $\delta_{2^M} \subseteq \mathcal{R}_M \times \Sigma \times \mathcal{R}_M$), which we show in Corollary A.0.11.

We write 2^M for the construction, since traditionally 2^Q is used for its set of states. We use only $\mathcal{R}_M$, since those are the only reachable states in the resulting DFA.

Lemma 2.1.19. *Given an NFA M, the unique state reachable by $w \in \Sigma^*$ in 2^M is equal to the set reachable by w in M.*

Lemma 2.1.20. *The language of the subset construction DFA of an NFA M is equal to the language of M. Formally, $L(2^M) = L(M)$.*

Definition 2.1.21. The *unambiguous state complexity of a regular language* L, $\operatorname{usc}(L)$, is the size of the smallest UFA M which accepts L, i.e. $L(M) = L$.

2.2 Graphs and cliques

Definition 2.2.1. An *(undirected, simple, finite) graph* is a pair $G = (V, E)$ where V is a finite set of nodes and E is a finite set of edges. Here, an *edge* is a set of two distinct nodes. We define $Poss_V$ to be the *set of all possible edges*, i.e. $Poss_V = \{\, \{u, v\} \mid u, v \in V \wedge u \neq v \,\}$. Thus, $E \subseteq Poss_V$.

Definition 2.2.2. A *clique (complete subgraph)* of a graph $G = (V, E)$ is a set $S \subseteq V$ such that there is an edge between every pair of distinct nodes in S. Formally, $\forall u, v \in S \cdot (u \neq v \to \{u, v\} \in E)$. Additionally, a clique S is said to be a *k-clique* if $|S| = k$. Similarly, a *coclique (independent set)* of G is a set $T \subseteq V$ such that there is no edge between any pair of distinct nodes in T. Formally, $\forall u, v \in S \cdot (u \neq v \to \{u, v\} \notin E)$. Finally, we define *Cliques*(G) to be the set of all cliques of G and *Cocliques*(G) to be the set of all cocliques of G.

Note that we do not require cliques to be maximal.

Definition 2.2.3. The *cograph* of a graph $G = (V, E)$ is a graph $\overline{G} = (V, \mathit{Poss}_V \setminus E)$.

Lemma 2.2.4. *The set of cocliques of a graph G is equal to the set of cliques of its cograph and vice versa. Formally, $\mathit{Cocliques}(G) = \mathit{Cliques}(\overline{G})$ and $\mathit{Cliques}(G) = \mathit{Cocliques}(\overline{G})$.*

Proof. We prove the first equality by considering each possible element $S \subseteq V$ of the two sets:

$$\begin{aligned}
& S \in \mathit{Cocliques}(G) & \\
\Leftrightarrow\ & \forall u, v \in S \cdot (u \neq v \to \{u, v\} \notin E) & \text{Definition 2.2.2} \\
\Leftrightarrow\ & \forall u, v \in S \cdot (u \neq v \to \{u, v\} \in \mathit{Poss}_V \setminus E) & \text{Since } E \subseteq \mathit{Poss}_V \\
\Leftrightarrow\ & S \in \mathit{Cliques}(\overline{G}) & \text{Definitions 2.2.2 and 2.2.3} \\
& \text{Therefore } \mathit{Cocliques}(G) = \mathit{Cliques}(\overline{G}) &
\end{aligned}$$

The proof for the second equality is analogous. □

Lemma 2.2.5. *A graph G with n nodes and a largest clique of size ω can have no more than $\binom{\omega}{k}\frac{n^k}{\omega^k}$ k-cliques.*

Proof. Shown by [Zykov, 1949]; see also Corollary 1. in [Fisher and Ryan, 1992]. □

Turán graphs were first introduced by [Turán, 1941], who showed that they maximize the number of edges a graph with n nodes and a maximal clique of size ω can have. Since then, they have also been shown to be the solutions of other extremal graph theory problems, including the one in Lemma 2.2.5.

Definition 2.2.6. A *Turán graph* $T(n, \omega)$ when n is divisible by ω is a graph with n nodes split into ω equally sized *components*. It has an edge between a pair of vertices, if and only if they are in separate components. An example of such a graph, $T(6, 3)$, is shown in fig. 2.1.

There is also a slightly more general notion of a Turán graph $T(n, \omega)$ for arbitrary n and ω, but it is not important to us, so we do not state it here.

The next two propositions establish tightness the bound from Lemma 2.2.5 on Turán graphs.

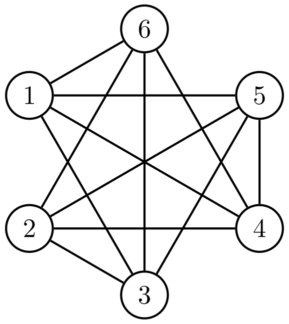

Figure 2.1: A Turán graph $T(6,3)$. Its components are $\{1,2\}$, $\{3,4\}$ and $\{5,6\}$.

Proposition 2.2.7. *A Turán graph $T(n,\omega)$ when n is divisible by ω has a largest clique of size ω.*

Proof. Clearly, there is a clique of size ω which contains one node from each component, since there is an edge between any pair of nodes from distinct components. However, there cannot be a clique of size larger than ω, since by the pigeonhole principle, it would contain at least two nodes from the same component and there would not be an edge between them. □

Proposition 2.2.8. *A Turán graph $T(n,\omega)$ when n is divisible by ω has $\binom{\omega}{k}\frac{n^k}{\omega^k}$ k-cliques.*

Proof. Any clique can contain at most one node from a given component, since there are no edges between nodes from the same component. Therefore, a k-clique contains nodes from exactly k distinct components. There are $\binom{\omega}{k}$ ways to choose them. Then, from each of the k chosen components, a single node out of the $\frac{n}{\omega}$ has to be chosen. There are $\frac{n^k}{\omega^k}$ ways to do that. Therefore, multiplying the two, there are $\binom{\omega}{k}\frac{n^k}{\omega^k}$ cliques of size k. □

Chapter 3

Proofs of upper bounds

3.1 Complement

This section contains our main original contributions, with only Lemma 3.1.1 being a reproduction of a result by [Jirásek et al., 2018].

We fix, for the remainder of this section, a set Q with $|Q| = n$.

Lemma 3.1.1. *Let $L \subseteq \Sigma^*$ be a regular language accepted by a UFA M. Then $\operatorname{usc}(\overline{L}) \leq \min\{|\mathcal{R}_M|, |\mathcal{C}_M|\}$.*

Proof. Consider the following two UFAs: $M_1 = \overline{2^M}$ and $M_2 = \overline{2^{M^R}}^R$. First, note that these are valid complement constructions (they are performed on DFAs) by Definition 2.1.18. Furthermore, the resulting NFAs are indeed UFAs by Definition 2.1.16 and Corollaries 2.1.13 and 2.1.15. Finally, by Lemmas 2.1.7, 2.1.17 and 2.1.20, $L(M_1) = L(M_2) = \overline{L(M)} = \overline{L}$. Therefore, these are both constructions for UFAs whose language is $\overline{L}$ and thus $\operatorname{usc}(\overline{L}) \leq \min\{|M_1|, |M_2|\}$ by Definition 2.1.21. However, by construction, $\mathcal{R}_M$ is the set of states of M_1 and similarly, by Lemma 2.1.9, $\mathcal{C}_M$ is the set of states of M_2. This proves the lemma, $\operatorname{usc}(\overline{L}) \leq \min\{|\mathcal{R}_M|, |\mathcal{C}_M|\}$, from Definition 2.1.2. □

The following defines the set of possible pairs of reachable and co-reachable sets that a UFA with states Q could have without violating its unambiguousness, abstracting away the details about

UFAs and only keeping the relevant properties between them.

Definition 3.1.2. We define *Comp* to be the set of pairs of sets of reachable and co-reachable sets that do not violate the unambiguousness property of UFAs. We call such pairs *compatible*. Formally, $\mathit{Comp} = \{\, (\mathcal{R}, \mathcal{C}) \mid \mathcal{R}, \mathcal{C} \subseteq 2^Q \wedge \forall S \in \mathcal{R} \cdot \forall T \in \mathcal{C} \cdot |S \cap T| \leq 1 \,\}$.

To avoid dealing with arbitrary compatible pairs, we define the notion of maximal elements of *Comp*, i.e. ones whose $\mathcal{R}$ and $\mathcal{C}$ cannot be further enlarged.

Definition 3.1.3. We define *MaxComp* to be the set of *maximal* elements of *Comp* where a pair $(\mathcal{R}, \mathcal{C})$ is maximal if, for every set $S \subseteq Q$ not in $\mathcal{R}$, adding it to $\mathcal{R}$ would make the pair incompatible, and similarly for $\mathcal{C}$. Formally:

$$\begin{aligned} \mathit{MaxComp} = \{(\mathcal{R}, \mathcal{C}) \mid (\mathcal{R}, \mathcal{C}) \in \mathit{Comp} \\ \wedge \forall S \in 2^Q \setminus \mathcal{R} \cdot (\mathcal{R} \cup \{S\}, \mathcal{C}) \notin \mathit{Comp} \\ \wedge \forall T \in 2^Q \setminus \mathcal{C} \cdot (\mathcal{T}, \mathcal{C} \cup \{T\}) \notin \mathit{Comp}\} \end{aligned}$$

The next is an intuitive consequence of *MaxComp* containing the maximal elements of *Comp*.

Lemma 3.1.4. *Every compatible pair is a pointwise subset of a maximal compatible pair. Formally,* $\forall (\mathcal{R}, \mathcal{C}) \in \mathit{Comp} \cdot \exists (\mathcal{R}', \mathcal{C}') \in \mathit{MaxComp} \cdot (\mathcal{R} \subseteq \mathcal{R}' \wedge \mathcal{C} \subseteq \mathcal{C}')$.

Proof. We prove the lemma by induction in decreasing order of the pointwise subset relation on *Comp*:

Let $(\mathcal{R}, \mathcal{C}) \in \mathit{Comp}$:

Case $\neg \exists (\mathcal{R}', \mathcal{C}') \in \mathit{Comp} \setminus \{(\mathcal{R}, \mathcal{C})\} \cdot (\mathcal{R} \subseteq \mathcal{R}' \wedge \mathcal{C} \subseteq \mathcal{C}')$:

$\Rightarrow \neg \exists S \in 2^Q \setminus \mathcal{R} \cdot (\mathcal{R} \cup \{S\}, \mathcal{C}) \in \mathit{Comp}$ — Otherwise $\mathcal{R}' = \mathcal{R} \cup \{S\}$ and $\mathcal{C}' = \mathcal{C}$

$\wedge \neg \exists T \in 2^Q \setminus \mathcal{C} \cdot (\mathcal{T}, \mathcal{C} \cup \{T\}) \in \mathit{Comp}$ — Analogous

$\Rightarrow (\mathcal{R}, \mathcal{C}) \in \mathit{MaxComp}$ — Definition 3.1.3

Case $\exists(\mathcal{R}', \mathcal{C}') \in \mathit{Comp} \setminus \{(\mathcal{R}, \mathcal{C})\} \cdot (\mathcal{R} \subseteq \mathcal{R}' \wedge \mathcal{C} \subseteq \mathcal{C}')$:

$\exists(\mathcal{R}'', \mathcal{C}'') \in \mathit{MaxComp} \cdot (\mathcal{R}' \subseteq \mathcal{R}'' \wedge \mathcal{C}' \subseteq \mathcal{C}'')$ Inductive hypothesis

$\Rightarrow \mathcal{R} \subseteq \mathcal{R}'' \wedge \mathcal{C} \subseteq \mathcal{C}'$

□

Intuitively, the next lemma says that for a maximal compatible pair $(\mathcal{R}, \mathcal{C})$, $\mathcal{R}$ is fully determined by $\mathcal{C}$ and vice versa. Furthermore, $\mathcal{R}$ includes precisely those sets that would not make it incompatible with $\mathcal{C}$ (and analogously in the opposite direction).

Lemma 3.1.5. *If* $(\mathcal{R}, \mathcal{C}) \in \mathit{MaxComp}$*, then* $\mathcal{R} = \{\, S \mid S \subseteq Q \wedge \forall T \in \mathcal{C} \cdot |S \cap T| \leq 1 \,\}$ *and* $\mathcal{C} = \{\, T \mid T \subseteq Q \wedge \forall S \in \mathcal{R} \cdot |S \cap T| \leq 1 \,\}$.

Proof. We set we set $\mathcal{R}' = \{\, S \mid S \subseteq Q \wedge \forall T \in \mathcal{C} \cdot |S \cap T| \leq 1 \,\}$. Then we prove that $\mathcal{R} = \mathcal{R}'$ by double inclusion:

$\forall S \in \mathcal{R} \cdot (S \subseteq Q \wedge \forall T \in \mathcal{C} \cdot |S \cap T| \leq 1)$ Definition 3.1.2

$\Rightarrow \mathcal{R} \subseteq \mathcal{R}'$

For a contradiction, suppose that $\mathcal{R}' \nsubseteq \mathcal{R}$

$\Rightarrow \exists S \in \mathcal{R}' \cdot S \notin \mathcal{R}$

$\Rightarrow \forall S' \in \mathcal{R} \cup \{S\} \cdot \forall T \in \mathcal{C} \cdot |S' \cap T| \leq 1$ Definition 3.1.2 and definition of $\mathcal{R}'$

$\Rightarrow (\mathcal{R} \cup \{S\}, \mathcal{C}) \in \mathit{Comp}$ Definition 3.1.2

$\Rightarrow (\mathcal{R}, \mathcal{C}) \notin \mathit{MaxComp}$ Definition 3.1.3

Therefore $\mathcal{R}' \subseteq \mathcal{R}$ By contradiction

The proof for the $\mathcal{C}$ equality is analogous. □

Then the following corollary is a straightforward consequence of the previous lemma:

Corollary 3.1.6. *If $(\mathcal{R}, \mathcal{C})$ is a maximal compatible pair, then both $\mathcal{R}$ and $\mathcal{C}$ are subset closed. Formally, $\forall S \in \mathcal{R} \cdot \forall S' \subseteq S \cdot S' \in \mathcal{R}$ and similarly for $\mathcal{C}$.*

From now on, we consider graphs on Q. For brevity, we write just *Poss* to refer to $Poss_Q$ (the set of all possible edges of such graphs). Thus, 2^{Poss} is isomorphic to the set of all such graphs.

Lemma 3.1.7. *If $(\mathcal{R}, \mathcal{C}) \in MaxComp$, then every possible edge $e \in Poss$ is either in $\mathcal{R}$ or in $\mathcal{C}$. Formally, $\forall e \in Poss \cdot (e \in \mathcal{R} \leftrightarrow e \notin \mathcal{C})$.*

Proof. The $e \in \mathcal{R} \rightarrow e \notin \mathcal{C}$ direction is trivial, since $|e \cap e| = |e| = 2 > 1$, and thus $e \in \mathcal{R} \wedge e \in \mathcal{C}$ is a contradiction (from Definition 3.1.2).

For the opposite direction:

$$\begin{array}{lr}
\text{Suppose that } e \notin \mathcal{C} & \\
\Rightarrow \forall T \in \mathcal{C} \cdot e \not\subseteq T & \text{Corollary 3.1.6} \\
\Rightarrow \forall T \in \mathcal{C} \cdot T \cap e \subset e & \\
\Rightarrow \forall T \in \mathcal{C} \cdot |T \cap e| \leq |e| - 1 = 1 & \\
\Rightarrow e \in \mathcal{R} & \text{Lemma 3.1.5}
\end{array}$$

□

From the above lemma, the next corollary follows:

Corollary 3.1.8. *If $(\mathcal{R}, \mathcal{C}) \in MaxComp$, then $\mathcal{C} \cap Poss = Poss \setminus (\mathcal{R} \cap Poss)$.*

The following key lemma shows that the defining property of a maximal compatible pair $(\mathcal{R}, \mathcal{C})$ is which (unordered) pairs of elements of Q occur together in elements of $\mathcal{R}$. This is equivalent to the defining property of a graph on Q, i.e. which (unordered) pairs of nodes have an edge between them. Thinking of $(\mathcal{R}, \mathcal{C})$ as the reachable and co-reachable sets of states of some UFA, the lemma is saying that only pairwise-reachability is important (as opposed to set-reachability).

Lemma 3.1.9. *There is a bijection f from MaxComp to 2^{Poss}, defined by $f(\mathcal{R}, \mathcal{C}) = \mathcal{R} \cap Poss$ with an inverse $f^{-1}(E) = (\mathcal{R}', \mathcal{C}')$ where $\mathcal{R}' = \{ S \mid S \subseteq Q \wedge \forall e \in Poss \setminus E \cdot |S \cap e| \leq 1 \}$ and $\mathcal{C}' = \{ T \mid T \subseteq Q \wedge \forall e \in E \cdot |e \cap T| \leq 1 \}$.*

Proof. We need to show that f^{-1} is indeed the inverse of f. First, we show that it is its left inverse, i.e. that $\mathcal{R}' = \mathcal{R}$ and $\mathcal{C}' = \mathcal{C}$ where $(\mathcal{R}', \mathcal{C}') = f^{-1}(f(\mathcal{R}, \mathcal{C})) = f^{-1}(\mathcal{R} \cap Poss)$. We prove the second equality (the one for $\mathcal{C}$) by double inclusion, considering an arbitrary $T \subseteq Q$:

First, suppose that $T \in \mathcal{C}$

$\Rightarrow \forall S \in \mathcal{R} \cdot |S \cap T| \leq 1$ Lemma 3.1.5

$\Rightarrow \forall S \in \mathcal{R} \cap Poss \cdot |S \cap T| \leq 1$ Since $S \in \mathcal{R} \subseteq S$

$\Rightarrow T \in \mathcal{C}'$

Therefore $\mathcal{C} \subseteq \mathcal{C}'$

Second, suppose that $T \notin \mathcal{C}$

$\Rightarrow \neg \forall S \in \mathcal{R} \cdot |S \cap T| \leq 1$ Lemma 3.1.5

$\Rightarrow \exists S \in \mathcal{R} \cdot |S \cap T| \geq 2$

Consider an arbitrary $e \subseteq S \cap T$ such that $|e| = 2$

$\Rightarrow e \in Poss$ Definition 2.2.1 since $|e| = 2 \wedge e \subseteq Q$

$\wedge\, e \in \mathcal{R}$ Corollary 3.1.6 since $e \subseteq S \cap T \subseteq S \in \mathcal{R}$

$\Rightarrow e \in \mathcal{R} \cap Poss$

$\wedge\, |e \cap T| = 2$ Since $e \subseteq S \cap T \subseteq T$ and thus $e \cap T = e$

$\Rightarrow \exists e \in \mathcal{R} \cap Poss \cdot |e \cap T| \geq 2$

$\Rightarrow \neg \forall e \in \mathcal{R} \cap Poss \cdot |e \cap T| < 1$

$\Rightarrow T \notin \mathcal{C}'$

Therefore $\mathcal{C}' \subseteq \mathcal{C}$

The proof for the first equality is analogous, since $\mathit{Poss} \setminus (\mathcal{R} \cap \mathit{Poss}) = \mathcal{C} \cap \mathit{Poss}$ by Corollary 3.1.8.

Lastly, we need to show that f^{-1} is also the right inverse of f, i.e. that $f(f^{-1}(E)) = E$:

$$\begin{aligned}
& f(f^{-1}(E)) \\
&= \{\, S \mid S \subseteq Q \wedge \forall e \in \mathit{Poss} \setminus E \cdot |S \cap e| \leq 1 \,\} \cap \mathit{Poss} \\
&= \{\, e \mid e \in \mathit{Poss} \wedge \forall e' \in \mathit{Poss} \setminus E \cdot |e \cap e'| \leq 1 \,\} && \text{Since } \mathit{Poss} \subseteq Q \\
&= \{\, e \mid e \in \mathit{Poss} \wedge \forall e' \in \mathit{Poss} \setminus E \cdot e \neq e' \,\} && \text{Since } |e| = |e'| = 2 \text{ by Definition 2.2.1} \\
&= \{\, e \mid e \in \mathit{Poss} \wedge e \notin \mathit{Poss} \setminus E \,\} \\
&= \{\, e \mid e \in \mathit{Poss} \cap E \,\} \\
&= \{\, e \mid e \in E \,\} && \text{Since } E \subseteq \mathit{Poss} \\
&= E
\end{aligned}$$

□

The next lemma builds upon the previous one and states that we can identify the elements of $\mathcal{R}$ and $\mathcal{C}$ with the cliques and cocliques of the graph corresponding to $(\mathcal{R}, \mathcal{C})$.

Lemma 3.1.10. *For any $E \subseteq \mathit{Poss}$, $f^{-1}(E) = (\mathit{Cliques}(Q, E), \mathit{Cocliques}(Q, E))$.*

Proof. We prove that $\mathcal{R}' = \{\, S \mid S \subseteq Q \wedge \forall e \in \mathit{Poss} \setminus E \cdot |S \cap e| \leq 1 \,\} = \mathit{Cliques}(Q, E)$ by considering each possible element $S \subseteq Q$ of the two sets:

$$\begin{aligned}
& S \in \mathcal{R}' \\
&\Leftrightarrow \forall e \in \mathit{Poss} \setminus E \cdot |S \cap e| \leq 1 && \text{Definition of } f^{-1} \text{ in Lemma 3.1.9} \\
&\Leftrightarrow \forall \{u, v\} \in \mathit{Poss} \setminus E \cdot (u \notin S \vee v \notin S) \\
&\Leftrightarrow \forall \{u, v\} \in \mathit{Poss} \cdot (\{u, v\} \notin E \rightarrow (u \notin S \vee v \notin S)) \\
&\Leftrightarrow \forall \{u, v\} \in \mathit{Poss} \cdot ((u \in S \wedge v \in S) \rightarrow \{u, v\} \in E) \\
&\Leftrightarrow \forall u, v \in Q \cdot (u \neq v \rightarrow ((u \in S \wedge v \in S) \rightarrow \{u, v\} \in E)) && \text{Definition 2.2.1} \\
&\Leftrightarrow \forall u, v \in Q \cdot ((u \in S \wedge v \in S) \rightarrow (u \neq v \rightarrow \{u, v\} \in E))
\end{aligned}$$

$\Leftrightarrow \forall u, v \in S \cdot (u \neq v \to \{u, v\} \in E)$ Since $S \subseteq Q$

$\Leftrightarrow S \in \mathit{Cliques}(Q, E)$ Definition 2.2.2

Therefore $\mathcal{R}' = \mathit{Cliques}(Q, E)$

The proof for the second part is analogous by considering the cograph following Lemma 2.2.4 and Corollary 3.1.8. □

The following two lemmas are purely graph theoretical. The first of the two gives an upper bound for the total number of cliques of a graph with a bounded maximal clique size.

Lemma 3.1.11. *A graph G with n nodes and no cliques larger ω can have no more than $(\frac{n}{\omega}+1)^\omega$ cliques. Formally, $|\mathit{Cliques}(G)| \leq \left(\frac{n}{\omega}+1\right)^\omega$.*

Proof. First, we show that, if G has a largest clique of size ω, then it has at most $c(\omega)$ cliques where $c(x) = \left(\frac{n}{x}+1\right)^x$. Let $\mathit{Cliques}_k(G)$ be the set of k-cliques of G. Then:

$$
\begin{aligned}
&|\mathit{Cliques}(G)| \\
&= \left|\bigcup_{k=0}^{\omega} \mathit{Cliques}_k(G)\right| && \text{Definition 2.2.2 since the graph has no cliques larger than } \omega \\
&= \sum_{k=0}^{\omega} |\mathit{Cliques}_k(G)| && \text{Since } \mathit{Cliques}_{k_1}(G) \cap \mathit{Cliques}_{k_2}(G) = \emptyset \text{ for } k_1 \neq k_2 \\
&\leq \sum_{k=0}^{\omega} \binom{\omega}{k} \frac{n^k}{\omega^k} && \text{Lemma 2.2.5} \\
&= \sum_{k=0}^{\omega} \binom{\omega}{k} \left(\frac{n}{\omega}\right)^k 1^{\omega-k} \\
&= \left(\frac{n}{\omega}+1\right)^\omega && \text{Binomial theorem}
\end{aligned}
$$

Now we need to show that the statement holds for any graph G with no cliques larger than ω, i.e. with a largest clique of size $\omega' \leq \omega$. To that end, we prove that $c(x)$ is non-decreasing in x for

$1 \leq x \leq n$, from which $c(\omega') \leq c(\omega)$ follows:

$$\frac{dc}{dx} = \left(\frac{n}{x}+1\right)^x \left(\ln\left(\frac{n}{x}+1\right) - \frac{n}{n+x}\right)$$

Now consider the two parts of the product:

$$\left(\frac{n}{x}+1\right)^x \geq 0 \qquad \text{Since } n, x \geq 1$$

$$\ln\left(\frac{n}{x}+1\right) - \frac{n}{n+x}$$

$$\geq \frac{\frac{n}{x}}{1+\frac{n}{x}} - \frac{n}{n+x} \qquad \text{Since } \ln(1+x) \geq \frac{x}{1+x} \text{ [3]}$$

$$= \frac{n}{x+n} - \frac{n}{n-x}$$

$$= 0$$

Therefore $\frac{dc}{dx} \geq 0$

□

The following proposition establishes tightness of the bound above on Turán graphs.

Proposition 3.1.12. *A Turán graph $T(n, \omega)$ when n is divisible by ω has $\left(\frac{n}{\omega}+1\right)^\omega$ cliques.*

Proof. The proof consists of an application of the Binomial Theorem to Proposition 2.2.8, analogous to the first part of the proof of the previous lemma. □

The lemma below is also a purely graph theoretical result. Intuitively, it is saying that a graph cannot have both a large number of cliques and a large number of cocliques. We believe that this result is likely to be of independent interest.

Lemma 3.1.13. *For a graph $G = (V, E)$ with n nodes, the following bound on its cliques and cocliques holds:* $\min\{|\mathit{Cliques}(G)|, |\mathit{Cocliques}(G)|\} \leq (n+1)e^{n/e}$.

[3]

$$\ln(1+x) = \int_1^{1+x} \frac{1}{y}\, dy \geq x\left(\min_{1\leq y\leq 1+x} \frac{1}{y}\right) = \frac{x}{1+x} \text{ when } x \geq 0$$

Proof. Let X be the largest clique (or one of the largest cliques) of G and, similarly, Y be a largest coclique of G. Also, let $|X| = k$ and $|Y| = l$.

We start by getting a bound on $|\mathit{Cliques}(G)|$. Note that, since Y is a coclique, G has no edges between any pair of nodes in Y. Thus, no clique of G can contain more than one node from Y. Therefore, for every $S \in \mathit{Cliques}(G)$, either $S \subseteq V \setminus Y$ or $S = S' \cup \{y\}$ where $S' \subseteq V \setminus Y$ and $y \in Y$. So, we define the following subgraph of G: $G \setminus Y = (V \setminus Y, E \cap \mathit{Poss}_{V \setminus Y})$. Then it follows that $\mathit{Cliques}(G) \subseteq \mathit{Cliques}(G \setminus Y) \cup \{\, S' \cup \{y\} \mid S' \in \mathit{Cliques}(G \setminus Y) \wedge y \in Y \,\}$ and thus $|\mathit{Cliques}(G)| \leq (l+1)|\mathit{Cliques}(G \setminus Y)| \leq (n+1)|\mathit{Cliques}(G \setminus Y)|$. However, no clique of $G \setminus Y$ can have size greater than k, since X is a largest clique of G. Therefore, by Lemma 3.1.11 $|\mathit{Cliques}(G \setminus Y)| \leq (\frac{n-l}{k} + 1)^k$ and thus $|\mathit{Cliques}(G)| \leq (n+1)(\frac{n-l}{k} + 1)^k$.

Analogously, by Lemma 2.2.4, $|\mathit{Cocliques}(G)| \leq (n+1)(\frac{n-k}{l} + 1)^l$. Therefore, if we define $g(a,b) = (\frac{n-b}{a} + 1)^a$ for $1 \leq a, b \leq n$, we can obtain the following bound on the quantity of interest: $\min\{|\mathit{Cliques}(G)|, |\mathit{Cocliques}(G)|\} \leq (n+1)\min\{g(k,l), g(l,k)\}$.

Trivially, g is decreasing in b (since $a \geq 1$). Therefore, if $k \leq l$, then $g(k,l) \leq g(k,k)$, and, if $l \leq k$, then $g(l,k) \leq g(l,l)$. Thus, $\min\{g(k,l), g(l,k)\} \leq \max_{1 \leq x \leq n} g(x,x)$. Finally, we find $\max_{1 \leq x \leq n} g(x,x)$ by examining the critical points of $g(x,x) = (\frac{n-x}{x} + 1)^x = (\frac{n}{x})^x$:

Case $x = 1$:

$$g(1,1) = (\frac{n}{1})^1 = n$$

Case $x = n$:

$$g(n,n) = (\frac{n}{n})^n = 1$$

Case $\dfrac{dg(x,x)}{dx} = 0$:

$$\frac{dg(x,x)}{dx} = \left(\ln\frac{n}{x} - 1\right)\left(\frac{n}{x}\right)^x = 0$$

$$\Rightarrow \ln\frac{n}{x} = 1 \qquad\qquad \text{Since } \left(\frac{n}{x}\right)^x > 0$$

$$\Rightarrow x = \frac{n}{e}$$
$$\Rightarrow g(x,x) = \left(\frac{n}{\frac{n}{e}}\right)^{n/e} = e^{n/e}$$

$$\text{Therefore } \max_{1\leq x\leq n} g(x,x) = e^{n/e} \qquad\qquad \text{Since } e^{n/e} < n \text{ has no solutions}^{4}$$

Thus, we obtain the desired upper bound: $\min\{|\mathit{Cliques}(G)|, |\mathit{Cocliques}(G)|\} \leq (n+1)e^{n/e}$.

$\square$

Finally, we prove our main result, by chaining all of our lemmas one after another.

Theorem 3.1.14. *If L is a regular language over Σ with $\operatorname{usc}(L) = n$, then $\operatorname{usc}(\overline{L}) \leq (n+1)e^{n/e}$ and thus $\operatorname{usc}(\overline{L}) \leq 2^{0.5308n+\log(n+1)}$.*

Proof. Let M be a UFA with states Q that accepts L. Then:

$$\begin{aligned}
&\forall S \in \mathcal{R}_M \cdot \forall T \in \mathcal{C}_M \cdot |S \cap T| \leq 1 && \text{Lemma 2.1.12}\\
\Rightarrow\ &(\mathcal{R}_M, \mathcal{C}_M) \in \mathit{Comp} && \text{Definition 3.1.2}\\
\Rightarrow\ &\exists(\mathcal{R}', \mathcal{C}') \in \mathit{MaxComp} \cdot (\mathcal{R}_M \subseteq \mathcal{R}' \wedge \mathcal{C}_M \subseteq \mathcal{C}') && \text{Lemma 3.1.4}
\end{aligned}$$

$$\begin{aligned}
&\operatorname{usc}(\overline{L}) && \\
\leq\ &\min\{|\mathcal{R}_M|, |\mathcal{C}_M|\} && \text{Lemma 3.1.1}\\
\leq\ &\min\{|\mathcal{R}'|, |\mathcal{C}'|\} && \text{Since } |\mathcal{R}_M| \leq |\mathcal{R}'| \wedge |\mathcal{C}_M| \leq |\mathcal{C}'|\\
=\ &\min\{|\mathcal{R}''|, |\mathcal{C}''|\} \text{ where } (\mathcal{R}'', \mathcal{C}'') = f^{-1}(f(\mathcal{R}', \mathcal{C}')) && \text{Lemma 3.1.9}\\
=\ &\min\{|\mathit{Cliques}(Q, f(R', C'))|, |\mathit{Cocliques}(Q, f(R', C'))|\} && \text{Lemma 3.1.10}\\
\leq\ &(n+1)e^{n/e} && \text{Lemma 3.1.13}\\
\leq\ &2^{0.5308n+\log(n+1)} &&
\end{aligned}$$

[4]Clearly, $e^{n/e}$ grows faster than n, so we only need to verify the claim for small values. Doing that, we find that the two functions touch at a single point between 2 and 3, but n is never larger than $e^{n/e}$.

□

Note that by Proposition 3.1.15 (below) any maximal compatible pair is a possible pair of sets of reachable and co-reachable sets that an actual UFA M could have. Therefore, the only two places where we lose tightness of the bound (excluding the final numerical approximation) are the initial construction in Lemma 3.1.1 and the bound on the cliques and cocliques in Lemma 3.1.13. However, not much further improvement is possible using this approach, since on some (not necessarily minimal for their language) automata, this construction produces results with $2^{0.5n}$ states. One such case is when $Q = Q_1 \cup Q_2$ with $|Q_1| = |Q_2| = n/2$, $\mathcal{R}_M = 2^{Q_1}$ and $\mathcal{C}_M = 2^{Q_2}$. A stronger $2^{0.5n+\omega(1)}$ lower bound for the worst-case behaviour of the considered construction is shown in section 3.3.

Proposition 3.1.15. *For any $(\mathcal{R}, \mathcal{C}) \in MaxComp$, there exists some UFA M such that $\mathcal{R}_M = \mathcal{R}$ and $\mathcal{C}_M = \mathcal{C}$.*

Proof. We can use one symbol of the alphabet per reachable set and one symbol per co-reachable set (though there are likely more succinct constructions): $M = (Q, \Sigma_1 \cup \Sigma_2, \delta_1 \cup \delta_2, \{q_0\}, \{q_0\})$ where q_0 is an arbitrary state in Q, $\Sigma_1 = \{1\} \times \mathcal{R}$, $\Sigma_2 = \{2\} \times \mathcal{C}$, $\delta_1 = \{\, (q_0, (1, S), q) \mid q \in S \in \mathcal{R} \,\}$ and $\delta_2 = \{\, (q, (2, T), q_0) \mid q \in T \in \mathcal{C} \,\}$.

Now we need to prove that $\mathcal{R}_M = \mathcal{R}$. Clearly, the set of sets reachable by strings $w = (1, S)$ for $S \in \mathcal{R}$ is precisely equal to $\mathcal{R}$, so $\mathcal{R} \subseteq \mathcal{R}_M$. Then we just need to show that there are no reachable sets not in $\mathcal{R}$. From the definition of δ_1, the sets reachable by words ending in $(1, S)$ for $S \in \mathcal{R}$ are either $\emptyset$ or S. On the other hand, from the definition of δ_2, the sets reachable by words ending in $(2, T)$ for $T \in \mathcal{C}$ are either $\emptyset$ or $\{q_0\}$. Furthermore, $\{q_0\}$ is also the set reachable by ϵ. Finally, S is in $\mathcal{R}$ by the definition of Σ_1, while $\emptyset$ and $\{q_0\}$ must be in $\mathcal{R}$, since $(\mathcal{R}, \mathcal{C})$ is a maximal compatible pair. Therefore, $\mathcal{R}_M \subseteq \mathcal{R}$ and thus $\mathcal{R}_M = \mathcal{R}$. The proof for $\mathcal{C}_M$ is analogous.

Finally, M is a UFA since for any reachable set $S \in \mathcal{R}_M$ and for any co-reachable set $T \in \mathcal{C}_M$, $|S \cap T| \leq 1$. This follows from Definition 3.1.2 since $\mathcal{R}_M = \mathcal{R}$ and $\mathcal{C}_M = \mathcal{C}$.

□

3.2 Other regular operations

Most of the results in this section are reproductions of [Jirásek et al., 2018]'s results, though with more details in the proofs and with us plugging in our improved bound from Theorem 3.1.14, in order to improve the final bounds.

Theorem 3.2.1. *If K and L are regular languages over Σ with* $\operatorname{usc}(K) = m$ *and* $\operatorname{usc}(L) = n$, *then* $\operatorname{usc}(K \cap L) \leq mn$.

Proof. Let $M = (Q_M, \Sigma, \delta_M, I_M, F_M)$ be a UFA with m states that accepts K and let $N = (Q_N, \Sigma, \delta_N, I_N, F_N)$ be a UFA n states that accepts L. Now we construct a UFA $M \cap N$, such that $L(M \cap N) = K \cap L$. Define $M \cap N = (Q_M \times Q_N, \Sigma, \delta_{M\cap N}, I_M \times I_N, F_M \times F_N)$ where $\delta_{M\cap N} = \{\, ((p_M, p_N), a, (q_M, q_N)) \mid (p_M, a, q_M) \in \delta_M \wedge (p_N, a, q_N) \in \delta_N \,\}$. Clearly, $M \cap N$ is an NFA. First, we prove that it accepts the correct language by considering each possible element $w \in \Sigma^*$ of the two sets $K \cap L$ and $L(M \cap N)$:

$w \in K \cap L = L(M) \cap L(N)$

$\Leftrightarrow$ $w \in L(M) \wedge w \in L(N)$

$\Leftrightarrow$ There are accepting computations on w in M and N — Definition 2.1.4

$\Leftrightarrow$ There are computations on w:
$q_M^0 q_M^1 \cdots q_M^{|w|} \in Q_M^*$ in M and $q_N^0 q_N^1 \cdots q_N^{|w|} \in Q_N^*$ in N
where $q_M^0 \in I_M \wedge q_M^{|w|} \in F_M \wedge q_N^0 \in I_N \wedge q_N^{|w|} \in F_N$ — Definition 2.1.3 and Lemma A.0.3

$\Leftrightarrow$ There is a computation on w in $M \cap N$:
$(q_M^0, q_N^0)(q_M^1, q_N^1) \cdots (q_M^{|w|}, q_N^{|w|}) \in (Q_M \times Q_N)^*$ — Formally, by induction
where $(q_M^0, q_N^0) \in I_M \times I_N \wedge (q_M^{|w|}, q_N^{|w|}) \in F_M \times F_N$ — following Definition 2.1.3

$\Leftrightarrow$ There is an accepting computation on w in $M \cap N$ — Definition 2.1.3

$\Leftrightarrow$ $w \in L(M \cap N)$ — Definition 2.1.4

Therefore $L(M \cap N) = K \cap L$

However, since the computations in M and N are unique by Definition 2.1.11, it follows that their pointwise pairing (i.e. the computation in $M \cap N$) is also unique. Therefore, $M \cap N$ is a UFA. Thus, we conclude that $\mathrm{usc}(K \cap L) \leq |Q_M \times Q_N| = mn$. $\square$

Lemma 3.2.2. *If K and L are regular languages over Σ with $K \cap L = \emptyset$, $\mathrm{usc}(K) = m$ and $\mathrm{usc}(L) = n$, then $\mathrm{usc}(K \cup L) \leq m + n$.*

Proof. Let $M = (Q_M, \Sigma, \delta_M, I_M, F_M)$ be a UFA with m states that accepts K and also let $N = (Q_N, \Sigma, \delta_N, I_N, F_N)$ be a UFA n states that accepts L. Furthermore, without loss of generality, let Q_M and Q_N be disjoint. Now we construct a UFA $M \cup N$, such that $L(M \cup N) = K \cup L$. Define $M \cup N = (Q_M \cup Q_N, \Sigma, \delta_M \cup \delta_N, I_M \cup I_N, F_M \cup F_N)$. Clearly, $M \cup N$ is an NFA. First, we prove that $M \cup N$ accepts the correct language by considering each possible element $w \in \Sigma^*$ of the two sets $K \cup L$ and $L(M \cup N)$:

$$\begin{aligned}
& w \in K \cup L = L(M) \cup L(N) && \\
\Leftrightarrow\ & w \in L(M) \vee w \in L(N) && \\
\Leftrightarrow\ & w \in L(M) && \text{WLOG since } K \cap L = \emptyset \\
\Leftrightarrow\ & \text{There is a computation on } w \text{ in } M\text{:} && \text{Definitions 2.1.3 and 2.1.4} \\
& q_M^0 q_M^1 \cdots q_M^{|w|} \in Q_M^* \text{ where } q_M^0 \in I_M \wedge q_M^{|w|} \in F_M && \text{and Lemma A.0.3} \\
\Leftrightarrow\ & \text{There is a computation on } w \text{ in } M \cup N\text{:} && \\
& q_M^0 q_M^1 \cdots q_M^{|w|} \in Q_M^* \in (Q_M \cup Q_N)^* && \text{Formally, by induction} \\
& \text{where } (q_M^0, q_N^0) \in I_M \cup I_N \wedge (q_M^{|w|}, q_N^{|w|}) \in F_M \cup F_N && \text{following Definition 2.1.3} \\
\Leftrightarrow\ & w \in L(M \cap N) && \text{Definitions 2.1.3 and 2.1.4} \\
& \text{Therefore } L(M \cup N) = K \cup L &&
\end{aligned}$$

Now note that $M \cup N$ has no transition from a state in Q_M to a state in Q_N and vice verse. Therefore, all of its computations are either in Q_M^* or in Q_N^*. Finally, we prove that $M \cup N$ is a UFA by contradiction. Suppose that there are two accepting computations, $\bar{p}$ and $\bar{q}$, on some string

$w \in \Sigma^*$. If $\bar{p}, \bar{q} \in Q_M^*$, then both are computations on w in M, which is a contradiction, since M is a UFA. Similarly, $\bar{p}, \bar{q} \in Q_N^*$ leads to a contradiction. So, without loss of generality, $\bar{p} \in Q_M^*$ and $\bar{q} \in Q_N^*$ must be the case. However, this means that there are computations on w in both M and N, which implies that $w \in K$ and $w \in L$. This is a contradiction with $K \cap L = \emptyset$. Therefore, $M \cup N$ is a UFA and thus $\mathrm{usc}(K \cup L) \leq m + n$. □

Theorem 3.2.3. *If K and L are regular languages over Σ with $\mathrm{usc}(K) = m$ and $\mathrm{usc}(L) = n$, then $\mathrm{usc}(K \cup L) \leq m + n(m+1)e^{m/e} \leq m + n \cdot 2^{0.5308 \cdot m + \log(m+1)}$.*

Proof.

$$\begin{aligned}
& \mathrm{usc}(K \cup L) & \\
= {} & \mathrm{usc}(K \cup (L \setminus K)) & \\
\leq {} & m + \mathrm{usc}(L \setminus K) & \text{Lemma 3.2.2} \\
= {} & m + \mathrm{usc}(L \cap \overline{K}) & \\
\leq {} & m + n \cdot \mathrm{usc}(\overline{K}) & \text{Theorem 3.2.1} \\
\leq {} & m + n(m+1)e^{m/e} & \text{Theorem 3.1.14} \\
\leq {} & m + n \cdot 2^{0.5308 \cdot m + \log(m+1)} &
\end{aligned}$$

□

Theorem 3.2.4. *If L is a regular language over Σ with $\mathrm{usc}(L) = n$, then $\mathrm{usc}(L^2) \leq (n+1)e^{n/e}2^n$ and thus $\mathrm{usc}(L^2) \leq 2^{1.5308n + \log(n+1)}$.*

Proof. Let $M = (Q, \Sigma, \delta, I, F)$ be a UFA with n states that accepts L. First, we construct an NFA N, such that $L(N) = L^2$. Define $N = (Q \times \{1,2\}, \Sigma, \delta_N, I_N, F \times \{2\})$ where $\delta_N = \{\, ((p,k), a, (q,k)) \mid (p,a,q) \in \delta \wedge k \in \{1,2\} \,\} \cup \{\, ((p,1), a, (q,2)) \mid q \in I \wedge \exists q' \in F \cdot (p,a,q') \in \delta \,\}$ and $I_N = I \times \{1\}$, if $I \cap F = \emptyset$, and $I_N = I \times \{1,2\}$, otherwise. Now we prove that N accepts the correct language by double inclusion.

($L(N) \subseteq L^2$) Suppose that $w \in L(N)$. It is easy to see that any accepting computation on $w \in \Sigma^*$ in N is of the form $(q_0,1)(q_1,1)\cdots(q_{s-1},1)(q_s,2)(q_{s+1},2)\cdots(q_{s+t},2)$ with $q_s \in I$, $q_{s+t} \in F$ and $q_0 \in I$, if $s > 0$. This is because, there are no transitions of the form $((p,2),a,(q,1))$ in δ_N. Now let q' be a state, such that $q' \in I \cap F$, if $s = 0$, and $q' \in F$ with $(q_{s-1}, w_s, q') \in \delta$. Such a state must exist due to the definitions of δ_N and I_N. It follows that $q_0 q_1 \cdots q_{s-1} q'$ is a computation on $u = w_1 w_2 \cdots w_s$ in M. Also, $q_s q_{s+1} \cdots q_t$ is a computation on $v = w_{s+1} w_{s+2} \cdots w_{s+t}$ in M. Furthermore, these are accepting computations, since their first states are both in I and their last states are both in F. Therefore, $u, v \in L(M) = L$ and thus $w = uv \in L^2$.

($L^2 \subseteq L(N)$) Suppose that $w \in L^2$, so $w = uv$ where $u, v \in L = L(M)$. Let $p_0 p_1 \cdots p_{|u|}$ and $q_0 q_1 \cdots q_{|v|}$ be the accepting computations in M on u and v, respectively. Since they are accepting, $p_0, q_0 \in I$ and $p_{|u|}, p_{|v|} \in F$. It follows that $(p_0,1)(p_1,1)\cdots(p_{|u|-1},1)(q_0,2)(q_1,2)\cdots(q_{|v|},2)$ is an accepting computation on $w = uv$ in N. The only non-trivial part is verifying that the transition to $(q_0,2)$ is valid. If $u = \epsilon$, then $(p_0,1)(p_1,1)\cdots(p_{|u|-1},1) = \epsilon$ and thus $(q_0,2)$ is the first state of the computation, but we already have that $q_0 \in I$ and that $I \cap F \neq \emptyset$ (because $p_0 \in I$ and $p_0 \in F$), so $(q_0,2) \in I_N$ by its definition. If $u \neq \epsilon$, then, since $q_0 \in I$ and $p_{|u|} \in F$, there is a transition $((p_{|u|-1},1), u_{|u|}, (q_0,2))$ in δ_N by its definition.

Now we consider two ways for constructing a UFA M^2, such that $L(M^2) = L^2$. One construction is 2^N and another is $(2^{N^R})^R$. Since $L(2^N) = L((2^{N^R})^R) = L(N) = L^2$, both are valid. Therefore, $\operatorname{usc}(L^2) \leq \min\{|2^N|, |2^{N^R})^R|\} = \min\{|\mathcal{R}_N|, |\mathcal{C}_N|\}$. However, clearly the reachability of states of the form $(p,1)$ is not affected by the addition of states of the form $(q,2)$. Therefore, $\mathcal{R}_N \subseteq \{\, (S \times \{1\}) \cup (B \times \{2\}) \mid S \in \mathcal{R}_M \wedge B \subseteq Q \,\}$. Similarly for co-reachability, but with the two types of states swapped, so $\mathcal{C}_N \subseteq \{\, (T \times \{2\}) \cup (A \times \{1\}) \mid T \in \mathcal{C}_M \wedge A \subseteq Q \,\}$. Thus, $|\mathcal{R}_N| \leq |\mathcal{R}_M| 2^n$ and $|\mathcal{C}_N| \leq |\mathcal{C}_M| 2^n$. Finally, $\operatorname{usc}(L^2) \leq \min\{|\mathcal{R}_M|, |\mathcal{C}_M|\} 2^n \leq (n+1) e^{n/e} 2^n \leq 2^{1.5308n + \log(n+1)}$, using the result from the proof of Theorem 3.1.14. $\square$

3.3 Lower bound for the complement construction

In this section, we present a lower bound for the worst-case behaviour of the considered complement construction (which is by [Jirásek et al., 2018]). This bound is importantly in $2^{0.5n+\omega(1)}$. Note that this is not a lower bound on the unambiguous state complexity of the complement operation, only on this specific construction for it (though no one has proposed other constructions so far). However, the bound can be straightforwardly translated into a lower bound for the graph problem examined in Lemma 3.1.13 by just taking the graph corresponding to the given UFA (following Lemma 3.1.9).

Proposition 3.3.1. *For any $n \geq 1$, there exists a UFA M with states Q where $|Q| = n$ such that* $\min\{|\mathcal{R}_M|, |\mathcal{C}_M|\} \geq 2^{0.5n+0.5\log(n+1)-1}$.

Proof. We take advantage of Lemma 3.1.4 and Proposition 3.1.15 and only give a compatible pair $(\mathcal{R}, \mathcal{C})$ for which the above holds, since the existence of a UFA M such that $\mathcal{R} \subseteq \mathcal{R}_M$ and $\mathcal{C} \subseteq \mathcal{C}_M$ is implied.

Let $Q = Q_1 \cup Q_2$ where $|Q_1| = k$ and $|Q_2| = n - k$ for some k such that $0 \leq k \leq n$. Then let $\mathcal{R} = 2^{Q_1}$ and $\mathcal{C} = 2^{Q_2} \cup \{\, T \cup \{q\} \mid T \subseteq Q_2 \wedge q \in Q_1 \,\}$. Clearly, for any $S \in \mathcal{R}$ and any $T \in \mathcal{C}$, $|S \cap T| \leq 1$, so $(\mathcal{R}, \mathcal{C})$ is a compatible pair. Furthermore, $|\mathcal{R}| = 2^k$ and $|\mathcal{C}| = (k+1)2^{n-k}$.

Now we set $k = \left[\frac{n}{2} + \frac{1}{2}\log\frac{n+1}{2}\right]$ where $[x]$ is the nearest integer to x.

$$\begin{aligned}
|\mathcal{R}| &= 2^{\left[\frac{n}{2}+\frac{1}{2}\log\frac{n+1}{2}\right]} \\
&\geq 2^{\frac{n}{2}+\frac{1}{2}\log\frac{n+1}{2}-\frac{1}{2}} \\
&= 2^{0.5n+0.5\log(n+1)-1}
\end{aligned}$$

$$\begin{aligned}
|\mathcal{C}| &= \left[\frac{n}{2}+\frac{1}{2}\log\frac{n+1}{2}+1\right]2^{n-\left[\frac{n}{2}+\frac{1}{2}\log\frac{n+1}{2}\right]} \\
&\geq \left(\frac{n}{2}+\frac{1}{2}\log\frac{n+1}{2}+\frac{1}{2}\right)2^{n-\frac{n}{2}-\frac{1}{2}\log\frac{n+1}{2}-\frac{1}{2}} \\
&\geq \frac{\frac{n+1}{2}2^{\frac{n}{2}-\frac{1}{2}}}{\sqrt{\frac{n+1}{2}}}
\end{aligned}$$

$$= \sqrt{\frac{n+1}{2}} 2^{\frac{n}{2}-\frac{1}{2}}$$
$$= 2^{0.5n+0.5\log(n+1)-1}$$

□

3.4 Other explored approaches

In this section, we shall briefly describe what other approaches for bounding the complement operation we tried, what results (if any) they led to and why they were ultimately not used. Our final proof consists of three main steps: the construction from Lemma 3.1.1, the reduction to a problem about the number of cliques and cocliques of a graph in Lemma 3.1.10 and the bound for graphs in Lemma 3.1.13.

The general approach used for the last step is to examine some constrained property of both the graph and cograph, take into account the effect that constraint has on the cograph and graph, respectively, and finally get a bound on the number of cliques for each of the two, independently from each other. Clearly, the tightness of the obtained upper bound depends on the property used. To make this concrete, our final approach is to examine the maximal clique and the maximal coclique of the graph, which effectively "removes" nodes from the cograph and the graph, respectively (more accurately, it "removes" all edges between any pair of those nodes). Finally, we get a bound for the number of cliques in each of the two graphs, independently from each other, and then analyze the minimum of those. An earlier idea was to instead use the number of edges in the graph and in the cograph. Since the two sum up to $\binom{n}{2} \leq \frac{n^2}{2}$, the relationship between them is easy to analyze. Then, using a result by [Wood, 2007] and a similar style of analysis as in our proof of Lemma 3.1.13, we can obtain an upper bound of $2^{n/\sqrt{2}} + n \leq 2^{0.7071n} + n$. This is still an improvement on [Jirásek et al., 2018]'s bound, but is worse than our final result. This makes sense, as examining the maximal clique and coclique preserves much more of the structure between the graph and the cograph, compared to using just the number of edges in each. The option of using both properties at once was also considered. However, we found that this could not improve the constant factor in the linear term of the exponent, only the log term. This is because the worst-case for the graph, i.e. the case that maximizes the minimum of the number of cliques and the number of cocliques, has a maximal clique and a maximal coclique size of $\frac{n}{e}$. Therefore, the part of the construction, which produces the $e^{n/e}$ factor in the number of cliques, has no more than $n^2\left(\frac{e-1}{e}\right)^2 \leq 0.3995n^2$ edges, but this is less than $\binom{n}{2}/2$, so there are enough edges for both the graph and the cograph. Thus, taking the number of edges into account does not break this worst-case construction and therefore cannot improve the linear term in the exponent in the final upper bound.

We also considered another unrelated approach for obtaining an upper bound on $\min\{|\mathcal{R}|, |\mathcal{C}|\}$ before discovering the reduction to a problem about graphs. It was to instead find a bound for $|\mathcal{R}| \cdot |\mathcal{C}|$. Then the square root of that would be a bound for $\min\{|\mathcal{R}|, |\mathcal{C}|\}$. Note that this is not a good idea in general, e.g. trying the same with $|\mathcal{R}| + |\mathcal{C}|$ would clearly be pointless. However, as we will discuss in chapter 4, our computer-aided search suggests that this approach can lead to a bound not much larger than one based on directly considering $\min\{|\mathcal{R}|, |\mathcal{C}|\}$. This led us to Conjectures 3.4.1 and 3.4.2, which we state below. Unfortunately, no good way to work with this product was found, during this project, so this approach did not lead to any proven results here.

Conjecture 3.4.1. *For a graph $G = (V, E)$ with n nodes, the following bound on its cliques and cocliques holds: $|Cliques(G)| \cdot |Cocliques(G)| \leq (n+1)2^n$.*

This implies the following conjecture (using that $\min\{|\mathcal{R}|, |\mathcal{C}|\} \leq |\mathcal{R}| \cdot |\mathcal{C}|$):

Conjecture 3.4.2. *If L is a regular language with $\operatorname{usc}(L) = n$, then $\operatorname{usc}(\overline{L}) \leq 2^{0.5n+0.5\log(n+1)}$.*

Note that this is asymptotically equal to our lower bound for the worst-case behaviour (in the number of states produced) of the examined complement construction. Therefore, if true, this upper bound would be asymptotically equal to the optimal upper bound achievable with this approach.

Chapter 4

Computer-aided exploration of the problem

In this chapter, we outline a search routine that we developed while exploring the problem before trying to obtain any sort of upper bound. We also describe what optimizations were made and what results the search produced. A full code listing can be found in appendix C.

4.1 Description

The search iterates through all compatible pairs $(\mathcal{R}, \mathcal{C})$ for a given n and outputs the maximum $\min\{|\mathcal{R}|, |\mathcal{C}|\}$ found. Of course, this cannot be expected to terminate in a reasonable amount of time, so the program also outputs any improvements to the maximum so far, any time such an improvement is made. Thus, the faster the search routine is, the closer to the actual maximum its result (in some fixed amount of time) is. Now we describe the general approach of the search, as well as any optimizations made.

First of all, it is important to note that we represent the elements of $\mathcal{R}$ and $\mathcal{C}$ (i.e. sets of states) as 32-bit integers. Each state (enumerated from 0 to $n-1$) is associated with some bit of the integer. If that bit is set, then the state is in the set, and otherwise, it is not in the set. This allows for very efficient implementations of set operations as single CPU instructions. This is most critical for intersection, since checking whether $|S \cap T| \leq 1$ holds for some pair of sets S and T is a very frequent operation, so we compute $\cap$ as a bitwise AND, like so: `inter = S & T`. Then we need to check whether the size of the intersection is less than 2. The naïve approach would be to count the number of set bits in `inter`, but that takes N iterations and is thus too slow. We can take advantage of the way binary arithmetic works and note that subtracting 1 from a number flips all its least significant bits until the first set bit (including the set bit), but leaves all bits after that as they were. Therefore, `inter & (inter - 1)` is equal to zero, if and only if `inter` has at most one set bit, which is precisely what we want.

After covering how we operate with these sets, we now outline the structure of the search routine. First, we note that we can always include the empty set and all singleton sets in both $\mathcal{R}$ and $\mathcal{C}$, so we call such sets "boring" and we call sets of size greater than 1 "interesting". Thus, we start by generating a list of all interesting sets (there are $2^n - n - 1$ such sets). Then we need to place each interesting set in $\mathcal{R}$, in $\mathcal{C}$ or in neither. However, that produces an exponential term with base 3, which would be quite slow. Instead, for each interesting set, we either include it in $\mathcal{R}$ or not; we do not keep track of $\mathcal{C}$ at all. Of course, this is done recursively, like so:

```
R.push_back(curr_set);
rec_solve(pos + 1);
R.pop_back();
rec_solve(pos + 1);
```

Then, at the leaves of the recursion, we can compute the maximal $\mathcal{C}$ that is compatible with the current $\mathcal{R}$ by iterating through all interesting sets not in $\mathcal{R}$ and, for each, checking it against every interesting set in $\mathcal{R}$ using bitwise operations, as described above. Note that we exclude the interesting sets in $\mathcal{R}$ using a simple two pointers approach, like so:

```
int ptr = 0;
for (int T : interesting_sets)
{
    while (ptr < R.size() && R[ptr] < T) ++ptr;
    if (ptr < R.size() && R[ptr] == T) continue;
    if (test_set(R, T)) C.push_back(T);
}
```

The search described so far would work, but is still quite inefficient, so we add several more optimizations. First, we actually compute the maximal compatible $\mathcal{C}$ at each step of the recursion, not only in the leaves. This allows us to obtain upper bounds on $|\mathcal{R}|$ and $|\mathcal{C}|$ for any $\mathcal{R}$ and $\mathcal{C}$ that could be produced by this branch of the recursion, like so:

```
int max_R_size = R.size() + interesting_sets.size() - pos + n + 1;
int max_C_size = C.size() + n + 1;
```

Then, if the maximum possible answer this branch can lead to is not greater than the maximum found so far, we can instantly prune it. This gives us an incentive to quickly produce good estimates for the actual maximum, so we can prune branches earlier and more often. One idea is to simply update the answer in all steps of the recursion and not just the leaves. However, we can improve that even further. Instead of using the current $\mathcal{R}$ with its maximal $\mathcal{C}$, we can use the maximal $\mathcal{R}'$ with respect to the maximal $\mathcal{C}$. We find this in the same way as the maximal $\mathcal{C}$, which we already described, but we skip elements already in $\mathcal{R}$, in addition to elements in $\mathcal{C}$. This allows us to approach the actual maximum more quickly and thus take more advantage of the pruning.

Finally, we take care to consider only maximal compatible pairs while searching. More accurately, we want to consider pairs that might lead to maximal pairs in the leaves of the recursion. Note that this does not invalidate the optimization above, since it is performed before reaching the leaves and thus before the full $\mathcal{R}$ is constructed. We do this while computing the maximal $\mathcal{R}'$: for each interesting set not in $\mathcal{R}$ or $\mathcal{C}$, we check whether it could be added to $\mathcal{R}'$; if yes and if it is after the current interesting set, we simply include it in $\mathcal{R}'$, but if it is before the current interesting set, i.e. it has already been considered, then this $\mathcal{R}$ cannot lead to a maximal pair, as it does not include a set that is guaranteed to always be okay to include without breaking compatibility. Therefore, if such an interesting set is found, we can prune the current branch. An additional, small optimization to that is to not look into the child branch that does not include the current set, if it is included in $\mathcal{R}'$ (such a branch would be eventually pruned anyway):

```
bool fits = test_set(C, curr_set);
R.push_back(curr_set);
rec_solve(pos + 1);
R.pop_back();
if (!fits) rec_solve(pos + 1);
```

This concludes the description of our main search routine. Note that the search remains the same regardless of whether we are looking for the maximal possible $\min\{|\mathcal{R}|, |\mathcal{C}|\}$ or $|\mathcal{R}| \cdot |\mathcal{C}|$, so we can simply toggle between the two using a Boolean constant and few `constexpr if`-s without needing to rewrite any of the code.

4.2 Results

After running the search for values of n up to $n = 20$, the results (in fig. 4.1) suggested that a significantly better upper bound than [Jirásek et al., 2018]'s one is possible using the same construction, which motivated our work. In fact, examining the constructions found by a version of the search routine (which only considered maximal compatible pairs) and analyzing its behaviour and run time inspired the idea that maximal compatible pairs are fully determined by the reachability or co-reachability of the sets of size 2. This then leads to thinking of the reachable and co-reachable sets as the cliques and cocliques, respectively, of a graph whose edges are precisely the reachable sets of size 2. Without this novel idea, none of our subsequent results would have been possible.

Furthermore, the results for $\sqrt{|\mathcal{R}| \cdot |\mathcal{C}|}$ indicate that the upper bound of $\sqrt{n+1} \cdot 2^{n/2}$ likely holds for that quantity, which leads to Conjectures 3.4.1 and 3.4.2. Examining the output of the program reveals that this conjectured upper bound is met when $|\mathcal{R}| = n + 1$ and $|\mathcal{C}| = 2^n$ or vice versa, i.e. when $\mathcal{R}$ contains only the boring sets (ones of size less than 2) and $\mathcal{C} = 2^Q$ (or vice versa).

n	$\min\{\lvert\mathcal{R}\rvert, \lvert\mathcal{C}\rvert\}$	$\sqrt{\lvert\mathcal{R}\rvert \cdot \lvert\mathcal{C}\rvert}$	$\sqrt{n+1} \cdot 2^{n/2}$	$(n+1)e^{n/e}$
1	**2**	**2.00**	2.00	2.89
2	**3**	**3.46**	3.46	6.26
3	**5**	**5.66**	5.66	12.06
4	**8**	**8.94**	8.94	21.78
5	**12**	**13.86**	13.86	37.76
6	**18**	**21.17**	21.17	63.64
7	**26**	**32.00**	32.00	105.07
8	38	48.00	48.00	170.76
9	56	71.55	71.55	274.10
10	80	106.13	106.13	435.58
11	128	156.77	156.77	686.48
12	165	230.76	230.76	1074.38
13	263	338.66	338.66	1671.52
14	352	495.74	495.74	2587.28
15	526	724.08	724.08	3986.94
16	768	1055.52	1055.52	6119.80
17	1062	1536.00	1536.00	9361.14
18	1544	2231.76	2231.76	14275.06
19	2176	3238.17	3238.17	21708.12
20	3328	4692.56	4692.56	32929.07

Figure 4.1: Results from the described search routine. Bold entries indicate that the search terminated, non-bold ones indicate that these are just partial results. The third and fourth columns contain our conjectured and proven upper bounds, respectively. Note that the second column exactly matches the third one, which is strong indication that the conjecture is true.

Chapter 5

Conclusions

We have lowered the upper bound for the number of states needed for complementing a UFA, or more formally, for the unambiguous state complexity of the complement operation. This is our most interesting and significant result. Our proof consists of a reduction to a problem in the domain of extremal graph theory and then obtaining an upper bound for that. Both of these steps are likely to be of independent interest. We then used this better bound to also improve the bounds for the unambiguous state complexities of the union and square operations. Furthermore, we have established a non-trivial lower bound for the worst-case behaviour (in the number of states produced) of the examined complement construction (which is also a lower bound for the corresponding graph problem). We have also run an optimized brute-force search, which suggests that it is possible to further improve our upper bound. We have stated this as a conjecture. However, even if that conjecture is true,[5] the upper bound would still be much higher than the best known lower bound for the unambiguous state complexity of the complement operation, but the same is not the case for our proven lower bound for the extremal graph theory problem. Finally, a paper [Author and Kiefer, 2021][6] expanding on our results has been uploaded to `arXiv.org` and submitted to the journal "Information Processing Letters".

[5]As stated in chapter 1, following an insight by my supervisor, a proof for the conjecture was found after completion of the project. It is included in our joint paper [Author and Kiefer, 2021].[6]

[6]Project author name replaced with "Author" to preserve anonymity.

Bibliography

[Author and Kiefer, 2021] Author and Kiefer, S. (2021). On complementing unambiguous automata and graphs with many cliques and cocliques. arXiv: 2105.07470.

[Baier et al., 2016] Baier, C., Kiefer, S., Klein, J., Klüppelholz, S., Müller, D., and Worrell, J. (2016). Markov chains and unambiguous büchi automata. In Chaudhuri, S. and Farzan, A., editors, *Computer Aided Verification*, pages 23–42, Cham. Springer International Publishing.

[Colcombet, 2015] Colcombet, T. (2015). Unambiguity in automata theory. In Shallit, J. O. and Okhotin, A., editors, *Descriptional Complexity of Formal Systems - 17th International Workshop, DCFS 2015*, volume 9118 of *Lecture Notes in Computer Science*, pages 3–18. Springer.

[Fisher and Ryan, 1992] Fisher, D. C. and Ryan, J. (1992). Bounds on the number of complete subgraphs. *Discrete Mathematics*, 103(3):313–320.

[Jirásek et al., 2018] Jirásek, J., Jiraskova, G., and Šebej, J. (2018). Operations on unambiguous finite automata. *International Journal of Foundations of Computer Science*, 29:861–876.

[Kiefer and Widdershoven, 2019] Kiefer, S. and Widdershoven, C. (2019). Efficient analysis of unambiguous automata using matrix semigroup techniques. arXiv: 2105.1906.10093.

[Leung, 2004] Leung, H. (2004). Descriptional complexity of nfas of different ambiguity. *International Journal of Foundations of Computer Science*, 16:98–107.

[Raskin, 2018] Raskin, M. A. (2018). A superpolynomial lower bound for the size of non-deterministic complement of an unambiguous automaton. In Chatzigiannakis, I., Kaklamanis, C., Marx, D., and Sannella, D., editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 138:1–138:11.

[Turán, 1941] Turán, P. (1941). On an external problem in graph theory. *Mat. Fiz. Lapok*, 48:436–452.

[Wood, 2007] Wood, D. (2007). On the maximum number of cliques in a graph. *Graphs and Combinatorics*, 23:337–352.

[Zykov, 1949] Zykov, A. A. (1949). On some properties of linear complexes. *Matematicheskii Sbornik (Novaya Seriya)*, 24 (66):163–188.

Appendices

Appendix A

Supplementary proofs about finite automata

This appendix provides the proofs of lemmas stated in section 2.1, but it does not restate the definitions there, so it only serves as a supplement for readers unfamiliar with these concepts.

Lemma A.0.1. *The reverse of the reverse of the transition relation $\delta \subseteq Q \times \Sigma \times Q$ of an NFA M is equal to δ. Formally, $(\delta^R)^R = \delta$.*

Proof.

$$\begin{aligned}
& (\delta^R)^R \\
&= \{\, (p,a,q) \mid (q,a,p) \in \delta^R \,\} && \text{Definition 2.1.6} \\
&= \{\, (p,a,q) \mid (q,a,p) \in \{\, (q,a,p) \mid (p,a,q) \in \delta \,\} \,\} && \text{Definition 2.1.6} \\
&= \{\, (p,a,q) \mid (p,a,q) \in \delta \,\} \\
&= \delta
\end{aligned}$$

□

Lemma A.0.2. *The reverse of the reverse of an NFA M is equal to M. Formally, $(M^R)^R = M$.*

Proof.

$$
\begin{aligned}
& (M^R)^R \\
&= ((Q, \Sigma, \delta, I, F)^R)^R \\
&= (Q, \Sigma, \delta^R, F, I)^R && \text{Definition 2.1.6} \\
&= (Q, \Sigma, (\delta^R)^R, I, F) && \text{Definition 2.1.6} \\
&= (Q, \Sigma, \delta, I, F) && \text{Lemma A.0.1} \\
&= M
\end{aligned}
$$

□

Lemma A.0.3. *If an NFA M has a computation $\bar{q} \in Q^*$ on $w \in \Sigma^*$, then $|q| = |w| + 1$.*

Proof. We prove the lemma by induction on w:

Case $w = \epsilon$:

$\Rightarrow \bar{q} = q$ where $q \in Q$ — Definition 2.1.3

$\Rightarrow |\bar{q}| = 1 = |w| + 1$ — Definition B.1.4

Case $w = av$ where $a \in \Sigma$ and $v \in \Sigma^*$:

$\Rightarrow \bar{q} = pq\bar{p}$ where $(p, a, q) \in \delta$ and M has a computation $q\bar{p}$ on v — Definition 2.1.3

$\Rightarrow q\bar{p} = \bar{v} + 1$ — Inductive hypothesis

$\Rightarrow pq\bar{p} = \bar{v} + 2 = \bar{w} + 1$ — Definition B.1.4

□

Lemma A.0.4. *If an NFA M has a computation $\bar{q}_u p \in Q^*$ on $u \in \Sigma^*$ and a computation $p\bar{q}_v p \in Q^*$ on $v \in \Sigma^*$ where $p \in Q$, then M has a computation $\bar{q}_u p \bar{q}_v$ on uv.*

Proof. We prove the lemma by induction on u:

Case $u = \epsilon$:

$\Rightarrow \bar{q}_u p = p$ — Definition 2.1.3

$\Rightarrow uv = v$ and $\bar{q}_u p \bar{q}_v = p\bar{q}_v$ — Definition B.1.2

Therefore M has a computation $\bar{q}_u p \bar{q}_v$ on uv

Case $u = a$ where $a \in \Sigma$:

$\Rightarrow \bar{q}_u p = qp$ where $(q', a, p) \in \delta$ — Definition 2.1.3

$\Rightarrow uv = av$ and $\bar{q}_u p \bar{q}_v = qp\bar{q}_v$ — Definition B.1.2

Therefore M has a computation $\bar{q}_u p \bar{q}_v$ on uv — Definition 2.1.3

Case $u = abw$ where $a, b \in \Sigma$ and $w \in \Sigma^*$:

$\Rightarrow \bar{q}_u p = p' q \bar{p} p$ where $(p', a, q) \in \delta$ and M has a computation $q\bar{p}p$ on bw — Definition 2.1.3

$\Rightarrow M$ has a computation $q\bar{p}p\bar{q}_v$ on bwv — Inductive hypothesis

and $uv = abwv$ and $\bar{q}_u p \bar{q}_v = p' q \bar{p} p \bar{q}_v$ — Definition B.1.2

Therefore M has a computation $\bar{q}_u p \bar{q}_v$ on uv — Definition 2.1.3

□

A consequence of the above lemma is the following corollary (since pq is a computation on a):

Corollary A.0.5. *If an NFA M has a computation $\bar{q}p \in Q^*$ on $w \in \Sigma^*$ and $(p, a, q) \in \delta$, then M also has a computation $barqpq$ on wa.*

Lemma A.0.6. *If an NFA M has a computation $\bar{q} \in Q^*$ on $w \in \Sigma^*$, then M^R has a computation $\bar{q}^R$ on w^R.*

Proof. We prove the lemma by induction w:

Case $w = \epsilon$:

$\Rightarrow \bar{q} = q$ where $q \in Q$	Definition 2.1.3
$\Rightarrow w^R = \epsilon$ and $\bar{q}^R = q$	Definition B.1.6
$\Rightarrow M^R$ has a computation $\bar{q}^R$ on w^R	Definition 2.1.3

Case $w = av$ where $a \in \Sigma$ and $v \in \Sigma^*$:

$\Rightarrow \bar{q} = pq\bar{p}$ where $(p, a, q) \in \delta$ and M has a computation $q\bar{p}$ on v	Definition 2.1.3
$\Rightarrow (q, a, p) \in \delta^R$	Definition 2.1.6
and M^R has a computation $(\bar{p}^R)q$ on v^R	Inductive hypothesis
$\Rightarrow M^R$ has a computation $(\bar{p}^R)qp$ on $(v^R)a$	Corollary A.0.5
$\Rightarrow M^R$ has a computation $\bar{q}^R$ on w^R	Definition B.1.6

□

A consequence of the above lemma and Lemma B.1.11 is the following corollary:

Corollary A.0.7. *If $p \xrightarrow{w}_M q$, then $q \xrightarrow{w^R}_{M^R} p$ for an NFA M, $p, q \in Q$ and $w \in \Sigma^*$.*

And a further consequence of this is the next corollary:

Corollary A.0.8. *If an NFA M has an accepting computation $\bar{q} \in Q^*$ on $w \in \Sigma^*$, then M^R has an accepting computation $\bar{q}^R$ on w^R.*

Lemma A.0.9. *If an NFA M has a computation $\bar{q}_u p \bar{q}_v \in Q^*$ on $uv \in \Sigma^*$ where $|u| = |\bar{q}_u|$ and $p \in Q$, then M has a computation $\bar{q}_u p$ on u and a computation $\bar{q}_u p$ on v.*

Proof. First, we prove the second part of the lemma by induction on u:

Case $u = \epsilon$:

$\Rightarrow \bar{q}_u = \epsilon$	Definition B.1.4 since $\|u\| = \|\bar{q}_u\|$
$\Rightarrow uv = v$ and $\bar{q}_u p \bar{q}_v = p\bar{q}_v$	Definition B.1.2
$\Rightarrow M$ has a computation $p\bar{q}_v$ on v	Since $\bar{q}_u p \bar{q}_v$ is a computation on uv

Case $u = a$ where $a \in \Sigma$:

$\Rightarrow \bar{q}_u = q$ where $q \in Q$	Definition B.1.4 since $\|u\| = \|\bar{q}_u\|$
$\Rightarrow uv = av$ and $\bar{q}_u p \bar{q}_v = qp\bar{q}_v$	Definition B.1.2
$\Rightarrow (q, a, p) \in \delta$ and $p\bar{q}_v$ is a computation on v	Definition 2.1.3
	since $qp\bar{q}_v$ is a computation on av

Case $u = abw$ where $a, b \in \Sigma$ and $w \in \Sigma^*$:

$\Rightarrow \bar{q}_u = p'q\bar{p}$ where $p'q \in Q \in Q$	
and $\|bw\| = \|q\bar{p}\|$	Definition B.1.4 since $\|u\| = \|\bar{q}_u\|$
$\Rightarrow uv = abwv$ and $\bar{q}_u p \bar{q}_v = p'q\bar{p}p\bar{q}_v$	Definition B.1.2
$\Rightarrow (p', a, q) \in \delta$ and $q\bar{p}p\bar{q}_v$ is a computation on bwv	Definition 2.1.3
	since $p'q\bar{p}p\bar{q}_v$ is a computation on $abwv$
$\Rightarrow p\bar{q}_v$ is a computation on v	Inductive hypothesis

Then the second part is analogous by using the same proof on the reverse computation, as per Lemma A.0.6, since $|v| = |\bar{q}_v|$ due to Lemma A.0.3 and Lemma B.1.5. $\square$

Lemma 2.1.7. *The language of the reverse of an NFA M is equal to the reverse of the language of M. Formally, $L(M^R) = (L(M))^R$.*

Proof. We prove the equality by double inclusion:

First, we show that $L(M^R) \subseteq (L(M))^R$:

Suppose that $w \in L(M^R)$

$\Rightarrow M^R$ has an accepting computation on w — Definition 2.1.4

$\Rightarrow (M^R)^R$ has an accepting computation on w^R — Corollary A.0.8

$\Rightarrow M$ has an accepting computation on w^R — Lemma A.0.2

$\Rightarrow w^R \in L(M)$ — Definition 2.1.4

$\Rightarrow (w^R)^R \in (L(M))^R$ — Definition B.2.2

$\Rightarrow w \in (L(M))^R$ — Lemma B.1.8

Therefore $L(M^R) \subseteq (L(M))^R$

Then we show that $(L(M))^R \subseteq L(M^R)$:

$L((M^R)^R) \subseteq (L(M^R))^R$ — From the first part

$\Rightarrow L(M) \subseteq (L(M^R))^R$ — Lemma A.0.2

$\Rightarrow (L(M))^R \subseteq ((L(M^R))^R)^R$ — Lemma B.2.4

$\Rightarrow (L(M))^R \subseteq (L(M^R)$ — Lemma B.2.3

□

Lemma 2.1.9. *The set of reachable sets of the reverse of an NFA M is equal to the set of co-reachable sets of M and vice versa. Formally, $\mathcal{R}_{M^R} = \mathcal{C}_M$ and $\mathcal{C}_{M^R} = \mathcal{R}_M$.*

Proof. First, we prove the first part.

$\mathcal{R}_M$

$= \{ \{ q \in Q \mid \exists p \in I \cdot p \xrightarrow{w}_M q \} \mid w \in \Sigma^* \}$ — Definition 2.1.8

$$= \{ \{ q \in Q \mid \exists p \in I \cdot q \xrightarrow{w}_{M^R} p \} \mid w \in \Sigma^* \} \qquad \text{Corollary A.0.7}$$

$$= \mathcal{C}_{M^R} \qquad \text{Definitions 2.1.6 and 2.1.8}$$

The proof of the second part is analogous, by starting with M^R and applying Lemma A.0.2 at the end. □

Lemma 2.1.12. *An NFA M is a UFA if and only if $|S \cap T| \leq 1$ for every reachable set S and every co-reachable set T.*

Proof. The proof here is similar to the one in [Jirásek et al., 2018], but a bit more formal. Note that the proof is essentially the same in both directions, but is intentionally written separately for clarity.

(If) For a contradiction, suppose that $|S \cap T| \leq 1$ for every $S \in \mathcal{R}_M$ and every $T \in \mathcal{T}_M$ and that there is a string w with two distinct accepting computations $\bar{p}, \bar{q} \in Q^*$. From $\bar{p} \neq \bar{q}$ and Lemma A.0.3, it follows that $\bar{p} = \bar{p}_u p \bar{p}_v$ and $\bar{q} = \bar{q}_u q \bar{q}_v$ where $w = uv$, $|\bar{p}|_u = |\bar{q}|_u = |u| + 1$, $|\bar{p}|_v = |\bar{q}|_v = |v| + 1$, $p, q \in Q$ and $p \neq q$. From Lemma A.0.9, it follows that $\bar{p}_u p$ and $\bar{q}_u q$ are both computations on u and $q\bar{p}_v$ and $q\bar{q}_v$ are both computations on v. Finally, note that $\bar{p}_u p \bar{p}_v$ and $\bar{q}_u q \bar{q}_v$ are both accepting, and thus the first states of $\bar{p}_u p$ and $\bar{q}_u q$ are both initial and that the last states of $p\bar{p}_v$ and $q\bar{q}_v$ are both final. Therefore, $p, q \in S \cap T$ where S is u's reachable set and T is v's reachable set, and thus $|S \cap T| \geq 2$, which is a contradiction.

(Only if) For a contradiction, suppose that M is unambiguous and there exist a pair of sets $S \in \mathcal{R}_M$ and $T \in \mathcal{C}$ such that $|S \cap T| \geq 2$. Then there exists a pair of distinct states $p, q \in S \cap T$. Since S is reachable, let $u \in \Sigma^*$ be the string that makes it reachable, from Definition 2.1.8. Similarly, let $v \in \Sigma^*$ be the string that makes C co-reachable. Now as per Definition 2.1.8, let $\bar{p}_u p, \bar{q}_u q, p\bar{q}_v, q\bar{q}_v \in Q^*$ be computations on u and v that reach/co-reach p and q, respectively. From Lemma A.0.4, it follows that $\bar{p}_u p \bar{p}_v$ and $\bar{q}_u q \bar{q}_v$ are both computations on uv. They are also distinct, since $p \neq q$ and $|\bar{p}| = |\bar{q}| = |u|$ by Lemma A.0.3. Finally, note that the first states of $\bar{p}_u p$ and $\bar{q}_u q$ are both initial and that the last states of $p\bar{p}_v$ and $q\bar{q}_v$ are both final, and thus $\bar{p}_u p \bar{p}_v$ and $\bar{q}_u p \bar{p}_v$ are both accepting. Therefore, M has two distinct accepting computations on w, which is a contradiction. ◻

Lemma A.0.10. *Let us write S^w_M for the set reachable by string $w \in \Sigma^*$ in an NFA M. Then $S^{wa}_M = \{\, q \in Q \mid \exists q' \in S^w_M \cdot (q', a, q) \in \delta) \,\}$.*

Proof.

$$\begin{aligned}
& S^{wa}_M \\
&= \{\, q \in Q \mid \exists p \in I \cdot p \xrightarrow{wa}_M q \,\} && \text{Definition 2.1.8} \\
&= \{\, q \in Q \mid \exists p \in I \cdot \exists q' \in Q \cdot (p \xrightarrow{w}_M q' \wedge q' \xrightarrow{a}_M q \in \delta) \,\} && \text{Lemmas A.0.4 and A.0.9} \\
&= \{\, q \in Q \mid \exists p \in I \cdot \exists q' \in Q \cdot (p \xrightarrow{w}_M q' \wedge (q', a, q) \in \delta) \,\} && \text{Definition 2.1.3} \\
&= \{\, q \in Q \mid \exists q' \in Q \cdot ((\exists p \in I \cdot p \xrightarrow{w}_M q') \wedge (q', a, q) \in \delta) \,\} \\
&= \{\, q \in Q \mid \exists q' \in S^w_M \cdot (q', a, q) \in \delta) \,\} && \text{Definition 2.1.8}
\end{aligned}$$

◻

Lemma 2.1.14. *Every reachable set of a DFA M has size exactly* 1.

Proof. We prove that $|S^w_M| = 1$ by induction on w in reverse, which is valid by Corollary B.1.9:

Case $w = \epsilon$:

$$\begin{aligned}
& |S^w_M| \\
&= |\{\, q \in Q \mid \exists p \in I \cdot p \xrightarrow{\epsilon}_M q \,\}| && \text{Definition 2.1.8} \\
&= |\{\, q \in Q \mid \exists p \in I \cdot p = q \,\}| && \text{Definition 2.1.3} \\
&= |I| \\
&= 1 && \text{Definition 2.1.10}
\end{aligned}$$

Case $w = va$ where $a \in \Sigma$ and $v \in \Sigma^*$:

$|S^w_M| = |\{ q \in Q \mid \exists q' \in S^v_M \cdot (q', a, q) \in \delta) \}|$ Lemma A.0.10

But there is exactly one $q' \in S^v_M$ since $|S^v_M| = 1$ Inductive hypothesis

And thus there is exactly one q such that $(q', a, q) \in \delta$ Definition 2.1.10

Therefore $|S^w_M| = 1$

□

Lemma 2.1.17. *The language of the complement of a DFA M is equal to the complement of the language of M. Formally, $L(\overline{M}) = \overline{L(M)}$.*

Proof. First, following Lemma 2.1.14, let $q^w_M \in Q$ be the unique state reachable by string $w \in \Sigma^*$ in M. Then by Definition 2.1.4, $w \in L(M) \leftrightarrow q^w_M \in F$. Also, note that $q^w_M = q^w_{\overline{M}}$, since M and $\overline{M}$ have the same initial states and transition relations. Now we prove the equality by considering each possible element $w \in \Sigma^*$ of the two sets:

$$\begin{aligned}
& w \in \overline{L(M)} && \\
\Leftrightarrow\ & w \notin L(M) && \text{Definition B.2.5} \\
\Leftrightarrow\ & q^w_M \notin F && \text{Shown above} \\
\Leftrightarrow\ & q^w_M \in Q \setminus F && \text{Since } q^w \in Q \\
\Leftrightarrow\ & q^w_{\overline{M}} \in Q \setminus F && \text{Since } q^w_M = q^w_{\overline{M}} \\
\Leftrightarrow\ & w \in L(\overline{M}) && \text{Shown above} \\
& \text{Therefore } L(\overline{M}) = \overline{L(M)} &&
\end{aligned}$$

□

Lemma 2.1.19. *Given an NFA M, the unique state reachable by $w \in \Sigma^*$ in 2^M is equal to the set reachable by w in M.*

Proof. Notice that the definition of the transition relation of 2^M is equivalent to the characterization of reachable sets presented in Lemma A.0.10. Therefore, following the same proof, we can show $q_{2^M}^w = S_M^w$. □

Thus, since $q_{2^M}^w = S_M^w \in \mathcal{R}_M$ for all strings $w \in \Sigma^*$, we get the following corollary:

Corollary A.0.11. *The transition relation, δ_{2^M}, of the subset construction DFA, 2^M, is valid, i.e. $\delta_{2^M} \subseteq \mathcal{R}_M \times \Sigma \times \mathcal{R}_M$.*

Lemma 2.1.20. *The language of the subset construction DFA of an NFA M is equal to the language of M. Formally, $L(2^M) = L(M)$.*

Proof. We prove the equality by considering each possible element $w \in \Sigma^*$ of the two sets:

$$
\begin{array}{lr}
\quad w \in L(M) & \\
\Leftrightarrow M \text{ has an accepting computation on } w & \text{Definition 2.1.4} \\
\Leftrightarrow \exists p \in I \cdot \exists q \in F \cdot p \xrightarrow{w}_M q & \text{Definition 2.1.3} \\
\Leftrightarrow \exists q \in F \cdot q \in S_M^w & \text{Definition 2.1.8} \\
\Leftrightarrow S_M^w \cap F \neq \emptyset & \\
\Leftrightarrow q_{2^M}^w \cap F \neq \emptyset & \text{Lemma 2.1.19} \\
\Leftrightarrow q_{2^M}^w \in \{\, S \mid S \in \mathcal{R}_M \wedge S \cap F \neq \emptyset \,\} & \\
\Leftrightarrow q_{2^M}^w \in F_{2^M} & \text{Definition 2.1.18} \\
 & (\Leftarrow) \text{ Since } 2^M \text{ is a DFA} \\
\Leftrightarrow \exists q \in F_{2^M} \cdot \{I\} \xrightarrow{w}_{2^M} q & \text{Definition 2.1.3} \\
\Leftrightarrow w \in L(2^M) & \text{Definition 2.1.4} \\
\quad \text{Therefore } L(\overline{M}) = \overline{L(M)} &
\end{array}
$$

□

Appendix B

Formal presentation of strings and languages

This appendix contains formal definitions and proofs of necessary results about strings and languages.

B.1 Finite strings over finite alphabets

Definition B.1.1. The *set of all finite strings over a finite set of symbols* Σ, the *alphabet*, is Σ^*, which is defined to be the least set such that $\epsilon \in \Sigma^*$ and $a \cdot w \in \Sigma^*$ for $a \in \Sigma$ and $w \in \Sigma^*$ where $\cdot : \Sigma \times \Sigma^* \to \Sigma^*$ is the *string construction operator*.

We usually refer to finite strings as just strings, since we do not deal with infinite strings. We also refer to strings as *sequences* when talking about sets other than Σ in order to avoid confusion.

Definition B.1.2. The *string concatenation operator* $\cdot^* : \Sigma^* \times \Sigma^* \to \Sigma^*$ is inductively defined by $\epsilon \cdot^* w = w$ and $(a \cdot v) \cdot^* w = a \cdot (v \cdot^* w)$ for $a \in \Sigma$ and $w, v \in \Sigma^*$.

Lemma B.1.3. *The string ϵ is a unit of string concatenation. Formally, $\epsilon \cdot^* w = w$ and $w \cdot^* \epsilon = w$ for $w \in \Sigma^*$.*

Proof. The first part is immediate from Definition B.1.2.

We prove the second part by induction on w:

Case $w = \epsilon$:

$$\begin{aligned}
& w \cdot^* \epsilon \\
= {} & \epsilon \cdot^* \epsilon \\
= {} & \epsilon && \text{Definition B.1.2} \\
= {} & w
\end{aligned}$$

Case $w = a \cdot v$ where $a \in \Sigma$ and $v \in \Sigma^*$:

$$\begin{aligned}
& w \cdot^* \epsilon \\
= {} & (a \cdot v) \cdot^* \epsilon \\
= {} & a \cdot (v \cdot^* \epsilon) && \text{Definition B.1.2} \\
= {} & a \cdot v && \text{Inductive hypothesis} \\
= {} & w
\end{aligned}$$

□

From now on, we omit writing the string construction operator $\cdot$ as well as the final ϵ of a finite string. E.g. instead of writing $a \cdot (b \cdot \epsilon)$, we write ab. We also omit writing the string concatenation operator $\cdot^*$. E.g. instead of writing $w \cdot^* v$, we write wv.

Definition B.1.4. The *length of string* $w \in \Sigma^*$ is $|w|$, which is inductively defined by $|\epsilon| = 0$ and $|av| = 1 + |v|$ for $a \in \Sigma$ and $v \in \Sigma^*$.

Lemma B.1.5. *The length of the concatenation of two strings $w, u \in \Sigma^*$ is equal to the sum of their lengths. Formally, $|wu| = |w| + |u|$.*

Proof. We prove the lemma by induction on w:

Case $w = \epsilon$:

$$
\begin{aligned}
& |wu| \\
&= |\epsilon u| \\
&= |u| && \text{Definition B.1.2} \\
&= |\epsilon| + |u| && \text{Definition B.1.4} \\
&= |w| + |u|
\end{aligned}
$$

Case $w = a \cdot v$ where $a \in \Sigma$ and $v \in \Sigma^*$:

$$
\begin{aligned}
& |wu| \\
&= |(av)u| && \text{Definition B.1.2} \\
&= 1 + |a(vu)| && \text{Definition B.1.4} \\
&= 1 + |v| + |u| && \text{Inductive hypothesis} \\
&= |av| + |u| && \text{Definition B.1.4} \\
&= |w| + |u|
\end{aligned}
$$

□

Definition B.1.6. The *reverse of a string* $w \in \Sigma^*$, is another string w^R inductively defined by $\epsilon^R = \epsilon$ and $(av)^R = (v^R)a$ for $a \in \Sigma$ and $v \in \Sigma^*$. Thus, informally, $(a_1 a_2 \cdots a_m)^R = a_m a_{m-1} \cdots a_1$ for all $a_i \in \Sigma$.

Lemma B.1.7. $(wa)^R = a(w^R)$ *for* $a \in \Sigma$ *and* $w \in \Sigma^*$.

Proof. We prove the lemma by induction on w:

Case $w = \epsilon$:

$$
\begin{aligned}
& (wa)^R && \\
= {}& (\epsilon a)^R && \\
= {}& (a\epsilon)^R && \text{Lemma B.1.3} \\
= {}& (\epsilon^R)a && \text{Definition B.1.6} \\
= {}& \epsilon a && \text{Definition B.1.6} \\
= {}& a\epsilon && \text{Lemma B.1.3} \\
= {}& a(\epsilon^R) && \text{Definition B.1.6} \\
= {}& a(w^R) &&
\end{aligned}
$$

Case $w = bv$ where $b \in \Sigma$ and $v \in \Sigma^*$:

$$
\begin{aligned}
& (wa)^R && \\
= {}& ((bv)a)^R && \\
= {}& (b(va))^R && \text{Definition B.1.2} \\
= {}& (va)^R b && \text{Definition B.1.6} \\
= {}& a(v^R)b && \text{Inductive hypothesis} \\
= {}& a(bv)^R && \text{Definition B.1.6} \\
= {}& a(w^R) &&
\end{aligned}
$$

□

Lemma B.1.8. *The reverse of the reverse of a string $w \in \Sigma$ is equal to w. Formally, $(w^R)^R = w$.*

Proof. We prove the lemma by case analysis on w:

Case $w = \epsilon$:

$$\begin{aligned}
&(w^R)^R \\
&= (\epsilon^R)^R \\
&= (\epsilon)^R && \text{Definition B.1.6} \\
&= \epsilon && \text{Definition B.1.6} \\
&= w
\end{aligned}$$

Case $w = av$ where $a \in \Sigma$ and $v \in \Sigma^*$:

$$\begin{aligned}
&(w^R)^R \\
&= ((av)^R)^R \\
&= (v^R a)^R && \text{Definition B.1.6} \\
&= a((v^R)^R) && \text{Lemma B.1.7} \\
&= av && \text{Inductive hypothesis} \\
&= w
\end{aligned}$$

□

A consequence of the above lemma is the following corollary:

Corollary B.1.9. *String reversal is a bijection and its own inverse and thus $w = v$ if and only if $w^R = v^R$ for $w, r \in \Sigma^*$.*

Definition B.1.10. The *first symbol* of a string aw is a and the *last symbol* of a string wa is a for $a \in \Sigma$ and $w \in \Sigma^*$ where $w \neq \epsilon$.

Lemma B.1.11. *The first symbol of a non-empty string $w \in \Sigma^*$ is equal to the last symbol of its reverse w^R and the last symbol of w is equal to the first symbol of w^R.*

Proof. First, we prove the first part:

$$
\begin{aligned}
& \text{Suppose } w \text{ has } a \in \Sigma \text{ as its first symbol} & \\
\Rightarrow\ & w = av \text{ where } v \in \Sigma^* & \text{Definition B.1.10} \\
\Rightarrow\ & w^R = (v^R)a & \text{Definition B.1.6} \\
\Rightarrow\ & w^R \text{ has } a \in \Sigma \text{ as its last symbol} & \text{Definition B.1.10}
\end{aligned}
$$

The proof of the second part is analogous, by starting with w^R and applying Lemma B.1.8 at the end. □

B.2 Languages

Definition B.2.1. A *language over a finite alphabet* Σ is a set $L \subseteq \Sigma^*$.

Definition B.2.2. The *reverse of a language* L is another language $L^R = \{\, w^R \mid w \in L \,\}$.

Lemma B.2.3. *The reverse of the reverse of a language L is equal to L. Formally, $(L^R)^R = L$.*

Proof.

$$
\begin{aligned}
& (L^R)^R & \\
={}& \{\, w^R \mid w \in L^R \,\} & \text{Definition B.2.2} \\
={}& \{\, w^R \mid w \in \{\, v^R \mid v \in L \,\}\,\} & \text{Definition B.2.2} \\
={}& \{\, (w^R)^R \mid w^R \in \{\, v^R \mid v \in L \,\}\,\} & \text{Corollary B.1.9} \\
={}& \{\, w \mid w^R \in \{\, v^R \mid v \in L \,\}\,\} & \text{Lemma B.1.8} \\
={}& \{\, w \mid w \in \{\, v \mid v \in L \,\}\,\} & \text{Corollary B.1.9} \\
={}& \{\, w \mid w \in L \,\} & \\
={}& L &
\end{aligned}
$$

□

Lemma B.2.4. *The subset relation is preserved under language reversal. formally, if $L_1 \subseteq L_2$, then $L_1^R \subseteq L_2^R$ for $L_1, L_2 \subseteq \Sigma^*$.*

Proof.

$$\begin{array}{lr}
\quad \text{Let } w \in L_1^R & \\
\Rightarrow w^R \in (L_1^R)^R & \text{Definition B.2.2} \\
\Rightarrow w^R \in L_1 & \text{Lemma B.2.3} \\
\Rightarrow w^R \in L_2 & \text{Since } L_1 \subseteq L_2 \\
\Rightarrow (w^R)^R \in L_2^R & \text{Definition B.2.2} \\
\Rightarrow w \in L_2^R & \text{Lemma B.1.8} \\
\quad \text{Therefore } L_1^R \subseteq L_2^R &
\end{array}$$

□

Definition B.2.5. The *complement of a language* $L \subseteq \Sigma^*$ is another language $\overline{L} = \Sigma^* \setminus L$.

Definition B.2.6. The *square of a language* L is another language $L^2 = \{ uv \mid u, v \in L \}$.

Appendix C

Code listing

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <math.h>
#include <assert.h>

const bool USE_PRODUCT = false;

std::vector<int> generate_interesting_sets(int n)
{
    std::vector<int> interesting_sets;
    int curr = 0;
    int last = 1 << n;
    while (curr < last)
    {
        if (curr & (curr - 1))
            interesting_sets.push_back(curr);
        ++curr;
    }
    return interesting_sets;
}

bool test_set(const std::vector<int>& R, int T)
{
    for (int S : R)
    {
        int inter = S & T;
        if (inter & (inter - 1)) return false;
    }
    return true;
}

std::vector<int> interesting_sets;
std::vector<int> R;
std::vector<int> C;
long long ans;
int n;

void rec_solve(int pos)
{
    C.resize(0);
    int ptr = 0;
    for (int T : interesting_sets)
    {
        while (ptr < R.size() && R[ptr] < T) ++ptr;
        if (ptr < R.size() && R[ptr] == T) continue;
        if (test_set(R, T)) C.push_back(T);
    }
```

```
int max_R_size = R.size() + interesting_sets.size() - pos + n + 1;
int max_C_size = C.size() + n + 1;

long long max_ans;
if constexpr (USE_PRODUCT) max_ans = (long long) max_R_size *
    max_C_size;
else max_ans = std::min(max_R_size, max_C_size);

if (max_ans <= ans) return;

ptr = 0;
int ptr2 = 0;
int extra_R_size = 0;
for (int i = 0; i < interesting_sets.size(); ++i)
{
    int S = interesting_sets[i];
    while (ptr < R.size() && R[ptr] < S) ++ptr;
    if (ptr < R.size() && R[ptr] == S) continue;
    while (ptr2 < C.size() && C[ptr2] < S) ++ptr2;
    if (ptr2 < C.size() && C[ptr2] == S) continue;
    if (test_set(C, S))
    {
        if (i < pos) return;
        ++extra_R_size;
    }
}

int R_size = R.size() + extra_R_size + n + 1;
int C_size = C.size() + n + 1;
```

```
    long long curr_ans;
    if constexpr (USE_PRODUCT) curr_ans = (long long) R_size * C_size;
    else curr_ans = std::min(R_size, C_size);
    if (curr_ans > ans)
    {
        ans = curr_ans;
        std::cerr << "|R|: " << R_size << " (" << extra_R_size << ")" << "
            |C|: " << C_size << " : ";
        if constexpr (USE_PRODUCT) std::cerr << sqrt(ans);
        else std::cerr << ans;
        std::cerr << std::endl;
    }

    if (pos < interesting_sets.size())
    {
        int curr_set = interesting_sets[pos];
        bool fits = test_set(C, curr_set);
        R.push_back(curr_set);
        rec_solve(pos + 1);
        R.pop_back();
        if (!fits) rec_solve(pos + 1);
    }
}

double usc_compliment(int curr_n)
{
    n = curr_n;
    ans = -1;
    interesting_sets = generate_interesting_interesting_sets(n);
    std::cerr << "interesting_sets size: " << interesting_sets.size() <<
        std::endl;
    R = {};
    rec_solve(0);
    if constexpr (USE_PRODUCT) return sqrt(ans);
    else return ans;
}

int main()
{
    int n;
    std::cin >> n;
    double ans = usc_compliment(n);
    std::cout << n << " : " << ans << std::endl;
    return 0;
}
```