Algebraic Approaches to State Complexity of Regular Operations

by

Sylvie Davies

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Pure Mathematics

Waterloo, Ontario, Canada, 2019

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Jean-Éric Pin Director of Research Institut de Recherche en Informatique Fondamentale Université Paris-Diderot
Supervisors:	Jason Bell Professor Department of Pure Mathematics University of Waterloo
	Janusz Brzozowski Distinguished Professor Emeritus David R. Cheriton School of Computer Science University of Waterloo
Internal Members:	Barbara Csima Professor Department of Pure Mathematics University of Waterloo
	Kevin Hare Professor Department of Pure Mathematics University of Waterloo
Internal-External:	Jeffrey Shallit Professor David R. Cheriton School of Computer Science University of Waterloo

Abstract

The state complexity of operations on regular languages is an active area of research in theoretical computer science. Through connections with algebra, particularly the theory of semigroups and monoids, many problems in this area can be simplified or completely reduced to combinatorial problems. We describe various algebraic techniques for attacking state complexity problems. We present a general method for constructing witness languages for operations – languages that attain the worst-case state complexity when used as the argument(s) of the operation. Our construction is based on full transformation monoids, which contain all functions from a finite set into itself. When a witness for an operation is known, determining the state complexity essentially becomes a counting problem.

These counting problems, however, are not necessarily easy, and the witness languages produced by this method are not ideal in the sense that they have extremely large alphabets. We thus investigate some commonly used operations in detail, and look for algebraic techniques to simplify the combinatorial side of state complexity problems and to simplify the search for small-alphabet witnesses. For boolean operations (e.g., union, intersection, difference) we show that these combinatorial problems can be solved easily in special cases by studying the subgroup of permutations in the syntactic monoid of a witness candidate. If the subgroup of permutations is known to have some strong transitivity property, such as primitivity or 2-transitivity, we can draw conclusions about the worst-case state complexity when this language is used in a boolean operation. For the operations of concatenation and Kleene star (an iterated version of concatenation), we describe a "construction set" method to simplify state complexity lower-bound proofs, and determine some algebraic conditions under which this method can be applied. For the reversal operation, we show that the state complexity of the reverse of a language is closely related to the syntactic monoid of the language, and use this fact to investigate a generalized version of the reversal state complexity problem.

After describing our techniques, we demonstrate them by applying them to some classical state complexity problems. We obtain complex generalizations of the classical results that would be difficult to prove without the machinery we develop.

Acknowledgements

Much of the work in this thesis would have been impossible without the aid of the GAP computer algebra system [38] and the GAP Automata package [27]. This thesis was financially supported by the Natural Sciences and Engineering Research Council of Canada grant No. OGP0000871.

On a more personal note: I thank my supervisors Jason Bell and Janusz Brzozowski for their support and guidance throughout my PhD. I thank my family doctor Dana Quinn and my former and current therapists Susan Sargeant and Tracy Morgan for helping me deal with depression and anxiety. I thank Cathy, Mardsen and Ceri for their lifetime of support. I thank Chantelle for never forgetting about me. I thank Delphine for their gentle demeanor and interesting conversations. I thank Hubol for his caring personality, his sense of humor, and his musical talent. I thank Katherine for an uncountable number of things. I thank Suzanne for befriending me when I felt completely alone, and eating tasty food with me. I miss you! I thank the members of the University of Waterloo Game Development Club for giving me a place to belong. I thank everyone else who has shown me kindness, and I apologize if I omitted your name. The kindness of others makes life worth living.

Dedication

This thesis is dedicated to my supervisor and mentor, Janusz Brzozowski, with whom I have worked since my undergraduate years. He taught me how to do research and write papers, and was always kind and supportive. Even when I failed him numerous times, he was patient and lenient with me. I cannot overstate his importance to my development as a researcher and a person. I will miss working with him greatly.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 What is State Complexity?

This thesis concerns *regular languages* and *finite automata*, two classes of objects whose study is traditionally within the purview of computer science. Finite automata model computers with a finite amount of memory, which execute simple "recognizer" programs: given an input, determine whether the input is "valid" or "invalid". Regular languages are the sets of inputs that are considered "valid" by some such finite-memory recognizer. Not all sets are regular; some may require unbounded memory to recognize, and some may even be impossible for any sort of computer to recognize, such as Turing's famous "halting set" consisting of all computer programs that do not get stuck in an infinite loop when executed on a given input. The development of finite automata and regular languages coincides with the development of early electronic computers, and these objects have many practical applications. Nonetheless, they also have rich mathematical properties and enjoy deep connections to algebra, topology and logic. This work gives only a small glimpse of the pure-mathematical side of the theory of languages and automata; we consider connections to the theory of monoids and groups, and use these connections to attack a rather small and particular class of problems. We are interested in the *deterministic state complexity of operations on regular languages.*

Many kinds of finite automata have been defined, but we are concerned mostly with two classical models: *deterministic* and *nondeterministic* finite automata. In both the deterministic and nondeterministic models, the finite memory of the automaton is divided into *states*, and there are rules for *transitioning* between states. The input is given as a *word* – a finite-length sequence of *letters* – and the automaton reads one letter at a time.

After each letter is read, the automaton then chooses a transition rule, which must match the current state and the letter that was read, and transitions to a new state (or it may stay in the same state, depending on the rule). If there is exactly one choice of transition rule for each state-letter pair, the automaton is called *deterministic*. More general automata, which do not necessarily meet this condition, are called *nondeterministic*. In a nondeterministic finite automaton, some state-letter pairs may have multiple choices of rule, or no choice of rule.

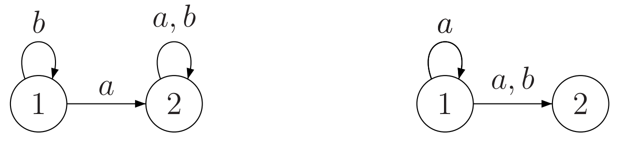


Figure 1.1.1: Deterministic automaton (left) and nondeterministic automaton (right).

Figure 1.1.1 illustrates the difference between determinism and nondeterminism. The automata are represented as directed graphs, where the nodes are states and the edges, labelled with letters, correspond to transition rules. For example, an edge of the form $1 \xrightarrow{a} 2$ means that from state 1, if the letter a is read, the automaton should transition to state 2. Notice that in the right automaton, if the state is 1 and a letter a is read, there are two choices for the next transition: the automaton may remain in state 1, or transition to state 2. If the state is 2 and a letter b is read, there are no choices for the next transition. Thus the right automaton is nondeterministic. However, in the left automaton, there is always exactly one choice of transition corresponding to each state-letter pair – the left automaton is deterministic.

As mentioned, finite automata correspond to finite-memory "recognizer" programs, which decide whether input words are "valid" or "invalid", and a set of words is called a *regular language* if it is the set of "valid" words of a recognizer. The exact semantics of language recognition by automata will be discussed later; for now, it is sufficient to know that this correspondence exists. With this connection between languages and automata established, we may ask the following question: given a regular language, what is the least amount of memory needed to recognize it? That is, what is the *minimal number of states* needed to recognize the language with a finite automaton? If we restrict our attention to deterministic finite automata, this minimal number of states is called the *deterministic state complexity* of the language. By considering the more general nondeterministic automata, we may also define the *nondeterministic state complexity* of a regular language; however, this thesis is concerned only with deterministic state complexity. We will generally refer to deterministic state complexity as simply *state complexity*, with no further qualifiers.

State complexity of regular languages is an active area of research, part of the more general study of *descriptional complexity of formal languages*. Roughly speaking, descriptional complexity is concerned with the size of various representations or models of formal languages, including but not limited to automata and other sorts of "recognizers". A 2016 survey on state complexity by Gao, Moreira, Reis and Yu [37] identifies two general types of results in descriptional complexity. The first type are *representational* or *transformational* results; these concern the change in size caused by conversions between different models. For example, if we convert a nondeterministic finite automaton to a deterministic finite automaton, how much extra memory (that is, how many new states) does the deterministic automaton need? The other type are *operational* results; these concern the change in model size when an operation is performed on a language or on multiple languages. For example, how large can the state complexity of the union of two regular languages be compared to the state complexities of the original languages?

It should now be clear what it means when we say we are interested in deterministic state complexity of operations on regular languages – we are studying operational descriptional complexity, using deterministic state complexity as our measure of choice, and focusing specifically on regular languages as opposed to more general sets of words. Despite this laser-sharp focus – on one type of descriptional complexity result, on one particular measure, on one particular class of languages (and its subclasses) – there is a wealth of interesting problems in this area.

An overview of early work on (deterministic) state complexity can be found in the introductory sections of [5]. Papers dealing with "representational" state complexity issues appeared as early as the 1960s, but to our knowledge, the first paper on operational state complexity was by Maslov, published in 1970 [62]. Maslov's paper studied the operational state complexity of union, concatenation, Kleene star, and a few other operations on regular languages; however, some proofs were omitted and the result for Kleene star contained a small error. Furthermore, the paper was published in a Russian journal and was overlooked by Western researchers for many years. In 1994, Yu, Zhuang and Salomaa rediscovered and proved several of Maslov's results [79]. Their paper sparked a great deal of interest in state complexity, and many papers on the subject followed.

The papers of Maslov and Yu, Zhuang and Salomaa left open many interesting directions for future research. One direction was to study the complexity of other operations these authors did not consider. Many operations have been considered, including other binary boolean operations [5] and boolean operations of higher arity [35], cyclic shift [55], shuffle [13, 23] and more generally shuffles on trajectories [32], square and higher powers [31], the root operation [58, 59], and proportional removals [29]. There are also many papers on *combined operations* formed by composing operations, such as "star of union" [73] and "star-complement-star" [45, 52].

Another direction was to reduce the alphabet size of the *witness languages* used for particular operations – the languages which attain the maximal possible state complexity for a particular operation. For example, Yu, Zhuang and Salomaa provided witnesses for concatenation over a three-letter alphabet, and proved that the maximal state complexity cannot be reached with a one-letter alphabet, but left it open whether two letters are sufficient. Jiŕasková was later able to find witnesses for concatenation over a two-letter alphabet [46]. (In fact, Maslov had already provided a two-letter witness in 1970, but did not give a proof, and as mentioned this paper was overlooked for some time.)

Yet another direction was to consider the *range* of state complexities that may result from an operation, as opposed to simply the worst-case (largest possible) value. Numbers which represent gaps in the range of possible complexities are called *magic numbers* [39], and so these range problems are referred to as *magic number problems*. Operations that have been studied include union and intersection [44], reversal [74], star [47, 51], and concatenation [48, 53]; in all of these cases there are no magic numbers, meaning the range of complexities for each operation is a contiguous interval of integers.

One of the most fruitful directions was to consider operational state complexity in *proper subclasses* of the regular languages. For example, all finite languages are regular, but not all regular languages are finite; thus finite languages form a proper subclass of the regular languages. Thus we can ask questions such as: how large can the state complexity of the union of two *finite* languages be compared to the state complexity of the original languages? And then as above we may consider additional operations, optimal alphabet sizes, magic number problems, and so forth in this new context of finite languages. A survey of previously studied subclasses can be found in [8]. Aside from finite languages [22], subclasses that have been studied in a state complexity context include unary (one-letter alphabet) and finite unary languages [78], star-free languages [15], non-returning languages [9, 34], left ideal, right ideal, two-sided ideal, and all-sided ideal languages [10, 11, 19], prefix-free languages [18, 42, 49, 57], suffix-free languages [16, 20, 25, 41, 50], bifix-free, factor-free and subword-free languages [12, 36], prefix-closed, suffix-closed, factor-closed and subword-closed languages [14, 16, 18, 42], proper prefix-convex languages [17], and proper suffix-convex languages [75].

The goal of this thesis is to demonstrate the power of an algebraic viewpoint in solving state complexity problems. It is common to introduce automata as a type of directed graph with labelled edges (as we did above). This is a very intuitive representation of automata, widely used in introductory courses and lectures for non-specialists, but even in more advanced contexts it is often useful. However, this view of automata leads one to think of the behaviour of automata in a *local* sense. That is, one thinks of the transitions of an automaton as *edges* leading from *one particular state to another*. We believe that for state complexity problems, this point of view is not ideal and can even obscure understanding. It is better to think of a deterministic finite automaton as specifying an *action of a monoid on the set of states*. The action is defined in a *global* way: for each letter which may appear in an input word, one considers *all transitions labelled with that letter* and how these transitions affect the set of states *as a whole*. We will see that this small shift in viewpoint leads to new insights and a stronger understanding of the principles underlying state complexity of operations.

The remainder of the thesis is structured as follows. Chapter 2 covers the basic definitions, notations, and results that will be needed in subsequent chapters. Chapter 3 describes a general technique for finding witness languages which maximize the state complexity of particular operations, called the "One Letter Per Action" approach. This approach is the cornerstone of the thesis – it connects state complexity with algebra, reduces many state complexity problems to combinatorics, and illustrates the basic principles behind solving state complexity problems.

The next three chapters look at the state complexity problems for particular operations in greater detail, and describe specialized techniques for solving them more easily. We focus on the most widely used and studied operations. Chapter 4 discusses boolean operations, such as union, intersection and set difference, and connects these operations with the theory of transitive and primitive permutation groups. Chapter 5 covers the concatenation and Kleene star operations, focusing on techniques for solving reachability problems. Chapter 6 looks at a generalization of the reversal state complexity problem to a different machine model called *deterministic finite automata with output*.

At the end of Chapter 2, we take a look at the basic state complexity problems for the operations mentioned above. In the final chapter, Chapter 7, we revisit these problems and see how our new techniques enabled them to be solved more simply, and enable us to obtain far more general theorems. We also discuss possible avenues for future work in this area.

Chapter 2

Preliminaries

2.1 Notation and Conventions

Below, we summarize the notation and mathematical conventions used in this thesis. The remaining sections of this chapter define the concepts mentioned in detail and prove some basic results on them. We recommend reading this summary in full before proceeding to later chapters; the other sections can be read as necessary.

Results given without attribution in this chapter are generally either elementary or part of the "folklore" of the relevant field. None of the results presented in this chapter are originally due to the thesis author.

The natural numbers, denoted by $\mathbb{N}$, contain zero.

Singleton sets are identified with the elements they contain; we often write x instead of $\{x\}$.

The power set of a set X, the set of all subsets of X, is denoted $\mathcal{P}(X)$.

The complement of a set $S \subseteq X$ is the set $X \setminus S$ and is denoted $\overline{S}$. The set relative to which we are taking the complement will always be clear from the context.

Binary relations between X and Y are subsets of $X \times Y$. A relation ρ can be viewed as an *element-to-set map* $\rho\colon X \to \mathcal{P}(Y)$, or as a *set-to-set map* $\rho\colon \mathcal{P}(X) \to \mathcal{P}(Y)$ by applying the element-to-set map to each element of the argument set.

Maps are generally written to the *right* of their arguments: we write $(x)\rho$ or simply $x\rho$, rather than $\rho(x)$, for the image of x under ρ.

Composition of binary relations is denoted by *juxtaposition* and performed from *left to right*. The composition $\rho\tau$ of $\rho \subseteq X \times Y$ and $\tau \subseteq Y \times Z$ is defined as the composition of the element-to-set map $\rho\colon X \to \mathcal{P}(Y)$ with the set-to-set map $\tau\colon \mathcal{P}(Y) \to \mathcal{P}(Z)$.

Functions are treated as a special case of binary relations, and follow the same rules: we write xfg to mean "apply f to x, then apply g to the result", as opposed to something like $(g \circ f)(x)$.

Equivalence relations: note that if ρ is an equivalence relation on X, then the image $x\rho$ is precisely the equivalence class of $x \in X$. The set of equivalence classes is denoted X/ρ.

Function equality: consider the functions $f\colon \mathbb{N} \to \mathbb{N}$ and $g\colon \mathbb{N} \to \mathbb{Z}$ defined by $nf = ng = n + 1$. Are they equal, because they are equal as relations, or are they distinct, because they have different codomains? It turns out one's viewpoint on this issue affects one of our most important definitions! So, we introduce terminology for the two opposing viewpoints. We say that a person who considers the functions distinct has the *restricted viewpoint* on function equality, because their conditions for function equality are more restrictive. A person who considers the functions equal has the *unrestricted viewpoint*. For most of the thesis, either the viewpoint is irrelevant or we use the restricted viewpoint, but we will occasionally explore the consequences of the unrestricted viewpoint.

Transformations of X are functions $t\colon X \to X$. The identity transformation is denoted id. The transformation $(S \to x)$ sends every element of S to x, and fixes all elements of $X \setminus S$. The transformation $({}_i^j x \to xt)$, where $t\colon X \to X$, sends x to xt if $i \leq x \leq j$ and otherwise fixes x; in this context, usually X is $\{1, 2, \ldots, n\}$ and t is $x \mapsto x+1$ or $x \mapsto x-1$, with arithmetic performed modulo n. The *full transformation monoid* on X, is denoted T_X, or T_n if $X = \{1, 2, \ldots, n\}$. A *transformation monoid* on X is a submonoid of T_X and $|X|$ is called the *degree*.

Permutations are bijective transformations and can be described using *cycle notation*. Let $K = \{x_1, \ldots, x_k\} \subseteq X$; the permutation of X that maps x_i to x_{i+1} for $i < k$, maps x_k to x_1, and fixes $x \in X \setminus K$ is denoted $(x_1, \ldots, x_k)$ and is called a *k-cycle*. A 2-cycle is also called a *transposition*. The *symmetric group* on X is denoted S_X or S_n if $X = \{1, 2, \ldots, n\}$. A *permutation group* on X is a subgroup of S_X and is a special type of transformation monoid. The alternating group is denoted A_X or A_n.

Monoids and groups: when working with arbitrary monoids and groups, usually the operation is denoted by juxtaposition and the identity element is denoted e. A submonoid or subgroup of a monoid M must contain the identity element of M; it cannot have a different identity. We write $G \leq M$ to mean that G is a subgroup of M. For a group G and $S \subseteq G$, the group generated by S is denoted $\langle S \rangle$. We write $M \cong M'$ to mean that monoids M and M' are isomorphic.

Actions: a monoid action of M on X can be defined in two ways: as a function $\psi\colon X \times M \to X$ or as a family of transformations $\psi_m\colon X \to X$ for each $m \in M$. Instead of writing $(x, m)\psi$ or $x\psi_m$ we usually just write xm. Often we do not specify a symbol such as ψ for the action at all and just say, "let M act on X". We use the term *group action* when M is a group. A transformation monoid on X has a *natural action* which just applies the given transformation to the given element of X.

Kernels and quotients: the kernel of a monoid homomorphism is a congruence, but the kernel of a group homomorphism is a normal subgroup. We write $\ker\varphi$ for the kernel of the homomorphism φ. If ρ is a congruence on a monoid M, we write M/ρ for the quotient monoid. If N is a normal subgroup of a group G, we write G/N for the quotient group.

Finite fields of order p^k are denoted $\mathbb{F}_{p^k}$. The field $\mathbb{F}_p$ is the integers modulo p. We may explicitly construct $\mathbb{F}_{p^k}$ as a quotient $\mathbb{F}_p[x]/\langle f\rangle$, where f is an irreducible polynomial of degree k over $\mathbb{F}_p$ and (f) is the ideal generated by f.

Alphabets, words, languages: alphabets are finite sets whose elements are called *letters*. Words over an alphabet are finite in length, and the length of a word w is denoted $|w|$. The set of all words over an alphabet Σ is denoted Σ^*. The empty word is denoted ε. If $L \subseteq \Sigma^*$, we say L *has alphabet* Σ. Formally, a word over Σ of length n is a function from $\{1, \ldots, n\}$ into Σ; thus under the *restricted viewpoint*, two words over different alphabets are never equal, and every language has a unique alphabet. Under the *unrestricted viewpoint*, words over different alphabets can be equal, and a language can have multiple alphabets; however every language has a unique alphabet of minimal size called the *minimal alphabet*.

Regular expressions use $\cup$ as the union symbol, as opposed to $+$ or $|$ like some authors. The star operator has the highest precedence and union has the lowest. For example, the expression $aa^* \cup b$ means $(a(a^*)) \cup b$.

Finite semiautomata (FSAs), or simply *semiautomata*, are triples (Q, Σ, T) where Q is a finite set whose elements are called *states*, Σ is an alphabet, and $T \subseteq (Q\times\Sigma)\times Q$ is a binary relation whose elements are called *transitions*. For elements of T we write (q, a, q') instead of $((q, a), q')$. The binary relation T extends to a monoid action $T\colon \mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$ of Σ^* on $\mathcal{P}(Q)$, and so we can use our usual notation for monoid actions and write $ST_w = S'$ or simply $Sw = S'$ instead of $(S, w)T = S'$. We also write $S \xrightarrow{w} S'$ to mean $Sw = S'$. If T is a function, then the semiautomaton is called *deterministic*; we write DFSA for deterministic finite semiautomaton. In the deterministic case, we can extend T to a monoid action $T\colon Q \times \Sigma^* \to Q$ of Σ^* on Q, and the actions of words are transformations of Q; these word actions form a transformation monoid on Q called the *transition monoid* of the DFSA, which acts naturally on Q. This interpretation of FSAs and DFSAs as essentially being a way to specify *monoid actions on finite sets* is fundamental to this thesis.

Finite automata (FAs), or simply *automata*, are quintuples (Q, Σ, T, I, F) where (Q, Σ, T) is an FSA, $I \subseteq Q$ is a set of *initial states* and $F \subseteq Q$ is a set of *final states*. The triple (Q, I, F) is called the *state configuration* of the FA, and it is *deterministic* if $|I| = 1$. An FA is *deterministic* if the semiautomaton (Q, Σ, T) and the state configuration (Q, I, F) are both deterministic; we write DFA for deterministic finite automaton. Usually the state set Q is assumed to be a finite set of integers such as $\{1, 2, \ldots, n\}$ and the unique initial state is taken to be 1 without loss of generality. All definitions for semiautomata are inherited by automata, e.g., a DFA has a transition monoid which is equal to the transition monoid of the underlying DFSA. If $\mathcal{A}$ is an FA, its language is denoted $L(\mathcal{A})$. The indistinguishability relation on states of a DFA with respect to $S \subseteq Q$ (defined in Section 2.4.1) is denoted Δ_S; if $S = F$ we simply write Δ rather than Δ_F.

Reachability and accessibility: as usual, a state of an automaton is called *reachable* if there is a path to it from an initial state in the underlying state graph. Automata in which all states are reachable are called *accessible*.

State complexity: the state complexity of a regular language L is denoted $\operatorname{sc}(L)$. The state complexity of an operation Φ on regular languages is denoted $\operatorname{sc}(\Phi)$. The state complexity of an operation with two or more operands is affected by one's choice of restricted or unrestricted viewpoint. Under the *restricted viewpoint* we assume that all operands as well as the result share the same alphabet. Under the *unrestricted viewpoint*, we make no such assumption, but we compute the state complexity of the operands and the result with respect to their *minimal alphabets*.

Concatenation automata: we use some slightly unorthodox notation for automata representing the concatenation of two languages. Applying the usual construction to two DFAs with state sets Q and Q' respectively, we obtain a DFA whose states are subsets of $\mathcal{P}(Q \cup Q')$ which contain at most one state of Q. We write these special subsets as *ordered pairs* (X, Y), where X is a subset of Q with at most one element, and Y is an arbitrary subset of Q'. In other words, we treat the state set as a subset of $(Q \cup \{\emptyset\}) \times \mathcal{P}(Q')$.

2.2 Relations and Functions

Binary relations. A *binary relation* ρ between X and Y is a subset of $X \times Y$. We often write $x \,\rho\, y$ to mean $(x, y) \in \rho$. When $X = Y$, we simply refer to ρ as a *binary relation on* X.

If $\rho \subseteq X \times Y$ and $\tau \subseteq Y \times Z$, the *composition* of ρ and τ is the relation

$$\rho\tau = \{(x, z) \in X \times Z : \text{ there exists } y \in Y \text{ such that } (x, y) \in \rho \text{ and } (y, z) \in \tau\}.$$

For $x \in X$ and $\rho \subseteq X \times Y$, the *image* of x under ρ is the set $x\rho = \{y \in Y : (x, y) \in \rho\}$. For $x \notin X$ we define $x\rho = \emptyset$.

The *converse* of a binary relation $\rho \subseteq X \times Y$ is the relation $\rho^{-1} = \{(y, x) : (x, y) \in \rho\} \subseteq Y \times X$. The set $y\rho^{-1} = \{x \in X : (x, y) \in \rho\}$ is called the *preimage* of y under ρ. Elements of this set are called *preimages* of y; for example, if $x \in y\rho^{-1}$ we say that x is a preimage of y.

Relations as maps. If we write $\mathcal{P}(S)$ for the power set of a set S (that is, the set of all subsets of S), then we can view ρ as a map $\rho\colon X \to \mathcal{P}(Y)$. We may also *extend ρ by union* to a map $\rho\colon \mathcal{P}(X) \to \mathcal{P}(Y)$ as follows: for $S \subseteq X$, we define

$$S\rho = \bigcup_{s \in S} s\rho.$$

We thus have two ways to make sense of an expression like $x\rho\tau$: it is the image of x under the composite relation $\rho\tau \subseteq X \times Z$, and it is also the image of the set $x\rho \subseteq Y$ under the map $\tau\colon \mathcal{P}(Y) \to \mathcal{P}(Z)$. Additionally, we have a way to make sense of a composition $\rho\tau\colon X \to \mathcal{P}(Z)$ of maps $\rho\colon X \to \mathcal{P}(Y)$ and $\tau\colon Y \to \mathcal{P}(Z)$: take the composition of the corresponding relations.

Equivalence relations. An *equivalence relation* on X is a binary relation ρ on X with the following properties:

- *Reflexivity*: $(x, x) \in \rho$ for all $x \in X$.
- *Symmetry*: if $(x, x') \in \rho$, then $(x', x) \in \rho$.
- *Transitivity*: if $(x, x') \in \rho$ and $(x', x'') \in \rho$, then $(x, x'') \in \rho$.

The set $x\rho$ is called the *equivalence class* of ρ containing x. The equivalence classes of an equivalence relation on X form a partition of the set X. The set of equivalence classes of X under ρ is denoted X/ρ.

Partial orders. A *partial order* on X is a binary relation $\leq$ on X that is reflexive, transitive, and *antisymmetric*: if x and x' are distinct and $x \leq x'$, then we must not have $x' \leq x$. If $x \leq x'$ or $x' \leq x$, then we say x and x' are *comparable* with respect to the partial order $\leq$. A *total order* on X is a partial order on X such that every pair of elements in $X \times X$ is comparable.

Functions. A *function* $f\colon X \to Y$ is a binary relation $f \subseteq X \times Y$ such that $|xf| = 1$ for all $x \in X$. The set X is the *domain* of the function and the set Y is the *codomain*. Following our notation for binary relations, we write functions to the *right* of their arguments.

Composition of functions is defined by composing the corresponding relations. Thus the order of composition is *left-to-right*; in a composition fg, first f is applied and then g.

The restricted and unrestricted viewpoints. There are two different points of view one can take on equality of functions. We outline them here, as some of our later definitions will be significantly affected by one's point of view. The simplest and least restrictive point of view is that two functions are equal if they are equal as sets of ordered pairs. We call this the *unrestricted viewpoint* on function equality. The *restricted viewpoint* also requires that the domain and codomain of the two functions are equal. For example, consider the functions $f\colon \mathbb{N} \to \mathbb{N}$ and $g\colon \mathbb{N} \to \mathbb{Z}$ defined by $nf = ng = n+1$. These functions are equal under the unrestricted viewpoint, since the sets $f \subseteq \mathbb{N} \times \mathbb{N}$ and $g \subseteq \mathbb{N} \times \mathbb{Z}$ contain the same elements, but they are not equal under the restricted viewpoint, since the codomain $\mathbb{N}$ of f is not the same as codomain $\mathbb{Z}$ of g.

Transformations and permutations. A *transformation* of a set X is a function $t\colon X \to X$, that is, a function from X into itself. We say t is a *permutation* of X if $Xt = X$. We say t *acts as a permutation* on $S \subseteq X$ if $St = S$. If t acts as a permutation on S, then every element of S has at least one preimage under t, that is, for all $s \in S$, the set $st^{-1} = \{x \in X : xt = s\}$ is non-empty.

A *cyclic permutation* of a set $\{x_1, \dots, x_k\} \subseteq X$ is a permutation p such that $x_i p = x_{i+1}$ for $1 \leq i < k$, $x_k p = x_1$, and $xp = x$ for all $x \in X \setminus \{x_1, \dots, x_k\}$. We denote such a permutation as $(x_1, \dots, x_k)$, and we call it a *cycle of length k* or simply *k-cycle*. A cyclic permutation of a two-element set is also called a *transposition*. It is well known that every permutation can be expressed as a product of cyclic permutations; in fact, a product of transpositions suffices. A permutation is called *even* if it can be expressed as a product of an even number of transpositions, and *odd* if it cannot.

2.3 Abstract Algebra

2.3.1 Monoids and Groups

Basic definitions. A *monoid* is a set M equipped with an associative binary operation $\cdot$ and an identity element e such that $m \cdot e = e \cdot m = m$ for all $m \in M$. Typically we omit the symbol for the operation; so the previous equation could be written as $me = em = m$. For $n \geq 1$ we write m^n for the n-fold product of m with itself, and define $m^0 = e$ for all $m \in M$. If for each $m \in M$, there exists $m' \in M$ such that $mm' = m'm = e$, then M is called a *group*, and m' is called the *inverse* of m and denoted m^{-1}. A group with a commutative binary operation is called an *abelian group*. The *order* of a monoid M is the size of the set M. The *order* of an *element* m of a monoid is the least integer $n \geq 1$ such that $m^n = e$, if such an integer exists; otherwise the element is said to have infinite order.

Submonoids and subgroups. A *submonoid* of M is a subset $M' \subseteq M$ which is closed under $\cdot$ and contains the identity e of M. If additionally M' is a group, it is called a *subgroup* of M; we write $M' \leq M$ to mean that M' is a subgroup of M. Note that we do not allow submonoids or subgroups of M to have an identity element different from that of M. If $x_1, \ldots, x_k$ are elements of a group G, then $\langle x_1, \ldots, x_k \rangle$ denotes the *group generated by* $x_1, \ldots, x_k$, the smallest subgroup of G containing $x_1, \ldots, x_k$. A group generated by a single element is called a *cyclic* group.

Homomorphisms. Let M and M' be monoids with identity elements e and e' respectively. A *homomorphism* from M to M' is a function $\varphi \colon M \to M'$ such that $(mm')\varphi = (m)\varphi(m')\varphi$ for all $m, m' \in M$ and $e\varphi = e'$. A bijective homomorphism is called an *isomorphism*, and two monoids are said to be *isomorphic* if there exists an isomorphism from one to the other. We write $M \cong M'$ to mean that M and M' are isomorphic.

Congruences and quotients. An equivalence relation ρ on M is a *congruence* if $m \,\rho\, m'$ and $n \,\rho\, n'$ implies $mn \,\rho\, m'n'$ for all $m, m', n, n' \in M$. If ρ is a congruence, then whenever $m\rho = m'\rho$ and $n\rho = n'\rho$, we have $(mn)\rho = (m'n')\rho$. This means we can extend the operation of the monoid to equivalence classes in a well-defined way: for $m, m' \in M$, define $(m\rho)(m'\rho) = (mm')\rho$. Thus M/ρ, the set of equivalence classes of M under ρ, becomes a monoid when equipped with this equivalence class product. This monoid is called the *quotient monoid* of M by ρ.

Kernels and the First Isomorphism Theorem. If M and M' are monoids and $\varphi \colon M \to M'$ is a homomorphism, the *kernel* of φ is the relation $\ker \varphi = \{(m, m') : m\varphi = m'\varphi\}$. The kernel of a homomorphism is always a congruence, and the converse also holds: every congruence of a monoid is the kernel of a homomorphism. Thus the quotient monoid $M/\ker \varphi$ can be formed, and in fact it is isomorphic to $M\varphi$, the image of φ. This is called the *first isomorphism theorem*.

If G and G' are groups and $\varphi \colon G \to G'$ is a homomorphism, the kernel of φ is defined differently: it is the set $\ker \varphi = \{g \in G : g\varphi = e'\}$, that is, the set of elements of G that map to the identity of G'. These two definitions are related as follows: the "group-kernel" of a homomorphism from G to G' is precisely the equivalence class of the "monoid-kernel" of the homomorphism that contains the identity element of G. For groups, the full set of equivalence classes of the "monoid-kernel" is completely determined by just the equivalence class of the identity element.

Normal subgroups. If G is a group, $N \leq G$, and $gng^{-1} \in N$ for all $g \in G$ and $n \in N$, we say N is a *normal subgroup* of G. A group G is *simple* if it has no non-trivial proper normal subgroups, that is, the only normal subgroups of G are G itself and the trivial group (containing just the identity element of G). The kernel of a homomorphism from G to another group is always a normal subgroup of G, and conversely every normal subgroup of G arises as a kernel of a homomorphism of groups. Occasionally we use the following elementary facts about normal subgroups and homomorphisms.

Proposition 2.3.1. *If $\varphi\colon G \to G'$ is a homomorphism of groups and $\ker\varphi$ is the trivial one-element subgroup of G, then φ is injective.*

Proof. Let e be the identity of G and e' the identity of G'. Fix $g, h \in G$ with $g \neq h$. If $g\varphi = h\varphi$ then $(g\varphi)(h\varphi)^{-1} = e'$. Thus $(gh^{-1})\varphi = e'$ and it follows that $gh^{-1} = e$, since a homomorphism must map identity elements to identity elements. Thus $g = h$, which is a contradiction; so $g\varphi \neq h\varphi$ and it follows that φ is injective. $\square$

Proposition 2.3.2. *If $\varphi\colon G \to G'$ is a* surjective *homomorphism of groups and N is a normal subgroup of G, then $N\varphi$ is a normal subgroup of G'.*

Proof. We leave it as an exercise to prove the following fact: if H is a subgroup of G, and $\varphi\colon G \to G'$ is a group homomorphism, then $H\varphi$ is a subgroup of G'. Thus $N\varphi$ is a subgroup of G'; it remains to prove that it is normal.

Fix $g' \in G'$ and $n' \in N\varphi$. Since φ is surjective, there exists $g \in G$ with $g\varphi = g'$. Also, since $n' \in N\varphi$, there exists $n \in N$ with $n\varphi = n'$. Since N is normal in G, we have $gng^{-1} \in N$. Thus $(gng^{-1})\varphi = g'n'(g')^{-1} \in N\varphi$, and it follows that $N\varphi$ is normal in G'. $\square$

Quotient groups and cosets. The notion of a *quotient group* is usually defined in terms of normal subgroups, rather than congruences. There is no fundamental difference mathematically, since there are correspondences between normal subgroups and kernels and between kernels and congruences. The quotient of a group G by a normal subgroup N, denoted G/N, is the set of *left cosets* $gN = \{gn : n \in N\}$ for $g \in G$, with the operation on cosets defined by $(gN)(g'N) = (gg')N$. *Right cosets* may be defined analogously and produce an isomorphic group.

2.3.2 Actions, Transitivity and Primitivity

Actions. A *monoid action* of M on a set X is a function $\psi\colon X \times M \to X$ such that $((x, m)\psi, m')\psi = (x, mm')\psi$ and $(x, e)\psi = x$ for all $m, m' \in M$ and $x \in X$. Equivalently, it is a family of functions $\psi_m \colon X \to X$ such that $\psi_m \psi_{m'} = \psi_{mm'}$ for all $m, m' \in M$ and ψ_e is the identity map on X. The map ψ_m is called the *action of* m. To simplify the notation, we often omit the action symbol ψ and just write xm instead of $x\psi_m$ or $(x, m)\psi$. Furthermore, we often avoid assigning a symbol to the action at all; rather than "let ψ be a monoid action of M on X" we write "let M be a monoid acting on X", meaning that M has a specific but nameless action on X associated with it. If M is a group, we use the term *group action* rather than monoid action.

The following fact is useful for defining monoid and group actions.

Proposition 2.3.3. *If $S \subseteq M$ generates the monoid M, a monoid action ψ is completely determined by its values on elements of S.*

Proof. Let $\psi \colon X \times S \to X$ be an arbitrary function; we show that ψ has a unique extension to a monoid action $\psi \colon X \times M \to X$.

To see that an extension exists, fix $m \in M$. Since S generates M, we can write $m = s_1 s_2 \cdots s_k$ for some elements $s_1, \dots, s_k \in S$. We define the extension $\psi \colon X \times M \to X$ by induction on k. If $k = 0$, then m is the "empty product" which is by convention defined to be the identity element e; we set $(x, e)\psi = x$. If $k = 1$ then set $(x, m)\psi = (x, s_1)\psi$. If $k > 1$ then set $(x, m)\psi = ((x, s_1)\psi, s_2 \cdots s_k)\psi$. To see that this is a monoid action, note that if $s, s' \in S$ then $\psi_{ss'} = \psi_s \psi_{s'}$. Now suppose $m = s_1 \cdots s_k$ and $m' = s'_1 \cdots s'_\ell$. Then we have

$$x\psi_{mm'} = x\psi_{s_1 \cdots s_k s'_1 \cdots s'_\ell} = x\psi_{s_1 \cdots s_k} \psi_{s'_1 \cdots s'_\ell} = x\psi_m \psi_{m'}.$$

Thus $(x, mm')\psi = ((x, m)\psi, m')\psi$. Since we also have $(x, e)\psi = x$, it follows that ψ is a monoid action.

To see that this extension is unique, let $\psi' \colon X \times M \to X$ be a monoid action such that $(x, s)\psi' = (x, s)\psi$ for all $x \in X$ and $s \in S$. Then $\psi_s = \psi'_s$ for all $s \in S$. Thus we have

$$(x, m)\psi = x\psi_m = x\psi_{s_1} \cdots \psi_{s_k} = x\psi'_{s_1} \cdots \psi'_{s_k} = x\psi'_m = (x, m)\psi'.$$

Hence $\psi = \psi'$. ☐

Stabilizers. Let G be a group acting on X. For $x \in X$, the *stabilizer subgroup* or simply *stabilizer* of x is the subgroup $\{g \in G : xg = x\}$ of G. For $S \subseteq X$, the *setwise stabilizer* of S is the subgroup $\{g \in G : Sg = S\}$. Elements of the setwise stabilizer need not fix every element of S; for example, if $1g = 2$ and $2g = 1$ then g is in the setwise stabilizer of $\{1, 2\}$.

Transformation monoids and the natural action. Let X be a finite set. Recall that a function $t \colon X \to X$ is called a *transformation* of X. The set of all transformations of X

is a monoid under composition called the *full transformation monoid* T_X. A submonoid of T_X is called a *transformation monoid* on X. The *degree* of a transformation monoid on X is the size of X. If M is a transformation monoid on X, the monoid action $\psi\colon X \times M \to X$ given by $(x, t)\psi = xt$ for $x \in X$, $t \in M$ is called the *natural action* of M. If $X = \{1, \ldots, n\}$ we write T_n for T_X.

Permutation groups, symmetric and alternating groups. Recall that a bijective transformation of X is called a *permutation* of X. The set of all permutations of X is a subgroup of T_X called the *symmetric group* S_X. A subgroup of S_X is called a *permutation group* on X; this is a special type of transformation monoid and we have the same notions of degree and natural action. The *alternating group* A_X is the subgroup of S_X consisting of all permutations that can be expressed as a product of an *even number of transpositions* (cycles of length two). If $X = \{1, \ldots, n\}$ we write S_n for S_X and A_n for A_X.

Congruences of actions. Separate from the notion of monoid congruences, there is a notion of congruence for *monoid actions* specifically. A *congruence* of a monoid action of M on X is an equivalence relation on X that is *M-invariant* in the following sense: if E is an equivalence class of the congruence, then for all $m \in M$, there exists an equivalence class E' of the congruence such that $Em \subseteq E'$. In other words, if x and x' are equivalent, then xm and $x'm$ are equivalent for all $m \in M$. The *equality congruence* $\{(x, x) : x \in X\}$ in which elements are equivalent only if they are equal, and the *full congruence* $X \times X$ in which all elements are equivalent, are called *trivial congruences*. If M is a transformation monoid on X, a congruence of the natural action is called an *M-congruence*.

We now introduce the notions of transitivity and primitivity, which are used heavily in Chapter 4 but seldom appear outside of it. The reader may wish to skip this material until they arrive at Chapter 4.

Transitivity. Let G be a group acting on X. We say that the action of G is *transitive* or that G *acts transitively* on X if for all $x, x' \in X$, there exists $g \in G$ such that $xg = x'$. We say the action of G is *k-transitive* or G *acts k-transitively* on X if for all pairs of k-tuples $(x_1, \ldots, x_k), (x'_1, \ldots, x'_k) \in X^k$, there exists $g \in G$ such that for $1 \leq i \leq k$ we have $x_i g = x'_i$; informally, k-transitive means "transitive on k-tuples".

Blocks and primitivity. A non-empty set $B \subseteq X$ is called a *block* for G if for all $g \in G$, either $Bg \cap B = B$ (equivalently, $Bg = B$) or $Bg \cap B = \emptyset$. A block B is *trivial* if it is a singleton or the entire set X. We say the action of G is *primitive* or that G *acts primitively* on X if it is transitive and all of its blocks are trivial. Equivalently, a transitive group action of G is primitive if for every set $S \subsetneq X$ with at least two elements, there exists $g \in G$ such that $\emptyset \subsetneq Sg \cap S \subsetneq S$.

Primitivity, block systems and congruences. There is an important alternate characterization of primitivity in terms of congruences. In the case of a permutation group G on X, notice that for all $S \subseteq X$ and $g \in G$, the set Sg has the same size as S. Hence a G-congruence has the following property: if E is an equivalence class, then for all $g \in G$, the set Eg is also an equivalence class. In particular, we either have $E \cap Eg = E$ or $E \cap Eg = \emptyset$ for all $g \in G$; thus the classes of G-congruences are blocks.

In fact, if G is transitive, then every G-congruence arises from the blocks of G as follows. If B is a block for G, the *block system* corresponding to B is the set $\{Bg : g \in G\}$. As the name implies, each set in a block system is also a block for G. Indeed, for all $g' \in G$, we either have $Bgg' \cap Bg = \emptyset$ or $Bgg' \cap Bg \neq \emptyset$, and in the latter case, $Bgg'g^{-1} \cap B \neq \emptyset$. But B is a block, so this implies $Bgg'g^{-1} = B$ and thus $Bgg' = Bg$. Thus every set in a block system is a block, so in particular, all distinct sets in a block system are pairwise disjoint. Furthermore, since G is transitive, each element of X appears in at least one block of the system. It follows that block systems are partitions of X, and thus equivalence relations on X. It is easy to see that block systems are G-invariant, and thus are G-congruences.

Thus every block gives rise to a block system that is a G-congruence, and every G-congruence consists of blocks; it follows block systems and G-congruences are one and the same if G is transitive. If all G-congruences are trivial, then all block systems of G consist only of trivial blocks, and vice versa. Thus we obtain our alternate characterization of primitivity:

Proposition 2.3.4. *A transitive permutation group G on X is primitive if and only if all G-congruences are trivial.*

Transitive and primitive groups. If G is a permutation group and the natural action of G is transitive (k-transitive, primitive), then we say G is a *transitive group* (*k-transitive group*, *primitive group*). For example, the cyclic group $\langle(1, 2, 3, 4)\rangle \leq S_4$ is a transitive group, since its natural action on $\{1, 2, 3, 4\}$ is transitive. This terminology can cause confusion, since transitivity, k-transitivity and primitivity are properties of *actions* and not groups; statements like "G is transitive" or "G is primitive" are statements about a particular action of G (the natural action) rather than the abstract group itself. In particular, these properties are not preserved under isomorphism; for example, the group $\langle(5, 6, 7, 8)\rangle \leq S_8$ is not transitive, but it is isomorphic to the transitive group $\langle(1, 2, 3, 4)\rangle \leq S_4$.

The following fact is frequently useful:

Proposition 2.3.5. *If H is a subgroup of G and H is transitive (primitive), then G is also transitive (primitive).*

Proof. Suppose H acts transitively on X. Then for all $x, x' \in X$, there exists $h \in H$ such that $xh = x'$. Since H is a subgroup of G, it follows that for all $x, x' \in X$, there exists $g \in G$ such that $xg = x'$ (taking g to be the appropriate $h \in H$). Thus G acts transitively on X.

Suppose H is primitive. Then H is transitive, so G is transitive by the above argument. It remains to show that all blocks of G are trivial. Suppose B is a block for G. Then for all $g \in G$, we either have $Bg \cap B = B$ or $Bg \cap B = \emptyset$. Since H is a subgroup of G, it follows that for all $h \in H$, we either have $Bh \cap B = B$ or $Bh \cap B = \emptyset$. Thus B is a block for H. But H is primitive, so B is a trivial block. Thus G is primitive, since all of its blocks are trivial. $\square$

2.3.3 Examples of Transitive and Primitive Groups

Since the notions of transitivity and primitivity are central to Chapter 4, we give a number of examples to aid with understanding.

Example 2.3.6 (An imprimitive cyclic group)**.** Consider the group $G = \langle(1, 2, 3, 4, 5, 6)\rangle \leq S_6$. This group is clearly transitive, since its natural action on $\{1, 2, 3, 4, 5, 6\}$ is transitive. However, it is imprimitive, since $\{1, 3, 5\}$ and $\{2, 4, 6\}$ are non-trivial blocks. Indeed, if we let $a = (1, 2, 3, 4, 5, 6)$, then $\{1, 3, 5\}a = \{2, 4, 6\}$ and $\{2, 4, 6\}a = \{3, 5, 1\}$. Hence for all $k \geq 0$, we either have $\{1, 3, 5\}a^k \cap \{1, 3, 5\} = \emptyset$ or $\{1, 3, 5\}a^k \cap \{1, 3, 5\} = \{1, 3, 5\}$, and similarly for $\{2, 4, 6\}$. One may also verify that $\{1, 4\}$, $\{2, 5\}$ and $\{3, 6\}$ are non-trivial blocks, and that there are no blocks of size 4 or 5.

How do the block systems we have found correspond to non-trivial G-congruences? Put an equivalence relation ρ on $X = \{1, \ldots, 6\}$ by writing $i \,\rho\, j$ if i and j have the same parity (odd or even). Notice that for $i \in X$, the elements i and ia have opposite parity. Thus ρ is a G-congruence, since if $i \,\rho\, j$ then $ia \,\rho\, ja$, and so for each equivalence class $i\rho$, the set $(i\rho)a = (ia)\rho$ is also an equivalence class. In fact, the classes of ρ are just the blocks $\{1, 3, 5\}$ and $\{2, 4, 6\}$; thus the G-congruence ρ corresponds to the block system $\{\{1, 3, 5\}, \{2, 4, 6\}\}$. If we define an equivalence relation by $i \,\rho\, j$ if i and j are equivalent modulo 3, we obtain a non-trival G-congruence corresponding to the block system $\{\{1, 4\}, \{2, 5\}, \{3, 6\}\}$. As for the trivial G-congruences, the equality congruence corresponds to the block system $\{\{1\}, \{2\}, \ldots, \{6\}\}$ containing the singletons, and the full congruence corresponds to the block system $\{\{1, \ldots, 6\}\}$ that just contains X. ■

Example 2.3.7 (A primitive cyclic group)**.** Consider the group $G = \langle(1, 2, 3, 4, 5)\rangle \leq S_5$. This group is clearly transitive, and it is also primitive. To see this, suppose for a contradiction that B is a non-trivial block. Let $a = (1,2,3,4,5)$ and let $k = |b - b'|$, where b and b' are distinct elements of B. Then $Ba^k \cap B \neq \emptyset$, so we must have $Ba^k = B$ since B is a block. Thus for each $i \in B$, we have $ia^k \in Ba^k$, and thus $ia^k \in B$. Then since $ia^k \in B$, we have $ia^{2k} \in Ba^k$, and thus $ia^{2k} \in B$. By induction it follows that $\{ia^{nk} : n \geq 0\} \subseteq B$. We claim $\{ia^{nk} : n \geq 0\} = \{1,2,3,4,5\}$, which contradicts the fact that B is a *non-trivial* block. Indeed, for $j \in \{1,2,3,4,5\}$, we have $ia^{nk} = j$ if and only if $i + nk \equiv j \pmod 5$. Since 5 is prime and $0 < k < 5$, we see that k is coprime with 5. Hence by elementary number theory, there exists n such that $nk \equiv j - i \pmod 5$ and so $i + nk \equiv i + j - i \equiv j \pmod 5$ as required. Hence $j \in \{ia^{nk} : n \geq 0\}$ for all $j \in \{1,2,3,4,5\}$, which proves the claim. It follows G has no non-trivial blocks, and so is primitive.

Using the notion of G-congruences, we can give an alternate, simpler proof that this group is primitive. Fix a G-congruence on X. By G-invariance, all classes of the G-congruence must have the same size, say m. If the congruence has n classes, then we have $mn = |X| = 5$. So m is either 1 or 5 since 5 is prime, which means the classes are either singletons (giving the equality congruence) or the full set X (giving the full congruence). Thus all G-congruences are trivial, and thus G is primitive. Alternatively, we could make the same argument in terms of block systems, using the fact that all blocks in a system have the same size to show all blocks must be trivial. This argument actually shows that not only are cyclic groups of prime order primitive, but *all transitive groups of prime degree* are primitive (since $|X|$ is the degree of a permutation group on X). ■

Example 2.3.8 (An intransitive subgroup of S_6)**.** Consider $G = \langle (1,2,3), (4,5,6) \rangle \leq S_6$. This group is intransitive, since (for example) it does not contain a permutation mapping 1 to 4. Thus it is imprimitive by definition. Alternatively, observe that $\{1,2,3\}$ and $\{4,5,6\}$ are non-trivial blocks for G.

Generally an intransitive group will always have non-trivial blocks, but there is one exception: the trivial subgroup of S_2 (containing only the identity element). The natural action of this group is clearly not transitive on $\{1,2\}$, but its only blocks are the trivial blocks $\{1\}$, $\{2\}$ and $\{1,2\}$. To avoid dealing with this exceptional case, we require primitive groups to be transitive by definition. ■

The next example shows that we have the following hierarchy of permutation group properties:

$$(\text{2-transitive}) \implies (\text{primitive}) \implies (\text{transitive}).$$

These implications do not reverse. Cyclic groups of composite order give examples of transitive imprimitive groups, while cyclic groups of prime order $p \geq 5$ give examples of primitive, non-2-transitive groups. (For example, the group $\langle (1,2,3,4,5) \rangle \leq S_5$ is not 2-transitive on $\{1,2,3,4,5\}$ since nothing maps the pair $(1,2)$ to the pair $(1,3)$.)

Example 2.3.9 (A two-transitive group)**.** The alternating group A_n is 2-transitive for $n \geq 4$. Indeed, given $i, i', j, j' \in \{1, \ldots, n\}$, the permutation $(i, i')(j, j')$ is the product of an even number of 2-cycles, and it maps the pair (i, j) to (i', j'). We claim A_n is also primitive for $n \geq 2$. To see this, first note that A_n is a cyclic group of prime order for $2 \leq n \leq 3$. For $n \geq 4$, suppose for a contradiction that B is a non-trivial block. Then B has at least two elements i and j, but B is not all of $\{1, \ldots, n\}$. Choose $k \in \{1, \ldots, n\} \setminus B$. Since A_n is 2-transitive, there exists an element $g \in A_n$ which maps the pair (i, j) to (j, k). Then $Bg \cap B \neq \emptyset$ (since Bg and B contain j), and thus $Bg \cap B = Bg = B$ since B is a block. But Bg contains k and B does not, which is a contradiction. Thus all blocks of A_n are trivial, and thus A_n is primitive. In fact, this argument shows that all 2-transitive groups are primitive. ■

By Proposition 2.3.5, the symmetric group S_n is primitive for $n \geq 2$, since it contains the primitive group A_n. We could also show the symmetric group is primitive directly by mimicking the argument above.

So far, we have mostly only looked at cyclic groups and the symmetric and alternating groups. For our last pair of examples, we consider two subgroups of S_6 that are a little more interesting.

Example 2.3.10 (A transitive, imprimitive subgroup of S_6)**.** Define $a = (2, 4, 6)$, $b = (1, 5)(2, 4)$ and $c = (1, 4, 5, 2)(3, 6)$, and let $G = \langle a, b, c \rangle$. We claim this group is transitive on $\{1, \ldots, 6\}$. For $g \in G$ and $i, j \in \{1, \ldots, 6\}$, we write $i \xrightarrow{g} j$ to mean $ig = j$. Observe that

$$1 \xrightarrow{c^3} 2 \xrightarrow{a^2} 6 \xrightarrow{c} 3, \quad 1 \xrightarrow{c} 4 \xrightarrow{c} 5.$$

Thus for each $i \neq 1$, there is some group element that maps 1 to i. If $g \in G$ maps 1 to i, then g^{-1} maps i to 1. It follows for each i, j, there is some element x that maps i to 1, and another element y that maps 1 to j, giving

$$i \xrightarrow{x} 1 \xrightarrow{y} j.$$

Thus G is transitive. It is also imprimitive, with non-trivial blocks $\{1, 3, 5\}$ and $\{2, 4, 6\}$. Indeed, we see that

$$\{1, 3, 5\} \xrightarrow{a} \{1, 3, 5\}, \ \{1, 3, 5\} \xrightarrow{b} \{5, 3, 1\}, \ \{1, 3, 5\} \xrightarrow{c} \{4, 6, 2\}.$$

Hence these sets are non-trivial blocks. ■

Example 2.3.11 (A primitive subgroup of S_6)**.** Define permutations $a = (1, 2, 3, 4, 6)$ and $b = (1, 2)(3, 4)(5, 6)$ and let $G = \langle a, b\rangle$. It is easy to see that this group is transitive on $\{1, \dots, 6\}$: just verify that 1 can be mapped to every other element and use the argument from the previous example. This group is also primitive. To see this, first note that the subgroup $\langle a\rangle$ acts primitively on $\{1, 2, 3, 4, 6\}$, since it is a cyclic group of prime order. Hence a non-trivial block of G cannot be a subset of $\{1, 2, 3, 4, 6\}$, so in particular a non-trivial block of G must contain 5. Suppose B is a non-trivial block that contains 5; then $Ba \cap B$ contains 5 and hence $Ba \cap B = Ba = B$. Since B is non-trivial, it contains some element $i \neq 5$, and since $Ba = B$ we have $\{i, ia, ia^2, ia^3, ia^4\} = \{1, 2, 3, 4, 6\} \subseteq B$. This implies $B = \{1, 2, 3, 4, 5, 6\}$, and so B is trivial, which is a contradiction. Thus all blocks of G are trivial, and thus G is primitive. ■

Examples 2.3.8, 2.3.10 and 2.3.11 show that a primitive group such as S_6 may contain intransitive subgroups, transitive but imprimitive subgroups, and primitive subgroups.

2.3.4 Finite Fields

The following background material on finite fields is needed only for a single construction in Chapter 4. We omit many proofs throughout this section; proofs can be found in abstract algebra textbooks such as [33].

Basic definitions. A *field* is a set F equipped with two binary operations $+$ and $\cdot$ (*addition* and *multiplication*) and two distinguished elements $0, 1 \in F$, satisfying the following *field axioms*:

1. F is an abelian group under $+$ with identity 0. That is:
   (a) $+$ is associative and commutative.
   (b) For all $a \in F$ we have $a + 0 = a$.
   (c) Each $a \in F$ has an *additive inverse* $-a \in F$ such that $a + (-a) = 0$.
2. F is an abelian group under $\cdot$ with identity 1. That is:
   (a) $\cdot$ is associative and commutative.
   (b) For all $a \in F$ we have $a \cdot 1 = a$.
   (c) Each $a \in F$ has a *multiplicative inverse* $a^{-1} \in F$ such that $a \cdot a^{-1} = 1$.
3. $+$ and $\cdot$ satisfy the *distributive law*: for all $a, b, c \in F$, we have $a \cdot (b + c) = a \cdot b + a \cdot c$.

As for groups and monoids, we typically write ab rather than $a \cdot b$. The group F with operation $+$ is called the *additive group* of the field, and 0 is called the *additive identity* of the field. Similarly, F with operation $\cdot$ is called the *multiplicative group* of the field, and 1 is called the *multiplicative identity* of the field. The *order* of the field F is the size of the set F. A *finite field* is a field of finite order.

Bases. Let F and F' be finite fields with $F \subseteq F'$. A *basis* for F' over F is a set $B = \{b_1, \ldots, b_k\} \subseteq F'$ satisfying the following conditions:

- B is a *spanning set* for F', that is, we have $F' = \{a_1b_1 + \cdots + a_kb_k : a_1, \ldots, a_k \in F\}$.
- B is *linearly independent*: for $a_1, \ldots, a_k \in F$, we have $a_1b_1 + \cdots + a_kb_k = 0$ if and only if $a_1 = \cdots = a_k = 0$.

Existence and construction. A finite field of order n exists if and only if $n = p^k$ for some prime p and integer $k \geq 1$. Furthermore, up to isomorphism there is only one field of order p^k. This field is denoted $\mathbb{F}_{p^k}$. The finite field $\mathbb{F}_p$ is simply the integers modulo p with the usual addition and multiplication. We can construct $\mathbb{F}_{p^k}$ for $k > 1$ as a field of equivalence classes of polynomials, as follows.

Let $\mathbb{F}_p[x] = \{a_0 + a_1x + a_2x^2 + \cdots + a_nx^n : a_0, \ldots, a_n \in \mathbb{F}_p, n \geq 0\}$. That is, $\mathbb{F}_p[x]$ is the set of all polynomials with coefficients in $\mathbb{F}_p$. A polynomial $f(x) \in \mathbb{F}_p[x]$ is *reducible* if there exist non-constant polynomials $g(x), h(x) \in \mathbb{F}_p[x]$ such that $f(x) = g(x)h(x)$; otherwise it is *irreducible*. It is a non-trivial fact that $\mathbb{F}_p[x]$ contains an irreducible polynomial of degree k for all $k > 1$. Let $f(x) \in \mathbb{F}_p[x]$ be an irreducible polynomial of degree k.

Define the set $\langle f(x) \rangle = \{f(x)q(x) : q(x) \in \mathbb{F}_p[x]\}$. One may verify that the binary relation ρ on $\mathbb{F}_p[x]$ given by $g(x)\,\rho\,h(x)$ if and only if $g(x) - h(x) \in \langle f(x) \rangle$ is an equivalence relation. Define $\mathbb{F}_p[x]/\langle f(x) \rangle$ to be the set of equivalence classes of $\mathbb{F}_p[x]$ under ρ. We define addition and multiplication of two polynomial equivalence classes $g(x)\rho$ and $h(x)\rho$ in the expected way: $g(x)\rho + h(x)\rho = (g(x) + h(x))\rho$ and $g(x)\rho \cdot h(x)\rho = (g(x)h(x))\rho$. One can show that these operations are well-defined.

We claim that $\mathbb{F}_{p^k} = \mathbb{F}_p[x]/\langle f(x) \rangle$ with operations $+$ and $\cdot$ as defined above, additive identity 0ρ, and multiplicative identity 1ρ is a finite field of order p^k.

In practice, when working with $\mathbb{F}_p[x]/\langle f(x) \rangle$, we generally omit the equivalence relation symbol ρ for convenience and readability. So if x^3 is equivalent to $x + 1$, we would simply write $x^3 = x + 1$. This should not cause confusion, as long as we are clear whether we are working in $\mathbb{F}_p[x]/\langle f(x) \rangle$ versus $\mathbb{F}_p[x]$.

Properties. There are two important properties of finite fields we will use:

- There exists a basis for $\mathbb{F}_{p^k}$ over $\mathbb{F}_p$ with k elements. In particular, if we construct $\mathbb{F}_{p^k}$ as above, the set $\{1, x, x^2, \ldots, x^{k-1}\}$ is a basis.
- The multiplicative group of $\mathbb{F}_{p^k}$ is cyclic.

We now illustrate these concepts and results with an example.

Example 2.3.12 (A finite field of order 8)**.** The polynomial $x^3 + x + 1$ is irreducible in $\mathbb{F}_2[x]$; if it were reducible, it would have a factor of the form $(x - a)$ for $a \in \mathbb{F}_2$, and thus it would have a root in $\mathbb{F}_2$. However, neither 0 nor 1 is a root. It follows that we can construct the finite field $\mathbb{F}_{2^3} = \mathbb{F}_8$ as $\mathbb{F}_2[x]/\langle x^3 + x + 1\rangle$. One can show that $\mathbb{F}_8$ constructed this way is indeed a field; we omit a proof of this, as it is rather tedious to check all the axioms.

Let $\mathbb{F}_2$ denote the field of integers modulo two. Let $f(x) = x^3 + x + 1 \in \mathbb{F}_2[x]$. Note that in $\mathbb{F}_2$, addition and subtraction behave the same; thus we have $x^3 + x + 1 = x^3 - x - 1 = x^3 - (x + 1)$. Let $g(x) = x^3$ and $h(x) = x + 1$. Then $g(x) - h(x) = f(x)$, so $g(x) - h(x) \in \langle f(x)\rangle$. Thus in $\mathbb{F}_8$ we have $x^3 = x + 1$.

From this fact, it follows that every element of $\mathbb{F}_8$ can be represented by a polynomial of degree at most two. To see this, consider an element $a_0 + a_1x + a_2x_2 + \cdots + a_nx^n \in \mathbb{F}_8$ with $a_n \neq 0$ and $n \geq 3$. If $n \geq 3$, we have

$$\begin{aligned} a_0 + a_1x + a_2x_2 + \cdots + a_nx^n &= a_0 + a_1x + a_2x^2 + \cdots + a_nx^{n-3}x^3 \\ &= a_0 + a_1x + a_2x^2 + \cdots + a_nx^{n-3}(x+1) \\ &= a_0 + a_1x + a_2x^2 + \cdots + a_nx^{n-3} + a_nx^{n-2}. \end{aligned}$$

By making the substitution $x^3 = x + 1$, we reduce the degree of the polynomial by two. We can keep making these substitutions until the degree of the polynomial is two or less.

Intuitively, we can think of $\mathbb{F}_8$ as being the set of polynomials over $\mathbb{F}_2$ of degree at most two, with ordinary addition and a special "two-step" multiplication operation: first multiply the polynomials as usual, then apply the rule $x^3 = x + 1$ to reduce the result to a polynomial of degree at most two. To illustrate this, we show that the multiplicative group of $\mathbb{F}_8$ is cyclic and is generated by x.

In addition to the rule $x^3 = x + 1$, we know that addition and subtraction behave the same in $\mathbb{F}_2$, and thus for all polynomials $f(x) \in \mathbb{F}_8$, we have $2f(x) = 0$. For example, $x + x = x(1 + 1) = x(1 - 1) = x0 = 0$. Combining these facts, we calculate:

$$x^3 = x + 1, \quad x^4 = x^2 + x, \quad x^5 = x^3 + x^2 = x^2 + x + 1,$$

$$x^6 = x^3 + x^2 + x = x^2 + x + x + 1 = x^2 + 1, \quad x^7 = x^3 + x = x + 1 + x = 1.$$

We see from these calculations that every polynomial of degree at most two is equivalent to a power of x, and thus every element of $\mathbb{F}_8$ is expressible as a power of x. Thus x is a generator of the multiplicative group of $\mathbb{F}_8$. In general, it is not always true that x generates the multiplicative group of a finite field of order $\mathbb{F}_{p^k}$ for $k > 1$; it depends on the particular irreducible polynomial used to construct the field.

Next, we show that $\mathbb{F}_8$ has order eight. To see this, first observe that no two distinct polynomials $a_0 + a_1x + a_2x^2, b_0 + b_1x + b_2x^2 \in \mathbb{F}_2[x]$ are equivalent in $\mathbb{F}_8$. If they were, we would have

$$(a_0 - b_0) + (a_1 - b_1)x + (a_2 - b_2)x^2 = f(x)q(x)$$

for some $q(x) \in \mathbb{F}_2[x]$. But the left hand side is a non-zero polynomial of degree at most two. The right hand side is either the zero polynomial (if $q(x) = 0$), or it has degree at least three since $f(x)$ has degree three. It follows that there is a one-to-one correspondence between triples $(a, b, c) \in \mathbb{F}_2^3$ and elements $a + bx + cx^2$ of $\mathbb{F}_8$. There are eight distinct triples $(a, b, c) \in \mathbb{F}_2^3$, and thus $\mathbb{F}_8$ has order eight.

Finally, we prove that $B = \{1, x, x^2\}$ is a basis for $\mathbb{F}_8$ over $\mathbb{F}_2$. Clearly B is a spanning set for $\mathbb{F}_8$. To see that it is a basis, suppose it is not linearly independent; then $a+bx+cx^2 = 0$ in $\mathbb{F}_8$ for some $a, b, c \in \mathbb{F}_2$, with a, b, c not all zero. Thus the polynomial $a+bx+cx^2 \in \mathbb{F}_2[x]$ is equivalent to the zero polynomial. Thus in $\mathbb{F}_2[x]$, we have $a + bx + cx^2 - 0 = f(x)q(x)$ for some $q(x)$. Again, this is impossible since $a + bx + cx^2$ is non-zero with degree at most two, while $f(x)q(x)$ is either the zero polynomial or has degree at least three. ■

2.4 Languages and Automata

2.4.1 Basic Concepts

Alphabets, words and languages. An *alphabet* Σ is a finite set whose elements are called *letters*. A *word* over Σ is a finite-length sequence of letters. Formally, a finite-length sequence of letters from Σ is a function from an initial segment $\{1, 2, \ldots, n\}$ of the positive integers into Σ. We write $a_1 \cdots a_n$ for the sequence that sends i to the letter $a_i \in \Sigma$. Two words x and y over Σ can be *concatenated* as follows: if $x = a_1 \cdots a_m$ and $y = b_1 \cdots b_n$, then the concatenation xy is the sequence $a_1 \cdots a_m b_1 \cdots b_n$. The set of all words over Σ forms a monoid under the operation of concatenation, called the *free monoid generated by* Σ and denoted Σ^*. The identity element of this monoid is the *empty word*, denoted ε. A *language* over Σ is a subset of Σ^*.

The restricted and unrestricted viewpoints. Since words are formally defined as functions, the definition of word equality depends on one's viewpoint on function equality (as discussed in Section 2.2). Under the unrestricted viewpoint, words over different alphabets may be equal; for example, the word $acab$ over the alphabet $\{a, b, c\}$ is the same as the word $acab$ over the alphabet $\{a, b, c, d\}$. Under the restricted viewpoint, these words are different.

Semiautomata and automata. A *finite semiautomaton* (FSA) is a triple (Q, Σ, T) where Q is a finite set of *states*, Σ is an alphabet, and $T \subseteq (Q \times \Sigma) \times Q$ is a binary relation called the *transition relation*. Elements of T are called *transitions*. Rather than the cumbersome notation $((q, a), q')$ for transitions, we just write (q, a, q'). Additionally, the notation $q \xrightarrow{a} q'$ means $(q, a, q') \in T$. This notation can be "chained": for example, the expression $q \xrightarrow{a} q' \xrightarrow{b} q''$ means $(q, a, q'), (q', b, q'') \in T$.

A *state configuration* is a triple (Q, I, F) where Q is a finite set of *states*, $I \subseteq Q$ is a set of *initial states*, and $F \subseteq Q$ is a set of *final states*. A 5-tuple $\mathcal{A} = (Q, \Sigma, T, I, F)$ where (Q, Σ, T) is a finite semiautomaton and (Q, I, F) is a state configuration is called a *finite automaton* (FA). If $\mathcal{S} = (Q, \Sigma, T)$ is a finite semiautomaton, we write $\mathcal{S}(I, F)$ for the finite automaton (Q, Σ, T, I, F). We often omit the word "finite" when speaking of finite (semi)automata, since we do not consider any form of "infinite" (semi)automata. All concepts that we define for semiautomata carry over to automata by looking at the "underlying semiautomaton".

The distinction between semiautomata and automata is as follows. Automata are *language recognizers*; the additional information given by the state configuration is enough to associate a unique language with each automaton. We will describe the correspondence between automata and languages shortly. Semiautomata, which do not recognize languages, are used as templates for automata, or used for extra conciseness in contexts where languages are not important or relevant.

Determinism. A semiautomaton (Q, Σ, T) is *deterministic* if T is a function from $Q \times \Sigma$ to Q. A state configuration (Q, I, F) is *deterministic* if $|I| = 1$; in this case the unique initial state is generally taken to be 1. An automaton $\mathcal{A} = (Q, \Sigma, T, I, F)$ is *deterministic* if it is comprised of a deterministic semiautomaton and a deterministic state configuration, that is, if (Q, Σ, T) and (Q, I, F) are both deterministic. We write DFSA for *deterministic finite semiautomaton* and DFA for *deterministic finite automaton*.

Automata as monoid actions. Given an semiautomaton $\mathcal{A} = (Q, \Sigma, T)$, we can view the transition relation as a map $T\colon Q \times \Sigma \to \mathcal{P}(Q)$. We can extend this map to a monoid action $T\colon \mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$ of Σ^* on $\mathcal{P}(Q)$ in two steps. First, extend to $T\colon Q \times \Sigma^* \to \mathcal{P}(Q)$ by induction on words: set $(q, \varepsilon)T = \{q\}$ and for $w = xa$ with $w, x \in \Sigma^*$ and

$a \in \Sigma$, set $(q, w)T = ((q, x)T, a)T$. Then, extend to $T\colon \mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$ by setting $(S, w)T = \bigcup_{q \in S}(q, w)T$. Now we can use our usual notation for monoid actions and write $Sw = S'$ instead of the cumbersome $(S, w)T = S'$. We also sometimes write $S \xrightarrow{w} S'$ to mean $Sw = S'$, as an extension of our notation for transitions.

If we have a *deterministic* semiautomaton, we get an even nicer monoid action. If T is a function, then notice that our extension to words $T\colon Q \times \Sigma^* \to \mathcal{P}(Q)$ actually maps each state to a *singleton* subset of Q. So we can view this extension as a map $T\colon Q \times \Sigma^* \to Q$, and in fact this is an action of Σ^* on Q. Now, recall that an action of Σ^* on Q can be specified in two ways: as a function $T\colon Q \times \Sigma^* \to Q$, or as a family of transformations $T_w\colon Q \to Q$, one for each word $w \in \Sigma^*$, which are called the *actions* of the words $w \in \Sigma^*$. This family of word actions under composition forms a transformation monoid on Q. This monoid is called the *transition monoid* of the DFSA and it acts naturally on Q.

This shows deterministic semiautomata can be viewed as *free monoids acting on finite sets* or as *transformation monoids acting naturally on finite sets*. This point of view is fundamental to our work.

The above definition of the transition monoid is abstract to the point of being nearly incomprehensible, so let us give an more concrete definition that constructs it directly from the transitions of the semiautomaton. Afterwards, we will look at some examples.

In a deterministic semiautomaton, the transition relation T is a function $T\colon Q \times \Sigma \to Q$. This means that for each state-letter pair (q, a) with $q \in Q$ and $a \in \Sigma$, there is exactly one transition $q \xrightarrow{a} q'$. If we view our semiautomaton as a directed graph with labelled edges, where the nodes are states and the labelled edges are transitions, determinism means that each node has exactly one outgoing edge for each letter. Now, let us build some transformations of Q out of the transitions. First we construct *letter actions* $T_a\colon Q \to Q$ for each letter $a \in \Sigma$. The letter action of $a \in \Sigma$ takes an element of q, and produces the unique element q' such that $q \xrightarrow{a} q'$. In terms of the directed graph, the letter action of a tells us, for each node of the graph, which node we will move to if we follow the unique outgoing edge labelled with a. Next we construct *word actions* $T_w\colon Q \to Q$ for each word $w \in \Sigma^*$. The action T_ε of the empty word is just the identity transformation of Q. The action of a non-empty word is given by composing the letter actions for each letter in the word: if $w = a_1 \cdots a_k$, where $a_1, \ldots, a_k \in \Sigma$, then $T_w = T_{a_1} \cdots T_{a_k}$. In terms of the directed graph, the word action of w tells us, for each node, which node we will end up at if we follow the *path* whose edge labels spell out w. Determinism guarantees that this path is unique. The transition monoid of the semiautomaton is just the set $\{T_w : w \in \Sigma^*\}$ of all word actions. Composing word actions behaves just like concatenating words: $T_xT_y = T_{xy}$. When working with letter actions and word actions, it is convenient to just write a or

w instead of T_a or T_w in cases when it is clear from context what the relevant transition relation is.

Example 2.4.1 (A semiautomaton with transition monoid T_2.)**.** Consider the deterministic semiautomaton depicted in Figure 2.4.1. We will show that its transition monoid is T_2, the set of all transformations of $\{1, 2\}$.

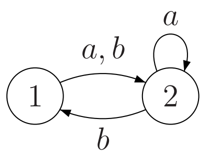


Figure 2.4.1: A deterministic semiautomaton with transition monoid T_2

First, what are the letter actions? For the letter a, we have transitions $1 \xrightarrow{a} 2$ and $2 \xrightarrow{a} 2$. Thus the corresponding letter action is the constant transformation $(\{1, 2\} \to 2)$, which sends every state to state 2. For the letter b, we have transitions $1 \xrightarrow{b} 2$ and $2 \xrightarrow{b} 1$. The corresponding letter action is the permutation $(1, 2)$, which swaps states 1 and 2.

Now, the transition monoid is the monoid of word actions generated by composing these letter actions. To compute the word action ab, we compose the transformations a and b:

$$1ab = 1(\{1, 2\} \to 2)(1, 2) = 2(1, 2) = 1,$$
$$2ab = 2(\{1, 2\} \to 2)(1, 2) = 2(1, 2) = 1.$$

We see that ab is the constant transformation $(\{1, 2\} \to 1)$ that maps everything to 1. We could have also determined this by looking at the path labelled with ab starting from each state:

$$1 \xrightarrow{a} 2 \xrightarrow{b} 1,$$
$$2 \xrightarrow{a} 2 \xrightarrow{b} 1.$$

Both methods of computation give the same result.

There are only four transformations of $\{1, 2\}$, and we have seen three of them so far. The last one is the identity transformation. This turns out to be the word action of bb:

$$1bb = 1(1, 2)(1, 2) = 2(1, 2) = 1,$$
$$2bb = 2(1, 2)(1, 2) = 1(1, 2) = 2.$$

We can also see this by looking at the path labelled with *bb*:

$$1 \xrightarrow{b} 2 \xrightarrow{b} 1,$$
$$2 \xrightarrow{b} 1 \xrightarrow{b} 2.$$

This shows that the transition monoid of the semiautomaton is T_2. Because we have seen all transformations of $\{1, 2\}$, the word actions of other words will all be equivalent to one of the word actions we have seen. For example, consider the word action of *ababba*. Because this word ends with the constant transformation a, it should just be equivalent to a. We can confirm this by looking at paths:

$$1 \xrightarrow{a} 2 \xrightarrow{b} 1 \xrightarrow{a} 2 \xrightarrow{b} 1 \xrightarrow{b} 2 \xrightarrow{a} 2,$$
$$2 \xrightarrow{a} 2 \xrightarrow{b} 1 \xrightarrow{a} 2 \xrightarrow{b} 1 \xrightarrow{b} 2 \xrightarrow{a} 2.$$

Indeed, we get the transformation $(\{1, 2\} \to 2)$ as expected. ■

Example 2.4.2 (A semiautomaton with transition monoid S_n)**.** Consider the n-state deterministic semiautomaton depicted in Figure 2.4.2. We will show for $n \geq 2$ that its transition monoid is S_n, the set of all permutations of $\{1, \ldots, n\}$.

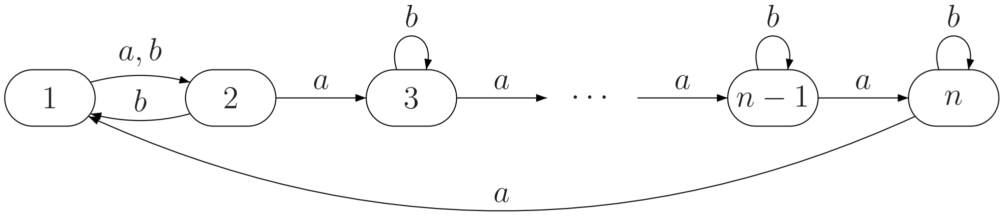


Figure 2.4.2: A deterministic semiautomaton with transition monoid S_n.

In case the definition of the transition function is not clear from the image, here is a formal definition. Fix a state $i \in \{1, 2, \ldots, n\}$. We define:

- $T(i, a) = (i + 1) \bmod n$.
- $T(1, b) = 2$ and $T(2, b) = 1$.
- $T(i, b) = i$ for $i \notin \{1, 2\}$.

It is not hard to see from this definition and from the picture that the letter actions are as follows:

- a is the cyclic permutation $(1, 2, \ldots, n)$.
- b is the transposition $(1, 2)$.

In the future, we will start using letter actions to *define* the transition functions of (semi)automata. That is, instead of writing out the value of T for each state-letter pair, we will just specify the action of each letter. It is much more convenient to just write "$b = (1, 2)$" than to write "$T(1, b) = 2$, $T(2, b) = 1$ and $T(i, b) = i$ for $i \notin \{1, 2\}$", and the same information is conveyed.

To prove that the transition monoid of this semiautomaton is S_n, we must show that these two letter actions generate all permutations of $\{1, 2, \ldots, n\}$; that is, we must show that each permutation can be represented as a word action. We must also show that no other elements can be generated, but this part is easy: the letter actions are both permutations, and the composition of two permutations is a permutation, so all the word actions will be permutations.

We will take for granted the intuitive fact that if you have an ordered collection of objects, you can rearrange the collection in any order by just repeatedly swapping elements one at a time. Mathematically, this is equivalent to saying that the set of all *transpositions* of $\{1, 2, \ldots, n\}$, the permutations that swap two elements, generates the symmetric group S_n. So we just need to find a word action corresponding to each transposition.

First we show that all transpositions of the form $(i, i + 1)$ are word actions. This is true for $i = 1$, since $b = (1, 2)$. Now, if we have $w = (i, i + 1)$ for some word w, then we claim $a^{n-1}wa = (i + 1, i + 2)$. First we show that this word action swaps $i + 1$ and $i + 2$ (arithmetic below is performed modulo n):

$$\begin{aligned}(i + 1)a^{n-1}wa &= (i + 1 + n - 1)wa = iwa = (i + 1)a = i + 2,\\ (i + 2)a^{n-1}wa &= (i + 2 + n - 1)wa = (i + 1)wa = ia = i + 1.\end{aligned}$$

Next we show that all other elements $j \notin \{i + 1, i + 2\}$ are unaffected. Since $ja^{n-1} = j - 1 \notin \{i, i + 1\}$, it follows that $(j - 1)w = j - 1$. Then $(j - 1)a = j$. It follows that $a^{n-1}wa = (i + 1, i + 2)$. Hence we have

$$\begin{aligned}b &= (1, 2),\\ a^{n-1}ba &= (2, 3),\\ a^{n-1}a^{n-1}baa &= (3, 4),\\ &\cdots\end{aligned}$$

To construct word actions for all transpositions of the form $(1, i)$, notice that

$$(i, i+1)(1, i)(i, i+1) = (1, i+1).$$

Thus if $w_i = (i, i+1)$, we have:

$$\begin{aligned} w_1 &= (1,2), \\ w_2 w_1 w_2 &= (1,3), \\ w_3 w_2 w_1 w_2 w_3 &= (1,4), \\ &\dots \end{aligned}$$

Finally, for arbitrary transpositions (i, j), notice that $(1, i)(1, j)(1, i) = (i, j)$. Thus all transpositions have corresponding word actions, and it follows that all permutations have corresponding word actions.

For example, if $n = 4$, consider the transposition $(2, 4)$. We have:

$$\begin{aligned} (2,4) &= (1,2)(1,4)(1,2) \\ &= w_1(w_3 w_2 w_1 w_2 w_3)w_1 \\ &= (1,2)(3,4)(2,3)(1,2)(2,3)(3,4)(1,2) \\ &= b(a^3a^3baa)(a^3ba)b(a^3ba)(a^3a^3baa)b \\ &= ba^6ba^5baba^3ba^7ba^2b. \end{aligned}$$

We can simplify this word a bit by noting that a^4 is the identity transformation.

$$ba^6ba^5baba^3ba^7ba^2b = ba^2bababa^3ba^3ba^2b.$$

If you wish, you can verify using Figure 2.4.3 that the action of this word really does swap 2 and 4 and leave the other states undisturbed. We leave it as an exercise to construct a word action for the cyclic permutation $(1, 2, 3)$. ■

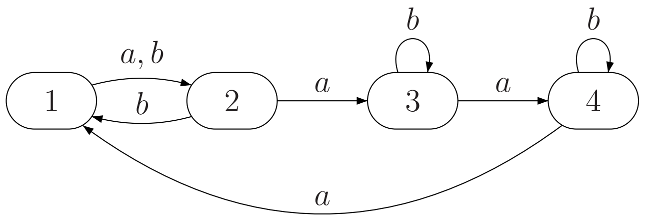


Figure 2.4.3: A deterministic semiautomaton with transition monoid S_4.

Example 2.4.3 (A semiautomaton with transition monoid T_n)**.** Consider the n-state deterministic semiautomaton depicted in Figure 2.4.4. We will show for $n \geq 2$ that its transition monoid is T_n, the set of all transformations of $\{1, \ldots, n\}$. The transition monoid of an n-state automaton cannot be larger than T_n, since every word action is a transformation. Thus this automaton has a *maximal* transition monoid for each n. Automata with maximal transition monoids turn out to be really useful in state complexity.

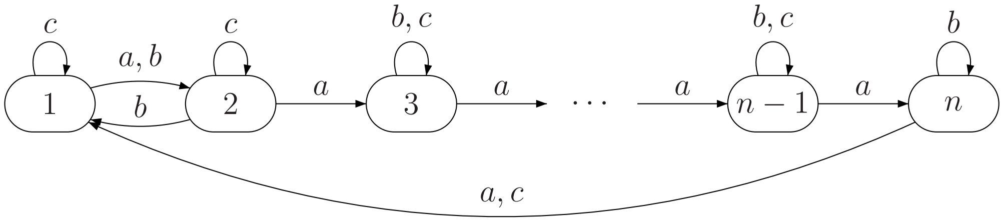


Figure 2.4.4: A deterministic semiautomaton with transition monoid T_n.

This semiautomaton is a modified version of the semiautomaton from Figure 2.4.2, with one new letter action added: the action $c = (n \to 1)$ that sends state n to state 1 and fixes all other states. Together with the letter actions $a = (1, 2, \ldots, n)$ and $b = (1, 2)$ that generate the symmetric group, these three transformations are sufficient to generate the full transformation monoid T_n.

To show that the transition monoid of this semiautomaton is T_n, first we prove a technical lemma.

Lemma 2.4.4. *T_n is generated by permutations together with the set of transformations of the form $(i \to j)$.*

Proof. Fix an arbitrary transformation t of $\{1, \ldots, n\}$. Construct a directed graph G where the vertices are $\{1, \ldots, n\}$ and the edges are $\{(i, it) : 1 \leq i \leq n\}$. Divide the graph into connected components $C_1, \ldots, C_m$. For each component C_k, we will construct a transformation t_k such that the composition $t_1 \cdots t_m$ is t, and such that t_i is a composition of permutations and transformations of the form $(i \to j)$.

First we observe that each component has exactly one cycle. Indeed, suppose $u_1 \to u_2 \to \cdots \to u_k \to u_1$ and $v_1 \to v_2 \to \cdots \to v_\ell \to v_1$ are distinct cycles. If the cycles overlap at any point, say $u_i = v_j$, then the next edge has the from $(u_i, u_i t) = (v_j, v_j t)$, and so $u_{i+1} = v_{j+1}$. Applying this fact repeatedly shows then that the cycles are identical. So if the cycles are distinct, they are also disjoint, and there cannot be an edge leading from one cycle to the other. But these cycles are part of a single connected component, so there must be a vertex i outside the cycles and two distinct directed paths leading from i to each of the cycles. This is impossible, since each vertex i has exactly one outgoing edge (i, it). So there must be exactly one cycle $v_1 \to v_2 \to \cdots \to v_\ell \to v_1$ in each component C_k. For each C_k, define $c_k = (v_1, v_2, \ldots, v_\ell)$; this will be the "permutation part" of t_k.

Since each C_k has at most one cycle, the vertices and edges outside of the cycle form acyclic paths that lead into the cycle. We call the vertices outside the cycle the *external vertices*. Let d_k be the maximum length of a path from an external vertex to a cycle vertex. We define transformations $t_{k,1}, \ldots, t_{k,d_k}$ as follows. Let $\{u_1, \ldots, u_{k_d}\}$ be the set of external vertices of C_k with distance d from the cycle, i.e., those vertices which have a path of length d leading into the cycle. (Note that for each vertex there is exactly one path leading to the cycle, since each vertex has exactly one outgoing edge.) Then set $t_{k,d} = (u_1 \to u_1 t) \cdots (u_{k_d} \to u_{k_d} t)$. Finally, we define

$$t_k = c_k t_{k,1} \cdots t_{k,d_k}.$$

We claim that $t = t_1 \cdots t_m$. To see this, consider the value of $it_1 \cdots t_m$. The transformation t_k affects only vertices in component C_k, so if i belongs to component C_j, then we have

$$it_1 \cdots t_m = it_j = ic_j t_{j,1} \cdots t_{j,d_j}.$$

Notice that c_j affects only vertices in the cycle of C_j, and $t_{j,d}$ affects only vertices that are distance d from the cycle. If i belongs to the cycle, we have $ic_j = it$, and it still belongs to the cycle, so it is unaffected by the subsequent transformations. Thus $it_1 \cdots t_m = it$ as required. If i is distance d from the cycle, then we have $ic_j t_{j,1} \cdots t_{j,d} = it_{j,d} = it$. Now, it is distance $d-1$ from the cycle (or in the cycle if $d = 1$), so it is unaffected by all the transformations that come after $t_{j,d}$ since they operate on elements of distance greater than d. It follows that $it_1 \cdots t_m = it$ as required. This shows that each transformation t can be expressed by a composition of permutations and transformations of the form $(i \to j)$. □

This lemma shows that it suffices to just find word actions for transformations of the form $(i \to j)$ with $i \neq j$ (since if $i = j$ then $(i \to j)$ is the identity permutation). First note that if $i \notin \{1, n\}$, then we have

$$(i \to 1) = (i, n)(n \to 1)(i, n).$$

So we can construct a word action for $(i \to 1)$ from $c = (n \to 1)$ and a word action for (i, n). Then for $j \notin \{i, 1\}$, we have

$$(i \to j) = (1, j)(i \to 1)(1, j).$$

So using a word action for $(1,j)$ we can get word actions for all transformations $(i \to j)$, as desired. It follows that our semiautomaton has transition monoid T_n, the monoid of all transformations of $\{1, 2, \ldots, n\}$. ■

Reachability and distinguishability. Let (Q, Σ, T) be a semiautomaton. For $p, q \in Q$, we say *q is reachable from p via w* if $q \in pw$ (or $pw = q$ in the deterministic case). If we view the semiautomaton $\mathcal{A}$ as a directed graph, this is equivalent to saying there is a directed path from p to q in the graph, and the labels of the transitions in the path spell out w. We say *q is reachable from p* if there exists a word w such that q is reachable from p via w. In an automaton (Q, Σ, T, I, F), we say *q is reachable* if $Iw = q$, that is, every path from a state of I on w leads to q. In a DFA, this is just saying there is a path from the unique initial state to q.

These notions extend to subsets of states: for $S, S' \subseteq Q$, we say *S' is reachable from S via w* if $Sw = S'$. We say *S' is reachable from S* if there exists a word w such that S' is reachable from S via w. We say *S is reachable* if S is reachable from I. An automaton is called *accessible* if every state is reachable from some state of I.

Let (Q, Σ, T) be a *deterministic* semiautomaton. Fix $S \subseteq Q$ and define an equivalence relation Δ_S on Q as follows: for $p, q \in Q$, let $p \,\Delta\, q$ if and only if for all $z \in \Sigma^*$, $pz \in S \iff qz \in S$. Equivalent states under this relation are called *indistinguishable under S*, and non-equivalent states are called *distinguishable under S*. If $z \in \Sigma^*$ is a word such that $pz \in S \iff qz \notin S$, we say that *$p$ and q are distinguishable under S by z*, or that *z distinguishes p and q under S*. For a deterministic automaton (Q, Σ, T, I, F), in the case $S = F$ we simply say states are *indistinguishable* or *distinguishable* without mentioning the set, and we write Δ instead of Δ_F for the indistinguishability relation.

The notions of reachability and distinguishability play an important role in the theory of *minimal automata*, which will be discussed in Section 2.4.2.

DFA isomorphism. We say two DFAs $\mathcal{D} = (Q, \Sigma, T, 1, F)$ and $\mathcal{D}' = (Q', \Sigma', T', 1', F')$ are *isomorphic* if $\Sigma = \Sigma'$ and there is a bijection $\varphi\colon Q \to Q'$ such that $1\varphi = 1'$, $F\varphi = F'$, and $p \xrightarrow{a} q$ if and only if $p\varphi \xrightarrow{a} q\varphi$. Informally, two DFAs are isomorphic if they have the same alphabet, and they are identical in structure, but they possibly have different names for their states.

Automata as language recognizers. Let $\mathcal{A} = (Q, \Sigma, T, I, F)$ be a finite automaton. The *language recognized by $\mathcal{A}$*, or simply the *language of $\mathcal{A}$*, is the set of words $L(\mathcal{A}) = \{w \in \Sigma^* : Iw \cap F \neq \emptyset\}$. If $\mathcal{A}$ is deterministic, this becomes $L(\mathcal{A}) = \{w \in \Sigma^* : 1w \in F\}$. In other words, a word w is in the language of $\mathcal{A}$ if the action of w maps an initial state to a final state. If w is in the language of $\mathcal{A}$, we say that $\mathcal{A}$ *accepts* w.

There is also a useful notion of the *language accepted from a state*. Fix a state $q \in Q$. The language of the automaton $\mathcal{A}_q = (Q, \Sigma, T, q, F)$, which has q as its sole initial state, is called the *language accepted from* q or simply the *language of* q. If w is in the language of q, we say q *accepts* w. Observe that two states p and q are indistinguishable (under F) if and only if they have the exact same language.

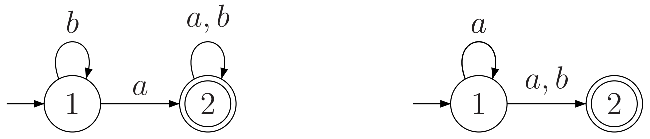


Figure 2.4.5: Deterministic automaton (left) and nondeterministic automaton (right).

To illustrate the concept of language recognition, let us revisit our example automata from Chapter 1. See Figure 2.4.5. The automata from Figure 1.1.1 of Chapter 1 have now been assigned initial states and final states. Initial states are denoted by an unlabelled arrow pointing into the state, and final states are denoted by a double-circled node. In this case, both automata have 1 as the sole initial state and 2 as the sole final state. Deterministic automata can only have one initial state, but in nondeterministic automata, multiple initial states are allowed. In both types of automata, multiple final states are allowed, but there is only one final state in this example.

What is the language recognized by each automaton? In both cases, it will be a subset of $\Sigma^* = \{a, b\}^*$, the set of all finite-length words comprised of as and bs. In both cases, a word $w \in \Sigma^*$ is in the language of the automaton if w maps an initial state to a final state. Both automata have a unique initial state, 1, and a unique final state, 2, so we are looking for words w such that $2 \in 1w$. That is, we are looking for words w such that 2 *is reachable from* 1 *via* w. Recall that in terms of the directed graph representation of the automata, this corresponds exactly to saying that there is a path from 1 to 2 for which the edge labels spell out w.

We will use this observation to figure out which languages are recognized by the automata of Figure 2.4.5. First, let us focus on the left automaton, the deterministic one. Notice that we can reach the final state 2 from the initial state 1 only by taking the transition $1 \xrightarrow{a} 2$; thus all words accepted by the automaton must contain at least one a. In fact, all words over $\{a, b\}$ that contain at least one a are accepted, and no others. We can see that since there is a loop $1 \xrightarrow{b} 1$, we can have any number of bs before the first a occurs. After reading the first a, we are forced to transition to state 2. In state 2 there are loops $2 \xrightarrow{a} 2$ and $2 \xrightarrow{b} 2$, so every possible word we can read will keep us in state 2, and thus arbitrary words containing at least one a are accepted. If a word does not contain at least one a, then it must consist entirely of bs, and the corresponding path will leave us stuck in the non-final state 1. Thus the language of the left automaton consists of all words over $\{a, b\}$ that contain at least one a.

Now consider the right automaton, which is not deterministic. The determinism of a deterministic automaton implies that for each word over the alphabet Σ, there is a *exactly one* path labelled with that word. In a nondeterministic automaton, there could be multiple paths or no paths corresponding to a word. We claim the language of the right automaton consists of all non-empty words which either contain no bs, or contain exactly one b occurring at the very end of the word. First we show all such words are accepted by the right automaton. A word w of the form described above can be written as $w = x\sigma$, where x is a (possibly empty) word consisting only of as, and σ is either a or b. Then there is an accepting path labelled with w: take the transition $1 \xrightarrow{a} 1$ for each a in x, then take the transition $1 \xrightarrow{\sigma} 2$. Next we show that no other words are accepted by the right automaton. The empty word is accepted only if the initial state is final, and this is not the case. If a non-empty word w contains a b, but this b does not occur at the end of the word, we can write $w = xb\sigma y$, where x and y are possibly empty words and σ is either a or b. A path starting at state 1 and labelled with xb necessarily ends in state 2, since b forces a transition from state 1 to state 2. There are no paths starting from state 2 except from the "empty path", so in particular there are no paths with non-empty labels starting from state 2. But σy is non-empty, so it follows that $w = xb\sigma y$ cannot be the label of a path at all, let alone an accepting path. This proves the claim.

Regular languages. A language L over Σ^* is called *regular* if it satisfies one of the following conditions:

- $L = \emptyset$, $L = \{\varepsilon\}$, or $L = \{a\}$ for $a \in \Sigma$.
- L is the *union* of two regular languages.
- L is the *concatenation* of two regular languages: $L = R'R = \{xy : x \in R', y \in R\}$, where R' and R are regular languages over Σ.
- L is the *Kleene star* of a regular language: $L = R^* = \{\varepsilon\} \cup R \cup RR \cup RRR \cup \cdots$ where R is a regular language over Σ.

Kleene's theorem. It was proved by Kleene [56] that a language is regular *if and only if* it is recognized by a finite automaton (in fact a deterministic automaton suffices). In light of this theorem, we treat the term "regular language" as essentially equivalent to "language recognized by a DFA"; we rarely use the inductive definition above. We will essentially prove one direction of this theorem in Section 2.4.4 by constructing DFAs for the union and concatenation of two regular languages and the star of a regular language. Then to show that every regular language has a DFA, it remains just to construct DFAs for $\emptyset$, ε and languages of the form $\{a\}$, which is an easy task. We omit a proof of the other direction, that every language recognized by a DFA is regular, since the techniques used to prove it are not especially useful in the context of state complexity.

2.4.2 Minimal Automata and the Syntactic Monoid

An automaton is *minimal* if it has the least number of states amongst all automata which recognize the same language. Minimal automata are useful for efficiently representing languages. A language can have multiple minimal automata, and in the case of *nondeterministic* automata it is generally quite difficult to describe the set of all minimal automata for a particular language, or even to produce one example of a minimal automaton. However, for deterministic automata, the situation is much nicer. For each regular language, there is a *canonical* minimal DFA which is essentially unique: given two minimal DFAs for the same language, they are necessarily isomorphic (that is, they are either equal or differ only in the names assigned to the states). In this section, we present three equivalent constructions of this canonical minimal DFA. We will also define the *syntactic monoid* of a language and prove that it is isomorphic to the transition monoid of the language's minimal DFA.

The Myhill-Nerode theorem. The first construction is based on a fundamental characterization of regular languages due to Myhill and Nerode [65, 66].

Theorem 2.4.5 (Myhill and Nerode)**.** *Let $L \subseteq \Sigma^*$ and define the following equivalence relation ρ_L on Σ^*: let $x \mathrel{\rho_L} y$ if and only if for all $z \in \Sigma^*$, $xz \in L \iff yz \in L$. The following are equivalent:*

1. *L is a regular language.*
2. *Σ^*/ρ_L is finite.*

Furthermore, every minimal DFA for L has Σ^*/ρ_L states.

Proof. Suppose L is regular. Then there exists a DFA $\mathcal{D} = (Q, \Sigma, T, 1, F)$ for L. We may define an equivalence relation $\rho_\mathcal{D}$ on Σ^* as follows: $x \mathrel{\rho_\mathcal{D}} y$ if and only if for all $z \in \Sigma^*$, $1xz \in F \iff 1yz \in F$. We make two claims: that $\rho_\mathcal{D}$ has at most $|Q|$ classes, and that $\rho_L \subseteq \rho_\mathcal{D}$.

To see that $\rho_{\mathcal{D}}$ has at most $|Q|$ classes, we define a surjection from a subset of Q to $\Sigma^*/\rho_{\mathcal{D}}$. The subset of Q we will use is $R = \{1w : w \in \Sigma^*\}$. For each $q \in R$, select a word w_q such that $1w_q = q$ and consider the map $q \mapsto w_q\rho_{\mathcal{D}}$. This map is surjective since for each class $x\rho_{\mathcal{D}} \in \Sigma^*/\rho_{\mathcal{D}}$, we have $x\rho_{\mathcal{D}} = w_{1x}\rho_{\mathcal{D}}$.

To see that $\rho_L \subseteq \rho_{\mathcal{D}}$, suppose $(x, y) \in \rho_L$. Then for all $z \in \Sigma^*$, we have $1xz \in F \iff xz \in L \iff yz \in L \iff 1yz \in F$, so $(x, y) \in \rho_{\mathcal{D}}$. Since $\rho_L \subseteq \rho_{\mathcal{D}}$, ρ_L either has the same number or fewer equivalence classes than $\rho_{\mathcal{D}}$, and thus the set Σ^*/ρ_L is finite and has size at most $|Q|$.

Now, suppose Σ^*/ρ_L is finite. To show L is regular, we construct a DFA $\mathcal{D}_L = (\Sigma^*/\rho_L, \Sigma, T_L, \varepsilon\rho_L, F_L)$ for L, as follows:

- The state set is Σ^*/ρ_L, the equivalence classes of ρ_L.
- The initial state is $\varepsilon\rho_L$, the equivalence class of the empty word.
- The final state set is $F_L = \{w\rho_L : w \in L\}$, the equivalence classes of words in L.
- The transitions T_L have the form $w\rho_L \xrightarrow{a} (wa)\rho_L$, for all $w \in \Sigma^*$ and $a \in \Sigma$.

We must justify two things: that the transition relation is a well-defined function (so this automaton is indeed deterministic) and that this DFA recognizes L.

To see that $T_L : (\Sigma^*/\rho_L) \times \Sigma \to (\Sigma^*/\rho_L)$ is a function, it suffices to show for all $a \in \Sigma$ that if $x\rho_L = y\rho_L$, then $(xa)\rho_L = (ya)\rho_L$. If $x\rho_L = y\rho_L$, then for all $z \in \Sigma^*$ we have $xz \in L \iff yz \in L$. In particular, this holds for $z = az'$, where $z' \in \Sigma^*$. Thus for all $z' \in \Sigma^*$ we have $(xa)z' \in L \iff (ya)z' \in L$, and it follows that $(xa)\rho_L = (ya)\rho_L$.

To see that $\mathcal{D}_L$ recognizes L, suppose $w \in L$. Then $\varepsilon\rho_L \xrightarrow{w} w\rho_L \in F_L$, so w is accepted by $\mathcal{D}_L$. Conversely, if w is accepted by $\mathcal{D}_L$, then $(\varepsilon\rho_L)w \in F_L$. But $(\varepsilon\rho_L)w = w\rho_L$, and $w\rho_L \in F_L$ implies $w \in L$. So $\mathcal{D}_L$ recognizes L.

Finally, to see that every minimal DFA for L has Σ^*/ρ_L states, note that we have proved that if $\mathcal{D}$ is an *arbitrary* DFA for L, then the size of Σ^*/ρ_L is bounded above by the number of states in $\mathcal{D}$. So $|\Sigma^*/\rho_L|$ is a lower bound on the number of states in a minimal DFA. But we also constructed a minimal DFA for L with exactly Σ^*/ρ_L states. Thus $|\Sigma^*/\rho_L|$ is also an upper bound, and the result follows. □

Example 2.4.6 (A minimal automaton via the Myhill-Nerode theorem). Let $\Sigma = \{a, b\}$ and consider the following language:

$$L = \{w \in \Sigma^* : w \text{ has an even number of } a\text{s and an odd number of } b\text{s}\}.$$

What are the classes of the equivalence relation ρ_L? First consider the equivalence class $\varepsilon\rho_L$ of the empty word ε. We have $w\,\rho_L\,\varepsilon$ if and only if for all $z \in \Sigma^*$, we have $wz \in L \iff z \in L$. If z is in L, then the number of as in z is even and the number of bs in z is odd. When we concatenate z onto a word w, we essentially add an even number of as and an odd number of bs to w. So to have $wz \in L$, we need that:

- w has an even number of as, so that wz has an even number of as.
- w has an even number of bs, so that wz has an odd number of bs.

Therefore, a word w is ρ_L-related to ε if and only if it has an even number of as and a even number of bs.

The other equivalence classes of ρ_L are based on the other possibilities for the parities of as and bs in a word. One may verify the following statements using similar arguments to above:

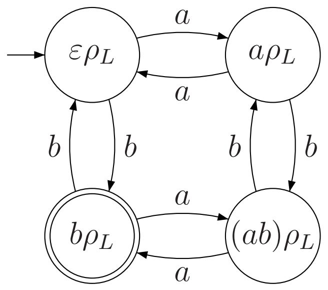
- $a\rho_L$ consists of all words with an odd number of as and an even number of bs.
- $b\rho_L$ consists of all words with an even number of as and an odd number of bs.
- $(ab)\rho_L$ consists of all words with an odd number of as and an odd number of bs.

There are no other possible parities, so ρ_L has exactly four equivalence classes. The Myhill-Nerode theorem therefore tells us that:

- L is a regular language.
- Every minimal DFA for L has exactly four states.
- A minimal DFA for L can be constructed from the classes of ρ_L.

The minimal DFA we obtain by following the procedure in the proof of Theorem 2.4.5 is shown below, in Figure 2.4.6. ■

The quotient automaton. The second minimal DFA construction we will discuss was first described by Brzozowski [4, 5], and uses the notion of *quotients* of languages. The *left quotient* of a language $L \subseteq \Sigma^*$ by a language $L' \subseteq \Sigma^*$ is $(L')^{-1}L = \{y \in \Sigma^* : \exists x \in L',\ xy \in L\}$. Similarly, the *right quotient* of L by a L' is $L'(L)^{-1} = \{x \in \Sigma^* : \exists y \in L',\ xy \in L\}$. This is the most general definition of quotients, but we are mostly interested in the special



$$\begin{tikzcd}
\arrow[r] & \varepsilon\rho_L \arrow[r, bend left, "a"] \arrow[d, bend left, "b"] & a\rho_L \arrow[l, bend left, "a"] \arrow[d, bend left, "b"] \\
& b\rho_L \arrow[u, bend left, "b"] \arrow[r, bend left, "a"] & (ab)\rho_L \arrow[l, bend left, "a"] \arrow[u, bend left, "b"]
\end{tikzcd}$$

Figure 2.4.6: Minimal DFA for the language of words over $\{a, b\}$ with an even number of as and an odd number of bs.

case of *left quotients by single words*: sets of the form $w^{-1}L = \{x \in \Sigma^* : wx \in L\}$. When we speak of the *quotients of a language* without further qualification, we mean the left quotients of the language by words.

The following theorem, similar in spirit to the Myhill-Nerode theorem, characterizes regular languages in terms of their quotients.

Theorem 2.4.7. *Let $L \subseteq \Sigma^*$. The following are equivalent:*

1. *L is a regular language.*
2. *L has finitely many quotients.*

Furthermore, the number of quotients of L is equal to the number of states in a minimal DFA for L.

Proof. Suppose L is regular. Then there exists a DFA $\mathcal{D} = (Q, \Sigma, T, 1, F)$ for L. For each $q \in Q$, let L_q be the language of state q in $\mathcal{D}$. We claim that each quotient of L lies in the set $\{L_q : q \in Q\}$. Since there are only $|Q|$ languages in this set, this means there are at most $|Q|$ distinct quotients of L. To prove the claim, simply note that

$$x \in w^{-1}L \iff wx \in L \iff 1wx \in F \iff x \in L_{1w},$$

and thus $w^{-1}L = L_{1w}$ for all $w \in \Sigma^*$.

Now, suppose L has finitely many quotients. We construct a DFA $\mathcal{D} = (Q, \Sigma, T, L, F)$ for L:

- The state set Q is $\{w^{-1}L : w \in \Sigma^*\}$, the set of all quotients of L. We know by assumption that this set is finite.

- The initial state is L itself, which is a quotient of L (in particular, the quotient $\varepsilon^{-1}L$).
- The final state set is $F = \{w^{-1}L : w \in L\}$, the quotients of L by words that belong to L. Equivalently, these are the quotients that contain the empty word.
- The transitions T have the form $w^{-1}L \xrightarrow{a} (wa)^{-1}L$, for all $w \in \Sigma^*$ and $a \in \Sigma$.

We must justify that T is a well-defined function and that $\mathcal{D}$ recognizes L.

To see that T is a function, it suffices to show that if $x^{-1}L = y^{-1}L$, then $(xa)^{-1}L = (ya)^{-1}L$. If $x^{-1}L = y^{-1}L$ then $w \in (xa)^{-1}L \iff xaw \in L \iff aw \in x^{-1}L \iff aw \in y^{-1}L \iff yaw \in L \iff w \in (ya)^{-1}L$, so this holds.

To see that $\mathcal{D}$ recognizes L, simply note that $w \in L \iff w^{-1}L \in F \iff (\varepsilon^{-1}L)w \in F$, where $\varepsilon^{-1}L = L$ is the initial state of $\mathcal{D}$. Thus $\mathcal{D}$ accepts w if and only if $w \in L$.

Finally, to see that the number of quotients of L is equal to the number of states in a minimal DFA for L, note that we proved that if $\mathcal{D} = (Q, \Sigma, T, 1, F)$ is an arbitrary DFA for L, then L has at most $|Q|$ quotients. If we take $\mathcal{D}$ to be a minimal DFA, we see that the number of quotients of L is upper-bounded by the number of states in a minimal DFA for L. But we also proved there is a DFA for L with exactly as many states as there are quotients of L, giving a matching lower bound. □

Example 2.4.8 (A minimal automaton using quotients)**.** As before, let $\Sigma = \{a, b\}$ and consider the following language:

$$L = \{w \in \Sigma^* : w \text{ has an even number of } a\text{s and an odd number of } b\text{s}\}.$$

One of the quotients of this language is L itself, the quotient by the empty word. Now, consider an arbitrary quotient $w^{-1}L = \{x \in \Sigma^* : wx \in L\}$. If we have $wx \in L$, then wx has an even number of as and an odd number of bs. So if $x \in w^{-1}L$, there are four possibilities depending on the parity of as and bs in w:

- If w has an even number of as and an even number of bs, then x must have an even number of as and an odd number of bs.
- If w has an even number of as and an odd number of bs, then x must have an even number of as and an even number of bs.
- If w has an odd number of as and an even number of bs, then x must have an odd number of as and an odd number of bs.

- If w has an odd number of as and an odd number of bs, then x must have an odd number of as and an even number of bs.

It follows that L has exactly four quotients, which can be represented as L, $b^{-1}L$, $a^{-1}L$, and $(ab)^{-1}L$.

- If $x \in L$, then x must have an even number of as and an odd number of bs.
- If $x \in b^{-1}L$, then x must have an even number of as and an even number of bs.
- If $x \in a^{-1}L$, then x must have an odd number of as and an odd number of bs.
- If $x \in (ab)^{-1}L$, then x must have an odd number of as and an even number of bs.

Thus L is regular, and we can construct a minimal DFA out of these quotients. The minimal DFA we obtain by following the procedure in the proof of Theorem 2.4.7 is shown below, in Figure 2.4.7. Note that (for example) the quotient $a^{-1}L$ is *not* equal to the Myhill-Nerode class $a\rho_L$ as a set of words, even though they correspond to the same state in the minimal DFA! Quotients are not equivalence classes. ■

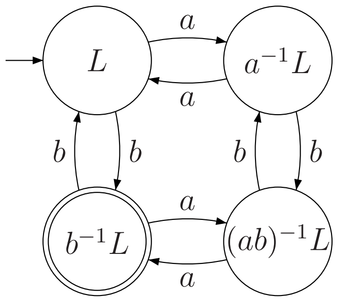


Figure 2.4.7: Minimal DFA for the language of words over $\{a, b\}$ with an even number of as and an odd number of bs.

Quotients and languages of states. Recall that in the proof of Theorem 2.4.7, we showed that that given a DFA for L, each quotient of L is equal to the *language accepted from a particular state of the DFA*. In particular, if q is the state reached by following the path labelled by w from the initial state, the quotient $w^{-1}L$ is equal to the language accepted from q. This is a very important property of quotients! You may wish to verify that this property holds for the DFA of Figure 2.4.7.

Minimization, reachability and indistinguishability. We have seen two constructions of a minimal DFA. We now prove that not only are these constructions equivalent, but that *all* minimal DFAs for a language, no matter how they are obtained, are isomorphic. The key result gives a third way of constructing a minimal DFA: via a *minimization procedure* which converts an arbitrary DFA to a minimal DFA.

Theorem 2.4.9. *Given $\mathcal{D} = (Q, \Sigma, T, 1, F)$ be a DFA for a regular language L, we may construct a minimal DFA $\mathcal{D}_M$ for L by modifying $\mathcal{D}$ as follows:*

1. *Remove all unreachable states from $\mathcal{D}_R$ to obtain a DFA $\mathcal{D}_R = (Q_R, \Sigma, T_R, 1, F_R)$, where $Q_R = \{1w : w \in \Sigma^*\}$ is the set of reachable states of $\mathcal{D}$, the transition function $T_R \colon Q_R \times \Sigma \to Q_R$ is the restriction of T to reachable states, and $F_R = Q_R \cap F$ is the set of reachable final states.*

2. *Let $\Delta = \Delta_{F_R}$, the indistinguishability relation under the reachable final states F_R. Set $\mathcal{D}_M = (Q_M, \Sigma, T_M, 1\Delta, F_M)$, where $Q_M = Q_R/\Delta$, the set of equivalence classes obtained by merging the reachable states of $\mathcal{D}$ according to indistinguishability; the function T_M is given by transitions $q\Delta \xrightarrow{a} (qa)\Delta$ for all $q \in Q_R$ and $a \in \Sigma$, and $F_M = \{f\Delta : f \in F_R\}$ is the set of equivalence classes of reachable final states.*

Furthermore, every minimal DFA for L is isomorphic to $\mathcal{D}_M$.

In summary, an arbitrary DFA can be *minimized* by *removing unreachable states* and *combining indistinguishable states*.

Proof. First, we prove that $\mathcal{D}_M$ is in fact a DFA by showing that T_M is a well-defined function. Suppose $p\Delta = q\Delta$; we want to show for all $a \in \Sigma$ that $(pa)\Delta = (qa)\Delta$. For all $z \in \Sigma^*$, we have $pz \in F_R \iff qz \in F_R$; in particular this holds for $z = az'$ for arbitrary $z' \in \Sigma^*$, and thus for all $z' \in \Sigma^*$ we have $(pa)z' \in F_R \iff (qa)z' \in F_R$. Hence $(pa)\Delta = (qa)\Delta$ as required.

Next, we prove that $\mathcal{D}_M$ recognizes L. We have $w \in L \iff 1w \in F \iff 1w \in F_R$, since a state which can be written as $1w$ is necessarily reachable. Now, $1w \in F_R \iff (1w)\Delta \in F_M \iff (1\Delta)w \in F_M$, and thus $w \in L$ if and only if w is accepted by $\mathcal{D}_M$.

Finally, we show two things: that $\mathcal{D}_M$ is minimal, and that every minimal DFA for L is isomorphic to $\mathcal{D}_M$. To see this, let $\mathcal{D}' = (Q', \Sigma, T', 1', F')$ be a minimal DFA for L. Note that if we apply the minimization procedure to $\mathcal{D}'$, then we must obtain a DFA for L with the same number of states as $\mathcal{D}'$, since $\mathcal{D}'$ is minimal. Thus $\mathcal{D}'$ cannot have any unreachable states, and it cannot have any pairs of indistinguishable states.

Now, we construct an isomorphism between $\mathcal{D}_M$ and $\mathcal{D}'$ as follows. For each equivalence class $q\Delta \in Q_M$, since $q \in Q_R$, we may write $q = 1w_q$ for some $w_q \in \Sigma^*$. Fix such a word w_q for each $q \in Q_R$. Now define $\varphi \colon Q_M \to Q'$ by $q\Delta \mapsto 1'w_q$. We must show that φ is well-defined, bijective, and satisfies the requirements for a DFA isomorphism.

To see that φ is well-defined, suppose $p\Delta = q\Delta$. We want to show that $1'w_p = 1'w_q$. Suppose not; then as we observed, $1'w_p$ and $1'w_q$ are distinguishable, and so there exists a word z such that $1'w_pz \in F' \iff 1'w_qz \notin F'$. Without loss of generality, assume $1'w_pz \in F'$. Then $w_pz \in L$ and $w_qz \notin L$. Thus $1w_pz = pz \in F_R$ and $1w_qz = qz \notin F_R$, which contradicts the fact that $p\Delta = q\Delta$.

To see that φ is injective, suppose $p\Delta \neq q\Delta$; we want to show that $1'w_p \neq 1'w_q$. Since $p\Delta \neq q\Delta$, states p and q are distinguishable by some word z. Without loss of generality, assume $pz \in F_R$ and $qz \notin F_R$. Then $1w_pz \in F_R$ and $1w_qz \notin F_R$, so $w_pz \in L$ and $w_qz \notin L$. It follows that $1'w_pz \in F'$ and $1'w_qz \notin F'$, so $1'w_p$ and $1'w_q$ are distinguishable. Since they are distinguishable, they cannot be equal.

To see that φ is surjective, fix a state $q' \in Q'$. Every state in Q' is reachable, and so we can write $q' = 1'w$ for some $w \in \Sigma^*$. Let $p = 1w$; we claim that φ maps $p\Delta$ to $1'w$. It suffices to show that $1'w_p = 1'w$. If not, then $1'w_p$ and $1'w$ are distinguishable by some word z. So without loss of generality, assume $w_pz \in L$ and $wz \notin L$. Then $(1w)z = pz \in L$ and $(1w_p)z = pz \notin L$, which is a contradiction.

Finally, we show that φ is a DFA isomorphism. To see that it preserves the initial state, note that 1Δ is mapped to $1'w_1$, and we can take $w_1 = \varepsilon$ without loss of generality. To see that it preserves final states, let $f\Delta$ be a final state in $\mathcal{D}_M$. We want to show that $1'w_f$ is final in $\mathcal{D}'$. Suppose $1'w_f$ is non-final; then $w_f \notin L$. But $1w_f = f \in F_R$, so $w_f \in L$, which is a contradiction. Finally, to see that it preserves transitions, fix $q\Delta \in Q_M$ and $a \in \Sigma$. Then we have $(q\Delta)a = (qa)\Delta$. We must show that $(1'w_q)a = 1'w_{qa}$. Suppose not; then these states are distinguishable by some word z. Without loss of generality, assume $w_qaz \in L$ and $w_{qa}z \notin L$. Since $w_qaz \in L$, we have $1w_qaz = (qa)z \in F_R$, but since $w_{qa}z \notin L$ we have $1w_{qa}z = (qa)z \notin F_R$, which is a contradiction.

Thus φ is a DFA isomorphism between $\mathcal{D}_M$ and the minimal DFA $\mathcal{D}'$. Since $\mathcal{D}_M$ is isomorphic to a minimal DFA for L, it follows that $\mathcal{D}_M$ is itself a minimal DFA for L. □

Example 2.4.10 (Minimizing an automaton)**.** Consider the following language over alphabet $\Sigma = \{a, b, c\}$:

$$L = \{w \in \Sigma^* : w \text{ starts with } a \text{ or } b, \text{ ends with } c \text{ and has length at most 3.}\}.$$

Figure 2.4.8 shows an automaton for this language. Although the transition diagram is complicated, this automaton is based on a fairly simple idea: track the first, second and third character of the word and decide whether it should be accepted accordingly. If the word does not start with a or b, we go to the "first c" state, which is inescapable; this enforces that the word must start with a or b. If the word starts with a or b, but only has two symbols, we will either reject it from the "second a" or "second b" state, or we will accept it from the "second c" state since then it ends in c. If the word has three symbols, we either reject it from the "third a" or "third b" state, or accept it from the "third c" state. If the word has any more symbols, we go to the inescapable, non-final "too long" state and the word is rejected.

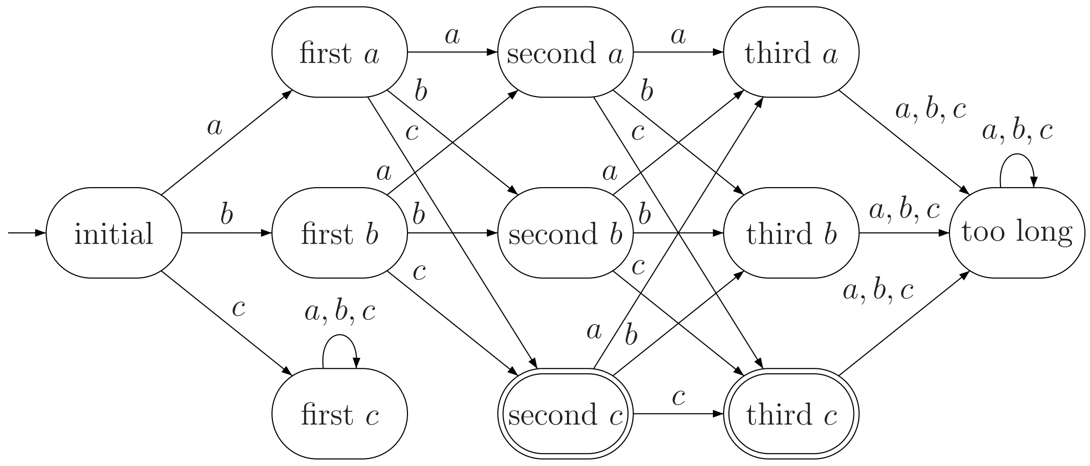


Figure 2.4.8: Overly complicated DFA for the language over $\{a, b, c\}$ of words which start with a or b, end with c, and have length at most 3.

It is not hard to see that we can find a smaller DFA for this language; for example, the "first c" and "too long" states could be combined into a single "fail" state. Let us perform the minimization procedure on the DFA of Figure 2.4.8 to see which other states can be combined. (We don't need to remove unreachable states, since all states are reachable.)

First, let us give simpler names to the states for convenience, as shown in Figure 2.4.9.

Now we compute the indistinguishability classes with respect to the final state set $F = \{2c, 3c\}$. Recall that two states p, q are indistinguishable if for all $z \in \Sigma^*$, we have $pz \in F \iff qz \in F$. Intuitively, this means that if we put our fingers on the states p and q in the graph of the DFA, and follow the two paths on z from the two states, the pair of states we end up at is always either a pair of final states or a pair of non-final states, no matter what z is chosen. Distinguishability means that we can find some z where one path leads to a final state and the other leads to a non-final state.

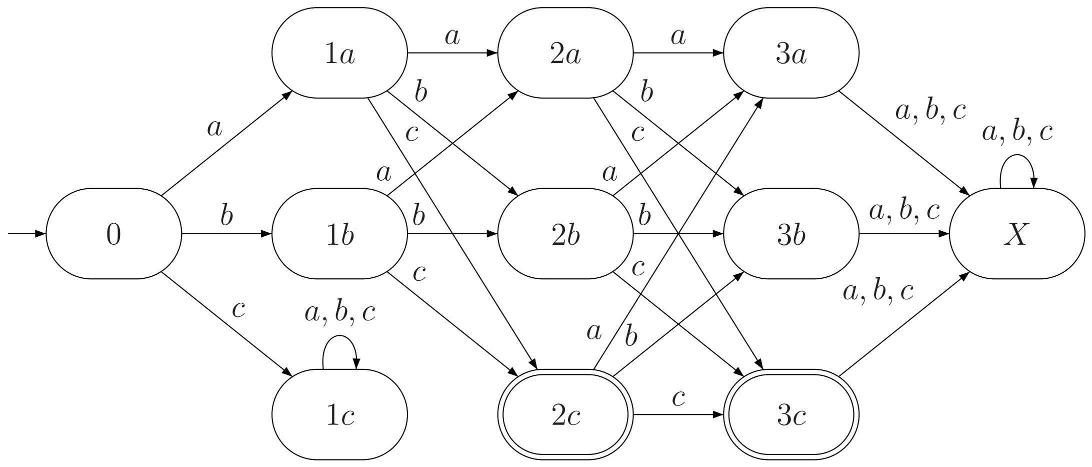


Figure 2.4.9: DFA of Figure 2.4.8 with the states renamed.

Indistinguishability can also be defined in terms of the *language recognized from a state*. If two states recognize the same language, they are indistinguishable, and otherwise they are distinguishable by a word in the symmetric difference of their two languages. Thus one way to find indistinguishable states in an automaton is to look for states which clearly recognize the same language. For example, states $1c$ and X both obviously recognize the empty language, so they are indistinguishable. States $3a$ and $3b$ also both recognize the empty language. Thus X, $1c$, $3a$ and $3b$ all belong to one indistinguishability class. For each of the remaining states, we can always find a path from the state to a final state, so the remaining states recognize a non-empty language and thus are distinguishable from the empty language states. It follows that one of the indistinguishability classes is $X\Delta = \{X, 1c, 3a, 3b\}$. As an intermediate step, let us construct an automaton with these states merged; see Figure 2.4.10.

Let us look for more indistinguishability classes. Notice that the final state $3c$ accepts only the word ε. The only other state which accepts ε is $2c$; but $2c$ also accepts c, so $2c$ and $3c$ are distinguishable. These final states are distinguishable from all non-final states, and thus they both belong to their own one-element indistinguishability classes. That is, we have $(2c)\Delta = \{2c\}$ and $(3c)\Delta = \{3c\}$.

What about the remaining non-final states? Let's start with the ones closest to the final states, $2a$ and $2b$. Notice that both of these states recognize exactly the language $\{c\}$; every other word leads to the inescapable non-final state $X\Delta$. Thus $2a$ and $2b$ are indistinguishable. They are distinguishable from 0, $1a$ and $1b$ since those states each recognize a word of length greater than one. Hence we have $(2a)\Delta = \{2a, 2b\}$. We merge these states; see Figure 2.4.11.

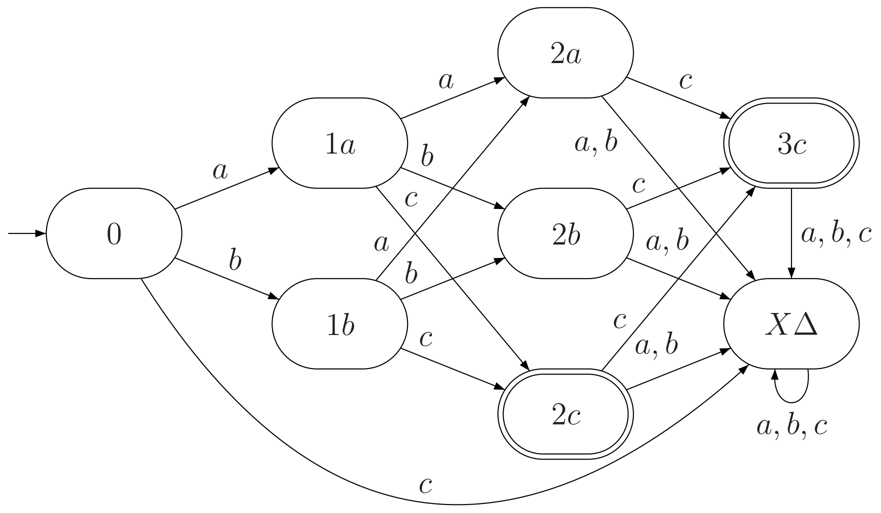


Figure 2.4.10: DFA of Figure 2.4.9 with the states in $X\Delta$ merged.

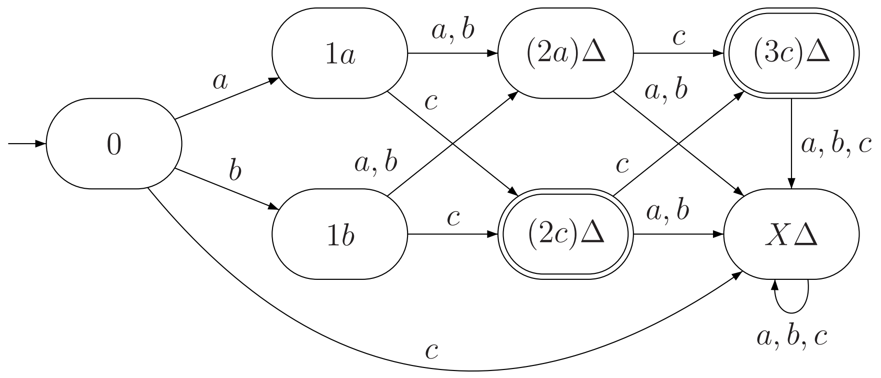


Figure 2.4.11: DFA of Figure 2.4.10 with the states in $(2a)\Delta$ merged.

Notice now that $1a$ and $1b$ both recognize the language $\{c, ac, bc, cc\}$. These states are indistinguishable from 0 since 0 recognizes words of length three. Thus $(1a)\Delta = \{1a, 1b\}$ and $0\Delta = \{0\}$. We have now fully determined the equivalence classes of Δ:

$$0\Delta = \{0\},\ (1a)\Delta = \{1a, 1b\},\ (2a)\Delta = \{2a, 2b\},$$
$$(2c)\Delta = \{2c\},\ (3c)\Delta = \{3c\},\ X\Delta = \{X, 1c, 3a, 3b\}.$$

Merging the remaining states gives a minimal DFA for our language, shown in Figure 2.4.12.

Let us consider how the states of this minimal DFA correspond to the logic required to recognize the language.

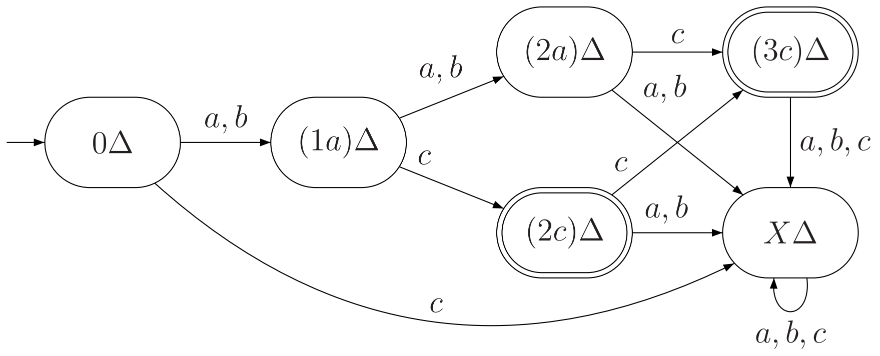


Figure 2.4.12: Minimization of DFA of Figure 2.4.10.

- State 0Δ is the initial state. We don't know yet what the word starts or ends with, so it's non-final.
- State $X\Delta$ is a failure state – once we reach it, we know the word is invalid, either because it is too long or because it does not satisfy the necessary conditions on the starting and ending letter.
- State $(1a)\Delta$ represents that we have read a valid starting letter (either a or b).
- State $(2a)\Delta$ represents that we have read a valid starting letter, but haven't yet read a valid ending letter.
- State $(2c)\Delta$ represents that we have read a valid starting letter *followed by* a valid ending letter, so it is accepting.
- State $(3c)\Delta$ represents that we have read a valid starting letter and a valid ending letter, so it is accepting. We can't combine this with state $(2c)\Delta$ even though they seem to keep track of similar things, because if we did, we would lose track of how many total letters have been read. ■

The syntactic monoid. Given a regular language L over Σ, we define the following congruence on Σ^*:

$$x \,\sigma_L\, y \iff \forall u, v \in \Sigma^*, uxv \in L \iff uyv \in L.$$

The quotient monoid Σ^*/σ_L is called the *syntactic monoid* of L. It turns out that this monoid is isomorphic to the transition monoid of the minimal DFA of L. There is deep algebraic theory surrounding syntactic monoids, but we are interested mainly in transition monoids in this thesis, and we define the syntactic monoid mostly to have a convenient shorthand for the unwieldy phrase “the transition monoid of the minimal DFA”.

Proposition 2.4.11. *Let $L \subseteq \Sigma^*$ be a regular language with minimal DFA $\mathcal{D}$. The syntactic monoid of L is isomorphic to the transition monoid of $\mathcal{D}$.*

Proof. For $\mathcal{D}$ we use the Myhill-Nerode construction described in Theorem 2.4.5. The states of $\mathcal{D}$ are equivalence classes of ρ_L. Thus the elements of the transition monoid are transformations $t\colon \Sigma^*/\rho_L \to \Sigma^*/\rho_L$. The syntactic monoid of L is Σ^*/σ_L. We must produce an isomorphism which maps equivalence classes of σ_L to transformations of equivalence classes of ρ_L.

Let φ be the map that sends $w\sigma_L$ to the transformation $z\rho_L \mapsto (zw)\rho_L$. We must prove this is well-defined, injective, surjective, and a homomorphism.

To see that it is well-defined, suppose $x\sigma_L = y\sigma_L$. We must show that for all classes $z\rho_L$, we have $(zx)\rho_L = (zy)\rho_L$. We know that for all $u, v \in \Sigma^*$, we have $uxv \in L \iff uyv \in L$, so in particular this holds for $u = z$; hence for all $v \in \Sigma^*$, we have $(zx)v \in L \iff (zy)v \in L$, and so $(zx)\rho_L = (zy)\rho_L$ as required.

To see that it is injective, suppose $x\sigma_L \neq y\sigma_L$. We must produce a class $z\rho_L$ such that $(zx)\rho_L \neq (zy)\rho_L$. Since $x\sigma_L \neq y\sigma_L$, there exist $u, v \in \Sigma^*$ such that $uxv \in L \iff uyv \notin L$. We may set $z = u$ and it follows that $(zx)\rho_L \neq (zy)\rho_L$.

To see that it is surjective, simply note that every transformation in the transition monoid has the form $z\rho_L \mapsto (z\rho_L)w$ for some $w \in \Sigma^*$. But $(z\rho_L)w = (zw)\rho_L$ by definition.

Finally, to see that φ is a homomorphism, we must show that $(x\sigma_L)\varphi(y\sigma_L)\varphi = (xy)\sigma_L\varphi$. Note that $(x\sigma_L)\varphi$ is the transformation $z\rho_L \mapsto (zx)\rho_L$, and $(y\sigma_L)\varphi$ is the transformation $z\rho_L \mapsto (zy)\rho_L$. Composing these gives $z\rho_L \mapsto (zxy)\rho_L$, which is precisely $(xy)\sigma_L\varphi$. □

2.4.3 State Complexity

Definition. The *state complexity* of a regular language is the number of states in its minimal DFA. This is sometimes called *deterministic state complexity* elsewhere in the literature, to distinguish it from other measures such as *nondeterministic state complexity* (the number of states in a minimal NFA for the language). However, we do not consider other types of state complexity in this thesis. We write $\mathrm{sc}(L)$ for the state complexity of a regular language L.

Computing state complexity. To determine the state complexity of a particular language, we must count the number of states in its minimal DFA. The minimization procedure described in Theorem 2.4.9 shows that it is sufficient to find an *arbitrary* DFA for the language, count the number of reachable states in this DFA, and then count the number of indistinguishability equivalence classes among the reachable states. This will be our standard technique for computing state complexity. The other minimization theorems in Section 2.4.2 give two additional methods of computing state complexity: by Theorem 2.4.5, we can count the equivalence classes of the Myhill-Nerode relation ρ_L, and by Theorem 2.4.7, we can count the number of left quotients of the language by words.

State complexity of operations. A *regular operation* of arity k (or k*-ary regular operation*) is a function that takes k regular languages as input and outputs a regular language. This thesis is about methods for determining the *state complexity of regular operations*.

What do we mean by the state complexity of an *operation*, as opposed to a language? This is similar to the notion of *space complexity* of a computable function, a fundamental topic in computer science. Space complexity is a measure of how much memory a computer needs to compute the function in the worst case. Specifically, the space complexity is the worst-case amount of memory needed to compute the function, expressed in terms of the amount of memory needed to store the inputs to the function. Similarly, the state complexity of an operation is the maximal possible state complexity of the result of the operation, expressed in terms of the state complexities of the operands.

More formally, the *state complexity of an k-ary regular operation* Φ is the following function, which takes k positive integers as input and outputs a positive integer:

$$(n_1, n_2, \dots, n_k) \mapsto \max\{\mathrm{sc}((L_1, L_2, \dots, L_k)\Phi) : \mathrm{sc}(L_i) \leq n_i, 1 \leq i \leq k\}.$$

The inputs to the function do not represent the exact state complexities of the operands, but rather *upper bounds* on the state complexities of the operands. The state complexity of the operation is computed by taking the maximum over all operand languages whose state complexity is bounded by the specified values. The statement "$\mathrm{sc}(L_i) \leq n_i$" is equivalent to "$L_i$ is recognized by an n_i-state DFA", so one may also view the input integers as specifying the numbers of states in the DFAs for the operands.

Restricted and unrestricted state complexity. The definition of state complexity of operations is significantly affected by one's choice between the restricted and unrestricted viewpoints (introduced in Section 2.2). When using the restricted viewpoint, if we are working with an operation of arity two or higher, we assume that all operands *share the same alphabet*. This is justified by the fact that in the restricted viewpoint, words over different alphabets are necessarily distinct; hence, for example, the union of two languages over different alphabets would be a set containing a mixture of words over different alphabets, which is not a language.

To perform a multiary operation on languages over different alphabets, one must first convert the operands to languages over a common alphabet. This model of state complexity (where we assume operands have the same alphabet, and convert them to languages over the same alphabet if not) is called *restricted state complexity*. When using the unrestricted viewpoint, there is no issue with performing operations on languages over different alphabets, and so we consider a larger class of operands when computing the state complexity of an operation. This model of state complexity (where operands may have different alphabets, and no conversion is needed) is called *unrestricted state complexity*.

The notions of restricted and unrestricted state complexity were introduced by Brzozowski in 2016 [7]. Prior to Brzozowski's work, it was standard in the state complexity literature to use restricted state complexity. Brzozowski showed that restricted state complexity actually leads to incorrect state complexity bounds when applied to languages over different alphabets, since converting the input languages to a common alphabet can change their state complexities. Unrestricted state complexity produces correct bounds for languages over different alphabets. We consider mainly restricted state complexity in this thesis, but will occasionally discuss unrestricted state complexity.

2.4.4 Automaton Constructions

To determine the state complexity of an operation, we need to find the maximal possible state complexity of the result of the operation. If we can find a generic DFA construction for the result of the operation, the size of this DFA gives an upper bound on the maximal state complexity. We can also obtain lower bounds by choosing particular languages, applying the operation, and constructing the minimal DFA of the result. For both upper and lower bounds, it is useful to know how to construct a DFA that describes the result of a regular operation.

However, it is not always obvious how to do these constructions. For example, suppose we have two regular languages L and L'. How can we construct a DFA for the union $L \cup L'$? Does a DFA even exist in all cases? If we figure out how to construct an FA for the union, can we convert it to a DFA? Questions like these have been studied heavily since the beginning of the theory of automata. In this section, we describe automaton constructions for some commonly used regular operations.

Determinization: the subset construction. The *subset construction* allows one to convert an arbitrary FA to a DFA. The idea is to construct a DFA which has one state for each *subset* of states in the FA. Let $\mathcal{A} = (Q, \Sigma, T, I, F)$. Define $\mathcal{D} = (\mathcal{P}(Q), \Sigma, T^{\mathcal{D}}, I, F^{\mathcal{D}})$ by setting $T^{\mathcal{D}} = \{(S, a, ST_a) : S \in \mathcal{P}(Q), a \in \Sigma\}$ and $F^{\mathcal{D}} = \{S \in \mathcal{P}(Q) : S \cap F \neq \emptyset\}$. We refer to $\mathcal{D}$ as the *determinization* of $\mathcal{A}$.

Example 2.4.12 (Determinization of an FA)**.** Let $\mathcal{A} = (Q, \Sigma, T, I, F)$ be the finite automaton shown in Figure 2.4.13.

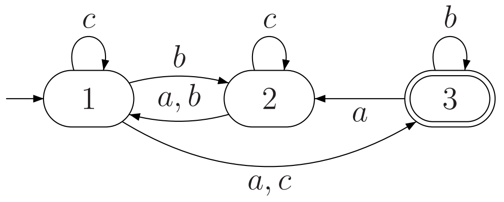


Figure 2.4.13: A finite automaton $\mathcal{A}$.

Observe that the letter actions of a and b are just $a = (3, 2, 1)$ and $b = (1, 2)$. The letter action of c is the following element-to-set map:

$$c = \begin{pmatrix} 1 & 2 & 3 \\ \{1,3\} & \{2\} & \emptyset \end{pmatrix}$$

Using these letter actions, we construct a transition table for the determinization. We will use a determinization algorithm that computes only the *reachable* states of the determinization. In general this saves some work, although in this particular case we will see that all states in $\mathcal{P}(Q)$ are reachable. We start with just one row, for the initial state $\{1\}$, and add new rows as we reach new states.

S	a	b	c
$\{1\}$	$\{3\}$	$\{2\}$	$\{1,3\}$

We have reached three new states: $\{2\}$, $\{3\}$ and $\{1, 3\}$. It is straightforward to compute $\{2\}\sigma$ and $\{3\}\sigma$ for $\sigma \in \Sigma$. To compute $\{1, 3\}\sigma$, we apply the letter action σ to *both* states of the set $\{1, 3\}$, and take the union. For $\{1, 3\}c$, we have $1c = \{1, 3\}$ and $3c = \emptyset$, so $\{1, 3\}c = \{1, 3\} \cup \emptyset = \{1, 3\}$. For $\{1, 3\}a$ and $\{1, 3\}b$ we simply apply the permutation to each element of the set.

S	a	b	c
$\{1\}$	$\{3\}$	$\{2\}$	$\{1,3\}$
$\{2\}$	$\{1\}$	$\{1\}$	$\{2\}$
$\{3\}$	$\{2\}$	$\{3\}$	$\emptyset$
$\{1,3\}$	$\{2,3\}$	$\{2,3\}$	$\{1,3\}$

The new states we have reached are $\emptyset$ and $\{2,3\}$. The empty set is simply fixed by every letter action, and we compute $\{2,3\}\sigma$ similarly to how we handled $\{1,3\}\sigma$.

S	a	b	c
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\{1\}$	$\{3\}$	$\{2\}$	$\{1,3\}$
$\{2\}$	$\{1\}$	$\{1\}$	$\{2\}$
$\{3\}$	$\{2\}$	$\{3\}$	$\emptyset$
$\{1,3\}$	$\{2,3\}$	$\{2,3\}$	$\{1,3\}$
$\{2,3\}$	$\{1,2\}$	$\{1,3\}$	$\{2\}$

We get one new state, $\{1,2\}$, on this iteration.

S	a	b	c
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\{1\}$	$\{3\}$	$\{2\}$	$\{1,3\}$
$\{2\}$	$\{1\}$	$\{1\}$	$\{2\}$
$\{3\}$	$\{2\}$	$\{3\}$	$\emptyset$
$\{1,2\}$	$\{1,3\}$	$\{1,2\}$	$\{1,2,3\}$
$\{1,3\}$	$\{2,3\}$	$\{2,3\}$	$\{1,3\}$
$\{2,3\}$	$\{1,2\}$	$\{1,3\}$	$\{2\}$

Now we have reached $\{1,2,3\}$.

S	a	b	c
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\{1\}$	$\{3\}$	$\{2\}$	$\{1,3\}$
$\{2\}$	$\{1\}$	$\{1\}$	$\{2\}$
$\{3\}$	$\{2\}$	$\{3\}$	$\emptyset$
$\{1,2\}$	$\{1,3\}$	$\{1,2\}$	$\{1,2,3\}$
$\{1,3\}$	$\{2,3\}$	$\{2,3\}$	$\{1,3\}$
$\{2,3\}$	$\{1,2\}$	$\{1,3\}$	$\{2\}$
$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$

There are no new reached states, so the determinization algorithm terminates. We have found all the reachable states (in this case, every set in $\mathcal{P}(Q)$ is reachable) and computed the transitions between them. The determinized automaton is shown in Figure 2.4.14. Since state 3 was final in the original FA, the final states of the determinization are all the sets that contain 3.

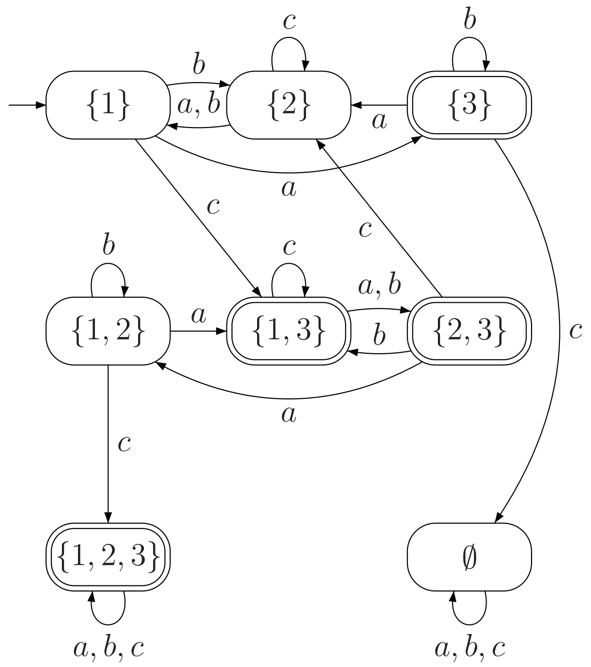


Figure 2.4.14: The determinization of the finite automaton $\mathcal{A}$.

As we see from Figure 2.4.14, in some cases the determinization of an FA can be significantly more complicated than the original and have many more states (up to 2^n if the original automaton has n states). Nonetheless, determinization is incredibly useful. Several of the DFA constructions for operations in this section are done by first constructing an FA, and then determinizing it to obtain a DFA. ■

Proposition 2.4.13. *The determinization $\mathcal{D}$ as defined above is deterministic, and $L(\mathcal{D}) = L(\mathcal{A})$.*

Proof. It is clear that $\mathcal{D}$ is deterministic since there is exactly one transition for each state-letter pair. Also, since $IT_w = IT_w^{\mathcal{D}}$, we have

$$w \in L(\mathcal{A}) \iff IT_w \cap F \neq \emptyset \iff IT_w \in F^{\mathcal{D}} \iff IT_w^{\mathcal{D}} \in F^{\mathcal{D}} \iff w \in L(\mathcal{D}),$$

and so $L(\mathcal{D}) = L(\mathcal{A})$ as required. □

Reversal. Given a word $w = a_1 \cdots a_k$, with $w \in \Sigma^*$ and $a_1, \ldots, a_k \in \Sigma$, define the *reverse* of w to be $w^R = a_k \cdots a_1$. The reverse of a *language* L is $L^R = \{w^R : w \in L\}$. The *reverse automaton* is a DFA used to recognize the reverse of a regular language.

Let L be a regular language with FA $\mathcal{A} = (Q, \Sigma, T, I, F)$. We construct a DFA $\mathcal{R}$ for L^R as follows. First construct an FA $(Q, \Sigma, T^{\mathcal{R}}, F, I)$ where $T^{\mathcal{R}} = \{(q, a, p) : (p, a, q) \in T\}$. Essentially, this FA is obtained by reversing the transitions of $\mathcal{A}$ and swapping the initial and final state sets. This FA recognizes L^R. Now let $\mathcal{R}$ be the determinization of this FA.

Example 2.4.14 (Reversal of a DFA)**.** Let $\mathcal{A} = (Q, \Sigma, T, 1, F)$ be the deterministic automaton shown in Figure 2.4.15.

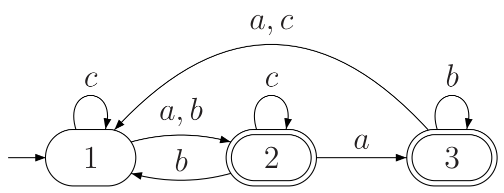


Figure 2.4.15: A deterministic automaton $\mathcal{A}$.

The FA used as an intermediate step to construct the reverse automaton $\mathcal{R}$ is shown in Figure 2.4.16. It is obtained by swapping the roles of the initial and final states, and reversing the transition arrows.

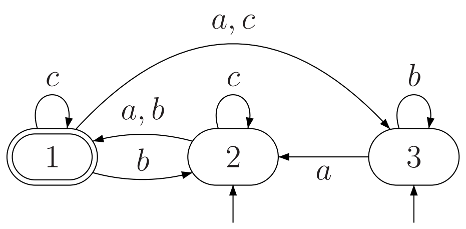


Figure 2.4.16: FA recognizing the reverse of $L(\mathcal{A})$.

Notice that this FA is has the same transition structure as the FA of Figure 2.4.13, but different initial and final states. Thus the reverse automaton $\mathcal{R}$, obtained by determinizing this FA, will look like the DFA of Figure 2.4.14 but with different initial state and final states.

Let us pick a word in $L(\mathcal{A})$, and verify that its reverse is accepted by the FA. We choose *bacaba*. In the reverse FA we have the following path from the initial state 2 to the final state 1 with label $(bacaba)^R = abacab$:

$$2 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{a} 1 \xrightarrow{c} 3 \xrightarrow{a} 2 \xrightarrow{b} 1.$$

So the reverse automaton accepts *abacab*. ■

Proposition 2.4.15. *Let $\mathcal{A} = (Q, \Sigma, T, I, F)$ be an FA. For the reverse automaton $\mathcal{R}$ of $\mathcal{A}$, we have $L(\mathcal{R}) = L(\mathcal{A})^R$.*

Proof. First we prove that $T_w^{\mathcal{R}} = T_{w^R}^{-1}$ for all $w \in \Sigma^*$. This is clear for $w = \varepsilon$. For $a \in \Sigma$ we have $T_a = \{(p, q) : (p, a, q) \in T\}$, and thus

$$T_a^{\mathcal{R}} = \{(q, p) : (p, a, q) \in T\} = \{(q, p) : (p, q) \in T_a\} = T_a^{-1} = T_{a^R}^{-1}.$$

Then if $w = a_1 \cdots a_k$, we have $T_w^{\mathcal{R}} = T_{a_1}^{-1} \cdots T_{a_k}^{-1} = (T_{a_k} \cdots T_{a_1})^{-1} = T_{w^R}^{-1}$.

Now, note that the initial state of $\mathcal{R}$ is the set F, and the final state set is $\{S \in \mathcal{P}(Q) : S \cap I \neq \emptyset\}$. Thus we have

$$\begin{aligned} w \in L(\mathcal{R}) \iff FT_w^{\mathcal{R}} \cap I \neq \emptyset &\iff FT_{w^R}^{-1} \cap I \neq \emptyset \\ &\iff \exists f \in F,\ fT_{w^R}^{-1} \in I \\ &\iff \exists f \in F,\ f \in IT_{w^R} \\ &\iff IT_{w^R} \cap F \neq \emptyset \iff w^R \in L(\mathcal{A}). \end{aligned}$$

Thus $L(\mathcal{R}) = L(\mathcal{A})^R$ as required. □

Concatenation. The *concatenation automaton* recognizes the concatenation of two regular languages. Let L' and L be regular languages with DFAs $\mathcal{A}' = (Q', \Sigma', T', 1', F')$ and $\mathcal{A} = (Q, \Sigma, T, 1, F)$, respectively. (This construction can be done with arbitrary FAs, but we will only use it with DFAs, and using DFAs makes it a bit simpler to describe.) We construct a DFA $\mathcal{C}$ for the concatenation $L'L = \{xy : x \in L', y \in L\}$ as follows. First define an FA $(Q' \cup Q, \Sigma' \cup \Sigma, T^{\mathcal{C}}, I^{\mathcal{C}}, F)$, where:

- $T^{\mathcal{C}} = T' \cup T \cup \{(q', a, 1) : q' \in Q', q'a \in F', a \in \Sigma\}$.
- $I^{\mathcal{C}} = \{1'\}$ if $1' \notin F'$, and otherwise $I^{\mathcal{C}} = \{1', 1\}$.

We define $\mathcal{C}$ be the determinization of this FA.

While the state set of $\mathcal{C}$ is $\mathcal{P}(Q' \cup Q)$, the structure of the DFA means that a reachable state can only contain at most one state of Q'. Thus instead of writing the states of $\mathcal{C}$ as sets, we write them as *ordered pairs* (q', S) with $q' \in Q'$ and $S \subseteq Q$ (or $(\emptyset, S)$ for sets which contain no elements of Q'). In other words, we are treating the state set as if it was $Q' \times \mathcal{P}(Q)$.

Here is some intuition behind this construction. A word in the concatenation $L'L$ has the form xy, with $x \in L'$ and $y \in L$. Let's call the position in the word where y begins the “split point”. Now, suppose our automaton $\mathcal{C}$ is given some word w. The automaton should use $\mathcal{A}'$ to read the first part of w (before the split point) and check if this part belongs to L', and then use $\mathcal{A}$ to check that the remaining part of the word (after the split point) belongs to L. The problem is that the automaton has no way to know where the split point is! It could be at any position in the word, or it may not even exist if there is no way to split up the word into a part from L' followed by a part from L.

The key insight is that we can use nondeterminism to *guess* where the split point is. This is how our automaton works. It starts reading the input word using the automaton $\mathcal{A}'$, up until the point where a final state is reached. At this point, the automaton has read a complete word from L', so it knows that the split point *could* be at the current position. So the computation splits off into two *branches*: a “stay” branch that stays within the automaton $\mathcal{A}'$, and a “split” branch takes the appropriate transition of the form $(q', a, 1)$, leading to the automaton $\mathcal{A}$. Now, if this “split” branch reaches a final state in $\mathcal{A}$, this means the split point was guessed correctly, and the word really can be split into a part from L' and a part from L; thus the word is accepted. It’s possible that this never happens, and then the split branch just dies off once the word is fully processed. Meanwhile, the “stay” branch keeps processing the word in $\mathcal{A}'$, looking for new places where a split point might occur, and spawning new “split” branches as necessary.

This process of guessing and checking split points allows the automaton $\mathcal{C}$ to recognize the concatenation. The states of $\mathcal{C}$ have the form $\{q'\} \cup S$, where q' is the state of the “stay” branch, and S keeps track of which states are occupied by the different “split” branches.

Example 2.4.16 (Concatenation of two DFAs)**.** Let $\mathcal{A}'$ and $\mathcal{A}$ be the two automata shown in Figure 2.4.17.

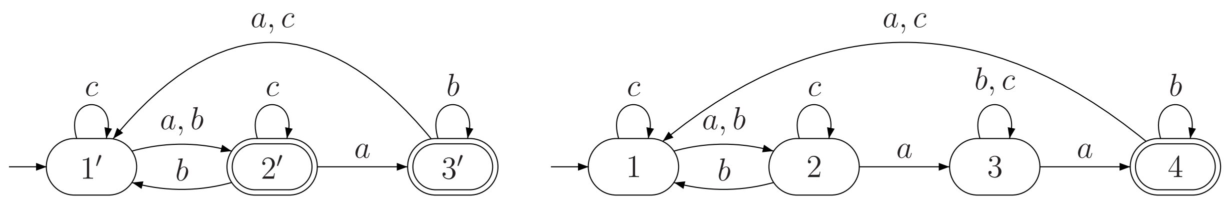


Figure 2.4.17: Deterministic automata $\mathcal{A}'$ (left) and $\mathcal{A}$ (right).

The FA used as an intermediate step to construct the concatenation automaton $\mathcal{C}$ is shown in Figure 2.4.18. We take copies of the DFAs $\mathcal{A}'$ and $\mathcal{A}$ and join them with new transitions. For each transition $p \xrightarrow{\sigma} q$ such that q is a final state, we add a new transition from p to the initial state 1 of $\mathcal{A}$:

- Since $1' \xrightarrow{a} 2'$ and $1' \xrightarrow{b} 2'$, and $2'$ is final, we add transitions $1' \xrightarrow{a} 1$ and $1' \xrightarrow{b} 1$.
- Since $2' \xrightarrow{a} 3'$ and $2' \xrightarrow{c} 2'$, and $2'$ and $3'$ are both final, we add transitions $2' \xrightarrow{a} 1$ and $2' \xrightarrow{c} 1$.
- Since $3' \xrightarrow{b} 3'$ and $3'$ is final, we add a transition $3' \xrightarrow{b} 1$.

After adding these new transitions, we make state 1 non-initial and make states $2'$ and $3'$ non-final.

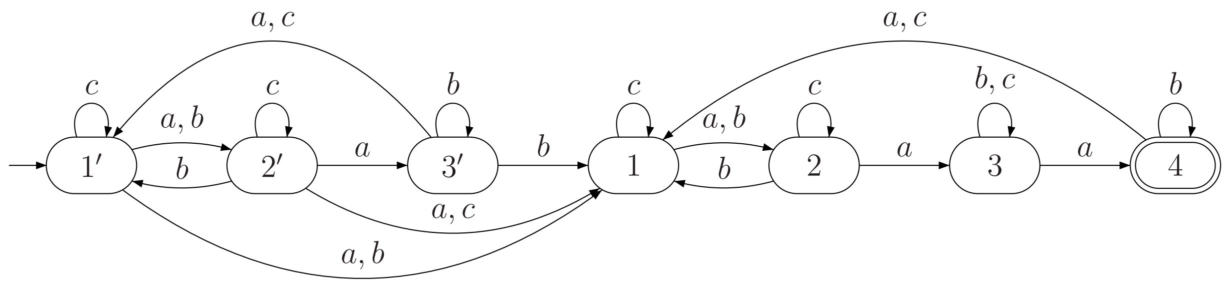


Figure 2.4.18: FA for the concatenation $L(\mathcal{A}')L(\mathcal{A})$.

The determinization of this FA is rather large, so we will not compute it. But let us verify that the word *acabbaca* is accepted by this FA. The word *acab* is accepted by $\mathcal{A}'$, and the word *baca* is accepted by $\mathcal{A}$, so *acabbaca* should be in the concatenation. Indeed, we have the following path in our FA:

$$1' \xrightarrow{a} 2' \xrightarrow{c} 2' \xrightarrow{a} 3' \xrightarrow{b} 1 \xrightarrow{b} 2 \xrightarrow{a} 3 \xrightarrow{c} 3 \xrightarrow{a} 4,$$

with state $1'$ initial and state 4 final as required.

To illustrate the "nondeterministic branching" process we described, let use choose a shorter word to verify: *abaa*. The automaton verifies that this word belongs to the concatenation as follows:

- We begin in state $1'$ with a single branch: the "stay" branch.
- Process a: the "stay" branch goes to state $2'$. Since $2'$ is final, we spawn a "split" branch in state 1.
- Process b: the "stay" branch goes to state $1'$ and the "split" branch to state 2.

- Process a: the "stay" branch goes to state $2'$ and the "split" branch to state 3. Since $2'$ is final, we spawn a second "split" branch in state 1.
- Process a: the "stay" branch goes to state $2'$, the first "split" branch to state 4, and the second "split" branch to state 2. Since $2'$ is final, we spawn a third "split" branch in state 1.
- We have reached the end of the word. Since the first "split" branch is in the final state 4, we accept the word; on this branch, the split point was correctly guessed to be after the first letter. ■

Now, we prove formally that the construction works.

Proposition 2.4.17. *Let $\mathcal{A}' = (Q', \Sigma', T', 1', F')$ and $\mathcal{A} = (Q, \Sigma, T, 1, F)$ be DFAs, and set $L' = L(\mathcal{A}')$ and $L = L(\mathcal{A})$. Let $\mathcal{C}$ is the concatenation automaton constructed from these DFAs. We have $L(\mathcal{C}) = L'L$.*

Proof. First we show that $L(\mathcal{C}) \subseteq L'L$. Fix $w \in L(\mathcal{C})$. If $w = \varepsilon$, then we must have $I^{\mathcal{C}} = \{1', 1\}$ and $1 \in F$. So $\varepsilon \in L$. But since $I^{\mathcal{C}} = \{1', 1\}$ we have $1' \in F'$; thus $\varepsilon \in L'$. So $w = \varepsilon \in L'L$.

Suppose w is nonempty and write $w = a_1 \cdots a_k$. Since w is accepted by $\mathcal{C}$, there exists a sequence of transitions $q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{k-1}} q_k \xrightarrow{a_k} q_{k+1}$, where $q_1, \ldots, q_{k+1} \in Q \cup Q'$, $q_1 \in I^{\mathcal{C}}$, and $q_{k+1} \in F$.

If $q_1 = 1$, then $1w \in F$ and thus $w \in L$. Also, by definition $1' \in F'$ and so $\varepsilon \in L'$. It follows $w = \varepsilon w \in L'L$.

Suppose $q_1 = 1'$. Note that there are no transitions that lead from a state of $\mathcal{A}$ to a state of $\mathcal{A}'$. Also, the only transitions that lead from $\mathcal{A}'$ to $\mathcal{A}$ are those of the form $(q', a, 1)$ with $q' \in Q'$ and $q'a \in F'$ in $\mathcal{A}'$. Thus we can partition the aforementioned sequence of transitions into three sections: the "left" which consists only of transitions of $\mathcal{A}'$, the "right" which consists only of transitions of $\mathcal{A}$, and the "bridge" which is the unique transition in the sequence that leads from $\mathcal{A}'$ to $\mathcal{A}$.

Suppose $q_i \xrightarrow{a_i} q_{i+1}$ is the "bridge"; by definition we have $q_i \in Q'$, $q_i a_i \in F'$ in $\mathcal{A}'$, and $q_{i+1} = 1$. The "left" is the sequence $q_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{i-1}} q_i$ and the "right" is the sequence $q_{i+1} \xrightarrow{a_{i+1}} \cdots \xrightarrow{a_k} q_{k+1}$. Now, since $q_i a_i \in F'$, it follows that $1'a_1 \cdots a_i \in F'$ in $\mathcal{A}'$, and thus $a_1 \cdots a_i \in L'$. Also, since $q_{i+1} = 1$ we have $1a_{i+1} \cdots a_k = q_{k+1} \in F$, and thus $a_{i+1} \cdots a_k \in L$. Hence $w \in L'L$.

This proves that $L(\mathcal{C}) \subseteq L'L$. For the other containment, fix $w \in L'L$. Write $w = xy$ where $x \in L'$ and $y \in L$. If $x = \varepsilon$, then $\varepsilon \in L'$ and thus $1' \in F'$. Thus $I^{\mathcal{C}} = \{1', 1\}$ and since $1y \in F$ we have $I^{\mathcal{C}} y \cap F \neq \emptyset$ in $\mathcal{C}$; it follows that $w = \varepsilon y = y \in L(\mathcal{C})$.

If $x \neq \varepsilon$, write $x = za$ where $z \in \Sigma^*$ and $a \in \Sigma$. Since $1'za \in F'$, there is a transition $(1'z, a, 1)$ in $T^{\mathcal{C}}$. Thus in $\mathcal{C}$ we have transitions $1' \xrightarrow{z} 1'z \xrightarrow{a} 1 \xrightarrow{y} 1y$, with $1y \in F$ since $y \in L$. It follows that $w = zay \in L(\mathcal{C})$. Thus $L'L = L(\mathcal{C})$ as required. □

Kleene star. The *star automaton* is used to recognize the star of a regular language. The star of L is the language $L^* = \{\varepsilon\} \cup L \cup L^2 \cup L^3 \cup \cdots$, where L^2 is the concatenation LL, L^3 is the concatenation LLL, and so on. Let L be a regular language with DFA $\mathcal{A} = (Q, \Sigma, T, I, F)$ (again the construction can be made to work with arbitrary FAs, but we will only use it with DFAs). We construct a DFA $\mathcal{S}$ for L^* as follows. Assume $I = \{1\}$. Define an FA $(Q \cup \{s\}, \Sigma, T^{\mathcal{S}}, \{s\}, F \cup \{s\})$, where $T^{\mathcal{S}} = T \cup \{(s, a, 1a) : a \in \Sigma\} \cup \{(q, a, 1) : q \in Q \cup \{s\}, qa \in F, a \in \Sigma\}$. Define $\mathcal{S}$ be the determinization of this FA.

Since star is an iterated version of concatenation, the intuition behind the construction is very similar. Words can now have multiple "split points" instead of at most one, but we can use the same idea of nondeterministically "guessing" split point locations and branching the computation on each guess. The main difference is now instead of jumping to a second automaton on each "split" branch, we just have one automaton and we jump back to its initial state. This means split branches can themselves spawn new split branches upon reaching a final state! Fortunately, keeping track of all these branches is not difficult since all we care about is what state of the automaton a particular branch is in. If two branches occupy the same state, they are functionally equivalent and will have identical behavior moving forward. Thus we can just use sets of states to keep track of all the branches.

Since the star of a language always contains the empty word ε, the star automaton also needs an special initial state s to guarantee that ε is accepted. The state s behaves exactly the same as the original initial state 1 in terms of transitions, but it is final and thus accepts ε.

Example 2.4.18 (Star of a DFA)**.** Let $\mathcal{A}$ be the DFA of Figure 2.4.19.

We construct the FA for the star. For each transition leading into a final state, we add a transition back to the initial state:

- Since $1 \xrightarrow{a} 2$ and $1 \xrightarrow{b} 2$, and 2 is final, we add transitions $1 \xrightarrow{a} 1$ and $1 \xrightarrow{b} 1$.
- Since $2 \xrightarrow{a} 3$ and $2 \xrightarrow{c} 2$, and 2 and 3 are both final, we add transitions $2 \xrightarrow{a} 1$ and $2 \xrightarrow{c} 1$.

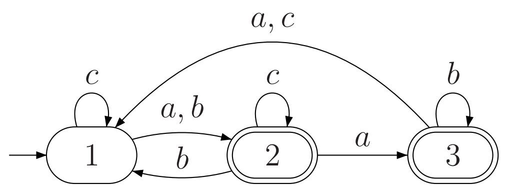


Figure 2.4.19: A deterministic automaton $\mathcal{A}$.

- Since $3 \xrightarrow{b} 3$ and 3 is final, we add a transition $3 \xrightarrow{b} 1$.

Then we add the new initial-final state s and the transitions $s \xrightarrow{\sigma} 1\sigma$ for each $\sigma \in \Sigma$. Since $s \xrightarrow{a} 2$ and $s \xrightarrow{b} 2$, and 2 is final, we add transitions $s \xrightarrow{a} 1$ and $s \xrightarrow{b} 1$. See Figure 2.4.20.

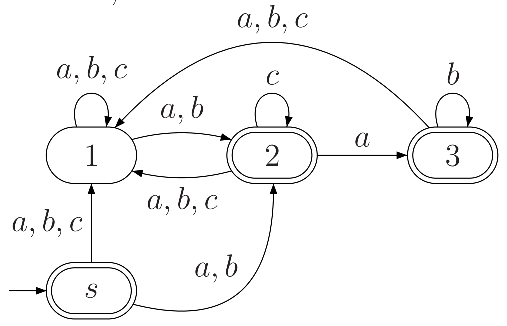


Figure 2.4.20: FA for the star $L(\mathcal{A})^*$.

To obtain the star DFA $\mathcal{S}$, we take the determinization. Using the algorithm from Example 2.4.12, we obtain the following transition table:

S	a	b	c
$\{s\}$	$\{1,2\}$	$\{1,2\}$	$\{1\}$
$\{1\}$	$\{1,2\}$	$\{1,2\}$	$\{1\}$
$\{1,2\}$	$\{1,2,3\}$	$\{1,2\}$	$\{1,2\}$
$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2\}$

The star automaton $\mathcal{S}$ is shown in Figure 2.4.21.

Notice that this final DFA is not minimal; states $\{1,2\}$ and $\{1,2,3\}$ are both final, and we can never move from one of these states to a non-final state, and thus these states are indistinguishable (they both accept the language Σ^*). We can merge these states to obtain a minimal DFA for the star; see Figure 2.4.22.

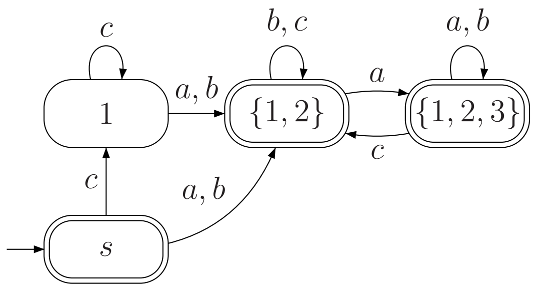


Figure 2.4.21: Star automaton $\mathcal{S}$ for the DFA $\mathcal{A}$.

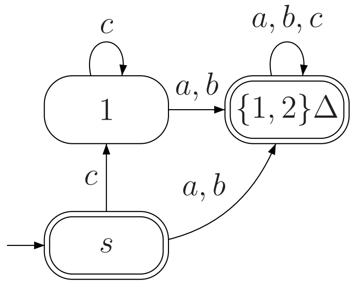


Figure 2.4.22: Minimized star automaton for the DFA $\mathcal{A}$.

It is clear now that $\mathcal{S}$ accepts the following language:

$$L = \{w \in \Sigma^* : w \text{ is either empty, or contains at least one } a \text{ or } b\}.$$

Is this really the star of the language of $\mathcal{A}$? The language of $\mathcal{A}$ is difficult to describe, but we can see that it contains a and b, and thus the star contains all words in $\{a, b\}^*$. In fact, the language of $\mathcal{A}$ contains all words of the form ac^k and bc^k for $k \geq 0$. An arbitrary word in the language L can be split into blocks of the form $w\sigma c^k$, where w is a word over $\{a, b\}^*$ and $\sigma \in \{a, b\}$. Therefore, every word in L can be written as a concatenation of words in $L(\mathcal{A})$, that is, $L \subseteq L(\mathcal{A})^*$. Also, every word accepted by $\mathcal{A}$ contains at least one a or b, so $L(\mathcal{A})^* \subseteq L$. Thus the language of $\mathcal{S}$ is indeed the star of the language of $\mathcal{A}$. ■

Proposition 2.4.19. *Let $\mathcal{A} = (Q, \Sigma, T, 1, F)$ be a DFA with language L and let $\mathcal{S}$ be the star automaton of $\mathcal{A}$. We have $L(\mathcal{S}) = L^*$.*

Proof. First we show that $L(\mathcal{S}) \subseteq L^*$. Fix $w \in L(\mathcal{S})$. If $w = \varepsilon$, then $w \in L^*$, so suppose w is non-empty. We will call a transition of $T^{\mathcal{S}}$ a *start* transition if it has the form $(s, a, 1a)$, and a *split* transition if it has the form $(q, a, 1)$ with $qa \in F$. Note that if a transition is *not* a split transition, it must lie in T. However, T might contain split transitions if $1 \in F$.

Now, we prove the following statement: let $q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{k-1}} q_k$ be a sequence of transitions in $T^{\mathcal{S}}$ such that $q_1 \in \{s, 1\}$ and there are ℓ split transitions. If T contains a transition $q_k \xrightarrow{a_k} q$ with $q \in F$, then $a_1 \cdots a_k \in L^{\ell+1}$.

We proceed by induction on ℓ. If $\ell = 0$, then every transition in our sequence except possibly the first lies in T. The only way we can have $q_1 \xrightarrow{a_1} q_2$ not in T is if $q_1 = s$. But then $q_1 \xrightarrow{a_1} q_2$ is a start transition, and thus we have a corresponding transition $1 \xrightarrow{a_1} q_2$ in T. So in $\mathcal{A}$ we have $1a_1 \cdots a_{k-1} = q_k$. Since the transition $q_k \xrightarrow{a_k} q$ is also in T, in fact we have $1a_1 \cdots a_k = q \in F$. Thus $a_1 \cdots a_k \in L = L^1$.

If $\ell > 0$, assume the statement holds for sequences with fewer than ℓ split transitions. Let $q_j \xrightarrow{a_j} q_{j+1}$ be the last split transition in our sequence. Then the sequence $q_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{j-1}} q_j$ has fewer than ℓ split transitions. Furthermore, in $\mathcal{A}$ we have $q_ja_j \in F$, so T contains a transition $q_j \xrightarrow{a_j} q$ with $q \in F$. By induction we have $a_1 \cdots a_j \in L^\ell$. Now, note that since $q_j \xrightarrow{a_j} q_{j+1}$ is a split transition, we have $q_{j+1} = 1$. Also, by assumption T contains a transition $q_k \xrightarrow{a_k} q$ with $q \in F$. Thus the sequence $q_{j+1} \xrightarrow{a_{j+1}} \cdots \xrightarrow{a_{k-1}} q_k$, which contains no split transitions, also satisfies the induction assumption. It follows that $a_{j+1} \cdots a_k \in L$. Hence $a_1 \cdots a_k \in L^{\ell+1}$.

With this statement proved, we can show that $L(\mathcal{S}) \subseteq L^*$. If $w \in L(\mathcal{S})$ with $w = a_1 \cdots a_k$, we must have a sequence of transitions $q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{k-1}} q_k \xrightarrow{a_k} q_{k+1}$ with $q_1 = s$ and $q_{k+1} \in F$ and finitely many split transitions. As long as the final transition $q_k \xrightarrow{a_k} q_{k+1}$ lies in T, we can use the above statement to get $w \in L^*$. So suppose that $q_k \xrightarrow{a_k} q_{k+1}$ does not lie in T. Then it must be a split transition $q_k \xrightarrow{a_k} 1$ such that $q_ka_k \in F$ in $\mathcal{A}$. But this means that T contains some transition $q_k \xrightarrow{a_k} q$ with $q \in F$; so we can replace the final transition in our sequence with a transition from T, and then apply the statement. In any case, we have $w \in L^*$.

Next we show that $L^* \subseteq L(\mathcal{S})$. Fix $w \in L^*$. If $w = \varepsilon$ then it is accepted by $L(\mathcal{S})$. Otherwise, for some k we can write $w = w_1 \cdots w_k$ with $w_1, \ldots, w_k \in L$. Assume without loss of generality that each w_i is nonempty and write $w_i = x_ia$, where $x_i \in \Sigma^*$ and $a \in \Sigma$. Then since $x_ia \in L$ for all i, in $\mathcal{A}$ we have $1x_ia_i \in F$ for all i. Thus in $\mathcal{S}$ there are split transitions $1x_i \xrightarrow{a_i} 1$ for all i. Also, there is a transition $1x_k \xrightarrow{a_k} q$ for some $a_k \in \Sigma$ and $q \in F$. It follows that in $\mathcal{S}$ we have a sequence of transitions

$$s \xrightarrow{x_1} 1x_1 \xrightarrow{a_1} 1 \xrightarrow{x_2} 1x_2 \xrightarrow{a_2} 1 \xrightarrow{x_3} \cdots \xrightarrow{a_{k-1}} 1 \xrightarrow{x_k} 1x_k \xrightarrow{a_k} q,$$

and thus $w = x_1a_1x_2a_2 \cdots x_ka_k \in L(\mathcal{S})$. It follows that $L(\mathcal{S}) = L^*$. □

The state set of $\mathcal{S}$ is $\mathcal{P}(Q \cup \{s\})$, but only states of the following forms can be reached:

- The singleton $\{s\}$.
- Sets $S \subseteq Q \setminus F$.
- Sets $S \subseteq Q$ such that $S \cap F \neq \emptyset$ and $i \in S$.

We assume without loss of generality that the state set of $\mathcal{S}$ consists of only states of these forms.

Union, intersection, and other boolean operations. The *direct product* construction is used to recognize boolean operations on regular languages. For simplicity, we consider only binary boolean operations below, but the construction has a straightforward generalization to boolean operations of arbitrary arity.

Fix a function $\circ\colon \{0,1\}^2 \to \{0,1\}$; this is called a *binary boolean function*. For a set S, let χ_S denote the *characteristic function* of S, defined by $x\chi_S = 1$ if $x \in S$ and $x\chi_S = 0$ otherwise. (The inputs to χ_S are assumed to come from some "universal set" containing S.) We can think of $x\chi_S$ as giving the "truth value" of the proposition "$x \in S$", where 0 is false and 1 is true. For each binary boolean function $\circ$, we define a corresponding *binary boolean operation* $L' \circ L$ on languages $L \subseteq \Sigma^*$ and $L' \subseteq (\Sigma')^*$ as follows:

$$L' \circ L = \{w \in (\Sigma \cup \Sigma')^* : w\chi_{L'} \circ w\chi_L = 1\}.$$

For example, if $\circ\colon \{0,1\}^2 \to \{0,1\}$ is the "logical or" function, then $L' \circ L = L' \cup L$, since $w \in L' \circ L$ if $w \in L'$ or $w \in L$. Similarly, the "logical and" function gives the intersection $L' \cap L$.

Now, let L' and L be regular languages with DFAs $\mathcal{A}' = (Q', \Sigma', T', I', F')$ and $\mathcal{A} = (Q, \Sigma, T, I, F)$, respectively. Fix a binary boolean operation $\circ$. We construct a DFA $\mathcal{B} = \mathcal{A}' \times \mathcal{A}$ for $L' \circ L$ as follows. Define

$$\mathcal{B} = ((Q' \cup \{\emptyset\}) \times (Q \cup \{\emptyset\}), \Sigma' \cup \Sigma, T^{\mathcal{B}}, (1', 1), F' \circ F),$$

where $T^{\mathcal{B}}$ contains a transition $(q', q) \xrightarrow{a} (q'a, qa)$ for each $q \in Q \cup \{\emptyset\}$, $q' \in Q' \cup \{\emptyset\}$, and $a \in \Sigma' \cup \Sigma$, the initial state is $(1', 1)$, and the final state set is $F \circ F' = \{(q', q) \in (Q' \cup \{\emptyset\}) \times (Q \cup \{\emptyset\}) : q'\chi_{F'} \circ q\chi_F = 1\}$. If $\Sigma' = \Sigma$, then states with $\emptyset$ in one or both of the components are not reachable, and we can just take the state set to be $Q' \times Q$.

Intuitively, the idea behind this construction is to *run two automata in parallel* on the same input word, and accept depending on the results of this parallel computation. The states of the direct product are simply pairs of states from the original automata, and the transitions are given by following the transitions in the original automaton componentwise. (The $\emptyset$ marker is needed to account for the possibility that one of the automata has no transitions on a particular letter.) Acceptance is based on checking whether the states reached at the end of this parallel computation satisfy a particular boolean condition; for example, for intersection we want that *both* automata reached a final state, but for union we want that *at least one* reached a final state. For boolean operations of higher arity, we use the same idea but with a larger number of automata.

Example 2.4.20 (Direct product of DFAs)**.** Let us construct a DFA for the following language over $\Sigma = \{a, b\}$:

$$L = \{w \in \Sigma^* : w \text{ has an even number of } a\text{s and an odd number of } b\text{s}\}.$$

We already saw the minimal DFA for this language in Examples 2.4.6 and 2.4.8. However, we can also construct it by taking a direct product of DFAs for the following languages:

$$\{w \in \Sigma^* : w \text{ has an even number of } a\text{s}\} \text{ and } \{w \in \Sigma^* : w \text{ has an odd number of } b\text{s}\}.$$

DFAs $\mathcal{A}'$ and $\mathcal{A}$ for these languages are shown in Figure 2.4.23.

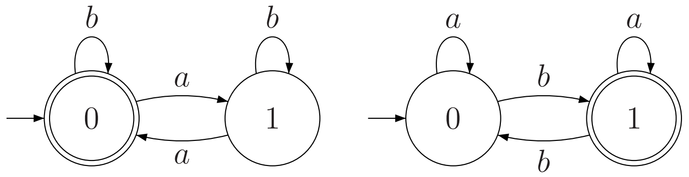


Figure 2.4.23: DFAs for the languages over $\{a, b\}$ of words an even number of as (left) and words with an odd number of bs (right).

The direct product of $\mathcal{A}'$ and $\mathcal{A}$ is shown in Figure 2.4.24. The states are pairs in $\{0, 1\} \times \{0, 1\}$. If we are in state (i, j), this means the parity of as is i modulo 2, and the parity of bs is j modulo 2. The initial state is the pair $(0, 0)$; the components are the initial states of the original DFAs. The final state set is $\{(0, 1)\}$, the unique pair which represents reaching a final state in both of the original DFAs simultaneously.

We can modify the final state set of this direct product to recognize other boolean operations. For example, suppose we are interested in the following language:

$$L = \{w \in \Sigma^* : w \text{ has an even number of } a\text{s } \textit{or} \text{ an odd number of } b\text{s}\}.$$

We just take the final state set to be the set of all pairs in which *one of* the components is final: this is the set $\{(0, 0), (0, 1), (1, 1)\}$. ■

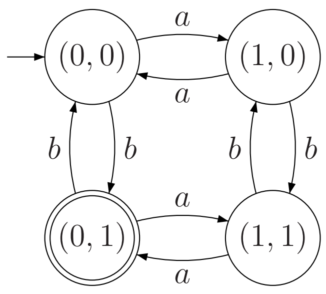


Figure 2.4.24: Direct product DFA for the language of words over $\{a, b\}$ with an even number of as and an odd number of bs.

Proposition 2.4.21. *Let $\mathcal{A}' = (Q', \Sigma', T', 1', F')$ and $\mathcal{A} = (Q, \Sigma, T, 1, F)$ be DFAs with languages L' and L respectively. Let $\circ$ be a binary boolean function, and let $\mathcal{B}$ be the direct product $\mathcal{A}' \times \mathcal{A}$ with final state set $F' \circ F$. The automaton $\mathcal{B}$ is deterministic, and $L(\mathcal{B}) = L' \circ L$.*

Proof. The automaton is deterministic because there is one transition for each state-letter pair. To see that $L(\mathcal{B}) = L' \circ L$, note that

$$1w\chi_F = 1 \iff 1w \in F \iff w \in L \iff w\chi_L = 1.$$

Thus $1w\chi_F = w\chi_L$ and similarly $1'w\chi_{F'} = w\chi_{L'}$. It follows that

$$w \in L(\mathcal{B}) \iff 1'w\chi_{F'} \circ 1w\chi_F = 1 \iff w\chi_{L'} \circ w\chi_L = 1 \iff w \in L' \circ L,$$

as required. □

Inverse homomorphism. Let $\varphi\colon \Gamma^* \to \Sigma^*$ be a homomorphism. Let L be a regular language with DFA $\mathcal{A} = (Q, \Sigma, T, 1, F)$. We may define a DFA recognizing the inverse homomorphism language $L\varphi^{-1}$ as follows. Define $\mathcal{A}\varphi^{-1} = (Q, \Gamma, T', 1, F)$ with $T' = \{(q, a, qT_{a\varphi}) : q \in Q, a \in \Sigma\}$.

Note that the inverse homomorphism DFA $\mathcal{A}\varphi^{-1}$ has the same state configuration as $\mathcal{A}$. The only difference is that the transitions have been "rewired". In particular, this implies that $\text{sc}(L\varphi^{-1}) \leq \text{sc}(L)$. This fact will be frequently useful.

Proposition 2.4.22. *Let L be a regular language with DFA $\mathcal{A} = (Q, \Sigma, T, 1, F)$. Fix a homomorphism $\varphi\colon \Gamma^* \to \Sigma^*$ and let $\mathcal{A}\varphi^{-1}$ be the inverse homomorphism DFA of $\mathcal{A}$. We have $L(\mathcal{A}\varphi^{-1}) = L\varphi^{-1}$.*

Proof. We have $w \in L(\mathcal{A}\varphi^{-1}) \iff 1w\varphi \in F \iff w\varphi \in L \iff w \in L\varphi^{-1}$. □

2.4.5 State Complexity of Basic Operations

We have defined the notion of state complexity of operations in Section 2.4.3, and demonstrated how to construct automata for these operations in Section 2.4.4. Now, we put all these concepts together and compute the state complexities of the following operations: reversal, concatenation, star, and binary boolean operations.

The n-state DFSA defined in Example 2.4.3, which has T_n as its transition monoid, will play an important role in the following proofs. Figure 2.4.25 shows an automaton derived from this semiautomaton, where state 1 has been made initial and state n has been made final. Brzozowski was the first to demonstrate the power of this automaton in a state complexity context; he used it to prove all the results we will prove in this section [6]. The results were originally proved by Maslov [62] and independently by Yu, Zhuang and Salomaa [79] using an assortment of different automata.

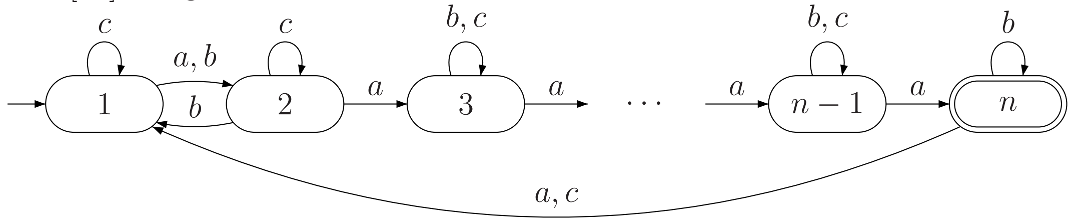


Figure 2.4.25: Deterministic automaton $\mathcal{A}_n$ with transition monoid T_n.

This DFA has letter actions $a = (1, 2, \dots, n)$, $b = (1, 2)$, and $c = (n \to 1)$, and every transformation of $\{1, \dots, n\}$ can be represented as the action of a word over $\{a, b, c\}$. Throughout this section, we will call this automaton $\mathcal{A}_n = (Q, \Sigma, T, 1, F)$. We write L_n for the language of $\mathcal{A}_n$.

We also introduce an automaton $\mathcal{A}'_n$, which is the same as $\mathcal{A}_n$ but with the states renamed to $\{1', 2', \dots, n'\}$ and the letter actions of a and b swapped: $a = (1', 2')$ and $b = (1', 2', \dots, n')$. This automaton is shown in Figure 2.4.26. We write L'_n for its language.

Reversal. Let $\mathcal{A}$ be an n-state automaton with state set Q. Then the state complexity of $L(\mathcal{A})^R$ is at most 2^n. To see this, recall that we can construct an automaton $\mathcal{R}$ for $L(\mathcal{A})^R$ with state set $\mathcal{P}(Q)$, and there are 2^n subsets of a set of size n. Thus $L(\mathcal{A})^R$ cannot have state complexity higher than 2^n, since there is a 2^n-state automaton that recognizes it.

Our goal is to show that this upper bound 2^n is tight for $n \geq 2$, i.e., for each n there is a DFA $\mathcal{A}$ with n states such that the state complexity of $L(\mathcal{A})^R$ is *exactly* 2^n. For $n = 1$, the only languages recognizable by one-state DFAs are Σ^* and $\emptyset$, which are both equal to

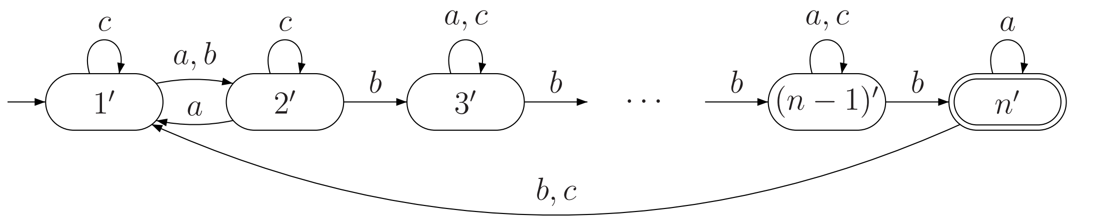


Figure 2.4.26: Deterministic automaton $\mathcal{A}'_n$ with transition monoid T_n; this automaton is essentially the same as $\mathcal{A}_n$ but the roles of a and b are swapped.

their own reverse. Thus for $n = 1$, the worst-case state complexity of reversal is 1 rather than $2^1 = 2$.

We use the DFA $\mathcal{A}_n = (Q, \Sigma, T, 1, F)$ with language L_n as our witness. We construct the reverse automaton, which we denote by $\mathcal{R}_n = (Q, \Sigma, T^{\mathcal{R}}, F, 1)$. Figure 2.4.27 shows the intermediate FA that is determinized to obtain the DFA $\mathcal{R}_n$.

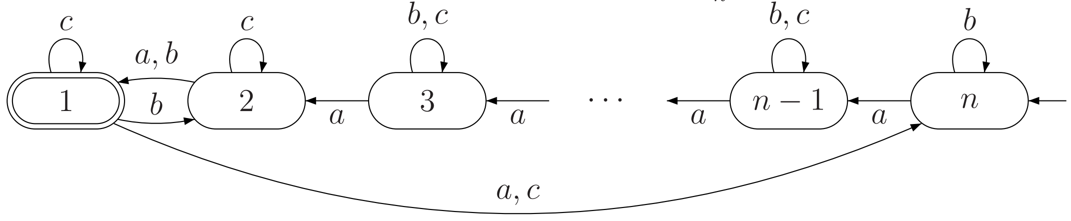


Figure 2.4.27: FA for the reverse of L_n.

Now we prove that the reverse of our witness L_n reaches the state complexity upper bound 2^n.

Theorem 2.4.23. *For $n \geq 2$, the DFA $\mathcal{R}_n$ recognizing L_n^R is minimal and has 2^n states.*

To prove that $\mathcal{R}_n$ is minimal, we must show that all states of $\mathcal{R}_n$ are reachable and all pairs of states are distinguishable. Before jumping into the proof, we establish some general facts about reverse automata that will be useful.

Recall from the proof of Proposition 2.4.15 that if $\mathcal{A}$ is an automaton with transition function T, and $\mathcal{R}$ is the reverse of $\mathcal{A}$ with transition function $T^{\mathcal{R}}$, then we have $T^{\mathcal{R}}_w = T^{-1}_{w^R}$. Thus each reachable state of $\mathcal{R}$ has the form Fw^{-1} for some word action w of $\mathcal{A}$. This means that determining which states are reachable in the reverse automaton $\mathcal{R}$ is equivalent to computing the preimages of the set F under the word actions of $\mathcal{A}$. Furthermore, if $\mathcal{R}$ is constructed from a DFA with all states reachable, the following proposition shows that it is easy to determine which states of the reverse automaton are distinguishable:

Lemma 2.4.24. *Let $\mathcal{A} = (Q, \Sigma, T, 1, F)$ be a DFA in which all states are reachable. Let $\mathcal{R}$ be the reverse of $\mathcal{A}$. Let $X, Y \subseteq Q$ be distinct reachable states of $\mathcal{R}$. Then X and Y are distinguishable.*

Proof. Choose a state q that is in X but not in Y. (We can do this without loss of generality; if X is a proper subset of Y then just swap the names of X and Y.) Since q is reachable in $\mathcal{A}$, there exists w such that $1w = q$. Then $1 \in qw^{-1}$. Furthermore, for all $p \neq q$, we have $1 \notin pw^{-1}$; otherwise we would have $p \in 1w$, but $\mathcal{A}$ is deterministic. Hence we have $1 \in XT_w^{\mathcal{R}} = XT_w^{-1}$ and $1 \notin YT_w^{\mathcal{R}} = YT_w^{-1}$, so $XT_w^{\mathcal{R}}$ is final and $YT_w^{\mathcal{R}}$ is non-final. That is, X and Y are distinguishable. $\square$

These facts make it easy to prove Theorem 2.4.23.

Proof of Theorem 2.4.23. By Lemma 2.4.24, all reachable states of $\mathcal{R}_n$ are distinguishable. So it suffices to show that $\mathcal{R}_n$ has 2^n reachable states. This means we need to show that each set $S \subseteq Q$ is reachable. As we remarked, the reachable states of $\mathcal{R}_n$ are those of the form Fw^{-1}, where w is a word action of $\mathcal{A}_n$. Since $F = \{n\}$, we must show that for every set $S \subseteq Q$, there exists $w \in \Sigma^*$ such that $nw^{-1} = S$.

This turns out to be fairly simple, since we have proved that $\mathcal{A}_n$ has transition monoid T_n. Thus for every transformation of $Q = \{1, \ldots, n\}$, there is a corresponding word action. Fix S and let w be a transformation such that $Sw = n$ and $(Q \setminus S)w = 1$. Since $n \geq 2$, we have $nw^{-1} = S$ and we are done. $\square$

Notice that constructing the reverse automaton $\mathcal{R}_n$ was not really necessary for this proof, since we reduced the problem to just looking at preimages of $\mathcal{A}_n$'s final state set under $\mathcal{A}_n$'s word actions.

This proof illustrates the importance of the transition monoid is in state complexity. There is a direct relationship between the transition monoid of an automaton $\mathcal{A}$ and the number of reachable states in its reverse $\mathcal{R}$. By choosing a witness automaton with a maximal transition monoid, we were able to show very easily that the maximal number of states is reachable. No matter what the operation is, increasing the number of transformations in the transition monoids of the operands will potentially allow new states to be reached or to be distinguished in the DFA resulting from the operation. Thus, a general rule of thumb when looking for state complexity witnesses is to choose automata with maximal transition monoids relative to the constraints of the problem. This is not a foolproof strategy, but it is a good heuristic.

Concatenation. Because concatenation is a binary operation, its state complexity is affected by our choice between the restricted and unrestricted viewpoints. In this section we will just compute the restricted state complexity of concatenation, since the process is simpler.

Let $\mathcal{A}' = (Q', \Sigma', T', 1', F')$ and $\mathcal{A} = (Q, \Sigma, T, 1, F)$ be DFAs with m states and n states, respectively, and a common alphabet $\Sigma' = \Sigma$. We claim that their concatenation automaton $\mathcal{C}$ has at most $(m-1)2^n + 2^{n-1}$ reachable states. By our convention of writing the states of $\mathcal{C}$ as ordered pairs, the initial state is either $(1', \emptyset)$ (if $1' \notin F'$) or $(1', 1)$ (if $1' \in F'$). The transition from a state (q', S) on a letter a can be described as follows:

$$(q', S) \xrightarrow{a} \begin{cases} (q'a, Sa \cup 1), & \text{if } q'a \in F; \\ (q'a, Sa), & \text{if } q'a \notin F. \end{cases}.$$

Why is this? Recall that the transition set of the FA used to constructed $\mathcal{C}$ is just $T' \cup T \cup \{(q', a, 1) : q' \in Q', a \in \Sigma, q'a \in F'\}$. Thus if $q'a$ is non-final, we just follow the transition $q' \xrightarrow{a} q'a$ from T' and the transitions $s \xrightarrow{a} sa$ from T for each $s \in S$, ending up at $(q'a, Sa)$. But if $q'a$ is final, we have the additional transition $q' \xrightarrow{a} 1$ to follow, so the state 1 gets added to the second component.

Because of this transition structure, if a state of the form (f', S) with $f' \in F$ is reachable, then S *must* contain 1. Therefore the reachable states fall into one of two categories:

1. States of the form (q', S) with $q' \notin F'$ and $S \subseteq Q$.
2. States of the form (f', S) with $f' \in F'$, $S \subseteq Q$, and $1 \in S$.

Suppose $|F'| = k$. Then there are $(m-k)2^n$ states in the first category and $k2^{n-1}$ states in the second category. Thus $(m-k)2^n + k2^{n-1}$ is an upper bound on the number of reachable states of $\mathcal{C}$. This number is maximized when $k = 1$, giving the upper bound $(m-1)2^n + 2^{n-1}$. This upper bound on the number of reachable states of $\mathcal{C}$ is also an upper bound on the state complexity of the concatenation of two languages, where the first is recognized by an m-state DFA and the second by an n-state DFA.

Now, we prove this upper bound is tight, using $\mathcal{A}'_m$ and $\mathcal{A}_n$ as our witness languages. Let $\mathcal{C}_{m,n}$ be the concatenation automaton constructed from $\mathcal{A}'_m$ and $\mathcal{A}_n$. The intermediate FA used to construct $\mathcal{C}_{m,n}$ is shown in Figure 2.4.28.

The following table shows the letter actions of $\mathcal{A}'_m$ and $\mathcal{A}_n$:

	a	b	c
$\mathcal{A}'_m$	$(1', 2')$	$(1', 2', \dots, m')$	$(m' \to 1')$
$\mathcal{A}_n$	$(1, 2, \dots, n)$	$(1, 2)$	$(n \to 1)$

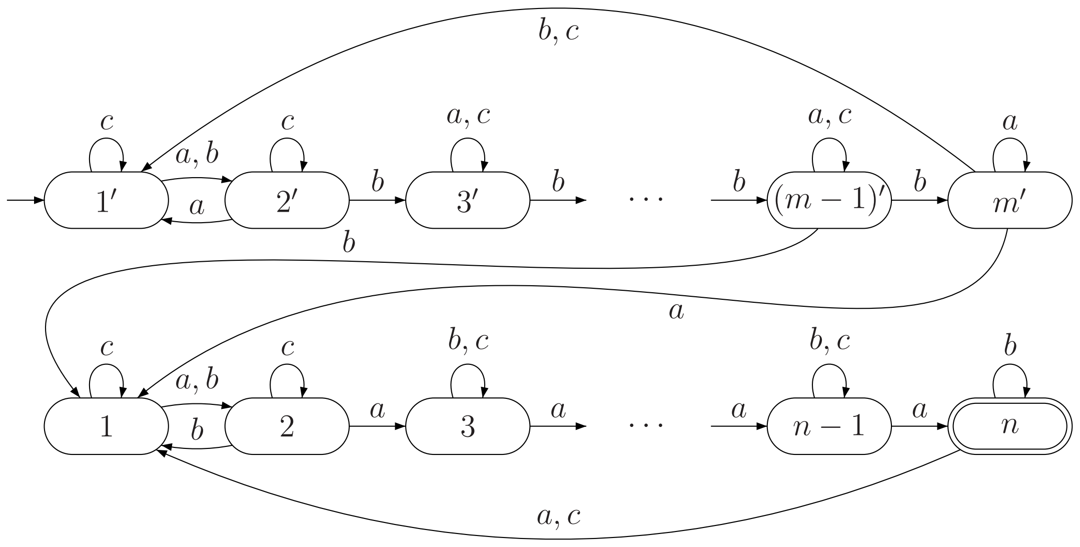


Figure 2.4.28: FA for the concatenation $L'_m L_n$.

Theorem 2.4.25. *If $m, n \geq 3$, the minimization of $\mathcal{C}_{m,n}$ has exactly $(m-1)2^n + 2^{n-1}$ states.*

Proof. First we show that $(m-1)2^n + 2^{n-1}$ states are reachable. The initial state is $(1', \emptyset)$. We prove by induction on $|S|$ that all states of the form (q', S) with $q' \neq m'$ and $(m', S \cup 1)$ are reachable.

The base case is $|S| = 0$. From $(1', \emptyset)$, we reach $(q', \emptyset)$ for $q' \neq m'$ via b^{q-1}. We reach $(m', 1)$ via b^{m-1}.

To show (q', S) for $q' \neq m'$ and $(m', S \cup 1)$ are reachable for $|S| > 0$, first consider the case $q' = 1'$. Let q be the minimal element of S.

If q is even, set $i = q-2$ and set $T = (S \setminus q)a^{-i}b^{-1}$. Since a and b are both permutations in $\mathcal{A}_n$, the set T has size $|S|-1$. Assume by induction that we can reach $(m', T \cup 1)$. Then $(m', T \cup 1)b = (1', (S \setminus q)a^{-i} \cup 2)$. Since i is even and $a = (1', 2')$ in $\mathcal{A}'_m$, we have $1'a^i = 1'$, and the path from $1'$ labelled with a^i just cycles between $1'$ and $2'$ and never passes through a final state. Thus $(1', (S \setminus q)a^{-i} \cup 2) = (1', (S \setminus q) \cup 2a^i)$. But $2a^i = 2 + q - 2 = q$, so we reach $(1', S)$.

If q is odd and n is odd, set $i = 2n + (n+1)(q-2)$ (note that this is even and is a positive integer, since $q \geq 1$ and thus $q - 2 \geq -1$). By the same argument as above we can reach $(1', (S \setminus q) \cup 2a^i)$. Modulo n, we have $2a^i \equiv 2 + 2n + (n+1)(q-2) \equiv 2 + q - 2 \equiv q$. Hence we reach $(1', S)$.

If q is odd and n is even, set $i = n-2$ and consider the set $T = (S \setminus q)(ba^iba^q)^{-1}$. Note that i is even and q is odd. Thus $1'a^i = 1'$, but $1'a^q = 2'$ and $2'a^q = 1'$. Assume as before that we can reach $(m', T \cup 1)$. We have

$$(m', T \cup 1) \xrightarrow{b} (1', Tb \cup 2) \xrightarrow{a^i} (1', Tba^i \cup n) \xrightarrow{b} (2', Tba^ib \cup n) \xrightarrow{a^q} (1', Tba^iba^a \cup q).$$

Since $Tba^iba^q = S \setminus q$, we reach $(1', S)$.

To reach (q', S) for $q' \neq m'$ and $q' > 1'$, first reach $(1', Sb^{-(q'-1)})$ and then apply $b^{q'-1}$. To reach $(m', S \cup 1)$, first reach $((m-1)', Sb^{-1})$ and then apply b.

This completes the inductive proof; we have shown that all states of the form (q', S) with $q' \neq m$ and $(m', S \cup 1)$ are reachable. We counted earlier that there are $(m-1)2^n + 2^{n-1}$ such states.

To finish the proof, we show that all of these reachable states are pairwise distinguishable. Let (p', S) and (q', T) be two distinct reachable states. If $S \neq T$, without loss of generality we can find $q \in S$ such that $q \notin T$. Applying a^{n-q} to both states sends S to a set containing n, and T to a set that does not contain n. Thus $(p', S)a^{n-q}$ is final, but $(q', T)a^{n-q}$ is not final; we have distinguished the states.

If $S = T$, then since the states are distinct we must have $p' \neq q'$. Assume without loss of generality that $p' < q'$. Apply $b^{m'-q'}$ to both states to reach $((m+p-q)', Sb^{m'-q'})$ and $(m', Sb^{m'-q'} \cup 1)$. If $Sb^{m'-q'} \neq Sb^{m'-q'} \cup 1$, we can distinguish these latter states by the same argument from earlier. We cannot apply this argument in the case where $Sb^{m'-q'}$ contains 1. Since $b = (1, 2)$ in $\mathcal{A}_n$, this can only happen if S contains 1 or 2.

Suppose S contains 1 or 2. Assume without loss of generality that $p' < q' < m'$; otherwise we can repeatedly apply b until neither p' nor q' is equal to m', and then rename the variables if necessary so that they are ordered as desired. Now, let q be the minimal element of S.

- By applying a^{n-q} we reach a set containing n.
- Then via c we reach a set containing 1 but not n.
- Via a we reach a set containing 2 but not 1.
- Via a^{n-2} we reach a set containing n but not $n-1$.

- Via c we reach a set containing 1, but neither n nor $n-1$.
- Via a^2 we reach a set containing 3, but neither 1 nor 2.

So if S contains 1 or 2, then $w = a^{n-q}ca^{n-1}ca^2$ sends S to a set that does not contain 1 or 2. Furthermore, since $a = (1', 2')$ in $\mathcal{A}'_m$ and $m \geq 3$, applying a will not send p' or q' to m'. So applying c will leave p' and q' untouched. Thus $p'w$ and $q'w$ are distinct states. It follows that by starting in (p', S) and (q', S) and applying w, we reach states $(p'w, Sw)$ and $(q'w, Sw)$ with $p'w \neq s'w$ and $1, 2 \notin Sw$. We proved in a previous case that states of this form are distinguishable; it follows that (p', S) and (q', S) are distinguishable.

We have shown that all reachable states are distinguishable, completing the proof. □

This argument was rather long and technical compared to the one for reversal. The relationship between the word actions of the concatenation automaton and the word actions in the operand automata is not as simple and direct as for reversal, so we needed to make careful arguments by building up words from individual letter actions. Despite having a maximal transition monoid in both of the operand automata, we could not really take advantage of these transition monoids in the same way as for reversal. Later in the thesis, we will learn some techniques that make concatenation state complexity proofs simpler.

Star. As before, we first derive an upper bound on the state complexity of star. Let $\mathcal{A} = (Q, \Sigma, T, 1, F)$ be an n-state DFA and construct its star automaton $\mathcal{S}$. We claim that $\mathcal{S}$ has at most $2^{n-1} + 2^{n-2}$ reachable states. The states of $\mathcal{S}$ (aside from the initial state $\{s\}$) are subsets of Q, and the transitions between subsets can be described as follows:

$$S \xrightarrow{a} \begin{cases} Sa \cup 1, & \text{if } Sa \cap F \neq \emptyset; \\ Sa, & \text{if } Sa \cap F = \emptyset. \end{cases}$$

Thus if a reachable state S contains a final state, it must also contain the initial state 1. We can divide the potentially reachable states into three groups:

- The initial state $\{s\}$.
- Non-empty subsets of Q which do not contain a final state.
- Subsets of Q which contain a final state and also 1.

Let us count these states. Suppose there are k final states. Then there are 2^{n-k} subsets of Q that do not contain a final state. The empty subset is not reachable, so we subtract 1 to get a count of $2^{n-k} - 1$.

Now consider subsets S of Q which contain a final state and also contain 1. We can write S as a disjoint union $X \cup Y$, where $X \subseteq Q \setminus F$ and $Y \subseteq F$. There are two cases, depending on whether $1 \in F$.

If $1 \in F$, then X is potentially empty, so there are 2^{n-k} choices for X. Since Y must contain 1, there are 2^{k-1} choices for Y. In total there are $2^{n-k}(2^{k-1}) = 2^{n-1}$ choices in this case.

If $1 \notin F$, then there are 2^{n-k-1} choices for X since it must contain 1. Since Y must contain a final state, it cannot be empty, so there are $2^k - 1$ choices for Y. In total there are $2^{n-k-1}(2^k - 1) = 2^{n-1} - 2^{n-k-1}$ choices in this case.

That deals with subsets of Q; we also need to add 1 for the initial state $\{s\}$. In summary:

- If $1 \in F$, there are at most $(2^{n-k} - 1) + 2^{n-1} + 1 = 2^{n-1} + 2^{n-k}$ reachable states.
- If $1 \notin F$, there are at most $(2^{n-k} - 1) + (2^{n-1} - 2^{n-k-1}) + 1 = 2^{n-1} + 2^{n-k-1}$ reachable states.

The maximum value of the bound in both cases is given by taking $k = 1$. However, in the case where $1 \in F$ and $k = 1$ (that is, $\mathcal{A}$ has a unique final state which is also initial) we claim that only $\{s\}$ and the n singleton subsets $\{q\}$, $q \in Q$ are reachable. Indeed, if qa is non-final then we just have $\{q\}a = \{qa\}$. If qa is final, then $qa = 1$, and thus $\{q\}a = \{1\} \cup \{1\} = \{1\}$.

Additionally, if $1 \in F$ then the states $\{s\}$ and $\{1\}$ are both final in $\mathcal{S}$, and they both accept the same language, so they are indistinguishable, which reduces the upper bound on state complexity by 1. Thus we have the following more precise upper bounds:

- If $F = \{1\}$, there are at most n indistinguishability classes of reachable states.
- If $F \neq \{1\}$ and $1 \in F$, there are at most $2^{n-1} + 2^{n-k} - 1$ indistinguishability classes of reachable states.
- If $1 \notin F$, there are at most $2^{n-1} + 2^{n-k-1}$ reachable states.

The maximal upper bound is $2^{n-1} + 2^{n-2}$, given by taking $1 \notin F$ and $k = 1$.

Later on we will prove that the upper bounds above are tight for all configurations of initial and final states, but for now we will just prove that the absolute upper bound $2^{n-1} + 2^{n-2}$ is tight. We use $\mathcal{A}_n$ as our witness language; let $\mathcal{S}_n$ be the star automaton constructed from $\mathcal{A}_n$. The intermediate FA used to construct $\mathcal{S}_n$ is shown in Figure 2.4.29.

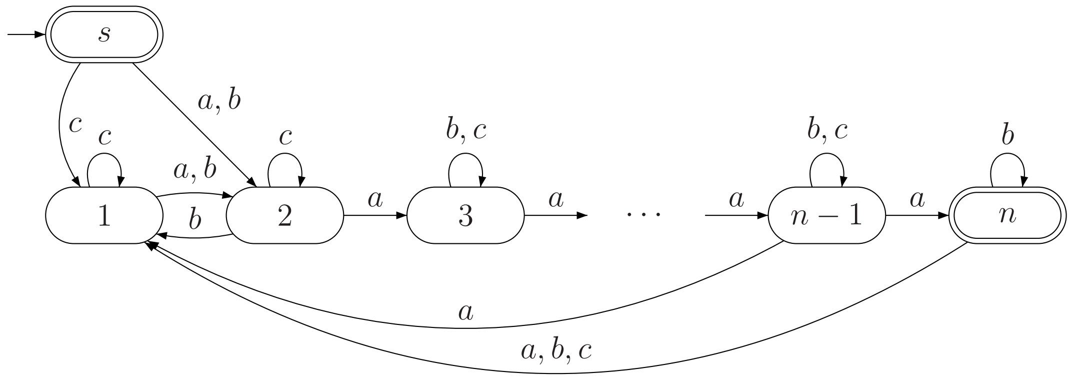


Figure 2.4.29: FA for the star of L_n.

Theorem 2.4.26. *If $n \geq 3$, the minimization of $\mathcal{S}_n$ has exactly $2^{n-1} + 2^{n-2}$ states.*

Proof. First we show that $2^{n-1} + 2^{n-2}$ states are reachable in $\mathcal{S}_n$. The initial state $\{s\}$ is trivially reachable. We prove by induction on $|S|$ that all non-empty sets $S \subset Q \setminus F$, and all sets $1 \cup S$ with $1 \notin S$ and $S \cap F \neq \emptyset$, are reachable. Note that $S \cap F \neq \emptyset$ is equivalent to $n \in S$, since $F = \{n\}$.

For the base case $|S| = 1$, we have

$$\{s\} \xrightarrow{c} \{1\} \xrightarrow{a} \{2\} \xrightarrow{a} \{3\} \cdots \xrightarrow{a} \{n-1\} \xrightarrow{a} \{1, n\}.$$

Now suppose $|S| = k$. Write $S = \{q_1, \dots, q_k\}$ with $q_1 < \cdots < q_k$.

Suppose $S \subseteq Q \setminus F$, and $q_1 = 1$. Let $T = \{q_3 - q_2 + 1, \dots, q_k - q_2 + 1, n\}$. This set has size $|S| - 1$, so we may assume that $1 \cup T$ is reachable by induction. Let $w = a(ab)^{q_2 - 2}$. We claim that $(1 \cup T)w = S$. Indeed, observe that

$$\{1, n\} \xrightarrow{a} \{1, 2\} \xrightarrow{ab} \{1, 3\} \xrightarrow{ab} \cdots \xrightarrow{ab} \{1, n-1\}.$$

Thus $\{1, n\}a(ab)^k = \{1, 2 + k\}$ for $k \leq n - 3$. In particular, $\{1, n\}a(ab)^{q_2 - 2} = \{1, q_2\}$. Now consider $(T \setminus n)w = \{q_3 - q_2 + 1, \dots, q_k - q_2 + 1\}w$; we have $q_i - q_2 + 1 \xrightarrow{a} q_i - q_2 + 2 \xrightarrow{(ab)^{q_2 - 2}} = q_i$. Thus $(1 \cup T)w = \{1, q_2, q_3, \dots, q_k\} = S$.

Now suppose $S \subseteq Q \setminus F$ and $q_1 > 1$. Reach $\{1, q_2 - q_1 + 1, \dots, q_k - q_1 + 1\}$ by the above argument, and apply $a^{q_1 - 1}$ to reach S. Finally, consider $1 \cup S$ with $1 \notin S$ and $n \in S$. Reach $\{q_1 - 1, \dots, q_k - 1\}$ by previous arguments, then apply a to reach $1 \cup S$.

This shows that $2^{n-1} + 2^{n-2}$ states are reachable. Now we show all of these states are pairwise distinguishable. If S and T are distinct states, without loss of generality we can find $q \in S$ such that $q \notin T$. Applying a^{n-q} sends S to a final state (containing n), and T to a non-final state (not containing n). Hence we have distinguished the states. $\square$

Boolean operations. We consider only binary boolean operations, and as with concatenation, we consider only restricted state complexity. Recall that binary boolean operations such as union and intersection are recognized by a direct product of two DFAs. If the two DFAs have m and n states respectively, the direct product has $(m+1)(n+1)$ states in the unrestricted case, but in the restricted case mn states suffice. Thus we immediately get an upper bound of mn on the state complexity of all binary boolean operations.

For the lower bound, we once again use $\mathcal{A}'_m$ and $\mathcal{A}_n$ as our witnesses. Figure 2.4.30 shows an example direct product for $m = 3$, $n = 4$.

There are 16 binary boolean operations; six of these are *improper* operations which are either constant or depend on only one argument. The other ten are *proper* operations which depend on both arguments: union $L' \cup L$, intersection $L' \cap L$, two difference operations $L' \setminus L$ and $L \setminus L'$, symmetric difference $L' \oplus L = (L' \setminus L) \cup (L \setminus L')$, and the complements of these operations: $(\Sigma' \cup \Sigma)^* \setminus (L' \circ L)$, where $\circ$ is one of the previous five operations and Σ' (respectively Σ) is the alphabet of L' (respectively L).

To prove a particular boolean operation has state complexity mn, the idea is to first show that all mn states of the direct product $\mathcal{A}'_m \times \mathcal{A}_n$ are reachable. Then we look at the final state set corresponding to the boolean operation of interest, and show that all mn states are distinguishable under this final state set. This method requires a separate distinguishability argument for each boolean operation we consider, and these arguments are often very similar. To avoid too much repetition, we will just look at union and intersection, which should be enough to get a rough idea of how these proofs work. Later we will learn some general techniques that allow us to deal with many operations at once and avoid splitting the proof into individual cases for each operation.

For convenience, we reproduce the table of letter actions for $\mathcal{A}'_m$ and $\mathcal{A}_n$:

	a	b	c
$\mathcal{A}'_m$	$(1', 2')$	$(1', 2', \dots, m')$	$(m' \to 1')$
$\mathcal{A}_n$	$(1, 2, \dots, n)$	$(1, 2)$	$(n \to 1)$

Lemma 2.4.27. *In the direct product $\mathcal{A}'_m \times \mathcal{A}_n$, all mn states are reachable using words over $\{a, b\}$.*

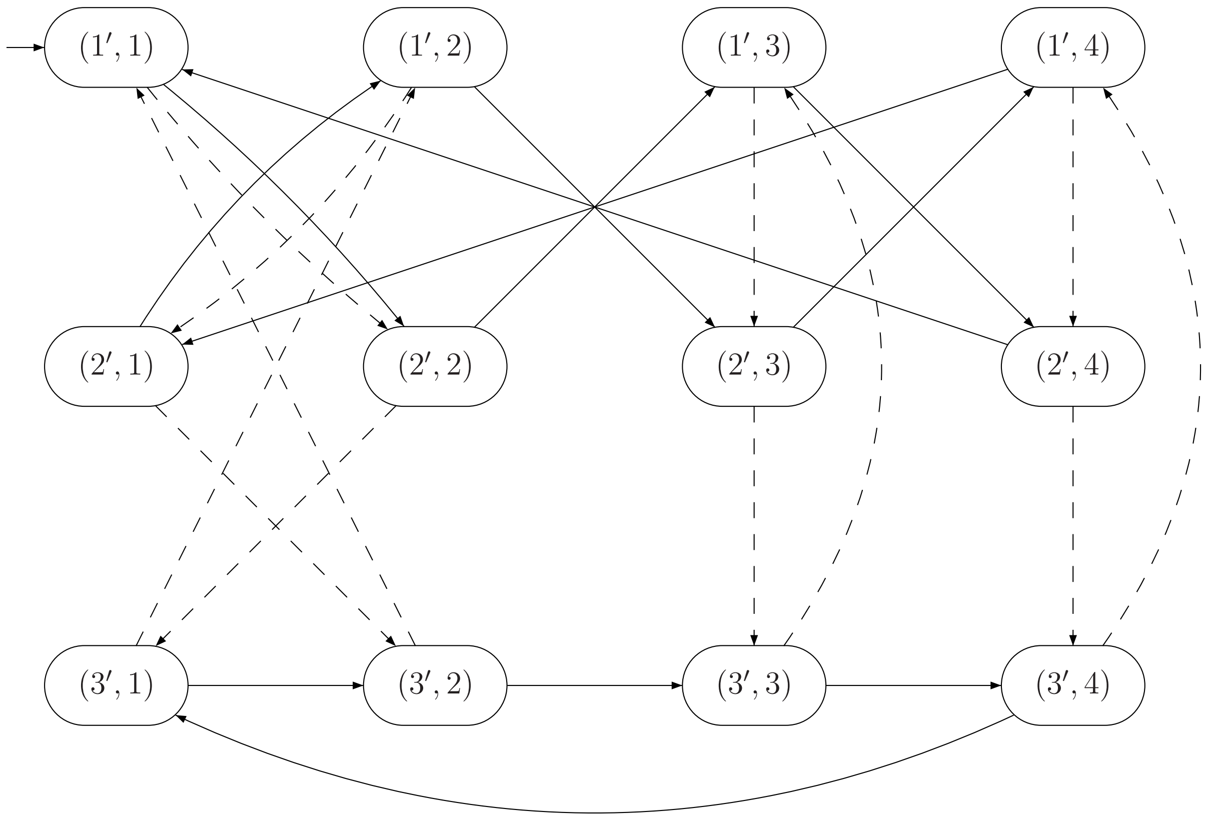


Figure 2.4.30: Direct product $\mathcal{A}'_3 \times \mathcal{A}_4$, with no final states assigned. Solid lines denote transitions on a and dashed lines denote transitions on b. Transitions on c are omitted.

Proof. First we show that all states of the form $(3', q)$ are reachable. From the initial state $(1', 1)$, via ab we reach $(3', 1)$. Then via b^{q-1} we reach $(3', q)$. Now to reach (p', q), first reach $(3', qa^{-(m+p-3)})$ and then apply a^{m+p-3} to reach $(m' + p', q) = (p', q)$. ☐

Theorem 2.4.28. *Let $\mathcal{B}_{m,n}$ be the direct product $\mathcal{A}'_m \times \mathcal{A}_n$ with final state set $(Q' \times F) \cup (F' \times Q)$; this DFA recognizes the union $L'_m \cup L_n$. If $m, n \geq 3$, then $\mathcal{B}_{m,n}$ is minimal with mn states.*

Proof. By Lemma 2.4.27 we know all mn states are reachable. For distinguishability, let (p', q) and (r', s) be distinct states. Since they are distinct, we have $p' \neq r'$ or $q' \neq s'$. Call the first component of a state the “row” and the second component the “column”. Since

$F = \{n\}$ and $F' = \{m'\}$, a state in the direct product is final if the row is m' or the column is n'.

Suppose $q \neq s$; without loss of generality assume $q > s$. Apply a^{n-q} to send (p', q) to a final state in column n, and (r', s) to a state outside of column n. If $(r', s)a^{n-q}$ is non-final, we have distinguished the states; otherwise $(r', s)a^{n-q}$ is final and so must lie in row m'. So our new states are $(p'a^{n-q}, n)$ and (m', sa^{n-q}). Applying b distinguishes these states, since $n \geq 3$ implies the first state will stay in column n and stay final, while the second state will move from row m' to row $1'$ and stay non-final. Now suppose $p' \neq r'$; without loss of generality assume $p' > r'$. Then a symmetric argument shows that since $m \geq 3$, either b^{m-p} or $b^{m-p}a$ distinguishes the states. Thus all mn states are distinguishable. □

Theorem 2.4.29. *Let $\mathcal{B}_{m,n}$ be the direct product $\mathcal{A}'_m \times \mathcal{A}_n$ with final state set $F' \times F$; this DFA recognizes the intersection $L'_m \cap L_n$. If $m, n \geq 3$, then $\mathcal{B}_{m,n}$ is minimal with mn states.*

Proof. Again it suffices to show all states are pairwise distinguishable. Let (p', q) and (r', s) be distinct states. This time there is a unique final state, which is (m', n).

If $q \neq s$, assume without loss of generality that $q > s$. Then a^{n-q} sends (p', q) to the state $(p'a^{n-q}, n)$ in column n and (r', s) to a state outside of column n. Now let $i' = p'a^{n-q}$ and apply b^{m-i}; the first state is sent to the final state (m', n) and the second is sent to some non-final state, thus distinguishing the two states. □

Conclusions. We have now established the state complexity of what are commonly called the "basic operations" on regular languages: reversal, concatenation, star and boolean operations. As discussed in the introduction, we can look at many variations of the problems we solved in this section:

- We can try to reduce the number of letters used in our witnesses. In particular, for reversal, concatenation and star it is possible to only use two letters instead of three (for boolean operations we already used just a and b).
- We can try to prove more general tight upper bounds. For example, we saw in the case of star that the upper bound $2^{n-1} + 2^{n-2}$ is only tight in the case where $1 \notin F$ and $|F| = 1$; we can try to establish tight bounds for other state configurations.
- For concatenation and boolean operations, we can look at the unrestricted state complexity – state complexity with respect to the unrestricted viewpoint, where the operands are allowed to have different alphabets.

- We can consider basic operations on *subclasses* of the regular languages.

It is feasible to attack all of these problems using ad-hoc combinatorial arguments, like we did for concatenation, star and boolean operations. But after writing many proofs this way, it starts to become tedious. These proofs are similar in structure, but not quite similar enough that it is easy to reuse arguments. These proofs are elementary and feel routine, but can still require significant amounts of time and thought to produce.

The proof for reversal is more like what we want – it is made up of straightforward, reusable arguments. While it took some effort to establish that the transition monoid of our witness is the full transformation monoid T_n, afterwards the reachability argument was extremely easy. Furthermore, this reachability argument applies to *all* DFAs whose transition monoid is T_n. The distinguishability argument is also very general and can be reused in other reversal state complexity proofs. If we had results like this for other operations, the corresponding state complexity proofs would be much simpler.

The goal of this thesis is to make state complexity proofs easier for researchers to produce. In the following chapters, we describe general, reusable techniques to shorten and simplify state complexity proofs. In Chapter 7, we will revisit the state complexity problems discussed in this section and see how our new techniques help with them.

Chapter 3

The One-Letter-Per-Action Approach

3.1 Witnesses for Worst-Case State Complexity

As we saw in Section 2.4.5, to establish the state complexity of a regular operation, we must first derive an upper bound on the state complexity and then find *witness languages* that attain this upper bound. State complexity papers rarely elaborate on how witnesses are discovered. The witnesses $\mathcal{A}'_m$ and $\mathcal{A}_n$ we used in Section 2.4.5 are taken from a 2013 paper of Brzozowski [6]; according to a private communication, he discovered them by hand, playing around with 3-state automata until he was able to make a general conjecture. (Brzozowski notes that similar automata have appeared before; in 1963, Lupanov [61] used the *reverse* of an automaton very much like $\mathcal{A}_n$, but with letter action $(2 \to 1)$ replacing $(n \to 1)$, to establish the state complexity of FA to DFA conversion. Mirkin [64] observed in 1966 that this witness can be used to establish a lower bound on the state complexity of reversal.) Witnesses are often found by computer search rather than by hand; in one paper, Domaratzki and Okhotin describe checking all minimal 4-state DFAs over a 4-letter alphabet (364,644,290 DFAs in total) to find examples that attain the maximal state complexity for the "cube" operation [31].

In the early papers on state complexity of basic operations by Maslov [62] and Yu, Zhuang, and Salomaa [79], the witnesses $\mathcal{A}'_m$ and $\mathcal{A}_n$ were not used. In fact, totally different witnesses were generally presented for each operation studied (though Yu, Zhuang and Salomaa did use essentially the same witnesses for both union and intersection). It was quite a breakthrough when Brzozowski showed that just one "universal" witness and a slight modification of that witness are sufficient for the basic operations.

Part of the reason the witnesses $\mathcal{A}'_m$ and $\mathcal{A}_n$ are so effective is that they have maximal transition monoids. A witness with a maximal transition monoid is guaranteed to maximize the state complexity of reversal. However, it is not the case that a witness with maximal transition monoid must maximize the state complexity of star, or that a pair of witnesses with maximal transition monoids must maximize the state complexities of concatenation or binary boolean operations. Nonetheless, having a maximal transition monoid is *helpful* because the number of reachable and distinguishable states in an automaton depends directly on the transformations in its transition monoid. If we take automata with maximal transition monoids and apply an operation to them, the resulting automaton will not necessarily have a maximal transition monoid, but will usually at least have a *large* transition monoid that hopefully has useful transformations in it.

Usefulness of the transformations in the transition monoid is the other part of the equation when it comes to maximizing state complexity. For example, a constant transformation can be used to reach one particular state, but no others, and it cannot distinguish pairs of states. By contrast, a cyclic permutation of the entire state set is much more useful since it can be used to reach every state, and may also be able to distinguish many states depending on what the final state set is. We do not just want a maximal transformation monoid, but we want the letter actions that generate it to be as useful as possible. Unfortunately, this notion of "usefulness" is rather vague and hard to quantify, compared to "maximality" which has a mathematical definition. It is not clear how to choose the most useful transformations for a witness – at best we can make educated guesses based on intuition and data (such as which transformations commonly appear in known witnesses).

This is the situation we are stuck with when it comes for searching for witnesses, *provided we want our witnesses to have small alphabets.* State complexity researchers often are concerned with this – partly because witnesses with small alphabets are easier to describe and more useful in a practical context, but also just because minimizing the alphabet size of witnesses is an interesting theoretical problem. But if we do not care about alphabet size, there is a simple solution to the "usefulness" problem that works in a large number of cases. Instead of trying to decide which transformations are the most useful to have as letter actions, *what if we just include them all?* That is, what if our witness simply includes *one letter for each possible action* on the state set?

We will henceforth refer to this idea as the *one-letter-per-action* (OLPA) approach. This will be our primary approach to finding witnesses in this thesis. The purpose of this section is to formalize this approach and establish its power and limitations – what kinds of regular operations do OLPA witnesses work for? We will initially focus on the simple special case of unary operations, and then generalize to operations of arbitrary arity.

The way we have presented this approach might make it seem almost obvious – having more letter actions generally helps with reachability and distinguishability, so of course including all possible letter actions will maximize state complexity in most cases. But this is only obvious if one has learned to think of automata in terms of letter actions! Automata are often presented to students as directed graphs, whose fundamental components are *states and transitions* rather than *states and actions*. From the viewpoint of graphs, "the function determined by the collection of transitions associated with a letter" does not seem like a natural concept to define. Compare the following two equivalent problems:

- You are given an automaton with states $\{1, 2, \ldots, n\}$, initial state 1, and unique final state n, but no transitions. Draw transitions between the states such that the automaton remains deterministic and such that when the automaton is reversed and determinized, it has 2^n reachable states.
- Construct a transformation monoid T on $\{1, 2, \ldots, n\}$ such that for each subset S of $\{1, 2, \ldots, n\}$, there is a transformation $t \in T$ with $nt^{-1} = S$.

The second problem has an immediately obvious solution: take T to be the full transformation monoid. Whereas the first seems to require much more thought – the natural approach is to try to figure out the correspondence between transitions in the original automaton and the reversed determinized version, which takes some effort to work out.

In summary, the reason the OLPA approach might seem obvious is that we have chosen to view automata primarily as monoid actions on finite sets – as *algebraic* objects. This algebraic viewpoint makes state complexity problems more approachable and their underlying principles more understandable compared to the common graph-theoretic viewpoint. The OLPA approach to finding witnesses arises naturally from this viewpoint; it is the cornerstone of this thesis and the centerpiece of our *algebraic approaches to state complexity of regular operations*.

Remark. We assume throughout this chapter that all regular operations considered are *alphabet-preserving*. For a unary operation, this means the alphabet of the output language must be the same as the alphabet of the input language. For operations of higher arity, this means all input languages must have the same alphabet, and the output language must also have this alphabet.

3.2 History of the OLPA Approach

In 2018, I posted a paper to the arXiv [26] which is, to my knowledge, the first comprehensive account of the OLPA approach in the specific context of deterministic state complexity of regular operations. Caron, Hamel-De le court, Luque and Patrou independently discovered the idea and posted a similar paper on the arXiv ten days later [24]. However, we were not the first to think of this idea, or even the first to apply it to state complexity problems. The OLPA approach has a rather long and interesting history.

The key insight dates back to a 1978 paper of Sakoda and Sipser [72]. They constructed languages wherein the alphabet letters were directed graphs representing behaviours of ordinary finite automata and a different model known as "two-way deterministic" finite automata, with one letter corresponding to each possible behaviour of these models. They used these languages to prove results on the state complexity of conversions between various automaton models.

Perhaps the closest ancestor of my work (and that of Caron et al.) is a 1990 paper of Ravikumar [69], who treated the "Sakoda-Sipser technique" as a "systematic method to prove lower bounds on the size complexity of finite automata", and applied it to five different problems, two of which were operational state complexity problems. This is the first work we are aware of to present the OLPA approach as a general problem-solving method. Unfortunately, the field of state complexity was not so well-developed at the time, and Ravikumar seemingly did not realize the full generality and applicability of the approach in operational state complexity. Furthermore, Ravikumar's use of the OLPA approach was less refined than the version we will present; OLPA witnesses for unary operations were applied to k-ary operations without a proper generalization.

In 1992, building on Ravikumar's work, Birget [3] used this "unrefined" version of the OLPA approach to prove lower bounds on the state complexity of k-ary intersection and union.

In 1994, Yu, Zhuang and Salomaa [79] published their seminal paper on the state complexities of basic operations. Notably, even though Yu, Zhuang and Salomaa cited Ravikumar's work, they did *not* use or mention the "Sakoda-Sipser technique" anywhere in their paper. The technique then seemingly faded into obscurity for a while. It reappeared as the "full automata technique" in a 2006 paper of Yan [77], who credited Sakoda and Sipser for the idea, and applied it to nondeterministic state complexity of finite automata, as well as to automata on infinite words (ω-automata). Yan's paper is frequently cited in the field of ω-automata, so the technique seems to have gained some currency there.

Despite not being used by Yu, Zhuang and Salomaa, the OLPA approach made occasional appearances in deterministic state complexity papers prior to 2018. Jirásková and Okhotin [55] and later Domaratzki and Okhotin [31] used the OLPA approach to compute the exact worst-case complexity of the "cyclic shift" and "power" operations for small values. Brzozowski, Jirásková, Liu, Rajasekaran, and Szykuła [13] used the OLPA approach to obtain reachability results for the state complexity of the "shuffle" operation. However, these authors did not present the approach as a general technique or demonstrate its wider applicability.

3.3 Unary Operations

In this section, we formalize the OLPA approach for unary operations, that is, operations which take a single regular language as input. There are three steps to the formalization:

1. Define the one-letter-per-action DFAs that will be used as witnesses.
2. Define a class of unary regular operations for which the OLPA approach works.
3. Prove that our witnesses maximize the state complexity of all these operations.

3.3.1 Full Transformation Languages

We now formally define the witness languages that are used in the OLPA approach.

Fix Q and let Σ be a set of transformations of Q. For $1 \in Q$ and $F \subseteq Q$, the *transformation language* $\Sigma(1, F)$ is the language of the DFA $(Q, \Sigma, T, 1, F)$, where $T = \{(q, t, qt) : q \in Q, t \in \Sigma\}$. This DFA is called the *standard DFA* for the transformation language. A word $w \in \Sigma^*$ lies in the transformation language $\Sigma(1, F)$ if and only if the corresponding transformation (given by composing the letters of w) maps 1 into F.

Definition 3.3.1. Recall that T_Q denotes the full transformation monoid on Q. The language $T_Q(1, F)$ is called the *full transformation language* with respect to the state configuration $(Q, 1, F)$.

Notice that the language $T_Q(1, F)$ has alphabet T_Q, and the standard DFA for $T_Q(1, F)$ has transitions $\{(q, t, qt) : q \in Q, t \in T_Q\}$. This DFA has *one letter per transformation* of the state set Q, that is, one letter per possible action on the DFA's states. Full transformation languages are the languages used as witnesses when applying the OLPA approach to unary operations.

Figure 3.3.1 shows the standard DFA for the language $T_{\{1,2\}}(1, \{2\})$. To illustrate the construction with a three-state DFA, we would need $3^3 = 27$ alphabet letters!

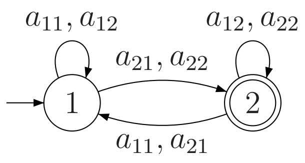


Figure 3.3.1: Standard DFA of the full transformation language $T_{\{1,2\}}(1, \{2\})$, where a_{ij} represents the transformation that sends 1 to i and 2 to j.

Definition 3.3.2. Let L be a regular language over Σ recognized by $\mathcal{D} = (Q, \Sigma, T, 1, F)$. The *standard transformation map* of L (with respect to $\mathcal{D}$), denoted by $\varphi_L \colon \Sigma^* \to T_Q^*$, is the monoid homomorphism defined by $w\varphi_L = T_w$. That is, each word over the alphabet of L is mapped to the transformation the word induces in $\mathcal{D}$.

Using the standard transformation map, we may establish the following identity, which shows that every regular language is the preimage of a full transformation language under a homomorphism of free monoids.

Proposition 3.3.3. $L = T_Q(1, F)\varphi_L^{-1}$.

Proof. We have $w \in L \iff 1w \in F \iff T_w \in T_Q(1, F) \iff w\varphi_L \in T_Q(1, F) \iff w \in T_Q(1, F)\varphi_L^{-1}$. □

3.3.2 Uniform Unary Operations

Our goal in this section is to define a large class of unary regular operations for which the OLPA approach "works", in the sense that full transformation languages can be used as witnesses for the maximal state complexity of all operations in the class. The approach certainly does not work for all regular operations. For example, consider an operation which sends all languages recognized by DFAs with one letter per action to the empty language, and acts as the identity on all other languages. Clearly our witnesses will not maximize the state complexity of this operation.

One problem with this operation is that its behaviour is not "uniform" across all languages; it detects particular languages and has special behaviour for them. Consider how this operation behaves on DFAs. Here are two ways we could implement this operation as a DFA-to-DFA mapping:

- If the input DFA has one letter per action, output a DFA with no final states. Otherwise, output the input DFA.
- If the input DFA has one letter per action, output a DFA in which the initial state is non-final and all the actions send the initial state to a sink state. Otherwise, output the input DFA.

In the first case, the operation does not behave uniformly on states: for most DFAs it preserves the final state set, but for DFAs with one letter per action it can change the final state set. In the second case, the operation is not uniform on states *or* on actions: for most DFAs it preserves the state configuration and actions, but for DFAs with one letter per action, it can change whether the initial state is final, and also replace the actions by completely different actions.

By contrast, consider a common operation like star. Recall that the star operation can be implemented by the following DFA construction, which takes an input DFA $(Q, \Sigma, T, 1, F)$:

1. Add a new state s, make it the only initial state, and also make it final.
2. Add transitions $(s, a, 1a)$ for each $a \in \Sigma$.
3. Add transitions $(f, a, 1a)$ for each $f \in F$.
4. Perform the subset construction to obtain a DFA.

The behaviour on states is "uniform": no matter what the input DFA looks like, we always add a single new state that is both initial and final, and perform the subset construction. The behaviour on actions is also "uniform": it depends on the choice of initial and final states, but this dependence is consistent across all DFAs, in the sense that if two DFAs have the same state configuration then the star operation will modify their actions in a consistent way. For each action in the input DFA, there is a corresponding action in the output DFA, and this corresponding action depends only on the input action and the state configuration.

If we have an operation which behaves uniformly like this, the OLPA approach should work. If we add a new action to the input DFA, then the output DFA will gain a corresponding new action, which will potentially mean new states are reachable or pairwise distinguishable in the output DFA; so adding new actions to the input DFA does not decrease the state complexity of the output DFA. Additionally, if we have two letters which perform the same action in the input DFA, those two letters will perform the same corresponding action in the output DFA. It follows that if the input DFA has every possible action, as the DFAs of full transformation languages do, then adding more letters will not affect the state complexity; the state complexity of the output DFA can only be potentially increased by changing the state configuration. But if the number of states is fixed, the number of possible state configurations is finite. Thus there exists a state configuration for which the corresponding full transformation language maximizes the state complexity of the operation!

We now attempt to formally define this idea of "uniformity" for unary operations. Let Φ be a unary regular operation. Let Ψ be a unary *DFA operation*, that is, a function which maps DFAs to DFAs. We say Φ is *equivalent* to Ψ if for all DFAs $\mathcal{D}$, we have $L(\mathcal{D})\Phi = L(\mathcal{D}\Psi)$; that is, if performing the language operation on the language of a DFA $\mathcal{D}$ is the same as performing the DFA operation on $\mathcal{D}$ and taking the language of the result.

Definition 3.3.4. A unary DFA operation Ψ is *uniform* if for every pair of DFAs $\mathcal{A} = (Q, \Sigma, T_{\mathcal{A}}, 1, F)$ and $\mathcal{B} = (Q, \Gamma, T_{\mathcal{B}}, 1, F)$ with the same state configuration, the image DFAs $\mathcal{A}\Psi = (Q'_{\mathcal{A}}, \Sigma, T'_{\mathcal{A}}, 1'_{\mathcal{A}}, F'_{\mathcal{A}})$ and $\mathcal{B}\Psi = (Q'_{\mathcal{B}}, \Gamma, T'_{\mathcal{B}}, 1'_{\mathcal{B}}, F'_{\mathcal{B}})$ satisfy the following conditions:

1. $(Q'_{\mathcal{A}}, 1'_{\mathcal{A}}, F'_{\mathcal{A}}) = (Q'_{\mathcal{B}}, 1'_{\mathcal{B}}, F'_{\mathcal{B}})$.
2. Whenever $(T_{\mathcal{A}})_a = (T_{\mathcal{B}})_b$ for $a \in \Sigma$ and $b \in \Gamma$, we have $(T'_{\mathcal{A}})_a = (T'_{\mathcal{B}})_b$.

A unary regular operation Φ is *uniform* if there exists a uniform unary DFA operation equivalent to Φ.

We can interpret this definition intuitively as follows. The first condition says that the operation is uniform with respect to state configurations: if the operation is given two input DFAs with the same state configuration, it will produce two output DFAs with the same state configuration. The second condition says that the operation is uniform with respect to actions: if the operation is given two input DFAs with the same state configuration and a common action, then it will produce two output DFAs with a common action, and furthermore the same letters which induce the common action in the input DFAs will induce the common action in the output DFAs.

The definition of uniformity is heavily dependent on DFAs. Thus, it may come as a surprise that there is a simple and purely language-theoretic characterization of uniformity, as follows. A monoid homomorphism $\varphi\colon \Sigma^* \to \Gamma^*$ is called *1-uniform* if it maps letters to letters.

Proposition 3.3.5. *The following are equivalent:*

1. *The regular operation Φ is uniform.*

2. *For all 1-uniform homomorphisms $\varphi\colon \Sigma^* \to \Gamma^*$, and all regular languages $L \subseteq \Sigma^*$ and $K \subseteq \Gamma^*$ such that $L = K\varphi^{-1}$, we have $L\Phi = K\Phi\varphi^{-1}$.*

Proof. (1) $\implies$ (2): Since Φ is uniform, there is a uniform DFA operation Ψ equivalent to Φ. Fix a 1-uniform homomorphism $\varphi\colon \Sigma^* \to \Gamma^*$ such that $L = K\varphi^{-1}$. Let $\mathcal{A} = (Q_\mathcal{A}, \Sigma, T_\mathcal{A}, 1_\mathcal{A}, F_\mathcal{A})$ be a DFA for L, and let $\mathcal{B} = (Q_\mathcal{B}, \Gamma, T_\mathcal{B}, 1_\mathcal{B}, F_\mathcal{B})$ be a DFA for K. We write $w_\mathcal{A}$ for $(T_\mathcal{A})_w$, and $w_\mathcal{B}$ for $(T_\mathcal{B})_w$.

Note that we can choose our DFAs so that they have the same state configuration. This follows from the fact that $L = K\varphi^{-1}$, and thus we can take $\mathcal{A} = \mathcal{B}\varphi^{-1}$ which has the same state configuration as $\mathcal{B}$. Henceforth write $Q_\mathcal{A} = Q_\mathcal{B} = Q$, $1_\mathcal{A} = 1_\mathcal{B} = 1$, and $F_\mathcal{A} = F_\mathcal{B} = F$.

Let $\mathcal{A}\Psi = \mathcal{A}' = (Q'_\mathcal{A}, \Sigma, T'_\mathcal{A}, 1'_\mathcal{A}, F'_\mathcal{A})$ and let $\mathcal{B}\Psi = \mathcal{B}' = (Q'_\mathcal{B}, \Gamma, T'_\mathcal{B}, 1'_\mathcal{B}, F'_\mathcal{B})$. Write $w'_\mathcal{A}$ for $(T'_\mathcal{A})_w$ and $w'_\mathcal{B}$ for $(T'_\mathcal{B})_w$. The DFA $\mathcal{A}'$ recognizes $L\Phi$, and the DFA $\mathcal{B}'$ recognizes $K\Phi$. By the uniformity of Ψ, we can write $Q'_\mathcal{A} = Q'_\mathcal{B} = Q'$, $1'_\mathcal{A} = 1'_\mathcal{B} = 1'$, and $F'_\mathcal{A} = F'_\mathcal{B} = F'$.

Now, we want to show that $L\Phi = K\Phi\varphi^{-1}$. Since $\mathcal{A} = \mathcal{B}\varphi^{-1}$, for all $q \in Q$ and $a \in \Sigma$ we have $qa_\mathcal{A} = q(a\varphi)_\mathcal{B}$ by definition. Thus $a_\mathcal{A}$ and $(a\varphi)_\mathcal{B}$ are equal as transformations of Q for all $a \in \Sigma$. By the uniformity of Ψ, $a'_\mathcal{A}$ and $(a\varphi)'_\mathcal{B}$ are equal as transformations of Q'. It follows that $w'_\mathcal{A}$ and $(w\varphi)'_\mathcal{B}$ are equal as transformations of Q' for all $w \in \Sigma^*$. Hence we have

$$w \in L\Phi \iff 1'w'_\mathcal{A} \in F' \iff 1'(w\varphi)'_\mathcal{B} \in F' \iff w\varphi \in K\Phi \iff w \in K\Phi\varphi^{-1}.$$

This proves that $L\Phi = K\Phi\varphi^{-1}$.

(2) $\implies$ (1): We are given a regular operation Φ. We want to produce a uniform DFA operation Ψ such that for all DFAs $\mathcal{A}$, we have $L(\mathcal{A}\Psi) = L(\mathcal{A})\Phi$.

Fix an n-state DFA $\mathcal{A} = (Q, \Sigma, T, 1, F)$ and let L be its language. We define $\mathcal{A}\Psi$ as follows. By Proposition 3.3.3, we have $L = T_Q(1, F)\varphi_L^{-1}$, where $\varphi_L\colon \Sigma^* \to T_Q^*$ is the standard transformation map of L. By assumption, we then have $L\Phi = T_Q(1, F)\Phi\varphi_L^{-1}$. Let $\mathcal{D}' = (Q', T_Q, T', 1', F')$ be a minimal DFA for $T_Q(1, F)\Phi$, and set $\mathcal{A}\Psi = \mathcal{D}'\varphi_L^{-1}$.

It is clear that we have $L(\mathcal{A}\Psi) = T_Q(1, F)\Phi\varphi_{L(\mathcal{A})}^{-1} = L(\mathcal{A})\Phi$ as required. To see that Ψ is uniform, fix DFAs $\mathcal{A} = (Q, \Sigma, T_\mathcal{A}, 1, F)$ and $\mathcal{B} = (Q, \Gamma, T_\mathcal{B}, 1, F)$. We compute the images $\mathcal{A}\Psi = \mathcal{A}' = (Q'_\mathcal{A}, \Sigma, T'_\mathcal{A}, 1'_\mathcal{A}, F'_\mathcal{A})$ and $\mathcal{B}\Psi = \mathcal{B}' = (Q'_\mathcal{B}, \Gamma, T'_\mathcal{B}, 1'_\mathcal{B}, F'_\mathcal{B})$. Now, let $\mathcal{D}' =$

$(Q', T_Q, T'_{\mathcal{D}}, 1', F')$ be the minimal DFA for $T_Q(1, F)\Phi$. By definition, we have $\mathcal{A}' = \mathcal{D}'\varphi^{-1}_{L(\mathcal{A})}$ and $\mathcal{B}' = \mathcal{D}'\varphi^{-1}_{L(\mathcal{B})}$. So $\mathcal{A}'$ and $\mathcal{B}'$ both have the same state configuration as $\mathcal{D}'$. It follows that $(Q'_{\mathcal{A}}, 1'_{\mathcal{A}}, F'_{\mathcal{A}}) = (Q'_{\mathcal{B}}, 1'_{\mathcal{B}}, F'_{\mathcal{B}})$, as required.

Next, fix $a \in \Sigma$ and $b \in \Gamma$ such that $(T_{\mathcal{A}})_a = (T_{\mathcal{B}})_b$. We have $q(T'_{\mathcal{A}})_a = q(T'_{\mathcal{D}})_{a\varphi_{L(\mathcal{A})}}$ for all q. Also, $q(T'_{\mathcal{B}})_b = q(T'_{\mathcal{D}})_{b\varphi_{L(\mathcal{B})}}$ for all q. By the definition of the standard transformation map, we have $a\varphi_{L(\mathcal{A})} = b\varphi_{L(\mathcal{B})}$, since $(T_{\mathcal{A}})_a = (T_{\mathcal{B}})_b$. It follows that $(T'_{\mathcal{A}})_a = (T'_{\mathcal{B}})_b$, as required. Thus Ψ is uniform. □

3.3.3 Main Theorem: The Unary Case

Now we prove that the OLPA approach works for all uniform unary operations. Thanks to Proposition 3.3.5, this is not especially difficult.

The following lemma formalizes a "weak" version of the OLPA approach for unary operations. The short proof contains all the essential ideas, but the expression it gives for the state complexity function is not practical, since it involves taking a maximum over all sets of size n. Obtaining a more practical expression for the state complexity function is just a technical matter that we will deal with after this proof.

Lemma 3.3.6. *Let Φ be a uniform unary regular operation. Let L be a regular language recognized by a DFA $(Q, \Sigma, T, 1, F)$. Then $\mathrm{sc}(L\Phi) \leq \mathrm{sc}(T_Q(1, F)\Phi)$. In particular, the state complexity of Φ is given by the following function:*

$$n \mapsto \max\{\mathrm{sc}(T_Q(1, F)\Phi) : |Q| = n, 1 \in Q, F \subseteq Q\}.$$

Proof. Fix L, and recall that $L = T_Q(1, F)\varphi^{-1}_L$, where φ_L is the standard transformation morphism of L. Since Φ is uniform, we have $L\Phi = T_Q(1, F)\Phi\varphi^{-1}_L$ by Proposition 3.3.5. Recall that an inverse homomorphic image of a regular language has state complexity bounded by that of the original language; thus $\mathrm{sc}(L\Phi) \leq \mathrm{sc}(T_Q(1, F)\Phi)$ as required. □

Now, we show that to compute the state complexity function, it suffices to just consider the set $\{1, \dots, n\}$ instead of all sets of size n. Furthermore, we show that we may assume that F is either $\{1, \dots, k\}$ or $\{n - k + 1, \dots, n\}$ for some $k \leq n$. Thus it suffices to just check $2n$ OLPA witnesses. Let $F_{n,k,1} = \{1, \dots, k\}$ and let $F_{n,k,0} = \{n - k + 1, \dots, n\}$.

Theorem 3.3.7. *Let Φ be a uniform unary regular operation. Let L be a regular language recognized by a DFA $(Q, \Sigma, T, 1, F)$. Then $\mathrm{sc}(L\Phi) \leq \mathrm{sc}(T_n(1, F_{n,k,j})\Phi)$, where $n = |Q|$,*

$k = |F|$, and j is defined to be 1 if $1 \in F$ and 0 if $1 \notin F$. The state complexity of Φ is the following function:

$$n \mapsto \max\{\mathrm{sc}(T_n(1, F_{n,k,j})\Phi) : 0 \leq j \leq 1, 0 + j \leq k \leq n - 1 + j\}.$$

Proof. We know from Lemma 3.3.6 that $\mathrm{sc}(L\Phi) \leq \mathrm{sc}(T_Q(1, F)\Phi)$. We will prove that $\mathrm{sc}(T_Q(1, F)\Phi) \leq \mathrm{sc}(T_n(1, F_{n,k,j})\Phi)$, with n, k and j defined as in the statement of the theorem.

By the uniformity of Φ, it suffices to exhibit a 1-uniform homomorphism $\varphi\colon T_Q^* \to T_n^*$ such that $T_Q(1, F) = T_n(1, F_{n,k,j})\varphi^{-1}$. To define φ, first we define a bijection $\beta\colon Q \to \{1, \dots, n\}$. We take β to be a bijection with the following properties: $1\beta = 1$ and $F\beta = F_{n,k,j}$. The remaining elements of Q are mapped to the remaining elements of $\{1, \dots, n\}$ arbitrarily. Note that this definition of β is only possible because of our choice of the parameter j. Indeed, we are mapping 1β to 1, so if $i \in F$ then we better have $1 \in F_{n,k,j}$; but $i \in F$ implies $j = 1$ and thus $F_{n,k,j} = \{1, \dots, k\}$. On the other hand, if $i \notin F$ then we better have $1 \notin F_{n,k,j}$, but in this case we have $j = 0$ giving $F_{n,k,j} = \{n - k + 1, \dots, n\}$. Given this definition of β, for each $t\colon Q \to Q$, we define $t\varphi$ be the transformation of $\{1, \dots, n\}$ that sends m to $m\beta^{-1}t\beta$.

We show that $w \in T_Q(1, F)$ if and only if $w \in T_n(1, F_{n,k,j})\varphi^{-1}$. Let $w = t_1 \cdots t_k$ for $t_1, \dots, t_k \in T_Q$.

$$\begin{aligned} w \in T_Q(1, F) &\iff 1w \in F \iff 1\beta^{-1}w \in F \iff 1\beta^{-1}w\beta \in F\beta \\ &\iff 1\beta^{-1}w\beta \in F_{n,k,j} \iff 1\beta^{-1}t_1t_2 \cdots t_k\beta \in F_{n,k,j} \\ &\iff 1\beta^{-1}t_1\beta\beta^{-1}t_2\beta \cdots \beta^{-1}t_k\beta \in F_{n,k,j} \\ &\iff 1(t_1\varphi)(t_2\varphi) \cdots (t_k\varphi) \in F_{n,k,j} \iff 1(w\varphi) \in F_{n,k,j} \\ &\iff w\varphi \in T_n(1, F_{n,k,j}) \iff w \in T_n(1, F_{n,k,j})\varphi^{-1}. \end{aligned}$$

Thus $T_Q(1, F) = T_n(1, F_{n,k,j})\varphi^{-1}$, as required. This completes the proof. □

3.4 Multiary Operations

We now extend our formalization of the OLPA approach to operations of arbitrary arity. We must define new witnesses, extend the notion of uniformity to multiary operations, and prove that our witnesses maximize the state complexity of uniform operations.

3.4.1 Full Transformation Tuple Languages

Full transformation languages do not suffice as OLPA witnesses for operations of arity greater than one. When applying the OLPA approach to operations of arity m, we want to use an m-tuple $(\mathcal{D}_1, \dots, \mathcal{D}_m)$ of DFAs (where $\mathcal{D}_j = (Q_j, \Sigma, T_j, 1, F_j)$ for $1 \leq j \leq m$) with the following property: for each m-tuple $(t_1 \colon Q_1 \to Q_1, \dots, t_m \colon Q_m \to Q_m)$, there exists a letter $a \in \Sigma$ such that a induces transformation t_j in $\mathcal{D}_j$ for $1 \leq j \leq m$. That is, we have one letter for every possible combination of actions across all the input DFAs.

For this purpose, we define *transformation tuple languages*. Let $Q_1, \dots, Q_m$ be finite sets and let $\mathbb{T} = T_{Q_1} \times \cdots \times T_{Q_m}$. For $\Sigma \subseteq \mathbb{T}$, j with $1 \leq j \leq m$, $1 \in Q_j$, and $F \subseteq Q_j$, the *transformation tuple language* $\Sigma_j(1, F)$ is the language of the DFA $(Q_j, \Sigma, T, 1, F)$ where $T = \{(q, (t_1, \dots, t_m), qt_j) : q \in Q_j, (t_1, \dots, t_m) \in \Sigma\}$. This DFA is called the *standard DFA* of the transformation tuple language.

Definition 3.4.1. *A transformation tuple language of the form $\mathbb{T}_j(1, F)$ is called a full transformation tuple language.*

The full transformation tuple languages are our OLPA witnesses for m-ary operations. There is also a generalization of Proposition 3.3.3 for full transformation tuple languages.

Definition 3.4.2. Let $(L_1, \dots, L_m)$ be an m-tuple of regular languages over Σ, where L_j is recognized by the DFA $\mathcal{D}_j = (Q_j, \Sigma, T_j, 1, F_j)$ for $1 \leq j \leq m$. Let $\mathbb{T} = T_{Q_1} \times \cdots \times T_{Q_m}$ as before. The *standard transformation tuple map* of $(L_1, \dots, L_m)$ (with respect to $(\mathcal{D}_1, \dots, \mathcal{D}_m)$), denoted $\varphi_{(L_1, \dots, L_m)} \colon \Sigma^* \to \mathbb{T}^*$, is defined by $a\varphi_{(L_1, \dots, L_m)} = ((T_1)_a, \dots, (T_m)_a)$.

As shorthand, let $\varphi = \varphi_{(L_1, \dots, L_m)}$ in the following proposition and proof.

Proposition 3.4.3. *We have $(L_1, \dots, L_m) = (\mathbb{T}_1(1, F_1)\varphi^{-1}, \dots, \mathbb{T}_m(1, F_m)\varphi^{-1})$.*

Proof. It suffices to show for all $w \in \Sigma^*$ that $w \in L_j \iff w\varphi \in \mathbb{T}_j(1, F_j)$. Fix j and let $(Q_j, \mathbb{T}, T, 1, F_j)$ be the standard DFA of $\mathbb{T}_j(1, F_j)$. Then we have

$$w \in L_j \iff 1(T_j)_w \in F_j \iff 1T_{w\varphi} \in F_j \iff w\varphi \in \mathbb{T}_j(1, F_j),$$

as required.

The second two-way implication may not be obvious. To see that it holds, first note that if w is empty, then $(T_j)_w$ and $T_{w\varphi}$ are both the identity map on Q_j. Otherwise, suppose $w = a_1 \cdots a_k$ with $a_1, \dots, a_k \in \Sigma$. We may write $w\varphi = (a_1\varphi) \cdots (a_k\varphi)$, and thus

$T_{w\varphi} = T_{a_1\varphi} \cdots T_{a_k\varphi}$. By definition, we have $a_i\varphi = ((T_1)_{a_i}, \dots, (T_m)_{a_i})$ for $1 \leq i \leq k$. This m-tuple of transformations is a "letter" of the alphabet $\mathbb{T} = T_{Q_1} \times \cdots T_{Q_m}$. Then $T_{a_i\varphi}$ is the transformation of Q_j induced by the "letter" $a_i\varphi$. By definition, this induced transformation is the map $q \mapsto q(T_j)_{a_i}$ for $q \in Q_j$. Thus $T_{a_i\varphi} = (T_j)_{a_i}$ for $1 \leq i \leq k$. It follows that

$$T_{w\varphi} = T_{a_1\varphi} \cdots T_{a_k\varphi} = (T_j)_{a_1} \cdots (T_j)_{a_k} = (T_j)_w.$$

Hence the implication holds. □

3.4.2 Uniform Operations

We now generalize the notion of uniformity to operations of higher arity. We say an m-ary regular operation Φ is equivalent to an m-ary DFA operation Ψ if for all m-tuples of DFAs $(\mathcal{D}_1, \dots, \mathcal{D}_m)$, we have $L((\mathcal{D}_1, \dots, \mathcal{D}_m)\Psi) = (L(\mathcal{D}_1), \dots, L(\mathcal{D}_m))\Phi$.

Definition 3.4.4. An m-ary DFA operation Ψ is *uniform* if for every pair of m-tuples of DFAs $(\mathcal{A}_1, \cdots, \mathcal{A}_m)$ and $(\mathcal{B}_1, \dots, \mathcal{B}_m)$, where for each j with $1 \leq j \leq m$, the DFAs $\mathcal{A}_j$ and $\mathcal{B}_j$ have the same state configuration, the DFA $\mathcal{A}_j$ has alphabet Σ and transition set $T_{\mathcal{A}_j}$, and the DFA $\mathcal{B}_j$ has transition set $T_{\mathcal{B}_j}$ and alphabet Γ; the image DFAs $\mathcal{A} = (\mathcal{A}_1, \cdots, \mathcal{A}_m)\Psi$ and $\mathcal{B} = (\mathcal{B}_1, \dots, \mathcal{B}_m)\Psi$ have transition sets $T_{\mathcal{A}}$ and $T_{\mathcal{B}}$ respectively, and the following conditions hold:

1. The image DFAs $\mathcal{A}$ and $\mathcal{B}$ have the same state configuration.
2. If there exist letters $a \in \Sigma$ and $b \in \Gamma$ such that $(T_{\mathcal{A}_j})_a = (T_{\mathcal{B}_j})_b$ for each j with $1 \leq j \leq m$, then $(T_{\mathcal{A}})_a = (T_{\mathcal{B}})_b$.

An m-ary regular operation Φ is *uniform* if it is equivalent to a uniform m-ary DFA operation.

While this definition may seem complicated, it is simply a natural generalization of the unary definition. As in the unary case, we have a language-theoretic characterization of uniformity. Recall that a monoid homomorphism $\varphi \colon \Sigma^* \to \Gamma^*$ is called *1-uniform* if it maps letters to letters.

Proposition 3.4.5. *The following are equivalent:*

1. *The m-ary regular operation Φ is uniform.*

2. *For all* 1*-uniform homomorphisms* $\varphi\colon \Sigma^* \to \Gamma^*$*, and all* m*-tuples* $(L_1, \dots, L_m)$ *and* $(K_1, \dots, K_m)$ *where* L_j *is a regular language over* Σ *and* K_j *is a regular language over* Γ *for* $1 \leq j \leq m$*, if* $L_j = K_j\varphi^{-1}$ *for* $1 \leq j \leq m$*, then* $(L_1, \dots, L_m)\Phi = (K_1, \dots, K_m)\Phi\varphi^{-1}$.

The proof is very similar to the proof of Proposition 3.3.5, except the general definition of uniformity is used and full transformation tuple languages are used instead of full transformation languages.

Proof. (1) $\implies$ (2): Since Φ is uniform, there is a uniform DFA operation Ψ equivalent to Φ. Fix a 1-uniform homomorphism $\varphi\colon \Sigma^* \to \Gamma^*$ such that $L_j = K_j\varphi^{-1}$ for $1 \leq j \leq m$. We want to show that $(L_1, \dots, L_m)\Phi = (K_1, \dots, K_m)\Phi\varphi^{-1}$.

Since $L_j = K_j\varphi^{-1}$, for each j we can find a DFA $\mathcal{A}_j$ for L_j and a DFA $\mathcal{B}_j$ for K_j such that $\mathcal{A}_j = \mathcal{B}_j\varphi^{-1}$. Each pair of DFAs $\mathcal{A}_j$ and $\mathcal{B}_j$ has a common state configuration $(Q_j, 1, F_j)$. For $1 \leq j \leq m$, let $\mathcal{A}_j = (Q_j, \Sigma, T_{\mathcal{A}_j}, 1, F_j)$ be the DFA for L_j and let $\mathcal{B}_j = (Q_j, \Gamma, T_{\mathcal{B}_j}, 1, F_j)$ be the DFA for K_j. By the uniformity of Ψ, the image DFAs $\mathcal{A} = (\mathcal{A}_1, \dots, \mathcal{A}_m)\Psi$ and $\mathcal{B} = (\mathcal{B}_1, \dots, \mathcal{B}_m)\Psi$ have a common state configuration $(Q, 1, F)$. Write $\mathcal{A} = (Q, \Sigma, T_{\mathcal{A}}, 1, F)$ and $\mathcal{B} = (Q, \Sigma, T_{\mathcal{B}}, 1, F)$.

Since $\mathcal{A}_j = \mathcal{B}_j\varphi^{-1}$ for $1 \leq j \leq m$, for all $q \in Q_j$ and $a \in \Sigma$, we have $q(T_{\mathcal{A}_j})_a = q(T_{\mathcal{B}_j})_{a\varphi}$ by definition. Thus $(T_{\mathcal{A}_j})_a = (T_{\mathcal{B}_j})_{a\varphi}$ for all $a \in \Sigma$ and all $1 \leq j \leq m$. By the uniformity of Ψ, it follows that $(T_{\mathcal{A}})_a = (T_{\mathcal{B}})_{a\varphi}$. Hence $(T_{\mathcal{A}})_w = (T_{\mathcal{B}})_{w\varphi}$ for all $w \in \Sigma^*$. Thus we have

$$\begin{aligned} w \in (L_1, \dots, L_m)\Phi &\iff 1(T_{\mathcal{A}})_w \in F \iff 1(T_{\mathcal{B}})_{w\varphi} \in F \\ &\iff w\varphi \in (K_1, \dots, K_m)\Phi \iff w \in (K_1, \dots, K_m)\Phi\varphi^{-1}. \end{aligned}$$

This proves that $(L_1, \dots, L_m)\Phi = (K_1, \dots, K_m)\Phi\varphi^{-1}$.

(2) $\implies$ (1): We want to produce a uniform m-ary DFA operation Ψ such that for all tuples of DFAs $(\mathcal{A}_1, \dots, \mathcal{A}_m)$ over a common alphabet, we have $L((\mathcal{A}_1, \dots, \mathcal{A}_m)\Psi) = (L(\mathcal{A}_1), \dots, L(\mathcal{A}_m))\Phi$.

Fix a tuple $(\mathcal{A}_1, \dots, \mathcal{A}_m)$ of DFAs over Σ, where $\mathcal{A}_j$ has state configuration $(Q_j, 1, F_j)$, and let $L_j = L(\mathcal{A}_j)$ for $1 \leq j \leq m$. We define the image $\mathcal{A} = (\mathcal{A}_1, \dots, \mathcal{A}_m)\Psi$ as follows. By Proposition 3.4.3 we have $L_j = \mathbb{T}_j(1, F_j)\varphi^{-1}_{(L_1,\dots,L_m)}$, where $\varphi_{(L_1,\dots,L_m)}$ is the standard transformation tuple map of $(L_1, \dots, L_m)$ with respect to $(\mathcal{A}_1, \dots, \mathcal{A}_m)$, and $\mathbb{T} = T_{Q_1} \times \dots \times T_{Q_m}$. Let $\mathcal{D}$ be a minimal DFA for $(\mathbb{T}_1(1, F_1), \dots, \mathbb{T}_m(1, F_m))\Phi$, and set $(\mathcal{A}_1, \dots, \mathcal{A}_m)\Psi = \mathcal{D}\varphi^{-1}_{(L_1,\dots,L_m)}$.

We claim that $L((\mathcal{A}_1, \dots, \mathcal{A}_m)\Psi) = (L(\mathcal{A}_1), \dots, L(\mathcal{A}_m))\Phi$. Indeed, since we have $L_j = \mathbb{T}_j(1, F_j)\varphi^{-1}_{(L_1,\dots,L_m)}$, we have

$$(L_1, \dots, L_m)\Phi = (\mathbb{T}_1(1, F_1), \dots, \mathbb{T}_m(1, F_m))\Phi\varphi^{-1}_{(L_1,\dots,L_m)}.$$

It follows that

$$\begin{aligned} L((\mathcal{A}_1, \dots, \mathcal{A}_m)\Psi &= L(\mathcal{D}\varphi^{-1}_{(L_1,\dots,L_m)}) = (\mathbb{T}_1(1, F_1), \dots, \mathbb{T}_m(1, F_m))\Phi\varphi^{-1}_{(L_1,\dots,L_m)} \\ &= (L_1, \dots, L_m)\Phi = (L(\mathcal{A}_1), \dots, L(\mathcal{A}_m))\Phi, \end{aligned}$$

as required.

To see that Ψ is uniform, fix m-tuples of DFAs $(\mathcal{A}_1, \dots, \mathcal{A}_m)$ and $(\mathcal{B}_1, \dots, \mathcal{B}_m)$ such that for $1 \leq j \leq m$, the DFAs $\mathcal{A}_j$ and $\mathcal{B}_j$ have the same state configuration $(Q_j, 1, F_j)$, the DFA $\mathcal{A}_j$ has alphabet Σ and transition set $T_{\mathcal{A}_j}$, and the DFA $\mathcal{B}_j$ has alphabet Γ and transition set $T_{\mathcal{B}_j}$. Write $(\mathcal{A}_1, \dots, \mathcal{A}_m)\Psi = \mathcal{A} = (Q_{\mathcal{A}}, \Sigma, T_{\mathcal{A}}, 1, F_{\mathcal{A}})$ and $(\mathcal{B}_1, \dots, \mathcal{B}_m)\Psi = \mathcal{B} = (Q_{\mathcal{B}}, \Gamma, T_{\mathcal{B}}, 1, F_{\mathcal{B}})$. Let $\mathcal{D}$ be the minimal DFA for $(\mathbb{T}_1(1, F_1), \dots, \mathbb{T}_m(1, F_m))\Phi$ used in the definition of Ψ. Then $\mathcal{A}$ and $\mathcal{B}$ are both inverse homomorphism DFAs constructed from $\mathcal{D}$, so they both have the same state configuration as $\mathcal{D}$. Write $(Q, 1, F)$ for this common state configuration.

It remains to show that whenever we have $a \in \Sigma$ and $b \in \Gamma$ such that $(T_{\mathcal{A}_j})_a = (T_{\mathcal{B}_j})_b$ for $1 \leq j \leq m$, it follows that $(T_{\mathcal{A}})_a = (T_{\mathcal{B}})_b$. Fix $a \in \Sigma$ and $b \in \Gamma$ with this property. Write $\varphi_{\mathcal{A}}$ as shorthand for $\varphi_{(L(\mathcal{A}_1),\dots,L(\mathcal{A}_m))}$, and write $\varphi_{\mathcal{B}}$ for $\varphi_{(L(\mathcal{B}_1),\dots,L(\mathcal{B}_m))}$. By definition, we have $\mathcal{A} = \mathcal{D}\varphi^{-1}_{\mathcal{A}}$ and $\mathcal{B} = \mathcal{D}\varphi^{-1}_{\mathcal{B}}$. Let $T_{\mathcal{D}}$ be the transition set of $\mathcal{D}$. Then for $q \in Q$, we have $q(T_{\mathcal{A}})_a = q(T_{\mathcal{D}})_{a\varphi_{\mathcal{A}}}$ and $q(T_{\mathcal{B}})_b = q(T_{\mathcal{D}})_{b\varphi_{\mathcal{B}}}$. By the definition of the standard transformation tuple map, we have $a\varphi_{\mathcal{A}} = ((T_{\mathcal{A}_1})_a, \dots, (T_{\mathcal{A}_m})_a)$ and $b\varphi_{\mathcal{B}} = ((T_{\mathcal{B}_1})_b, \dots, (T_{\mathcal{B}_m})_b)$. But we are assuming that $(T_{\mathcal{A}_j})_a = (T_{\mathcal{B}_j})_b$ for $1 \leq j \leq m$, so in fact $a\varphi_{\mathcal{A}} = b\varphi_{\mathcal{B}}$. It then follows that $(T_{\mathcal{A}})_a = (T_{\mathcal{B}})_b$, as required.

This proves that Ψ is uniform, and thus Φ is uniform, since it is equivalent to a uniform DFA operation. $\square$

3.4.3 Main Theorem: The General Case

We now consider uniform operations of arbitrary arity. The proof strategies in this case are much the same, except full transformation tuple languages are used as witnesses, rather than full transformation languages.

Lemma 3.4.6. *Let Φ be a uniform m-ary regular operation. Let $(L_1, \dots, L_m)$ be regular languages, where L_j is recognized by a DFA $(Q_j, \Sigma, T_j, 1, F_j)$. Let $\mathbb{T} = T_{Q_1} \times \cdots \times T_{Q_m}$. Then* $\operatorname{sc}((L_1, \dots, L_m)\Phi) \leq \operatorname{sc}((\mathbb{T}_1(1, F_1), \dots, \mathbb{T}_m(1, F_m))\Phi)$.

Proof. Fix $(L_1, \dots, L_m)$, and recall from Proposition 3.4.3 that we have $(L_1, \dots, L_m) = (\mathbb{T}_1(1, F_1)\varphi^{-1}, \dots, \mathbb{T}_m(1, F_m)\varphi^{-1})$, where the homomorphism $\varphi = \varphi_{(L_1,\dots,L_m)}$ is the standard transformation tuple map of $(L_1, \dots, L_m)$. Since Φ is uniform, we have

$$(L_1, \dots, L_m)\Phi = (\mathbb{T}_1(1, F_1), \dots, \mathbb{T}_m(1, F_m))\Phi\varphi^{-1},$$

by Proposition 3.4.5. Thus we have

$$\operatorname{sc}((L_1, \dots, L_m)\Phi) \leq \operatorname{sc}((\mathbb{T}_1(1, F_1), \dots, \mathbb{T}_m(1, F_m)\Phi),$$

as required. □

As before, it suffices to only check a finite number of witnesses. Recall that we defined $T_n = T_{\{1,\dots,n\}}$, and for $k \leq n$ we defined $F_{n,k,1} = \{1, \dots, k\}$ and $F_{n,k.0} = \{n - k + 1, \dots, n\}$.

Theorem 3.4.7 (The "Fundamental Theorem of the OLPA Approach")**.** *Let Φ be a uniform m-ary regular operation. Let $(L_1, \dots, L_m)$ be regular languages, where L_j is recognized by a DFA $(Q_j, \Sigma, T_j, 1, F_j)$. Let $n_j = |Q_j|$ and let $\mathbb{T} = T_{n_1} \times \dots T_{n_m}$. Then we have*

$$\operatorname{sc}((L_1, \dots, L_m)\Phi) \leq \operatorname{sc}((\mathbb{T}_1(1, F_{n_1,k_1,\ell_1}), \dots, \mathbb{T}_m(1, F_{n_m,k_m,\ell_m}))\Phi),$$

where $k_j = |F_j|$, and ℓ_j is defined to be 1 if $1 \in F_j$ and 0 if $1 \notin F_j$. The state complexity of Φ is the function

$$(n_1, \dots, n_m) \mapsto \max \operatorname{sc}((\mathbb{T}_1(1, F_{n_1,k_1,\ell_1}), \dots, \mathbb{T}_m(1, F_{n_m,k_m,\ell_m}))\Phi),$$

where the maximum is taken over all possible values in the following ranges: $1 \leq j \leq m$, $0 \leq \ell_j \leq 1$, and $0 + \ell_j \leq k_j \leq n_j - 1 + \ell_j$.

To compute the worst-case state complexity of an m-ary operation for m input DFAs of sizes n_1 through n_m, we use $2(n_1 + \cdots + n_m)$ different languages, each with an alphabet of size $n_1^{n_1} \cdots n_m^{n_m}$. The number of input m-tuples that must be tested is $2^m n_1 \cdots n_m$, since for the j-th component there are $2n_j$ choices.

Proof. Define $\mathbb{T}' = T_{Q_1} \times \ldots \times T_{Q_m}$. We know from Lemma 3.4.6 that

$$\mathrm{sc}((L_1, \ldots, L_m)\Phi) \leq \mathrm{sc}((\mathbb{T}'_1(1, F_1), \ldots, \mathbb{T}'_m(1, F_m))\Phi).$$

Let us prove the following:

$$\mathrm{sc}((\mathbb{T}'_1(1, F_1), \ldots, \mathbb{T}'_m(1, F_m))\Phi) \leq \mathrm{sc}((\mathbb{T}_1(1, F_{n_1,k_1,\ell_1}), \ldots, \mathbb{T}_m(1, F_{n_m,k_m,\ell_m}))\Phi),$$

where n_j, k_j, ℓ_j for $1 \leq j \leq m$ are defined as in the statement of the theorem.

Since Φ is uniform, it suffices to exhibit a homomorphism $\varphi\colon (\mathbb{T}')^* \to \mathbb{T}^*$ such that $\mathbb{T}'_j(1, F_j) = \mathbb{T}_j(1, F_{n_j,k_j,\ell_j})\varphi^{-1}$ for $1 \leq j \leq m$. To define φ, first we define bijections $\beta_j\colon Q_j \to \{1, \ldots, n_j\}$ for $1 \leq j \leq m$. As in the proof of Theorem 3.3.7, we take each β_j to be a bijection with the following properties: $1\beta = 1$ and $F_j\beta = F_{n_j,k_j,\ell_j}$. The remaining elements of Q_j are mapped to the remaining elements of $\{1, \ldots, n_j\}$ arbitrarily. Then for each tuple $(t_1, \ldots, t_m) \in \mathbb{T}'$, we define $(t_1, \ldots, t_m)\varphi$ to be the transformation tuple $(\beta_1^{-1}t_1\beta_1, \ldots, \beta_m^{-1}t_m\beta_m)$, which lies in $\mathbb{T}$.

Now, for $1 \leq j \leq m$, we show that $w \in \mathbb{T}'_j(1, F_j) \iff w\varphi \in \mathbb{T}_j(1, F_{n_j,k_j,\ell_j})$. Let $w = (t_{1,1}, \ldots, t_{m,1}) \cdots (t_{1,k}, \ldots, t_{m,k})$, where each of these transformation tuples lies in $\mathbb{T}'$.

$$\begin{aligned} w \in \mathbb{T}'_j(1, F_j) &\iff 1w \in F_j \iff 1t_{j,1}t_{j,2}\cdots t_{j,k} \in F_j \\ &\iff 1\beta_j^{-1}t_{j,1}t_{j,2}\cdots t_{j,k}\beta_j \in F_{n_j,k_j,\ell_j} \\ &\iff 1\beta_j^{-1}t_{j,1}\beta_j\beta_j^{-1}t_{j,2}\beta_j\cdots\beta_j^{-1}t_{j,k}\beta_j \in F_{n_j,k_j,\ell_j} \\ &\iff 1w\varphi \in F_{n_j,k_j,\ell_j} \iff w\varphi \in \mathbb{T}_j(1, F_{n_j,k_j,\ell_j}). \end{aligned}$$

Thus $\mathbb{T}'_j(1, F_j) = \mathbb{T}_j(1, F_{n_j,k_j,\ell_j})\varphi^{-1}$, as required. □

3.5 Uniform and Non-Uniform Operations

To demonstrate the wide applicability of the OLPA approach, we will now prove that a number of common operations (as well as a few more esoteric ones) are uniform, and that the class of uniform operations is closed under composition. We also give some examples of non-uniform operations.

3.5.1 Uniform Operations

First we consider a class of operations called *shuffles on trajectories* [32, 63]. Operations in this class include shuffle, literal shuffle, balanced literal shuffle, insertion, balanced insertion, concatenation, and anti-concatenation [63, Remark 3.1]. We denote the shuffle of languages L and L' along the set of trajectories $X \subseteq \{0,1\}^*$ by $L \sqcup\!\sqcup_X L'$. The shuffle on trajectories $L \sqcup\!\sqcup_X L'$ is regular if and only if X is regular [63, Theorem 5.1]. For the definition of $L \sqcup\!\sqcup_X L'$, see [63, Section 3]; for the following proof we only need to know the DFA construction.

Proposition 3.5.1. *For all regular languages $X \subseteq \{0,1\}^*$, the shuffle on trajectories operation $(L, L') \mapsto L \sqcup\!\sqcup_X L'$ is uniform.*

Proof. Following [63], we define a DFA operation Ψ equivalent to the shuffle on trajectories operation. Let $\mathcal{D}_1 = (Q_1, \Sigma, T_1, 1, F_1)$ and $\mathcal{D}_2 = (Q_2, \Sigma, T_2, 1, F_2)$ be arbitrary DFAs. Let $\mathcal{D}_X = (Q_X, \{0,1\}, T_X, 1, F_X)$ be a DFA for the set of trajectories X. We set $(\mathcal{D}_1, \mathcal{D}_2)\Psi$ to be the determinization of the following FA $\mathcal{D}$. The FA $\mathcal{D}$ has state set $Q_1 \times Q_X \times Q_2$, alphabet Σ, initial state $(1,1,1)$, final state set $F_1 \times F_X \times F_2$, and transition set T defined as follows: for each $a \in \Sigma$, we have

$$(q_1, q_X, q_2)T_a = \{(q_1(T_1)_a, q_X(T_X)_0, q_2), (q_1, q_X(T_X)_1, q_2(T_2)_a)\}.$$

It was proved in [63, Theorem 5.1] that $L((\mathcal{D}_1, \mathcal{D}_2)\Psi) = L(\mathcal{D}_1) \sqcup\!\sqcup_X L(\mathcal{D}_2)$.

Let $(\mathcal{A}_1, \mathcal{A}_2)$ and $(\mathcal{B}_1, \mathcal{B}_2)$ be pairs of DFAs such that for $1 \leq j \leq 2$, the DFAs $\mathcal{A}_j$ and $\mathcal{B}_j$ have the same state configuration $(Q_j, 1, F_j)$, the DFA $\mathcal{A}_j$ has alphabet Σ and transition set $T_{\mathcal{A}_j}$, and the DFA $\mathcal{B}_j$ has alphabet Γ and transition set $T_{\mathcal{B}_j}$.

It is clear that the image DFAs $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)\Psi$ and $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)\Psi$ will have the same state configuration. Additionally, by inspecting the definitions of the transition sets $T_\mathcal{A}$ of $\mathcal{A}$ and $T_\mathcal{B}$ of $\mathcal{B}$, it is clear that if $(T_{\mathcal{A}_j})_a = (T_{\mathcal{B}_j})_b$ for $a \in \Sigma$, $b \in \Gamma$ and $1 \leq j \leq 2$, then $(T_\mathcal{A})_a = (T_\mathcal{B})_b$. Indeed, let $S \subseteq Q_1 \times Q_X \times Q_2$. Then we have

$$\begin{aligned}
S(T_\mathcal{A})_a &= \bigcup_{(q_1,q_X,q_2)\in S} \{(q_1, q_X, q_2)\}(T_\mathcal{A})_a \\
&= \bigcup_{(q_1,q_X,q_2)\in S} \{(q_1(T_{\mathcal{A}_1})_a, q_X(T_X)_0, q_2), (q_1, q_X(T_X)_1, q_2(T_{\mathcal{A}_2})_a)\} \\
&= \bigcup_{(q_1,q_X,q_2)\in S} \{(q_1(T_{\mathcal{B}_1})_b, q_X(T_X)_0, q_2), (q_1, q_X(T_X)_1, q_2(T_{\mathcal{B}_2})_b)\} \\
&= \bigcup_{(q_1,q_X,q_2)\in S} \{(q_1, q_X, q_2)\}(T_\mathcal{B})_b = S(T_\mathcal{B})_b.
\end{aligned}$$

Thus Ψ is uniform, and it follows that the shuffle on trajectories operation is uniform. □

The above proof illustrates the fact that once one understands the definition of uniformity, it is often easy to determine whether an operation is uniform just by inspecting the DFA construction. There are no difficult ideas in this proof; it is just a statement of a DFA construction and a rudimentary calculation.

One can also use the language-theoretic characterization of uniformity to prove that operations are uniform. Typically, proofs using the language-theoretic characterization require somewhat more thought to write and read, but are shorter and have less of the "boilerplate" needed for DFA-based proofs. The rest of our uniformity proofs will use the language-theoretic characterization.

Proposition 3.5.2. *The reversal operation $L \mapsto L^R$ is uniform.*

Proof. Fix a 1-uniform homomorphism $\varphi\colon \Sigma^* \to \Gamma^*$ and suppose $L = K\varphi^{-1}$. Since φ is 1-uniform, we have $(w^R)\varphi = (w\varphi)^R$ for all $w \in \Sigma^*$. It follows that

$$\begin{aligned} w \in L^R &\iff w^R \in L \iff w^R \in K\varphi^{-1} \iff (w^R)\varphi \in K \\ &\iff (w\varphi)^R \in K \iff w\varphi \in K^R \iff w \in K^R\varphi^{-1}. \end{aligned}$$

Thus $L^R = K^R\varphi^{-1}$. Therefore, by Proposition 3.3.5, reversal is uniform. □

The *cyclic shift* operation [55] is defined by $L^{\text{cyc}} = \{uv : vu \in L\}$.

Proposition 3.5.3. *The cyclic shift operation $L \mapsto L^{\text{cyc}}$ is uniform.*

Proof. Fix a 1-uniform homomorphism $\varphi\colon \Sigma^* \to \Gamma^*$ and suppose $L = K\varphi^{-1}$. We want to show that $L^{\text{cyc}} = K^{\text{cyc}}\varphi^{-1}$.

If $w \in L^{\text{cyc}}$, we can write $w = uv$ for some $u, v \in \Sigma^*$ such that $vu \in L$. Since $L = K\varphi^{-1}$, we have $(vu)\varphi = (v\varphi)(u\varphi) \in K$. Thus $(u\varphi)(v\varphi) = w\varphi \in K^{\text{cyc}}$, and it follows that $L^{\text{cyc}} \subseteq K^{\text{cyc}}\varphi^{-1}$.

If $w \in K^{\text{cyc}}\varphi^{-1}$, then $w\varphi \in K^{\text{cyc}}$. Thus we can write $w\varphi = uv$ for some $u, v \in \Sigma^*$ such that $vu \in K$. Since φ is 1-uniform, w has length $|u| + |v|$. Write $w = xy$ where $|x| = |u|$ and $|y| = |v|$. Then $(yx)\varphi = (y\varphi)(x\varphi) = vu \in K$. It follows that $yx \in L$, which implies $xy = w \in L^{\text{cyc}}$. Hence $L^{\text{cyc}} = K^{\text{cyc}}\varphi^{-1}$. By Proposition 3.3.5, cyclic shift is uniform. □

Proposition 3.5.4. *The star operation $L \mapsto L^*$ is uniform.*

Proof. Fix a 1-uniform homomorphism $\varphi\colon \Sigma^* \to \Gamma^*$ and suppose $L = K\varphi^{-1}$. For a language M, let $t(M)$ be the set of all finite-length tuples of elements of M, and let $\psi_M\colon t(M) \to M^*$ be the map $(w_1, \ldots, w_n)\psi_M = w_1 \cdots w_n$ (the empty tuple is sent to ε). We claim that $w\psi_L^{-1} \neq \emptyset \iff (w\varphi)\psi_K^{-1} \neq \emptyset$. Indeed, if $(w_1, \ldots, w_n) \in w\psi_L^{-1}$ then $(w_1\varphi, \ldots, w_n\varphi) \in (w\varphi)\psi_K^{-1}$. Conversely, if we have $(x_1, \ldots, x_n) \in (w\varphi)\psi_K^{-1}$, then $w\varphi = x_1 \cdots x_n$. Since φ is 1-uniform, we can write $w = w_1 \cdots w_n$ with $|w_j| = |x_j|$ and $w_j\varphi = x_j$ for $1 \leq j \leq n$. Then since $x_j = w_j\varphi \in K \implies w_j \in K\varphi^{-1} = L$, we have $(w_1, \ldots, w_n) \in w\varphi_L^{-1}$ as required. It follows that:

$$w \in L^* \iff w\psi_L^{-1} \neq \emptyset \iff (w\varphi)\psi_K^{-1} \neq \emptyset \iff w\varphi \in K^* \iff w \in K^*\varphi^{-1}.$$

Thus $L^* = K^*\varphi^{-1}$. Therefore, by Proposition 3.3.5, star is uniform. □

An m-ary *boolean function* is a function $\beta\colon \{0,1\}^m \to \{0,1\}$. Each m-ary boolean function defines a corresponding m-ary *boolean operation* on languages over Σ^*, as follows. For $L \subseteq \Sigma^*$, let $\chi_L\colon \Sigma^* \to \{0,1\}$ be the *characteristic function* of L: if $w \in L$ then $w\chi_L = 1$, and if $w \notin L$ then $w\chi_L = 0$. Then we define

$$(L_1, \ldots, L_m)\beta = \{w \in \Sigma^* : (w\chi_{L_1}, \ldots, w\chi_{L_m})\beta = 1\}.$$

Examples of commonly used boolean operations on languages include union and intersection (m-ary for $m \geq 2$), difference and symmetric difference (binary), and complement (unary).

Proposition 3.5.5. *Boolean operations on languages are uniform.*

Proof. Let β be an m-ary boolean operation on languages. Fix a 1-uniform homomorphism $\varphi\colon \Sigma^* \to \Gamma^*$ and suppose $L_j = K_j\varphi^{-1}$ for $1 \leq j \leq m$. We have

$$\begin{aligned} w \in (L_1, \ldots, L_m)\beta &\iff (w\chi_{L_1}, \ldots, w\chi_{L_m})\beta = 1 \\ &\iff (w\chi_{K_1\varphi^{-1}}, \ldots, w\chi_{K_m\varphi^{-1}})\beta = 1 \\ &\iff (w\varphi\chi_{K_1}, \ldots, w\varphi\chi_{K_m})\beta = 1 \\ &\iff w\varphi \in (K_1, \ldots, K_m)\beta \iff w \in (K_1, \ldots, K_m)\beta\varphi^{-1}. \end{aligned}$$

Therefore, by Proposition 3.4.5, β is uniform. □

We have seen that binary concatenation is uniform, since concatenation belongs to the class of shuffles on trajectories. Next we give a direct proof that m-ary concatenation is uniform.

Proposition 3.5.6. *The m-ary concatenation operation $(L_1, \ldots, L_m) \mapsto L_1 \cdots L_m$ is uniform.*

Proof. Fix a 1-uniform homomorphism $\varphi\colon \Sigma^* \to \Gamma^*$ and suppose $L_j = K_j\varphi^{-1}$ for $1 \leq j \leq m$. Define $\psi_L\colon L_1 \times \cdots \times L_m \to L_1 \cdots L_m$ by $(w_1, \ldots, w_m)\psi_L = w_1 \cdots w_m$ and similarly define $\psi_K\colon K_1 \times \cdots \times K_m \to K_1 \cdots K_m$. Using similar arguments to the proof of Proposition 3.5.4, we can show that $w\psi_L^{-1} \neq \emptyset \iff (w\varphi)\psi_K^{-1} \neq \emptyset$. Thus:

$$w \in L_1 \cdots L_m \iff w\psi_L^{-1} \neq \emptyset \iff (w\varphi)\psi_K^{-1} \neq \emptyset \iff w\varphi \in K_1 \cdots K_m.$$

Therefore, by Proposition 3.4.5, m-ary concatenation is uniform. □

Next, we show that the class of uniform operations is closed under composition. It is easy to see that this holds for unary uniform operations: if Φ and Φ' are uniform and $L = K\varphi^{-1}$ for a 1-uniform homomorphism φ, then $L\Phi = K\Phi\varphi^{-1}$ and subsequently $(L\Phi)\Phi' = (K\Phi)\Phi'\varphi^{-1}$. The general case is not much harder; the only difficulty is in dealing with the notation.

Proposition 3.5.7. *Let Φ be an m-ary uniform operation, and let $\Phi_1, \ldots, \Phi_m$ be uniform operations where Φ_j has arity n_j. Set $N_j = n_1 + \cdots + n_j$ for $1 \leq j \leq m$, and consider the operation of arity N_m that maps $(L_1, \ldots, L_{N_m})$ to*

$$((L_1, \ldots, L_{N_1})\Phi_1, (L_{N_1+1}, \ldots, L_{N_2})\Phi_2, \ldots (L_{N_{m-1}+1}, \ldots, L_{N_m})\Phi_m)\Phi.$$

This operation, which we denote by Φ', is uniform.

Proof. Fix a 1-uniform homomorphism $\varphi\colon \Sigma^* \to \Gamma^*$ and suppose $L_j = K_j\varphi^{-1}$ for $1 \leq j \leq N_m$. By Proposition 3.4.5, it suffices to show that $(L_1, \ldots, L_{N_m})\Phi' = (K_1, \ldots, K_{N_m})\Phi'\varphi^{-1}$. Let $N_0 = 0$; then by the uniformity of Φ_j, for $1 \leq j \leq m$ we have

$$(L_{N_{j-1}+1}, \ldots, L_{N_j})\Phi_j = (K_{N_{j-1}+1}, \ldots, K_{N_j})\Phi_j\varphi^{-1}.$$

Set $M_j = (L_{N_{j-1}+1}, \ldots, L_{N_j})\Phi_j$ and $M_j' = (K_{N_{j-1}+1}, \ldots, K_{N_j})\Phi_j$. Then $M_j = M_j'\varphi^{-1}$ for $1 \leq j \leq m$. By the uniformity of Φ, we have

$$(L_1, \ldots, L_{N_m})\Phi' = (M_1, \ldots, M_m)\Phi = (M_1', \ldots, M_m')\Phi\varphi^{-1} = (K_1, \ldots, K_{N_m})\Phi'\varphi^{-1},$$

as required. □

This shows that all "combined operations" formed by compositions of the uniform operations we have seen so far are also uniform.

The following "substitution lemma" can also be used to construct new uniform operations from known ones.

Lemma 3.5.8. *Let Φ be a k-ary operation. Fix $m \geq 1$ and $i_1, \dots, i_k \in \{1, \dots, m\}$. Then the operation Φ' defined by $(L_1, \dots, L_m) \mapsto (L_{i_1}, \dots, L_{i_k})\Phi$ is uniform.*

Proof. Fix a 1-uniform homomorphism $\varphi \colon \Sigma^* \to \Gamma^*$ and suppose $L_j = K_j\varphi^{-1}$ for $1 \leq j \leq m$. Then by the uniformity of Φ, we have

$$(L_1, \dots, L_m)\Phi' = (L_{i_1}, \dots, L_{i_k})\Phi = (K_{i_1}, \dots, K_{i_k})\Phi\varphi^{-1} = (K_1, \dots, K_m)\Phi'\varphi^{-1}.$$

Therefore, by Proposition 3.4.5, the operation Φ' is uniform. □

As an example, we show that the power operation is uniform. Define $L^0 = \{\varepsilon\}$ and for $n \geq 1$, set $L^n = L^{n-1}L$.

Proposition 3.5.9. *For $n \geq 0$, the power operation $L \mapsto L^n$ is uniform.*

Proof. For $n \geq 2$, in Lemma 3.5.8, let Φ be the n-ary concatenation operation, set $m = 1$ and set $i_1, \dots, i_n = 1$. For $n = 1$, it is immediate that $L \mapsto L$ is uniform. For $n = 0$, see Proposition 3.5.10 below. □

Another example is the anti-concatenation operation $(L, L') \mapsto L'L$. This belongs to the class of shuffles on trajectories, so we already know that it is uniform, but an alternate proof could be given using Lemma 3.5.8: let Φ be binary concatenation, set $m = 2$, set $i_1 = 2$ and set $i_2 = 1$.

Next we consider some operations which depend only on the alphabet of the input languages. These are not interesting from a state complexity perspective, but can be used to construct interesting combined operations.

Proposition 3.5.10. *Let $S \subseteq \mathbb{N}$. The operation $(L_1, \dots, L_m) \mapsto \bigcup_{n \in S} \Sigma^n$, where Σ is the common alphabet of the inputs, is uniform. In particular, the following operations are uniform for all arities m:*

1. $(L_1, \dots, L_m) \mapsto \emptyset$.
2. $(L_1, \dots, L_m) \mapsto \{\varepsilon\}$.

3. $(L_1, \ldots, L_m) \mapsto \Sigma^*$.

4. $(L_1, \ldots, L_m) \mapsto \Sigma^+$.

Proof. Fix a 1-uniform homomorphism $\varphi\colon \Sigma^* \to \Gamma^*$ and suppose $L_j = K_j\varphi^{-1}$ for $1 \leq j \leq m$. We claim that $\Sigma^n = \Gamma^n\varphi^{-1}$ for all $n \geq 0$. Indeed, take a word $w \in \Sigma^n$; then $w\varphi$ is in Γ^n by 1-uniformity, and so $w \in \Gamma^n\varphi^{-1}$. Conversely, if $w \in \Gamma^n\varphi^{-1} = \{x \in \Sigma^* : x\varphi \in \Gamma^n\}$ then certainly $w \in \Sigma^n$. It follows that:

$$\bigcup_{n \in S} \Sigma^n = \bigcup_{n \in S} \Gamma^n\varphi^{-1} = \left(\bigcup_{n \in S} \Gamma^n\right)\varphi^{-1}.$$

By Proposition 3.4.5, operations of this type are uniform. □

For a language L over Σ, the *right ideal* generated by L is Σ^*L, the *left ideal* generated by L is $L\Sigma^*$, the *two-sided ideal* generated by L is $\Sigma^*L\Sigma^*$, and the *all-sided ideal* generated by L is $L \sqcup\!\sqcup \Sigma^*$, where $\sqcup\!\sqcup$ is (ordinary) shuffle. By combining our earlier results, we can show that the operations which map L to one of the ideals it generates are uniform. For example, let Φ be ternary concatenation, let Φ_1 and Φ_3 be $L \mapsto \Sigma^*$, and let Φ_2 be $L \mapsto L$. Then the operation $(L_1, L_2, L_3) \mapsto (L_1\Phi_1, L_2\Phi_2, L_3\Phi_3)\Phi$ is uniform by closure under composition. Then by substitution, $L \mapsto (L\Phi_1, L\Phi_2, L\Phi_3)\Phi = \Sigma^*L\Sigma^*$ is uniform.

In summary, we have proved that the following operations are uniform: reversal, cyclic shift, star, power, m-ary concatenation, m-ary boolean operations (including union, intersection, difference, symmetric difference and complement), shuffles on trajectories (including shuffle, literal shuffle, balanced literal shuffle, insertion, balanced insertion, and anti-concatenation), and the "alphabet-dependent" operations of Proposition 3.5.10. We also proved that the class of uniform operations is closed under composition, meaning that *all combined operations* formed by composing the aforementioned operations are uniform, such as "star-complement-star" or "star of union". Additionally, we proved a substitution lemma that gives another way to construct new uniform operations from old, such as the "ideal generated by" operations.

3.5.2 Non-Uniform Operations

First we remark that *constant operations*, which output a fixed language regardless of the input, are not in general uniform. One issue is that our theoretical framework assumes that all regular operations are *alphabet-preserving*, so we cannot even define true "constant operations" that take arbitrary regular languages as inputs; we must restrict the inputs to have the same alphabet as the constant output language. The more fundamental problem is that constant operations need not behave uniformly with respect to transformations. For example, let Ψ be a constant DFA operation, and suppose that in DFA $\mathcal{A}$, the letter a induces transformation t, and in DFA $\mathcal{B}$, the letter b also induces transformation t. If Ψ is uniform, then the transformation induced by a in $\mathcal{A}\Psi$ will be the same as the transformation induced by b in $\mathcal{B}\Psi$. But the constant operation Ψ could produce a DFA in which a and b induce different transformations, violating uniformity. The only way to ensure uniformity is if Ψ produces a DFA in which every letter induces the same transformation; if we enforce this condition, we essentially obtain the alphabet-dependent operations of Proposition 3.5.10.

Our first example of an interesting non-uniform operation is the following:

$$\frac{1}{2}L = \{x \in \Sigma^* : xy \in L, |x| = |y|\}.$$

This "half" operation is an example of a *proportional removal*; the state complexity of proportional removals was studied by Domaratzki [29]. We could prove that this operation is not uniform directly from the definition, or using the language-theoretic characterization, but instead we will show something even stronger: the OLPA approach does not maximize the state complexity of this operation.

If the OLPA approach worked for this operation, then by Lemma 3.3.6, the state complexity of the operation would be maximized by a language of the form $\frac{1}{2}T_Q(1, F)$ for some state configuration $(Q, 1, F)$. However, it is not hard to see that if F is non-empty, then $\frac{1}{2}T_Q(1, F)$ is either T_Q^* or $T_Q^* \setminus \{\varepsilon\}$, depending on whether $i \in F$. Indeed, let w be a non-empty word in T_Q^*. We have $1w = q$ for some $q \in Q$. Let t be a transformation that sends q into F. Then $wt\,\mathrm{id}_Q^{|w|-1}$ maps 1 into F, and so this word is in the language $T_Q(1, F)$. But w is exactly half the length of this word, so $w \in \frac{1}{2}T_Q(1, F)$. This means that $\mathrm{sc}(\frac{1}{2}T_Q(1, F)) \leq 2$; but Domaratzki [29] shows that there are languages L of state complexity n such that $\mathrm{sc}(\frac{1}{2}L) = n$. A similar argument shows that OLPA approach fails for many other proportional removal operations as well, although we have not tried to characterize the proportional removals for which the approach fails.

Next we consider *deletions along trajectories* [30, 40], a class of operations which includes left quotient, right quotient, deletion, scattered deletion, bi-polar deletion, and k-deletion [30, p. 296]. We will show that the left quotient operation and the deletion operation are not uniform. We have not investigated uniformity for other deletions along trajectories.

The case of left quotient is interesting, because the OLPA approach actually works for this operation despite its non-uniformity. The left quotient of L by L' is $(L')^{-1}L = \{x \in \Sigma^* : wx \in L \text{ for some } w \in L'\}$. This operation satisfies a weak version of the language-theoretic characterization of uniformity:

For all 1*-uniform homomorphisms* $\varphi\colon \Sigma^* \to \Gamma^*$*, if* $L_j = K_j\varphi^{-1}$ *and* $L_j \neq \emptyset$ *for* $1 \leq j \leq 2$*, then* $(L_2^{-1}L_1)\Phi = (K_2^{-1}K_1)\Phi\varphi^{-1}$*.*

Because empty languages are excluded here, the OLPA approach would fail if maximizing the state complexity in certain cases required the use of empty languages. But this does not happen for left quotient.

To see that left quotient is not uniform, let $\Sigma = \{a, b\}$ and define $\varphi\colon \Sigma^* \to \Sigma^*$ by $a\varphi = b\varphi = b$. Then define $K_1 = \{ab\}$, $K_2 = \{a\}$, $L_1 = K_1\varphi^{-1}$, and $L_2 = K_2\varphi^{-1}$. If left quotient was uniform, we would have $L_2^{-1}L_1 = (K_2^{-1}K_1)\varphi^{-1}$. But $L_1 = L_2 = \emptyset$, and so $L_2^{-1}L_1 = \emptyset$. Meanwhile, $(K_2^{-1}K_1)\varphi^{-1} = (\{a\}^{-1}\{ab\})\varphi^{-1} = \{b\}\varphi^{-1} = \{a, b\}$.

The *deletion* of L' from L is $L \rightsquigarrow L' = \{xz \in \Sigma^* : xyz \in L \text{ for some } y \in L'\}$. We will show that the OLPA approach fails for this operation.

If the OLPA approach worked, the state complexity would be maximized by some pair of OLPA witnesses. Consider the language $\mathbb{T}_1(1, F_1) \rightsquigarrow \mathbb{T}_2(1, F_2)$ where $\mathbb{T} = T_{Q_1} \times T_{Q_2}$ for some finite sets Q_1 and Q_2. We claim that $\mathbb{T}_1(1, F_1) \rightsquigarrow \mathbb{T}_2(1, F_2) = \mathbb{T}^*$, which has state complexity one.

Indeed, fix a word $w \in \mathbb{T}^*$. Write $w = (t_{1,1}, t_{2,1}) \cdots (t_{1,k}, t_{2,k})$. Let $w_1 = t_{1,1} \cdots t_{1,k}$, set $q_1 = 1w_1$, and choose a transformation $t_1\colon Q_1 \to Q_1$ that sends q_1 into F_1. Next, choose a transformation $t_2\colon Q_2 \to Q_2$ that sends 1 into F_2. Then $1w_1t_1 \in F_1$, so $w(t_1, t_2) \in \mathbb{T}_1(1, F_1)$. However, $1t_2 \in F_2$, so $(t_1, t_2) \in \mathbb{T}_2(1, F_2)$. It follows $w \in \mathbb{T}_1(1, F_1) \rightsquigarrow \mathbb{T}_2(1, F_2)$ since it can be obtained by deleting a word in $\mathbb{T}_2(1, F_2)$ from a word in $\mathbb{T}_1(1, F_1)$.

This shows that using OLPA witnesses for deletion only produces languages of state complexity one. However, Han, Ko and Salomaa [40] proved that if L has state complexity n, then $n2^{n-1}$ is a tight upper bound on the state complexity of $L \rightsquigarrow L'$. Hence the state complexity of deletion is not maximized by the OLPA approach.

It is interesting that our main examples of operations for which the OLPA approach fails involve the idea of "deletion" in some sense.

Chapter 4

Techniques for Boolean Operations

4.1 Introduction

In this chapter, we take a deep dive into some of the principles behind maximizing the state complexity of boolean operations on regular languages. We focus on the special case of regular languages whose syntactic monoid is a group, or equivalently, DFAs in which all transformations in the transition monoid are permutations. This case is simpler to deal with than the general case, and allows us to use group-theoretic arguments to obtain results that turn out to be applicable even in a more general setting. We also focus exclusively on binary boolean operations, as this is the case most commonly studied in state complexity research, and the relevant problems are already rich and challenging enough without generalizing to higher arity.

The main inspiration for the work in this chapter is a paper of Bell, Brzozowski, Moreira, and Reis [2], which considers the following question: for which pairs of languages (L_m, L'_n) (with state complexities m and n respectively) does $L_m \circ L'_n$ reach the maximal state complexity mn for *every* proper binary boolean operation $\circ$? ("Proper" means that the operation is not constant or a function of only one argument; in these cases the worst-case state complexity is lower.) Bell et al. give sufficient conditions for this to occur. The conditions are based on the transition monoids of the minimal DFAs of L_m and L'_n; essentially, if the transition monoids contain the symmetric groups S_m and S_n, then "usually" (i.e., excluding a known class of counterexamples) the language $L_m \circ L'_n$ will have state complexity mn. The main result of this chapter is a refinement of these sufficient conditions. We prove that if the transition monoids contain 2-transitive groups, then "usually" $L_m \circ L'_n$ has state complexity mn (though our notion of "usually" is more restrictive than that of Bell et al.). We obtain an number of other results, including necessary and sufficient conditions for $L_m \circ L'_n$ to have state complexity mn in the special case where the minimal automata for L_m and L'_n have exactly one final state and their transition monoids contain a transitive permutation group.

To obtain these results, we exploit a connection between a class of permutation groups called *primitive groups*, previously discussed in Section 2.3.2, and the notion of *uniformly minimal* automata introduced by Restivo and Vaglica [70]. A minimal DFA is *uniformly minimal* if it remains minimal whenever the final state set is replaced by a non-empty proper subset of the state set. For a DFA whose transition monoid is a permutation group, uniform minimality is equivalent to primitivity of the transition monoid. Although uniform minimality played an important role in the paper of Bell et al., this connection with primitive groups was not used in their paper.

This chapter is structured as follows. In Section 4.2 we explore the aforementioned relationship between primitive groups and uniform minimality. In Section 4.3 we introduce the notion of *uniform boolean minimality*, a weakening of uniform minimality that is more relevant to boolean operations. In Section 4.4 we investigate conditions for boolean operation automata to be accessible, that is, have all states reachable. In Section 4.5 we consider distinguishability of states in boolean operation automata. In Section 4.6 we define a notion of *similarity* for DFAs, and prove our main result for pairs of dissimilar DFAs. In Section 4.7 we briefly look at the case of similar DFAs, where our main result does not hold. In Section 4.8 we show how to extend these results to DFAs whose transition monoid contains non-permutations. In Section 4.9 we construct a counterexample to show that a certain hypothesis in our main result is necessary. Finally, in Section 4.10 we summarize all the results proved in this chapter.

We suggest the reader be familiar with the definitions and examples of transitive and primitive groups given in Section 2.3.2 before proceeding further with this chapter. It may also help to review the direct product construction for binary boolean operations given in Section 2.4.4.

4.2 Primitive Groups and Uniform Minimality

A DFA $\mathcal{A} = (Q, \Sigma, T, 1, F)$ is called a *permutation DFA* if the transition monoid of $\mathcal{A}$ is a permutation group on Q. In this case we use the term *transition group* rather than transition monoid. The languages recognized by permutation DFAs are called *group languages*.

Figure 4.2.1 shows a permutation DFA, on the left, and a non-permutation DFA (that

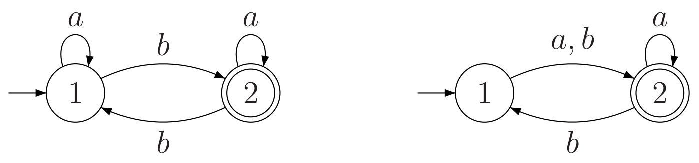


Figure 4.2.1: Example of a permutation DFA (left) and a non-permutation DFA (right).

is, a DFA that is not a permutation DFA) on the right. In the left DFA, the action of a is the identity permutation id and the action of b is the 2-cycle $(1, 2)$. Thus the transition group is the permutation group generated by id and $(1, 2)$. In the right DFA, the action of a is not a permutation: it maps 1 to 2 and 2 to itself, so nothing is mapped to 1. Note that the action of b is still $(1, 2)$ in the right DFA; the transition monoid of the right DFA *contains* permutations, but it is not a permutation group since it also contains non-permutations.

Recall that an automaton (Q, Σ, T, I, F) is *accessible* if every state in Q is reachable from some state of I. For a DFA $(Q, \Sigma, T, 1, F)$, this is equivalent to every state being reachable from 1. We say an automaton is *strongly connected* if for each ordered pair of states (p, q), state q is reachable from state p. This is equivalent to the automaton being strongly connected when viewed as a directed graph.

Proposition 4.2.1. *For a permutation DFA $\mathcal{A}$ with transition group G, the following are equivalent:*

1. *$\mathcal{A}$ is accessible.*
2. *$\mathcal{A}$ is strongly connected.*
3. *G is transitive.*

Proof. (1) $\implies$ (3): Since $\mathcal{A}$ is accessible, for each $q \in Q$ there exists $w_q \in \Sigma^*$ such that $1w_q = q$. Since G is a group, each element w_q has an inverse. Furthermore, since G is a permutation group, the identity element is the identity permutation. Thus $q(w_q)^{-1}w_q = q$, and it follows that we must have $q(w_q)^{-1} = 1$ for all $q \in Q$. Thus for all $p, q \in Q$ we have $p(w_p)^{-1}w_q = 1w_q = q$. It follows G is transitive.

(3) $\implies$ (2): Since G is transitive, for all $p, q \in Q$ there exists $w \in \Sigma^*$ such that $pw = q$. This is precisely saying that $\mathcal{A}$ is strongly connected.

The last implication (2) $\implies$ (1) is immediate. $\square$

Note that (2) $\iff$ (3) holds for arbitrary DFAs, not only permutation DFAs.

Let $\mathcal{A} = (Q, \Sigma, T, 1, F)$ be a DFA and let $L = L(\mathcal{A})$ be its language. For $S \subseteq Q$, we write $\mathcal{A}(S)$ for the DFA $\mathcal{A} = (Q, \Sigma, T, 1, S)$ obtained by replacing the final state set of $\mathcal{A}$ with S. We say a regular language L' is a *cognate* of L if $L' = L(\mathcal{A}(S))$ for some $S \subseteq Q$. We say a DFA $\mathcal{A}'$ is a *cognate* of $\mathcal{A}$ if $\mathcal{A}' = \mathcal{A}(S)$ for some S; so a language is a cognate of L if and only if it is recognized by a cognate of $\mathcal{A}$. If $S = Q$ or $S = \emptyset$, then $\mathcal{A}(S)$ is called a *trivial* cognate of $\mathcal{A}$, since $L(\mathcal{A}(S))$ is either Σ^* or the empty language $\emptyset$.

We say $\mathcal{A}$ is *uniformly minimal* if all non-trivial cognates of $\mathcal{A}$ are minimal. That is, we can reassign the final state set of the DFA in any non-trivial way and the new DFA will always be minimal. Equivalently, all cognates of $L = L(\mathcal{A})$ have the same state complexity $|Q|$. This definition is essentially restricted to accessible DFAs, since if $\mathcal{A}$ is not accessible, then not all states are reachable and hence no cognate of $\mathcal{A}$ can be minimal.

Restivo and Vaglica introduced and studied uniformly minimal DFAs in [71]. Their notion of uniform minimality is almost the same as ours, except it is restricted to *strongly connected* DFAs. Presumably, Restivo and Vaglica were interested in DFAs that are minimal for every reassignment of *initial and final* states; if a DFA is not strongly connected, we can reassign the initial state to obtain a new DFA which is not accessible and hence not minimal. However, for strongly connected DFAs, the choice of initial state has no effect on minimality since every state is reachable from each possible choice of initial state. Hence we lose nothing by fixing an initial state and generalizing to accessible DFAs.

Remark. Restivo and Vaglica also studied uniformly minimal DFAs in [70], but they used different terminology. They used the term "almost uniformly minimal" for the notion discussed above, and used "uniformly minimal" for a stronger condition that can only be met by "incomplete" DFAs (which we do not discuss in this thesis).

Recall that if M is a transformation monoid on X, an *M-congruence* is an equivalence relation on X that is invariant under the natural action of M on X. That is, if x and x' are equivalent, then xm and $x'm$ are equivalent for all $m \in M$. An M-congruence is *trivial* if it is either the *equality congruence*, in which elements are equivalent if and only if they are equal, or it is the *full congruence*, in which all elements are equivalent.

Proposition 4.2.2. *Let $\mathcal{A}$ be a DFA with transition monoid M. The indistinguishability relation Δ_S, under which two states are equivalent if and only if they are indistinguishable under $S \subseteq Q$, is an M-congruence.*

Proof. Let p and q be states of $\mathcal{A}$ that are equivalent under Δ_S. Then by definition, for all $w \in \Sigma^*$, we have $pw \in S \iff qw \in S$. It follows that for a fixed $x \in \Sigma^*$ and for all

$w \in \Sigma^*$, we have $pxw \in S \iff qxw \in S$. This shows that if p and q are equivalent under Δ_S, then for all $x \in \Sigma^*$, the states px and qx are equivalent under Δ_S. In other words, Δ_S is an M-congruence. □

A DFA $\mathcal{A}$ with transition monoid M is called *simple* if all M-congruences are trivial. Ésik proved the following result for strongly connected DFAs (in a private communication to Restivo and Vaglica; the result appeared as Proposition 1 in [71]). The same proof works for accessible DFAs.

Proposition 4.2.3. *An accessible DFA $\mathcal{A}$ is uniformly minimal if and only if it is simple.*

Proof. Let M be the transition monoid of $\mathcal{A}$. Suppose $\mathcal{A}$ is simple, that is, all M-congruences are trivial. Then in particular, for every $S \subseteq Q$, the indistiguishability relation Δ_S is trivial. If Δ_S is the equality relation, then each state lies in its own class, so all pairs of states are distinguishable. Since $\mathcal{A}$ is accessible, all states are reachable, and hence $\mathcal{A}$ is minimal. If Δ_S is the full relation, then all states are indistinguishable. But final states are always distinguishable from non-final states, so this can only happen if all states are final ($S = Q$) or all states are non-final ($S = \emptyset$). Hence if $\emptyset \subsetneq S \subsetneq Q$, then $\mathcal{A}(S)$ is minimal, so it follows that $\mathcal{A}$ is uniformly minimal.

Conversely, suppose $\mathcal{A}$ is not simple, and there exists a non-trivial M-congruence. Then this congruence has a class E which has at least two elements, but is not all of Q. Let E be the final state set of $\mathcal{A}$ and let $p, q \in E$. For all $w \in \Sigma^*$, the states pw and qw both lie in the set Ew, which is contained in some congruence class E'. If $E' = E$, then we have $pw, qw \in E$. If $E' \cap E = \emptyset$, then we have $pw, qw \notin E$. Thus for all $w \in \Sigma^*$, we have $pw \in E \iff qw \in E$, and so p and q are not distinguishable under E. Hence $\mathcal{A}$ is not uniformly minimal, since $\mathcal{A}(E)$ is not minimal. □

In the special case of permutation DFAs, we have:

Corollary 4.2.4. *An accessible permutation DFA $\mathcal{A}$ is uniformly minimal if and only if its transition group G is primitive.*

Proof. By Proposition 4.2.3, if $\mathcal{A}$ is uniformly minimal, then it is simple, and so all G-congruences are trivial. Now, by Proposition 2.3.4, a group G is primitive if and only if all G-congruences are trivial. Hence G is primitive.

Conversely, if G is primitive, then all G-congruences are trivial. Hence $\mathcal{A}$ is simple and hence uniformly minimal by Proposition 4.2.3. □

Note that both implications in Corollary 4.2.4 are vacuously true if $\mathcal{A}$ is not accessible: $\mathcal{A}$ cannot be uniformly minimal, and G cannot be transitive and thus cannot be primitive. Thus one can technically omit the accessible assumption.

The wealth of results on primitive groups makes Corollary 4.2.4 quite useful for studying and constructing uniformly minimal DFAs. For example, we can use this corollary to easily prove that for each $n \geq 2$, there exists a uniformly minimal DFA with n states. Restivo and Vaglica proved this using a rather complicated construction [70, Theorem 3].

Proposition 4.2.5. *For each $n \geq 2$, there exists a uniformly minimal permutation DFA with n states.*

Proof. The symmetric group S_n is primitive for all $n \geq 2$, and clearly for each $n \geq 2$ there exists an n-state DFA with transition group S_n. For example, let $\{g_1, \dots, g_k\}$ be a generating set of the symmetric group; such a generating set exists since S_n is a finite group (for example, we could just take the set of all elements of S_n). Once we have a generating set, let $\mathcal{A}$ be a DFA with states $\{1, \dots, n\}$, alphabet $\Sigma = \{a_1, \dots, a_k\}$, and letter actions $a_i = g_i$ for $1 \leq i \leq k$.

In fact, we can use a binary alphabet, since S_n has generating sets of size two for all $n \geq 2$. For example, the set $\{(1, \dots, n), (1, 2)\}$ generates S_n. □

This proof illustrates a technique that we frequently use for producing examples of DFAs. If we have a generating set for a transformation monoid, we can construct a DFA which has that monoid as its transition monoid.

Example 4.2.6. Let $\mathcal{A}$ be the DFA with alphabet $\{a, b\}$ defined as follows.

- The states are $\{1, 2, 3, 4\}$, the initial state is 1, and the final states are $\{3, 4\}$.
- The letter actions are the permutations $a = (2, 3, 4)$ and $b = (1, 2)(3, 4)$.

The permutations $(1, 2)(3, 4)$ and $(2, 3, 4)$ generate the alternating group A_4; this can be verified computationally. Thus the transition group of $\mathcal{A}$ is A_4. We saw in Example 2.3.9 that A_4 is transitive and primitive. Hence by Proposition 4.2.1, $\mathcal{A}$ is strongly connected, and by Corollary 4.2.4, $\mathcal{A}$ is uniformly minimal.

A state diagram of $\mathcal{A}$ is given in Figure 4.2.2. It is tedious, but possible to verify that $\mathcal{A}$ is uniformly minimal by checking that it is minimal with respect to every non-empty, proper subset of $\{1, 2, 3, 4\}$. ■

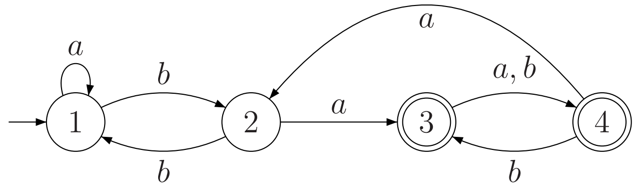


Figure 4.2.2: Uniformly minimal DFA $\mathcal{A}$ of Example 4.2.6.

Example 4.2.7. Let $\mathcal{A}$ be the DFA with alphabet $\{a\}$, states $\{1, \dots, 6\}$, initial state 1, final states $F = \{1, 3, 5\}$ and $a = (1, 2, 3, 4, 5, 6)$. A diagram is in Figure 4.2.3.

The transition group G of $\mathcal{A}$ is the cyclic group of order six, which is imprimitive. We saw in Example 2.3.6 that $F = \{1, 3, 5\}$ is a block for this group. Hence for all k, we either have $Fa^k = F$ or $Fa^k \cap F = \emptyset$. Thus if $i, j \in F$, then for all k we have $ia^k \in F \iff ja^k \in F$. This means all pairs of states in F are indistinguishable under F, and hence $\mathcal{A}$ is not minimal.

This argument actually shows that whenever F is a non-trivial block of G, the DFA $\mathcal{A}$ is not minimal. In fact, this also holds whenever F is a union of non-trivial blocks of G (see Lemma 4.2.8 below).

Note that if we construct a DFA from a cyclic group of *prime* order, we get a uniformly minimal DFA, since cyclic groups of prime order are primitive. ■

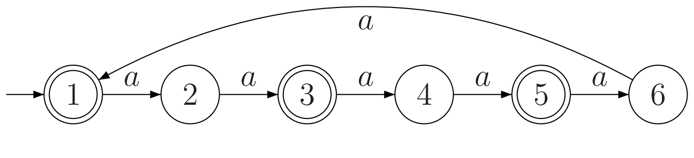


Figure 4.2.3: Non-minimal DFA $\mathcal{A}$ of Example 4.2.7 with an imprimitive transition group.

Remark. Steinberg has extended the notion of primitivity to transformation monoids [76]. Steinberg defines a transformation monoid M to be *primitive* if there are no non-trivial M-congruences. Under this definition, an accessible DFA $\mathcal{A}$ is uniformly minimal if and only if the transition monoid M is primitive. However, we have not investigated whether any of our other results that hold for primitive groups are also true for primitive monoids.

We close this section with a technical lemma that generalizes Proposition 4.2.3. If M is a transformation monoid on X and $S \subseteq X$, we say that S is *saturated* by an M-congruence if it is a union of classes of the M-congruence.

Lemma 4.2.8. *An accessible DFA $\mathcal{A}$ with $\emptyset \subsetneq F \subsetneq Q$ and transition monoid M is minimal if and only if there is no non-trivial M-congruence that saturates F.*

It follows that if all M-congruences are trivial, then $\mathcal{A}$ is uniformly minimal. Conversely, if there is a non-trivial M-congruence, then it saturates its own congruence classes and at least one class is a proper non-empty subset of Q, and thus $\mathcal{A}$ is not uniformly minimal. Hence this indeed generalizes Proposition 4.2.3.

Proof. We prove the contrapositive: $\mathcal{A}$ is not minimal if and only if there exists a non-trivial M-congruence that saturates F.

Suppose $\mathcal{A}$ is not minimal. Then the indistinguishability relation Δ of F is a non-trivial M-congruence, since at least two states are indistinguishable. Suppose there is an indistinguishability equivalence class E that is neither contained in F nor disjoint from F. Then there exist $p, q \in E$ such that $p \in F$ and $q \notin F$. But then p and q are distinguishable under F, which cannot happen since E is an indistinguishability class. Thus for each indistinguishability class $q\Delta$, we have $q\Delta \subseteq F$ or $q\Delta \cap F = \emptyset$. Then we have $F = \bigcup_{f \in F} f\Delta$, so F is saturated by its indistinguishability relation.

Conversely, let $E_1, \dots, E_k \subseteq Q$ be the congruence classes of a non-trivial M-congruence that saturates F. Choose a congruence class E_i of size at least two. Then for all $w \in \Sigma^*$ we have $E_i w \subseteq E_j$ for some j. Since F is a union of congruence classes, either $E_j \subseteq F$ or $E_j \cap F = \emptyset$. Hence for $p, q \in E_i$ and all $w \in \Sigma^*$, we have $pw \in F \iff E_j \subseteq F \iff E_i w \subseteq F \iff qw \in F$. It follows that states in E_i are indistinguishable, and thus $\mathcal{A}$ is not minimal. $\square$

In the special case of permutation DFAs, this has a nice consequence.

Corollary 4.2.9. *Let $\mathcal{A}$ be a permutation DFA with transition group G. If $|F| = 1$ or $|F| = |Q| - 1$, then $\mathcal{A}$ is minimal if and only if it is accessible.*

Proof. Recall that if G is a transitive permutation group and E and E' are classes of a G-congruence, then $|E| = |E'|$. It follows that if $|F| = 1$, then a non-trivial G-congruence cannot saturate F since all the congruence classes have size at least two. Furthermore, a G-congruence saturates F if and only if it saturates $Q \setminus F$, and if $|F| = |Q| - 1$ then a non-trivial G-congruence cannot saturate the set $Q \setminus F$ of size one. Hence if $\mathcal{A}$ is accessible, it is minimal by Lemma 4.2.8. On the other hand, if $\mathcal{A}$ is not accessible, it cannot be minimal. $\square$

4.3 Uniform Boolean Minimality

Throughout the remainder of this chapter, $\mathcal{A}' = (Q', \Sigma', T', 1, F')$ and $\mathcal{A} = (Q, \Sigma, T, 1, F)$ are minimal DFAs. We assume these DFAs have a common alphabet $\Sigma = \Sigma'$. The languages of $\mathcal{A}'$ and $\mathcal{A}$ are L' and L, and the transition monoids are M' and M, respectively. For $w \in \Sigma^*$, we abbreviate T'_w to w' and T_w to w. Often we will assume $\mathcal{A}'$ and $\mathcal{A}$ are permutation DFAs, and then we will use G' and G for the transition groups rather than M' and M. For the direct product $\mathcal{A}' \times \mathcal{A}$ (as defined in Section 2.4.4), we write $M_\times$ for its transition monoid, or $G_\times$ if $\mathcal{A}'$ and $\mathcal{A}$ are both permutation DFAs.

We recall the definition of binary boolean operations on languages. Given a binary boolean function $\circ\colon \{0,1\}^2 \to \{0,1\}$, the corresponding binary boolean operation on languages is defined by:

$$L' \circ L = \{w \in \Sigma^* : w\chi_{L'} \circ w\chi_L = 1\}.$$

Here $\chi_S\colon S \to \{0,1\}$ denotes the characteristic function of the set S, defined by $x\chi_S = 1$ if $x \in S$ and $x\chi_S = 0$ otherwise. We can also define binary boolean operations on sets of DFA states: for $S' \subseteq Q'$ and $S \subseteq Q$, we have $S' \circ S = \{(q', q) \in Q' \times Q : q'\chi_{S'} \circ q\chi_S\}$. The language $L' \circ L$ is recognized by the direct product DFA $\mathcal{A}' \times \mathcal{A}$ when equipped with the final state set $S' \circ S$.

Recall also that a binary boolean function (and the associated binary boolean operation on languages) is *proper* if its output depends on both of its arguments.

Definition 4.3.1. A pair of languages (L', L) has *maximal boolean complexity* if $\mathrm{sc}(L' \circ L) = \mathrm{sc}(L')\,\mathrm{sc}(L)$ for all proper binary boolean operations $\circ$; in other words, the pair (L', L) is a witness for every proper binary boolean operation simultaneously.

Our goal in this chapter is to describe techniques for proving pairs of languages have maximal boolean complexity. We can reformulate the definition of maximal boolean complexity in terms of automata by introducing the following notion:

Definition 4.3.2. Suppose that $\emptyset \subsetneq S' \subsetneq Q'$ and $\emptyset \subsetneq S \subsetneq Q$. We say a subset of $Q' \times Q$ is (S', S)-*compatible* if it is equal to $S' \circ S$ for some proper binary boolean operation $\circ$.

Notice now that if L' is recognized by $\mathcal{A}'$ and L is recognized by $\mathcal{A}$, then (L', L) has maximal boolean complexity if and only if $\mathcal{A}' \times \mathcal{A}$ is minimal for every (F', F)-compatible subset of $Q' \times Q$.

To attack the problem of proving languages have maximal boolean complexity, we study the following stronger notion.

Definition 4.3.3. The pair of DFAs $(\mathcal{A}', \mathcal{A})$ (or the direct product $\mathcal{A}' \times \mathcal{A}$) is *uniformly boolean minimal* if for every pair of sets (S', S) with $\emptyset \subsetneq S' \subsetneq Q'$ and $\emptyset \subsetneq S \subsetneq Q$ and every (S', S)-compatible set $S' \circ S$, the DFA $(\mathcal{A}' \times \mathcal{A})(S' \circ S)$ is minimal.

Let us compare the definitions of maximal boolean complexity and uniform boolean minimality. The pair (L, L') has maximal boolean complexity if $\mathcal{A}' \times \mathcal{A}$ is minimal for every (F', F)-compatible subset, where F' and F are fixed final state sets, chosen so that $\mathcal{A}'$ recognizes L' and $\mathcal{A}$ recognizes L. But what if this property holds for *every possible choice of F' and F*, excluding the trivial cases where F' or F is empty or the full state set? If this happens, $\mathcal{A}' \times \mathcal{A}$ is uniformly boolean minimal.

We give an example of a pair of DFAs that are not uniformly boolean minimal, as well as a pair of DFAs that are.

Example 4.3.4. Define two DFAs over alphabet $\Sigma = \{a, b, c\}$ as follows:

- $\mathcal{A}'$ has state set $Q' = \{1, 2\}$, initial state 1, final state set $F' = \{1\}$, and actions $a' = c' = \text{id}$, $b' = (1, 2)$.
- $\mathcal{A}$ has state set $Q = \{1, 2\}$, initial state 1, final state set $F = \{1\}$, and actions $a = b = \text{id}$, $c = (1, 2)$.

We show that $(L(\mathcal{A}'), L(\mathcal{A}))$ does not have maximal boolean complexity, and thus $\mathcal{A} \times \mathcal{A}$ is not uniformly boolean minimal.

To see this, consider the symmetric difference operator $\oplus$, arising from the "exclusive or" boolean function: for $u, v \in \{0, 1\}$, the "exclusive or" $u \oplus v$ is zero if $u = v$ and one if $u \neq v$. The corresponding operation on languages over Σ is

$$L' \oplus L = \{w \in \Sigma^* : w \in L' \text{ or } w \in L, \text{ but not both}\} = (L' \setminus L) \cup (L \setminus L').$$

The final state set that makes $\mathcal{A}' \times \mathcal{A}$ recognize $L(\mathcal{A}') \oplus L(\mathcal{A})$ is:

$$F' \oplus F = \{(q', q) \in Q' \times Q : q' \in F' \text{ or } q \in F, \text{ but not both}\} = \{(1, 2), (2, 1)\}.$$

State diagrams of the DFAs $\mathcal{A}'$, $\mathcal{A}$ and $(\mathcal{A}' \times \mathcal{A})(F' \oplus F)$ are shown in Figure 4.3.1. Notice that $(\mathcal{A}' \times \mathcal{A})(F' \oplus F)$ is not minimal: the states $(1, 2)$ and $(2, 1)$ cannot be distinguished. Since $F' \oplus F$ is an (F', F)-compatible set, it follows that $(L(\mathcal{A}'), L(\mathcal{A}))$ does not have maximal boolean complexity, and that $\mathcal{A}' \times \mathcal{A}$ is not uniformly boolean minimal. ■

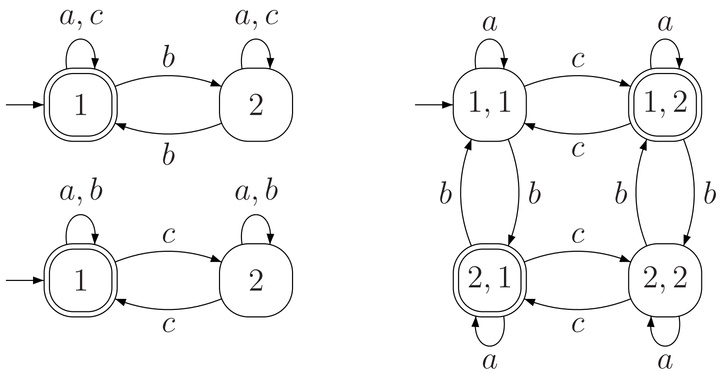


Figure 4.3.1: DFAs $\mathcal{A}'$, $\mathcal{A}$ and $\mathcal{A}' \times \mathcal{A}$ of Example 4.3.4. The final state set of $\mathcal{A}' \times \mathcal{A}$ is chosen so that $\mathcal{A}' \times \mathcal{A}$ recognizes the symmetric difference of the languages of $\mathcal{A}'$ and $\mathcal{A}$.

Example 4.3.5. Define two DFAs over alphabet $\Sigma = \{a, b\}$ as follows:

- $\mathcal{A}'$ has state set $Q' = \{1, 2\}$ and actions $a' = (1, 2)$, $b' = \text{id}$.
- $\mathcal{A}$ has state set $Q = \{1, 2, 3\}$ and actions $a = (1, 2)$, $b = (1, 2, 3)$.

The initial and final states are not important for this example.

The direct product $\mathcal{A}' \times \mathcal{A}$ is shown in Figure 4.3.2. Notice that the transition group of $\mathcal{A}$ is S_3. We will see much later (Theorem 4.6.6) that this implies $\mathcal{A}' \times \mathcal{A}$ is uniformly boolean minimal.

Note that $\mathcal{A}' \times \mathcal{A}$ is not uniformly minimal; for example, it is not minimal with respect to the final state set $\{(1, 1), (1, 2), (1, 3)\}$. If $|Q'|, |Q| \geq 2$, a direct product DFA with state set $Q' \times Q$ can never be uniformly minimal; in particular, it cannot be minimal for final state sets of the form $S' \times Q$ ("unions of rows") or $Q' \times S$ ("unions of columns"). However, the definition of uniform boolean minimality excludes these sets. ■

Bell, Brzozowski, Moreira and Reis found sufficient conditions for a pair of DFAs to be uniformly boolean minimal [2]. However, these conditions require that the transition monoids of the DFAs *contain the symmetric group*, in the sense that they contain every permutation of the DFA's state set. In particular, for permutation DFAs, these conditions only apply when the transition group *is* the symmetric group on the state set. We obtain more general sufficient conditions for uniform boolean minimality in permutation DFAs, which apply to a larger class of transition groups. Additionally, we show that DFAs whose

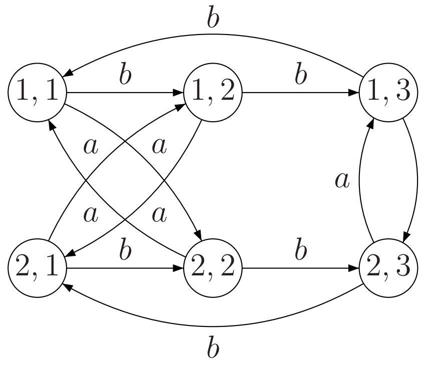


Figure 4.3.2: Uniformly boolean minimal DFA $\mathcal{A}' \times \mathcal{A}$ of Example 4.3.5.

transition monoids contain 2-transitive groups "usually" meet these conditions, up to some technical assumptions we will state later.

We also obtain necessary and sufficient conditions for a pair of languages (L', L) to have maximal boolean complexity in the special case where L' and L are recognized by permutation DFAs $\mathcal{A}'$ and $\mathcal{A}$ with exactly one final state. In this special case, it turns out (L', L) has maximal boolean complexity if and only if $\mathcal{A}' \times \mathcal{A}$ is accessible. We give several group-theoretic conditions that are equivalent to $\mathcal{A}' \times \mathcal{A}$ being accessible.

We begin with a proposition which characterizes (F', F)-compatible subsets. If Q is the state set of a DFA and $S \subseteq Q$, write $\overline{S}$ for $Q \setminus S$. Similarly, if L is a language over Σ, write $\overline{L}$ for $\Sigma^* \setminus L$.

Proposition 4.3.6. *Let $\emptyset \subsetneq F' \subsetneq Q'$ and $\emptyset \subsetneq F \subsetneq Q$. A subset of $Q' \times Q$ is (F', F)-compatible if and only if it is equal to one of the following sets:*

(a) $F' \times F$ (corresponding to $L' \cap L$).

(b) $F' \times \overline{F}$ (corresponding to $L' \cap \overline{L} = L' \setminus L$).

(c) $\overline{F'} \times F$ (corresponding to $\overline{L'} \cap L = L \setminus L'$).

(d) $\overline{F'} \times \overline{F}$ (corresponding to $\overline{L'} \cap \overline{L} = \overline{L' \cup L}$).

(e) $(F' \times \overline{F}) \cup (\overline{F'} \times F)$ (corresponding to symmetric difference $(L' \setminus L) \cup (L \setminus L')$).

(f) The complement $(Q' \times Q) \setminus S$, where S is one of the above sets.

Proof. Let $\circ$ be a proper binary boolean function. Let k be the number of pairs $(u, v) \in \{0,1\} \times \{0,1\}$ such that $u \circ v = 1$.

Case 1 ($k = 1$): If $k = 1$, then there is a unique pair (u, v) such that $u \circ v = 1$. Hence $F' \circ F = \{(q', q) : q'\chi_{F'} = u \text{ and } q\chi_F = v\}$. Consider possible values for (u, v):

- If $(u, v) = (0, 0)$ then $F' \circ F = \overline{F'} \times \overline{F}$.
- If $(u, v) = (0, 1)$ then $F' \circ F = \overline{F'} \times F$.
- If $(u, v) = (1, 0)$ then $F' \circ F = F' \times \overline{F}$.
- If $(u, v) = (1, 1)$ then $F' \circ F = F' \times F$.

Hence $F' \circ F$ is a set of type (a), (b), (c) or (d).

Case 2 ($k = 2$): If $k = 2$, there are exactly two pairs (u, v) and (u', v') such that $u \circ v = u' \circ v' = 1$. We claim that $u \neq u'$ and $v \neq v'$. To see this, suppose $u = u'$. Then we must have $v \neq v'$, or else the pairs are not distinct. Thus $\{v, v'\} = \{0, 1\}$ and it follows $u \circ 0 = u \circ 1 = 1$. Hence $\circ$ only depends on the value of the first argument, which contradicts the fact that $\circ$ is proper. By a symmetric argument, we cannot have $v = v'$. Now, observe that (q, q') is in $F' \circ F$ if and only if

$$q'\chi_{F'} = u \text{ and } q\chi_F = v \text{ or } q'\chi_{F'} = u' \text{ and } q\chi_F = v'.$$

Suppose $(u, v) = (1, 0)$. Then we necessarily have $(u', v') = (0, 1)$ and we get

$$F' \circ F = (F' \times \overline{F}) \cup (\overline{F'} \times F).$$

If $(u, v) = (0, 1)$, then $(u', v') = (1, 0)$ and we get the same set. If $(u, v) = (1, 1)$ or $(u, v) = (0, 0)$, then we get

$$F' \circ F = (F' \times F) \cup (\overline{F'} \times \overline{F}).$$

But this is simply the complement of the previous set. So we either get a set of type (e) or the complement of such a set, which is type (f).

Case 3 ($k = 3$): If $k = 3$, then there is a unique pair (u, v) such that $u \circ v = 0$. Hence $F' \circ F$ is the *complement* of a set of type (a), (b), (c) or (d), that is, a set of type (f).

This proves that every (F', F)-compatible set, that is, every set of the form $F' \circ F$ where $\circ$ is a proper binary boolean function, has one of the given forms (a)–(f). Conversely, if we are given sets F' and F and a set $X \subseteq Q' \times Q$ with one of the forms (a)–(f), the proof shows how to construct a proper binary boolean function $\circ$ such that $X = F' \circ F$. It follows X is (F', F)-compatible if and only if it has one of the forms (a)–(f). $\square$

4.4 Accessibility of the Direct Product

In this section, we consider the problem of determining when $\mathcal{A}' \times \mathcal{A}$ is accessible. This is essential for proving that $\mathcal{A}' \times \mathcal{A}$ is minimal for a certain final state set, and also an interesting question in its own right. By Proposition 4.2.1, if $\mathcal{A}' \times \mathcal{A}$ is a permutation DFA, then it is accessible if and only if its transition group is transitive. The following proposition describes the structure of the transition monoid of $\mathcal{A}' \times \mathcal{A}$.

Proposition 4.4.1. *Recall that $M_\times$ denotes the transition monoid of $\mathcal{A}' \times \mathcal{A}$.*

1. *$M_\times$ is isomorphic to the submonoid of $M' \times M$ generated by $\{(a', a) : a \in \Sigma\}$. We often identify $M_\times$ with this submonoid.*
2. *The projections $\pi' \colon M_\times \to M'$ and $\pi \colon M_\times \to M$ given by $(w', w)\pi' = w'$ and $(w', w)\pi = w$ are surjective.*
3. *If M' and M are groups, then $M_\times$ is a group.*

Proof. (1): Write $w_\times$ for the action of w in $M_\times$. Consider the map $\varphi \colon M_\times \to M' \times M$ given by $w_\times \mapsto (w', w)$. This map is clearly a monoid homomorphism. Furthermore, if $(x', x) = (y', y)$ then $q'x' = q'y'$ and $qx = qy$ for all $q' \in Q'$ and $q \in Q$, and thus in $M_\times$ we have $(q', q)x_\times = (q', q)y_\times$ for all $(q', q) \in Q \times Q'$. Hence $x_\times = y_\times$ whenever $x_\times\varphi = y_\times\varphi$, and it follows that φ is injective.

Since φ is injective, $(M_\times)\varphi$ is a finite monoid of the same size as $M_\times$. It follows φ is *bijective* when viewed as homomorphism between $M_\times$ and $(M_\times)\varphi$, and thus φ is an isomorphism between these monoids. Since $\{a_\times : a \in \Sigma\}$ generates $M_\times$, we see that $\{(a', a) : a \in \Sigma\}$ generates $(M_\times)\varphi$. Hence we have $M_\times \cong (M_\times)\varphi = \langle (a', a) : a \in \Sigma \rangle$ as required.

(2): Fix $w \in M$. Then for the element $(w', w) \in M_\times$ we have $(w', w)\pi = w$. Hence π' maps surjectively onto M'. Similarly, π' maps surjectively onto M'.

(3): Since $M_\times$ is a monoid, it suffices to show every element of $M_\times$ has an inverse. Recall that the identity elements of M' and M are ε' and ε respectively. For $(w', w) \in M_\times$, pick m and n such that $(w')^m = \varepsilon'$ in M' and $w^n = \varepsilon$ in M. This is possible since M' and M are finite groups. We have $((w')^{mn-1}, w^{mn-1}) \in M_\times$, and $(w', w)((w')^{mn-1}, w^{mn-1}) = ((w')^{mn}, w^{mn}) = (\varepsilon', \varepsilon)$, the identity of $M_\times$. It follows that $((w')^{mn-1}, w^{mn-1})$ is an inverse of (w', w). □

Recall that for permutation DFAs $\mathcal{A}'$ and $\mathcal{A}$, we denote the transition group of $\mathcal{A}'$ by G' and the transition group of $\mathcal{A}$ by G. In this case, by Proposition 4.4.1 (3), the transition monoid of $\mathcal{A}' \times \mathcal{A}$ is a group, and (as stated earlier) we denote it by $G_\times$. If $\mathcal{A}'$ and $\mathcal{A}$ are permutation DFAs, the transitivity of $G_\times$ is a necessary and sufficient condition for all states of $\mathcal{A}' \times \mathcal{A}$ to be reachable. However, the structure of $G_\times$ can be difficult to understand. Hence we will derive a simpler characterization of transitivity that depends only on properties of G' and G.

Suppose that $\mathcal{A}'$ and $\mathcal{A}$ are permutation DFAs. As before, let $\pi' \colon G_\times \to G'$ and $\pi \colon G_\times \to G$ denote the projections given by $(w', w)\pi' = w'$ and $(w', w)\pi = w$, respectively. Consider the subgroup $\ker \pi' \leq G_\times$. It contains all $(w', w) \in G_\times$ such that w' is the identity in G. View $Q' \times Q$ as a grid, where elements of Q' are "row indices" and elements of Q are "column indices". Then $\ker \pi'$ consists of the elements of $G_\times$ which fix all row indices. Similarly, the group $\ker \pi$ consists of the elements which fixes all column indices. Thus we make the following definitions:

Definition 4.4.2. The group $R = \ker \pi'$ is called the *full row stabilizer*. The group $C = \ker \pi$ is called the *full column stabilizer*.

Recall that kernels of homomorphisms are normal subgroups; thus both R and C are normal subgroups of $G_\times$.

Fix $q' \in Q'$ and $q \in Q$. Let $(q', *)$ denote the set $\{(q', i) \in Q' \times Q : i \in Q\}$, that is, the "$q'$-th row" of $Q' \times Q$. Similarly, let $(*, q) = \{(i', q) \in Q' \times Q : i \in Q\}$ denote the "q-th column" of $Q' \times Q$. Recall that if G is a group acting on X, then for $S \subseteq X$, the setwise stabilizer of S is the subgroup $\{g \in G : Sg = S\}$.

Definition 4.4.3. For $q' \in Q'$ and $q \in Q$, let $R_{q'} \leq G_\times$ be the setwise stabilizer of $(q', *)$ and let $C_q \leq G_\times$ be the setwise stabilizer of $(*, q)$. The subgroups $R_{q'}$ are called *single row stabilizers* and the subgroups C_q are called the *single column stabilizers*.

The group $R_{q'}$ is the subgroup of $G_\times$ consisting of all elements that fix the row index of elements in row q', and act arbitrarily on the column index. Hence if $g \in R_{q'}$ and $(q', q) \in (q', *)$, the element $(q', q)g$ will always lie in $(q', *)$. On the other hand, if $(p', p) \notin (q', *)$, we do not know whether g will fix the row index p'; it is only guaranteed to stabilize row $(q', *)$. Symmetrically, C_q consists of all elements of $G_\times$ that fix the *column index* of elements in *column* q, and act arbitrarily on the *row index*. The full row stabilizer R is the intersection of all single row stabilizers, and hence is a subgroup of each $R_{q'}$; the analogous fact holds for C.

We now give necessary and sufficient conditions for $\mathcal{A}' \times \mathcal{A}$ to be transitive in the case where $\mathcal{A}'$ and $\mathcal{A}$ are permutation DFAs.

Lemma 4.4.4. *Let $\mathcal{A}'$ and $\mathcal{A}$ be permutation DFAs. The following are equivalent:*

1. *$\mathcal{A}' \times \mathcal{A}$ is accessible.*
2. *G' and G are transitive and for all $q' \in Q'$ and $q \in Q$, the subgroups $R_{q'}\pi \leq G$ and $C_q\pi' \leq G'$ are transitive.*
3. *G is transitive and $R_{q'}\pi \leq G$ is transitive for some $q' \in Q'$, or G is transitive and $C_q\pi' \leq G'$ is transitive for some $q \in Q$.*
4. *$G_\times$ is transitive.*

Proof. Since $\mathcal{A}' \times \mathcal{A}$ is a permutation DFA, we see that (1) $\iff$ (4). Also, the implication (2) $\implies$ (3) is immediate.

(3) $\implies$ (4): Fix $(i', i), (j', j) \in Q' \times Q$. Suppose that G is transitive and some $C_q\pi'$ is transitive; the case where G' is transitive and $R_{q'}\pi$ is transitive for some $q' \in Q'$ is symmetric.

- Since G is transitive, there exists $x \in G$ such that $ix = q$. Let $k' \in Q'$ be the element such that $i'x' = k'$. Then $(i', i) \xrightarrow{x} (k', q)$.
- Since G is transitive, there exists $y \in G$ such that $qy = j$ in G.
- Since $C_q\pi' \leq G'$ is transitive on Q', there exists $z' \in C_q\pi'$ such that $k'z' = j'(y')^{-1}$. Since $(z', z) \in C_q$, we have $qz = q$. Hence $(k', q) \xrightarrow{z} (j'(y')^{-1}, q)$.

It follows that
$$(i', i) \xrightarrow{x} (k', q) \xrightarrow{z} (j'(y')^{-1}, q) \xrightarrow{y} (j', qy) = (j', j).$$
Thus $G_\times$ is transitive on $Q' \times Q$, since for all elements $(i', i), (j', j) \in Q' \times Q$, there exists an element of $G_\times$ that maps one to the other.

(4) $\implies$ (2): If $G_\times$ is transitive, then for all $q', i', j' \in Q'$ and $q, i, j \in Q$, there exist $x, y \in \Sigma^*$ such that
$$(q', i) \xrightarrow{x} (q', j) \text{ and } (i', q) \xrightarrow{y} (j', q).$$
Thus $q'x' = q'$ and $ix = j$, and also $qy = q$ and $i'y' = j'$. It follows that:

- Since $q'x = q'$, we have $(x', x) \in R_{q'}$, and since $qy = q$, we have $(y', y) \in C_q$.
- For all $i', j' \in Q'$, there exists a word $x \in R_{q'}\pi \leq G$ that maps i to j.

- For all $i, j \in Q$, there exists a word $y' \in C_q\pi' \leq G'$ that maps i' to j'.

Hence for all $q' \in Q'$ and $q \in Q$, we see that $C_q\pi'$ is transitive on Q' and $R_{q'}\pi$ is transitive on Q. Since $C_q\pi' \leq G'$ and $R_{q'}\pi \leq G$, it follows that G' and G are transitive.

This establishes a cycle of implications (2) $\implies$ (3) $\implies$ (4) $\implies$ (2). Since we also have (1) $\iff$ (4), all the statements are equivalent. □

This lemma reduces the problem of checking the transitivity of $G_\times$ to just checking the the transitivity of a row stabilizer on the column indices, or of a column stabilizer on the row indices. The following proposition reinterprets this in terms of reachability in DFAs; this version may be easier to understand and apply for readers who are not well-versed in group theory. It is not difficult to prove that accessibility of $\mathcal{A}' \times \mathcal{A}$ is equivalent to condition (1) of the following proposition without appeal to group theory; however, for illustrative purposes we will connect it with condition (3) of Lemma 4.4.4.

Proposition 4.4.5. *Let $\mathcal{A}'$ and $\mathcal{A}$ be permutation DFAs. The following statements are equivalent:*

1. *$\mathcal{A}'$ is accessible and there exists $q' \in Q'$ such that all states in $(q', *)$ are reachable, or $\mathcal{A}$ is accessible and there exists $q \in Q$ such that all states in $(*, q)$ are reachable.*

2. *G' is transitive and $R_{q'}\pi \leq G$ is transitive for some $q' \in Q'$, or G is transitive and $C_q\pi' \leq G'$ is transitive for some $q \in Q$.*

Proof. **(1)** $\implies$ **(2)**: Suppose $\mathcal{A}$ is accessible, and there exists $q \in Q$ such that all states in $(*, q)$ are reachable. By Proposition 4.2.1, since $\mathcal{A}$ is an accessible permutation DFA, G is transitive on Q.

To see that $C_q\pi' \leq G'$ is transitive on Q', fix $i', j' \in Q'$. Since all states in $(*, q)$ are reachable from the initial state $(1, 1)$ of $\mathcal{A} \times \mathcal{A}'$, there is some word $x \in G_\times$ such that $(1, 1) \xrightarrow{x} (i', q)$. Also, there is some $y \in G_\times$ such that $(1, 1) \xrightarrow{y} (j', q)$. Since $\mathcal{A}' \times \mathcal{A}$ is a permutation DFA, there is some $z \in \Sigma^*$ such that $z = x^{-1}$. Thus we have

$$(i', q) \xrightarrow{z} (1, 1) \xrightarrow{y} (j', q).$$

We see that zy maps q to itself, so $z'y' \in C_q\pi'$. Hence $C_q\pi'$ is transitive on Q'.

By a symmetric argument, if $\mathcal{A}'$ is accessible and there exists $q' \in Q'$ such that all states in $(q', *)$ are reachable, it follows that G' is transitive on Q' and $R_{q'}\pi \leq G$ is transitive on Q.

(2) $\implies$ **(1)**: Suppose G is transitive and $C_q\pi' \leq G'$ is transitive for some $q \in Q$. Since G is transitive, $\mathcal{A}$ is accessible. In particular, there exists $w \in \Sigma^*$ such that $1w = q$, and it follows that $(1,1) \xrightarrow{w} (1w', q)$ in $\mathcal{A}' \times \mathcal{A}$. Since $C_q\pi'$ is transitive on Q', for all $q' \in Q'$ there exists $x' \in C_q\pi'$ such that $(1w', q) \xrightarrow{x} (q', q)$. Hence every state in $(*, q)$ is reachable. In the case where G' and some $R_{q'}\pi$ are transitive, we can use a symmetric argument. □

We now prove one of our main results, which gives necessary and sufficient conditions for pairs of group languages recognized by DFAs with exactly one final state to have maximal boolean complexity.

Theorem 4.4.6. *Suppose $|Q'| \geq 3$ or $|Q| \geq 3$. Let $\mathcal{A}'$ and $\mathcal{A}$ be permutation DFAs with exactly one final state. Then the following are equivalent:*

1. *$\mathcal{A}' \times \mathcal{A}$ is accessible.*
2. *For all proper binary boolean operations $\circ$, the language $L' \circ L$ has maximal state complexity. That is, (L', L) has maximal boolean complexity.*
3. *There exists a proper binary boolean operation $\circ$ such that the language $L' \circ L$ has maximal state complexity.*

Normally, when one is trying to prove that $L' \circ L$ has maximal state complexity, proving that $\mathcal{A}' \times \mathcal{A}$ is accessible is just the first step in the process; the second is to show that $(\mathcal{A}' \times \mathcal{A})(F' \circ F)$ is minimal using distinguishability arguments. This theorem shows that for a special class of DFAs, the first step is actually sufficient. However, it is not always easy to prove that $\mathcal{A}' \times \mathcal{A}$ is accessible. The reachability-based condition of Proposition 4.4.5 can aid with this; it states that assuming $\mathcal{A}'$ and $\mathcal{A}$ are accessible, the direct product $\mathcal{A}' \times \mathcal{A}$ is accessible if either of the following holds.

- There exists a row $(q', *)$ such that all states in $(q', *)$ are reachable.
- There exists a column $(*, q)$ such that all states in $(*, q)$ are reachable.

This reduces the problem to just checking reachability for a single row or column.

The group-theoretic conditions of Lemma 4.4.4 may also help, but they are perhaps harder to understand. Later on (in Section 4.6) we will use these to obtain simpler group-theoretic conditions for accessibility of $\mathcal{A}' \times \mathcal{A}$. In particular, provided that $\mathcal{A}'$ and $\mathcal{A}$ are both accessible, and also satisfy an additional criterion called *dissimilarity* (which is usually easy to check), we have:

- If G' and G are transitive simple groups, then $\mathcal{A}' \times \mathcal{A}$ is accessible.
- If G' and G are primitive groups, then $\mathcal{A}' \times \mathcal{A}$ is accessible.

(Recall that a group G is *simple* if it has no non-trivial proper normal subgroups, that is, the only normal subgroups of G are G itself and the trivial group.)

Theorem 4.4.6. The only difficult implication here is **(1)** $\implies$ **(2)**. Suppose $\mathcal{A}' \times \mathcal{A}$ is accessible; we want to show that $L' \circ L$ has maximal state complexity for every proper binary boolean operation $\circ$. That is, we want to show that all pairs of states of $\mathcal{A}' \times \mathcal{A}$ are distinguishable under each (F', F)-compatible subset of $Q' \times Q$.

Note that since $\mathcal{A}' \times \mathcal{A}$ is accessible, by Lemma 4.4.4 we know that $G_\times$ is transitive and that $C_q\pi' \leq G'$ and $R_{q'}\pi \leq G$ are transitive for all $q' \in Q'$ and $q \in Q$. What this means is:

- For every pair of states (p', p) and (q', q) of $\mathcal{A}' \times \mathcal{A}$, there exists a word $w \in \Sigma^*$ such that $(p', p) \xrightarrow{w} (q', q)$. (Transitivity of $G_\times$)
- Fix a state $q \in Q$. For every pair of states $i', j' \in Q'$, there exists a word $w \in \Sigma^*$ such that $(i', q) \xrightarrow{w} (j', q)$. (Transitivity of $C_q\pi'$)
- Fix a state $q' \in Q'$. For every pair of states $i, j \in Q$, there exists a word $w \in \Sigma^*$ such that $(q', i) \xrightarrow{w} (q', j)$. (Transitivity of $R_{q'}\pi$)

We will use these facts repeatedly throughout the proof.

Let $F' = \{f'\}$ and $F = \{f\}$, so that $F' \times F = \{(f', f)\}$. Let (p', p) and (q', q) be distinct states of $\mathcal{A}' \times \mathcal{A}$ that we wish to distinguish. We will show these states are distinguishable under each type of set described in Proposition 4.3.6.

We only need to consider types (a) through (e), since sets of type (f) are just complements of sets of types (a) through (e), and two states are distinguishable under a set X if and only if they are distinguishable under the complement of X.

Case 1 (States in the same row or same column): Suppose $p = q$, that is, both states (p', p) and (q', q) are in the same column. Then we necessarily have $p' \neq q'$, since the states are distinct.

- By transitivity of $G_\times$, for all $r \in Q$ there exists $w \in \Sigma^*$ such that $(p', p) \xrightarrow{w} (f', r)$.
- Since $p = q$ and $p' \neq q'$, we have $(q', q) \xrightarrow{w} (s', r)$ for some $s' \neq f'$. (Since w' is a permutation, it must map p' and q' to different states.)

If $\boldsymbol{r \in F}$, we have $(f', r) \in F' \times F$ and $(s', r) \in \overline{F'} \times F$. Hence we can distinguish the states if the final state set is $\boldsymbol{F' \times F}$, $\boldsymbol{\overline{F'} \times F}$, or $\boldsymbol{(F' \times \overline{F}) \cup (\overline{F'} \times F)}$,

If $\boldsymbol{r \notin F}$, we have $(f', r) \in F' \times \overline{F}$ and $(s', r) \in \overline{F'} \times \overline{F}$. Hence we can distinguish the states if the final state set is $\boldsymbol{F' \times \overline{F}}$ or $\boldsymbol{\overline{F'} \times \overline{F}}$.

This covers all the possible sets of final states. If $p \neq q$ and $p' = q'$ (that is, the states are in the same row) we can use a symmetric argument.

Case 2 (States in different rows and different columns): Assume $p \neq q$ and $p' \neq q'$. We consider each possible set of final states in turn.

$\boldsymbol{F' \times F}$: Here $\mathcal{A}' \times \mathcal{A}$ has exactly one final state (f', f), so it is minimal by Corollary 4.2.9.

$\boldsymbol{F' \times \overline{F}}$: We make a few observations:

- By transitivity of $G_\times$, there exists $w \in \Sigma^*$ such that $(p', p) \xrightarrow{w} (f', f)$.
- Since $p \neq q$, $p' \neq q'$ and w is a permutation, we must have $qw \neq f$ and $q'w' \neq f'$.
- Since $R_{f'}\pi$ is transitive, there exists $x \in \Sigma^*$ such $(f', qw) \xrightarrow{x} (f', f)$.

It follows that

$$(p', p) \xrightarrow{w} (f', f) \xrightarrow{x} (f', fx), \quad (q', q) \xrightarrow{w} (q'w', qw) \xrightarrow{x} (q'w'x', f).$$

- Since $qwx = f$, $qw \neq f$ and x is a permutation, we have $fx \neq f$. It follows that $(f', fx) \in F' \times \overline{F}$.
- Since $f'x' = f'$, $q'w' \neq f'$ and x' is a permutation, we have $q'w'x' \neq f'$. It follows that $(q'w'x', f) \in \overline{F'} \times F$.

Hence wx maps (p, p') to a final state and (q, q') to a non-final state. Thus we have distinguished the two states.

$\boldsymbol{\overline{F'} \times F}$: We can use a symmetric argument to the previous case.

$\boldsymbol{\overline{F} \times \overline{F'}}$: As in the case of $F' \times \overline{F}$, pick w such that $(p', p) \xrightarrow{w} (f', f)$. Then $(q', q) \xrightarrow{w} (q'w', qw)$, which is in $\overline{F'} \times \overline{F}$ since $q'w' \neq f'$ and $qw \neq f$. Thus w sends (q', q) to a final state. But $(p', p) \xrightarrow{w} (f', f)$ is non-final, so we have distinguished the states.

$\boldsymbol{(F' \times \overline{F}) \cup (\overline{F'} \times F)}$: This is the most complicated case.

- By transitivity of $G_\times$, there exists $u \in \Sigma^*$ such that $(p', p) \xrightarrow{u} (r', f)$, where $r' \neq f'$.
- We have $(r', f) \in \overline{F'} \times F$, so u sends (p', p) to a final state. If $(q', q) \xrightarrow{u} (q'u', qu)$ is non-final, then u distinguishes the states, so we may assume without loss of generality that it is final.
- We cannot have $qu = f$, since $p \neq q$ and $pu = f$. Thus $qu \in \overline{F}$. Since $(q'u', qu)$ is final we therefore must have $q'u' \in F'$, that is, $q'u' = f'$.

Define $r = qu$; now we have reduced the problem to distinguishing two states of the forms (r', f) and (f', r), with $r' \neq f'$ and $r \neq f$.

Suppose $|Q| \geq 3$; if we only have $|Q'| \geq 3$ we can use a symmetric argument to the argument below.

- Since $R_{f'}\pi$ is transitive and $|Q| \geq 3$, there is a word $v \in \Sigma^*$ such that $(f', f) \xrightarrow{v} (f', s)$ for some $s \notin \{r, f\}$.
- It follows that $(r', f) \xrightarrow{v} (r'v', s)$, where $r'v' \neq f'$.
- The state $(r'v', s)$ is in $\overline{F'} \times \overline{F}$, and thus is non-final. If $(f', r) \xrightarrow{v} (f', rv)$ is final, then v distinguishes (r', f) and (f', r). Hence we may assume without loss of generality that (f', rv) is non-final.
- A non-final state either lies in $F \times F'$ or $\overline{F} \times \overline{F'}$. Since $f' \in F'$, we must have $(f', rv) \in F \times F'$. But then $rv = f$.

Thus we have

$$(p', p) \xrightarrow{u} (r', f) \xrightarrow{v} (r'v', s), \quad (q', q) \xrightarrow{u} (f', r) \xrightarrow{v} (f, f').$$

Now, apply v again to both states.

- Since $s \neq r$ and $rv = f$, we have $sv \neq f$.
- Since $f'v' = f'$ and $r' \neq f'$, we have $r'v' \neq f'$ and $r'v'v' \neq f'$.
- It follows that $(r'v', s) \xrightarrow{v} (r'v'v', sv) \in \overline{F'} \times \overline{F}$, and thus is non-final.
- However, recall that $fv = s$ and $s \neq f$; thus $(f', f) \xrightarrow{v} (f', s)$ is in $F' \times \overline{F}$.

Hence (p', p) and (q', q) are distinguished by uv^2.

We have shown that all pairs of states of $\mathcal{A} \times \mathcal{A}'$ are distinguishable under all (F', F)-compatible sets of final states, and so this proves (1) $\implies$ (2).

The implication (2) $\implies$ (3) is immediate. For (3) $\implies$ (1), just note that for each proper binary boolean operation $\circ$, the language $L' \circ L$ is recognized by $(\mathcal{A}' \times \mathcal{A})(X)$ for some set of final states X. If $(\mathcal{A}' \times \mathcal{A})(X)$ is not accessible, it cannot be minimal and so $L' \circ L$ cannot have maximal state complexity. $\square$

Note that Example 4.3.4 gives a pair of languages recognized by two-state permutation DFAs which have maximal complexity for intersection, but not symmetric difference (see also [2, Example 2]). Hence in the previous theorem, it was necessary to assume that at least one DFA has three or more states.

Note that Theorem 4.4.6 also holds in the following cases:

- $\mathcal{A}'$ and $\mathcal{A}$ both have exactly one *non-final* state.
- $\mathcal{A}'$ has exactly one final state and $\mathcal{A}$ has exactly one non-final state.
- $\mathcal{A}'$ has exactly one non-final state and $\mathcal{A}$ has exactly one final state.

The same arguments we gave in Theorem 4.4.6 can be used in the above three cases, but the role of each argument is changed. For example, consider the case where $\mathcal{A}'$ has one final state and $\mathcal{A}$ has one non-final state. Let $F' = \{f'\}$ and let $\overline{F} = Q \setminus F = \{q\}$. We can use the same arguments as in the original proof of Theorem 4.4.6, except wherever F appears we substitute $\overline{F}$. So for example, we deal with the case of $F' \times \overline{F} = \{(f', q)\}$ by appealing to Corollary 4.2.9, just like we did for $F' \times F$ in the original proof. This works because distinguishability arguments are the same whether we distinguish with respect to a set of final states or a set of non-final states.

We now apply Theorem 4.4.6 to show that the original witnesses for the maximal state complexity of union (found by Maslov and later by Yu, Zhuang and Salomaa) are in fact witnesses for all proper binary boolean operations.

Example 4.4.7. In [62], Maslov defined two families of DFAs over alphabet $\{0, 1\}$ as follows, and claimed that the languages they recognize are witnesses for union. The DFA $\mathcal{A}$ has states $\{S_0, \dots, S_{m-1}\}$ with S_0 initial and S_{m-1} final, and the transitions are given by $S_i0 = S_i$, $S_i1 = S_{i+1}$ for $i \neq m-1$, and $S_{m-1}1 = S_0$. The DFA $\mathcal{B}$ has states $\{P_0, \dots, P_{n-1}\}$ with P_0 initial and P_{n-1} final, and the transitions are given by $P_i1 = P_i$, $P_i0 = P_{i+1}$ for

$i \neq n-1$, and $P_{n-1}0 = P_0$. It is easy to see that $\mathcal{A} \times \mathcal{B}$ is accessible: the state (S_i, P_j) can be reached from (S_0, P_0) via the word 1^i0^j. Furthermore, $\mathcal{A}$ and $\mathcal{B}$ are permutation DFAs: the symbol 0 acts as the identity permutation in $\mathcal{A}$ and as a cyclic permutation of the states in $\mathcal{B}$, while 1 acts as a cyclic permutation in $\mathcal{A}$ and the identity in $\mathcal{B}$. They also have exactly one final state. So in fact, for $m, n \geq 3$, the pair of languages $(L(\mathcal{A}), L(\mathcal{B}))$ has maximal boolean complexity by Theorem 4.4.6. That is, Maslov's languages are witnesses for all proper binary boolean operations, not only for union.

Yu, Zhuang and Salomaa gave a different family of witnesses in [79]. For a word $w \in \Sigma^*$ and $a \in \Sigma$, let $|w|_a$ denote the number of occurrences of the letter a in w. Yu et al. defined languages $L_m = \{w \in \{a,b\}^* : |w|_a \equiv 0 \pmod{m}\}$ and $L_n = \{w \in \{a,b\}^* : |w|_b \equiv 0 \pmod{n}\}$, then proved that $L_m \cap L_n$ and $\overline{L_m} \cup \overline{L_n}$ both have state complexity mn. In fact, (L_m, L_n) has maximal boolean complexity for $m, n \geq 3$. Indeed, one may verify that the minimal DFA of L_m has m states, with the initial and final states equal; the letter a acts as a cyclic permutation of the state set, and the letter b acts as the identity. The minimal DFA of L_n is similar, except there are n states, b is the cyclic permutation, and a is the identity. These DFAs are almost identical to the DFAs defined by Maslov, except with a different choice of final state. This does not change the fact that they are permutation DFAs with one final state and an accessible direct product, and hence meet the conditions of Theorem 4.4.6. ■

4.5 Distinguishability in the Direct Product

We now give sufficient conditions for a pair of permutation DFAs $(\mathcal{A}', \mathcal{A})$ to be uniformly boolean minimal. This involves examining the distinguishability of states. Just as with uniform minimality, primitive groups play an important role in our conditions for uniform boolean minimality. To state our conditions, we need some new notation.

Definition 4.5.1. For $p', q' \in Q'$ and $p, q \in Q$, we write $R_{p',q'}$ for the setwise stabilizer of $(p', *) \cup (q', *)$, and $C_{p,q}$ for the setwise stabilizer of $(*, p) \cup (*, q)$. If $p' = q'$ then $R_{p',q'} = R_{p'} = R_{q'}$, and similarly if $p = q$ then $C_{p,q} = C_p = C_q$. We call these subgroups *double row stabilizers* and *double column stabilizers*.

Under this definition, single row stabilizers are special cases of double row stabilizers, and similarly for column stabilizers. Note that $R_{p'}$ and $R_{q'}$ are not necessarily subgroups of $R_{p',q'}$, nor the other way around: the group $R_{p'}$ might contain elements that map q' to some state $r' \notin \{p', q'\}$, while the group $R_{p',q'}$ might contain elements that swap p' and

q'. However, the full row stabilizer R is a common subgroup of $R_{p'}$, $R_{q'}$ and $R_{p',q'}$. The analogous facts hold for column stabilizers.

Lemma 4.5.2. *Suppose $\mathcal{A}'$ and $\mathcal{A}$ are permutation DFAs.*

1. *If $|Q| \geq 3$, the group G is primitive, and $C_{p,q}\pi' \leq G'$ is primitive for all $p, q \in Q$, then $\mathcal{A}' \times \mathcal{A}$ is uniformly boolean minimal.*
2. *If $|Q'| \geq 3$, the group G' is primitive, and $R_{p',q'}\pi \leq G$ is primitive for all $p', q' \in Q'$, then $\mathcal{A}' \times \mathcal{A}$ is uniformly boolean minimal.*

Note that we do not require $p \neq q$ or $p' \neq q'$, so in case (1) both the single and double column stabilizers must have primitive projections, and in case (2) both the single and double row stabilizers must have primitive projections.

Proof. Suppose that the conditions of (1) hold, that is, $|Q| \geq 3$, G is primitive and for all $p, q \in Q$, the subgroup $C_{p,q}\pi' \leq G'$ is primitive. The case where the conditions of (2) hold is symmetric. We want to show that for every pair of sets $\emptyset \subsetneq S' \subsetneq Q'$ and $\emptyset \subsetneq S \subsetneq Q$ and each (S', S)-compatible subset $X \subseteq Q' \times Q$, the DFA $(\mathcal{A}' \times \mathcal{A})(X)$ is minimal.

It suffices to consider the cases where $X = S' \times S$ and where $X = (S' \times S) \cup (\overline{S'} \times \overline{S})$. It may seem that this would only cover sets of type (a) and complements of sets of type (e) from Proposition 4.3.6. However, these two cases actually cover all possible types of (S', S)-compatible sets.

To see this, consider a set $S' \times \overline{S}$ of type (b). If we prove that $(\mathcal{A}' \times \mathcal{A})(T' \times T)$ is minimal for *all* pairs of sets $\emptyset \subsetneq T' \subsetneq Q'$ and $\emptyset \subsetneq T \subsetneq Q$, then in particular we can take $T' = S'$ and $T = \overline{S}$ to show that $(\mathcal{A}' \times \mathcal{A})(S' \times \overline{S})$ is covered. A similar argument works for sets of type (c) and (d) .

Now note that if $(\mathcal{A}' \times \mathcal{A})(X)$ is minimal, then $(\mathcal{A}' \times \mathcal{A})(\overline{X})$ is also minimal, so we also cover sets of type (e) and (f). So all types of sets from Proposition 4.3.6 are covered by just looking at the cases $X = S' \times S$ and $X = (S' \times S) \cup (\overline{S'} \times \overline{S})$.

Let (i', i) and (j', j) be distinct states of $\mathcal{A}' \times \mathcal{A}$; we will show they are distinguishable under X.

Case 1 (States in the same column): Suppose $i = j$, that is, the states are in the same column. Since the states are distinct, we have $i' \neq j'$.

- Since G is primitive, it is transitive, and thus there exists a word $w \in G$ that maps $i = j$ to some element $s \in S$. Thus $(i', i) \xrightarrow{w} (i'w', s)$ and $(j', j) \xrightarrow{w} (j'w', s)$.

- Suppose w does *not* distinguish (i', i) and (j', j) with respect to X. Then $(i'w', s) \in X \iff (j'w', s) \in X$.
- Since $C_s\pi'$ is primitive, all permutation DFAs with state set Q' and transition group $C_s\pi'$ are uniformly minimal by Corollary 4.2.4. It follows there exists $x' \in C_s\pi'$ that distinguishes $i'w'$ and $j'w'$ with respect to S'.

We have:

$$(i, i') \xrightarrow{w} (i'w', s) \xrightarrow{x} (i'w'x', s), \quad (j, j') \xrightarrow{w} (j'w', s) \xrightarrow{x} (j'w'x', s).$$

- Since x' distinguishes $i'w'$ and $j'w'$ with respect to S', we see that $i'w'x' \in S \iff j'w'x' \notin S$.
- Hence $(i'w'x', s) \in S' \times S \iff (j'w'x', s) \in \overline{S'} \times S$.

It follows that either w or wx distinguishes (i, i') and (j, j') with respect to X, regardless of whether we have $X = S \times S'$ or $X = (S \times S') \cup (\overline{S} \times \overline{S'})$.

Case 2 (States in the same row): Suppose $i' = j'$, that is, the states are in the same row but different columns. Since G is primitive, by Corollary 4.2.4, all permutation DFAs with state set Q and transition group G are uniformly minimal. Hence there exists $x \in G$ that distinguishes i and j under S. Suppose without loss of generality that $ix \in S$ and $jx \notin S$. Since $C_{ix,jx}\pi'$ is primitive, it is transitive, and thus there exists $y' \in C_{ix,jx}\pi'$ such that $i'x'y' \in S'$. Since $C_{ix,jx}$ stabilizes the set $(ix, *) \cup (jx, *)$, for $(y', y) \in C_{ix,jx}$, we must have $\{ix, jx\}y = \{ixy, jxy\} = \{ix, jx\}$. Thus y either fixes ix and jx, or it swaps ix and jx. If y fixes ix and jx, then $(i'x'y', ixy) = (i'x'y', ix) \in S' \times S$, and $(j'x'y', jxy) = (j'x'y', jx) \in S' \times \overline{S}$ since $i' = j'$ implies $i'x'y' = j'x'y'$. If y swaps ix and jx then $(i'x'y', ixy) = (i'x'y', jx) \in S' \times \overline{S}$ and $(j'x'y', jxy) = (j'x'y', ix) \in S' \times S$. In either case, it follows that xy distinguishes (i', i) and (j', j) under X, regardless of whether we have $X = S' \times S$ or $X = (S' \times S) \cup (\overline{S'} \times \overline{S})$.

Case 3 (States in different rows and different columns): Suppose $i' \neq j'$ and $i \neq j$. We divide this case into two subcases.

Subcase 3a ($\boldsymbol{X = S \times S'}$): We may assume without loss of generality that (i', i) and (j', j) are both in X. To see this, observe that since G is transitive and $C_q\pi' \leq G'$ is transitive for each $q \in Q$, we know that $G_\times$ is transitive by Lemma 4.4.4. Thus there exists $w \in \Sigma^*$ that maps (i', i) to a state in X. Hence either w distinguishes the states, or w also maps (j', j) into X.

Suppose we have $(i', i), (j', j) \in X = S' \times S$. Since $C_{i,j}\pi'$ is primitive, there exists $w \in C_{i,j}\pi'$ that distinguishes i' and j' under S'. Without loss of generality, assume $i' \in S'$ and $j' \notin S'$. Then $(i,' i)(w', w) = (i', i) \in S' \times S$ and $(j', j)(w', w) \in \overline{S'} \times S$. Hence w distinguishes the states.

Subcase 3b ($\boldsymbol{X = (S \times S') \cup (\overline{S} \times \overline{S'})}$): This is the final case we must deal with, and most complicated part of the proof. We introduce a notion of *polarity* to simplify the arguments. We assign a polarity of 1 or -1 to each state in $Q' \times Q$ as follows. First, let $q \in Q'$ have polarity 1 if $q' \in S'$ and polarity -1 if $q' \notin S'$. Similarly, $q \in Q$ has polarity 1 if $q \in S$ and polarity -1 if $q \notin S$. Then the polarity of $(q', q) \in Q' \times Q$ is the product of the polarities of q' and q.

Next, we partition $Q' \times Q$ into four *quadrants*: $S' \times S$, $S' \times \overline{S}$, $\overline{S'} \times S$, and $\overline{S'} \times \overline{S}$. Notice that in each quadrant, all states have the same polarity. Furthermore, the set $X = (S' \times S) \cup (\overline{S'} \times \overline{S})$ is the set of all states with *positive* polarity, and the set $\overline{X} = (S' \times \overline{S}) \cup (\overline{S'} \times S)$ is the set of all states with *negative* polarity. Hence to show all states are distinguishable under X, we must show that for each pair of states of equal polarity, there is a word that preserves the polarity of one state and reverses the polarity of the other state.

We now prove two claims, which together complete the proof of this subcase. First we show that pairs of states in the same quadrant are distinguishable, and then we show that pairs of states in different quadrants are distinguishable.

Claim 1 (States in the same quadrant are distinguishable): Suppose (i', i) and (j', j) are in the same quadrant. This means that (i', i) and (j', j) have the same polarity; furthermore, i' and j' have the same polarity, and i and j have the same polarity.

- Choose a word $w' \in C_{i,j}\pi'$ that distinguishes i' and j' under S' (by primitivity of $C_{i,j}\pi'$).
- Notice that w preserves the polarity of both i and j, since it either fixes both i and j or it swaps them, and i and j have the same polarity.
- Since (i', i) and (j', j) are in the same quadrant, we either have $i', j' \in S'$ or $i', j' \in \overline{S'}$.
- Since w' distinguishes i' and j' under S', it follows that w' acts on i' and j' by preserving the polarity of one state and reversing the polarity of the other.

It follows that (w', w) acts on (i', i) and (j', j) by preserving the polarity of one state and reversing the polarity of the other. In other words, w distinguishes these states.

Claim 2 (States in different quadrants are distinguishable): Suppose that (i', i) and (j', j) lie in different quadrants.

- We may assume without loss of generality that (i', i) and (j', j) have the same polarity; otherwise they are trivially distinguishable.
- We may also assume without loss of generality that $(i', i), (j', j) \in X$, by the same argument we used in Subcase 3a. Thus we must have $(i', i) \in S \times S'$ and $(j', j) \in \overline{S} \times \overline{S'}$, or vice versa.
- We may assume without loss of generality that $(i', i) \in S \times S'$ and $(j', j) \in \overline{S} \times \overline{S'}$, by swapping the names of (i', i) and (j', j) if necessary.

So we have reduced to the case where one state is in the quadrant $S' \times S$ and the other is in the quadrant $\overline{S'} \times \overline{S}$.

- Since G is primitive, S and $\overline{S}$ are either not blocks, or they are trivial blocks.
- S and $\overline{S}$ are proper non-empty subsets of Q, so they can only be trivial blocks if $|S| = |\overline{S}| = 1$.
- This would imply $|Q| = |S| + |\overline{S}| = 2$, and we are assuming $|Q| \geq 3$, so they cannot both be trivial blocks. So at least one of S or $\overline{S}$ is not a block. Note that in the symmetric case where G' is primitive, we would use $|Q'| \geq 3$ here.

If S is not a block, let $w \in G$ be a word such that $\emptyset \subsetneq Sw \cap S \subsetneq S$. Otherwise, $\overline{S}$ is not a block, so let $w \in G$ be a word such that $\emptyset \subsetneq \overline{S}w \cap \overline{S} \subsetneq \overline{S}$.

Now, we partition Q into two sets:

$$P = \{q \in S : qw \in S\} \cup \{q \in \overline{S} : qw \in \overline{S}\},$$

$$\overline{P} = \{q \in S : qw \in \overline{S}\} \cup \{q \in \overline{S} : qw \in S\}.$$

Note that P is non-empty, since if it were empty we would have $Sw \cap S = \emptyset$ and $\overline{S}w \cap \overline{S} = \emptyset$. Similarly, $\overline{P}$ is non-empty, since otherwise we would have $Sw \cap S = S$ and $\overline{S}w \cap \overline{S} = \overline{S}$. Thus both P and $\overline{P}$ are proper subsets of Q.

Observe that if $i, j \in P$, then P is a proper subset of Q with size at least two, so it cannot be a block for G. Hence some word in G distinguishes i and j under P. Similarly, if $i, j \in \overline{P}$, then i and j are distinguishable under $\overline{P}$. So we may assume that either $i \in P$ and $j \in \overline{P}$, or $i \in \overline{P}$ and $j \in P$.

Suppose that $i \in P$ and $j \in \overline{P}$. Recall that we have $(i', i) \in S' \times S$ and $(j', j) \in \overline{S'} \times \overline{S}$. Thus since $i \in S$ we have $iw \in S$, and since $j \in \overline{S}$ we have $jw \in S$. Hence $(i'w', iw), (j'w', jw) \in Q' \times S$, that is, w maps both (i', i) and (j', j) into $Q' \times S$. There are two possibilities:

- The states $(i'w', iw)$ and $(j'w', jw)$ are in the same quadrant, and thus are distinguishable by Claim 1.
- The states $(i'w', iw)$ and $(j'w', jw)$ are in different quadrants. Since iw and jw are both in S, one state must lie in $S' \times S$ and the other $\overline{S'} \times S$. Thus w distinguishes (i', i) and (j', j), and we are done.

So if $i \in P$ and $j \in \overline{P}$, we have proved the claim. If we have $i \in \overline{P}$ and $j \in P$, then we have $iw \in \overline{S}$ and $jw \in \overline{S}$, and so a symmetric argument shows that the states are distinguishable. This completes the proof of Claim 2. By Claim 1 and Claim 2, we see that all pairs of states are distinguishable under sets of the form $(S' \times S) \cup (\overline{S'} \times \overline{S})$, completing the proof of Subcase 3b and hence Case 3.

We have shown that for every pair of sets $\emptyset \subsetneq S' \subsetneq Q'$ and $\emptyset \subsetneq S \subsetneq Q$ and each (S', S)-compatible subset $X \subseteq Q' \times Q$, each pair of states in $(\mathcal{A}' \times \mathcal{A})(X)$ is distinguishable by X. Thus $\mathcal{A}' \times \mathcal{A}$ is uniformly boolean minimal. $\square$

While Lemma 4.5.2 gives sufficient conditions for uniform boolean minimality, the conditions are not necessary. We will demonstrate this later, in Example 4.9.4.

In the above proof, most of the difficulty came from dealing with the case $(S' \times S) \cup (\overline{S'} \times \overline{S})$, which corresponds to the operation of symmetric difference (or complement of symmetric difference). In fact, if we choose to ignore symmetric difference and its complement, we can obtain necessary and sufficient conditions for the corresponding weaker version of uniform boolean minimality. First we need a lemma.

Lemma 4.5.3. *Let G be a finite transitive group acting on X and fix $S \subseteq X$. If there exists $g \in G$ such that $ig \notin S$ and $jg \in S$, then there exists $h \in G$ such that $ih \in S$ and $jh \notin S$.*

Proof. Fix $g \in G$ such that $ig \notin S$ and $jg \in S$. By transitivity of G, we can choose $x \in G$ such that $(ig)x = jg$. Thus $igx \in S$. If $igx \in S$ and $jgx \notin S$, we can take $h = gx$. Otherwise, we construct h as follows:

- Step 1: We know that $jgx \in S$. Then since $igx = jg$, we have $igx^2 = jgx \in S$. If $jgx^2 \notin S$, take $h = gx^2$. Otherwise, we know that $jgx^2 \in S$.
- Step $k > 1$: Assume we know that $jgx^k \in S$. Then $igx^{k+1} = jgx^k \in S$. If $jgx^{k+1} \notin S$, take $h = gx^{k+1}$.

This construction necessarily terminates, since G is a finite group, and thus there exists k such that x^k is the identity. Once we reach step $k-1$, we know that $jgx^{k-1} \in S$, and so $igx^k = jgx^{k-1} = ig \in S$; also, $jgx^k = jg \notin S$. So we take $h = gx^k = g$ and stop. Hence we can always produce an element h with the desired property. □

Proposition 4.5.4. *Suppose $\mathcal{A}'$ and $\mathcal{A}$ are permutation DFAs. The following are equivalent:*

1. *$C_q\pi' \leq G'$ and $R_{q'}\pi \leq G$ are primitive for all $q \in Q$ and $q' \in Q'$.*
2. *For all sets $\emptyset \subsetneq S' \subsetneq Q'$ and $\emptyset \subsetneq S \subsetneq Q$, the DFA $(\mathcal{A}' \times \mathcal{A})(S' \times S)$ is minimal.*

Proof. (1) $\implies$ (2): We proceed as in the proof of Lemma 4.5.2. Let (i', i) and (j', j) be distinct states of $\mathcal{A}' \times \mathcal{A}$; we will show they are distinguishable under $S' \times S$.

Case 1 (States in the same row): In the proof of Lemma 4.5.2, we proved that states in the same row are distinguishable, using the facts that G is transitive and $C_q\pi'$ is primitive for all $q \in Q$. Those facts still hold under our new hypotheses, so the same argument can be used here.

Case 2 (States in the same column): The argument we used for this case in Lemma 4.5.2 relied on the double row stabilizers being transitive, so we cannot use it here. However, under our new hypotheses, we know that G' is transitive and $R_{q'}\pi$ is primitive for all $q' \in Q$. Thus we can just use a symmetric argument to the one for states in the same row.

Case 3 (States in different rows and different columns): Suppose $i' \neq j'$ and $i \neq j$. As in the proof of Lemma 4.5.2, we may assume without loss of generality that $(i', i), (j', j) \in S' \times S$. Since $R_{i'}\pi$ is primitive, there exists $w \in R_{i'}\pi$ that distinguishes i and j under S. Then we either have $iw \in S$ and $jw \notin S$, or we have $iw \notin S$ and $jw \in S$. In the latter case, by Lemma 4.5.3, there exists $x \in R_{i'}\pi$ such that $ix \in S$ and $jx \notin S$. So without loss of generality, we may assume $iw \in S$ and $jw \notin S$. Then we have $(i', i) \xrightarrow{w} (i', iw) \in S \times S'$, but $(j', j) \xrightarrow{w} (j'w', jw) \notin S \times S'$ since $jw \notin S$. Thus w distinguishes the states. We have now shown that all pairs of states (i', i) and (j', j) are distinguishable.

(2) $\implies$ (1): Suppose that for all sets $\emptyset \subsetneq S' \subsetneq Q'$ and $\emptyset \subsetneq S \subsetneq Q$, the DFA $(\mathcal{A}' \times \mathcal{A})(S' \times S)$ is minimal. Assume for a contradiction that there exists $q' \in Q'$ such that $R_{q'}\pi$ is not primitive. Then there exists a non-trivial block $B \subsetneq Q$ for $R_{q'}\pi$. We claim that $(\mathcal{A}' \times \mathcal{A})(\{q'\} \times B)$ is not minimal:

- Since B is a non-trivial block, it contains at least two distinct elements. Let $i, j \in B$ be distinct and consider the states (q', i) and (q', j) of $\mathcal{A}' \times \mathcal{A}$.
- If $q'w' \neq q'$, then w does not distinguish (q', i) and (q', j). Indeed, if $q'w' \neq q'$, then $(q'w', iw)$ and $(q'w', jw)$ both lie outside of $\{q'\} \times B$.
- Hence if $w \in \Sigma^*$ distinguishes (q', i) and (q', j), we must have $q'w' = q'$, and thus $w \in R_{q'}\pi$.
- Since B is a block for $R_{q'}\pi$, we either have $iw, jw \in B$ (if $Bw = B$) or $\{iw, jw\} \cap B = \emptyset$ (if $Bw \cap B = \emptyset$).
- Thus $w \in R_{q'}\pi$ cannot distinguish (q', i) and (q', j): if $iw, jw \in B$ then w maps both states to $\{q'\} \times B$; otherwise it maps both states to $\{q'\} \times \overline{B}$.
- It follows that no word can distinguish (q', i) and (q', j) under $\{q'\} \times B$.

This shows that $(\mathcal{A}' \times \mathcal{A})(\{q'\} \times B)$ is not minimal, which is a contradiction. It follows that $R_{q'}\pi$ must be primitive for all $q' \in Q'$. A symmetric argument shows that $C_q\pi'$ must be primitive for all $q \in Q$. □

One may wonder whether condition (1) of Proposition 4.5.4 is actually sufficient to prove Lemma 4.5.2. We will show later (Example 4.9.3) that this is not the case.

4.6 Dissimilar DFAs and the Main Result

While the conditions of Lemma 4.5.2 are somewhat complicated, there are cases where we can easily verify that they hold. We consider some of these cases next.

Recall that the projection maps $\pi' \colon M_\times \to M'$ and $\pi \colon M_\times \to M$, given by $(w', w)\pi' = w'$ and $(w', w)\pi = w$ respectively, are surjective by Proposition 4.4.1. We will say that the DFAs $\mathcal{A}'$ and $\mathcal{A}$ are *similar* if the maps π' and π are injective. If these maps are both surjective and injective, they are isomorphisms. Hence if $\mathcal{A}'$ and $\mathcal{A}$ are similar, the map

$(\pi')^{-1}\pi\colon M' \to M$ given by $w' \mapsto w$ is a well-defined monoid isomorphism. If $\mathcal{A}'$ and $\mathcal{A}$ are not similar, we say they are *dissimilar*. If π' and π *both* fail to be injective, we say that $\mathcal{A}$ and $\mathcal{A}'$ are *strongly dissimilar*.

We give some examples of dissimilar DFAs. All DFAs will be over the two-letter alphabet $\{a, b\}$, and we will not specify the initial and final states since they do not affect whether DFAs are similar.

Example 4.6.1. Let $\mathcal{A}'$ have states $\{1,2\}$ and actions $a' = (1,2)$ and $b' = \mathrm{id}$. Let $\mathcal{A}$ have states $\{1,2\}$ and actions $a = \mathrm{id}$ and $b = (1,2)$. Notice that in the transition group of $\mathcal{A} \times \mathcal{A}'$, we have $(b', b) = (\mathrm{id}, (1,2))$ and $(\varepsilon', \varepsilon) = (\mathrm{id}, \mathrm{id})$. Thus $(b', b)\pi' = (\varepsilon', \varepsilon)\pi' = \mathrm{id}$ and it follows that π' is not injective. Hence $\mathcal{A}$ and $\mathcal{A}'$ are dissimilar. In fact, a symmetric argument shows that π is not injective, and thus these DFAs are strongly dissimilar.

Another way to see that these DFAs are dissimilar is to consider the binary relation $w' \mapsto w$. This relation is not a well-defined function, since we have $b' \mapsto b$ which gives $\mathrm{id} \mapsto (1,2)$, but we also have $\varepsilon' \mapsto \varepsilon$ which gives $\mathrm{id} \mapsto \mathrm{id}$. This means $\mathcal{A}'$ and $\mathcal{A}$ cannot be similar, since we know that if they are similar, then $(\pi')^{-1}\pi$ must be a well-defined isomorphism which sends w' to w.

Alternatively, without even checking whether the relation $w' \mapsto w$ is a function, we can see that since $\mathrm{id} = b' \mapsto b = (1,2)$, this relation sends an element of order one to an element of order two, and thus it cannot possibly be a group isomorphism (since group isomorphisms preserve the order of elements). But then $\mathcal{A}'$ and $\mathcal{A}$ cannot be similar. ■

Usually, the easiest way to prove that a pair of DFAs is dissimilar is to examine the relation $w \mapsto w'$ and show that it is not an isomorphism.

Example 4.6.2. Let $\mathcal{A}'$ have states $\{1,2\}$ and actions $a' = (1,2)$ and $b' = (1,2)$. Let $\mathcal{A}$ have states $\{1,2,3,4\}$ and actions $a = (1,2)(3,4)$ and $b = (1,3)(2,4)$. The transition group of $\mathcal{A}'$ has two elements: $\varepsilon' = \mathrm{id}$ and $a' = b' = (1,2)$. However, the transition group of $\mathcal{A}$ has four elements:

$$\varepsilon = \mathrm{id},\ a = (1,2)(3,4),\ b = (1,3)(2,4),\ ab = (1,4)(2,3).$$

Hence these DFAs must be dissimilar, since they have different transition groups. Similar DFAs always have isomorphic transition monoids or groups.

These DFAs are not strongly dissimilar. To see this, observe that the transition group of $\mathcal{A} \times \mathcal{A}'$ has four elements:

$$(\varepsilon', \varepsilon) = (\mathrm{id}, \mathrm{id}), \quad (a', a) = ((1,2), (1,2)(3,4)),$$

$$(b', b) = ((1,2),(1,3)(2,4)), \quad (a'b', ab) = (\mathrm{id}, (1,4)(2,3)).$$

It is easy to verify that any other product of elements will be equal to one of these four. Now note that π is a bijection but π' is not, and thus $\mathcal{A}'$ and $\mathcal{A}$ are not strongly dissimilar. In this case, the relation $w' \mapsto w$ is not a function. However, the relation $\pi^{-1}\pi'$ given by $w \mapsto w'$ *is* a function, and in fact is a group homomorphism (but not an isomorphism). ■

As for examples of similar DFAs, we have the following fact: isomorphic DFAs are necessarily similar. Indeed, suppose there is an isomorphism $f\colon Q' \to Q$. Then for all $q' \in Q'$ and $a \in \Sigma$, we have $(q'a')f = (qf)a$, and thus $q'a' = q(faf^{-1})$. It follows that $a' = faf^{-1}$ for all $a \in \Sigma^*$, and thus $w' = fwf^{-1}$ for all $w \in \Sigma^*$. Hence $\pi'\colon M_\times \to M'$ is given by $(w', w)\pi' = (fwf^{-1}, w)\pi' = fwf^{-1}$, and this map is clearly injective since if $fwf^{-1} = fxf^{-1}$ then $w = x$. Similarly, we have $w = f^{-1}w'f$ and $\pi\colon M_\times \to M$ is injective.

This shows that DFA similarity is a generalization of DFA isomorphism. However, the next example shows that it is possible for two DFAs with different numbers of states to be similar, and that the monoid isomorphism $w' \mapsto w$ does not necessarily have to have the form $w' \mapsto f^{-1}wf$ for a permutation f.

Example 4.6.3. Consider the symmetric group S_{10}. This group contains an intransitive subgroup that is isomorphic to S_5, given by permutations of the set $\{1,\dots,10\}$ which fix every point in $\{6,\dots,10\}$. This can be considered the "natural" embedding of S_5 in S_{10}. However, there is also a primitive subgroup of S_{10} that is isomorphic to S_5. Using the computer algebra system GAP [38], which contains a library of primitive groups, this subgroup can be accessed with the command `PrimitiveGroup(10,2)`. We can use GAP to compute an explicit isomorphism between S_5 and this subgroup:

```
gap> IsomorphismGroups(SymmetricGroup(5),PrimitiveGroup(10,2));
[ (3,4), (1,2,3)(4,5) ] ->
[ (1,2)(6,8)(7,9), (2,6,4,5,3,7)(8,10,9) ]
```

The GAP output tells us that the map sending $(3,4)$ to $(1,2)(6,8)(7,9)$ and $(1,2,3)(4,5)$ to $(2,6,4,5,3,7)(8,10,9)$ can be extended multiplicatively to an isomorphism between S_5 and the aforementioned primitive subgroup of S_{10}. We use this isomorphism to construct similar permutation DFAs of different sizes.

Let $\mathcal{A}'$ have states $\{1,\dots,5\}$ and actions $a' = (3,4)$ and $b' = (1,2,3)(4,5)$. Let $\mathcal{A}$ have states $\{1,\dots,10\}$ and actions $a = (1,2)(6,8)(7,9)$ and $b = (2,6,4,5,3,7)(8,10,9)$.

The transition group G' of $\mathcal{A}'$ is S_5, and the transition group G of $\mathcal{A}$ is a primitive subgroup of S_{10} that is isomorphic to S_5. Furthermore, the map $w' \mapsto w$ is an isomorphism of G and G'. Hence $\mathcal{A}'$ and $\mathcal{A}$ are similar. Note that the map $w' \mapsto w$ cannot be written in the form $w' \mapsto f^{-1}wf$ for a permutation f; if it could, we would have $a' = f^{-1}af$, but one can show that $f^{-1}af = (1f, 2f)(6f, 8f)(7f, 9f)$. This pair of DFAs has other interesting properties; we will revisit them in Example 4.7.1.

Notice that similarity of DFAs is a very fragile property; if we simply switch the roles of a and b in $\mathcal{A}$, giving $a = (3, 4)$ and $a' = (2, 6, 4, 5, 3, 7)(8, 10, 9)$, then $\mathcal{A}$ and $\mathcal{A}'$ are no longer similar. Indeed, after the switch, we see that $w' \mapsto w$ sends an element of order two to an element of order six, which means it cannot be an isomorphism of G' and G. ■

Dissimilar permutation DFAs have the following nice property. Recall that $R = \ker \pi'$ and $C = \ker \pi$, the *full row stabilizer* and *full column stabilizer* (respectively) are normal subgroups of $G_\times$, the transition group of $\mathcal{A}' \times \mathcal{A}$. Indeed, they are kernels of homomorphisms, which are always normal.

Proposition 4.6.4. *If $\mathcal{A}'$ and $\mathcal{A}$ are dissimilar permutation DFAs, then at least one of the following statements holds:*

1. *$C\pi'$ is a non-trivial normal subgroup of G'.*
2. *$R\pi$ is a non-trivial normal subgroup of G.*

If $\mathcal{A}'$ and $\mathcal{A}$ are strongly dissimilar, then both hold.

Proof. Recall that by Proposition 2.3.2, if N is a normal subgroup of G and $\varphi\colon G \to H$ is a surjective homomorphism, then $N\varphi$ is a normal subgroup of H. Since π' and π are surjective, $C\pi'$ is a normal subgroup of G' and $R\pi$ is a normal subgroup of G.

- We have $\ker \pi = \{(w', w) \in G_\times : w = \varepsilon\}$, so $C\pi' = \{w' \in G' : w = \varepsilon\}$ and similarly $R\pi = \{w \in G : w' = \varepsilon'\}$.
- If $C\pi'$ is trivial, then whenever $w = \varepsilon$ we have $w' = \varepsilon'$, and so $C = \ker \pi = \{(\varepsilon', \varepsilon)\}$ is trivial; hence π is injective.
- Similarly, if $R\pi$ is trivial, then $R = \ker \pi'$ is trivial, and thus π' is injective.

Thus we see that if $\mathcal{A}'$ and $\mathcal{A}$ are dissimilar, then π' and π cannot both be injective, and so $C\pi'$ and $R\pi$ cannot both be trivial. Furthermore, if $\mathcal{A}'$ and $\mathcal{A}$ are strongly dissimilar, then neither π' nor π can be injective, and so neither $C\pi'$ nor $R\pi$ can be trivial. □

This leads to a powerful lemma:

Lemma 4.6.5. *Let $\mathcal{A}'$ and $\mathcal{A}$ be dissimilar permutation DFAs and suppose $|Q'|, |Q| \geq 3$.*

1. *Suppose G' and G are transitive. If all non-trivial normal subgroups of G' and of G are transitive, then $\mathcal{A}' \times \mathcal{A}$ is accessible.*

2. *Suppose G' and G are primitive. If all non-trivial normal subgroups of G' and of G are primitive, then $\mathcal{A}' \times \mathcal{A}$ is uniformly boolean minimal.*

Proof. By Proposition 4.6.4, since $\mathcal{A}'$ and $\mathcal{A}$ are dissimilar, one of $C\pi'$ or $R\pi$ is a non-trivial normal subgroup. Suppose that $R\pi \leq G$ is non-trivial; the other case is symmetric.

(1): Since all non-trivial normal subgroups of G are transitive, $R\pi$ is transitive. Hence $R_{q'}\pi$ is transitive for all $q' \in Q'$, since $R_{q'}\pi \geq R\pi$. Since G' is transitive, we see that condition (3) of Lemma 4.4.4 holds. Thus $\mathcal{A}' \times \mathcal{A}$ is accessible.

(2): Since all non-trivial normal subgroups of G are primitive, $R\pi$ is primitive. Hence $R_{p',q'}\pi$ is primitive for all $p', q' \in Q$, since $R_{p',q'}\pi \geq R\pi$. Since G' is primitive, we see that $\mathcal{A}'$ and $\mathcal{A}$ meet the conditions of Lemma 4.5.2. Thus $\mathcal{A}' \times \mathcal{A}$ is uniformly boolean minimal. $\square$

Several interesting classes of groups have the property that all non-trivial normal subgroups are transitive or primitive. The following theorem gives numerous examples of when Lemma 4.6.5 can be applied.

Theorem 4.6.6. *Let $\mathcal{A}'$ and $\mathcal{A}$ be dissimilar permutation DFAs and suppose $|Q'|, |Q| \geq 3$.*

1. *Suppose G' and G are transitive.*

   (a) *If G' and G are transitive simple groups, then $\mathcal{A}' \times \mathcal{A}$ is accessible.*

   (b) *If G' and G are primitive groups, then $\mathcal{A}' \times \mathcal{A}$ is accessible.*

2. *Suppose G' and G are primitive.*

   (a) *If G' and G are primitive simple groups, then $\mathcal{A}' \times \mathcal{A}$ is uniformly boolean minimal.*

   (b) *If G' is $S_{Q'}$ or $A_{Q'}$, and G is S_Q or A_Q, and $(|Q'|, |Q|) \notin \{(3,4),(4,3),(4,4)\}$, then $\mathcal{A}' \times \mathcal{A}$ is uniformly boolean minimal.*

(c) *If G' and G are 2-transitive groups which are not of affine type, then $\mathcal{A}' \times \mathcal{A}$ is uniformly boolean minimal.*

Before proving this, we will explain what we mean by "affine type". The notion of "affine type" comes from the O'Nan-Scott theorem [28, Theorem 4.1A], a structure theorem for primitive groups. The O'Nan-Scott theorem divides the primitive groups into different types based on their *socle*. The socle of a group G is the subgroup generated by all the *minimal normal subgroups* of G, that is, the normal subgroups N of G for which there does not exist a non-trivial normal subgroup N' of G with $N' \subsetneq N$.

A primitive group with an abelian socle is necessarily of *affine type*, which means it is a permutation group of degree p^d for p prime and $d \geq 1$, and is isomorphic to a subgroup of the *affine group* $AGL(d, p)$. This is a group of permutations of the set $\mathbb{F}_p^d$, where $\mathbb{F}_p$ is the finite field with p elements; it consists of all maps of the form $(\gamma_1, \dots, \gamma_d) \mapsto (\alpha\gamma_1 + \beta, \dots, \alpha\gamma_d + \beta)$, where $\alpha, \beta, \gamma_1, \dots, \gamma_d \in \mathbb{F}_p$. In Section 4.9, we will show that there exist DFAs $\mathcal{A}'$ and $\mathcal{A}$ whose transition groups are 2-transitive groups of affine type, such that $\mathcal{A}' \times \mathcal{A}$ is not uniformly boolean minimal. Hence in the statement of Theorem 4.6.6 (2c), excluding 2-transitive groups of affine type is necessary. We remark that A_3, S_3, A_4 and S_4 happen to be groups of affine type; we will see in the proof that this is not true for symmetric and alternating groups of larger degree.

Proof. (1a): Recall that a group is *simple* if it has no non-trivial proper normal subgroups. Hence if G is a transitive simple group, then the only non-trivial normal subgroup of G is G itself. Similarly, the only non-trivial normal subgroup of G' is G' itself. Thus the conditions of Lemma 4.6.5 hold.

(1b) Suppose G' and G are primitive. It is an easy exercise in group theory to show that all non-trivial normal subgroups of a primitive group are transitive (e.g., see [28, Theorem 1.6A]). Thus the conditions of Lemma 4.6.5 hold in this case.

(2a): Since G is a primitive simple group, then the only non-trivial normal subgroup of G is G itself, and similarly for G'. Thus the conditions of Lemma 4.6.5 hold.

(2b): First suppose $|Q'| \neq 4$ and $|Q| \neq 4$. It is well-known that A_n is simple for $n \neq 4$ (e.g., see [28, Corollary 3.3A]). Thus $A_{Q'}$ and A_Q are primitive simple groups. So if $G' = A_{Q'}$ and $G' = A_Q$, this case follows from (2a).

Now, consider the symmetric groups. We claim that for $n \neq 4$, the only non-trivial normal subgroups of S_n are A_n and S_n. This is also a fairly well-known fact, but in this case we will give a proof.

Let N be a non-trivial normal subgroup of S_n. If $A_n \leq N$, we claim that either $N = A_n$ or $N = S_n$. Indeed, if N properly contains A_n, then it contains some permutation p that cannot be written as a product of an even number of 2-cycles. But every permutation can be written as a product of 2-cycles, so p can be written as a product of an odd number of 2-cycles. Thus for each 2-cycle (i, j), we have $(i, j)p \in A_n \leq N$. So N contains $(i, j)pp^{-1} = (i, j)$; since N contains all 2-cycles, we must have $N = S_n$.

If $N \leq A_n$, consider ghg^{-1} for $g \in A_n$ and $h \in N$. Since g, h and g^{-1} are all in A_n, each can be written a product of an even number of 2-cycles, and thus ghg^{-1} can be written this way as well. So N is also a non-trivial normal subgroup of A_n; but A_n is simple for $n \neq 4$, so we must have $N = A_n$.

Thus a non-trivial normal subgroup is either S_n or A_n, both of which are primitive. Thus if $(G', G) \in \{(A_{Q'}, S_Q), (S_{Q'}, A_Q), (S_{Q'}, S_Q)\}$, Lemma 4.6.5 applies and gives the result. Note we have also shown that for $n \geq 5$, the alternating group A_n is the unique minimal normal subgroup of S_n and A_n. Thus the socle of S_n and A_n is non-abelian, which shows that S_n and A_n are not of affine type when $n \geq 5$.

Now, suppose $|Q'| = 4$ or $|Q| = 4$ and $(|Q'|, |Q|) \notin \{(3, 4), (4, 3), (4, 4)\}$. Then $|Q'| \geq 5$ or $|Q| \geq 5$. Assume without loss of generality that $|Q'| \geq 5$ and $|Q| = 4$; the other case is symmetric. Consider the normal subgroup $C\pi'$ of G'. If $C\pi'$ is trivial, then as we argued in the proof of Proposition 4.6.4, the map $\pi \colon G_\times \to G$ must be injective. Since G is either A_4 or S_4, this means $|G_\times| \leq |S_4| = 24$. But the map $\pi' \colon G_\times \to G'$ is surjective, and G' is either $A_{Q'}$ or $S_{Q'}$ for $|Q'| \geq 5$. This means $|G_\times| \geq |A_5| = 60$. So we have $60 \leq 24$, which is a contradiction. Thus $C\pi'$ cannot be trivial. So $C\pi'$ is a non-trivial normal subgroup of G', and thus it is primitive, since G' is $S_{Q'}$ or $A_{Q'}$ and $|Q'| \geq 5$. Thus the arguments in the proof of Lemma 4.6.5 apply and $\mathcal{A}' \times \mathcal{A}$ is uniformly boolean minimal.

(2c): The results from permutation group theory that we use for this case are somewhat more advanced. We will need the fact that a 2-transitive group has a unique minimal normal subgroup; this follows from two theorems in [28] (Theorem 4.1B and Theorem 4.3B), or alternatively is stated as a single result in [21] (Proposition 5.2). The other fact we need is that if the socle of a 2-transitive group is non-abelian, then the socle is necessarily primitive [28, Theorem 7.2E].

Now, suppose G is 2-transitive. The socle of G is the subgroup generated by all the minimal normal subgroups; but G has a unique minimal normal subgroup N, so the socle of G is just equal to N. If N is abelian, then G is of affine type. Thus we may assume the socle N is non-abelian; then it follows that N is primitive. Since N is the unique minimal normal subgroup of G, every non-trivial normal subgroup of G contains N, and thus is primitive. Similarly, every non-trivial normal subgroup of G' is primitive. It follows that

Lemma 4.6.5 applies. □

4.7 Similar DFAs

In this subsection, we briefly consider what happens when the DFAs $\mathcal{A}'$ and $\mathcal{A}$ are similar. We have not investigated this case very deeply. In some ways, it seems much more difficult than the dissimilar case. Particularly, most of our results rely on the projections of various kinds of row and column stabilizers being transitive or primitive. For similar DFAs, the projections of the full row and column stabilizers $R\pi$ and $C\pi'$ are both trivial. Hence there is no guarantee that other types of stabilizers such as $C_q\pi'$ or $R_{p',q'}\pi$ have useful properties, or even that they are non-trivial, and so we cannot necessarily use these groups to our advantage.

On the other hand, similarity imposes the very strong condition that the groups G', G and $G_\times$ are all isomorphic. It may be possible to exploit this to prove some interesting things in the similar case.

It is not difficult to prove that if $\mathcal{A}'$ and $\mathcal{A}$ are isomorphic as DFAs, then $G_\times$ is necessarily intransitive, so the case of isomorphic similar DFAs is uninteresting for our purposes. We give two examples demonstrating what can happen with non-isomorphic similar DFAs.

Example 4.7.1. Recall that in Example 4.6.3, we constructed two similar DFAs that are of different sizes (and hence are non-isomorphic) and have primitive transition groups. DFA $\mathcal{A}'$ has states $\{1, \dots, 5\}$ and actions $a' = (3, 4)$ and $b' = (1, 2, 3)(4, 5)$. DFA $\mathcal{A}$ has states $\{1, \dots, 10\}$ and actions $a = (1, 2)(6, 8)(7, 9)$ and $b = (2, 6, 4, 5, 3, 7)(8, 10, 9)$. We have verified computationally that $\mathcal{A}' \times \mathcal{A}$ has an intransitive transition group, and so is not accessible. Thus even if non-isomorphic similar DFAs have primitive transition groups, their direct product might have an intransitive transition group (compare this with Theorem 4.6.6 for dissimilar DFAs).

Note that in Example 4.6.3, we also showed that by simply swapping the roles of a and b in $\mathcal{A}$, the two DFAs $\mathcal{A}'$ and $\mathcal{A}$ become dissimilar. Furthermore, $\mathcal{A}'$ has the symmetric group S_5 as its transition group. Thus by Theorem 4.6.6, the two DFAs actually become uniformly boolean minimal if we swap the roles of a and b. ■

Example 4.7.2. There is a primitive subgroup of S_6 that is isomorphic to S_5. Using GAP, we can obtain an explicit isomorphism between S_5 and this subgroup, just as we did in Example 4.6.3.

```
gap> IsomorphismGroups(SymmetricGroup(5),PrimitiveGroup(6,2));
[ (3,4), (1,2,3)(4,5) ] -> [ (1,2)(3,4)(5,6), (1,2,3,5,4,6) ]
```

We then use this isomorphism to construct similar DFAs: DFA $\mathcal{A}'$ has states $\{1, \ldots, 5\}$ and actions $a' = (3, 4)$ and $b' = (1, 2, 3)(4, 5)$. DFA $\mathcal{A}$ has states $\{1, \ldots, 6\}$ and actions $a = (1, 2)(3, 4)(5, 6)$ and $b = (1, 2, 3, 4, 5, 6)$. Unlike the DFAs of Example 4.7.1, here we verified computationally that $\mathcal{A}' \times \mathcal{A}$ actually has a transitive transition group. Hence a direct product of non-isomorphic similar DFAs with primitive transition groups may or may not be accessible.

Note that $\mathcal{A}' \times \mathcal{A}$ is not uniformly boolean minimal. For example, we have verified computationally that $(\mathcal{A}' \times \mathcal{A})(X)$ is not minimal for $X = \{1\} \times \{1, 3, 5\}$. We have not found an example of two similar DFAs that are uniformly boolean minimal, but we also have not proved that no such example exists. ■

4.8 Beyond Permutation DFAs

Our next goal is to show that the results we have obtained in this chapter apply to a larger class of DFAs than just permutation DFAs. It is sufficient that the transition monoids of the DFAs $\mathcal{A}'$ and $\mathcal{A}$ merely *contain* permutation groups that satisfy the hypotheses of our results.

A *restriction* of $\mathcal{A} = (Q, \Sigma, T, 1, F)$ is a DFA $\mathcal{R} = (Q^{\mathcal{R}}, \Sigma^{\mathcal{R}}, T^{\mathcal{R}}, i^{\mathcal{R}}, F^{\mathcal{R}})$ such that:

- $Q^{\mathcal{R}}$ is a subset of Q and $\Sigma^{\mathcal{R}}$ is a subset of Σ.
- $T^{\mathcal{R}}$ is given by the restriction of the monoid action $T \colon Q \times \Sigma^* \to Q$ to the set $Q^{\mathcal{R}} \times (\Sigma^{\mathcal{R}})^*$, and we have $qT^{\mathcal{R}}_w \in Q^{\mathcal{R}}$ for all $q \in Q^{\mathcal{R}}$ and $w \in (\Sigma^{\mathcal{R}})^*$.
- $i^{\mathcal{R}} \in Q^{\mathcal{R}}$ is an arbitrary state, and $F^{\mathcal{R}} \subseteq Q^{\mathcal{R}}$ is an arbitrary set (they do not need to be related to 1 and F in any way).

The key idea of this subsection is to take DFAs which are not permutation DFAs, and form restrictions that are permutation DFAs. We can then apply our previous results to the restrictions. Since restrictions share the transition structure of the original DFAs, we can usually conclude something non-trivial about the original DFAs.

The following proposition can help extend results about restrictions to results about the original DFAs.

Proposition 4.8.1. *Let $\mathcal{R}$ be a restriction of $\mathcal{A}$. Let $p, q \in Q^{\mathcal{R}}$.*

1. *If q is reachable from p in $\mathcal{R}$, then q is reachable from p in $\mathcal{A}$.*

2. *Suppose $F = F^{\mathcal{R}} \cup X$ for some set $X \subseteq Q \setminus Q^{\mathcal{R}}$. If p and q are distinguishable under $F^{\mathcal{R}}$ in $\mathcal{R}$, then they are distinguishable under F in $\mathcal{A}$.*

Proof. (1): If q is reachable from p in $\mathcal{R}$, then there exists $w \in (\Sigma^{\mathcal{R}})^*$ such that $pT_w^{\mathcal{R}} = q$. But $T^{\mathcal{R}}$ is a restriction of T, so we have $pT_w = q$. Thus q is reachable from p in $\mathcal{A}$.

(2): Choose a word $w \in (\Sigma^{\mathcal{R}})^*$ that distinguishes p and q under $F^{\mathcal{R}}$ in $\mathcal{R}$. We claim that w distinguishes p and q under F in $\mathcal{A}$. We have $pT_w \in F \iff pT_w^{\mathcal{R}} \in F$, since $p \in Q^{\mathcal{R}}$, $w \in (\Sigma^{\mathcal{R}})^*$, and $T^{\mathcal{R}}$ is the restriction of T to $Q^{\mathcal{R}} \times (\Sigma^{\mathcal{R}})^*$. But $pT_w^{\mathcal{R}} \in Q^{\mathcal{R}}$, so $pT_w^{\mathcal{R}} \in F \iff pT_w^{\mathcal{R}} \in F \cap Q^{\mathcal{R}} = F^{\mathcal{R}}$. Thus we have $pT_w \in F \iff pT_w^{\mathcal{R}} \in F^{\mathcal{R}}$. Similarly, $qT_w \in F \iff qT_w^{\mathcal{R}} \in F^{\mathcal{R}}$. Since w distinguishes p and q under $F^{\mathcal{R}}$, it follows that

$$pT_w \in F \iff pT_w^{\mathcal{R}} \in F^{\mathcal{R}} \iff qT_w^{\mathcal{R}} \notin F^{\mathcal{R}} \iff qT_w \notin F,$$

as required. □ □

We give two examples of using DFA restrictions in conjunction with our previous results. Recall that for a set X and $i, j \in X$, the notation $(i \to j)$ denotes the transformation of X that maps i to j and fixes all other elements of X.

Example 4.8.2. Fix integers $m, n \geq 3$. Let $\mathcal{A}'$ have states $\{1, \dots, m\}$ and actions $a' = (1, \dots, m)$, $b' = (1, 2)$, and $c' = (m \to 1)$. Let $\mathcal{A}$ have states $\{1, \dots, n\}$ and actions $a = (1, 2)$, $b = (1, \dots, n)$, and $c = (n \to 1)$. It was proved in [6] that if $\mathcal{A}'$ has final state set $F' = \{m\}$ and $\mathcal{A}'$ has final state set $F = \{n\}$, then $L' \circ L$ has maximal state complexity mn for $\circ \in \{\cup, \cap, \setminus, \oplus\}$. One can then show using De Morgan's laws that (L', L) has maximal boolean complexity.

We can give an alternate proof that (L', L) has maximal boolean complexity using Proposition 4.8.1 in conjunction with Theorem 4.4.6 and Theorem 4.6.6. We also must recall the following group-theoretic facts:

- $\{a', b'\}$ generates S_m and $\{a, b\}$ generates S_n.

- S_m and S_n are primitive.

Let the restriction $\mathcal{R}'$ be obtained by restricting the alphabet of $\mathcal{A}'$ to $\{a, b\}$ and leaving everything else the same. Similarly, let $\mathcal{R}$ be obtained by restricting the alphabet of $\mathcal{A}$ to $\{a, b\}$. Then $\mathcal{R}'$ and $\mathcal{R}$ are permutation DFAs, and their transition groups are S_m and S_n respectively. Since S_m and S_n are primitive, it follows from Theorem 4.6.6 that $\mathcal{R}' \times \mathcal{R}$ is accessible. Let $L_{\mathcal{R}'}$ be the language recognized by $\mathcal{R}'$, and let $L_{\mathcal{R}}$ be the language recognized by $\mathcal{R}$. Then since $|F'| = 1$ and $|F| = 1$, by Theorem 4.4.6, the pair $(L_{\mathcal{R}'}, L_{\mathcal{R}})$ has maximal boolean complexity. Thus $(\mathcal{R}' \times \mathcal{R})(F' \circ F)$ is minimal for all proper binary boolean operations $\circ$. In particular:

- All states of $\mathcal{R}' \times \mathcal{R}$ are reachable.
- All states of $\mathcal{R}' \times \mathcal{R}$ are distinguishable under $F' \circ F$.

But the states and final states of $\mathcal{R}'$ are the same as the states and final states of $\mathcal{A}'$, and similarly for $\mathcal{R}$ and $\mathcal{A}$. Thus by Proposition 4.8.1:

- All states of $\mathcal{A}' \times \mathcal{A}$ are reachable.
- All states of $\mathcal{A}' \times \mathcal{A}$ are distinguishable under $F' \circ F$.

Thus $(\mathcal{A}' \times \mathcal{A})(F' \circ F)$ is minimal for all proper binary boolean operations $\circ$. Thus (L', L) has maximal boolean complexity, as required.

In fact, we can prove that if $(m, n) \notin \{(3, 4), (4, 3), (4, 4)\}$, then $\mathcal{A}' \times \mathcal{A}$ is uniformly boolean minimal. Since the transition groups of $\mathcal{R}'$ and $\mathcal{R}$ are symmetric groups, by Theorem 4.6.6, the DFA $\mathcal{R}' \times \mathcal{R}$ is uniformly boolean minimal. Then using Proposition 4.8.1 as before, we can conclude that $\mathcal{A}' \times \mathcal{A}$ is uniformly boolean minimal. ■

Example 4.8.3. Fix integers $m, n \geq 4$. Let $\mathcal{A}'$ have states $\{1, \dots, m\}$, final state set $\{m\}$, and actions $a' = (1, \dots, m-1)$, $b' = (2, \dots, m-1)$, and $c' = (m-1 \to 1)$. Let $\mathcal{A}$ have states $\{1, \dots, n\}$, final state set $\{n\}$, and actions $a = (2, \dots, n-1)$, $b = (1, \dots, n-1)$, and $c = (n-1 \to 1)$. The languages L and L' are *right ideals*, that is, languages R which satisfy the equation $R = R\Sigma^*$. It was proved in [10] that:

- $L' \cap L$ and $L' \oplus L$ have state complexity mn.
- $L' \setminus L$ has state complexity $mn - (m-1)$, and the state complexity of a difference of right ideals of state complexities m and n cannot exceed this value.
- $L' \cup L$ has state complexity $mn - (m+n-2)$, and the state complexity of an intersection of right ideals of state complexities m and n cannot exceed this value.

Hence $L' \circ L$ has maximal state complexity *relative to the class of right ideals* for $\circ \in \{\cup, \cap, \setminus, \oplus\}$. It can then be shown using De Morgan's laws that $L' \circ L$ has maximal state complexity (relative to right ideals) for all proper binary boolean operations.

We cannot obtain this result as a direct consequence of our results, but we can significantly reduce the work needed to prove it. Let $\mathcal{R}'$ be the restriction of $\mathcal{A}'$ with state set $\{1, \dots, m-1\}$ and alphabet $\{a, b\}$. Similarly, let $\mathcal{R}$ be the restriction of $\mathcal{A}$ with state set $\{1, \dots, n-1\}$ and alphabet $\{a, b\}$. Then $\mathcal{R}'$ and $\mathcal{R}$ are permutation DFAs. Furthermore, the transition group of $\mathcal{R}'$ is S_{m-1}, since $b^{m-3}a = (1, 2)$ and $(1, \dots, m-1)$ and $(1, 2)$ generate S_{m-1}. Similarly, the transition group of $\mathcal{R}$ is S_{n-1}. Thus by Theorem 4.6.6, the pair $(\mathcal{R}', \mathcal{R})$ is uniformly boolean minimal unless $(m-1, n-1) \in \{(3, 4), (4, 3), (4, 4)\}$.

In particular, if we take $F^{\mathcal{R}'} = \{m-1\}$ and $F^{\mathcal{R}} = \{n-1\}$, then $(\mathcal{R}' \times \mathcal{R})(F^{\mathcal{R}'} \circ F^{\mathcal{R}})$ is minimal. Thus every pair of states in the set $S = \{1, \dots, m-1\} \times \{1, \dots, n-1\}$ is distinguishable under $F^{\mathcal{R}'} \circ F^{\mathcal{R}}$. We claim that all states in S are pairwise distinguishable under $F' \circ F$. To see this, suppose that $w \in \{a, b\}^*$ distinguishes (p', p) and (q', q) under $F^{\mathcal{R}'} \circ F^{\mathcal{R}}$. We will show that wc distinguishes (p', p) and (q', q) under $F' \circ F$. To see this, recall that $F' \circ F = \{(q', q) \in Q' \times Q : q'\chi_{F'} \circ q\chi_F = 1\}$, where χ_S is the characteristic function of the set S that outputs 1 for elements in S and 0 for elements outside of S. For $i' \in Q'$, we have $i'w'c' = m$ if and only if $i'w' = m-1$, and for $i \in Q$ we have $iwc = n$ if and only if $iw = n-1$. Thus:

$$\begin{aligned}
(p'w'c', pwc) \in F' \circ F &\iff p'w'c'\chi_{\{m\}} \circ pwc\chi_{\{n\}} = 1 \\
&\iff p'w'\chi_{\{m-1\}} \circ pw\chi_{\{n-1\}} = 1 \\
&\iff (p'w', pw) \in F^{\mathcal{R}'} \circ F^{\mathcal{R}} \\
&\iff (q'w', qw) \notin F^{\mathcal{R}'} \circ F^{\mathcal{R}} \\
&\iff q'w'\chi_{\{m-1\}} \circ qw\chi_{\{n-1\}} = 0 \\
&\iff q'w'c'\chi_{\{m\}} \circ qwc\chi_{\{n\}} = 0 \\
&\iff (q'w'c', qwc) \notin F' \circ F.
\end{aligned}$$

Thus all states in S are pairwise distinguishable under $F' \circ F$. It remains to consider the states in $Q' \times Q \setminus S$, and show that these states are distinguishable from each other, as well as from states in S. This is far from trivial to show, but we have significantly reduced the size of the problem by using Theorem 4.6.6.

The authors of [10] used a similar proof technique to the above, except after restricting the DFAs, they applied the result of Bell, Brzozowski, Moreira and Reis [2, Theorem 1] that served as an inspiration for the work in this chapter. Theorem 4.6.6 essentially generalizes the result of Bell et al. (it is not quite a true generalization, since Bell et al. handle some additional small cases that we ignore). ■

4.9 An Affine Group Construction

We now construct an infinite family of pairs of dissimilar permutation DFAs which have 2-transitive transition groups of affine type, and are not uniformly boolean minimal. The details of the construction require some knowledge of finite fields; the reader may wish to review the material in Section 2.3.4 before proceeding.

We first describe the DFA construction in full generality, and then use the construction to produce an explicit pair of 8-state DFAs.

Example 4.9.1. For $k \geq 1$, let $\mathbb{F}_{2^k}$ denote the finite field of order 2^k. For $\alpha, \beta, \xi \in \mathbb{F}_{2^k}$ with $\alpha \neq 0$, define $t_{\alpha,\beta} \colon \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$ to be the map $\xi \mapsto \alpha\xi + \beta$. The set of all such maps forms a group of permutations of $\mathbb{F}_{2^k}$, which is called the 1-dimensional *affine group* on $\mathbb{F}_{2^k}$ and is denoted $AGL(1, 2^k)$. Multiplication (that is, composition of maps) in the affine group is given by the rule

$$t_{\alpha,\beta} t_{\gamma,\xi} = t_{\alpha\gamma,\beta\gamma+\xi}.$$

As one would expect, the affine group is indeed of "affine type"; it is proved in [28, Chapter 4] that the affine group has an abelian socle. It is also 2-transitive; a routine calculation shows that for $\alpha \neq \beta$ and $\gamma \neq \xi$, if we set $x = (\xi - \gamma)(\beta - \alpha)^{-1}$, $y = \gamma - x\alpha$, and $t = t_{x,y}$, then $(\alpha t, \beta t) = (\gamma, \xi)$.

Since the multiplicative group of a finite field is cyclic, let g be a generator for the multiplicative group of $\mathbb{F}_{2^k}$. We claim that the elements $t_{g,0}$ and $t_{1,1}$ generate $AGL(1, 2^k)$ (where 0 and 1 are respectively the additive and multiplicative identities of $\mathbb{F}_{2^k}$). Indeed, we can write every element of the form $t_{\alpha,0}$ as $(t_{g,0})^n = t_{g^n,0}$ for some n. Then $t_{\alpha^{-1},0} t_{1,1} t_{\alpha,0} = t_{\alpha^{-1},1} t_{\alpha,0} = t_{1,\alpha}$. Then $t_{1,\beta\alpha^{-1}} t_{\alpha,0} = t_{\alpha,\beta}$.

An element of $AGL(1, 2^k)$ of the form $t_{1,\beta}$ for some $\beta \in \mathbb{F}_{2^k}$ is called a *translation*. The translations form a subgroup of $AGL(1, 2^k)$, which we call T. We claim that the subgroup of translations T is imprimitive and contains a block B of size 2^{k-1}.

For $k = 1$, just take $B = \{0\}$. For $k > 1$, recall that there is a basis for $\mathbb{F}_{2^k}$ over $\mathbb{F}_2$ with k elements. Let $\{b_1, \ldots, b_k\}$ be such a basis. Define $B = \{a_1 b_1 + \cdots + a_{k-1} b_{k-1} : a_1, \ldots, a_{k-1} \in \mathbb{F}_2\}$. We claim that B has size 2^{k-1}. It suffices to show that there is a one-to-one correspondence between tuples $(a_1, \ldots, a_{k-1}) \in \mathbb{F}_2^{k-1}$ and elements of B. Suppose for a contradiction that this is not the case, and choose two distinct tuples $(a_1, \ldots, a_{k-1})$

and $(c_1, \dots, c_{k-1})$ such that $a_1b_1 + \cdots + a_{k-1}b_{k-1} = c_1b_1 + \cdots + c_{k-1}b_{k-1}$. Then $(a_1 - c_1)b_1 + \cdots + (a_{k-1} - c_{k-1})b_{k-1} + 0b_k = 0$, and $a_i - c_i \neq 0$ for at least one i, which implies $\{b_1, \dots, b_k\}$ is not linearly independent. Thus B has size 2^{k-1}.

Now, we prove that B is a block for T. First observe that B is closed under addition and subtraction. Then consider $Bt_{1,\beta} = \{\alpha + \beta : \alpha \in B\}$ for $\beta \in \mathbb{F}_{2^k}$.

- If $\beta \in B$, then $\alpha + \beta \in B$ for all $\alpha \in B$, since B is closed under addition; thus $Bt_{1,\beta} = B$.
- If $\beta \notin B$, then for all $\alpha \in B$ we have $\alpha + \beta \notin B$. Indeed, if $\alpha + \beta$ were in B then $(\alpha + \beta) - \alpha = \beta$ would be in B, since B is closed under subtraction. Thus if $\beta \notin B$, then $Bt_{1,\beta} \cap B = \emptyset$.

And so, we see that B is indeed a block for T.

Consider the subgroup A of $AGL(1, 2^k) \times AGL(1, 2^k)$ generated by the elements $a = (t_{g,0}, t_{g,0})$, $b = (t_{1,1}, t_{1,0})$ and $c = (t_{1,0}, t_{1,1})$. We claim that every element $(t_{\alpha,\beta}, t_{\gamma,\xi})$ of A has the property that $\alpha = \gamma$. For simplicity, we will call elements with this property *balanced.*

Certainly the elements a, b and c are balanced, and so is the identity element $(t_{1,0}, t_{1,0})$. We will show that multiplying a balanced element on the right by a, b or c results in a balanced element. Indeed, first observe that $t_{\alpha,\beta}t_{g,0} = t_{\alpha g,\beta g}$ and $t_{\alpha,\beta}t_{1,1} = t_{\alpha,\beta+1}$. Thus if we take an arbitrary balanced element $(t_{\alpha,\beta}, t_{\alpha,\gamma})$, then we have:

$$(t_{\alpha,\beta}, t_{\alpha,\gamma})a = (t_{\alpha,\beta}t_{g,0}, t_{\alpha,\gamma}t_{g,0}) = (t_{\alpha g,\beta g}, t_{\alpha g,\gamma g}).$$

$$(t_{\alpha,\beta}, t_{\alpha,\gamma})b = (t_{\alpha,\beta}t_{1,1}, t_{\alpha,\gamma}t_{1,0}) = (t_{\alpha,\beta+1}, t_{\alpha,\gamma}).$$

$$(t_{\alpha,\beta}, t_{\alpha,\gamma})c = (t_{\alpha,\beta}t_{1,0}, t_{\alpha,\gamma}t_{1,1}) = (t_{\alpha,\beta}, t_{\alpha,\gamma+1}).$$

Since a, b and c and the identity are balanced, and multiplying a balanced element by a, b or c results in a balanced element, it follows the group $\langle a, b, c \rangle = A$ consists solely of balanced elements.

Next, let $\overline{B} = \mathbb{F}_{2^k} \setminus B$ and consider the set $X = (B \times B) \cup (\overline{B} \times \overline{B}) \subseteq \mathbb{F}_{2^k} \times \mathbb{F}_{2^k}$. Notice that $(0,0), (1,1) \in \mathbb{F}_{2^k} \times \mathbb{F}_{2^k}$ both lie in X. We claim that elements of A cannot distinguish $(0,0)$ and $(1,1)$ under X, that is, for all $g \in A$ we have $(0,0)g \in X \iff (1,1)g \in X$.

- To see this, consider an arbitrary element $g = (t_{\alpha,\beta}, t_{\alpha,\gamma})$ of A.
- We have $0t_{\alpha,\beta} = \beta$ and $1t_{\alpha,\beta} = \alpha + \beta$. It follows that $(0,0)g = (\beta, \gamma)$ and $(1,1)g = (\alpha + \beta, \alpha + \gamma) = (\beta t_{1,\alpha}, \gamma t_{1,\alpha})$.

- Since B is a block for the subgroup of translations T, we either have $Bt_{1,\alpha} = B$ or $Bt_{1,\alpha} \cap B = \emptyset$.
- But B has size 2^{k-1}, which is exactly half the size of $\mathbb{F}_{2^k}$, so if $Bt_{1,\alpha} \cap B = \emptyset$ then $Bt_{1,\alpha} = \overline{B}$.

It follows that if (β, γ) is in $B \times B$ or $\overline{B} \times \overline{B}$ (that is, (β, γ) is in X) then $(\beta t_{1,\alpha}, \gamma t_{1,\alpha})$ is also in $B \times B$ or $\overline{B} \times \overline{B}$ (and thus in X).

Likewise, if (β, γ) is not in X, then it is either in $B \times \overline{B}$ or $\overline{B} \times B$, and it follows $(\beta t_{1,\alpha}, \gamma t_{1,\alpha})$ is not in X. Thus $(0,0)g \in X \iff (1,1)g \in X$ for all $g \in A$.

Finally, for each $k \geq 1$, we construct a pair of DFAs over the alphabet $\{a, b, c\}$ with 2^k states each, which both have $AGL(1, 2^k)$ as their transition group, but do not have a uniformly boolean minimal direct product.

We define a DFA $\mathcal{A}'$ as follows:

- The state set is $\mathbb{F}_{2^k}$, the initial state is 0, and the final state set is B.
- The actions are $a' = t_{g,0}$, $b' = t_{1,1}$ and $c' = t_{1,0}$.

We define $\mathcal{A}$ in the same way as $\mathcal{A}'$, except the roles of b and c are swapped:

- The actions are $a = t_{g,0}$, $b = t_{1,0}$ and $c = t_{1,1}$.

Since $t_{g,0}$ and $t_{1,1}$ generate $AGL(1, 2^k)$, it is clear that both DFAs have $AGL(1, 2^k)$ as their transition group.

Now consider $\mathcal{A}' \times \mathcal{A}$. Let $\circ$ be the "complement of symmetric difference" operation, so that $B \circ B = (B \times B) \cup (\overline{B} \times \overline{B})$. Observe that the transition group of $\mathcal{A}' \times \mathcal{A}$ is simply the group A. Thus the states $(0,0)$ and $(1,1)$ of $\mathcal{A}' \times \mathcal{A}$ are not distinguishable under $B \circ B$. Hence $(\mathcal{A}' \times \mathcal{A})(B \circ B)$ is not minimal, and it follows that $\mathcal{A}' \times \mathcal{A}$ is not uniformly boolean minimal. ■

Example 4.9.2. We now carry out the construction of Example 4.9.1 for $k = 3$, $2^k = 8$. As before, construct $\mathbb{F}_8$ as $\mathbb{F}_3[x]/\langle x^3 + x + 1 \rangle$. Recall from Example 2.3.12 that x is a generator of the multiplicative group of $\mathbb{F}_8$, and in particular we have:

$$x^3 = x + 1, \quad x^4 = x^2 + x, \quad x^5 = x^2 + x + 1,$$
$$x^6 = x^2 + 1, \quad x^7 = 1.$$

Using these calculations, we can explicitly write out the permutations $t_{x,0}$ and $t_{1,1}$ in cycle notation:

$$t_{x,0} = (x, x^2, x^3, x^4, x^5, x^6, x^7) = (x, x^2, x+1, x^2+x, x^2+x+1, x^2+1, 1).$$

$$t_{1,1} = (0,1)(x, x+1)(x^2, x^2+1)(x^2+x, x^2+x+1) = (0, x^7)(x, x^3)(x^2, x^6)(x^4, x^5).$$

Finally, we need a block B for the subgroup of translations of $AGL(1,8)$. The construction tells us to pick two elements b_1 and b_2 from a basis of $\mathbb{F}_8$ over $\mathbb{F}_2$, and let B be the set $\{a_1 b_1 + a_2 b_2 : a_1, a_2 \in \mathbb{F}_2\}$. If we take 1 and x, we get $B = \{0, 1, x, x+1\}$.

We now have all the information we need to construct $\mathcal{A}'$ and $\mathcal{A}$. A state diagram for $\mathcal{A}'$ is shown in Figure 4.9.1, with the self-loops on c omitted. One may verify computationally that $\mathcal{A}' \times \mathcal{A}$ is not minimal when it is assigned the final state set $(B \times B) \cup (\overline{B} \times \overline{B})$. ■

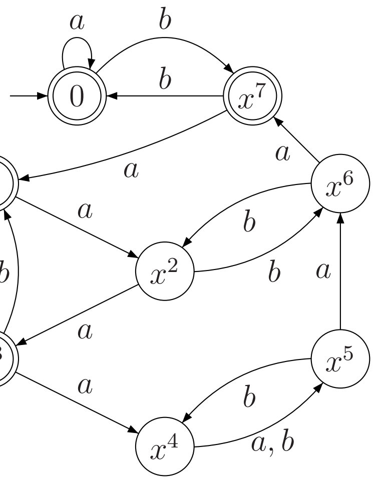


Figure 4.9.1: DFA $\mathcal{A}'$ of Example 4.9.2. Each state also has a self-loop on letter c; these transitions are omitted from the diagram. The final state set is $B = \{0, x^7 = 1, x, x^3 = x+1\}$, the block of the subgroup of translations of $AGL(1,8)$ that was found in Example 4.9.2.

For $k = 1$, we get DFAs $\mathcal{A}'$, $\mathcal{A}$ and $\mathcal{A}' \times \mathcal{A}$ that are isomorphic to the DFAs of Example 4.3.4. For $k = 2$, it happens that $AGL(1,4)$ is isomorphic to the alternating group A_4. Hence the $k = 2$ case gives an example of dissimilar DFAs that are not uniformly boolean minimal, and have alternating groups as their transition groups. As Theorem 4.6.6 shows, this example does not generalize to alternating groups of higher degree. The DFA $\mathcal{A}$ of

Example 4.2.6 is isomorphic to the DFA $\mathcal{A}'$ produced by the construction of Example 4.9.1 with $k = 2$.

The construction of Example 4.9.1 also shows that condition (1) of Proposition 4.5.4 is not sufficient for uniform boolean minimality.

Example 4.9.3. The DFAs $\mathcal{A}'$ and $\mathcal{A}$ constructed in Example 4.9.1 are not uniformly boolean minimal. However, we claim the subgroups $C_\alpha\pi' \leq G'$ and $R_\alpha\pi \leq G$ are primitive for all $\alpha \in \mathbb{F}_{2^k}$, and thus $\mathcal{A}'$ and $\mathcal{A}$ meet condition (1) of Proposition 4.5.4. In fact, the groups $C_\alpha\pi'$ and $R_\alpha\pi$ are equal to $AGL(1, 2^k)$. First, we show that $C_0\pi'$ and $R_0\pi$ are equal to $AGL(1, 2^k)$. Consider $(a', a) = (t_{g,0}, t_{g,0})$.

- Since $t_{g,0}$ fixes 0, it follows that $(a', a) \in C_0$ and $(a', a) \in R_0$.
  Thus $a' = t_{g,0} \in C_0\pi'$ and $a = t_{g,0} \in R_0\pi$.
- Since $(b', b) = (t_{1,1}, t_{1,0})$ and $b = t_{1,0}$ fixes 0, we see that $(b', b) \in C_0$.
  Thus $b' = t_{1,1} \in C_0\pi'$.
- Similarly, since $(c', c) = (t_{1,0}, t_{1,1})$, we see that $(c', c) \in R_0$.
  Thus $c = t_{1,1} \in R_0\pi$.

Since $t_{g,0}$ and $t_{1,1}$ generate $AGL(1, 2^k)$, and these elements are in $C_0\pi'$ and $R_0\pi$, it follows $C_0\pi'$ and $R_0\pi$ are equal to $AGL(1, 2^k)$ and thus are primitive.

To show that $R_\alpha\pi'$ and $C_\alpha\pi$ are primitive for all $\alpha \neq 0$, we prove a general fact about single row and column stabilizers: if $R_{p'}\pi \leq G$ is primitive for some $p' \in Q'$ and G' is transitive, then $R_{q'}\pi$ is primitive for all $q' \in Q'$ (and similarly for single column stabilizers).

- To see this, choose $w' \in G'$ such that $p'w' = q'$.
- Let B be a block for $R_{q'}\pi$. We claim Bw^{-1} is a block for $R_{p'}\pi$.
- To see this, choose $x \in R_{p'}\pi$ and consider $Bw^{-1}x \cap Bw^{-1}$.
- If $Bw^{-1}x \cap Bw^{-1} \neq \emptyset$, then $Bw^{-1}xw \cap B \neq \emptyset$.
- Now, for all $x \in R_{p'}\pi$, we have $p'x' = p'$ by definition.
- It follows $q'(w')^{-1}x'w' = p'x'w' = p'w' = q'$. Since $(w')^{-1}x'w'$ fixes q', we have $w^{-1}xw \in R_{q'}\pi$.
- Since B is a block for $R_{q'}\pi$ and $Bw^{-1}xw \cap B \neq \emptyset$, we have $Bw^{-1}xw = B$.

- Hence $Bw^{-1}x = Bw^{-1}$, which proves Bw^{-1} is a block for $R_{p'}\pi$.

It follows that if B is a block for $R_{q'}\pi$, it must be a trivial block; otherwise Bw^{-1} is a non-trivial block for the primitive group $R_{p'}\pi$, which is a contradiction.

Thus $R_\alpha\pi$ is primitive for all $\alpha \in \mathbb{F}_{2^k}$, and symmetrically we see that $C_\alpha\pi'$ is primitive for all $\alpha \in \mathbb{F}_{2^k}$. This shows that $\mathcal{A}'$ and $\mathcal{A}$ satisfy condition (1) of Proposition 4.5.4, yet $\mathcal{A}' \times \mathcal{A}$ is not uniformly boolean minimal. ■

We can also use DFAs derived from affine groups to show that the conditions of Lemma 4.5.2 are not necessary for uniform boolean minimality.

Example 4.9.4. As in Example 4.9.2, we define two DFAs $\mathcal{A}'$ and $\mathcal{A}$ that have transition group $AGL(1, 8)$. However, this time the direct product of the DFAs will be uniformly boolean minimal. We define $\mathcal{A}'$ as follows (leaving the final state set unspecified).

- The state set is $\mathbb{F}_8$, constructed as in Example 4.9.2, and the initial state is 0.
- The actions are $a' = t_{g,0}$ and $b' = t_{1,1}$.

Define $\mathcal{A}$ to have the same states as $\mathcal{A}$ and actions $a = t_{g,0}^{-1}$, $b = t_{1,1}$.

Since $\mathcal{A}'$ and $\mathcal{A}$ only have 8 states each, we were able to verify computationally that $\mathcal{A}' \times \mathcal{A}$ is uniformly boolean minimal by a brute force approach. We computed $\mathcal{A}' \times \mathcal{A}$, and for each pair of sets $\emptyset \subsetneq S' \subsetneq Q'$ and $\emptyset \subsetneq S \subsetneq Q$, we checked the minimality of $(\mathcal{A}' \times \mathcal{A})(X)$ for all (S', S)-compatible sets X.

We have also verified computationally that $C_{0,1}\pi'$ and $R_{0,1}\pi$ are imprimitive, and hence $\mathcal{A}'$ and $\mathcal{A}$ do not meet the conditions of Lemma 4.5.2. We verified this by using the cycle notation representation we found for $t_{g,0}$ and $t_{1,1}$ to explicitly construct the transition group $G_\times$ of $\mathcal{A}' \times \mathcal{A}$ in GAP. Then we computed the setwise stabilizer $C_{0,1}$ of $\{0, 1\} \times \mathbb{F}_8$ (the columns indexed by 0 and 1), and the setwise stabilizer $R_{0,1}$ of $\mathbb{F}_8 \times \{0, 1\}$ (the rows indexed by 0 and 1). Next, we computed $C_{0,1}\pi' \leq G'$ and $R_{0,1}\pi \leq G$. These groups turned out to both be equal to T, the subgroup of translations in $AGL(1, 8)$. We saw earlier than T is imprimitive. ■

We suspect that if this construction is generalized to $AGL(1, 2^k)$, the resulting DFAs will have the same property of being uniformly boolean minimal but having $C_{0,1}\pi'$ and $R_{0,1}\pi$ imprimitive. However, we were unable to prove this.

4.10 Summary

We summarize the major results proved in this chapter.

Theorem 4.4.6 gives necessary and sufficient conditions for a pair of regular languages (L', L) to have maximal boolean complexity, with the requirement that these languages are recognized by permutation DFAs $\mathcal{A}'$ and $\mathcal{A}$ with exactly one final state. In this special case, it turns out that (L', L) is uniformly boolean minimal if and only if $\mathcal{A}' \times \mathcal{A}$ is accessible. This gives a partial characterization of witnesses for the state complexity of proper binary boolean operations.

We have several results which may help to determine whether $\mathcal{A}' \times \mathcal{A}$ is accessible. **Lemma 4.4.4** gives group-theoretic conditions for accessibility, while **Proposition 4.4.5** reduces the problem to just showing accessibility of states in one row or column of $\mathcal{A}' \times \mathcal{A}$.

Lemma 4.6.5 gives a group-theoretic condition for accessibility of $\mathcal{A}' \times \mathcal{A}$, as well as a similar condition for uniform boolean minimality. The power of this lemma is demonstrated by **Theorem 4.6.6**, which gives several classes of groups where the condition of Lemma 4.6.5 holds. If one can show that the transition group of $\mathcal{A}'$ or of $\mathcal{A}$ lies in one of these classes, one immediately gets information about $\mathcal{A}' \times \mathcal{A}$. Theorem 4.6.6 can also be used to construct examples of DFAs whose direct product is accessible or uniformly boolean minimal: one may pick a pair of groups from the classes mentioned in the corollary, and use them as the transition groups of a pair of DFAs.

Lemma 4.5.2 gives some sufficient conditions for uniform boolean minimality of permutation DFAs. The conditions are stronger than those of Lemma 4.6.5, but more difficult to verify. Unfortunately, Example 4.9.4 shows that these conditions are not necessary. Necessary and sufficient conditions for uniform boolean minimality are still unknown.

Proposition 4.5.4 gives a necessary and sufficient condition (1) for a property that is slightly weaker than uniform boolean minimality to hold (in permutation DFAs). Specifically, condition (1) of Proposition 4.5.4 does not guarantee minimality for final state sets corresponding to the symmetric difference operation or its complement.

Unfortunately, Example 4.9.3 shows that condition (1) of Proposition 4.5.4 is not sufficient for uniform boolean minimality. This means a precise characterization of uniform boolean minimality lies strictly between the conditions given by Lemma 4.5.2 and Proposition 4.5.4.

All the results mentioned above are stated for permutation DFAs, that is, DFAs whose transition monoid is a permutation group. **Section 4.8** shows how to use *restrictions of DFAs* to extend our results to certain non-permutation DFAs.

Chapter 5

Techniques for Concatenation and Kleene Star

5.1 Introduction

In the section, we examine the operations of concatenation and star, focusing on reachability of states in concatenation and star automata. We develop a new technique for proving that the maximal possible number of states is reachable in these automata, allowing us to circumvent the induction-based methods we saw in Section 2.4.5.

We first discovered our technique in the context of concatenation. To test it, we applied it to a variety of concatenation witnesses taken from the literature. The state complexity of concatenation has been studied in the class of all regular languages, as well as many subclasses. Table 5.1.1 lists some examples of subclasses that have been studied, and the state complexity of concatenation in each subclass. See the cited papers for definitions of each subclass and the derivations and proofs of each complexity. The complexities listed are restricted complexities, that is, they are computed under the assumption that both inputs to the concatenation operation share the same alphabet. Unrestricted state complexity of concatenation (where the inputs may be languages over different alphabets) is not included in the table, but our technique works for unrestricted state complexity, and we will see an example of this later in the chapter.

If the state complexity of concatenation grows exponentially with n (indicated in Table 5.1.1 by **bold** type), it is typical to use an induction argument to prove the desired number of states is reachable. It is cases like this in which our technique is most likely to be useful.

Subclass	Complexity	Subclass	Complexity
Regular [6, 18, 62, 79]	$\mathbf{(m-1)2^n + 2^{n-1}}$	Prefix-closed [14, 18]	$\mathbf{(m+1)2^{n-2}}$
Unary [67, 68, 79]	$\sim mn$ (asymptotically)	Prefix-free [18, 42, 49]	$m+n-2$
Finite unary [22, 78]	$m+n-2$	Suffix-closed [14, 16]	$mn-n+1$
Finite binary [22]	$\mathbf{(m-n+3)2^{n-2} - 1}$	Suffix-free [16, 41]	$\mathbf{(m-1)2^{n-2}+1}$
Star-free [15]	$\mathbf{(m-1)2^n + 2^{n-1}}$	Right ideal [10, 11, 18]	$\mathbf{m+2^{n-2}}$
Non-returning [9, 34]	$\mathbf{(m-1)2^{n-1}+1}$	Left ideal [10, 11, 16]	$m+n-1$

Table 5.1.1: Subclasses of regular languages and the state complexity of the concatenation operation within each subclass. **Bold** type indicates that the complexity grows exponentially in terms of n.

We selected 16 concatenation witnesses, all from subclasses in which the state complexity of concatenation is exponential in n, and tried to apply our technique to these witnesses. In many cases we were able to produce shorter and simpler proofs than the original authors, and we only found two cases in which our technique did not work or was not useful. This suggests that our technique is widely applicable and should be considered as an viable alternative to the traditional induction argument when attempting reachability proofs in concatenation automata.

After conducting this research on concatenation, we discovered that a similar technique can be applied to star. However, we have not evaluated the effectiveness of the star version of the technique in the same way. Our discussion of star will be limited to a description of the technique and one brief example.

5.2 Reachability and Construction Sets

Let $\mathcal{A} = (Q^{\mathcal{A}}, \Sigma^{\mathcal{A}}, T^{\mathcal{A}}, 1', F^{\mathcal{A}})$ and $\mathcal{B} = (Q^{\mathcal{B}}, \Sigma^{\mathcal{B}}, T^{\mathcal{B}}, 1, F^{\mathcal{B}})$ be DFAs, where we have $Q^{\mathcal{A}} = \{1', 2', \dots, m'\}$ and $Q^{\mathcal{B}} = \{1, 2, \dots, n\}$ for positive integers m and n. Let $\mathcal{C} = (Q, \Sigma, T, I, F)$ denote the concatenation DFA of $\mathcal{A}$ and $\mathcal{B}$ as defined in Section 2.4.4.

Remark. Let $p', q' \in Q^{\mathcal{A}}$, let $X, Y, Z \subseteq Q^{\mathcal{B}}$, and let $w \in \Sigma^*$. Then in $\mathcal{C}$, if $(p', X)w = (q', Y)$, then $(p', X \cup Z)w = (q', Y \cup Zw)$. Indeed, recall that the pair (p', X) stands for the set $\{p'\} \cup X$. Thus $(\{p'\} \cup X)w = \{p'w\} \cup Xw = \{q'\} \cup Y$. It follows that $p'w = q'$ and $Xw = Y$. Hence $(\{p'\} \cup X \cup Z)w = \{p'w\} \cup Xw \cup Zw = \{q'\} \cup Y \cup Zw$, which in our pair notation is $(q', Y \cup Zw)$. We will readily use this basic fact in proofs.

Before stating our main result formally, we give some motivating exposition. Fix a state $s' \in Q^{\mathcal{A}}$ and a subset B of $Q^{\mathcal{B}}$. The state s' is called the *focus state* or simply *focus*; it is often taken to be the initial state $1'$ but in general can be any state. The subset B

is called the *base*. Fix a set T with $B \subseteq T \subseteq Q^{\mathcal{B}}$, called the *target*. Our goal is to give sufficient conditions under which starting from (s', B), we can reach (s', S) for all sets S with $B \subseteq S \subseteq T$. That is, we can reach any state of the concatenation DFA $\mathcal{C}$ in which the first component is the focus and the second component lies between the base and the target.

The idea is to first assume we can reach (s', B), the state consisting of the focus and the base. Now, for $q \in Q$, define a *q-word* to be a word w such that $(s', B)w = (s', B \cup q)$. We can think of this as a word that "adds" the state q to the base B. Our next assumption is that we have a q-word for each state q in the target T. To reach a set S with $B \subseteq S \subseteq T$, we will repeatedly use q-words to add each missing element of S to the base B.

There is a problem with this idea, which we illustrate with an example. Suppose w_p is a p-word and w_q is a q-word, and we want to reach $(s', B \cup \{p, q\})$. Starting from (s', B) we may apply w_p to reach $(s', B \cup p)$. But now if we apply w_q, we reach $(s', B \cup \{pw_q, q\})$. There is no guarantee that we have $pw_q = p$, and in many cases we will not. What we should really do is find a state r such that $rw_q = p$, use an r-word to reach $(s', B \cup r)$, and then apply w_q to reach $(s', B \cup \{p, q\})$. But this idea only works if p has a preimage under w_q, which may not be the case.

We resolve this by making a technical assumption, which ensures that preimages will always exist when we attempt constructions like the above. First, define a *construction set* for the target T to be a set of words consisting of exactly one q-word for each $q \in T$. If W is a construction set for T, we write $W[q]$ for the unique q-word in W.

We say a construction set is *complete* if there is a total order $\prec$ on the target T such that for all $p, q \in T$ with $p \prec q$, the state q has at least one preimage under the unique p-word $W[p]$, and at least one of these preimages lies in T. More formally, whenever $p \prec q$, the set $qW[p]^{-1} = \{s \in Q^{\mathcal{B}} : sW[p] = q\}$ intersects T non-trivially. Our final assumption is that we have a complete construction set for T.

Note that the definition of a q-word depends not only on q, but also on s' and B. Since a construction set for T is a set of q-words, the definition of construction set also depends on s' and B. For simplicity, we omit this dependence on s' and B from the notation for q-words and construction sets.

We summarize the definitions that have just been introduced:

Definition 5.2.1. Fix a state $s' \in Q^{\mathcal{A}}$, called the *focus*, and a set $B \subseteq Q^{\mathcal{B}}$ called the *base*.

- For $q \in Q^{\mathcal{B}}$, a *q-word* is a word w such that $(s', B)w = (s', B \cup q)$.

- Given a *target* set T with $B \subseteq T \subseteq Q^{\mathcal{B}}$, a *construction set* for T is a set of words that contains exactly one q-word for each $q \in T$.
- The unique q-word in a construction set W is denoted by $W[q]$.
- A construction set for T is *complete* if there exists a total order $\prec$ on T such that for all $p, q \in T$ with $p \prec q$, we have

$$qW[p]^{-1} \cap T = \{s \in Q^{\mathcal{B}} : sW[p] = q\} \cap T \neq \emptyset.$$

Now, we state our main theorem, which gives the formal version of the construction described above.

Theorem 5.2.2. *Fix a state $s' \in Q^{\mathcal{A}}$ and sets $B \subseteq T \subseteq Q^{\mathcal{B}}$. If there is a complete construction set for T, then all states of the form (s', S) with $B \subseteq S \subseteq T$ are reachable from (s', B) in $\mathcal{C}$. In particular, if (s', B) itself is reachable, then all states (s', S) with $B \subseteq S \subseteq T$ are reachable.*

Proof. Note that if $B \subseteq S \subseteq T$, we can write $S = R \cup B$ with $R \cap B = \emptyset$ and $R \subseteq T$. Thus it suffices to show that all states of the form $(s', R \cup B)$ with $R \cap B = \emptyset$ and $R \subseteq T$ are reachable from (s', B). We proceed by induction on $|R|$. When $|R| = 0$, the only state of this form is (s', B) itself.

Now suppose every state $(s', R \cup B)$ with $R \cap B = \emptyset$, $R \subseteq T$ and $0 \leq |R| < k$ is reachable from (s', B). We want to show this also holds for $|R| = k$. Let W be a complete construction set for T and let $\prec$ be the corresponding total order on T. Let p be the minimal element of R under $\prec$. Let w be $W[p]$, the unique p-word in W. For all $q \in R \setminus p$, we have $p \prec q$ and thus qw^{-1} contains an element of T (since W is complete).

Construct sets X and Y as follows: starting with $X = \emptyset$, for each $q \in R \setminus p$, choose an element of $qw^{-1} \cap T$ and add it to X. Then set $Y = X \setminus B$. Observe that X is a subset of T of size $|R \setminus p| = k - 1$. Hence Y is a subset of T of size at most $k - 1$ such that $Y \cap B = \emptyset$. It follows by the induction hypothesis that $(s', Y \cup B)$ is reachable from (s', B). But $Y \cup B = X \cup B$, so $(s', X \cup B)$ is reachable from (s', B). By the definition of X, we have $Xw = R \setminus p$. Since w is a p-word, we have $(s', B)w = (s', B \cup p)$, and thus

$$(s', X \cup B)w = (s', Xw \cup B \cup p) = (s', (R \setminus p) \cup B \cup p) = (s', R \cup B).$$

Hence $(s', R \cup B)$ is reachable from (s', B), as required. ☐

The definition of completeness is somewhat complicated, which makes it difficult to use Theorem 5.2.2. Thus, we next prove some results giving useful sufficient conditions for a construction set to be complete. Before stating our first such result, we introduce some notation.

Definition 5.2.3. Define $\Sigma_0 = \Sigma^{\mathcal{A}} \cap \Sigma^{\mathcal{B}}$. We call Σ_0 the *shared alphabet* of $\mathcal{A}$ and $\mathcal{B}$.

The following remark shows that when $\Sigma^{\mathcal{A}} \neq \Sigma^{\mathcal{B}}$, it is important to work exclusively with the shared alphabet when looking for complete construction sets. Of course, if $\Sigma^{\mathcal{A}} = \Sigma^{\mathcal{B}}$ then the shared alphabet is just the common alphabet of both automata, and there is nothing to worry about.

Remark. A construction set for a non-empty target cannot be complete unless it is a subset of Σ_0^*. To see this, suppose W is a construction set and let $w \in W$. If w contains a letter from $\Sigma^{\mathcal{A}} \setminus \Sigma^{\mathcal{B}}$, then w is not a word over $\Sigma^{\mathcal{B}}$. Recall that if w is not a word over $\Sigma^{\mathcal{B}}$, then $T_w^{\mathcal{B}}$ is defined to be the empty relation. Thus the converse relation $(T_w^{\mathcal{B}})^{-1}$ is also empty, which means qw^{-1} is empty for all q. It follows W cannot be complete. On the other hand, suppose w contains a letter from $\Sigma^{\mathcal{B}} \setminus \Sigma^{\mathcal{A}}$. Then $(s', B)w = (\emptyset, Bw)$. Hence w is not a q-word for any q, and so w cannot be an element of a construction set, which is a contradiction. It follows that all words in a complete construction set are words over the shared alphabet Σ_0.

Lemma 5.2.4. *Fix $s' \in Q^{\mathcal{A}}$ and sets $B \subseteq T \subseteq Q^{\mathcal{B}}$. Let $x_1, \dots, x_j$ be words over Σ_0 that act as permutations on T, and let y be an arbitrary word over Σ_0. Choose $x_0 \in \{\varepsilon, x_1, \dots, x_j\}$. Define*

$$W = \{x_1, x_2, \dots, x_j\} \cup \{x_0 y, x_0 y^2, \dots, x_0 y^k\}.$$

If W is a construction set for T, then it is complete.

Proof. For $1 \leq i \leq j$, let $w_i = x_i$. For $1 \leq i \leq k$, let $w_{j+i} = x_0 y^i$. Let $\ell = j + k$. Then we have $W = \{w_1, \dots, w_\ell\}$. Let q_i be the state in T such that $(s', B)w_i = (s', B \cup q_i)$. Define an order $\prec$ on T so that $q_1 \prec q_2 \prec \cdots \prec q_\ell$. We claim this order makes W complete. Notice that $w_r = W[q_r]$, the unique q_r-word in W. Thus we must show that whenever $q_r \prec q_s$, we have $q_s w_r^{-1} \cap T \neq \emptyset$.

Suppose $r < s$ and $r \leq j$. Then $w_r = x_r$ acts as a permutation on T. Thus $q_s w_r^{-1} \cap T$ is non-empty, since $q_s \in T$.

Suppose $r < s$ and $r > j$. Since $s - r > 0$, we can write $w_s = x_0 y^{s-j} = x_0 y^{s-r} y^{r-j} = w_{j+s-r} y^{r-j}$. Thus $(s', B)w_s = (s', B \cup q_{j+s-r})y^{r-j} = (s', B \cup q_s)$. There are two possibilities: $q_{j+s-r} y^{r-j} = q_s$, or $qy^{r-j} = q_s$ for some $q \in B$.

In either case, $q_s(y^{r-j})^{-1} \cap T$ is non-empty. That is, there exists $q \in T$ such that $qy^{r-j} = q_s$. Since x_0 acts as a permutation on T, there exists $p \in T$ such that $px_0 = q$. Thus $px_0y^{r-j} = pw_r = q_s$. It follows that $q_s w_r^{-1} \cap T$ is non-empty, as required. □

Usually, we will use one of the following corollaries instead of Lemma 5.2.4 itself.

Corollary 5.2.5. *Fix $s' \in Q^{\mathcal{A}}$ and sets $B \subseteq T \subseteq Q^{\mathcal{B}}$. Let x and y be words over Σ_0 such that x acts as a permutation on T. Suppose W is one of the following sets:*

1. $\{y, y^2, \ldots, y^k\}$.
2. $\{\varepsilon, y, y^2, \ldots, y^k\}$.
3. $\{x, xy, xy^2, \ldots, xy^k\}$.
4. $\{\varepsilon, x, xy, xy^2, \ldots, xy^k\}$.

If W is a construction set for T, then it is complete.

Proof. All statements follow easily from Lemma 5.2.4:

1. Set $j = 0$.
2. Set $j = 1$ and $x_0 = x_1 = \varepsilon$.
3. Set $j = 1$ and $x_0 = x_1 = x$.
4. Set $j = 2$, $x_1 = \varepsilon$ and $x_0 = x_2 = x$. □

Corollary 5.2.6. *Fix $s' \in Q^{\mathcal{A}}$ and sets $B \subseteq T \subseteq Q^{\mathcal{B}}$. Let $W \subseteq \Sigma_0^*$ be a construction set for T.*

1. *If every word in W acts as a permutation on T, then W is complete.*
2. *If there is a word $w \in W$ such that every word in $W \setminus w$ acts as a permutation on T, then W is complete.*

Proof. Both statements follow easily from Lemma 5.2.4:

1. Set $k = 0$ in Lemma 5.2.4.

2. Set $k = 1$, $x_0 = \varepsilon$ and $y = w$ in Lemma 5.2.4. $\square$

In the special case where W contains ε, Corollary 5.2.6 admits the following generalization, which we found occasionally useful.

Lemma 5.2.7. *Fix $s' \in Q^{\mathcal{A}}$ and sets $B \subseteq T \subseteq Q^{\mathcal{B}}$. Let $W = \{\varepsilon, w_1, \dots, w_k\}$ be a construction set for T, where $w_1, \dots, w_k$ are non-empty words over Σ_0. Suppose that for every word $w \in W$, there exists a set S with $T \setminus B \subseteq S \subseteq T$ such that w acts as a permutation on S. Then W is complete.*

Proof. Write $B = \{q_1, \dots, q_j\}$. Note that ε is a q_i-word for $1 \leq i \leq j$. Thus by the definition of a construction set, ε is the unique q_i-word in W for each $q_i \in B$, that is, $W[q_i] = \varepsilon$ for $1 \leq i \leq j$. In particular, each non-empty word in W is a q-word for some $q \in T \setminus B$. For $1 \leq i \leq k$, let q_{j+i} be the state such that $(s', B)w_i = (s', B \cup q_{j+i})$. Then $T = \{q_1, \dots, q_{j+k}\}$. Note that $W[q_i] = \varepsilon$ if $1 \leq i \leq j$, and $W[q_i] = w_{i-j}$ if $j+1 \leq i \leq j+k$.

Define an order $\prec$ on T by $q_1 \prec q_2 \prec \cdots \prec q_{j+k}$. We claim this order makes W complete. Choose $q_r, q_s \in T$ with $q_r \prec q_s$; we want to show that $q_s W[q_r]^{-1} \cap T \neq \emptyset$. Suppose $q_r \in B$. Then $W[q_r] = \varepsilon$, and we have $q_s \varepsilon^{-1} \cap T$ non-empty as required. Now if $q_r \notin B$, then since $q_r \prec q_s$ we also have $q_s \notin B$. In this case, $W[q_r] = w_{r-j}$, which acts as a permutation on some superset S of $T \setminus B$. Since $q_s \in T \setminus B$, it follows that q_s has a preimage under w_{r-j}, and furthermore this preimage lies in T, since S is a subset of T. Thus $q_s w_{r-j}^{-1} \cap T \neq \emptyset$ as required. This proves that W is complete. $\square$

Note that *all words* referred to in the above lemmas and corollaries are words over Σ_0, the shared alphabet of $\mathcal{A}$ and $\mathcal{B}$. When working with automata that have different alphabets, it is important to only use words over the shared alphabet when trying to find a complete construction set.

The following "master theorem" summarizes all the results of this section.

Theorem 5.2.8. *Let $\mathcal{A} = (Q^{\mathcal{A}}, \Sigma^{\mathcal{A}}, T^{\mathcal{A}}, 1', F^{\mathcal{A}})$ and $\mathcal{B} = (Q^{\mathcal{B}}, \Sigma^{\mathcal{B}}, T^{\mathcal{B}}, 1, F^{\mathcal{B}})$ be DFAs. Let $\mathcal{C} = (Q, \Sigma, T, I, F)$ denote the concatenation DFA of $\mathcal{A}$ and $\mathcal{B}$, as defined in Section 2.4.4. Let $\Sigma_0 = \Sigma^{\mathcal{A}} \cap \Sigma^{\mathcal{B}}$.*

Fix a state $s' \in Q^{\mathcal{A}}$ and sets $B \subseteq T \subseteq Q^{\mathcal{B}}$. Suppose that for each $q \in T$, there exists a word $w_q \in \Sigma_0^*$ such that $(s', B) \xrightarrow{w_q} (s', B \cup q)$ in $\mathcal{C}$. Let $W = \{w_q : q \in T\}$. Suppose that one of the following conditions holds:

1. *There exist words $x, y \in \Sigma_0^*$, where x acts as a permutation on T, such that W can be written in one of the following forms:*

- *$W = \{y, y^2, \ldots, y^k\}$.*
- *$W = \{\varepsilon, y, y^2, \ldots, y^k\}$.*
- *$W = \{x, xy, xy^2, \ldots, xy^k\}$.*
- *$W = \{\varepsilon, x, xy, xy^2, \ldots, xy^k\}$.*

2. *Every word in W acts as a permutation on T.*

3. *There exists $w \in W$ such that every word in $W \setminus w$ acts as a permutation on T.*

4. *W contains ε, and for every non-empty word $w \in W$, there exists a set S such that $T \setminus B \subseteq S \subseteq T$ and w acts as a permutation on S.*

5. *There exists a total order $\prec$ on T such that for all $p, q \in T$ with $p \prec q$, the set $qw_p^{-1} = \{s \in Q^{\mathcal{B}} : s \xrightarrow{w_p} q\}$ contains an element of T.*

If one of the above conditions holds, then every state of the form (s', X) with $B \subseteq X \subseteq T$ is reachable from (s', B) in $\mathcal{C}$.

5.3 Concatenation Witness Examples

We now demonstrate our technique by applying it to various concatenation witnesses from the literature.

Theorem 5.3.1 (Regular Language Witness. Brzozowski and Sinnamon, 2017 [18])**.** *Let $t\colon Q^{\mathcal{A}} \to Q^{\mathcal{A}}$ be a transformation such that $j't = 1'$. Define $\mathcal{A}$ and $\mathcal{B}$ as follows:*

	a	b	*Final States*
$\mathcal{A}$:	$(1', \ldots, m')$	t	$\{m'\}$
$\mathcal{B}$:	$(1, \ldots, n)$	$(2 \to 1)$	$\{n\}$

If $\gcd(j-1, n) = 1$, then $\mathcal{C}$ has $(m-1)2^n + 2^{n-1}$ reachable states. In particular, transformations t with $2't = 1'$ work for all m and n.

The authors of [18] proved this result with $t = (1', 2')$, but we prove a slightly more general statement.

Proof. The initial state of $\mathcal{C}$ is $(1', \emptyset)$. Set $x = a^m$ and $y = a^{j-1}b$. We have

$$(1', \emptyset) \xrightarrow{x} (1', 2) \xrightarrow{y^k} (1', 2 + k(j-1)).$$

(Addition in the second component is performed modulo n.) Since $j-1$ and n are coprime, it follows from elementary number theory that $W = \{x, xy, \dots, xy^{n-1}\}$ is a construction set for $Q^{\mathcal{B}}$ (with $s' = 1'$ and $B = \emptyset$). By Corollary 5.2.5, it is complete. Hence $(1', S)$ is reachable for all $S \subseteq Q^{\mathcal{B}}$. To reach (q', S) for q' non-final, first reach $(1', Sa^{-(q-1)})$ and then apply a^{q-1}. To reach $(m', S \cup 1)$ for $S \subseteq Q^{\mathcal{B}} \setminus 1$, first reach $((m-1)', Sa^{-1})$ and then apply a. □

Note that the above theorem only gives conditions for $(m-1)2^n + 2^{n-1}$ states to be *reachable*; it is not necessarily true that all the reachable states are pairwise distinguishable. For example, if t is the constant transformation $(Q^{\mathcal{A}} \to 1')$ then $(p', Q^{\mathcal{B}})$ and $(q', Q^{\mathcal{B}})$ are indistinguishable. However, in [18] the authors take t to be the transposition $(1', 2')$ and find that all reachable states are pairwise distinguishable.

In the remainder of our examples, all of the states we show are reachable will also be pairwise distinguishable. Since the focus of this paper is reachability, we refer to the original authors for distinguishability proofs in most cases. In cases where the original authors did not provide a distinguishability proof, we give a brief argument for completeness.

The next example involves two DFAs with different alphabets: we have $\Sigma^{\mathcal{A}} = \{a, b, c\}$ and $\Sigma^{\mathcal{B}} = \{a, b, d\}$. Our construction set will consist of words over the shared alphabet $\Sigma_0 = \Sigma^{\mathcal{A}} \cap \Sigma^{\mathcal{B}} = \{a, b\}$.

Theorem 5.3.2 (Regular Language Witness. Brzozowski, 2016 [7]). *Define $\mathcal{A}$ and $\mathcal{B}$ as follows:*

	a	b	c	d	*Final States*
$\mathcal{A}$:	$(1', \dots, m')$	$(1', 2')$	$(m' \to 1')$		$\{m'\}$
$\mathcal{B}$:	$(1, 2)$	$(1, \dots, n)$		id	$\{n\}$

Then $\mathcal{C}$ has $m2^n + 2^{n-1}$ reachable and pairwise distinguishable states.

Proof. The initial state is $(1', \emptyset)$. If n is odd, we have

$$(1', \emptyset) \xrightarrow{a^m} (1', 2) \xrightarrow{bb} (1', 4) \xrightarrow{bb} \cdots \xrightarrow{bb} (1', n-1),$$

$$(1', n-1) \xrightarrow{bb} (1', 1) \xrightarrow{bb} (1', 3) \xrightarrow{bb} \cdots \xrightarrow{bb} (1', n).$$

Thus $\{a^m, a^m bb, a^m(bb)^2, \dots, a^m(bb)^{n-1}\}$ is a construction set for $Q^{\mathcal{B}}$ (with $s' = 1'$ and $B = \emptyset$). By Corollary 5.2.5, it is complete (taking $x = a^m$ and $y = bb$).

If n is even, we have

$$(1', \emptyset) \xrightarrow{a^m} (1', 2) \xrightarrow{bb} (1', 4) \xrightarrow{bb} \cdots \xrightarrow{bb} (1', n),$$

$$(1', n) \xrightarrow{ab} (1', 1) \xrightarrow{bb} (1', 3) \xrightarrow{bb} \cdots \xrightarrow{bb} (1', n-1).$$

The words used to reach each state $(1', q)$ form a construction set for $Q^{\mathcal{B}}$ (with $s' = 1$ and $B = \emptyset$). We cannot use Corollary 5.2.5 to show it is complete (since the appearance of ab breaks the pattern), but notice that all the words in the construction set are words over $\{a, b\}$, and a and b both act as permutations on $Q^{\mathcal{B}}$. Thus all words in the construction set are permutations of $Q^{\mathcal{B}}$, and so by Corollary 5.2.6 it is complete.

In either case, we have a complete construction set for $Q^{\mathcal{B}}$ and so $(1', S)$ is reachable for all $S \subseteq Q^{\mathcal{B}}$. We can reach (q', S) for $q' \neq m'$ and $(m', S \cup 1)$ by words in a^*, as in Theorem 5.3.1. This gives $(m-1)2^n + 2^{n-1}$ reachable states. Additionally, from (q', S) we can reach $(\emptyset, S)$ by d, for an extra 2^n states.

For distinguishability of the reached states, see [7]. □

The main differences in reachability proofs between the different-alphabet case (unrestricted state complexity) and the same-alphabet case (restricted state complexity) are as follows:

- When looking for a complete construction set, we are restricted to using words over the shared alphabet $\Sigma_0 = \Sigma^{\mathcal{A}} \cap \Sigma^{\mathcal{B}}$.
- Usually some additional states can be reached using letters in $\Sigma^{\mathcal{A}} \setminus \Sigma^{\mathcal{B}}$ or $\Sigma^{\mathcal{B}} \setminus \Sigma^{\mathcal{A}}$, e.g., the states of the form $(\emptyset, S)$ in the previous example.

As these differences are not too significant, we will stick to the same-alphabet case for the remainder of our examples.

Theorem 5.3.3 (Regular Language Witness. Brzozowski, 2013 [6]). *Define $\mathcal{A}$ and $\mathcal{B}$ as follows:*

	a	b	c	*Final States*
$\mathcal{A}$:	$(1', \dots, m')$	$(1', 2')$	$(m' \to 1')$	$\{m'\}$
$\mathcal{B}$:	$(1, \dots, n)$	$(1, 2)$	$(n \to 1)$	$\{n\}$

Then $\mathcal{C}$ has $(m-1)2^n + 2^{n-1}$ reachable and pairwise distinguishable states.

Proof. The initial state of $\mathcal{C}$ is $(1', \emptyset)$. For $0 \leq k \leq n-2$ we have

$$(1', \emptyset) \xrightarrow{a^m} (1', 2) \xrightarrow{(ab)^k} (1', 2+k).$$

Also, $(1', n) \xrightarrow{c} (1', 1)$. Thus $\{a^m, a^m ab, a^m(ab)^2, \dots, a^m(ab)^{n-2}, a^m(ab)^{n-2}c\}$ is a construction set for $Q^{\mathcal{B}}$ (with $s' = 1'$ and $B = \emptyset$).

This construction set does not quite have the right form to apply Corollary 5.2.5, due to the last word $a^m(ab)^{n-2}c$. However, notice that all words in W except for $a^m(ab)^{n-2}c$ are in fact permutations of $Q^{\mathcal{B}}$, so Corollary 5.2.6 shows that W is complete. Hence all states $(1', S)$ with $S \subseteq Q^{\mathcal{B}}$ are reachable. We can reach (q', S) for $q' \neq m'$ and $(m', S \cup 1)$ by words in a^*, as in Theorem 5.3.1.

For distinguishability of the reached states, see [6]. $\square$

Theorem 5.3.4 (Regular Language Witness. Yu, Zhuang and Salomaa, 1994 [79]). *Define $\mathcal{A}$ and $\mathcal{B}$ as follows:*

	a	b	c	*Final States*
$\mathcal{A}$:	$(1', \dots, m')$	$(Q^{\mathcal{A}} \to 1')$	id	$\{m'\}$
$\mathcal{B}$:	id	$(1, \dots, n)$	$(Q^{\mathcal{B}} \to 2)$	$\{n\}$

Then $\mathcal{C}$ has $(m-1)2^n + 2^{n-1}$ reachable and pairwise distinguishable states.

Proof. The initial state of $\mathcal{C}$ is $(1', \emptyset)$. For $k \leq n-2$ we have $(1', \emptyset) \xrightarrow{a^m} (1', 2) \xrightarrow{b^k} (1', 2+k)$, and $(1', n) \xrightarrow{b} (1', 1)$. It follows that $\{a^m, a^m b, \dots, ab^{n-1}\}$ is a construction set for $Q^{\mathcal{B}}$ (with $s' = 1'$ and $B = \emptyset$). By Corollary 5.2.5, it is complete (taking $x = a^m$ and $y = b$). Hence all states $(1', S)$ with $S \subseteq Q^{\mathcal{B}}$ are reachable. We can reach (q', S) for $q' \neq m'$ and $(m', S \cup 1)$ by words in a^*.

Let (p', S) and (q', T) be distinct states of $\mathcal{C}$. If $S \neq T$, let r be a state in the symmetric difference of S and T. Then b^{n-r} distinguishes the states. If $S = T$ and $p' < q'$, then $ca^{m-q}b^{n-2}$ distinguishes the states. $\square$

Theorem 5.3.5 (Regular Language Witness. Maslov, 1970 [62]). *Define $\mathcal{A}$ and $\mathcal{B}$ as follows:*

	a	b	*Final States*
$\mathcal{A}$:	$(1', \dots, m')$	id	$\{m'\}$
$\mathcal{B}$:	$(n-1, n)$	$({}_{1}^{n-1} q \to q+1)$	$\{n\}$

Then $\mathcal{C}$ has $(m-1)2^n + 2^{n-1}$ reachable and pairwise distinguishable states.

Proof. The initial state is $(1', \emptyset)$. We have

$$(1', \emptyset) \xrightarrow{a^m} (1', 1) \xrightarrow{b^k} (1', 1+k).$$

Thus $\{a^m, a^m b, a^m b^2, \dots, a^m b^{n-1}\}$ is a construction set for $Q^{\mathcal{B}}$ (with $s' = 1'$ and $B = \emptyset$). By Corollary 5.2.5, it is complete. Hence $(1', S)$ is reachable for all $S \subseteq Q^{\mathcal{B}}$. We can reach (q', S) for $q' \neq m'$ and $(m', S \cup 1)$ by words in a^*, as in Theorem 5.3.1.

Let (p', S) and (q', T) be distinct states of $\mathcal{C}$. If $S \neq T$, let r be a state in the symmetric difference of S and T. Then b^{n-r} distinguishes the states. If $S = T$ and $p' < q'$, by b^n we reach (p', n) and (q', n). Then by a^{m-q} we reach $((p+m-q)', na^{m-q})$ and $(m', na^{m-q} \cup 1)$. These states differ in their second component, so they are distinguishable. $\square$

Theorem 5.3.6 (Star-Free Witness. Brzozowski and Liu, 2012 [15]). *Define $\mathcal{A}$ and $\mathcal{B}$ as follows:*

	a	b	c	d
$\mathcal{A}$:	$({}_{1}^{m-1}q' \to (q+1)')$	$({}_{2}^{m}q' \to (q-1)')$	id	$(Q^{\mathcal{A}} \to m')$
$\mathcal{B}$:	$({}_{2}^{n-1}q \to q+1)$	id	$({}_{1}^{n-1}q \to q+1)$	$({}_{2}^{n}q \to q-1)$

and let $F^{\mathcal{A}} = \{m'\}$ and $F^{\mathcal{B}} = \{n-1\}$. Then $\mathcal{C}$ has $(m-1)2^n + 2^{n-1}$ reachable and pairwise distinguishable states.

Proof. The initial state is $(1', \emptyset)$. We have

$$(1', \emptyset) \xrightarrow{a^m} (m', 1) \xrightarrow{c^k} (m', \{1, 1+k\}).$$

Hence $\{\varepsilon, c, c^2, \dots, c^{n-1}\}$ is a construction set for $Q^{\mathcal{B}}$ (with $s' = m'$ and $B = \{1\}$). By Corollary 5.2.5, it is complete. Thus $(m', S \cup 1)$ is reachable for all $S \subseteq Q^{\mathcal{B}}$.

To reach (q', S) for non-final $q' \in Q^{\mathcal{A}}$ and $S \subseteq Q^{\mathcal{B}}$, proceed as follows. If $1 \in S$, first reach $(m', S \cup 1)$ then apply b^{m-q}. If $1 \notin S$, let i be the smallest element of S. Set $T = \{q - (i-1) : q \in S \setminus i\}$ and reach $(m', T \cup 1)$. Then $(m', T \cup 1) \xrightarrow{b^{m-q}} (q', T \cup 1) \xrightarrow{c^{i-1}} (q', (S \setminus i) \cup i) = (q', S)$.

For distinguishability of the reached states, see [15]. $\square$

Theorem 5.3.7 (Non-Returning Witness. Brzozowski and Davies, 2017 [9]). *Define $\mathcal{A}$ and $\mathcal{B}$ as follows:*

	a	b	*Final States*
$\mathcal{A}$:	$(2', \dots, m')(1' \to 2')$	$(2', 3')(1' \to 3')$	$\{m'\}$
$\mathcal{B}$:	$(2, \dots, n)(1 \to 2)$	$(3, \dots, n)(2 \to 3)(1 \to 2)$	$\{n\}$

Then $\mathcal{C}$ has $(m-1)2^{n-1} + 1$ reachable and pairwise distinguishable states.

Proof. The initial state is $(1', \emptyset)$. Let $x = a^{m-1}$ and $y = ab$. If n is even,

$$(1', \emptyset) \xrightarrow{a} (2', \emptyset) \xrightarrow{x} (2', 2) \xrightarrow{y} (2', 4) \xrightarrow{y} (2', 6) \xrightarrow{y} \cdots \xrightarrow{y} (2', n),$$

$$(2', n) \xrightarrow{y} (2', 3) \xrightarrow{y} (2', 5) \xrightarrow{y} \cdots \xrightarrow{y} (2', n-1).$$

If n is odd,

$$(1', \emptyset) \xrightarrow{a} (2', \emptyset) \xrightarrow{x} (2', 2) \xrightarrow{y} (2', 4) \xrightarrow{y} (2', 6) \xrightarrow{y} \cdots \xrightarrow{y} (2', n-1),$$

$$(2', n-1) \xrightarrow{y} (2', 3) \xrightarrow{y} (2', 5) \xrightarrow{y} \cdots \xrightarrow{y} (2', n).$$

In both cases, Corollary 5.2.5 implies that $\{x, xy, \dots, xy^{n-2}\}$ is a complete construction set for $Q^{\mathcal{B}} \setminus 1$ (with $s' = 2'$ and $B = \emptyset$). It follows that $(2', S)$ is reachable for all $S \subseteq Q^{\mathcal{B}} \setminus 1$. This gives 2^{n-1} reachable states.

To reach (q', S) for non-final $q' \in Q^{\mathcal{A}} \setminus 1$ and $S \subseteq Q^{\mathcal{B}} \setminus 1$, note that a acts as a permutation on $Q^{\mathcal{B}} \setminus 1$, and so there exists $T \subseteq Q^{\mathcal{B}} \setminus 1$ such that $Ta^{q-2} = S$. Thus we can first reach $(2', T)$ and then apply a^{q-2}. To reach $(m', S \cup 1)$ for $S \subseteq Q^{\mathcal{B}} \setminus 1$, reach $((m-1)', T)$ where $Ta = S$ and apply a. Counting the initial state $(1, \emptyset)$, we get $(m-1)2^{n-1} + 1$ reachable states.

For distinguishability of the reached states, see [9]. $\square$

Theorem 5.3.8 (Non-Returning Witness. Eom, Han and Jirásková, 2016 [34]). *Define $\mathcal{A}$ and $\mathcal{B}$ as follows:*

$$\begin{array}{cccc}
 & a & b & c \\
\mathcal{A}: & (2', \dots, m')(1' \to 2') & (1' \to 2') & (1' \to 2') \\
\mathcal{B}: & (1 \to 2) & (2, \dots, n)(1 \to 2) & ({}_{3}^{n-1} q \to q+1)(1 \to 2)(n \to 2)
\end{array}$$

and let $F^{\mathcal{A}} = \{m'\}$ and $F^{\mathcal{B}} = \{n\}$. Then $\mathcal{C}$ has $(m-1)2^{n-1} + 1$ reachable and pairwise distinguishable states.

Proof. The initial state is $(1', \emptyset)$. We have

$$(1', \emptyset) \xrightarrow{a} (2', \emptyset) \xrightarrow{a^{m-1}} (2', 2) \xrightarrow{b^k} (2', 2+k).$$

Hence by Corollary 5.2.5, $\{a^{m-1}, a^{m-1}b, \dots, a^{m-1}b^{n-2}\}$ is a complete construction set for $Q^{\mathcal{B}} \setminus 1$ (with $s' = 2'$ and $B = \emptyset$). It follows that $(2', S)$ is reachable for all $S \subseteq Q^{\mathcal{B}} \setminus 1$. To reach (q', S) for q' non-final, reach $(2', S)$ and apply a^{q-2}. For $(m', S \cup 1)$, reach $((m-1)', S)$ and apply a.

For distinguishability of the reached states, see [34]. $\square$

Theorem 5.3.9 (Prefix-Closed Witness. Brzozowski, Jirásková and Zou, 2014 [14]). *Define $\mathcal{A}$ and $\mathcal{B}$ as follows:*

	a	b	c	*Final States*
$\mathcal{A}$:	id	id	$({}_1^{m-1} q' \to (q+1)')$	$\{1', \dots, (m-1)'\}$
$\mathcal{B}$:	$(1, \dots, n-1)$	$({}_2^{n-1} q \to q+1)$	id	$\{1, \dots, n-1\}$

Then $\mathcal{C}$ has $(m+1)2^{n-2}$ reachable and pairwise distinguishable states.

Proof. The initial state is $(1', 1)$. For $k \le n-2$ we have $(1', 1) \xrightarrow{a^k} (1', \{1, 1+k\})$. Thus by Corollary 5.2.5 the set $\{\varepsilon, a, a^2, \dots, a^{n-2}\}$ is a complete construction set for $Q^{\mathcal{B}} \setminus n$, with $s' = 1'$ and $B = \{1\}$. Hence $(1', S \cup 1)$ is reachable for each $S \subseteq Q^{\mathcal{B}} \setminus n$. From $(1', S \cup 1)$ with $S \subseteq Q^{\mathcal{B}} \setminus n$, we reach $(q', S \cup 1)$ for $2 \le q \le m$ by c^{q-1}. This gives $m2^{n-2}$ reachable states.

To reach (m', S) with $S \subseteq Q^{\mathcal{B}} \setminus n$, set S non-empty, and $1 \notin S$, let p be the smallest element of S. Let $T = Sa^{-(p-1)}$; then $1 \in T$ since $1a^{p-1} = p$. Reach (m', T) and apply a^{p-1} to reach (m', S). There are $2^{n-2} - 1$ non-empty sets that exclude 1 and n, and we can reach an additional state (m', n) from $(m', n-1)$ by b. This gives another 2^{n-2} reachable states, for a total of $(m+1)2^{n-2}$ states.

For distinguishability of the reached states, see [14]. ☐

Theorem 5.3.10 (Suffix-Free Witness. Brzozowski and Sinnamon, 2017 [16]). *Define $\mathcal{A}$ and $\mathcal{B}$ as follows:*

	a	b	c
$\mathcal{A}$:	$(1' \to m')(2', \dots, (m-1)')$	$(1' \to m')(2', 3')$	$(2', m')(1' \to 2')$
$\mathcal{B}$:	$(1 \to n)(2, 3)$	$(2, n)(1 \to 2)$	$(1 \to n)(2, \dots, n-1)$

and let $F^{\mathcal{A}} = \{(m-1)'\}$ and $F^{\mathcal{B}} = \{n-1\}$. Then $\mathcal{C}$ has $(m-1)2^{n-2} + 1$ reachable and pairwise distinguishable states.

Proof. The initial state is $(1', \emptyset)$. We have

$$(1', \emptyset) \xrightarrow{c} (2', \emptyset) \xrightarrow{a^{m-3}} ((m-1)', 1) \xrightarrow{c} ((m-1)', \{1, n\}).$$

Then for $k \le n-3$ we have

$$((m-1)', \{1, n\}) \xrightarrow{bb} ((m-1)', \{1, 2, n\}) \xrightarrow{c^k} ((m-1)', \{1, 2+k, n\}).$$

Thus $W = \{\varepsilon, bb, bbc, bbc^2, \dots, bbc^{n-3}\}$ is a construction set for $Q^{\mathcal{B}}$, with $s' = (m-1)'$ and $B = \{1, n\}$. In fact, W is complete by Lemma 5.2.7 since b and c act as permutations on $Q^{\mathcal{B}} \setminus 1$.

It follows that $((m-1)', S \cup \{1, n\})$ is reachable for all $S \subseteq Q^{\mathcal{B}}$. To reach $(q', S \cup n)$ for $2 \le q \le m-2$ and $1 \notin S$, note that a acts as a permutation on $Q^{\mathcal{B}} \setminus 1$. Thus we first reach $((m-1)', Sa^{-(q-1)} \cup \{1, n\})$ then apply a^{q-1}. To reach $(m', S \cup n)$ with $1 \notin S$, first reach $(2', Sc^{-1} \cup n)$ then apply c. Since there are 2^{n-2} subsets of $Q^{\mathcal{B}} \setminus \{1, n\}$, this gives $(m-1)2^{n-2}$ reachable states. Adding one for the initial state $(1', \emptyset)$ gives $(m-1)2^{n-2} + 1$.

For distinguishability of the reached states, see [16]. Note that the authors of [16] use a different concatenation DFA from our $\mathcal{C}$: they first delete the sink states m' from $\mathcal{A}$ and n from $\mathcal{B}$, and then form the concatenation of these modified DFAs. However, the same words used for distinguishing states in [16] can be used to distinguish states of $\mathcal{C}$. ☐

Theorem 5.3.11 (Suffix-Free Witness. Han and Salomaa, 2009 [41]). *Define $\mathcal{A}$ and $\mathcal{B}$ as follows:*

	$\mathcal{A}$:	$\mathcal{B}$:
a	$(2', \dots, (m-1)')(1' \to m')$	$(1 \to n)$
b	$(1' \to m')$	$(2, \dots, n-1)(1 \to n)$
c	$((Q^{\mathcal{A}} \setminus 1') \to m')(1' \to 2')$	$(1 \to n)$
d	$((Q^{\mathcal{A}} \setminus 2') \to m')$	$(1 \to 2)$

and let $F^{\mathcal{A}} = \{2'\}$ and $F^{\mathcal{B}} = \{2\}$. Then $\mathcal{C}$ has $(m-1)2^{n-2} + 1$ reachable and pairwise distinguishable states.

Proof. The initial state is $(1', \emptyset)$. For $k \le n-3$ we have

$$(1', \emptyset) \xrightarrow{cb} (2', \{1, n\}) \xrightarrow{d} (2', \{1, 2, n\}) \xrightarrow{b^k} (2', \{1, 2+k, n\}).$$

Thus $W = \{\varepsilon, d, db, \dots, db^{n-3}\}$ is a construction set for $Q^{\mathcal{B}}$, with $s' = 2'$ and $B = \{1, n\}$. By Lemma 5.2.7, W is complete, since d and b act as permutations on $Q^{\mathcal{B}} \setminus \{1, n\}$.

There are 2^{n-2} states of the form $(2', S \cup \{1, n\})$ with $S \subseteq Q^{\mathcal{B}}$ and $S \cap \{1, n\} = \emptyset$. For each of these states, we reach $(q', S \cup n)$ for $3 \le q \le m-1$ by a^{q-2}, and $(m', S \cup n)$ by c. Adding in the initial state $(1', \emptyset)$ gives a total of $(m-1)2^{n-2} + 1$ reachable states.

For distinguishability of the reached states, see [41]. Note that the authors of [41] work with a reduced concatenation DFA obtained by identifying, for each q' and S, the indistinguishable states (q', S) and $(q', S \cup n)$. Thus, for example, they write that $(q', \emptyset)$ is reachable for $3 \le q \le m-1$; these states are not reachable in our DFA $\mathcal{C}$, but states (q', n) for $3 \le q \le m-1$ are reachable. ☐

Theorem 5.3.12 (Right Ideal Witness. Brzozowski and Sinnamon, 2017 [18]). *Define $\mathcal{A}$ and $\mathcal{B}$ as follows:*

	a	b	c	*Final States*
$\mathcal{A}$:	$(1', \dots, (m-1)')$	$(2' \to 1')$	$({}_{1}^{m-1} q' \to (q+1)')$	$\{m'\}$
$\mathcal{B}$:	$(1, \dots, n-1)$	$(2 \to 1)$	$({}_{1}^{n-1} q \to q+1)$	$\{n\}$

Then $\mathcal{C}$ has $m + 2^{n-2}$ reachable and pairwise distinguishable states.

Proof. The initial state is $(1', \emptyset)$. Note that $(1', \emptyset) \xrightarrow{a^{q-1}} (q', \emptyset)$ for $1 \le q \le m-1$, so these $m-1$ states are reachable. For $0 \le k \le n-3$ we have

$$((m-1)', \emptyset) \xrightarrow{c} (m', 1) \xrightarrow{a} (m', \{1, 2\}) \xrightarrow{(ab)^k} (m', \{1, 2+k\}).$$

Hence $\{\varepsilon, a, aab, a(ab)^2, \dots, a(ab)^{n-3}\}$ is a construction set for $Q^{\mathcal{B}} \setminus n$, with $s' = m'$ and $B = \{1\}$. By Corollary 5.2.5, it is complete. Hence $(m', S \cup 1)$ is reachable for all $S \subseteq Q^{\mathcal{B}} \setminus n$.

We have reached $(m-1) + 2^{n-2}$ states so far. Additionally, we have $(m', \{1, n-1\}) \xrightarrow{cb} (m', \{1, n\})$, giving $m + 2^{n-2}$.

For distinguishability of the reached states, see [18]. $\square$

Theorem 5.3.13 (Right Ideal Witness. Brzozowski, Davies and Liu, 2016 [10]). *Define $\mathcal{A}$ and $\mathcal{B}$ as follows:*

	a	b	c	*Final States*
$\mathcal{A}$:	$(1', \dots, (m-1)')$	$(2', \dots, (m-1)')$	$((m-1)' \to m')$	$\{m'\}$
$\mathcal{B}$:	$(1, \dots, n-1)$	$(2, \dots, n-1)$	$(n-1 \to n)$	$\{n\}$

Then $\mathcal{C}$ has $m + 2^{n-2}$ reachable and pairwise distinguishable states.

Proof. The initial state is $(1', \emptyset)$. Note that $(1', \emptyset) \xrightarrow{a^{q-1}} (q', \emptyset)$ for $1 \le q \le m-1$, so these $m-1$ states are reachable. For $0 \le k \le n-3$ we have

$$((m-1)', \emptyset) \xrightarrow{c} (m', 1) \xrightarrow{a} (m', \{1, 2\}) \xrightarrow{b^k} (m', \{1, 2+k\}).$$

Hence $\{\varepsilon, a, ab, ab^2, \dots, ab^{n-3}\}$ is a construction set for $Q^{\mathcal{B}} \setminus n$, with $s' = m'$ and $B = \{1\}$. By Corollary 5.2.5, it is complete. Hence $(m', S \cup 1)$ is reachable for all $S \subseteq Q^{\mathcal{B}} \setminus n$.

We have reached $(m-1) + 2^{n-2}$ states so far. Additionally, we have $(m', \{1, n-1\}) \xrightarrow{c} (m', \{1, n\})$, giving $m + 2^{n-2}$.

For distinguishability of the reached states, see [10]. $\square$

Theorem 5.3.14 (Right Ideal Witness. Brzozowski, Jirásková and Li, 2013 [11])**.** *Define $\mathcal{A}$ and $\mathcal{B}$ as follows:*

	a	b	*Final States*
$\mathcal{A}$:	$({}_1^{m-1} q' \to (q+1)')$	$({}_1^{m-1} q' \to (q+1)')$	$\{m'\}$
$\mathcal{B}$:	$(1, \dots, n-1)$	$({}_2^{n-1} q \to q+1)$	$\{n\}$

Then $\mathcal{C}$ has $m + 2^{n-2}$ reachable and distinguishable states.

Proof. The initial state is $(1', \emptyset)$. Note that $(1', \emptyset) \xrightarrow{a^{q-1}} (q', \emptyset)$ for $1 \le q \le m-1$, so these $m-1$ states are reachable. For $0 \le k \le n-3$ we have

$$((m-1)', \emptyset) \xrightarrow{a} (m', 1) \xrightarrow{a} (m', \{1, 2\}) \xrightarrow{b^k} (m', \{1, 2+k\}).$$

Hence $\{\varepsilon, a, ab, ab^2, \dots, ab^{n-3}\}$ is a construction set for $Q^{\mathcal{B}} \setminus n$, with $s' = m'$ and $B = \{1\}$. By Corollary 5.2.5, it is complete. Hence $(m', S \cup 1)$ is reachable for all $S \subseteq Q^{\mathcal{B}} \setminus n$.

We have reached $(m-1) + 2^{n-2}$ states so far. Additionally, we have $(m', \{1, n-1\}) \xrightarrow{b} (m', \{1, n\})$, giving $m + 2^{n-2}$.

For distinguishability of the reached states, see [11]. Note that the authors of [11] use a different concatenation DFA, constructed by removing state m' from $\mathcal{A}$ and then forming the concatenation in the usual way. However, the same words used in [11] can be used to distinguish states in $\mathcal{C}$. □

We now give two examples where our method of proof does not seem applicable or helpful. When attempting concatenation state complexity proofs, it seems best to consider both traditional techniques and the technique we present in this chapter, switching between the two options if one does not yield an easy argument.

Example 5.3.15 (Prefix-Closed Witness. Brzozowski and Sinnamon, 2017 [18])**.** Our technique does not seem to work well with the following witness languages. Define $\mathcal{A}$ and $\mathcal{B}$ as follows:

	$\mathcal{A}$:	$\mathcal{B}$:
a	$(1', \dots, (m-1)')$	$(1, \dots, n-1)$
b	$(1', 2')$	$(2 \to 1)$
c	$(2' \to 1')$	$({}_1^{n-1} q \to q-1)$
d	$({}_1^{m-1} q' \to (q-1)')$	$(1, 2)$

and let $F^{\mathcal{A}} = \{1', \dots, (m-1)'\}$ and $F^{\mathcal{B}} = \{1, \dots, n-1\}$.

The inductive proof given by the authors of [18] has a different structure from the type of argument captured by Theorem 5.2.2. To reach a state (q', S), in Theorem 5.2.2 we start from some state (q', B) and apply a word that fixes the first component q'. In [18] the authors instead start from a state (p', B) and apply a word w such that $p'w = q'$. The proof in [18] is short and clean, whereas a proof in the style of Theorem 5.2.2 seems to require complicated arguments. It is possible that Theorem 5.2.2 could be generalized to cover arguments of the form used in [18], but we have not found such a generalization. ■

Example 5.3.16 (Finite Binary Witness. Câmpeanu, Culik, Salomaa and Yu, 2001 [22])**.** Our technique does not apply to the following witness languages. Define $\mathcal{A}$ and $\mathcal{B}$ as follows:

	a	b	Final States
$\mathcal{A}$:	$({}_1^{m-1} q' \to (q+1)')$	$({}_1^{m-1} q' \to (q+1)')$	$\{1', \dots, (m-1)'\}$
$\mathcal{B}$:	$({}_2^{n-1} q \to q+1)(1 \to n)$	$({}_1^{n-1} q \to q+1)$	$\{n-1\}$

Additionally, assume that $m+1 \geq n > 2$. Then $\mathcal{C}$ has $(m-n+3)2^{n-2}-1$ reachable and pairwise distinguishable states. This is the maximum for finite languages over a binary alphabet when $m+1 \geq n > 2$.

Let us consider why Theorem 5.2.2 cannot be used here. The point of Theorem 5.2.2 is to build up states (s', S) by starting from (s', B) and using words that fix the focus state s'. But in this witness, no state of $\mathcal{A}$ is fixed by any word except for the non-final sink state m'. So to use Theorem 5.2.2, the focus state must be m'. But from a state of the form (m', S), we can only reach sets (m', T) with $|T| \leq |S|$, since m' is a non-final sink state. So there is no way to start from some base state (m', B) and build up larger sets, which is the strategy of Theorem 5.2.2. ■

5.4 Construction Sets for Star

To close this chapter, we define a notion of construction sets for star and prove some results analogous to those proved in Section 5.2. Let $\mathcal{A} = (Q, \Sigma, T, 1, F)$ be a DFA and let $\mathcal{S} = (Q^{\mathcal{S}}, \Sigma, T^{\mathcal{S}}, \{s\}, F^{\mathcal{S}})$ be the star DFA as defined in Section 2.4.4.

Definition 5.4.1. Let $B, T \subseteq Q$ be sets with $B \cap T = \emptyset$; we call B the *base* and T the *target*. Write $T = \{q_1, \dots, q_k\}$ and let $W = \{w_1, \dots, w_k\}$ be a set of $|T|$ words. We say W is a *construction set* for B and T if $Bw_i = B \cup q_i$ in $\mathcal{S}$ for $1 \leq i \leq k$. A construction set W is *complete* if there exists a total order $\prec$ on $B \cup T$ such that whenever $q_i \prec q_j$, we have $q_j w_i^{-1} \cap (B \cup T) \neq \emptyset$.

Since our research on construction sets for star was conducted after defining construction sets for concatenation, hindsight allowed us to modify the definition of construction set slightly to make it easier to use. In the definition for concatenation automata, the base B was always a subset of the target T. Thus it was often necessary to include the empty word ε in a construction set solely to satisfy the condition that for each $q \in B$ there is a word w such that $Bw = B \cup q$. In construction sets for star, the base and target are disjoint, avoiding this issue.

It is easy to verify that the following results from Section 2.4.4 hold when updated to account for the definition of star construction sets. This is because their proofs do not rely on any properties of concatenation itself, but rather just properties of concatenation *construction sets* that are mimicked by star construction sets.

Lemma 5.4.2. *Let W be a construction set for B and T. If one of the following conditions holds, W is complete:*

1. *There exist words $x, y \in \Sigma^*$, where x acts as a permutation on $B \cup T$, such that $W = \{x, xy, xy^2, \dots, xy^k\}$.*
2. *There exists a word $y \in \Sigma^*$ such that $W = \{y, y^2, \dots, y^k\}$.*
3. *Every word in W acts as a permutation on $B \cup T$.*
4. *There exists $w \in W$ such that every word in $W \setminus w$ acts as a permutation on $B \cup T$.*
5. *For each $w \in W$, there exists a set S with $T \subseteq S \subseteq B \cup T$ such that w acts as a permutation on S.*

Complete construction sets can be used to demonstrate the reachability of certain subsets in star automata.

Theorem 5.4.3. *If there is a complete construction set for B and T, then all sets S with $B \subseteq S \subseteq B \cup T$ are reachable from B in $\mathcal{S}$.*

Proof. Set $R = S \setminus B$, so that S is the disjoint union $B \cup R$. We prove that S is reachable from B by induction on $|R|$. If $|R| = 0$, we just have $S = B$.

Now suppose that every state $S = B \cup R$ with $B \cap R = \emptyset$ and $|R| < \ell$ is reachable from B. We want to show this holds for $|R| = \ell$. Let $T = \{q_1, \dots, q_k\}$, let $W = \{w_1, \dots, w_k\}$ be a complete construction set for T, and let $\prec$ the corresponding total order on T. Note that $R \subseteq T$; let q_i be the minimal element of R under $\prec$. For all $q_j \in R \setminus q_i$, we have $q_i \prec q_j$ and thus $q_j w_i^{-1} \cap T \neq \emptyset$.

Construct sets X and Y as follows: for each $q \in R \setminus q_i$, choose an element of $q_j w_i^{-1} \cap (B \cup T)$ and add it to X. Then set $Y = X \setminus B$. Observe that X is a subset of $B \cup T$ of size $|R \setminus q_i| = \ell - 1$. Hence Y has size at most $\ell - 1$, and Y is disjoint from B, and so it follows by the induction hypothesis that $B \cup Y$ is reachable from B. But $B \cup Y = B \cup (X \setminus B) = B \cup X$, and thus $B \cup X$ is reachable from B. By the definition of X, we have $X w_i = R \setminus q_i$. Also, since W is a construction set, we have $B w_i = B \cup q_i$. It follows that

$$(B \cup X) w_i = B w_i \cup X w_i = (B \cup q_i) \cup (R \setminus q_i) = B \cup R = S.$$

Hence S is reachable from B, as required. □

The special initial state $\{s\}$ might get in the way when using Theorem 5.4.3. We can deal with this using the following proposition. Let t be the permutation of $Q \cup \{s\}$ that swaps 1 and s and fixes all other elements.

Proposition 5.4.4. *Let $S \neq B$. In $\mathcal{S}$, if S is reachable from B, then S is reachable from Bt.*

Proof. If B contains $\{1, s\}$, or if B is disjoint from $\{1, s\}$, then $B = Bt$. Assume that B contains exactly one element of $\{1, s\}$. Without loss of generality, assume $s \in B$. Note then that $Bt = (B \setminus s) \cup 1$.

Since $S \neq B$, it follows that S must be reachable from B via a *non-empty* word w. For each transition $(1, a, q)$, the star automaton is defined to have a corresponding transition (s, a, q). Thus if w is non-empty, then $sw = 1w$. Therefore

$$Bw = (B \setminus s)w \cup sw = (B \setminus s)w \cup 1w = ((B \setminus s) \cup 1)w = Btw.$$

Hence $Bw = Btw = S$ as required. □

Typically a star reachability proof using Theorem 5.4.3 will start by showing that all sets containing the initial state are reachable. To show the remaining sets are reachable, the following lemma is useful. A subset X of a totally ordered set $(\prec, Q)$ is *downward closed* if whenever $p \prec q$ for some $q \in X$, we have $p \in X$.

Lemma 5.4.5. *Let $\prec$ be a total order on Q and let qs denote the successor of q under the total order. Let X be a downward closed subset of $(Q \setminus F) \cup 1$ such that for some $a \in \Sigma$, we have $qa = qs$ in $\mathcal{A}$ for all $q \in X$. If every subset of X containing the minimal element of Q is reachable in $\mathcal{S}$, then every non-empty subset of X is reachable.*

Proof. Write $Q = \{q_1, \ldots, q_n\}$ with $q_1 \prec q_2 \prec \cdots \prec q_n$. Let S be a subset of X and let q_i be the minimal element of S. We may assume $i > 1$ since otherwise S contains the minimal element q_1 of Q and is reachable by assumption. Set $Y = \{q : qs^{i-1} \in S\}$. Since $q_1 s^{i-1} = q_i$, the set Y contains the minimal element q_1 of Q, and thus Y is reachable in $\mathcal{S}$.

We claim that $Ya^{i-1} = S$, and thus S is reachable. First note that for each element $q \in Y$ we have $q \prec qs \prec \cdots \prec qs^{i-1}$ with $qs^{i-1} \in X$. Since X is downward closed, it follows that for all $q \in Y$ and $0 \leq j \leq i-1$, we have $qs^j \in X$. Now, we claim that $Ya^j = \{qs^j : q \in Y\}$. We proceed by induction on j. The base case $j = 0$ is trivial.

Suppose that $0 < j \leq i-1$ and that $Ya^{j-1} = \{qs^{j-1} : q \in Y\}$. In $\mathcal{S}$, for $q \in Q$ we have $q \xrightarrow{a} qa$ if qa is non-final in $\mathcal{A}$ and $q \xrightarrow{a} \{1, qa\}$ if qa is final in $\mathcal{A}$. Note that $qs^{j-1} \in X$, so $qs^{j-1}a = qs^j$ in $\mathcal{A}$. Since $j \leq i-1$, the state qs^j belongs to $X \subseteq (Q \setminus F) \cup 1$. Thus either qs^j is non-final or $qs^j = 1$. In both cases, in $\mathcal{S}$ we have $qs^{j-1} \xrightarrow{a} qs^j$ for all $q \in Y$. It follows that $Ya^j = \{qs^j : q \in Y\}$, as required. Since this holds for $j = i-1$, we get $Ya^{i-1} = \{qs^{i-1} : q \in Y\} = S$, and the proof is complete. □

Corollary 5.4.6. *Assume we either have $F = \{n-k+1, \ldots, n\}$ for $k \geq 1$, or $F = \{1, n-k+2, \ldots, n\}$ for $k \geq 2$. Suppose that for some m and some $a \in \Sigma$, we have $qa = q+1$ for all $q \leq m$ in $\mathcal{A}$. If every subset of $\{1, \ldots, m\}$ containing 1 is reachable in $\mathcal{S}$, then every non-empty subset of $\{1, \ldots, m\}$ is reachable. In particular, if every subset of Q containing 1 is reachable, then at least $2^{n-1} + 2^{m-1}$ subsets are reachable.*

Proof. Suppose every subset of $\{1, \ldots, m\}$ containing 1 is reachable in $\mathcal{S}$. By applying Lemma 5.4.5 with the total order $<$ on Q and $X = \{1, \ldots, m\}$, we see that every non-empty subset of $\{1, \ldots, m\}$ is reachable.

Suppose now that every subset of Q containing 1 is reachable. Then there are 2^{n-1} subsets of Q containing 1, and $2^{m-1} - 1$ non-empty subsets of $\{1, \ldots, m\}$ that do not contain 1, and the initial subset $\{s\}$, giving $2^{n-1} + 2^{m-1}$ subsets. □

Remark. Corollary 5.4.6 can be used to show that $2^{n-1} + 2^{m-1}$ subsets are reachable in $\mathcal{S}$. If $1 \notin F$, we can take $m = n-k$ to get the upper bound $2^{n-1} + 2^{n-k-1}$. If $1 \in F$, we can take $m = n-k+1$ to get $2^{n-1} + 2^{n-k}$ reachable subsets; noting that sets $\{s\}$ and $\{1\}$ are always indistinguishable gives the upper bound $2^{n-1} + 2^{n-k} - 1$.

We close this section by applying our results on star to a witness from the literature.

Theorem 5.4.7 (Yu, Zhuang and Salomaa, 1994 [79]). *Define $\mathcal{A}$ as follows:*

$$\begin{aligned} a &= (1, \ldots, n) \\ b &= (n \to 1)({}_{2}^{n-1}\, q \to q+1) \\ F &= \{n\} \end{aligned}$$

Then $\mathcal{S}$ has $2^{n-1} + 2^{n-2}$ reachable states.

Proof. State $\{1\}$ is reachable from $\{s\}$ via b. We have

$$\{1\} \xrightarrow{a} \{2\} \xrightarrow{a} \cdots \xrightarrow{a} \{n, 1\} \xrightarrow{a} \{1, 2\} \xrightarrow{b^k} \{1, 2 + k\}.$$

Thus $\{a^n, a^n b, \dots, a^n b^{n-3}\}$ is a construction set for $B = \{1\}$ and $T = Q \setminus 1$. It is complete by Lemma 5.2.4. Hence every subset of Q containing 1 is reachable in $\mathcal{S}$. Taking $m = n-1$ in Corollary 5.4.6 gives $2^{n-1} + 2^{n-2}$ reachable states. $\square$

Chapter 6

Reversal and Deterministic Finite Automata with Output

6.1 Introduction

In this chapter, we take a bit of a detour and look at a state complexity problem for an extension of the DFA model. Compared with previous chapters, the results here are not as directly applicable to classical DFA state complexity problems. However, we will gain some new insights into the reversal operation, some of which apply to the ordinary DFA case. I thank Jeffrey Shallit for proposing this problem in his course on automatic sequences.

The problem of determining the worst-case state complexity of the reversal operation on regular languages has been well-studied. Work on this problem dates back to the 1960s; see Jirásková and Šebej [54] for a historical overview. It is known that if L is recognized by an n-state DFA, then state complexity of the reverse L^R is at most 2^n, and this bound can be reached over a binary alphabet; furthermore, it can be reached by DFAs which have only one final state.

In this section, we study a generalization of this problem to *deterministic finite automata with output* (DFAOs). Rather than a set of final states, in a DFAO, each state is assigned an output value from a finite *output alphabet* Δ. Rather than recognizing languages, DFAOs compute functions $f \colon \Sigma^* \to \Delta$, where Σ is the *input alphabet*. The value wf is defined to be the output value of the state reached by starting in the initial state and following the path corresponding to the input word w. Note that the case $|\Delta| = 2$ can be viewed as assigning a value of "final" or "non-final" to each state, so DFAOs directly generalize DFAs.

DFAOs are used in the study of *automatic sequences* [1]. If we treat the words $w \in \Sigma^*$ as representations of natural numbers in some base, we can view the function $f\colon \Sigma^* \to \Delta$ as a function $f\colon \mathbb{N} \to \Delta$, that is, an infinite sequence of elements of Δ. Sequences for which the corresponding function can be computed by a DFAO are called *automatic*.

The *reverse* of the function $f\colon \Sigma^* \to \Delta$ is the function $f^R\colon \Sigma^* \to \Delta$ defined by $wf^R = w^R f$. The reversal operation on DFAOs can be thus be viewed as changing the direction in which the DFAO reads input: from left-to-right to right-to-left, or vice versa. Some functions are easier to compute with respect to one input-reading direction than the other. For example, consider the function $f\colon \{0,1\}^* \to \{0,1\}$, which takes in the binary representation of a natural number and outputs 1 if the number can be written as $8n+5$, $n \geq 0$, and 0 otherwise. Numbers of the form $8n+5$ have binary representations of the form $w101$, where $w \in \{0,1\}^*$. Hence if the input is read from left-to-right, the entire string must be read to determine whether it ends in 101, but if the input is read from right-to-left, only three characters need to be checked. Likewise, some automatic sequences are easier to generate if we read the input numbers from least-significant digit to most-significant-digit, rather than the opposite way.

We are concerned with the maximal blow-up in size (number of states) when the input reading direction of a DFAO is reversed. That is, given a function f computed by an n-state DFAO, what is the worst-case state complexity of f^R? The standard construction for reversal of DFAOs [1, Theorem 4.3.3] gives an upper bound of $|\Delta|^n$, where Δ is the output alphabet. However, it does not seem to be known whether this bound is reachable.

We prove that when the input alphabet has size three or greater, the upper bound $|\Delta|^n$ is indeed reachable. When the input alphabet is binary, the problem becomes much more complicated. We conjecture that if $|\Delta| \geq 3$, the upper bound $|\Delta|^n$ is not reachable over a binary alphabet, despite the fact that it is known to be reachable for $|\Delta| = 2$ (the ordinary DFA case). While we could not prove that the upper bound is unreachable in all cases, we have proved it is unreachable when $|\Delta| = n$ and $|\Delta| \geq 3$, and verified computationally that it is unreachable for $(|\Delta|, n) \in \{(3,4),(3,5),(3,6),(4,5)\}$. We prove a lower bound for the case of a binary input alphabet and $3 \leq |\Delta| < n$.

We also demonstrate that the state complexity of DFAO reversal is completely determined by the transition monoid of the DFAO and the map which assigns outputs to states. In particular, if function f is computed by a minimal n-state DFAO with state set Q, transition monoid M, and output map $\tau\colon Q \to \Delta$, then the state complexity of f^R is exactly $|M\tau|$, where $M\tau = \{m\tau : m \in M\}$. Since DFAs are special cases of DFAOs, this gives a surprising new characterization of the state complexity of DFA reversal in terms of the transition monoid and the characteristic function of the final state set.

6.2 Deterministic Finite Automata with Output

A *deterministic finite automaton with output* (DFAO) is a 6-tuple $\mathcal{D} = (Q, \Sigma, T, 1, \Delta, \tau)$, where: Q, Σ, T and 1 are as in a DFA, Δ is the finite *output alphabet*, and $\tau\colon Q \to \Delta$ is the *output map*.

While traditional DFAs recognize languages, DFAOs instead compute functions. The *function computed by a DFAO* is the function $f\colon \Sigma^* \to \Delta$ defined by $wf = 1w\tau$. That is, we determine wf by starting in the initial state 1, following the path corresponding to w to reach some state q, then applying the output map τ to get the output value associated with q. A function that can be computed by a DFAO is called a *finite state function*.

Most definitions for DFAs that do not involve final states extend to DFAOs, such as the concepts of reachability and accessibility, and the definition of the transition monoid. Even notions which involve final states can often be generalized. For example, two states p and q in a DFAO are *distinguishable* if there exists $w \in \Sigma^*$ such that $pw\tau \neq qw\tau$. A DFAO is *minimal* if it has the least possible number of states among all DFAOs computing the same function. We can generalize fundamental results on DFA minimality to DFAOs:

Proposition 6.2.1. *A DFAO is minimal if and only if all states are reachable and every pair of distinct states is distinguishable.*

The proof is done by mimicking the corresponding proof for DFAs. For further reference on the DFAO model, see Allouche and Shallit [1].

When working with multiple DFAOs with different transition functions, say $\mathcal{D} = (Q, \Sigma, T, 1, \Delta, \tau)$ and $\mathcal{D}' = (Q', \Sigma', T', 1', \Delta', \tau')$, the shorthand notation w for word actions is ambiguous: it is unclear whether this is the action of w in $\mathcal{D}$ or in $\mathcal{D}'$. Thus in this section, we adopt the following convention: the notation w always refers to the action of w in a DFAO whose transition function is named "T". Thus w would refer to the action T_w of $\mathcal{D}$, rather than T'_w of $\mathcal{D}'$.

The *reverse* of a finite state function $f\colon \Sigma^* \to \Delta$ is the function $f^R\colon \Sigma^* \to \Delta$ defined by $wf^R = w^R f$. Following Allouche and Shallit [1, Theorem 4.3.3], we give a DFAO construction for f^R in terms of a DFAO for f.

Proposition 6.2.2. *Let $\mathcal{D} = (Q, \Sigma, T, 1, \Delta, \tau)$ be a DFAO computing the function f. There exists a DFAO $\mathcal{R}$ computing f^R.*

Proof. Let $\mathcal{R} = (\Delta^Q, \Sigma, T^{\mathcal{R}}, \tau, \Delta, \Omega)$, where:

- The state set is Δ^Q, the set of all functions from Q to Δ.
- The initial state is $\tau\colon Q \to \Delta$, the output map of $\mathcal{D}$.
- The transition function $T^{\mathcal{R}}$ is defined as follows: $gT^{\mathcal{R}}_a = ag$, for $g \in \Delta^Q$ and $a \in \Sigma$.
- The output map $\Omega\colon \Delta^Q \to \Delta$ is defined by $g\Omega = 1g$.

By definition, the function computed by $\mathcal{D}$ is $wf = 1w\tau$. The function computed by $\mathcal{R}$ is $\tau T^{\mathcal{R}}_w \Omega = 1\tau T^{\mathcal{R}}_w$; we must show this equals $wf^R = w^R f$. If $w = a_1 a_2 \cdots a_n$, then we have

$$\tau T^{\mathcal{R}}_w = \tau T^{\mathcal{R}}_{a_1} T^{\mathcal{R}}_{a_2} \cdots T^{\mathcal{R}}_{a_n} = a_1 \tau T^{\mathcal{R}}_{a_2} \cdots T^{\mathcal{R}}_{a_n} = a_2 a_1 \tau T^{\mathcal{R}}_{a_3} \cdots T^{\mathcal{R}}_{a_n} = \cdots = a_n \cdots a_2 a_1 \tau = w^R \tau.$$

It follows that

$$1\tau T^{\mathcal{R}}_w = 1 w^R \tau = w^R f = w f^R,$$

as required. □

The *state complexity* of a finite state function is the size of a minimal DFAO computing the function. If a function f is computed by an n-state minimal DFAO (i.e., the function has state complexity n), Proposition 6.2.2 shows that the state complexity of f^R is bounded above by $|\Delta|^n$, since the size of the state set Δ^Q of $\mathcal{R}$ is $|\Delta|^{|Q|} = |\Delta|^n$.

The following proposition, analogous to Lemma 2.4.24, makes it easier to compute the state complexity of f^R.

Proposition 6.2.3. *If $\mathcal{D}$ is accessible, then all states of $\mathcal{R}$ are pairwise distinguishable.*

Proof. Let g and h be distinct states of $\mathcal{R}$. There exists $q \in Q$ such that $qg \neq qh$. Since $\mathcal{D}$ is accessible, q is reachable. Choose $w \in \Sigma^*$ such that $q_0 w^R = q$. Observe that $gT^{\mathcal{R}}_w \Omega = q_0 w^R g = qg$, and similarly $hT^{\mathcal{R}}_w \Omega = qh$. Since $qg \neq qh$, g and h are distinguishable. □

If we take $\mathcal{R}$ and remove all unreachable states from it (which does not change the function computed), we obtain a DFAO for f^R with all states reachable and every pair of distinct states distinguishable. By Proposition 6.2.1, this is a minimal DFAO for f^R. Hence given a function f computed by an accessible DFAO $\mathcal{D}$, to determine the state complexity of f^R, we can simply count the number of reachable states in $\mathcal{R}$.

6.3 State Complexity of Reversal

We first prove an important proposition, which shows that the state complexity of reversal of DFAOs is completely determined by the transition monoid and the output map.

Definition 6.3.1. Let X and Y be finite sets; for a transformation monoid M on X and a function $f\colon X \to Y$, define Mf to be the set of functions $\{mf : m \in M\}$.

Proposition 6.3.2. *Let $\mathcal{D} = (Q, \Sigma, T, 1, \Delta, \tau)$ be an accessible DFAO computing function f. Let M be the transition monoid of $\mathcal{D}$. The state complexity of f^R is $|M\tau|$, where $M\tau = \{w\tau : w \in \Sigma^*\}$.*

Proof. The DFAO $\mathcal{R} = (\Delta^Q, \Sigma, T^{\mathcal{R}}, \tau, \Delta, \Omega)$ computes f^R. By Proposition 6.2.3, all states of $\mathcal{R}$ are distinguishable, so the state complexity of f^R is the number of reachable states in $\mathcal{R}$.

Recall from the proof of Proposition 6.2.2 that $gT^{\mathcal{R}}_w = w^R g$ for $g\colon Q \to \Delta$ and $w \in \Sigma^*$. In particular, since τ is the initial state of $\mathcal{R}$, every reachable state of $\mathcal{R}$ has the form $\tau T^{\mathcal{R}}_w = w^R\tau$. Hence the set of reachable states of $\mathcal{R}$ is $\{w^R\tau : w \in \Sigma^*\}$. But this is the same set as $M\tau = \{w\tau : w \in \Sigma^*\}$. It follows that the number of reachable states in $\mathcal{R}$ is precisely $|M\tau|$. □

Recall that for $|\Delta| = 2$, DFAOs are essentially the same as DFAs (if we view the output map as a boolean function telling us whether a state is final or non-final). Hence we have the following corollary:

Corollary 6.3.3. *Let $\mathcal{D} = (Q, \Sigma, T, 1, F)$ be an accessible DFA recognizing language L. Let M be the transition monoid of $\mathcal{D}$. The state complexity of L^R is $|M\chi_F|$, where $\chi_F\colon Q \to \{0, 1\}$ is the characteristic function of F.*

Despite the simple proof, we found this result rather surprising. We have not seen a similar characterization of the state complexity of DFA reversal anywhere in the literature.

Throughout the rest of this section, Q and Δ will be finite sets with $|Q| = n$ and $|\Delta| = k$, the monoid M will be a transformation monoid on Q, and $\tau\colon Q \to \Delta$ will be a surjective function. Note that the surjectivity of τ implies $|\Delta| \leq |Q|$. It is fine to make this assumption, since if $|\Delta| > |Q|$ there are more possible outputs than there are states, and so we can shrink Δ without loss of generality.

Theorem 6.3.4. *Let M be the full transformation monoid on Q. Then $|M\tau| = k^n$ for all surjective functions $\tau\colon Q \to \Delta$.*

Proof. It suffices to show that every function $h\colon Q \to \Delta$ lies in $M\tau$, i.e., every such function h can be written as $g\tau$ for some $g\colon Q \to Q$.

For $q \in Q$, we define qg as follows. Since τ is surjective, there exists $p_q \in Q$ such that $p_q\tau = qh$. Define $qg = p_q$. Then $qg\tau = p_q\tau = qh$ for all $q \in Q$, so $g\tau = h$ as required. □

Corollary 6.3.5. *Let f be a finite state function computed by a minimal DFAO $\mathcal{D} = (Q, \Sigma, T, 1, \Delta, \tau)$ with $|\Delta| \leq |Q|$ (i.e., $k \leq n$). The state complexity of f^R is at most $|\Delta|^{|Q|} = k^n$, and this bound can be reached when $|\Sigma| \geq 3$.*

Proof. The upper bound on f^R follows from the construction for $\mathcal{R}$. It suffices to prove this bound can be reached. We saw in Example 2.4.3 that the following three functions are sufficient to generate the full transformation monoid T_n:

$$f_1 = (1, 2, \ldots, n), \quad f_2 = (1, 2), \quad f_3 = (n \to 1).$$

Choose $\{a, b, c\} \subseteq \Sigma$ and let $\mathcal{D}$ be a DFAO with actions $a = f_1$, $b = f_2$ and $c = f_3$. Then the transition monoid M of $\mathcal{D}$ is the full transformation monoid. Furthermore, $\mathcal{D}$ is accessible (all states can be reached via a). Hence Proposition 6.3.2 applies. If we take the output map τ to be surjective, by Theorem 6.3.4 we get that the state complexity of f^R is $|M\tau| = k^n$, as required. □

The rest of this section is devoted to the case where $|\Sigma| = 2$, i.e., where the input alphabet of the DFAO is binary. This case is significantly more complicated and difficult than the $|\Sigma| \geq 3$ case. Note that if $|\Delta| = 2$, this case is equivalent to studying reversal of ordinary DFAs with binary alphabets, and it is known for DFAs that the upper bound of 2^n is reachable [54]. Thus we will only be concerned with $|\Delta| \geq 3$.

Since the state complexity of DFAO reversal is completely determined by the transition monoid and output map, naturally there are connections between the $|\Sigma| = 2$ case and the problem of finding the largest 2-generated transformation monoids of a particular degree. This problem has been studied by Holzer and König [43] and Krawetz, Lawrence and Shallit [59].

Following Holzer and König, we define two families of monoids. First and most important are the $U_{\ell,m}$ monoids [43, Definition 5]. The monoid $U_{\ell,m}$ is a transformation monoid on $Q = \{1, \ldots, \ell + m\}$ defined as follows. Let $\alpha\colon Q \to Q$ be the permutation $(1, \ldots, \ell)(\ell + 1, \ldots, \ell + m)$.

Definition 6.3.6. A function $\gamma\colon Q \to Q$ belongs to $U_{\ell,m}$ if and only if it satisfies one of the following conditions:

1. There exists $i \geq 0$ such that $\gamma = \alpha^i$.

2. $\{1, \ldots, \ell\}\gamma \cap \{\ell+1, \ldots, \ell+m\}\gamma \neq \emptyset$, and there exists an element $i \in \{\ell+1, \ldots, \ell+m\}$ such that i is not in the image of γ.

If $1 < \ell < m$ and $\gcd(\ell, m) = 1$, then $U_{\ell,m}$ can be generated by two elements [43, Theorem 8]. Krawetz [58] gives an explicit generating set: one of the generators is α, and the other is $\beta\colon Q \to Q$, where

$$\beta = \begin{pmatrix} 1 & 2 & 3 & 4 & \cdots & \ell+m-1 & \ell+m \\ \ell+1 & 2 & 3 & 4 & \cdots & \ell+m-1 & 1 \end{pmatrix}$$

if $k = 2$ or ℓ is even, and otherwise

$$\beta = \begin{pmatrix} 1 & 2 & 3 & 4 & \cdots & \ell+m-1 & \ell+m \\ \ell+1 & 3 & 2 & 4 & \cdots & \ell+m-1 & 1 \end{pmatrix}.$$

Let $n = \ell + m$. For $n \geq 7$ and n prime, Holzer and König proved that there exist ℓ and m with $1 < \ell < m$ and $\gcd(\ell, m) = 1$ such that $U_{\ell,m}$ is the largest 2-generated transformation monoid [43, Theorem 15]. They conjecture that this also holds when $n \geq 7$ and n is not prime.

When $n \leq 6$, the largest 2-generated transformation monoids belong to a different family: the V_n^d monoids [43, Definition 16]. Let α be the permutation $(1, 2, \ldots, n)$.

Definition 6.3.7. A function $\gamma\colon Q \to Q$ belongs to V_n^d if and only if it satisfies one of the following conditions:

1. There exists $i \geq 0$ such that $\gamma = \alpha^i$.

2. There exist $i, j \in \{1, \ldots, n\}$ such that $i\gamma = j\gamma$ and $j \equiv i + d \pmod{n}$.

For $2 \leq n \leq 6$, Holzer and König determined explicit generating sets for the largest 2-generated transformation monoids on $Q = \{1, \ldots, n\}$, which are all V_n^d monoids for some d. One of the generators is always $\alpha_n = (1, 2, \ldots, n)$. For $2 \leq n \leq 6$, the other generator β_n is:

$$\beta_2 = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}, \quad \beta_3 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 1 & 3 \end{pmatrix}, \quad \beta_4 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 1 & 4 & 3 \end{pmatrix},$$

$$\beta_5 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 1 & 4 & 5 & 3 \end{pmatrix}, \quad \beta_6 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 4 & 1 & 5 & 6 & 2 \end{pmatrix}.$$

Holzer and König also give a more general construction for 2-element generating sets of V_n^d monoids [43, Theorem 18].

With these definitions done, we return to the problem of computing worst-case state complexity of reversal for binary input alphabets. First we consider the special case $|Q| = |\Delta|$. Here it turns out that the state complexity problem almost completely reduces to the 2-generated monoid problem:

Theorem 6.3.8. *Let f be a finite state function computed by a minimal DFAO $\mathcal{D} = (Q, \Sigma, \cdot, q_0, \Delta, \tau)$ with $|\Sigma| = 2$ and $|Q| = |\Delta| = n$. Let $m_2(n)$ denote the size of the largest 2-generated transformation monoid on $Q = \{1, 2, \dots, n\}$ that occurs as the transition monoid of some accessible DFA. The state complexity of f^R is at most $m_2(n)$, and this bound is reachable.*

Proof. Let $\Sigma = \{a, b\}$. By assumption, we can construct an accessible DFAO $\mathcal{D}$ so that a and b generate a monoid of size $m_2(n)$. Let $\tau \colon Q \to \Delta$ be a bijection. By Proposition 6.3.2, the state complexity of f^R is $|M\tau|$. But τ is a bijection, so $|M\tau| = |M| = m_2(n)$. □

It may be the case that for some values of n, the largest transformation monoid on $\{1, 2, \dots, n\}$ generated by two elements does *not* occur as the transition monoid of a accessible DFA. Thus we do not quite get a complete reduction to the 2-generated monoid problem. It seems very unlikely that a monoid which is not a transition monoid of a accessible DFA could be maximal, since this would mean there is some state q for which the monoid contains no functions mapping a reachable state to q, which would exclude very many functions. Note that the $U_{\ell,m}$ and V_n^d monoids do occur as transition monoids of accessible DFAs.

It is well known that if $|Q| \geq 3$, the full transformation monoid on a finite set Q cannot be generated by two elements. Hence $m_2(n)$ never reaches the upper bound of $|\Delta|^{|Q|} = n^n$ except when $|Q| = n = 2$.

Table 6.3.1 shows the known values for $m_2(n)$ for $2 \leq n \leq 7$, taken from [43, Table 1]. The value is not known for $n > 7$ except when n is prime, in which case $m_2(n)$ is the size of the largest 2-generated $U_{\ell,m}$ monoid. The values of n^n are also shown for comparison.

We now turn to the case where $|\Delta| < |Q|$. Our main result in this case is a formula for the size of $|U_{\ell,m}\tau|$, which in turn leads to a lower bound on the worst-case state complexity of f^R.

n	2	3	4	5	6	7
$m_2(n)$	4	24	176	2110	32262	610871
n^n	4	27	256	3125	46656	823543

Table 6.3.1: Values of $m_2(n)$ for $2 \leq n \leq 7$.

Theorem 6.3.9. *Let $|\Delta| = k$ and let $|Q| = \ell + m = n$, with $2 \leq k < n$ and $1 \leq \ell \leq m$. We define two functions:*

$$(k, \ell, m)F = \sum_{i=1}^{\ell} \binom{k}{i} i! \left\{ {\ell \atop i} \right\} (k-i)^m.$$

$$(k, \ell, m)G = \begin{cases} \operatorname{lcm}(\ell, m), & \text{if } k \geq 4; \\ m, & \text{if } k = 3; \\ 1, & \text{if } k = 2. \end{cases}$$

There exists a function $\tau \colon Q \to \Delta$ such that

$$|U_{\ell,m}\tau| = k^n - (k, \ell, m)F + (k, \ell, m)G.$$

The notation $\left\{ {\ell \atop i} \right\}$ means the number of partitions of the set $\{1, \ldots, \ell\}$ into i parts (that is, a Stirling number of the second kind).

Proof. We start with a brief outline of the proof strategy. Without loss of generality, assume $\Delta = \{1, \ldots, k\}$ and $Q = \{1, \ldots, n = \ell + m\}$. Define $F_{\ell,m} = \{f \colon Q \to \Delta : \{1, \ldots, \ell\}f \cap \{\ell + 1, \ldots, \ell + m\}f = \emptyset\}$.

- First, we show that $\Delta^Q = U_{\ell,m}\tau \cup F_{\ell,m}$ for certain τ.
- After proving this, the inclusion-exclusion principle gives the formula

$$k^n = |\Delta^Q| = |U_{\ell,m}\tau| + |F_{\ell,m}| - |U_{\ell,m}\tau \cap F_{\ell,m}|.$$

- We show that $|F_{\ell,m}| = (k, \ell, m)F$.
- We show that $|U_{\ell,m}\tau \cap F_{\ell,m}| = (k, \ell, m)G$.
- Rearranging the inclusion-exclusion formula above gives the result.

Let us show that for an appropriate choice of $\tau\colon Q \to \Delta$, we have $\Delta^Q = U_{\ell,m}\tau \cup F_{\ell,m}$. That is, every function from Q to Δ lies in one of $U_{\ell,m}\tau$ or $F_{\ell,m}$.

Let $\alpha\colon Q \to Q$ be the permutation $(1,\dots,\ell)(\ell+1,\dots,\ell+m)$. We select τ with the following properties:

- $\tau\colon Q \to \Delta$ is surjective.
- $\{1,\dots,\ell\}\tau \cap \{\ell+1,\dots,\ell+m\}\tau = \emptyset$, that is, $\tau \in F_{\ell,m}$.
- There exist distinct $p, p' \in \{\ell+1,\dots,\ell+m\}$ such that $p\tau = p'\tau$.
- The size of the set $\{\alpha^i\tau : i \geq 0\}$ is precisely $(k,\ell,m)G$.

We demonstrate that such a function exists after this proof, in Lemma 6.3.11. In that lemma, we will see existence of such a function requires $k < n$ and $\ell \leq m$; this is the only place we use these hypotheses.

Now, let $g\colon Q \to \Delta$ be arbitrary. We will show that if g is *not* in $F_{\ell,m}$, then it must be in $U_{\ell,m}\tau$, thus proving that $\Delta^Q = U_{\ell,m}\tau \cup F_{\ell,m}$. To show that $g \in U_{\ell,m}\tau$, we define a function $f\colon Q \to Q$ such that $f \in U_{\ell,m}$ and $f\tau = g$.

Since $g \notin F_{\ell,m}$, there exist distinct elements $r \in \{1,\dots,\ell\}$ and $r' \in \{\ell+1,\dots,\ell+m\}$ such that $rg = r'g$. Since τ is surjective, there exists s such that $s\tau = rg$. Furthermore, we can choose s so that $s \neq p'$. Indeed, if p' is one of the possible choices for s, then by the fact that $p\tau = p'\tau$, we can choose $s = p$ instead. Now, we define $f\colon Q \to Q$ for each $q \in Q$ as follows:

- If $q \in \{r, r'\}$, define $qf = s$.
- If $qg = p\tau$ and $q \notin \{r, r'\}$, define $qf = p$.
- Otherwise, choose an element q' such that $q'\tau = qg$ (by surjectivity) and define $qf = q'$.

We verify in each case that $f\tau = g$:

- If $q = r$, then $rf = s$, so $rf\tau = s\tau = rg$.
- If $q = r'$, then $qf = s$, and since $rg = r'g$ we have $r'f\tau = s\tau = rg = r'g$.
- If $q \notin \{r, r'\}$ and $qg = p\tau$, then $qf = p$, so $qf\tau = p\tau = qg$.

- Otherwise, we have $qf = q'$ such that $qf\tau = q'\tau = qg$.

Now, we show that $f \in U_{\ell,m}$. First, note that there exist elements $r \in \{1, \ldots, \ell\}$ and $r' \in \{\ell+1, \ldots, \ell+m\}$ such that $rf = r'f$. Next, observe that the element $p' \in \{\ell+1, \ldots, \ell+m\}$ is not in the image of f. To see this, note that if we have $qf = p'$, then we have $qf\tau = p'\tau = p\tau$. But $qf\tau = qg$, so this implies $qg = p\tau$. In the case where $qg = p\tau$, we defined $qf = p \neq p'$, so this is a contradiction. It follows that f meets the conditions to belong to $U_{\ell,m}$.

This proves that if $g\colon Q \to \Delta$ is not in $F_{\ell,m}$, then $g \in U_{\ell,m}\tau$ and thus $\Delta^Q = U_{\ell,m}\tau \cup F_{\ell,m}$. Next, we show that $|F_{\ell,m}| = (k, \ell, m)F$.

Write $f \in F_{\ell,m}$ in "list notation" as $[a_1, a_2, \ldots, a_\ell, b_1, b_2, \ldots, b_m]$, where $if = a_i$ and $(\ell + i)f = b_i$. For this function to lie in $F_{\ell,m}$, we must have the property that $\{a_1, a_2, \ldots, a_\ell\} \cap \{b_1, b_2, \ldots, b_m\} = \emptyset$. Note that since $F_{\ell,m}$ is a set of functions from Q to Δ, we have $\{a_1, \ldots, a_\ell\}, \{b_1, \ldots, b_m\} \subseteq \Delta$. We count the number of distinct "function lists" in $F_{\ell,m}$ as follows:

- Fix a set $S \subseteq \Delta$ and assume $\{a_1, \ldots, a_\ell\} = S$. Let $|S| = i$.
- In the first segment $[a_1, \ldots, a_\ell]$ of the list, each a_i can be a arbitrary element of S. However, since $\{a_1, \ldots, a_\ell\} = S$, each element of S must appear at least once in the list. Thus the first segment $[a_1, \ldots, a_\ell]$ of the list represents a *surjective* function from $\{1, \ldots, \ell\}$ onto S. Since $|S| = i$, the number of such surjective functions is $i!\left\{{\ell \atop i}\right\}$, where $\left\{{\ell \atop i}\right\}$ denotes a Stirling number of the second kind (the number of partitions of $\{1, \ldots, \ell\}$ into i parts).
- In the second segment $[b_1, \ldots, b_m]$ of the list, each b_i must be an element of $\Delta \setminus S$, since we want $\{a_1, \ldots, a_\ell\} \cap \{b_1, \ldots, b_m\} = \emptyset$. Since $|S| = i$ and $|\Delta| = k$, there are $k - i$ elements to pick from in $\Delta \setminus S$, and we need to choose m of them. Thus there are $(k-i)^m$ choices for the second segment of the list.
- In total, for a fixed set S of size i, there are $i!\left\{{\ell \atop i}\right\}(k-i)^m$ distinct lists that have $\{a_1, \ldots, a_k\} = S$.
- Now, we take the sum over all possible choices for the set S. Since $S = \{a_1, \ldots, a_\ell\}$ and S is non-empty, we have $1 \leq |S| \leq \ell$. For each set size i, there are $\binom{k}{i}$ ways to choose $S \subseteq \Delta$ with $|S| = i$. Thus the total number of functions in $F_{\ell,m}$ is

$$\sum_{i=1}^{\ell} \binom{k}{i} i! \left\{{\ell \atop i}\right\} (k-i)^m = (k, \ell, m)F.$$

Next, we show that $|U_{\ell,m}\tau \cap F_{\ell,m}| = (k,\ell,m)G$. We claim that

$$U_{\ell,m}\tau \cap F_{\ell,m} = \begin{cases} \emptyset, & \text{if } \tau \notin F_{\ell,m}; \\ \{\alpha^i\tau : i \geq 0\}, & \text{if } \tau \in F_{\ell,m}. \end{cases}$$

Then the size equality with $(k,\ell,m)G$ follows from the properties of τ.

To see the claim, suppose that $g\tau \in F_{\ell,m}$ for some $g \in U_{\ell,m}$. Since $g \in U_{\ell,m}$, either $g = \alpha^i$ for some i, or there exists $p \in \{1,\dots,\ell\}$ and $q \in \{\ell+1,\dots,\ell+m\}$ such that $pg = qg$. In the latter case, $pg\tau = qg\tau$, which contradicts the assumption that $g\tau$ is in $F_{\ell,m}$. Hence $g = \alpha^i$ for some $i \geq 0$, and so $g\tau = \alpha^i\tau$. Now, note that $\{1,\dots,\ell\}\alpha^i\tau = \{1,\dots,\ell\}\tau$, and $\{\ell+1,\dots,\ell+m\}\alpha^i\tau = \{\ell+1,\dots,\ell+m\}\tau$. Thus $\alpha^i\tau$ is in $F_{\ell,m}$ if and only if τ is in $F_{\ell,m}$, and the claim follows.

Finally, we can conclude the proof. Recall that $|\Delta| = k$ and $|Q| = n$, and thus $|\Delta^Q| = |\Delta|^{|Q|} = k^n$. Thus by the inclusion-exclusion principle, we have

$$k^n = |\Delta^Q| = |U_{\ell,m}\tau| + |F_{\ell,m}| - |U_{\ell,m}\tau \cap F_{\ell,m}|.$$

Rearranging this, we get:

$$|U_{\ell,m}\tau| = k^n - |F_{\ell,m}| + |U_{\ell,m}\tau \cap F_{\ell,m}|.$$

We proved that $|F_{\ell,m}| = (k,\ell,m)F$ and $|U_{\ell,m}\tau \cap F_{\ell,m}| = (k,\ell,m)G$. It follows that $|U_{\ell,m}\tau| = k^n - (k,\ell,m)F + (k,\ell,m)G$, as required. □

This theorem gives the following lower bound on the worst-case state complexity of DFAO reversal when $|\Sigma| = 2$.

Corollary 6.3.10. *Let $|Q| = n \geq 2$ and $|\Delta| = k \geq 2$. There exists an accessible DFAO $\mathcal{D} = (Q,\Sigma,T,1,\Delta,\tau)$ computing function f, with $|\Sigma| = 2$ and $k < n$, such that the state complexity of f^R is*

$$\max\{k^n - (k,\ell,m)F + (k,\ell,m)G : 1 < \ell < m, \ell + m = n, \gcd(\ell,m) = 1\}.$$

Proof. Pick ℓ and m such that $1 < \ell < m$, $\ell + m = n$ and $\gcd(\ell,m) = 1$. Then $U_{\ell,m}$ can be generated by two elements. Hence we can construct a DFAO $\mathcal{D}$ over a binary alphabet with state set $Q = \{1,\dots,n\}$ and transition monoid $U_{\ell,m}$. This DFAO will be accessible: all states in $\{1,\dots,\ell\}$ are reachable by $\alpha = (1,\dots,\ell)(\ell+1,\dots,\ell+m)$, and $U_{\ell,m}$ contains elements which map 1 to $\ell+1$, so the rest of the states are reachable. By Theorem 6.3.9, there exists $\tau\colon Q \to \Delta$ such that

$$|U_{\ell,m}\tau| = k^n - (k,\ell,m)F + (k,\ell,m)G.$$

Take τ as the output map of $\mathcal{D}$. Then by Proposition 6.3.2, the state complexity of f^R is $|U_{\ell,m}\tau|$. Taking the maximum over all values of ℓ and m that satisfy the desired properties gives the result. □

Table 6.3.2 gives the values of this lower bound for various values of $|\Delta| = k$ and $|Q| = n$ with $k < n$. Note that for $n \in \{1,2,3,4,6\}$ there are no pairs (ℓ, m) such that $1 < \ell < m$, $\ell + m = n$ and $\gcd(\ell,m) = 1$, so those values of n are ignored. Note that for $|\Delta| = 2$, this

$k\backslash n$	5	6	7	8	9
2	31	–	127	255	511
3	216	–	2125	6452	19550
4	826	–	15472	63403	258360
5	–	–	71037	368020	1902365
6	–	–	243438	1539561	9657446

Table 6.3.2: Values for the lower bound of Corollary 6.3.10.

lower bound is off by one from the upper bound of 2^n. The known examples where the upper bound 2^n is achieved do not use $U_{\ell,m}$ monoids.

We suspect the lower bound of Corollary 6.3.10 may be optimal for $n \geq 7$. We were unable to find any examples exceeding the bound through computational experiments.

The case of $n = 5$, which is the only case below 7 where our lower bound is defined, is rather interesting. Holzer and König proved by brute force search that the largest 2-generated transformation monoid of degree 5 is V_5^1, so one might expect that the maximal values of $|M\tau|$ in the $n = 5$ case would be given by taking $M = V_5^1$. Indeed, for $(k,n) = (3,5)$, the true maximum is 218, and this is achieved by $|V_5^1\tau|$ with $\tau = [1,2,1,2,3]$. However, for $(k,n) = (4,5)$: the value 826 is achieved by $|U_{2,3}\tau|$ with $\tau = [1,2,3,4,4]$, while the maximal value of $|V_5^1\tau|$ over all τ and d is 789, despite the fact that $|U_{2,3}| = 1857$ and $|V_5^1| = 2110$. Thus maximal monoids M do not necessarily give the maximal values for $|M\tau|$.

In the proof of Theorem 6.3.9, we used the fact that a function with certain properties exists. We now give a rather tedious proof of this fact.

Lemma 6.3.11. *Let $\Delta = \{1, \dots, k\}$ and let $Q = \{1, \dots, n\}$, with $2 \leq k < n$. Fix ℓ and m such that $\ell + m = n$ and $1 \leq \ell \leq m$. Let $\alpha \colon Q \to Q$ be the permutation $\alpha = (1, \dots, \ell)(\ell + 1, \dots, \ell + m)$. There exists a function $\tau \colon Q \to \Delta$ with the following properties:*

- *$\tau \colon Q \to \Delta$ is surjective.*
- *$\{1, \dots, \ell\}\tau \cap \{\ell + 1, \dots, \ell + m\}\tau = \emptyset$.*
- *There exist distinct $p, p' \in \{\ell + 1, \dots, \ell + m\}$ such that $p\tau = p'\tau$.*
- *The size of the set $\{\alpha^i\tau : i \geq 0\}$ is precisely given by the function $(k, \ell, m)G$ defined in Theorem 6.3.9.*

Proof. For $|\Delta| = k \geq 3$ and $(\ell, m) \neq (2, 2)$, we define τ as follows:

1. We partition Δ into two sets L and M, with $|L| = \min\{k - 2, \ell\}$.
2. (a) If $|L| = \ell$, let $L = \{1, \dots, \ell\}$ and define $i\tau = i$ for $1 \leq i \leq \ell$.
   (b) If $|L| = k - 2 < \ell$, write $L = \{1, \dots, k - 2\}$ and define $i\tau = i$ for $1 \leq i \leq k - 2$. Define $i\tau = k - 2$ for $k - 1 \leq i \leq \ell$.
3. Consider $|M| = k - |L|$.
   (a) Suppose $k - 2 \geq \ell$, and thus $|L| = \ell$. Then $\ell + 2 \leq k < n = \ell + m$, so $2 \leq k - \ell < m$. Thus $2 \leq |M| < m$.
   (b) Suppose $k - 2 < \ell$, and thus $|L| = k - 2$. Then $|M| = k - \ell = 2$.
4. (a) If $|L| = \ell$, we have $M = \{\ell + 1, \dots, \ell + j\}$, where $j < m$ is such that $\ell + j = k$. Define $(\ell + i)\tau = \ell + i$ for $1 \leq i \leq j$. Define $(\ell + i)\tau = \ell + j = k$ for $j + 1 \leq i \leq m$.
   (b) If $|L| = k - 2$, we have $M = \{k - 1, k\}$. Define $(\ell + 1)\tau = k - 1$ and $(\ell + i)\tau = k$ for $2 \leq i \leq m$.

To illustrate this construction, we give three examples of τ for different values of k, ℓ and m. If $k = 6$, $\ell = 3$ and $m = 5$, we partition $\Delta = \{1, 2, 3, 4, 5, 6\}$ into $L = \{1, 2, 3\}$ and $M = \{4, 5, 6\}$, and we get

$$\tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 6 & 6 \end{pmatrix}.$$

If $k = 5$, $\ell = 4$ and $m = 5$, we partition $\Delta = \{1, 2, 3, 4, 5\}$ into $L = \{1, 2, 3\}$ and $M = \{4, 5\}$, and we get

$$\tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 3 & 4 & 5 & 5 & 5 & 5 \end{pmatrix}.$$

If $k = 3$, $\ell = 4$ and $m = 4$, we partition $\Delta = \{1, 2, 3, 4, 5\}$ into $L = \{1\}$ and $M = \{2, 3\}$, and we get

$$\tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 1 & 1 & 1 & 2 & 3 & 3 & 3 \end{pmatrix}.$$

Note that in all cases, $\{1, \dots, \ell\}\tau = L$ and $\{\ell + 1, \dots, \ell + m\}\tau = M$.

This covers the definition of τ for $k \geq 3$ and $(\ell, m) \neq (2, 2)$. In the special case where $k \geq 3$ and $(\ell, m) = (2, 2)$, the fact that $k < n = \ell + m = 4$ implies $k = 3$. Here we define

$$\tau = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 3 \end{pmatrix}.$$

Finally, for $k = 2$, we define τ by $i\tau = 1$ for $1 \leq i \leq \ell$ and $i\tau = 2$ for $\ell + 1 \leq i \leq \ell + m$.

We now demonstrate τ has all the desired properties. The surjectivity of τ and the fact that $\{1, \dots, \ell\}\tau \cap \{\ell + 1, \dots, \ell + m\}\tau = \emptyset$ can be easily verified by a close reading of the definition.

Consider the third property: there exist distinct $p, p' \in \{\ell + 1, \dots, \ell + m\}$ such that $p\tau = p'\tau$. We can see this from the construction of τ as follows:

- For $k \geq 3$ and $(\ell, m) \neq (2, 2)$, observe that we have $\{\ell + 1, \dots, \ell + m\}\tau = M$. If $|L| = \ell$, we have $|M| < m$, so τ must identify two elements of $\{\ell + 1, \dots, \ell + m\}$. If $|L| = k - 2$, then $|M| = 2$, so we have $|M| < m$ in all cases except $m = 2$. But if $m = 2$, then $\ell \leq m$ implies $\ell \leq 2$. Since $(\ell, m) \neq (2, 2)$, we must have $\ell = 1$; but the fact that $k < n = \ell + m$ implies $k < 3$, which is a contradiction. So if $k \geq 3$ and $(\ell, m) \neq (2, 2)$, then $\{\ell + 1, \dots, \ell + m\}$ gets mapped onto a set of size less than m by τ.
- For $k \geq 3$ and $(\ell, m) = (2, 2)$, we see that $3\tau = 4\tau$.
- For $k = 2$, by the fact that $k < n = \ell + m$ we must have $m \geq 2$, and so $(\ell + 1)\tau = (\ell + m)\tau = 2$ and $\ell + 1 \neq \ell + m$.

Finally, consider the last property: the set $\{\alpha^i\tau : i \geq 0\}$ has size $G(k, \ell, m)$. That is, it has size $\operatorname{lcm}(\ell, m)$ if $k \geq 4$, size m if $k = 3$, and size 1 if $k = 2$.

Write τ in "list notation" as $[a_1, a_2, \dots, a_n]$, where $i\tau = a_i$ for $1 \le i \le n$. Note that $L = \{a_1, \dots, a_\ell\}$ and $M = \{a_{\ell+1}, \dots, a_{\ell+m}\}$. Observe that the list notation for $\alpha\tau$ is $[a_{1\alpha}, a_{2\alpha}, \dots, a_{n\alpha}]$. Similarly, the list notation for $\alpha^i\tau$ is $[a_{1\alpha^i}, a_{2\alpha^i}, \dots, a_{n\alpha^i}]$. It follows that the number of distinct "function lists" in the set $\{\alpha^i\tau : i \ge 0\}$ is bounded by the order of the permutation α, which is $\operatorname{lcm}(\ell, m)$.

For $k \ge 4$, we will show that all $\operatorname{lcm}(\ell, m)$ of these lists are distinct. To see this, let q be the smallest element of M, and consider where the values 1 and q appear in the list $[a_{1\alpha^i}, a_{2\alpha^i}, \dots, a_{n\alpha^i}]$. By the definition of α, the value 1 must appear at some position p_i with $1 \le p_i \le \ell$ and $p_i + i \equiv 1 \pmod{\ell}$. Similarly, the value q must appear some position r_i with $\ell + 1 \le r_i \le \ell + m$ and $r_i + i \equiv q \pmod{m}$. Notice that by the definition of τ, the elements 1 and q have unique preimages under τ, and thus each only appears once in the list.

We claim that if $i \not\equiv j \pmod{\operatorname{lcm}(\ell, m)}$, then $(p_i, r_i) \ne (p_j, r_j)$. To see this, suppose for a contradiction that $(p_i, r_i) = (p_j, r_j)$. Since $p_i = p_j$, and we have $p_i + i \equiv p_j + j \equiv 1 \pmod{\ell}$, it follows that $i \equiv j \pmod{\ell}$. Similarly, we have $i \equiv j \pmod{m}$. Write $i = j + x\ell$ for some integer x; then we have $j + x\ell \equiv j \pmod{m}$. It follows $x\ell \equiv 0 \pmod{m}$ and thus m divides $x\ell$. Since m and ℓ both divide $x\ell$, it follows that $\operatorname{lcm}(\ell, m)$ divides $x\ell$, and so we can write $x\ell = y \operatorname{lcm}(m, \ell)$ for some integer y. Thus $i = j + y \operatorname{lcm}(m, \ell)$ and it follows that $i \equiv j \pmod{\operatorname{lcm}(m, \ell)}$.

This proves that if $\alpha^i \ne \alpha^j$, then the positions of 1 and q in the lists for $\alpha^i\tau$ and $\alpha^j\tau$ will be different. Thus $\alpha^i\tau \ne \alpha^j\tau$, and it follows the size of $\{\alpha^i\tau : i \ge 0\}$ is precisely the order of the permutation α, which is $\operatorname{lcm}(\ell, m)$.

This deals with the $k \ge 4$ case. For $k = 3$, first suppose $(\ell, m) \ne (2, 2)$. Since $k = 3$, we have $k - 2 = 1$, so we always have $\min\{k - 2, \ell\} = k - 2$. In this case, recall that we have $i\tau = k - 2 = 1$ for $1 \le i \le \ell$.

It follows that $L = \{a_1, \dots, a_\ell\} = \{1\}$, so the list notation for $\tau \circ \alpha^i$ looks like $[1, 1, \dots, 1, a_{(\ell+1)\alpha^i}, \dots, a_{(\ell+m)\alpha^i}]$. Hence in this case, we get m distinct lists, corresponding to the m possible positions of q (which still has a unique preimage under τ) in the second part of the list.

For $k = 3$ and $(\ell, m) = (2, 2)$, it is easy to see we get $m = 2$ lists: $[1, 2, 3, 3]$ and $[2, 1, 3, 3]$. Finally, for $k = 2$, we just have one list $[1, 1, \dots, 1, 2, 2, \dots, 2]$, where there are ℓ 1's and m 2's. This proves that $\{\alpha^i\tau : i \ge 0\} = (k, \ell, m)G$ in all cases, and thus completes the proof of the lemma. $\square$

This concludes our excursion into the world of DFAOs. Although most of this chapter was occupied by a challenging lower bound proof, we learned a useful fact about reversal that applies just as well to ordinary DFAs: the state complexity of reversal is completely determined by the transition monoid and the output map (or final state set in the case of DFAs). We also learned an alternative reverse automaton construction that is based on function composition, rather than the intuitive method of reversing the transitions. This gives us a new way of understanding DFA reversal, though it remains to be seen whether this new perspective is useful for proofs.

Chapter 7

Revisiting State Complexity of Regular Languages

7.1 Applying the OLPA Approach

Armed with our new techniques, we revisit the four operational state complexity problems covered in Section 2.4.5: reversal, concatenation, star and boolean operations. Throughout this chapter, we consider only *restricted state complexity* and thus assume that the operands of multiary operations are all over the same alphabet. This is because we have not developed a version of our OLPA theory that applies to unrestricted state complexity. While some of our other techniques can be applied to unrestricted state complexity problems, the OLPA approach is the cornerstone of our methods.

Reversal. Using OLPA witnesses, we give two short proofs that the state complexity of the reversal operation is 2^n, The fact that 2^n is an upper bound is clear from the construction of the reverse automaton, so we simply prove tightness of the bound.

Proof 1. Fix a non-empty proper subset F of $\{1,\dots,n\}$ and let $\mathcal{D}$ be the standard DFA for the full transformation language $T_n(1,F)$. The transition monoid of the DFA is T_n. By Corollary 6.3.3, the state complexity of the reverse $(T_n(1,F))^R$ is $|T_n\chi_F|$. We claim that $T_n\chi_F$ contains all 2^n functions $f\colon \{1,\dots,n\} \to \{0,1\}$. Indeed, since F is non-empty and proper, there exist states p and q of $\mathcal{D}$ such that $p\chi_F = 0$ and $q\chi_F = 1$. To construct $f\colon \{1,\dots,n\} \to \{0,1\}$, choose a transformation $t \in T_n$ that maps $i \in \{1,\dots,n\}$ to p if $if = 0$, and maps i to q if $if = 1$. Then $t\chi_F = f$. □

Proof 2. Now we give a more direct proof that does not rely on the developments of Chapter

6. Fix a non-empty proper subset F of $\{1,\dots,n\}$ and let $\mathcal{D} = (Q,\Sigma,T,1,F)$ be the standard DFA for the full transformation language $T_n(1,F)$. Let $\mathcal{R}$ be the corresponding reverse automaton. The transition relation $T^{\mathcal{R}}$ of $\mathcal{R}$ satisfies $ST_a^{\mathcal{R}} = \{q \in Q : qT_a \in S\}$; it maps a set S to the set of all states with transitions leading into S. The initial state of $\mathcal{R}$ is the set F. For each $S \subseteq Q$, let $t \in T_n$ be a function that maps S into F and $Q \setminus S$ into $Q \setminus F$; this is possible since F is non-empty and proper, so F and $Q \setminus F$ are both non-empty. Then $FT_t^{\mathcal{R}} = S$, and it follows that all 2^n states of $\mathcal{R}$ are reachable. By Lemma 2.4.24, all states of $\mathcal{R}$ are pairwise distinguishable. □

Star. Next, we apply OLPA witnesses to the star operation. In the case of reversal, the upper bound 2^n was independent of the number of final states in the input DFA for the operation. However, for star, the number of final states matters; the absolute upper bound $2^{n-1} + 2^{n-2}$ can only be reached if the input DFA has exactly one final state which is non-initial. We will establish more general upper bound formulas for star which account for the number of final states and whether or not the initial state is final.

Theorem 7.1.1. *Let L be a regular language recognized by $\mathcal{D} = (Q,\Sigma,T,1,F)$, where $|Q| = n$ and $|F| = k$. Suppose $1 \le |F| \le n-1$. If $F = \{1\}$ then $L = L^*$, and otherwise we have the following tight upper bounds on* $\mathrm{sc}(L^*)$*:*

$$\mathrm{sc}(L^*) \le \begin{cases} 2^{n-1} + 2^{n-k-1}, & \text{if } 1 \notin F; \\ 2^{n-1} + 2^{n-k} - 1, & \text{if } 1 \in F \text{ and } |F| \ge 2. \end{cases}$$

Proof. By Theorem 3.3.7, it suffices to just compute the state complexity of L^* for $L \in \{T_n(1,F_{n,k,j}) : 0 \le j \le 1, 0+j \le k \le n-1+j\}$ and show that the stated bounds are attained. Indeed, we know that these particular witnesses are guaranteed to maximize the state complexity, so whatever state complexity values we obtain for these languages will be tight upper bounds.

Fix L in the aforementioned set of witnesses and let $\mathcal{D} = (Q,T_n,T,1,F_{n,k,j})$ be the standard DFA for L. Let $\mathcal{S}$ be the star automaton corresponding to $\mathcal{D}$. Henceforth we assume that $F \ne \{1\}$, since otherwise $\mathcal{S}$ just recognizes L.

We show each non-empty set $S \subseteq Q$ in $\mathcal{S}$ is reachable by induction on $|S|$. Since we are working with OLPA witnesses, we are free to use whatever transformations we want in this proof. From $\{s\}$ we reach $\{q\}$ for each $q \in Q$ by a transformation that sends 1 to q. Now suppose $|S| \ge 2$ and smaller sets are reachable. Choose a set X of size $|S| - 1$ which contains a final state but does not contain 1; this is possible since $F \ne \{1\}$. Fix $q \in S$ and choose a transformation t that maps X onto $S \setminus \{q\}$ and 1 to q; then t sends X to S. Thus all non-empty subsets of Q are reachable.

We show that sets in the following collection are pairwise distinguishable:

$$\{S \subseteq Q : S \neq \emptyset \text{ and } S \cap F = \emptyset\} \cup \{S \subseteq Q : 1 \in S \text{ and } S \cap F \neq \emptyset\}.$$

If a non-empty set S is not in this collection, it contains a final state but does not contain 1, and the set S is indistinguishable from $S \cup \{1\}$. To distinguish distinct sets S and X in this collection, choose an element q which appears (without loss of generality) in S but not in X, and apply a transformation t which maps q into F and $Q \setminus \{q\}$ into $Q \setminus F$. Note also that if $1 \notin F$ then $\{s\}$ is distinguishable from all states in this collection, but if $1 \in F$ then $\{s\}$ and $\{1\}$ are indistinguishable.

Now we count the number of states in this collection. If $1 \notin F$, this collection has size $(2^{n-k} - 1) + 2^{n-k-1}(2^k - 1) = 2^{n-1} + 2^{n-k-1} - 1$. The final state $\{s\}$ is distinguishable from other final states in this collection by a transformation that sends 1 to a non-final state and fixes all other elements; thus we add one more distinguishable state and reach the stated bound. If $1 \in F$, this collection has size $(2^{n-k} - 1) + 2^{n-k}(2^{k-1}) = 2^{n-1} + 2^{n-k} - 1$. Here, $\{s\}$ and $\{1\}$ are indistinguishable, so there is no extra state to add. □

Remark. In principle, we might be able simplify the above proof slightly by using the results of Section 5.4 for the reachability argument, rather than using induction. However, the OLPA approach makes the induction proof so short and simple that this is not really worth the effort.

Concatenation. Similarly to star, the state complexity of binary concatenation is influenced by the final state set of the left input. We will use the OLPA approach to establish a general upper bound for binary concatenation.

Theorem 7.1.2. *Let L' be recognized by $\mathcal{D}' = (Q', \Sigma', T', 1, F')$, and let L be recognized by $\mathcal{D} = (Q, \Sigma, T, 1, F)$, with $\Sigma' = \Sigma$. Let $|Q'| = m$, $|Q| = n$, $|F'| = j$, and $|F| = k$. Suppose $1 \leq |F'| \leq m$ and $1 \leq |F| \leq n - 1$. We have the following tight upper bounds on* $\text{sc}(L'L)$*:*

$$\text{sc}(L'L) \leq \begin{cases} (m-j)2^n + j2^{n-1}, & \text{if } 1 \notin F'; \\ (m-j)2^n + j2^{n-1} - (m-j), & \text{if } 1 \in F'. \end{cases}$$

Additionally, if $|F| = n$, we have $\text{sc}(L'L) = m - j + 1$.

Proof. First we deal with the special case of $|F| = n$. It is clear that two states (p', S) and (q', T) are indistinguishable if S and T are non-empty, since then they are both final and there is no way to escape to a non-final state. All states of this form collapse into one indistinguishability class. All that remains are states of the form $(q', \emptyset)$ with q' non-final (if q' is final, then the second component will be non-empty). These states are clearly distinguishable from each other. There are $m-j$ distinguishable states of the form $(q', \emptyset)$, plus the one large indistinguishability class, for a total of $m-j+1$.

Now assume $1 \leq |F| \leq n-1$. By Theorem 3.4.7, it suffices to compute the state complexity for the OLPA witnesses of the form $\mathbb{T}_1(1, F_{m,j,\ell_1})$ and $\mathbb{T}_2(1, F_{n,k,\ell_2})$, where the parameters m, n, j, k are as in the theorem statement, and the parameters ℓ_i for $1 \leq i \leq 2$ determine whether the initial state is final in the standard DFA of each witness.

Let $\mathcal{C}$ be the concatenation automaton for our two witnesses. If $1 \in F'$, the initial state of $\mathcal{C}$ is $(1, \{1\})$. We will use the techniques of Chapter 5 to show that the desired states are reachable. Let 1 be the focus, let $\{1\}$ be the base set, and let Q be the target set. Let a_q be a letter with action $(\mathrm{id}, (1, q))$; that is, letter a acts as the identity in the left DFA $\mathcal{D}'$ and acts as the cycle $(1, 2, \dots, n)$ in the right DFA $\mathcal{D}$. Then we have $(1, \{1\}) \xrightarrow{a_q} (1, \{1, q\})$, so $\{a_1, \dots, a_n\}$ is a construction set. It is complete by Theorem 5.2.8, since each a_i acts as a permutation on the target Q. Thus every state $(1, S)$ with $S \subseteq Q$ and $1 \in S$ is reachable. Via the action $((1, q'), \mathrm{id})$ we reach (q', S) for each $q' \in Q'$. To reach (q', S) with q' non-final, S non-empty, and $1 \notin S$, fix $s \in S$ and apply the action $(\mathrm{id}, (1 \to s))$.

Let us count the states we have reached. For each of the $m-j$ non-final states q', we reach (q', S) for arbitrary non-empty S; this gives $(m-j)(2^n-1) = (m-j)2^n - (m-j)$ states. For each of the j final states q', we reach (q', S) for all S containing 1; this gives $j2^{n-1}$ states. In total we reach $(m-j)2^n + j2^{n-1} - (m-j)$ states, as required.

We claim that if $1 \in F'$, we cannot reach any more states. Indeed, suppose (q', S) is reachable. If q' is final, then S must contain 1 by the nature of the concatenation transition relation, and we have already reached all such states. If q' is non-final, we have reached all states except those of the form $(q', \emptyset)$; but these are not reachable since the second component of the initial state is a non-empty set, and there is no action that can make this component into an empty set.

This establishes that in the case where $1 \in F'$, the number of reachable states matches the stated upper bound. If $1 \notin F'$, then the initial state is $(1, \emptyset)$, and via the action $((1, q'), \mathrm{id})$ we can reach $(q', \emptyset)$ for all non-final q'. So an additional $m-j$ states are reachable in this case, as expected. It is clear that no more states are reachable, so once again the number of reachable states matches the stated upper bound.

Now we consider distinguishability. Let (p', S) and (q', T) be distinct states of $\mathcal{C}$. If $S \neq T$, we define a transformation t that maps the elements of Q as follows:

- Map $S \cap T$ outside of F.

- One of $S \setminus T$ or $T \setminus S$ must be non-empty. Choose one of the two that is non-empty, and map it into F. Map the other one outside of F.
- Map everything remaining outside of F.

When we apply the action (id, t), the result is that either S or T is mapped to a set that intersects F, and the other is mapped entirely outside of F. This distinguishes the states.

If $S = T$, then we must have $p' \neq q'$. First, if $S = T = Q$, apply $(\mathrm{id}, (Q \to 1))$ to reach $(p', \{1\})$ and $(q', \{1\})$. Next, if p' and q' are both final, choose a non-final state r' and a final state s' and apply the action $((p', r')(q', s'), \mathrm{id})$. Now we have reached $(r', \{1\})$ and $(s', \{1\})$ where r' is non-final and s' is final. Finally, apply $(\mathrm{id}, (1, 2))$ to reach $(r', \{2\})$ and $(s', \{1, 2\})$. Now the second components are not equal, and we can use previous arguments.

We have shown the desired numbers of states are reachable and pairwise distinguishable, completing the proof. □

Boolean operations. Next we use the OLPA approach to derive a general bound for m-ary boolean operations. This proof is more complicated than the others in this section, but only because the result is very general.

It is somewhat tricky to state a tight upper bound for the worst-case state complexity of an arbitrary m-ary boolean operation, because the operation's result might not depend on all of its arguments. For example, if the inputs to a binary boolean operation have state complexity n_1 and n_2 respectively, the worst-case state complexity can be 1, n_1, n_2, or $n_1 n_2$, depending on which arguments (if any) are relevant to the result.

To state our upper bound, we introduce some notation. Given an m-ary boolean function $\beta\colon \{0,1\}^m \to \{0,1\}$, we define functions $\beta_j\colon \mathbb{N} \to \mathbb{N}$ for $1 \leq j \leq m$ as follows. If there exist two binary m-tuples $(b_1, \dots, b_m)$ and $(b'_1, \dots, b'_m)$ which differ only in the j-th bit (that is, $b_j \neq b'_j$ and $b_i = b'_i$ for all $i \neq j$) such that $(b_1, \dots, b_m)\beta \neq (b'_1, \dots, b'_m)\beta$, then we define $n\beta_j = n$ for all $n \in \mathbb{N}$. Otherwise, it must be the case that for all binary m-tuples, flipping the j-th bit does not change the result of β; in this case we define $n\beta_j = 1$ for all $n \in \mathbb{N}$. If β_j is the identity map, we say that β *depends on argument* j, and if β_j is the constant function sending everything to 1, we say that β *does not depend on argument* j.

Theorem 7.1.3. *Let β be an m-ary boolean operation. Let $(L_1, \dots, L_m)$ be regular languages, where L_j is recognized by $(Q_j, \Sigma, T_j, 1, F_j)$ for $1 \leq j \leq m$. Set $n_j = |Q_j|$. Then $\mathrm{sc}((L_1, \dots, L_m)\beta) \leq (n_1\beta_1) \cdots (n_m\beta_m)$ and this bound is tight.*

Proof. The m-ary direct product construction for boolean operations can be defined as follows: $\mathcal{B} = (Q, \Sigma, T, (1, \dots, 1), F)$ where the state set is $Q = Q_1 \times \cdots \times Q_m$, the transition relation is $T = \{((q_1, \dots, q_m), a, (q_1(T_1)_a, \dots, q_m(T_m)_a)) : (q_1, \dots, q_m) \in Q, a \in \Sigma\}$ and the final state set is $F = \{(q_1, \dots, q_m) \in Q : (q_1\chi_{F_1}, \dots, q_m\chi_{F_m}) = 1\}$. This construction gives an upper bound of $n_1 \cdots n_m$. To get a tighter bound, we must consider distinguishability. Consider the following set of states:

$$\{(q_1, \dots, q_m) \in Q : q_j = 1 \text{ whenever } \beta \text{ does not depend on argument } j\}.$$

There are precisely $(n_1\beta_1) \cdots (n_m\beta_m)$ states in this set, and we claim that every state lying outside this set is indistinguishable from a state within the set. To see this, fix a state $(q_1, \dots, q_m)$ which is not in the above set. Then there exists j such that $q_j \neq 1$ and β does not depend on argument j. We claim state $(q_1, \dots, q_m)$ is indistinguishable from $(q_1, \dots, q_{j-1}, 1, q_{j+1}, \dots, q_m)$. Indeed, if two states differ only in component j, and β does not depend on argument j, then either both states are final or both states are non-final. Also, starting from a pair of states which differ only in component j, we can only reach other pairs that differ only in component j. Thus there is no way to distinguish these states.

Now we show that the upper bound is tight. Our witnesses will be OLPA witnesses $L_j = \mathbb{T}_j(1, F_j)$, where $\mathbb{T} = T_{n_1} \times \cdots \times T_{n_m}$ and $F_j = \{1\}$ for $1 \leq j \leq m$ (it does not really matter what we choose for F_j, as long as for $n_j \geq 2$ it is a proper non-empty subset of $\{1, \dots, n_j\}$).

The initial state of the direct product DFA is $(1, \dots, 1)$. For each state $(q_1, \dots, q_m) \in Q$, let t_j be a transformation that sends 1 to q_j. Then the letter $(t_1, \dots, t_m) \in \mathbb{T}$ sends the initial state to $(q_1, \dots, q_m)$; thus all states are reachable.

Next we show that all pairs of states in $\{(q_1\beta_1, \dots, q_m\beta_m) : (q_1, \dots, q_m) \in Q\}$, which has size $(n_1\beta_1) \cdots (n_m\beta_m)$, are distinguishable. Suppose we have two states $(q_1\beta_1, \dots, q_m\beta_m)$ and $(q'_1\beta_1, \dots, q'_m\beta_m)$. Since they are distinct, they must differ in some component j, and for this j we must have $q_j\beta_j = q_j$ and $q'_j\beta_j = q'_j$. Hence there exist two binary m-tuples $(b_1, \dots, b_m)$ and $(b'_1, \dots, b'_m)$ which differ only in component j such that $(b_1, \dots, b_m)\beta \neq (b'_1, \dots, b'_m)\beta$. Assume without loss of generality that $(b_1, \dots, b_m)\beta = 1$ and $(b'_1, \dots, b'_m)\beta = 0$.

Now, choose a tuple of transformations $(t_1, \dots, t_m) \in \mathbb{T}$ as follows:

- Choose t_j so that $q_jt_j\chi_{F_j} = b_j$ and $q'_jt_j\chi_{F_j} = b'_j$.
- For $i \neq j$, if β depends on argument i, choose t_i so that $(q_it_i)\chi_{F_i} = b_i = b'_i$. If β does not depend on argument i, let t_i be the identity map.

Now, we claim that $(q_1\beta_1, \dots, q_m\beta_m)(t_1, \dots, t_m)$ is a final state. To determine whether the reached state $(q_1\beta_1 t_1, \dots, q_m\beta_m t_m)$ is final, we look at the following binary m-tuple: $(q_1\beta_1 t_1 \chi_{F_1}, \dots, q_m\beta_m t_m \chi_{F_m})$. If β depends on argument i (including the case $i = j$) then we have $q_i\beta_i t_i \chi_{F_i} = b_i$. If β does not depend on argument i, then $q_i\beta_i = 1$, transformation t_i is the identity map, and $F_i = \{1\}$, so we have $q_i\beta_i t_i \chi_{F_i} = 1$. Thus $(q_1\beta_1 t_1 \chi_{F_1}, \dots, q_m\beta_m t_m \chi_{F_m}) = (\widetilde{b_1}, \dots, \widetilde{b_m})$, where $\widetilde{b_i}$ is b_i if β depends on argument i, and $\widetilde{b_i}$ is 1 otherwise. Now, we know that if β does not depend on argument i, then flipping the i-th bit in a binary m-tuple will not change the result of applying β to that m-tuple. So by flipping bits if necessary, we see that

$$(\widetilde{b_1}, \dots, \widetilde{b_m})\beta = (b_1, \dots, b_m)\beta = 1.$$

Hence $(q_1\beta_1, \dots, q_m\beta_m)(t_1, \dots, t_m)$ is a final state.

On the other hand, consider the state $(q'_1\beta_1, \dots, q'_m\beta_m)(t_1, \dots, t_m)$. For this state we have $(q'_1\beta_1 t_1 \chi_{F_1}, \dots, q'_m\beta_m t_m \chi_{F_m}) = (\widetilde{b'_1}, \dots, \widetilde{b'_m})$, where $\widetilde{b'_i}$ is b'_i if β depends on argument i, and $\widetilde{b'_i}$ is 1 otherwise. Thus by the same bit-flipping argument, we have

$$(\widetilde{b'_1}, \dots, \widetilde{b'_m})\beta = (b'_1, \dots, b'_m)\beta = 0.$$

Thus this state is not final, and we have distinguished the two states. □

Minimal alphabets. Since the OLPA approach uses extremely large alphabets, we will finish off this section by demonstrating that we can maximize the state complexity of reversal, star, concatenation, and binary boolean operations using witnesses with only two letters.

We have actually already seen this for three of these four operations:

- In Theorem 5.4.7, we saw a two-letter witness for star.
- In Section 5.3, we saw a number of two-letter witnesses for concatenation.
- In Example 4.4.7, we saw a two-letter witness that works for all proper binary boolean operations.

It remains to consider reversal. To define our witness, we will use a generating set for the monoid V_n^1 of Definition 6.3.7. A similar witness appeared in [60].

Theorem 7.1.4. *Let $\mathcal{D} = (Q, \Sigma, T, 1, F)$ be a DFA with $Q = \{1, \dots, n\}$, actions $a = (1, 2, \dots, n)$ and $b = (2 \to 1)$, and final state set $F = \{2k : 1 \leq k \leq \lfloor n/2 \rfloor\}$. The state complexity of $L(\mathcal{D})^R$ is 2^n.*

Proof. We know from Section 6.3 that the transition monoid of $\mathcal{D}$ is V_n^1. It suffices to show that $|V_n^1 \chi_F| = 2^n$. For each function $f \colon \{1, \dots, n\} \to \{0, 1\}$, we must find a transformation $t \in V_n^1$ such that $t\chi_F = f$.

Recall that a transformation $t \colon Q \to Q$ belongs to V_n^1 if and only if it satisfies one of the following conditions:

1. There exists $i \geq 0$ such that $t = (1, 2, \dots, n)^i = a^i$.
2. There exist $i, j \in \{1, \dots, n\}$ such that $it = jt$ and $j \equiv i + 1 \pmod{n}$.

If $if = 1$, we need $it\chi_F = 1$, and so it must be even. Similarly, if $if = 0$, then it must be odd. Thus, we define t as follows:

- If $if = 1$ and i is even, then $it = i$.
- If $if = 1$ and i is odd, then $it \equiv i - 1 \pmod{n}$.
- If $if = 0$ and i is even, then $it \equiv i - 1 \pmod{n}$.
- If $if = 0$ and i is odd, then $it = i$.

We claim this transformation belongs to V_n^1. To see this, first assume that $if = jf$ for some i and j such that $j \equiv i + 1 \pmod{n}$. Assume without loss of generality that $if = 1$ and i is even. Then $jf = 1$ and j is odd. Thus $it = i$, and $jt \equiv j - 1 \equiv i \pmod{n}$, so $jt = i$. Thus t identifies i and j and it follows that t belongs to V_n^1. Now suppose that this assumption does not hold. Then if we write f in "list notation" as $[1f, 2f, \dots, nf]$, this list cannot contain two adjacent elements that are equal. Furthermore, $1f$ and nf cannot be equal. This implies that n is even and the list looks like either $[0, 1, 0, 1, \dots, 0, 1]$ or $[1, 0, 1, 0, \dots, 1, 0]$. A routine computation shows then that either $t = a^{-1} = a^{n-1}$ or $t = \text{id}$, and so t belongs to V_n^1. It is then clear that $t\chi_F = f$, completing the proof. $\square$

7.2 Future Work

To conclude the thesis, we discuss possible avenues for future research in the area of algebraic approaches to state complexity.

The OLPA approach. The OLPA approach seems to be extensible in a number of ways. In particular, it seems that it should be useful in certain subclasses of regular languages. For example, in the class of group languages, it seems that one could mimic the approach, but use the symmetric group to play the role of the full transformation monoid. Rather than marginal extensions like this, however, what would be ideal is a generic theorem that applies to arbitrary subclasses. The author has made some progress in this direction, but there are some difficulties in formulated such a general result. There are some subclasses where the OLPA approach technically "works" but is simply not useful because the number of OLPA witnesses that must be considered is too large. A clear division should be articulated between classes like the group languages, where the approach works just as well as for regular languages, and more badly-behaved classes where the approach is not useful. Another issue is that the notation becomes very complex and cumbersome when working in such generality.

It also seems feasible to extend the OLPA approach to unrestricted state complexity. This would likely require looking at *incomplete DFAs*, where the transition function is a partial function rather than a total function, since these DFAs are used in computing unrestricted state complexity.

Only a handful of "interesting" operations are known for which the OLPA approach does not work. It may be a fruitful project to attempt to find more of these operations, and to study the state complexity problems for these operations.

Boolean operations. When studying boolean operations, we mostly restricted our attention to group languages and other languages which contain a "significant" subgroup of permutations in their syntactic monoid. However, there are languages whose syntactic monoid contains *no* permutations! Thus it would be very interesting to investigate more general monoids and see if some of our results can be generalized, or if useful new results can be discovered.

Our results on boolean operations relied on the input DFAs being *dissimilar*. We only briefly investigated the case of similar DFAs. Can we say anything interesting about this case?

We considered only binary boolean operations. What can we say about boolean operations of higher arity?

Concatenation and star. We found two examples of concatenation witnesses for which our construction set technique is not useful. Can we find more examples? Is there perhaps a different general technique that applies to these examples?

We have done very little investigation on the efficacy of the construction set method for star. It would be useful to have a survey of its effectiveness on different witnesses from across the literature, similar to the one we conducted for concatenation.

Reversal and DFAOs. The problem of finding a tight upper bound for DFAO reversal in the binary alphabet case remains unsolved. In fact, we do not even know how to determine the largest 2-generated transformation monoid! To our knowledge, the results of Holzer and König [43] and Krawetz, Lawrence and Shallit [59] have not been improved, even though it has been over a decade. Solving the 2-generated monoid problem would not immediately solve the binary DFAO reversal problem, but the techniques used might bring us closer to a solution.

In the ordinary DFA case, we have not deeply investigated necessary and sufficient conditions for state complexity of reversal to be maximized. Considering that the state complexity of reversal depends heavily on the transition monoid, it seems likely that there could be nice algebraic conditions, similar to the ones we discovered for boolean operations.

Conclusion. The goal of this thesis was to impress upon the reader the usefulness of a monoid-theoretic viewpoint in studying state complexity, and outline some interesting algebraic techniques for simplifying proofs. There are many more operations, subclasses and problems out there to investigate and understand. I hope this work will prove useful and inspiring to other researchers.

– Sylvie

References

[1] J.-P. Allouche and J. Shallit. *Automatic Sequences: Theory, Applications, Generalizations.* Cambridge University Press, 2003.

[2] J. Bell, J. A. Brzozowski, N. Moreira, and R. Reis. Symmetric groups and quotient complexity of boolean operations. In J. Esparza and et al., editors, *ICALP 2014*, volume 8573 of *LNCS*, pages 1–12. Springer, 2014.

[3] J.-C. Birget. Intersection and union of regular languages and state complexity. *Inform. Process. Lett.*, 43(4):185–190, 1992.

[4] J. A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.

[5] J. A. Brzozowski. Quotient complexity of regular languages. *J. Autom. Lang. Comb.*, 15(1/2):71–89, 2010.

[6] J. A. Brzozowski. In search of most complex regular languages. *Int. J. Found. Comput. Sc.*, 24(06):691–708, 2013.

[7] J. A. Brzozowski. Unrestricted state complexity of binary operations on regular languages. In C. Câmpeanu, F. Manea, and J. Shallit, editors, *DCFS 2016*, volume 9777 of *LNCS*, pages 60–72. Springer, 2016.

[8] J. A. Brzozowski. Towards a theory of complexity of regular languages. *J. Autom. Lang. Comb.*, 23(1–3):67–101, 2018.

[9] J. A. Brzozowski and S. Davies. Most complex non-returning regular languages. In G. Pighizzini and C. Câmpeanu, editors, *DCFS 2017*, pages 89–101. Springer, 2017.

[10] J. A. Brzozowski, S. Davies, and B. Y. V. Liu. Most complex regular ideal languages. *Discrete Math. Theoret. Comput. Sc.*, 18(3), 2016. Paper #15.

[11] J. A. Brzozowski, G. Jirásková, and B. Li. Quotient complexity of ideal languages. *Theoret. Comput. Sci.*, 470:36–52, 2013.

[12] J. A. Brzozowski, G. Jirásková, B. Li, and J. Smith. Quotient complexity of bifix-, factor-, and subword-free regular languages. *Acta Cybernet.*, 21(4):507–527, 2014.

[13] J. A. Brzozowski, G. Jirásková, B. Liu, A. Rajasekaran, and M. Szykuła. On the state complexity of the shuffle of regular languages. In C. Câmpeanu, F. Manea, and J. Shallit, editors, *DCFS 2016*, pages 73–86. Springer, 2016.

[14] J. A. Brzozowski, G. Jirásková, and C. Zou. Quotient complexity of closed languages. *Theory Comput. Syst.*, 54:277–292, 2014.

[15] J. A. Brzozowski and B. Liu. Quotient complexity of star-free languages. *Int. J. Found. Comput. Sc.*, 23(06):1261–1276, 2012.

[16] J. A. Brzozowski and C. Sinnamon. Complexity of left-ideal, suffix-closed and suffix-free regular languages. In F. Drewes, C. Martín-Vide, and B. Truthe, editors, *LATA 2017*, pages 171–182. Springer, 2017.

[17] J. A. Brzozowski and C. Sinnamon. Complexity of proper prefix-convex regular languages. In A. Carayol and C. Nicaud, editors, *CIAA 2017*, pages 52–63. Springer, 2017.

[18] J. A. Brzozowski and C. Sinnamon. Complexity of right-ideal, prefix-closed, and prefix-free regular languages. *Acta Cybernet.*, 23(1):9–41, 2017.

[19] J. A. Brzozowski and C. Sinnamon. Unrestricted state complexity of binary operations on regular and ideal languages. *J. Autom. Lang. Comb.*, 22(1–3):29–59, 2017.

[20] J. A. Brzozowski and M. Szykłua. Complexity of suffix-free regular languages. *J. Comput. System Sc.*, 89:270–287, 2017.

[21] P. J. Cameron. Finite permutation groups and finite simple groups. *Bull. London Math. Soc*, 13(1):1–22, 1981.

[22] C. Câmpeanu, K. Culik, K. Salomaa, and S. Yu. State complexity of basic operations on finite languages. In O. Boldt and H. Jürgensen, editors, *WIA 1999*, pages 60–70. Springer, 2001.

[23] C. Câmpeanu, K. Salomaa, and S. Yu. Tight lower bound for the state complexity of shuffle of regular languages. *J. Autom. Lang. Comb.*, 7(3):303–310, 2002.

[24] P. Caron, E. Hamel-De le court, J.-G. Luque, and B. Patrou. New tools for state complexity. *CoRR*, abs/1807.00663, 2018.

[25] R. Cmorik and G. Jirásková. Basic operations on binary suffix-free languages. In Z. Kotásek and et al., editors, *MEMICS 2012*, pages 94–102, 2012.

[26] S. Davies. A general approach to state complexity of operations: Formalization and limitations. *CoRR*, abs/1806.08476, 2018.

[27] M. Delgado, S. Linton, and J. Morais. Automata – a GAP package, Version 1.13, 19 November 2011.

[28] J. D. Dixon and B. Mortimer. *Permutation Groups*. Springer, 1996.

[29] M. Domaratzki. State complexity of proportional removals. *J. Autom. Lang. Comb.*, 7(4):455–468, 2002.

[30] M. Domaratzki. Deletion along trajectories. *Theoret. Comput. Sci.*, 320(2):293–313, 2004.

[31] M. Domaratzki and A. Okhotin. State complexity of power. *Theoret. Comput. Sci.*, 410(24):2377–2392, 2009.

[32] M. Domaratzki and K. Salomaa. State complexity of shuffle on trajectories. *J. Autom. Lang. Comb.*, 9:217–232, 2004.

[33] D. Dummit and R. Foote. *Abstract Algebra*. Wiley, 2004.

[34] H.-S. Eom, Y.-S. Han, and G. Jirásková. State complexity of basic operations on non-returning regular languages. *Fund. Inform.*, 144:161–182, 2016.

[35] Z. Ésik, Y. Gao, G. Liu, and S. Yu. Estimation of state complexity of combined operations. *Theoret. Comput. Sci.*, 410(35):3272–3280, 2009.

[36] R. Ferens and M. Szykuła. Complexity of bifix-free regular languages. In A. Carayol and C. Nicaud, editors, *CIAA 2017*, pages 76–88. Springer, 2017.

[37] Y. Gao, N. Moreira, R. Reis, and S. Yu. A survey on operational state complexity. *J. Autom. Lang. Comb.*, 21(4):251–310, 2016.

[38] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.9.2*, 2018.

[39] V. Geffert. Magic numbers in the state hierarchy of finite automata. *Inform. Comput.*, 205(11):1652–1670, 2007.

[40] Y.-S. Han, S.-K. Ko, and K. Salomaa. State complexity of deletion and bipolar deletion. *Acta Informatica*, 53(1):67–85, 2016.

[41] Y.-S. Han and K. Salomaa. State complexity of basic operations on suffix-free regular languages. *Theoret. Comput. Sci.*, 410(27-29):2537–2548, 2009.

[42] Y.-S. Han, K. Salomaa, and D. Wood. Operational state complexity of prefix-free regular languages. In Z. Ésik and Z. Fülöp, editors, *Automata, Formal Languages, and Related Topics*, pages 99–115. Institute of Informatics, University of Szeged, Hungary, 2009.

[43] M. Holzer and B. König. On deterministic finite automata and syntactic monoid size. *Theoret. Comput. Sci.*, 327(3):319–347, 2004.

[44] M. Hricko, G. Jirásková, and A. Szabari. Union and intersection of regular languages and descriptional complexity. In *DCFS 2005*, pages 170–181, 2005.

[45] J. Jirásek and G. Jirásková. The exact complexity of star-complement-star. In C. Câmpeanu, editor, *CIAA 2018*, pages 223–235. Springer, 2018.

[46] G. Jirásková. State complexity of some operations on binary regular languages. *Theoret. Comput. Sci.*, 330(2):287–298, 2005. DCFS 2005.

[47] G. Jirásková. On the state complexity of complements, stars, and reversals of regular languages. In M. Ito and M. Toyama, editors, *DLT 2008*, pages 431–442. Springer, 2008.

[48] G. Jirásková. Concatenation of regular languages and descriptional complexity. *Theory Comput. Syst.*, 49(2):306–318, 2011.

[49] G. Jirásková and M. Krausová. Complexity in prefix-free regular languages. In I. McQuillan, G. Pighizzini, and B. Trost, editors, *DCFS 2010*, pages 236–244. University of Saskatchewan, 2010.

[50] G. Jirásková and P. Olejár. State complexity of union and intersection of binary suffix-free languages. In H. Bordihn and et al., editors, *NCMA 2009*, pages 151–166. Austrian Computer Society, 2009.

[51] G. Jirásková, M. Palmovský, and J. Šebej. Kleene closure on regular and prefix-free languages. In M. Holzer and M. Kutrib, editors, *CIAA 2014*, pages 226–237. Springer, 2014.

[52] G. Jirásková and J. Shallit. The state complexity of star-complement-star. In H.-C. Yen and O. H. Ibarra, editors, *DLT 2012*, pages 380–391. Springer, 2012.

[53] G. Jirásková, A. Szabari, and J. Šebej. The complexity of languages resulting from the concatenation operation. In C. Câmpeanu, F. Manea, and J. Shallit, editors, *DCFS 2016*, pages 153–167. Springer, 2016.

[54] G. Jirásková and J. Šebej. Reversal of binary regular languages. *Theoret. Comput. Sci.*, 449:85–92, 2012.

[55] Jirásková, Galina and Okhotin, Alexander. State complexity of cyclic shift. *RAIRO-Theor. Inf. Appl.*, 42(2):335–360, 2008.

[56] S. C. Kleene. Representation of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, Annals of Mathematics Studies, no. 34, pages 3–41. Princeton University Press, 1956.

[57] M. Krausová. Prefix-free regular languages: Closure properties, difference, and left quotient. In Z. Kotásek, J. Bouda, I. Černá, L. Sekanina, T. Vojnar, and D. Antoš, editors, *MEMICS 2011*, pages 114–122, 2011.

[58] B. Krawetz. Monoids and the state complexity of root(L). Master's thesis, 'University of Waterloo, 2003. Available at https://cs.uwaterloo.ca/~shallit/krawetz.pdf.

[59] B. Krawetz, J. Lawrence, and J. Shallit. State complexity and the monoid of transformations of a finite set. *Internat. J. Found. Comput. Sci.*, 16(03):547–563, 2005.

[60] E. Leiss. Succinct representation of regular languages by boolean automata. *Theoret. Comput. Sci.*, 13:323–330, 2009.

[61] O. B. Lupanov. A comparison of two types of finite automata. *Problemy Kibernetiki*, 9:321–326 (Russian), 1963. German translation: Über den Vergleich zweier Typen endlicher Quellen. *Probleme der Kybernetik* **6** (1966), 328–335.

[62] A. N. Maslov. Estimates of the number of states of finite automata. *Dokl. Akad. Nauk SSSR*, 194:1266–1268 (Russian), 1970. English translation: Soviet Math. Dokl. **11** (1970) 1373–1375.

[63] A. Mateescu, G. Rozenberg, and A. Salomaa. Shuffle on trajectories: Syntactic constraints. *Theoret. Comput. Sci.*, 197(1):1–56, 1998.

[64] B. G. Mirkin. On dual automata. *Kibernetika (Kiev)*, 2:7–10 (Russian), 1966. English translation: Cybernetics **2**, (1966) 6–9.

[65] J. Myhill. Finite automata and representation of events. *Wright Air Development Center Technical Report*, 57–624, 1957.

[66] A. Nerode. Linear automaton transformations. *Proc. Amer. Math. Soc.*, 9:541–544, 1958.

[67] C. Nicaud. Average state complexity of operations on unary automata. In M. Kutyłowski, L. Pacholski, and T. Wierzbicki, editors, *MFCS 1999*, pages 231–240. Springer, 1999.

[68] G. Pighizzini and J. Shallit. Unary language operations, state complexity and Jacobsthal's function. *Internat. J. Found. Comput. Sci.*, 13(01):145–159, 2002.

[69] B. Ravikumar. Some applications of a technique of Sakoda and Sipser. *SIGACT News*, 21(4):73–77, 1990.

[70] A. Restivo and R. Vaglica. Extremal minimality conditions on automata. *Theor. Comput. Sci.*, 440–441:73–84, 2012.

[71] A. Restivo and R. Vaglica. A graph theoretic approach to automata minimality. *Theoret. Comput. Sci.*, 429:282–291, 2012.

[72] W. J. Sakoda and M. Sipser. Nondeterminism and the size of two way finite automata. In *STOC 1978*, pages 275–286. ACM, 1978.

[73] A. Salomaa, K. Salomaa, and S. Yu. State complexity of combined operations. *Theoret. Comput. Sci.*, 383(2):140–152, 2007.

[74] J. Šebej. Reversal on regular languages and descriptional complexity. In H. Jurgensen and R. Reis, editors, *DCFS 2013*, pages 265–276. Springer, 2013.

[75] C. Sinnamon. Complexity of proper suffix-convex regular languages. In C. Câmpeanu, editor, *CIAA 2018*, pages 324–338. Springer, 2018.

[76] B. Steinberg. A theory of transformation monoids: combinatorics and representation theory, 2010. Available at https://arxiv.org/abs/1004.2982.

[77] Q. Yan. Lower bounds for complementation of ω-automata via the full automata technique. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *ICALP 2006*, pages 589–600. Springer, 2006.

[78] S. Yu. State complexity of regular languages. *J. Autom. Lang. Comb.*, 6:221–234, 2001.

[79] S. Yu, Q. Zhuang, and K. Salomaa. The state complexities of some basic operations on regular languages. *Theor. Comput. Sci.*, 125(2):315–328, 1994.