4-21-2010

On the State Hierarchy of Exploding Automata

Matthew Maly
Trinity University, mmaly@trinity.edu

On the State Hierarchy of Exploding Automata

Matthew R. Maly

Abstract

A recently revisited question in finite automata theory considers the possible numbers n and d for which there exists an n-state minimal NFA with a minimal equivalent DFA of d states. We present a new class of finite automata, the NFA E_n of n states, which in a sense contains half of the state hierarchy $[n, 2^n]$; that is, by making small modifications to E_n, we can create a minimal equivalent DFA of d states for any $d \in (2^{n-1}, 2^n]$. Although this is not stronger than the most recent of work that has been done on the problem, the value of this result lies in the systematic and intuitive method by which we, given the parameter d, construct the appropriate NFA from E_n. Specifically, the construction from E_n is a direct reflection of the binary representation of $2^n - d$, each 1-bit of which indicates a single modification to make to E_n. We conclude the thesis with a discussion of computational results to suggest that these methods can be extended to reach the entire state hierarchy, that is, to answer the question for any $d \in [n, 2^n]$.

Acknowledgments

- For Dr. Maurice Eggen, who encouraged me to seek out my own niche of research.
- For Dr. Mark Lewis, who taught me how to think like a research scientist.
- For Dr. Berna Massingill, the helpful technical guru who almost always has the answer you seek, and otherwise will go to any length to find it.
- For Dr. Brian Miceli, who taught me how to think, write, and speak like a mathematician.
- For Dr. Paul Myers, who first introduced me to the beautiful area of theoretical computer science and jump-started this thesis.
- For Maddy, for motivation and perspective.

ON THE STATE HIERARCHY OF EXPLODING AUTOMATA

MATTHEW R. MALY

A DEPARTMENTAL HONORS THESIS SUBMITTED TO THE
DEPARTMENT OF COMPUTER SCIENCE AT TRINITY UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR GRADUATION
WITH DEPARTMENTAL HONORS.

APRIL 21, 2010

THESIS ADVISOR

DEPARTMENT CHAIR

ASSOCIATE VICE PRESIDENT
FOR
ACADEMIC AFFAIRS

On the State Hierarchy of Exploding Automata

Matthew R. Maly

Contents

List of Figures

Chapter 1

Introduction

This thesis is primarily concerned with the behavior of deterministic and nondeterministic finite automata (DFA's and NFA's, respectively). Arguably the simplest theoretical model of computation, a finite automaton is, in essence, a machine that accepts or rejects a string of input symbols, simply by changing states as it sequentially encounters each input symbol.

1.1 Preliminaries

To summarize the theory of finite automata, we draw from definitions and results in Hopcroft's textbook [2].

1.1.1 Alphabets and Strings

An *alphabet*, typically denoted Σ, is a finite, nonempty set of symbols. Common examples of alphabets include the binary alphabet $\{0, 1\}$, the letter alphabet $\{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\}$, and the set of all ASCII characters. A *string* w is a finite sequence of symbols taken from some alphabet. The *length* of w, typically denoted $|w|$, is the number of symbol positions in w.

For example, $w = 10001$ is a string from the alphabet $\Sigma = \{0, 1\}$, and $|w| = 5$. The *empty string*, denoted ϵ, is the unique string of length zero. The concatenation of two strings u and v, denoted $u \circ v$ and often abbreviated as uv, is the string formed by following the symbols of u by the symbols of v. For example, if $u = \texttt{race}$ and $v = \texttt{car}$ are strings from the letter alphabet, then $uv = \texttt{racecar}$. Furthermore, $w \circ \epsilon = \epsilon \circ w = w$ for any string w over any alphabet.

1.1.2 Strings of a Fixed Length

Let Σ be an alphabet. If k is a nonnegative integer, then we define Σ^k to be the set of all strings over Σ of length k. For example, if $\Sigma = \{0, 1\}$, then $\Sigma^0 = \{\epsilon\}$, $\Sigma^1 = \{0, 1\}$, $\Sigma^2 = \{00, 01, 10, 11\}$, and so forth. We define Σ^* to be the set of *all* strings over the alphabet Σ; that is,

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots.$$

1.1.3 Languages

If Σ is an alphabet, then a *language* is any subset of Σ^*. For example, if $\Sigma = \{0, 1\}$, then $\{1, 10, 100, \ldots\}$ is the language consisting of all binary powers of 2. A language can be infinite, finite, or empty.

The central focus of automata theory is the problem of deciding whether a given string is a member of a particular language. Finite automata are abstract machines that can, in some cases, be used to answer such a question. The class of *regular languages* comprises exactly the languages for which finite automata can make this decision.

1.1.4 Finite Automata

Finite automata are divided into two categories as to whether they are deterministic or nondeterministic.

Deterministic Finite Automata

For the DFA's we reproduce the most common definition [13].

Definition 1. *A* deterministic finite automaton (DFA) *is a quintuple* $(Q, \Sigma, \delta, q_0, F)$*, where*

1. Q *is a finite set of states,*
2. Σ *is an alphabet,*
3. $\delta : Q \times \Sigma \to Q$ *is a transition function,*
4. $q_0 \in Q$ *is a start state, and*
5. $F \subseteq Q$ *is a set of accept states.*

For an example, let $\Sigma = \{0, 1\}$ and consider the regular language

$$L = \{w \in \Sigma^* \mid w \text{ contains an even number of 1's}\}.$$

We wish to construct a DFA D that will "accept" (a notion that will be formally defined later) an input string w if and only if $w \in L$. This DFA will consist of two states to keep track of whether the number of 1's is even or odd as it reads each symbol of w. The start state should correspond to an even number of 1's, since the empty string ϵ contains zero 1's. Furthermore, this state should be the only accept state of D. To this end, set

$D = \{\{E, O\}, \Sigma, \delta, E, \{E\}\}$, where δ is the transition function given by

$$\delta(q, \sigma) = \begin{cases} q & \text{if } \sigma = 0, \\ E & \text{if } \sigma = 1 \text{ and } q = O, \\ O & \text{if } \sigma = 1 \text{ and } q = E. \end{cases}$$

Here the states E and O correspond to whether the number of 1's is even or odd, respectively.

Since it is often difficult to discern the behavior of a finite automaton from its formal definition, a graphical representation, called a *transition diagram*, is used. This diagram is a directed graph in which each state of D is represented by a node and each transition is represented by an arc from one node to another. The start state is designated with an arrow pointing to it without a source, and the accept state is designated with a double circle. Figure 1.1 contains the transition diagram for D.

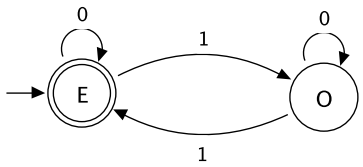

Figure 1.1: A simple DFA: D.

Before reading an input string, D is in state E, which corresponds to an even number of 1's. Reading a 1 at any time will change it to the opposite state, but reading a 0 will preserve its state, since the number of 0's in no way affects the number of 1's in an input string.

Consider running the DFA D on the input string $w = 100$. Beginning with the start

state E, we iterate the transition function δ on each symbol of w. Since $\delta(E, 1) = O$, the state of D after reading the first symbol of w changes to O. The next state of D is determined by this new state and the next symbol of w; that is, its next state is $\delta(O, 0) = O$. Reading the third and final symbol of w, D ends in the state $\delta(O, 0) = O$. Since O is not an accept state, D does not accept w, from which we conclude that $w \notin L$ (rightfully so, as w contains exactly one 1).

Extending the Transition Function of a DFA

To aid with modeling the computation of a DFA, we recursively extend the function δ to process not only symbols but also strings. Let $w \in \Sigma^*$ be a nonempty string; then $w = \sigma u$ for some $\sigma \in \Sigma$ and $u \in \Sigma^*$ such that $|u| = |w| - 1$. We set

$$\delta(q, w) = \delta(\delta(q, \sigma), u),$$

for any state q of a DFA, where $\delta(q, \sigma)$ is defined as in the DFA description in Section 1.1.4.

The Language of a DFA

As mentioned earlier, the task of a finite automaton is to decide whether an input string is a member of some regular language. Let $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Given an input string $w \in \Sigma^*$, it is said that D *accepts* w if $\delta(q_0, w) \in F$. Otherwise, D *rejects* w. The *language* of D, denoted $L(D)$, is exactly the set of strings accepted by D; that is,

$$L(D) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}.$$

Then it is said that D *describes* the language $L(D)$.

Nondeterministic Finite Automata

Nondeterministic finite automata are slightly different from deterministic ones in that they can be in multiple states at once. Furthermore, a state of an NFA can have any number of transitions (including zero) to other states on a given input symbol.

For the NFA's we present a slight adaptation of an uncommon definition given by Lewis and Papadimitriou [8].

Definition 2. *A* nondeterministic finite automaton (NFA) *is a quintuple* $(Q, \Sigma, \Delta, q_0, F)$*, where*

1. Q *is a finite set of states,*
2. Σ *is an alphabet,*
3. Δ *is a finite subset of* $Q \times \Sigma \times Q$*, called a transition relation,*
4. $q_0 \in Q$ *is a start state, and*
5. $F \subseteq Q$ *is a set of accept states.*

The Transition Relation of an NFA

The literature typically defines an NFA with a transition function $\delta : Q \times \Sigma \to \mathcal{P}(Q)$, where $\mathcal{P}(Q)$ denotes the power set of Q [2, 13]. This function is driven by its underlying transition relation, which we use instead. This decision has been made to provide greater intuition for the results of this thesis and in no way weakens the traditional definition. Indeed, given an NFA with transition relation Δ, we may immediately form the corresponding transition function δ by setting

$$\delta(q, \sigma) = \{p \in Q \mid (q, \sigma, p) \in \Delta\}$$

for all ordered pairs $(q, \sigma) \in Q \times \Sigma$. Conversely, given an NFA with transition function δ, the corresponding transition relation is

$$\Delta = \{(q, \sigma, p) \mid q \in Q,\ \sigma \in \Sigma, \text{ and } p \in \delta(q, \sigma)\}.$$

The definitions of certain NFA's in this thesis will exploit this implicit equivalence between Δ and δ.

Transitions on the Empty String

Often NFA's are also presented with additional transitions on the empty string ϵ, often called ϵ-moves. No NFA's in this thesis will use ϵ-moves, and so for any NFA M with state set Q and transition function δ, we have that

$$\delta(q, \epsilon) = \{q\} \tag{1.1}$$

for every $q \in Q$. Notice that this assumption does not cause us to lose any generality, since any NFA with ϵ-moves can be converted into an NFA without ϵ-moves by combining any states sharing ϵ transitions into a single state.

Extending the Transition Function of an NFA

For NFA's, we extend the transition function, as seen similarly with DFA's, to process input strings. Also, since the current "state" of an NFA at a given time is actually a *set* of states (specifically, an element of $\mathcal{P}(Q)$ in the definition of an NFA), we further extend the

transition function δ of an NFA M to process sets of states, given by

$$\delta(A, \sigma) = \bigcup_{q \in A} \delta(q, \sigma) \tag{1.2}$$

for all subsets A of the state set Q_M.

The Language of an NFA

The notion of an NFA's accepting or rejecting an input string is analogous to that of a DFA. An NFA $M = (Q, \Sigma, \Delta, q_0, F)$ accepts an input string $w \in \Sigma^*$ if, after reading w, its resulting set of states contains at least one accept state. Formally, M accepts w if

$$\delta(q_0, w) \cap F \neq \varnothing.$$

Otherwise, M rejects w. Finally, as with DFA's, the language described of M, denoted $L(M)$, contains exactly the strings accepted by M.

An Example of an NFA

Let $\Sigma = \{0, 1\}$ and consider the regular language of all strings $w \in \Sigma^*$ for which the second symbol from the end is a 1. We will construct an NFA that recognizes this language. Define the NFA $M = (\{q_0, q_1, q_2\}, \Sigma, \Delta, q_0, \{q_2\})$, where

$$\Delta = \{(q_0, 0, q_0), (q_0, 1, q_0), (q_0, 1, q_1), (q_1, 0, q_2), (q_1, 1, q_2)\}.$$

Figure 1.2 contains the transition diagram for M.

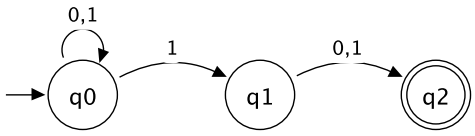

Figure 1.2: A simple NFA: M.

Consider running M on the input string $w = 100$. Beginning with the start state q_0, we have that $\delta(q_0, 1) = \{q_0, q_1\}$. Then, by (1.2),

$$\delta(\{q_0, q_1\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_2\}.$$

Note that at this point, M's current set of states includes the accept state q_2. Reading the final symbol of w changes this, as

$$\delta(\{q_0, q_2\}, 0) = \delta(q_0, 0) \cup \delta(q_2, 0) = \{q_0\}.$$

Since $\{q_0\} \cap \{q_2\} = \varnothing$, we conclude that M does not accept w.

1.1.5 The Subset Algorithm

Intuitively it may seem that NFA's are more powerful than DFA's in that they can describe more types of languages. However, they are one in the same, in that any language that can be described by an NFA can also be described by a DFA, and vice versa. To clarify this result, we first require a simple definition of equivalence for finite automata [2].

Definition 3. *Two finite automata M and N are* equivalent *if $L(M) = L(N)$.*

Notice that for any DFA D, there exists an equivalent NFA; such an NFA M can be

formed by creating a transition relation Δ_M from δ_M, and leaving all other components of D the same. Much more interesting is the converse of this statement, which is a crucial result in automata theory [2].

Theorem 4. *If M is an NFA, then there exists a DFA D equivalent to M.*

Proof. Given an NFA $M = (Q_M, \Sigma, \Delta_M, q_0, F_M)$, let $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ be a DFA such that the following hold.

1. $Q_D = \mathcal{P}(Q_M)$.

2. The transition function δ_D is formed by extending the transition relation Δ_M such that
$$\delta_D(A, \sigma) = \bigcup_{q \in A} \{p \in Q_M \mid (q, \sigma, p) \in \Delta_M\}$$
for each $A \in Q_D$ and $\sigma \in \Sigma$. Notice that if δ_M is the transition function of M created from Δ_M, then $\delta_D(A, \sigma) = \delta_M(A, \sigma)$ for each $A \in Q_D$, by the extension of the transition function of an NFA in (1.2).

3. The set F_D of accept states of D contains exactly all subsets of Q_M that include at least one accept state of M. Formally,
$$F_D = \{A \in Q_D \mid A \cap F_M \neq \varnothing\}.$$

We will show that $L(D) = L(M)$. Let δ_M be the transition function formed from Δ_M, and let $w \in \Sigma^*$. Notice that $w \in L(D)$ if and only if
$$\delta_D(\{q_0\}, w) \in F_D.$$

Analogously, $w \in L(M)$ if and only if

$$\delta_M(q_0, w) \cap F_M \neq \varnothing.$$

So, to complete the proof, it is sufficient to show that

$$\delta_D(\{q_0\}, w) = \delta_M(q_0, w),$$

which we will prove by induction on the length of w. In the base case, (1.1) implies that

$$\delta_D(\{q_0\}, \epsilon) = \{q_0\} = \delta_M(q_0, \epsilon),$$

and so the claim holds. Assume the inductive hypothesis for all strings of fixed length $k \geq 0$, and let $w \in \Sigma^{k+1}$. Then $w = u\sigma$ for some $u \in \Sigma^k$ and $\sigma \in \Sigma$. Let $A = \delta_M(q_0, u)$. Now,

$$\begin{aligned}
\delta_D(\{q_0\}, w) &= \delta_D(\delta_D(\{q_0\}, u), \sigma) \\
&= \delta_D(\delta_M(\{q_0\}, u), \sigma) && \text{(by the inductive hypothesis)} \\
&= \delta_D(A, \sigma) \\
&= \delta_M(A, \sigma) && \text{(by remark in \#2 above)} \\
&= \delta_M(\delta_M(q_0, u), \sigma) \\
&= \delta_M(q_0, w).
\end{aligned}$$

□

This result shows that the added power given to the NFA's from their ability to be in multiple states at once in no way strengthens their fundamental abilities.

The above procedure that was used to generate a DFA from an equivalent NFA is commonly called the *subset construction* or the *subset algorithm*. Although it requires some careful notation, the subset algorithm has an intuitive approach: since the current "state" of the NFA M at any given time is a set of states of Q_M, we create a DFA D with each state corresponding to each possible set of states of Q_M. If M has n states, then D will then have $|\mathcal{P}(Q_M)| = 2^n$ states, but not all of these states A are necessarily *accessible*; that is, there may not exist an input string $w \in \Sigma^*$ for which $\delta(\{q_0\}, w) = A$. It would then be advantageous to remove all inaccessible states from the DFA D, since their absence in no way affects the machine's behavior.

1.1.6 Minimization

When discussing the number of states of an n-state finite automaton N, be it NFA or DFA, to formalize the notion of minimality becomes crucial. Consider the equivalence class of N, following from Definition 3. We elect a finite automaton M equivalent to N with a least number of states to be a representative of this equivalence class. Such a machine is important in that it is a smallest finite automaton (with regard to number of states) to recognize the language $L(N)$.

Definition 5. *Let M be a DFA (NFA). Then M is* minimal *if, for any DFA (NFA) N for which $L(N) = L(M)$, the number of states of N is at least that of M.*

Let D be a DFA. We wish to *minimize* D; that is, find a minimal DFA equivalent to D. This can be done by finding two distinct states of D that can be replaced by a single state without altering the language recognized by D. We first define a notion of equivalence between two states, which will ensure that such a replacement can safely occur.

Definition 6. *Let $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Two states p and q of D are said to be*

equivalent *if, for all* $w \in \Sigma^*$*, we have that* $\delta(p, w) \in F$ *if and only if* $\delta(q, w) \in F$*. Otherwise,* p *and* q *are said to be* distinguishable*.*

Notice that state equivalence is an equivalence relation. As shown in the following lemma, to remove the inaccessible states from D and to replace all pairs of equivalent states are sufficient to minimize D [2].

Lemma 7 (Hopcroft)**.** *Let* $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$ *be a DFA. If all states of* D *are accessible and pairwise distinguishable, then* D *is minimal.*

Proof. Seeking a contradiction, suppose that D is not minimal. Then there exists a minimal DFA $C = (Q_C, \Sigma, \delta_C, q_C, F_C)$ such that $L(C) = L(D)$ and $|Q_C| < |Q_D|$. We intuitively extend the definition of equivalence of states across these two DFA's as follows: a state p of C and a state q of D are called *isomorphic*, denoted $p \approx q$, if for all $r \in \Sigma^*$ we have that $\delta_C(p, r) \in F_C$ if and only if $\delta_D(q, r) \in F_D$. As with state equivalence, notice that state isomorphism is also an equivalence relation.

Let $q \in Q_D$. Since all states of D are accessible by hypothesis, there exists an input string $v \in \Sigma^*$ such that $\delta_D(q_D, v) = q$. Let $p = \delta_C(q_C, v)$. We induct on the length of v to show that $q \approx p$. In the base case, $q = \delta_D(q_D, \varepsilon) = q_D$ by (1.1); similarly, $p = q_C$. Since $L(D) = L(C)$, the strings accepted by D are exactly those accepted by C, and so $\delta_D(q_D, r) \in F_D$ if and only if $\delta_C(q_C, r) \in F_C$ for all $r \in \Sigma^*$. Thus $q \approx p$ in the base case. Now assume the inductive hypothesis for some fixed length $k \geq 0$, and let $w \in \Sigma^{k+1}$. Then $w = u\sigma$ for some $u \in \Sigma^k$ and $\sigma \in \Sigma$. Let $r \in \Sigma^*$, and let $q' = \delta_D(q_D, u)$ and $p' = \delta_C(q_C, u)$. Then $q' \approx p'$ by the inductive hypothesis, and so $\delta_D(q', \sigma r) \in F_D$ if and

only if $\delta_C(p', \sigma r) \in F_C$. Notice that

$$\begin{aligned}\delta_D(q, r) &= \delta_D(\delta_D(q_D, w), r) \\ &= \delta_D(\delta_D(q_D, u), \sigma r) \\ &= \delta_D(q', \sigma r).\end{aligned}$$

Similarly,

$$\begin{aligned}\delta_C(p, r) &= \delta_C(\delta_C(q_C, w), r) \\ &= \delta_C(\delta_C(q_C, u), \sigma r) \\ &= \delta_C(p', \sigma r).\end{aligned}$$

Thus $\delta_D(q, r) \in F_D$ if and only if $\delta_C(p, r) \in F_C$. Since r is an arbitrary input string, $q \approx p$, which completes the induction.

We have just shown that for each $q \in Q_D$, there exists $p \in Q_C$ such that $q \approx p$. Now, since $|Q_D| > |Q_C|$, there must exist two distinct states $q_1, q_2 \in Q_D$ and one state $p_1 \in Q_C$ for which $q_1 \approx p_1 \approx q_2$. Then q_1 and q_2 are equivalent, which contradicts the hypothesis that all states of D are pairwise distinguishable. Hence C cannot exist, and so D is minimal. □

There exist many algorithms for minimizing DFA's; the most common of which is Hopcroft's algorithm, which minimizes an n-state DFA in $O(n \log n)$ steps [3]. Hopcroft's approach involves partitioning the DFA's set of states into a set of equivalence classes, using the notion of state equivalence from Definition 6. The algorithm then returns a minimal equivalent DFA for which each state represents one of these equivalence classes.

1.2 Original Plans

The specific nature of this thesis originally stemmed from a question posed by Dr. Paul Myers: *What characterizes an NFA with n states that, under the subset algorithm, becomes a DFA with 2^n states that cannot be simplified by the minimization algorithm?* This issue has not only been addressed but also extended and generalized in the literature. Following this prompt, the original goals of this thesis were to address the following questions.

(1) How does varying which of the n states of an NFA M are accept states affect whether its resulting minimal DFA has 2^n states?

(2) How can we characterize all of the NFA's that behave according to Dr. Myers' prompt? How do they relate? Can two or more be combined to create another?

It appears that nowhere in the current literature has question (1) been considered. Also, a semester of examination has shed light on how ambitious question (2) really is. Although these questions served as starting points for attempts at an original contribution to the theory, a close examination of what problems are of interest to current researchers has led this work in a slightly different direction.

1.3 Modifications

The literature contains much work on a generalization of Dr. Myers' prompt to ask if, given positive integers n and d with $n \leq d \leq 2^n$, there exists a minimal n-state NFA for which a minimal equivalent DFA has d states. The interval $[n, 2^n] \subseteq \mathbb{N}$ across which d ranges is often called the *state hierarchy* of a minimal n-state NFA. This question has been answered for input alphabets as small as 4 symbols in size, but the NFA's created to answer it vary widely with many distinct cases on the numbers n and d. The question has also

been partially answered for a binary alphabet, but only for certain cases on d instead of the entire state hierarchy.

This thesis details an attempt to answer the question, beginning by creating a special NFA E_n of n states over an n-symbol alphabet that answers the question for $d = 2^n$. To answer the question for another parameter $d' \in [n, 2^n]$ (that is, to construct a minimal n-state NFA for which the minimal equivalent DFA has d' states) will then require small intuitive modifications to E_n, which work for all $d' \in (2^{n-1}, 2^n]$. Thus, in a sense, there exists a special class of machines intimately related to E_n that allows us to reach a little more than half of the state hierarchy. Their construction will intuitively follow from the binary representation of the integer $k = 2^n - d'$.

1.4 Terminology and Symbols

Let $\mathbb{N}$ denote the set of positive integers $\{1, 2, \ldots\}$. For each $n \in \mathbb{N}$, define the alphabet $\Sigma_n = \{1, 2, \ldots, n\} \subseteq \mathbb{N}$. To prevent ambiguity when necessary, we mark the state sets and transition functions (or relations) of a finite automaton M with a subscript M. For example, δ_D is the transition function with values in the states Q_D of some finite automaton D. If M is an NFA, then let M' be the equivalent DFA obtained from the subset algorithm, with all inaccessible states removed.

Let $\gcd(a, b)$ denote the greatest common divisor of any two integers a and b. For any real number x, let $\lceil x \rceil$ (the "ceiling" function) denote the smallest integer m for which $m \geq x$. We use square brackets and parentheses to denote closed and open intervals of positive integers, respectively; for example, $[2, 5) \subseteq \mathbb{N}$ is equivalent to the set $\{2, 3, 4\}$. Finally, as defined in Section 1.1.4, for any set A, let $\mathcal{P}(A)$ (the "power set" of A) denote the set of all subsets of A.

Chapter 2

Related Work

As discussed in Section 1.1.5, applying the subset algorithm to an n-state NFA M will yield a DFA with up to 2^n states. Rabin posed the question of "whether the bound of 2^n on the number of states... may be considerably improved" [12].

2.1 Hitting the Bound

The first to answer Rabin's prompt, Moore gives an example of an NFA B_n over a binary alphabet such that B'_n is a minimal DFA with 2^n states [10]. For any $n \geq 2$, define the NFA

$$B_n = (\{q_1, \ldots, q_n\}, \{a, b\}, \delta, q_1, \{q_n\}),$$

where

$$\delta(q_i, a) = \begin{cases} \{q_{i+1}\} & \text{if } i < n \end{cases}$$

and

$$\delta(q_i, b) = \begin{cases} \{q_i\} & \text{if } i = 1 \\ \{q_{i+1}\} & \text{if } 1 < i < n \end{cases}$$

for each $i \in \{1, \ldots, n\}$. For an example, Figure 2.1 contains the transition diagram for the NFA B_4.

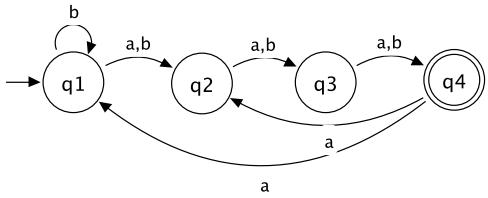

Figure 2.1: Moore's NFA B_4.

Moore shows in the following two lemmas, the proofs of which are omitted, that B'_n is a minimal DFA with 2^n states. As given by Lemma 7, to prove this fact first requires showing that all states of B'_n are pairwise distinguishable.

Lemma 8 (Moore)**.** *The states of B'_n are pairwise distinguishable; that is, for all $P, R \in \mathcal{P}(\{q_1, \ldots, q_n\})$, P and R are equivalent if and only if $P = R$.*

Next, we must show that all 2^n states of B'_n are accessible.

Lemma 9 (Moore)**.** *All states in B'_n are accessible; that is, for each $P \in \mathcal{P}(\{q_1, \ldots, q_n\})$, there exists $w \in \{a, b\}^*$ such that $\delta(q_1, w) = P$.*

Moore's B_n was the first NFA to address the issue of state hierarchy, as it was the first example of an n-state NFA for which the equivalent minimal DFA had 2^n states. Moore concluded his discussion with a proof that no n-state NFA over a 1-symbol alphabet can

have an equivalent minimal DFA of 2^n states, suggesting that his NFA B_n, which uses a binary alphabet, is an optimal answer to the problem.

2.2 Exploring the State Hierarchy

The first to extend the issue, Iwama posed the equation of whether there exists a minimal n-state NFA M such that M' is a minimal DFA with $2^n - k$ states, where $0 \leq k \leq 2^n - n$ [4]. Notice that this question is equivalent to the prompt given in Section 1.3, with the change of variables $d = 2^n - k$. All automata considered in their work use the binary alphabet $\{0, 1\}$, since, as shown by Moore and discussed in Section 2.1, a unary alphabet is not enough answer the question. The authors partially answer the question, presenting constructions for cases in which k can be expressed as $2^n - 2^r$ or $2^n - 2^r - 1$ for some nonnegative integer $r \leq n/2 - 2$.

2.2.1 A Slight Improvement

Continuing from previous work, Iwama and Matsuura present similar constructions with slightly weaker conditions on k [5]. In particular, they answer the question for n and k if $n \geq 7$ and $5 \leq k \leq 2n - 2$ and one of the following hold:

1. $\gcd(n, k - 1) = 1$,
2. $\gcd(n, k - 2) = 1$, or
3. $\gcd(n, \lceil k/2 \rceil - 1) = 1$.

2.3 Capturing the State Hierarchy with a Growing Alphabet

As Iwama writes, fully answering the question (reaching the entire state hierarchy) becomes easier with a larger alphabet [4]. Jirásková was the first to capture the state hierarchy, creating a minimal n-state NFA with a minimal equivalent d-state DFA for all $d \in [n, 2^n]$, but such constructions require an input alphabet that grows exponentially with n [7]. The work is concluded with a nonconstructive improvement, showing that $2n$ symbols would be sufficient for the input alphabet.

Geffert improves the work an additional step, using an input alphabet with $n+2$ symbols [1].

2.4 A Fixed Alphabet

Finally, Jirásek and Jirásková were the first to close the state hierarchy over a fixed alphabet, specifically using four input symbols [6]. Their work begins by creating separate NFA's for the cases in which $d = n$ and $d = 2^n$. For the remaining case in which $n < d < 2^n$, there exists an integer $\alpha \in [1, n)$ such that

$$n - \alpha + 2^\alpha \leq d < n - (\alpha + 1) + 2^{k+1}.$$

Then

$$d = n - (\alpha + 1) + 2^\alpha + m$$

for some integer $m \in [1, 2^\alpha)$. From there, they consider three separate cases on the form of m, and from each case construct different NFA's with different transition functions. Moreover, proving necessary properties about these NFA's require five further subcases on the forms of the integers α, n, and m.

Chapter 3

Results

Here we present the original results of this work.

3.1 A Few Helpful Definitions

These definitions are presented for convenience and will aid in the constructions of this chapter.

Definition 10. *Let M be an NFA with transition relation Δ. A* transition *of M is any element (q, σ, p) of Δ and is denoted $q \xrightarrow{\sigma} p$.*

3.1.1 Submachines

Often we create a new NFA from a given NFA by removing certain transitions and leaving all other components the same. This idea is formalized in the following definition.

Definition 11. *Let $M = (Q, \Sigma, \Delta, q_0, F)$ be an NFA. An NFA N is a* submachine *of M if*

$$N = (Q, \Sigma, \Delta \setminus A, q_0, F)$$

for some set of transitions $A \subseteq \Delta$. We denote N by $M \setminus A$.

Informally, the submachine $M \setminus A$ of M is formed by removing all transitions $\tau \in A$ from M. Since A can be empty, an NFA is always a submachine of itself. Moreover, an NFA with transition relation Δ has exactly $|\mathcal{P}(\Delta)| = 2^{|\Delta|}$ submachines.

3.1.2 Exploding Automata

A particular type of NFA has been central to much of this thesis and warrants a name.

Definition 12. *An* exploding automaton *is a minimal NFA with n states that is equivalent to a minimal DFA with 2^n states.*

The question of whether a minimal NFA M is exploding is always decidable, as M can be converted to a DFA with the subset algorithm and then minimized with the minimization algorithm.

3.2 A Distinguished Machine

For each $n \in \mathbb{N}$, we define an n-state NFA

$$E_n = (Q, \Sigma, \Delta, q_1, \{q_n\})$$

where $Q = \{q_1, \ldots, q_n\}$, and Δ underlies the transition function given by

$$\delta(q_i, k) = \begin{cases} \{q_{i+1}\} & \text{if } k = 1 \text{ and } i < n \\ \{q_i, q_k\} & \text{if } k > 1 \text{ and } 1 \leq i < k \end{cases} \tag{3.1}$$

Intuitively, for each $i < n$, the state q_i of the NFA E_n moves to the next state q_{i+1} on a 1, and it also has self-loops and transitions to q_j for each $j > i$. Finally, the state q_n is an accept state with no transitions. The two-state and three-state examples of this machine are given in Figure 3.1 and Figure 3.2, respectively.

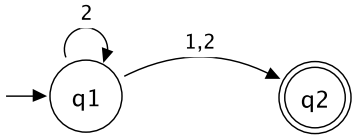

Figure 3.1: The NFA E_2.

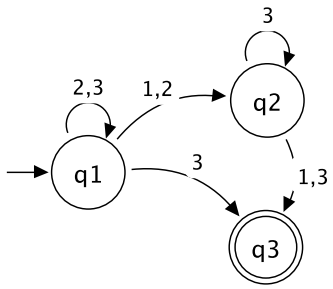

Figure 3.2: The NFA E_3.

3.3 On Greater Intuition

Although the past results given in Chapter 2 are, for the most part, constructive, the NFA's corresponding to each pair n and k depend on many distinct cases. Their construction is arguably unintuitive. Here each NFA will be intimitely related; in particular, for a fixed n, the NFA corresponding to each $k \in [0, 2^{n-1})$ will be a submachine of E_n.

3.4 Answering the Question

The ultimate goal of this work is the following theorem.

Theorem 13. *Let $n \geq 2$ with $0 \leq k < 2^{n-1}$. Then there exists an n-state submachine of E_n for which the minimal equivalent DFA has $2^n - k$ states.*

We will achieve this result by a combination of several interesting properties of the NFA E_n.

3.4.1 E_n Explodes

Our crucial result is that the NFA E_n is an exploding automaton, which we will show with two lemmas. First, consider the NFA E_3, previously illustrated in Figure 3.2. For E_3 to be exploding, its equivalent DFA E_3' must consist of exactly $2^3 = 8$ accessible states (in addition, these 8 states must be pairwise inequivalent, but for now we only consider accessibility). In other words, for each subset A of the state set $\{q_1, q_2, q_3\}$ of E_3, there must exist a corresponding $w \in \Sigma_3^*$ such that $\delta(q_1, w) = A$, where δ is the transition function of E_3. The state set $\{q_1\}$ is already taken care of, since it is the start state of E_3'. For the other singletons as well as the empty set, we see that $\delta(q_1, 1) = \{q_2\}$ and $\delta(q_1, 1^2) = \{q_3\}$ and $\delta(q_1, 1^3) = \varnothing$. Generally, it appears that any singleton state set can be accessed by a string of 1's, and the empty state set can be accessed by using too many 1's (in a sense, traveling off the edge of the transition diagram).

The four larger subsets of $\{q_1, q_2, q_3\}$ remain. Since there is more than one way to access each of these state sets, we consider the shortest possible input strings for each. In

particular, we have that

$$\delta(q_1, 2) = \{q_1, q_2\},$$
$$\delta(q_1, 3) = \{q_1, q_3\},$$
$$\delta(q_1, 2 \circ 3) = \{q_1, q_2, q_3\}, \text{ and}$$
$$\delta(q_1, 1 \circ 3) = \{q_2, q_3\}.$$

The above cases suggest a simple rule for accessing a state set in E_3'. To build an appropriate string w to reach subset A of $\{q_1, q_2, q_3\}$ in E_3, first use a certain number of 1's to reach the earliest state in A (if q_1 is the earliest state, use zero 1's). Once the earliest state has been reached, use the input symbol i for each remaining state q_i in A, in ascending order. For example, to reach the state set $\{q_1, q_2, q_3\}$ in E_3 above, we used zero 1's to reach state q_1, and then used a 2 and a 3, corresponding to states q_2 and q_3, respectively. This gives the string $2 \circ 3$, which will reach $\{q_1, q_2, q_3\}$ in E_3.

The subtle interplay of the state subscripts and alphabet symbols suggested above with E_3 can perhaps be made clearer with a larger example. Consider the machine E_5, which is illustrated in Figure 3.3. To reach the state set $\{q_2, q_4, q_5\}$ in E_5, we first input a 1 to move from q_1 to q_2. Then we input a 4 (corresponding to the state q_4) followed by a 5 (corresponding to q_5). Hence

$$\delta(q_1, 2 \circ 4 \circ 5) = \{q_2, q_4, q_5\},$$

where δ is the transition function of E_5.

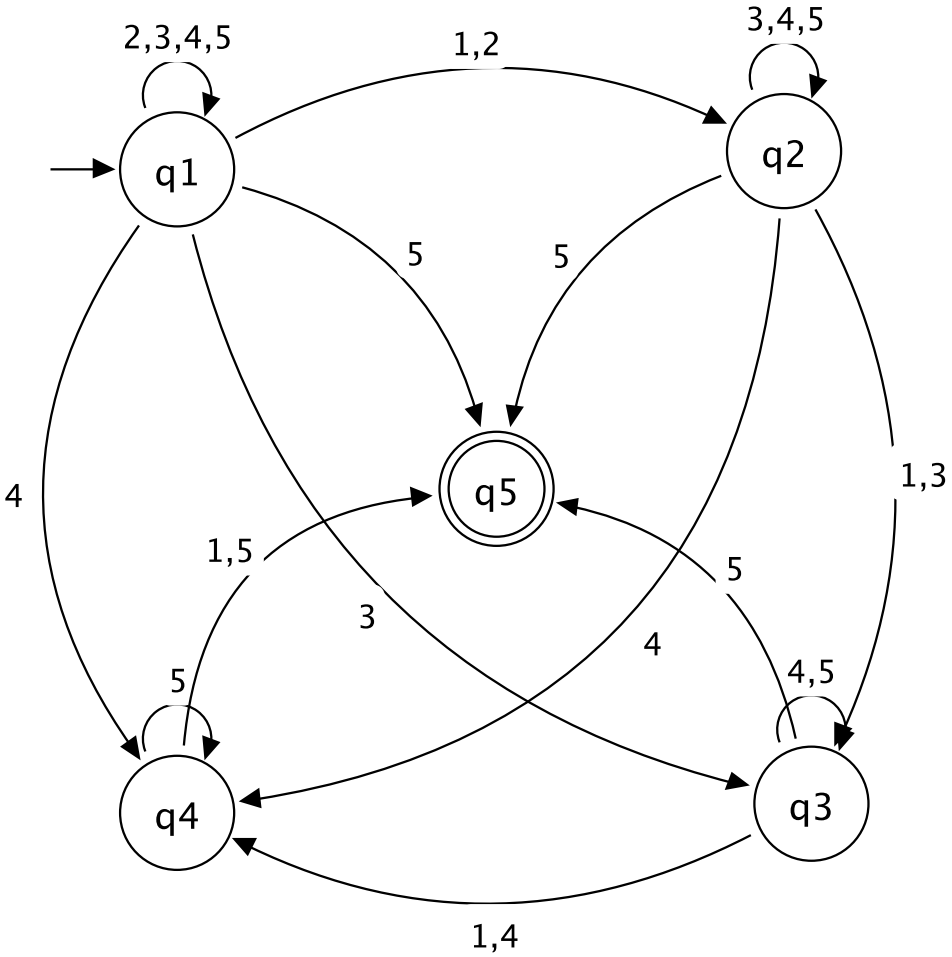

Figure 3.3: The NFA E_5.

Now armed with a bit of motivation, we may now formally prove that all 2^n possible states of E'_n are accessible, by using the method of string construction suggested above.

Lemma 14. *The DFA E'_n has 2^n states.*

Proof. Let Q and δ be the state set and transition function of E_n, respectively. Let $A \in \mathcal{P}(Q)$; we will show that A is accessible. If $A = \varnothing$, then

$$\delta(q_1, 1^n) = \delta(q_n, 1) = \varnothing = A.$$

Otherwise, $A = \{q_{a_1}, \ldots, q_{a_k}\}$ for some $k \in \mathbb{N}$ such that $1 \leq a_1 < \ldots < a_k \leq n$. We will show by induction on k that

$$\delta(q_1, 1^{a_1 - 1} a_2 \ldots a_k) = \{q_{a_1}, \ldots, q_{a_k}\}. \tag{3.2}$$

If $k = 1$, then $\delta(q_1, 1^{a_1 - 1}) = \{q_{a_1}\}$, which proves the base case. Now suppose that (3.2) holds true for some fixed $m \in \mathbb{N}$. Then

$$\begin{aligned}
\delta(q_1, 1^{a_1 - 1} a_2 \ldots a_m a_{m+1}) &= \delta(\delta(q_1, 1^{a_1 - 1} a_2 \ldots a_m), a_{m+1}) \\
&= \delta(\{q_{a_1}, \ldots, q_{a_m}\}, a_{m+1}) && \text{(by the inductive hypothesis)} \\
&= \bigcup_{i=1}^{m} \delta(q_{a_i}, a_{m+1}) && \text{(by the natural extension of } \delta \text{ in (1.2))} \\
&= \bigcup_{i=1}^{m} \{q_{a_i}, q_{a_{m+1}}\} && \text{(by construction in (3.1), since } a_i < a_{m+1}) \\
&= \{q_{a_1}, \ldots, q_{a_m}, q_{a_{m+1}}\}.
\end{aligned}$$

Hence all of the 2^n subsets of Q are accessible from the start state $\{q_1\}$ in E'_n. □

Minimality and Inequivalence

To show that E_n and E'_n are minimal, we employ a helpful lemma due to Jirásková [7].

Lemma 15 (Jirásková)**.** *Let $M = (\{q_1, \ldots, q_n\}, \Sigma_n, \Delta, q_1, \{q_n\})$ be an n-state NFA such that $\delta(q_i, 1) = \{q_{i+1}\}$ for each $i \in \{1, \ldots, n-1\}$, and $\delta(q_n, 1) = \varnothing$ (the other transitions may be arbitrary). Then*

1. *M is a minimal NFA, and*
2. *No two different states of the DFA obtained from M by the subset construction are equivalent.*

The NFA E_n satisfies the hypothesis of Lemma 15 by construction. Conveniently, this lemma not only implies the minimality of E_n but also eliminates the need to show that the states of E'_n are pairwise inequivalent, which is all that was left to show that E'_n is minimal.

Corollary 16. *The NFA E_n is an exploding automaton.*

Proof. By Lemma 15, E_n is a minimal n-state NFA that is equivalent to the minimal DFA E'_n, which, by Lemma 14, has 2^n states. □

For an additional demonstration, Appendix A contains a detailed example of the explosion from E_4 to E'_4.

3.4.2 The Submachines of E_n

What is unique about Theorem 13 is not the state hierarchy it reaches; rather, it is that each member of the state hierarchy will be reached by a submachine of E_n. Specifically, given n and k, we form the necessary NFA by plucking transitions off of E_n. To preserve

the minimality of E_n and E'_n guaranteed by Lemma 15, we preserve all transitions on the symbol 1 and only explore removing transitions on other symbols.

Consider the NFA E_4 with transition function δ. Following the method of string construction used in the proof of Lemma 14, Figure 3.4 contains the input strings that can be used to reach the non-singleton state sets in E_4; that is, each state set $A \subseteq \{q_1, q_2, q_3, q_4\}$ appearing in the table's left column is accompanied by a string $w \in \Sigma_4^*$ in the right column for which $\delta(q_1, w) = A$.

State Set	Input String
$\{q_1, q_2\}$	2
$\{q_1, q_3\}$	3
$\{q_1, q_4\}$	4
$\{q_2, q_3\}$	$1 \circ 3$
$\{q_2, q_4\}$	$1 \circ 4$
$\{q_3, q_4\}$	$1^2 \circ 4$
$\{q_1, q_2, q_3\}$	$2 \circ 3$
$\{q_1, q_2, q_4\}$	$2 \circ 4$
$\{q_1, q_3, q_4\}$	$3 \circ 4$
$\{q_2, q_3, q_4\}$	$1 \circ 3 \circ 4$
$\{q_1, q_2, q_3, q_4\}$	$2 \circ 3 \circ 4$

Figure 3.4: The state sets and corresponding input strings of the DFA E'_4.

We will first consider the effects of removing transitions from the start state q_1. As suggested in Figure 3.5, which contains a partial transition diagram of E'_4, removing the transition $q_1 \xrightarrow{4} q_4$ from E_4 renders the one state set $\{q_1, q_4\}$ inaccessible, but leaves all other state sets accessible. Removing the transition $q_1 \xrightarrow{3} q_3$ from E_4 causes us to lose the state sets $\{q_1, q_3\}$ and $\{q_1, q_3, q_4\}$. Finally, removing the transition $q_1 \xrightarrow{2} q_2$ has a more drastic effect on E_4, severing from it the four state sets $\{q_1, q_2\}$, $\{q_1, q_2, q_3\}$, $\{q_1, q_2, q_4\}$, and $\{q_1, q_2, q_3, q_4\}$.

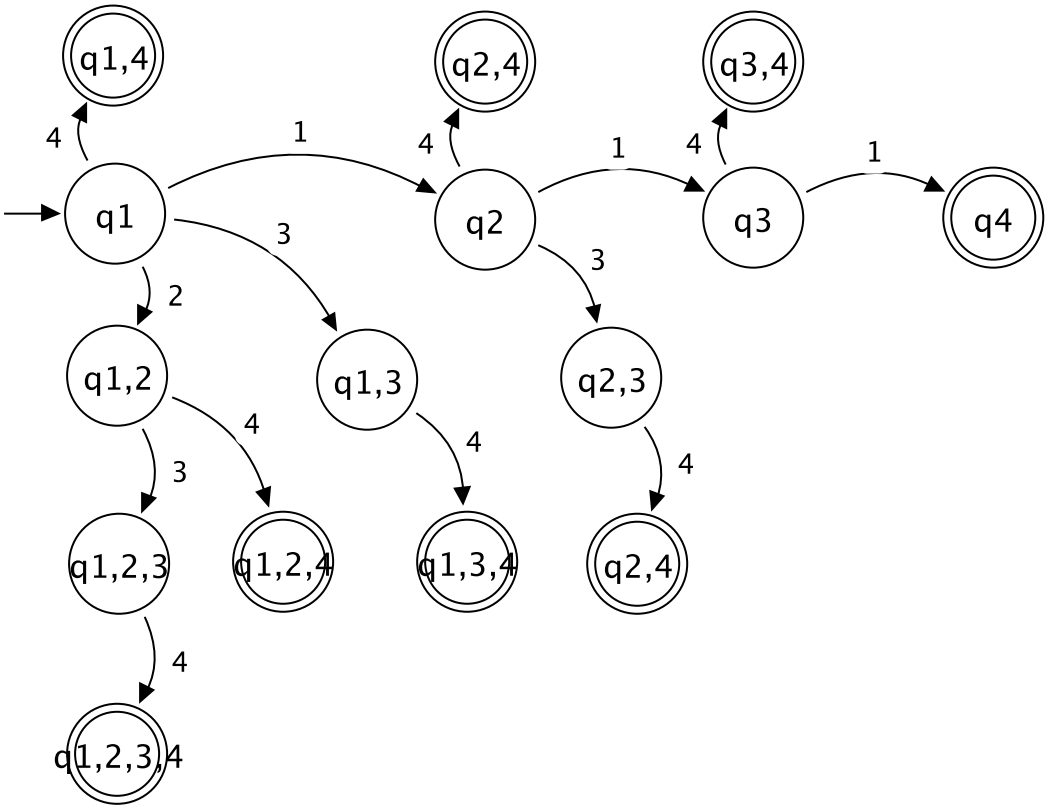

Figure 3.5: A partial transition diagram of the DFA E_4'.

As we will see later, it is no mistake that the number of state sets made inaccessible by the removals considered above is always a power of 2. For now, we will attempt to suggest a characterization of the state sets that remain accessible in E_4 following the removal of the transition $q_1 \xrightarrow{3} q_3$. Let M be the submachine resulting from this removal; then $M = E_4 \setminus \{q_1 \xrightarrow{3} q_3\}$. It seems that any state set $A \subseteq \{q_1, q_2, q_3, q_4\}$ that does not contain $\{q_1, q_3\}$ is accessible in M. Indeed, to reach such a state set A in the above figure does not require the path from $\{q_1\}$ to $\{q_1, q_3\}$. However, the converse of this statement is not necessarily true, since the state sets $\{q_1, q_2, q_3\}$ and $\{q_1, q_2, q_3, q_4\}$, both of which contain $\{q_1, q_3\}$, remain accessible in M. It turns out that what keeps them accessible is that they contain the state q_2 between q_1 and q_3.

Putting all of this together hints at a general characterization of accessible states in certain submachines of E_n.

Lemma 17. *Let A be a state set of the submachine $M = E_n \setminus \{q_1 \xrightarrow{m} q_m\}$ of E_n such that $1 < m \leq n$. Then A is accessible in M if and only if*

1. *$\{q_1, q_m\} \not\subseteq A$, or*

2. *A contains a state q_i for which $1 < i < m$.*

Proof. We prove the forward implication by contraposition. Let A be a set of states of M such that $\{q_1, q_m\} \subseteq A$ and $i \geq m$ for all $q_i \in A \setminus \{q_1\}$. Let $w \in \Sigma_n^*$. We induct on the length of w to show that if $\{q_1, q_m\} \subseteq \delta_M(q_1, w)$, then there exists a state $q_j \in \delta_M(q_1, w)$ for which $1 < j < m$. This will imply that $\delta_M(q_1, w) \neq A$ for all $w \in \Sigma_n^*$, or, equivalently, A is inaccessible in M. If $w = \epsilon$, then $\delta_M(q_1, w) = \{q_1\} \not\supseteq \{q_1, q_m\}$, and so the claim holds true by vacuousness. Assume the inductive hypothesis for some fixed $k \geq 0$. Let $w = \sigma v$ for some $\sigma \in \Sigma_n$ and $v \in \Sigma_n^*$ such that $|v| = k$, and let $B = \delta_M(q_1, v)$. Suppose that

$\{q_1, q_m\} \subseteq B$. By the inductive hypothesis, there exists $q_j \in B$ for which $1 < j < m$. If $\sigma = 1$, $q_1 \notin \delta_M(B, \sigma)$, satisfying the claim again by vacuousness. Similarly, if $1 < \sigma \leq j$, then $q_m \notin \delta_M(B, \sigma)$ since $j < m$. Finally, if $\sigma > j$, then M contains the transition $q_j \xrightarrow{\sigma} q_j$, and so $q_j \in \delta_M(B, \sigma)$.

The converse requires a slight alteration of the proof of Lemma 14. Let A be a state set of M such that $\{q_1, q_m\} \not\subseteq A$ or A contains a state q_i for which $1 < i < m$. If $A = \varnothing$, then $\delta_M(q_1, 1^n) = \varnothing = A$. Otherwise, $A = \{q_{a_1}, \ldots, q_{a_t}\}$ for some $t \in \mathbb{N}$ such that $1 \leq a_1 < \ldots < a_t \leq n$. We will show by induction on t that

$$\delta(q_1, 1^{a_1 - 1} a_2 \ldots a_t) = \{q_{a_1}, \ldots, q_{a_t}\}, \tag{3.3}$$

which implies that A is accessible in all cases. For the base case, we have that $\delta(q_1, 1^{a_1 - 1}) = \{q_{a_1}\}$. Suppose that (3.3) holds for some fixed $k \in \mathbb{N}$, and suppose that $t = k + 1$. Let

$$B = A \setminus \{q_{k+1}\} = \{q_{a_1}, \ldots, q_{a_k}\}.$$

Consider the two possible cases on A from the hypothesis of the converse. If $\{q_1, q_m\} \not\subseteq A$, then $\{q_1, q_m\} \not\subseteq B$ as well, since $B \subseteq A$. In the other case, A contains a state q_{a_j} for some $j \in \{1, \ldots, k+1\}$ such that $1 < a_j < m$. If $j = k + 1$, then $q_m \notin A$ and so $\{q_1, q_m\} \not\subseteq B$. Otherwise, $q_j \in B$. So, in any case, the state set B satisfies the hypothesis of the converse, and so the inductive hypothesis guarantees that

$$\delta(q_1, 1^{a_1 - 1} a_2 \ldots a_k) = \{q_{a_1}, \ldots, q_{a_k}\} = B.$$

Now, if $a_{k+1} \neq m$, then

$$\begin{aligned}
\delta(B, a_{k+1}) &= \bigcup_{i=1}^{k} \delta(q_{a_i}, a_{k+1}) \\
&= \bigcup_{i=1}^{k} \{q_{a_i}, q_{a_{k+1}}\} && \text{(by construction in (3.1), since } a_i < a_{k+1}) \\
&= \{q_{a_1}, \ldots, q_{a_{k+1}}\} \\
&= A.
\end{aligned}$$

Otherwise, $a_{k+1} = m$, and so $a_1 < \ldots < a_k < a_{k+1} = m$. If $q_1 \in B$, then $a_1 = 1$ and so

$$\begin{aligned}
\delta(B, a_{k+1}) &= \delta(q_1, m) \cup \bigcup_{i=2}^{k} \delta(q_{a_i}, m) \\
&= \{q_1\} \cup \bigcup_{i=2}^{k} \delta(q_{a_i}, m) && \text{(since } q_1 \xrightarrow{m} q_m \notin \Delta_M) \\
&= \{q_1\} \cup \bigcup_{i=2}^{k} \{q_{a_i}, q_m\} && \text{(by construction in (3.1), since } 1 < a_i < m) \\
&= \{q_1\} \cup \{q_{a_2}, \ldots, q_{a_k}\} \cup \{q_m\} \\
&= A.
\end{aligned}$$

If $q_1 \notin B$, then

$$\begin{aligned}
\delta(B, a_{k+1}) &= \bigcup_{i=1}^{k} \delta(q_{a_i}, m) \\
&= \bigcup_{i=1}^{k} \{q_{a_i}, q_m\} && \text{(by construction in (3.1), since } 1 < a_i < m) \\
&= \{q_{a_1}, \ldots, q_{a_k}\} \cup \{q_m\} \\
&= A.
\end{aligned}$$

We have shown that in all cases,

$$\delta(q_1, 1^{a_1 - 1} a_2 \ldots a_{k+1}) = \delta(B, a_{k+1}) = A.$$

Hence A is accessible in M. □

The Sizes of Such Submachines

Lemma 17 has provided a characterization of exactly which state sets are lost from E_n when certain transitions are removed. Using this characterization, we would like to count exactly how many state sets are rendered inaccessible upon the removal of a transition $q_1 \xrightarrow{m} q_m$ such that $1 < m \leq n$. In the previous discussion, we saw that removing the transition $q_1 \xrightarrow{4} q_4$ from E_4 causes us to lose exactly $2^0 = 1$ set state, meaning that the DFA $(E_4 \setminus \{q_1 \xrightarrow{4} q_4\})'$ has $2^n - 2^0$ states. Similarly, removing $q_1 \xrightarrow{3} q_3$ loses $2^1 = 2$ set states, and removing $q_1 \xrightarrow{2} q_2$ loses $2^2 = 4$ set states. A pattern is emerging.

Lemma 18. *Let $m \in \mathbb{N}$ such that $1 < m \leq n$, and let $M = E_n \setminus \{q_1 \xrightarrow{m} q_m\}$. Then M' is a minimal DFA with $2^n - 2^{n-m}$ states.*

Proof. By Lemma 17, a set state A of M is inaccessible if and only if $\{q_1, q_m\} \subseteq A$ and

$i \geq m$ for all $q_i \in A \setminus \{q_1\}$. Let S_m be the set of all such sets A. Then

$$S_m = \{T \cup \{q_1, q_m\} \mid T \in \mathcal{P}(\{q_i \mid m < i \leq n\})\}, \tag{3.4}$$

and so there are exactly

$$|S_m| = |\mathcal{P}(\{q_i \mid m < i \leq n\})| = 2^{n-m}$$

inaccessible states in M'. Since $m > 1$, Lemma 15 implies that M' is minimal once these inaccessible states are removed, which leaves $2^n - 2^{n-m}$ accessible states. □

Again referring back to the discussion of E_4, notice that state sets lost as a result of one transition removal are all different from those lost via another transition removal. For example, removing $q_1 \xrightarrow{4} q_4$ loses $\{q_1, q_4\}$ from E_4, and removing $q_1 \xrightarrow{3} q_3$ loses $\{q_1, q_3\}$ and $\{q_1, q_3, q_4\}$. Then removing both transitions at once will surely lose all three of these state sets, but does this work in general? As the following Lemma shows, it turns out that the effects of multiple transition removals such as these are always disjoint.

Lemma 19. *Let $r_1, \ldots, r_t$ be positive integers for some $t \in \mathbb{N}$ such that $1 < r_1 < \ldots < r_t \leq n$, and define S_{r_i} as in* (3.4) *from Lemma 18 for each $i \in \{1, \ldots, t\}$. Then $S_{r_1}, \ldots, S_{r_t}$ are pairwise disjoint.*

Proof. Induct on t. The base case is trivial. Assume the inductive hypothesis for some $k \in \mathbb{N}$, and let $r_1, \ldots, r_k, r_{k+1} \in \mathbb{N}$ such that $1 < r_1 < \ldots < r_{k+1} \leq n$. By the inductive hypothesis, $S_{r_2}, \ldots, S_{r_{k+1}}$ are pairwise disjoint. Let

$$S = \bigcup_{i=2}^{k+1} S_{r_i}.$$

To show that S_{r_1} and S are disjoint is sufficient to complete the proof. By construction in (3.4), $q_{r_1} \in X$ for all $X \in S_{r_1}$. Let $Y \in S$. Then, again by construction in (3.4),

$$\min\{r_i \mid q_{r_i} \in Y \setminus \{q_1\}\} \geq r_2 > r_1,$$

and so $q_{r_1} \notin Y$. Thus $X \neq Y$ for all $X \in S_{r_1}$ and $Y \in S$, and so $S_{r_1} \cap S = \varnothing$. □

We must prove one more subtle fact concerning the removal of such transitions from E_n. For an arbitrary NFA M with distinct transitions τ_1 and τ_2, let $S(\tau_1)$ and $S(\tau_2)$ be the sets of state sets made inaccessible in M upon the removal of the transitions τ_1 and τ_2, respectively. Furthermore, let $S(\tau_1, \tau_2)$ be the set of state sets made inaccessible in M upon the removal of *both* transitions τ_1 and τ_2. Even if $S(\tau_1)$ and $S(\tau_2)$ are disjoint, it is not necessarily true that

$$S(\tau_1, \tau_2) = S(\tau_1) \cup S(\tau_2). \tag{3.5}$$

For an example, consider the NFA

$$M = (\{q_a, q_b\}, \{1, 2\}, \Delta, q_a, \{q_b\})$$

with

$$\Delta = \{q_a \xrightarrow{1} q_b, q_a \xrightarrow{2} q_b\}.$$

Even though the sets $S(q_a \xrightarrow{1} q_b) = \varnothing$ and $S(q_a \xrightarrow{2} q_b) = \varnothing$ are disjoint, removing both transitions from M renders the state set $\{q_b\}$ inaccessible, which did not happen upon either of the individual removals. Thus,

$$S(q_a \xrightarrow{1} q_b, q_a \xrightarrow{2} q_b) = \{q_b\} \neq \varnothing = S(q_a \xrightarrow{1} q_b) \cup S(q_a \xrightarrow{2} q_b).$$

Fortunately, as the following lemma shows, the notion behind (3.5) does hold true in the case of E_n.

Lemma 20. *Let $r_1, \ldots, r_t$ be positive integers for some $t \in \mathbb{N}$ such that $1 < r_1 < \ldots < r_t \leq n$, and define S_{r_i} as in* (3.4) *from Lemma 18 for each $i \in \{1, \ldots, t\}$. Define C_t to be the set of all state sets made inaccessible in E_n upon the removal of the transitions $q_1 \xrightarrow{r_1} q_{r_1}, \ldots, q_1 \xrightarrow{r_t} q_{r_t}$. Then $C_t = \bigcup_{i=1}^{t} S_{r_i}$.*

Proof. Induct on t. The base case is trivial, since C_1 is the set of all state sets made inaccessible upon the removal of the transition $q_1 \xrightarrow{r_1} q_{r_1}$, which is S_{r_1} by definition. Assume the inductive hypothesis for some fixed $k \in \mathbb{N}$, and let $r_1, \ldots, r_k, r_{k+1} \in \mathbb{N}$ such that $1 < r_1 < \ldots < r_{k+1} \leq n$. Seeking a contradiction, suppose that

$$\bigcup_{i=1}^{k+1} S_{r_i} \not\subseteq C_{k+1}.$$

Then there exists some state set $A \in \bigcup_{i=1}^{k+1} S_{r_i}$ such that $A \notin C_{k+1}$. Then A is made inaccessible in E_n by the removal of some transition $q_1 \xrightarrow{r_j} q_{r_j}$ for some $j \in \{1, \ldots, k+1\}$, but A is made accessible again in E_n once the k remaining transitions are removed. This is a contradiction, since removing transitions from a finite automaton cannot cause inaccessible states to become accessible. Thus we have that

$$\bigcup_{i=1}^{k+1} S_{r_i} \subseteq C_{k+1}.$$

We show the reverse containment by contraposition. Let A be a state set of E_n such that

$$A \notin \bigcup_{i=1}^{k+1} S_{r_i}.$$

Then $A \notin C_k$ by the inductive hypothesis, and A is accessible in $E_n \setminus \{q_1 \xrightarrow{r_1} q_{r_1}\}$, ..., $E_n \setminus \{q_1 \xrightarrow{r_k} q_{r_k}\}$, and $E_n \setminus \{q_1 \xrightarrow{r_{k+1}} q_{r_{k+1}}\}$. Furthermore, Lemma 17 implies that $\{q_1, q_{r_{k+1}}\} \not\subseteq A$ or A contains a state q_{r_j} with $1 < r_j < r_{k+1}$. Let

$$M = E_n \setminus \{q_1 \xrightarrow{r_1} q_{r_1}, \ldots, q_1 \xrightarrow{r_{k+1}} q_{r_{k+1}}\}.$$

If $A = \varnothing$, then $\delta_M(q_1, 1^n) = \delta_M(q_n, 1) = \varnothing$, and so A is accessible in M. Otherwise, $A = \{q_{a_1}, \ldots, q_{a_m}\}$ for some $m \in \mathbb{N}$ such that $1 \leq a_1 < \ldots < a_m \leq n$.

Here we must employ a second level of induction. Specifically, we will induct on m to show that

$$\delta_M(q_1, 1^{a_1 - 1} a_2 \ldots a_m) = \{q_{a_1}, \ldots, q_{a_m}\}. \tag{3.6}$$

For the base case, we have that $\delta_M(q_1, 1^{a_1 - 1}) = \{q_{a_1}\}$. Suppose that (3.6) holds for some fixed $s \in \mathbb{N}$, and suppose that $m = s + 1$. Let

$$B = A \setminus \{q_{a_{s+1}}\}.$$

As in the proof of Lemma 17, since A is accessible in $E_n \setminus \{q_1 \xrightarrow{r_1} q_{r_1}\}$, ..., $E_n \setminus \{q_1 \xrightarrow{r_k} q_{r_k}\}$, and $E_n \setminus \{q_1 \xrightarrow{r_{k+1}} q_{r_{k+1}}\}$ and $B \subseteq A$, we have that B is accessible in each of the above $k + 1$ submachines as well. Thus

$$B \notin \bigcup_{i=1}^{k+1} S_{r_i}.$$

So, the inductive hypothesis guarantees that

$$\delta(q_1, 1^{a_1 - 1} a_2 \ldots a_s) = \{q_{a_1}, \ldots, q_{a_s}\} = B.$$

Now, if $a_{s+1} \notin \{r_1, \ldots, r_{k+1}\}$, then

$$\begin{aligned}
\delta_M(B, a_{s+1}) &= \bigcup_{i=1}^{s} \delta_M(q_{a_i}, a_{s+1}) \\
&= \bigcup_{i=1}^{s} \{q_{a_i}, q_{a_{s+1}}\} && \text{(by construction in (3.1), since } a_i < a_{s+1}) \\
&= \{q_{a_1}, \ldots, q_{a_{s+1}}\} \\
&= A.
\end{aligned}$$

Otherwise, $a_{s+1} = r_j$ for some $j \in \{1, \ldots, k+1\}$, and so $a_1 < \ldots < a_s < a_{s+1} = r_j$. If $q_1 \in B$, then $a_1 = 1$ and so

$$\begin{aligned}
\delta_M(B, a_{s+1}) &= \delta_M(q_1, r_j) \cup \bigcup_{i=2}^{s} \delta_M(q_{a_i}, r_j) \\
&= \{q_1\} \cup \bigcup_{i=2}^{s} \delta_M(q_{a_i}, r_j) && \text{(since } q_1 \xrightarrow{r_j} q_{r_j} \notin \Delta_M) \\
&= \{q_1\} \cup \bigcup_{i=2}^{s} \{q_{a_i}, q_{r_j}\} && \text{(by construction in (3.1), since } 1 < a_i < r_j) \\
&= \{q_1\} \cup \{q_{a_2}, \ldots, q_{a_s}\} \cup \{q_{r_j}\} \\
&= A.
\end{aligned}$$

Finally, if $q_1 \notin B$, then

$$\begin{aligned}
\delta_M(B, a_{s+1}) &= \bigcup_{i=1}^{s} \delta_M(q_{a_i}, r_j) \\
&= \bigcup_{i=1}^{s} \{q_{a_i}, q_{r_j}\} && \text{(by construction in (3.1), since } 1 < a_i < r_j\text{)} \\
&= \{q_{a_1}, \ldots, q_{a_s}\} \cup \{q_{r_j}\} \\
&= A.
\end{aligned}$$

We have shown that in all cases,

$$\delta_M(q_1, 1^{a_1-1}a_2 \ldots a_{s+1}) = \delta_M(B, a_{s+1}) = A.$$

Hence A is accessible in M, which completes the second induction. Furthermore, this implies that $A \notin C_{k+1}$, and so

$$C_{k+1} \subseteq \bigcup_{i=1}^{k+1} S_{r_i},$$

which completes the first induction. □

Since each transition removal will cause us to lose a unique collection of state sets from E_n, we may remove more than one at a time to move farther along the state hierarchy.

Lemma 21. *Let $r_1, \ldots, r_t$ be positive integers for some $t \in \mathbb{N}$ such that $1 < r_1 < \ldots < r_t \leq n$. If*

$$M = E_n \setminus \{q_1 \xrightarrow{r_1} q_{r_1}, \ldots, q_1 \xrightarrow{r_t} q_{r_t}\},$$

then M' is a minimal DFA with

$$2^n - 2^{n-r_1} - \ldots - 2^{n-r_t}$$

states.

Proof. Define S_{r_i} as in (3.4) for each $i \in \{1, \ldots, t\}$. Then, by Lemma 20, the number of inaccessible states in M' is

$$\left| \bigcup_{i=1}^{t} S_{r_i} \right| = \sum_{i=1}^{t} |S_{r_i}| \qquad \text{(by Lemma 19)}$$

$$= \sum_{i=1}^{t} 2^{n-r_i}. \qquad \text{(by Lemma 18)}$$

Since $r_t > \ldots > r_1 > 1$, Lemma 15 implies that M' is minimal once these inaccessible states are removed, which leaves

$$2^n - \sum_{i=1}^{t} 2^{n-r_i} = 2^n - 2^{n-r_1} - \ldots - 2^{n-r_t}$$

accessible states. □

Completing the Proof

With these key lemmas the proof of Theorem 13 results as an immediate corollary. We restate it here.

Theorem 13. *Let $k \in \mathbb{N}$ such that $0 \leq k < 2^{n-1}$. Then there exists an n-state submachine of E_n for which the minimal equivalent DFA has $2^n - k$ states.*

Proof. Write k in its binary form

$$k = a_{n-2} \cdot 2^{n-2} + \ldots + a_1 \cdot 2 + a_0$$

where $a_i \in \{0, 1\}$ for each $i \in \{0, 1, \ldots, n-2\}$. Define the set of transitions

$$A = \{q_1 \xrightarrow{n-i} q_{n-i} \mid a_i = 1\},$$

and let $M = E_n \setminus A$, which is minimal since $n - i > 1$ for each $i \in \{0, 1, \ldots, n-2\}$. By Lemma 21, M' is a minimal DFA, and its number of states is

$$2^n - \sum_{\substack{i=0 \\ a_i=1}}^{n-2} 2^i = 2^n - \sum_{i=0}^{n-2} a_i \cdot 2^i = 2^n - k.$$

□

A fully worked out example of the algorithm suggested by the proof of Theorem 13 is given in Appendix A using the machine E_4.

Chapter 4

Conclusion & Future Work

This thesis has presented an intuitive partial solution to the question of whether there exists a minimal n-state NFA with an equivalent minimal d-state DFA for any $n \in \mathbb{N}$ with $n \leq d \leq 2^n$. As discussed in Chapter 2, the examination of state hierarchy first began with a speculation by Rabin. Following the original creation of the subset algorithm from Theorem 4, Rabin noted the implicit bound of $d \leq 2^n$ and asked whether it was optimal or if $d < 2^n$ [11, 12]. Moore soonafter presented an example in which $d = 2^n$ with a binary alphabet. Nearly thirty years later came a resurgence of the question, when Iwama generalized Rabin's question to the other values of d in the interval $[n, 2^n]$ [4]. Here it became clear that finding NFA's to answer the question becomes easier when using alphabets with more than two symbols. Also, the change of variables $k = 2^n - d$ became a standard convention in approaching the question. Figure 4.1 contains a summary of the accomplishments made on this problem in chronological order.

Author	$\lvert\Sigma\rvert$	*Restrictions on* $k \in [0, 2^n - n]$
Moore [10]	2	$k = 0$
Iwama [4]	2	$k \in \{2^n - 2^r, 2^n - 2^r - 1\}$ for some $r \leq n/2 - 2$
Iwama & Matsuura [5]	2	many concerning relative primality; see Section 2.2.1
Jirásková [7]	$2^{n-1} + 1$	none
Jirásková [7]	$2n$	none, but argument is nonconstructive
Geffert [1]	$n + 2$	none
Jirásek & Jirásková [6]	4	none
Present work	n	$k < 2^{n-1}$, but class of machines is intuitive and may eventually yield no restrictions

Figure 4.1: A chronology of accomplishments regarding NFA state hierarchy.

Although the results of this thesis fit in the table with an alphabet size $|\Sigma| = n$ and restriction $k < 2^{n-1}$, the value of this work is not in the partial state hierarchy it achieves. What distinguishes the results of this thesis is that, each point in this partial state hierarchy is reached not through vastly different NFA constructions depending on various cases of k, but rather through intuitive and systematic modifications to the NFA E_n. By considering the value $k = 2^n - d$ in its binary form, we remove from E_n a transition corresponding to each 1-bit in k. What results is a minimal n-state submachine M of E_n for which the minimal equivalent DFA M' has $d = 2^n - k$ states.

Since this method only works for $k \in [0, 2^{n-1})$ (meaning it only answers the question when $2^{n-1} < d \leq 2^n$), in this chapter we present future work suggesting that there do exist similar methods to reach the remaining elements of the state hierarchy; that is, there do exists submachines of E_n for which the minimal equivalent DFA has d states with $n \leq d \leq 2^{n-1}$.

4.1 Other Minimal Submachines of E_n

This thesis has considered submachines of E_n of the form $E_n \setminus A$, where A contains any number of transitions of the form $q_1 \xrightarrow{m} q_m$ such that $1 < m \leq n$. Intuitively, these submachines are formed by plucking off transitions from E_n that start from the state q_1 and lead to any other state. There are, of course, many other transitions that we may consider for removal as well when forming a submachine. To keep all submachines minimal as guaranteed by Lemma 15, we will not remove any transitions on the symbol 1. By counting all other transitions from the transition function for E_n constructed in (3.1), we see that there are

$$\begin{aligned} 2[(n-1) + (n-2) + \ldots + 1] &= n(n-1) \\ &= n^2 - n \end{aligned}$$

transitions in E_n that are available for removal. Let A_n be the set containing these transitions. Since a minimal submachine of E_n can be formed by removing any number of these $n^2 - n$ transitions from E_n, there are then

$$|\mathcal{P}(A_n)| = 2^{n^2-n}$$

minimal submachines to consider. A small subset (call it W) of these submachines was considered in Chapter 3; in particular, it was shown that for any $d \in (2^{n-1}, 2^n]$, there exists $M \in W$ such that M' is a minimal DFA with d states. By considering these other submachines, we hope to reach each remaining $d \in [n, 2^{n-1}]$.

4.1.1 Scanning the State Hierarchy

To this end, we present an algorithm to take each submachine N from the 2^{n^2-n}, compute the DFA N', minimize it, and count its number of states. As we try all possible submachines, we keep a running tally of how many times a certain number of states (an element of the state hierarchy) has been hit. The Java package `dk.brics.automaton` is extremely helpful here; it provides implementations of NFA's, DFA's, the subset algorithm, and various minimization algorithms.

Program Design

The code for this algorithm, given in Appendix B, is fairly straightforward. The main class `StateRangeScanner` accepts a single integer argument, which is the value of n for which we test the $n^2 - n$ submachines of E_n. The program then builds the exploding automaton E_n as an object of the class `ExplodingAutomaton`. Objects of the class `Submachine` represent submachines of E_n, and are given a set of "transition removals" upon construction. The submachine is built as a copy of E_n, using all of its transitions except for the ones given in this set of removals. The submachine is then converted into a DFA via the subset algorithm and minimized using Hopcroft's minimization algorithm [3]. The number of states of the resultant DFA is then some element of the state hierarchy $[n, 2^n]$, and its occurrence is tallied. The algorithm then repeats with another submachine of E_n, which is created from another set of transitions to remove.

One interesting design issue for this code is in the class `PowersetIterator`, which is used to iterate over all subsets of the set of removable transitions (the transitions not on the symbol 1) from E_n. Since the quantity 2^{n^2-n} grows very quickly, it is infeasible to store all subsets at once and return them one-by-one. Instead, we use the binary representation

of the iterator's current index to decide which transitions to include in the next subset. For example, in the case that $n = 3$, the `PowersetIterator` has $2^{3^2-3} = 64$ subsets to consider from the set of removal transitions

$$A_3 = \{q_1 \xrightarrow{2} q_1, q_1 \xrightarrow{3} q_1, q_1 \xrightarrow{2} q_2, q_1 \xrightarrow{3} q_3, q_2 \xrightarrow{3} q_2, q_2 \xrightarrow{3} q_3\}.$$

When the `PowersetIterator` is created, its `index` value is initialized to 0. When a subset of transitions is requested from the `PowersetIterator`, it includes in this subset the ith transition from its set A_n of removal transitions if and only if the ith bit (starting from the least significant end) in the binary representation of `index` is 1. Once this subset is built, it increments the value of `index` by 1 before building the next subset. So, in the above example, since the value 0 has the binary representation `000000`, the first subset returned is the empty set $\varnothing \subseteq A_3$. Then `index` is incremented to 1, which has the binary representation `000001`, and so the next subset returned is $\{q_1 \xrightarrow{2} q_1\} \subseteq A_3$. This process continues until all 64 subsets have been iterated.

Program Results

The results of this program are quite positive. For each $n \in \{2, 3, 4, 5, 6\}$, it has been verified that there exists an n-state minimal submachine of E_n for which the minimal equivalent DFA has d states, for any $d \in (n, 2^n]$. The program's inability to produce a case when $d = n$ has shed light on a limitation of E_n. In particular, no minimal n-state submachine of E_n has an n-state minimal equivalent DFA. This is because $\delta(q_n, 1) = \varnothing$ in E_n, which introduces the empty state set $\varnothing$ in the equivalent DFA E'_n. Removing transitions from E_n to form submachines cannot change this fact, and so any n-state submachine will have a minimal equivalent DFA with at least $n + 1$ states.

Although the program's results suggest the existence of general methods (for any n) to construct submachines of E_n to reach any $d \in (n, 2^n]$, currently none are known to produce any $d \in (n, 2^{n-1}]$. Nonetheless, it is enough to suggest a conjecture with which we conclude this thesis.

Conjecture 22. *Let $k \in \mathbb{N}$ such that $2^{n-1} \leq k < 2^n - n$. Then there exists an n-state submachine of E_n for which the minimal equivalent DFA has $2^n - k$ states.*

Bibliography

[1] Viliam Geffert, "State hierarchy for one-way finite automata," *Journal of Automata, Languages, and Combinatorics* 12, pp. 139-145 (2007).

[2] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Pearson 2007).

[3] John E. Hopcroft, "An $n \log n$ algorithm for minimizing the states in a finite automaton," in Z. Kohavi (ed.) *The Theory of Machines and Computations*, Academic Press, New York, 1971, pp. 189-196.

[4] Kazuo Iwama, Yahiko Kambayashi, and Kazuya Takaki, "Tight bounds on the number of states of DFAs that are equivalent to n-state NFAs," *Theor. Comp. Sci* 237, pp. 485-494, 2000.

[5] Kazuo Iwama, Akihiro Matsuura, and Mike Paterson, "A family of NFA's which need $2^n - \alpha$ deterministic states," *Theoretical Computer Science*, 301, pp. 451-462, 2003.

[6] Jozef Jirásek, Galina Jirásková, and Alexander Szabari, "Deterministic blow-ups of minimal nondeterministic finite automata over a fixed alphabet," *Int. J. Found. Comp. Sci.*, 19, pp. 617-631, 2008.

[7] Galina Jirásková, "Deterministic blow-ups of minimal NFA's," *RAIRO Inform. Theor. Appl.* 40, pp. 485-499, 2006.

[8] Harry R. Lewis and Christos H. Papadimitriou, *Elements of the Theory of Computation* (Prentice-Hall 1981).

[9] Anders Møller, `dk.brics.automaton` Java package, `http://www.brics.dk/automaton/index.html`.

[10] F. R. Moore, "On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata," *IEEE Trans. Comput.* C-20, pp. 1211-1214, 1971.

[11] M. O. Rabin and D. Scott, "Finite automata and their decision problems," *IBM J. Res. Develop.* 3, pp. 114-125, 1959.

[12] M.O. Rabin, "Mathematical theory of automata," *Proceedings of Symposia in Applied Mathematics*, J. T. Schwartz, ed., American Mathematical Society, 1967, vol. 19.

[13] Michael Sipser, *Introduction to the Theory of Computation* (Thomson 2006).

Appendix A

An Analysis of E_4

The purpose of this appendix is to show that the NFA E_4 is exploding, as well as provide a demonstration of the algorithm suggested in Theorem 13 that, given any $k \in \{0, 1, \ldots, 7\}$, constructs a 4-state minimal submachine of E_4 for which the minimal equivalent DFA has $2^4 - k = 16 - k$ states. Figure A.1 contains the NFA E_4.

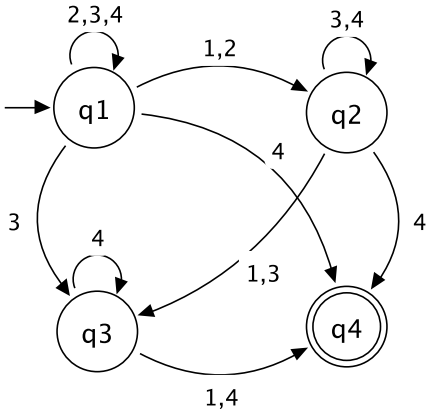

Figure A.1: The NFA E_4.

Notice that E_4 contains the chain of transitions $q_1 \xrightarrow{1} \ldots \xrightarrow{1} q_4$, and it contains no transition from q_4 on a 1. Then Lemma 15 guarantees that E_4 is minimal and that the states of its equivalent DFA E_4' are pairwise distinguishable. Thus, E_4', with all inaccessible states removed, is a minimal DFA by Lemma 7. So, to verify that E_4 is exploding, we must simply show that E_4' has $2^4 = 16$ accessible states. This can be done by running the subset algorithm, as described in Section 1.1.5, on E_4 to obtain E_4'. Figure A.2 contains the transition diagram of E_4' with 16 states, all of which are accessible from the start state set $\{q_1\}$.

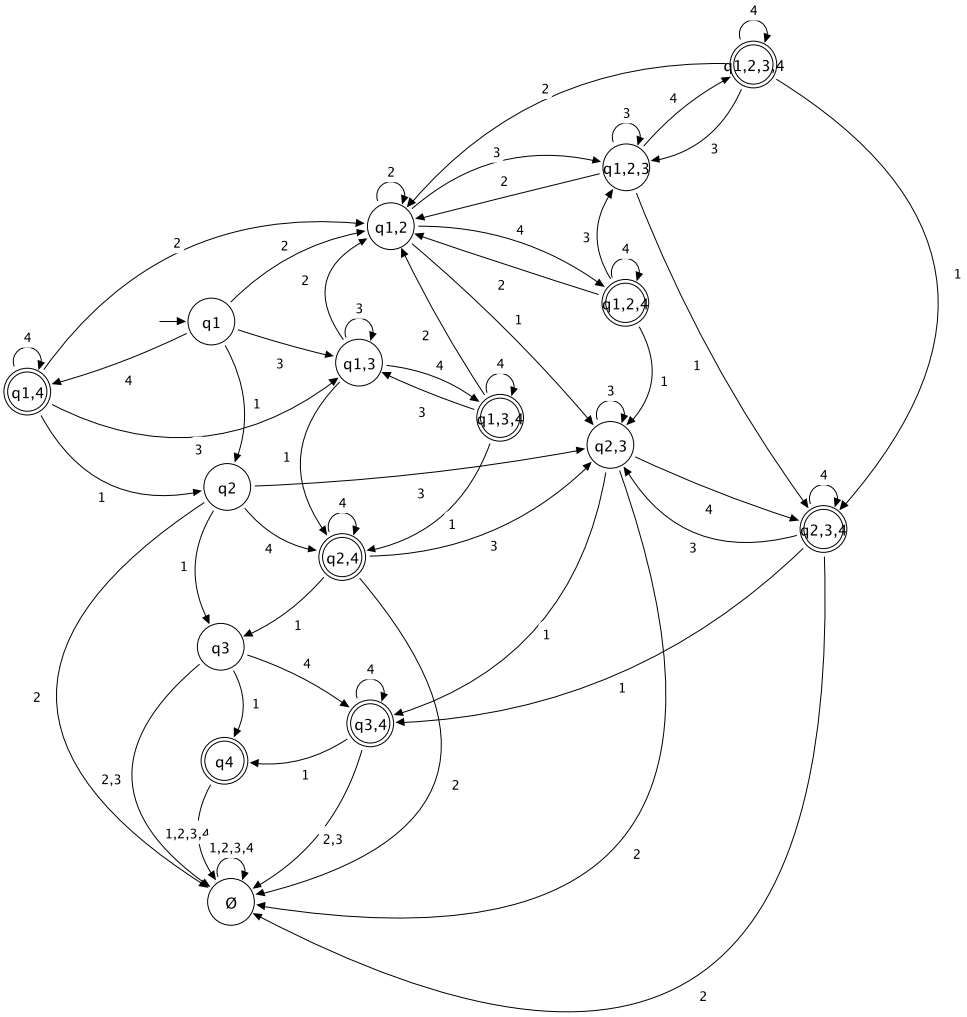

Figure A.2: The DFA E_4'.

Suppose we are asked to create a 4-state minimal submachine of E_4 for which the minimal equivalent DFA has 13 states. Since $13 = 16 - 3 = 2^4 - 3$, we set $k = 3$, which has the binary representation

$$k = 3 = 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0.$$

The binary representation of k tells us exactly which submachine of E_4 to use to answer the question. Specifically, each binary digit corresponds to a transition from the state q_1 in E_4, and the value of the digit serves as an indicator of whether its corresponding transition should be removed. In general, the least significant digit of k corresponds to the transition $q_1 \xrightarrow{n} q_n$, the next digit corresponds to the transition $q_1 \xrightarrow{n-1} q_{n-1}$, and so forth.

In this case, k has the binary representation `011`. Its least significant digit, `1`, indicates that we should remove the transition $q_1 \xrightarrow{4} q_4$ from E_4. The next digit, also `1`, indicates that we should remove the transition $q_1 \xrightarrow{3} q_3$ from E_4. The most significant digit, `0`, indicates that we should leave the transition $q_1 \xrightarrow{2} q_2$ alone. Hence the appropriate submachine to answer the question is

$$M = E_4 \setminus \{q_1 \xrightarrow{4} q_4, q_1 \xrightarrow{3} q_3\}.$$

Figure A.3 contains the transition diagram of the submachine M.

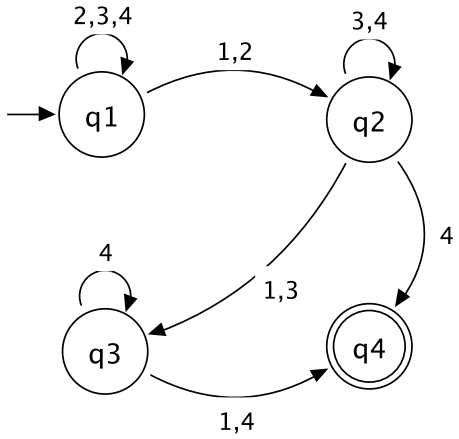

Figure A.3: The submachine M of E_4.

Since we have preserved the chain of transitions $q_1 \xrightarrow{1} \ldots \xrightarrow{1} q_4$ in M, Lemma 15 implies that M is minimal and that the states of its equivalent DFA M' are pairwise distinguishable. By Lemma 21, M', with its inaccessible states removed, has

$$2^4 - 2^{4-4} - 2^{4-3} = 16 - 1 - 2 = 13$$

states. Thus M answers the question. To obtain a closer view of how this process works, we use the subset algorithm to obtain M'. Figure A.4 contains the transition diagram of M' with 16 states, not all of which are accessible.

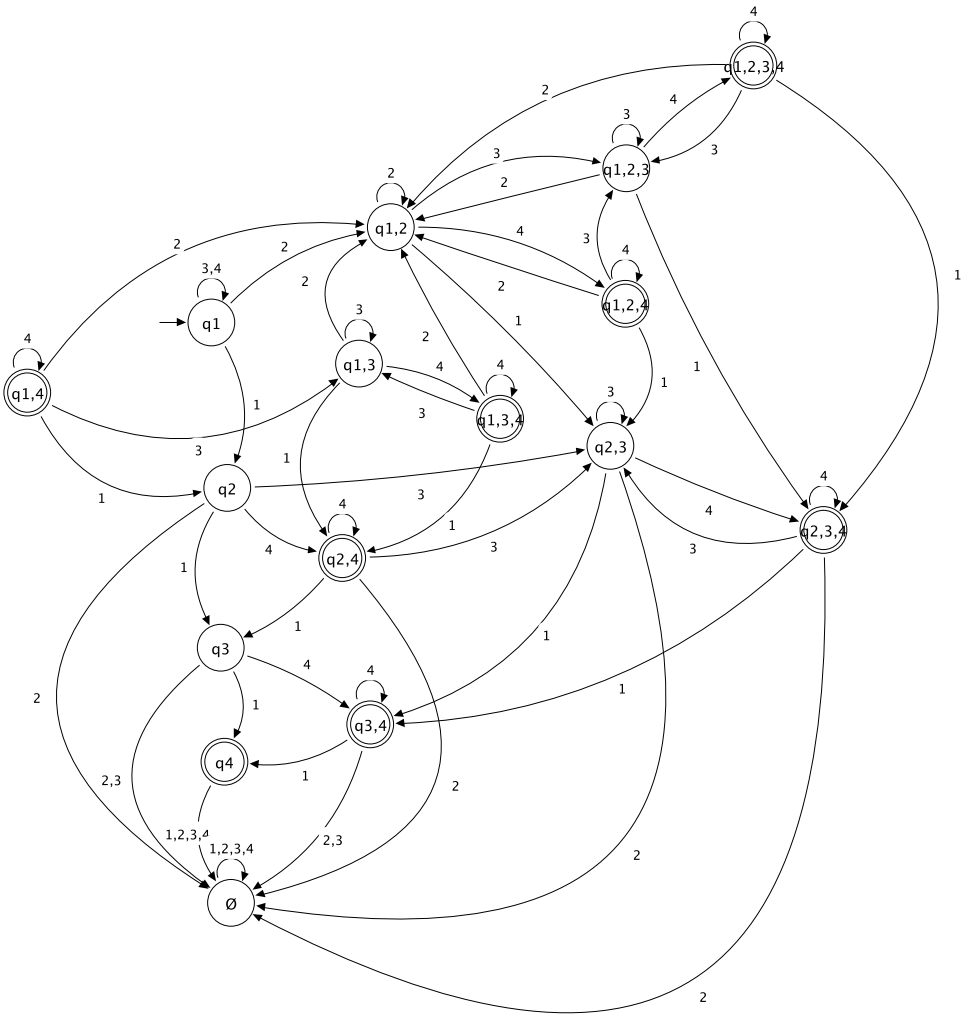

Figure A.4: The DFA M' with inaccessible states included.

Notice that the transition diagram of M' is nearly identical to that of E'_4, except for the subtle changes introduced by the removals of the NFA transitions $q_1 \xrightarrow{4} q_4$ and $q_1 \xrightarrow{3} q_3$. These removals have eliminated the paths from $\{q_1\}$ to $\{q_1, q_4\}$ and from $\{q_1\}$ to $\{q_1, q_3\}$ in the equivalent DFA. The elimination of the former only renders the state set $\{q_1, q_4\}$ inaccessible in M', as the other state sets to which it points can be accessed in some other way (for example, the state set $\{q_2\}$ can be reached by way of the transition $\{q_1\} \xrightarrow{1} \{q_2\}$). The elimination of the latter is a bit stronger, in that it renders not only $\{q_1, q_3\}$ inaccessible but also $\{q_1, q_3, q_4\}$ (notice that the only transition pointing to $\{q_1, q_3, q_4\}$ is from $\{q_1, q_3\}$).

Hence removing the transitions $q_1 \xrightarrow{4} q_4$ and $q_1 \xrightarrow{3} q_3$ to form M has made the state sets $\{q_1, q_4\}$, $\{q_1, q_3\}$, and $\{q_1, q_3, q_4\}$ inaccessible in M'. If we prune these inaccessible state sets from the transition diagram, we are left with $16 - 3 = 13$ accessible state sets. Figure A.5 contains the transition diagram of the final result.

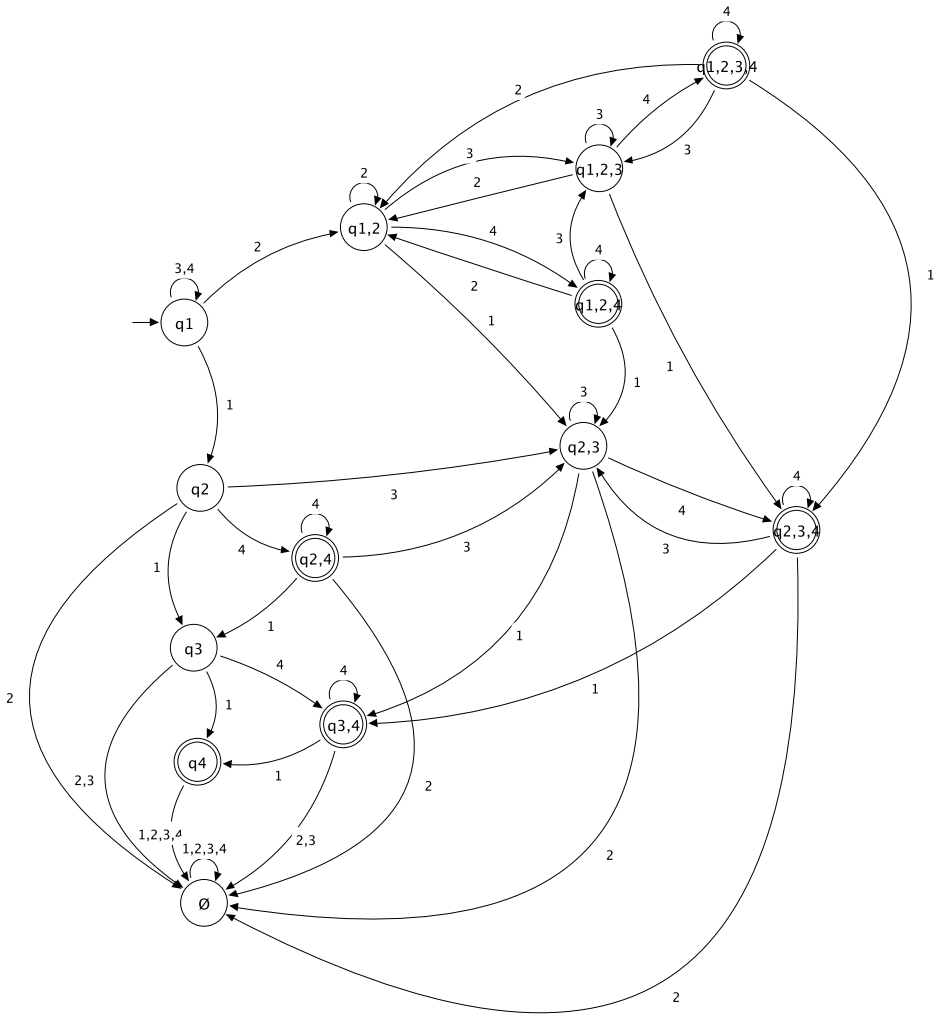

Figure A.5: The minimal DFA M'.

Appendix B

Code Used

B.1 StateRangeScanner

```
package fa.testing;

import java.util.ArrayList;
import java.util.List;

import fa.ExplodingAutomaton;
import fa.Submachine;
import fa.TransitionEntry;
import fa.util.PowersetIterator;

public class StateRangeScanner
{
        private static final String USAGE =
                "Usage: java StateRangeScanner <n>";

        public static void main(String[] args) {
                int n = 0;
                if (args.length == 1) {
                        try {
                                n = Integer.parseInt(args[0]);
                        } catch (NumberFormatException e) {
                                System.err.println(USAGE);
                                System.exit(-1);
```

```
                }
            }
            else {
                System.err.println(USAGE);
                System.exit(-1);
            }
            scanStateRange(n);
        }

        private static void scanStateRange(int numStates) {
            ExplodingAutomaton ex = new ExplodingAutomaton(numStates);
            ex.build();
            List<TransitionEntry> transitions =
                new ArrayList<TransitionEntry>();
            for (TransitionEntry t : ex.getTransitions()) {
                if (t.sigma() != 1)
                    transitions.add(t);
            }

            long[] hierarchy = new long[(1 << numStates) + 1];
            for (int i = 0; i < hierarchy.length; i++)
                hierarchy[i] = 0;

            PowersetIterator<TransitionEntry> powerset =
                new PowersetIterator<TransitionEntry>(transitions);
            for (List<TransitionEntry> removals : powerset) {
                Submachine m = new Submachine(numStates, removals);
                m.build();
                ++hierarchy[m.numDeterministicStates() + 1];
            }

            System.out.printf("[# of DFA states |");
            System.out.printf(" # of occurrences]");
            System.out.println();
            for (int i = 1; i < hierarchy.length; i++)
                System.out.printf("[%15d | %15d ]\n", i, hierarchy[i]);
            System.out.println();
        }
}
```

B.2 ExplodingAutomaton

```
package fa;

import java.util.ArrayList;
import java.util.List;

import dk.brics.automaton.Automaton;
import dk.brics.automaton.State;
import dk.brics.automaton.Transition;

public class ExplodingAutomaton extends Automaton
{
        private State[] q;
        private List<TransitionEntry> transitions;
        private int numStates;

        public ExplodingAutomaton(int n) {
                numStates = n;
        }

        public void build() {
                transitions = new ArrayList<TransitionEntry>();
                createStates();
                createTransitions();
        }

        public int numDeterministicStates() {
                if (!super.isDeterministic())
                        super.minimize();
                return super.getNumberOfStates();
        }

        public List<TransitionEntry> getTransitions() {
                return transitions;
        }

        protected void createStates() {
                q = new State[numStates + 1];
                for (int i = 1; i <= numStates; i++)
                        q[i] = new State();
                super.setDeterministic(false);
                super.setInitialState(q[1]);
                q[numStates].setAccept(true);
```

```
        }

        protected void createTransitions() {
                for (int i = 1; i <= numStates; i++) {
                        if (i < numStates)
                                addTransition(i, 1, i+1);
                        for (int j = i+1; j <= numStates; j++) {
                                addTransition(i, j, i);
                                addTransition(i, j, j);
                        }
                }
        }

        protected void addTransition(int q1, int s, int q2) {
                transitions.add(new TransitionEntry(q1, s, q2));
                q[q1].addTransition(
                                new Transition(Character.forDigit(s, 10), q[q2]));
        }
}
```

B.3 TransitionEntry

```
package fa;

public class TransitionEntry
{
        private final int sourceState, sigma, destState;

        public TransitionEntry(int q1, int s, int q2) {
                sourceState = q1;
                sigma = s;
                destState = q2;
        }

        public boolean equals(Object o) {
                if (o instanceof TransitionEntry) {
                        TransitionEntry t = (TransitionEntry) o;
                        return equals(t.sourceState, t.sigma, t.destState);
                }
                return false;
        }
```

```
        public boolean equals(int q1, int s, int q2) {
                return (sourceState==q1) && (sigma==s) && (destState==q2);
        }

        public String toString() {
                return String.format("(q%d,%d,q%d)",
                                sourceState, sigma, destState);
        }

        public int source() {
                return sourceState;
        }

        public int sigma() {
                return sigma;
        }

        public int dest() {
                return destState;
        }
}
```

B.4 Submachine

```
package fa;

import java.util.List;

public class Submachine extends ExplodingAutomaton
{
        private List<TransitionEntry> removals;

        public Submachine(int n, List<TransitionEntry> r) {
                super(n);
                removals = r;
        }

        protected void addTransition(int q1, int s, int q2) {
                TransitionEntry delta = new TransitionEntry(q1, s, q2);
                if (!removals.contains(delta))
                        super.addTransition(q1, s, q2);
        }
```

```
}
```

B.5 PowersetIterator

```
package fa.util;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class PowersetIterator<T> implements Iterator<List<T>>, Iterable<List<T>>
{
        private List<T> set;
        private BigInteger index, numElements;

        public PowersetIterator(List<T> s) {
                set = s;
                index = BigInteger.ZERO;
                numElements = BigInteger.ONE.shiftLeft(set.size());
        }

        public Iterator<List<T>> iterator() {
                return this;
        }

        public boolean hasNext() {
                return index.compareTo(numElements) < 0;
        }

        public List<T> next() {
                List<T> ret = new ArrayList<T>(index.bitLength());
                for (int i = 0; i < set.size(); i++) {
                        if (index.testBit(i))
                                ret.add(set.get(i));
                }
                index = index.add(BigInteger.ONE);
                return ret;
        }

        public void remove() { }
}
```