11-14-2007

Algebraic Theory of Minimal Nondeterministic Finite Automata with Applications

Daniel S. Cazalis
Florida International University, dcazal01@cis.fiu.edu

DOI: 10.25148/etd.FI08081507

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

ALGEBRAIC THEORY OF MINIMAL NONDETERMINISTIC FINITE AUTOMATA WITH APPLICATIONS

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Daniel Cazalis

2007

To: Interim Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by Daniel Cazalis, and entitled Algebraic Theory of Minimal Nondeterministic Finite Automata with Applications, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Steven Hudson

Masoud Milani

Giri Narasimhan

Geoffrey Smith, Major Professor

Date of Defense: November 14, 2007

The dissertation of Daniel Cazalis is approved.

Interim Dean Amir Mirmiran
College of Engineering and Computing

Dean George Walker
University Graduate School

Florida International University, 2007

DEDICATION

To my family: Ivonne, Begone, Yolanda (2), Carolina, Camila, Vanesa *y* Asier.

ACKNOWLEDGMENTS

To my great advisor and friend Dr. Geoffrey Smith, without whose help this marvelous trip to the arid lands of Automata would not have been successful, to Dr. Giri Narasimhan and Dr. Masoud Milani for guidance, and to Dr. Steve Hudson for counseling.

ABSTRACT OF THE DISSERTATION

ALGEBRAIC THEORY OF MINIMAL NONDETERMINISTIC FINITE AUTOMATA WITH APPLICATIONS

by

Daniel Cazalis

Florida International University, 2007

Miami, Florida

Professor Geoffrey Smith, Major Professor

Since the 1950s, the theory of deterministic and nondeterministic finite automata (DFAs and NFAs, respectively) has been a cornerstone of theoretical computer science. In this dissertation, our main object of study is *minimal NFAs*. In contrast with minimal DFAs, minimal NFAs are computationally challenging: first, there can be more than one minimal NFA recognizing a given language; second, the problem of converting an NFA to a minimal equivalent NFA is NP-hard, even for NFAs over a unary alphabet. Our study is based on the development of two main theories, *inductive bases* and *partials*, which in combination form the foundation for an incremental algorithm, `ibas`, to find minimal NFAs.

An *inductive basis* is a collection of languages with the property that it can generate (through union) each of the left quotients of its elements. We prove a fundamental *characterization theorem* which says that a language can be recognized by an n-state NFA if and only if it can be generated by an n-element inductive basis. A *partial* is an incompletely-specified language. We say that an NFA recognizes a partial if its language *extends* the partial, meaning that the NFA's behavior is unconstrained on unspecified strings; it follows that a minimal NFA for a partial is also minimal for its language. We therefore direct our attention to minimal NFAs recognizing a given partial. Combining inductive bases and partials, we generalize

our characterization theorem, showing that a partial can be recognized by an n-state NFA if and only if it can be generated by an n-element partial inductive basis.

We apply our theory to develop and implement `ibas`, an incremental algorithm that finds minimal partial inductive bases generating a given partial. In the case of unary languages, `ibas` can often find minimal NFAs of up to 10 states in about an hour of computing time; with brute-force search this would require many trillions of years.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

Finite automata are among the simplest mathematical models of computation, and since the 1940s their theory has been one of the cornerstones of theoretical computer science. A finite automaton reads an input string, symbol by symbol, moving among a finite set of *states* in response to the symbols that it reads. It begins in an *initial state* and, upon reaching the end of the input, it *accepts* the input if it is in one of its *final* states; otherwise it *rejects* the input. The *language* recognized by an automaton is the set of all the strings that it accepts; a language is *regular* if it is recognized by some finite automaton. The seminal 1959 paper by Rabin and Scott [RS59] introduced *nondeterministic finite automata*, in which the state transitions are not necessarily uniquely determined by the current state and input symbol; instead the automaton is imagined to have a magical ability to "guess" the correct transition at each step. Remarkably, Rabin and Scott proved that every nondeterministic finite automaton is equivalent to a deterministic finite automaton; however, the number of states in the deterministic automaton might be exponentially larger.

This result leads naturally to the question of the minimum number of states required for a finite automaton to recognize a given language. With respect to *deterministic finite automata* (DFAs), the Myhill-Nerode theorem [Ner58] shows that each regular language has a *unique* minimal DFA, up to isomorphism. Moreover, this minimal DFA can be computed efficiently: Hopcroft [Hop71] showed that an n-state DFA can be converted into the minimal equivalent DFA in time $O(n \log n)$. (More recently, some simpler implementations of Hopcroft's algorithm have been developed [Blu94]. Also, it has been shown that if the size m of the input alphabet is

not viewed as fixed, then the original algorithm does not run in time $O(mn \log n)$, as was commonly believed in the literature; however some recent variations do achieve that running time [Knu01].)

When we turn our attention to the minimization of nondeterministic finite automata (NFAs), the situation is much more challenging. First, there can be more than one minimal NFA recognizing a given language. Second, the problem of converting an NFA to a minimal equivalent NFA is NP-hard, even for NFAs over a unary alphabet [SM73, JR93, Gra03]. These negative results also apply to *unambiguous finite automata* (UFAs), which are NFAs with the property that each accepted string has just one accepting path [JR93]. It has been proved that even the slightest extension of the deterministic model, such as allowing just one nondeterministic move in every accepting computation, or having two initial states, results in intractable minimization problems [Mal04].

Such negative results have led to efforts, such as [MP95, Koz97, IY02], that are aimed at *reducing the size* of an NFA without necessarily finding a *minimal* NFA. Also results have been shown on the difficulty of even *approximating* the size of a minimal NFA [Gra03, GS05].

In this dissertation, we develop theoretical foundations and algebraic techniques to enable searches for minimal NFAs recognizing a specified language. In the case of unary languages, our techniques often allow us to find minimal NFAs of up to 10 states in about an hour of computing time; with brute-force search this would require many trillions of years.

Considering the continuous interest and complexity of the study of NFA minimization, an enterprise like ours, which is general, methodologically unique, and theoretically sound, will be a modest addition to this important field.

1.2 Dissertation Overview

Our main object of study is *minimal nondeterministic finite automata*. Our study is based on the development of two main theories: *inductive bases* and *partials*. Combining these theories, we develop and implement an incremental algorithm, `ibas`, to find minimal NFAs.

The scope of our theoretical work is wider than the `ibas` algorithm. Especially we feel that our *characterization theorems* (Theorem 3.1.7, Corollary 5.4.3, and Theorem 6.2.3) are significant. Characterization theorems are among the most powerful mathematical tools, because they allow the use of specific properties of the new representation in addition to the properties of original objects. In our case, our two main characterization theorems allow us to treat *languages* as *partials* and *NFAs* as *partial inductive bases*, taking good advantage of the properties of these novel representations.

Lexicographical order is important throughout our presentation. By ordering the strings in Σ^*, we enable the representation of a language as an infinite (0,1) *property vector*. Moreover, these property vectors can themselves be lexicographically ordered, allow us to put *bases*, which are finite collections of languages, into a canonical order—this is quite important for eliminating permutations.

We now outline each of the chapters of this dissertation.

Chapter 2 discusses preliminary concepts of formal languages and NFAs, including the *left quotient* operation, $a\backslash L$. Of particular importance is the notion of the *prophecy* of a state q of an NFA M, which we denote $Proph(q)$. This is the set of all strings that M accepts if started in state q. Thus if M chooses to enter q, we can view it as making the "prophecy" that the rest of the input will be in $Proph(q)$.

Chapter 3 discusses *bases*, which are collections of languages. A basis $\mathcal{B}$ is said to *generate* a language L if some subcollection of $\mathcal{B}$ has union L. Bases will often be represented as (0,1) matrices in which the rows are the property vectors of the languages in the basis. We introduce a fundamentally important type of basis, which we call an *inductive basis*, which has the property that it generates each of the left quotients of its elements. We show a fundamental characterization theorem, which says that a collection of languages can be the prophecies of the states of an NFA iff the collection is an inductive basis; thus a language can be recognized by an n-state NFA iff it can be generated by an n-element inductive basis. Chapter 3 also explores a weaker notion of *functional basis* and shows that functional bases have interesting connections to inductive bases.

Chapter 4 introduces *partials*, which are partially-specified languages. Formally a partial P is a function from Σ^* into the flat domain $Bool_\perp = \{True, False, \perp\}$. Having generalized the logical operators to this domain, we generalize language operations (like union and quotient) to partials, and prove various properties of our generalizations. We also introduce the important concept of *extension* to say that one partial is more specified than another; our notation for this is $P \sqsubseteq P'$.

Chapter 5 considers partials and NFAs. We say that an NFA M recognizes a partial P if its language extends P; this means that M is unconstrained on strings that P maps to $\perp$. A nice consequence is that if an NFA M is minimal for some partial P, then M must also be minimal for its language, $L(M)$; the reason is that a smaller NFA recognizing $L(M)$ would also be a smaller NFA recognizing P. We also establish the existence of the *canonical partial* for each regular language L, which is the "narrowest" partial P with the property that any minimal NFA recognizing P also recognizes L.

Chapter 6 combines inductive bases and partials to define *partial inductive bases*. We generalize our characterization theorem from Chapter 3, proving that a partial can be recognized by an n-state NFA iff it can be generated by an n-element partial inductive basis. We also generalize the notion of extension to bases, and explore the question of how a partial inductive basis can be extended to a wider one; this theory allows partial inductive bases to be constructed incrementally.

Chapter 7 presents our `ibas` algorithm, which searches for minimal NFAs recognizing a given partial P by incrementally searching for partial inductive bases generating P. The algorithm can be described as a depth-first search on a tree whose nodes represent classes of NFAs recognizing prefixes of P.

Chapter 8 gives some experimental results to show the capabilities of `ibas`. We find that, over unary languages, exhaustive search of 6-state NFAs takes over 4 hours, and exhaustive search of 7-state NFAs would take years. In contrast, we find that `ibas` is often (but not always) able to complete searches for 10-state unary NFAs in about an hour.

Chapter 9 presents some earlier work that we did prior to the development of inductive bases. Using something called the *characterization matrix* of a language L, previously studied by Condon, Hellerstein, Pottle, and Wigderson [CHPW98], we revisit some traditional theorems and constructions of automata theory from the point of view of the characterization matrix and its factorization. In this way we are able to give a simpler and more unified view of topics including the Myhill-Nerode theorem, the powerset construction, bideterminism, and lower bounds on the size of minimal NFAs.

Chapter 10 concludes and presents some future directions.

1.3 Other Related Work

Our theory imposes lattice-like structures on languages and NFAs. Lattice structure is also associated with *fuzzy sets* and *fuzzy automata*, but the lattice structure is quite different, because fuzzy automata are based on the idea of "fuzzily" accepting an input; in contrast we always totally accept or reject, but on some inputs we do not care which. Some variations of *fuzzy automata* to be used in lattice-valued regular languages are presented as *deterministic lattice automata*, and their minimization is studied in [LP07] and [LL07]. Some algebraic properties are presented in [Pet06]. A minimization algorithm in fuzzy automata is presented in [CM04]; similar results are in [Pee88], whose results are also valid for stochastic automata. Also [MMS99] minimizes fuzzy automata. Fuzzy automata applications in natural languages and speech recognition are presented [T94].

Another line of work involves generalizations of NFAs, such as *generalized automata* [GM99] and the closely-related *expression automata* [HW04]; generalized automata use *words* for transitions and expression automata use *regular expressions*. Other generalizations of NFAs deal with the set operation used to generalize the transition from the arrows; the most commonly-used *union* operation can be replaced with any well-defined set operation. Symmetric difference is an interesting case and leads to small automata for some languages. An example of an application using NFAs based on symmetric difference is [vZ04].

There are other types of automata that are aimed at reducing the descriptional complexity (a type of measurement on the size); one example is the *conjunctive acceptance criterion* that may greatly reduce the size of NFAs, and there are other acceptance criteria that rely on the whole computation of the string in order to decide whether to accept [HK05].

Other work involves studies of special forms of NFAs for which minimization is easier. One example is *canonical automata* where the minimal machine is a subautomaton and the minimal NFA can often be obtained quickly using heuristics [Mat]. Another case concerns *acyclic automata* for which minimization can be done in linear time [Rev92, Wat03]. Similarly, minimization can be done efficiently in the case of NFAs recognizing ω *languages* [Sta83].

Another line of work investigates the blow-up associated with concatenation and complementation; tight upper bounds are known [JJZ04].

Cover automata are similar to our notion of an NFA accepting a partial, but cover automata are deterministic and are applied to finite languages only; incremental algorithms for constructing minimal cover automata are given in [CCY01, CCS06], but they do not use the bases approach that we do.

Some papers model states as *grids* over a (0,1) matrix; this method has also been called *tiling*. We use similar techniques in our study of bideterminism in Section 9.7. The *reduced automaton matrix* RAM in [KW06] deals with minimization of NFAs by the use of grids on matrices.

To the best of our knowledge, there is no work in the literature that closely resembles our methods and results.

CHAPTER 2

PRELIMINARIES

In this chapter we establish some notation and initial definitions; for the most part we follow traditional references such as [HU79]. We give a formal definition of *alphabets*, *languages*, *nondeterministic finite automata* (*NFAs*), *history languages*, *prophecy languages*, and *quotient languages*; these are our basic tools. We also give a description of the graph representation of NFAs for a less formal intuitive approach.

In addition, we extend the ordering of the alphabet to get a *lexicographical order* on strings, languages, states and NFAs; this allows us to give a numerical position of any string, to represent languages as $(0, 1)$ property vectors, to give a canonical order to the states of an NFA, and ultimately to order NFAs.

2.1 Alphabets, Languages, and NFAs

An *alphabet* Σ is a finite set. Given an alphabet Σ, we let Σ^* denote the set of all finite-length strings of elements of Σ. We denote the *empty string* by ϵ, the *concatenation* of strings x and y by xy, and the *reverse* of string x by x^R. A *language* L *over* Σ is simply a subset of Σ^*. We write L^R for the *reverse* of L, formed by reversing each of the strings in L. We write AB for the *concatenation* of languages A and B, consisting of the set of all strings that can be formed by concatenating a string from A with a string from B. Also, assuming that Σ is understood, we write $\overline{A}$ for the *complement* of A, consisting of all the strings in Σ^* but not in A.

Unlike [HU79], but like [RS59], [Per90], and [Koz97], we allow our NFAs to have multiple start states and we do not allow ϵ-transitions:

Definition 2.1.1 *An NFA M is a tuple $(Q, \Sigma, \delta, I, F)$ where*

- *Q is a finite set (the states),*
- *Σ is a finite set (the alphabet),*

- $\delta : Q \times \Sigma \to 2^Q$ *(the transition function),*
- $I \subseteq Q$ *(the set of initial states), and*
- $F \subseteq Q$ *(the set of final states).*

As usual, we define the semantics of NFAs by extending δ inductively to $\widehat{\delta}$: $2^Q \times \Sigma^* \to 2^Q$ (the generalized transition function).

$$\widehat{\delta}(S, \epsilon) = S$$

$$\widehat{\delta}(S, xa) = \bigcup_{q \in \widehat{\delta}(S,x)} \delta(q, a).$$

The *language recognized by* M, denoted $L(M)$, is $\{w \in \Sigma^* \mid \widehat{\delta}(I, w) \cap F \neq \emptyset\}$. If $w \in L(M)$, we say that M *accepts* w.

We remark that although the generalized transition function $\widehat{\delta}$ is defined for *every* subset of Q, in some sense this is not necessary, since the only subsets of Q that actually arise when running M are the sets $\widehat{\delta}(I, w)$, for some $w \in \Sigma^*$. For example, in the extreme case when $I = \emptyset$, the only set that arises is $\emptyset$.

Our main concern in this dissertation is with NFAs that are *minimal* with respect to the number of states:

Definition 2.1.2 *NFA M is* minimal *if no NFA with fewer states than M recognizes $L(M)$.*

It is often clearer to represent an NFA as a *labeled directed graph* with a vertex for each state and an edge labeled a from vertex q to vertex r whenever $r \in \delta(q, a)$; we indicate initial states with arrows and final states with double circles. For example, Figure 2.1 shows the graphical representation of a 5-state NFA that recognizes the set of all strings of a's whose length is not a multiple of 6.

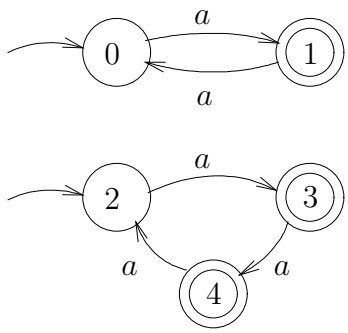

Figure 2.1: A 5-state NFA for $\{a^i \mid i \bmod 6 \neq 0\}$

The graphical representation of an NFA M allows us a more intuitive characterization of acceptance: M accepts string w iff there exists a path labeled with w from some initial state of M to some final state of M. (This path-based characterization of NFA acceptance fits better with the intuition of *angelic nondeterminism*, which views an NFA as having a magical "guessing" ability to make the right choice at every step.) For example, the NFA of Figure 2.1 accepts *aaaa* because there is a path labeled *aaaa* from initial state 2 to final state 3:

$$2 \xrightarrow{a} 3 \xrightarrow{a} 4 \xrightarrow{a} 2 \xrightarrow{a} 3$$

More precisely, we have the following characterization:

Lemma 2.1.3 *For all $S \subseteq Q$ and $w \in \Sigma^*$, $\widehat{\delta}(S, w)$ is the set of all states q such that there is a path labeled w from some state $s \in S$ to q.*

Proof. By induction on the length of w.

For the basis, $\widehat{\delta}(S, \epsilon) = S$, by definition. And a path from $s \in S$ labeled with ϵ has length 0, so it ends at s.

For the inductive step, we note that $t \in \widehat{\delta}(S, xa)$ iff $t \in \bigcup_{q \in \widehat{\delta}(S,x)} \delta(q, a)$ iff $t \in \delta(q, a)$, for some $q \in \widehat{\delta}(S, x)$. By induction, this holds iff $t \in \delta(q, a)$, for some q

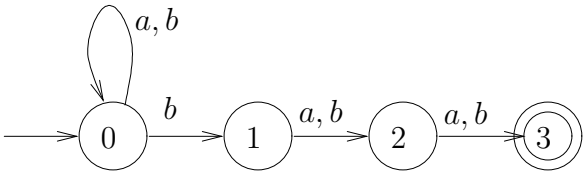

Figure 2.2: A UFA for $\{w \in \{a,b\}^* \mid$ the third-from-last symbol of w is $b\}$

such that there is a path labeled x from some state $s \in S$ to q. And this holds iff there is a path labeled xa from some state $s \in S$ to t. □

Corollary 2.1.4 *NFA M accepts w iff there is a path labeled w from some state in I to some state in F.*

Proof. By definition, M accepts w iff $\widehat{\delta}(I,w) \cap F \neq \emptyset$. By the lemma, this holds iff there is a path labeled w from some state in I to some state in F. □

We also consider two special kinds of NFAs, namely *deterministic finite automata* and *unambiguous finite automata*. A *deterministic finite automaton* (DFA) is an NFA such that $|I| \leq 1$ and $|\delta(q,a)| \leq 1$ for all $q \in Q$ and $a \in \Sigma$. Of course, this implies that $|\widehat{\delta}(I,w)| \leq 1$ for all $w \in \Sigma^*$. (Notice that we do not require our DFAs to include "dead" states; for example, under our definition we can recognize $\emptyset$ using a 0-state DFA.) Less well known, an *unambiguous finite automaton* (UFA) is an NFA in which each accepted string has just one accepting path [JR93]. Of course every DFA is a UFA. A more interesting UFA is the well-known $(n+1)$-state NFA that recognizes all strings over $\{a,b\}$ whose n^{th}-from-last symbol is b; note that the minimal DFA for this language requires 2^n states. The version of this UFA when $n = 3$ is shown in Figure 2.2.

Given an NFA $M = (Q, \Sigma, \delta, I, F)$ recognizing a language L, we can define a *reverse* NFA M^R that recognizes the reverse language L^R; we simply interchange I

and F and reverse all the arrows:

$$M^R = (Q, \Sigma, \delta^R, F, I),$$

where

$$\delta^R(q, a) = \{r \in Q \mid q \in \delta(r, a)\}.$$

Notice that for any NFA M, $(M^R)^R = M$. Also note that the reverse of a DFA need not be a DFA, but the reverse of a UFA is always a UFA.

2.2 Quotient Languages

Let L be a language over Σ and let $a \in \Sigma$.

Definition 2.2.1 *The right quotient of L by a, written L/a, is* $\{x \in \Sigma^* \mid xa \in L\}$.

Definition 2.2.2 *The left quotient of L by a, written $a\backslash L$, is* $\{x \in \Sigma^* \mid ax \in L\}$.

Quotient languages can be easily generalized to strings and languages:

Definition 2.2.3 *The right quotient of L by L', written L/L', is*

$$\{x \in \Sigma^* \mid \textit{there exists } y \in L' \textit{ such that } xy \in L\}.$$

2.3 History and Prophecy Languages

We will also find useful the concepts of *history* and *prophecy* of a state in an NFA M [ADN92]:

Definition 2.3.1 *For each state q in an NFA M, the* history *of q, denoted $Hist(q)$ is the set of all strings that can reach q from I:*

$$Hist(q) = \{w \in \Sigma^* \mid q \in \widehat{\delta}(I, w)\}.$$

Definition 2.3.2 *For each state q in an NFA M, the* prophecy *of q, denoted $Proph(q)$, is the set of all strings that can reach F from q:*

$$Proph(q) = \{w \in \Sigma^* \mid \widehat{\delta}(\{q\}, w) \cap F \neq \emptyset\}.$$

Equivalently,

$$Proph(q) = \{w \in \Sigma^* \mid q \in \widehat{\delta^R}(F, w^R)\}.$$

The name "prophecy" is appropriate, because when an NFA enters a state, it implicitly makes a prediction about what the rest of the input will be (assuming that the input is good). Note also that, in terms of the graphical representation, $Proph(q)$ is the set of all strings w such that there is a path labeled w from q to some final state.

As an example, the states of the NFA in Figure 2.1 have the following prophecies:

$$Proph(0) = \{a, a^3, a^5, a^7, a^9, \ldots\}$$
$$Proph(1) = \{\epsilon, a^2, a^4, a^6, a^8, \ldots\}$$
$$Proph(2) = \{a, a^2, a^4, a^5, a^7, \ldots\}$$
$$Proph(3) = \{\epsilon, a, a^3, a^4, a^6, \ldots\}$$
$$Proph(4) = \{\epsilon, a^2, a^3, a^5, a^6, \ldots\}$$

Notice that the history of q in M is the reverse of the prophecy of q in M^R. Also notice that a string w is accepted by NFA M iff it is in the prophecy of some initial state of M and iff it is in the history of some final state of M. That is,

$$L(M) = \bigcup_{q \in I} Proph(q)$$

and

$$L(M) = \bigcup_{q \in F} Hist(q).$$

Less obviously, we have the following theorem:

Theorem 2.3.3 *For any NFA* $M = (Q, \Sigma, \delta, I, F)$,

$$L(M) = \bigcup_{q \in Q} \mathit{Hist}(q)\mathit{Proph}(q).$$

Proof. Let $w \in \Sigma^*$. We have

$w \in L(M)$

iff there is a path labeled w from a state in I to a state in F

iff there exist $q \in Q$ and $x, y \in \Sigma^*$ such that $w = xy$ and there is a path labeled x from a state in I to q and a path labeled y from q to a state in F

iff there exist $q \in Q$ and $x, y \in \Sigma^*$ such that $w = xy$ and $x \in \mathit{Hist}(q)$ and $y \in \mathit{Proph}(q)$

iff there exists $q \in Q$ such that $w \in \mathit{Hist}(q)\mathit{Proph}(q)$

iff $w \in \bigcup_{q \in Q} \mathit{Hist}(q)\mathit{Proph}(q)$. □

2.4 Restricting the Generalized Transition Function $\widehat{\delta}$

Given an NFA M, suppose that we can run it on any string w and observe the set of states that can be reached. In other words, we can observe $\widehat{\delta}(I, w)$ for any $w \in \Sigma^*$. We might wonder whether this gives us enough information to "reverse engineer" the transition function δ.

Notice, however, that in this scenario we may not be able to observe the *entire* $\widehat{\delta}$ function, because we can only observe $\widehat{\delta}(S, w)$ if S is $\widehat{\delta}(I, x)$, for some $x \in \Sigma^*$.

This leads us to define a restricted function, which we denote as $\widehat{\delta_I}$, which is simply $\widehat{\delta}$ restricted to sets of the form $\widehat{\delta}(I, x)$, for some $x \in \Sigma^*$. For example, $\widehat{\delta_\emptyset}$ is defined only on $\emptyset$: $\widehat{\delta_\emptyset} : \{\emptyset\} \times \Sigma^* \to \{\emptyset\}$.

Dually, we can define $\widehat{\delta_F^R}$, which is $\widehat{\delta^R}$ restricted to sets of the form $\widehat{\delta^R}(F, x)$, for some $x \in \Sigma^*$.

As will be made clear in Chapter 3, $\widehat{\delta_I}$ is what we can see if we know the *histories* of the states of M, while $\widehat{\delta_F^R}$ is what we can see if we know the *prophecies* of the states of M.

2.5 Lexicographical Order

Given a total order on an alphabet Σ, we can define the total *lexicographic order* on Σ^*: we put shorter strings before longer strings, and we order strings of the same length based on the first position where they differ. For example, the lexicographic order on $\{a, b\}^*$ goes as follows:

$$\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, aaaa, \ldots$$

Definition 2.5.1 *Given $w \in \Sigma^*$, the* index of w*, denoted $index_\Sigma(w)$, is the 0-based position of w within Σ^*.*

For example, if $\Sigma = \{a, b\}$, then $index_\Sigma(abb) = 10$.

In view of the lexicographic order on Σ^*, we can represent each language L over Σ as an infinite (0, 1) property vector. These property vectors in turn induce a canonical order on languages over Σ:

Definition 2.5.2 *Given distinct languages L_1 and L_2 over Σ, we say that $L_1 < L_2$ if the smallest string (lexicographically) that belongs to one of them and not to the other belongs to L_2.*

Note that this makes the empty language, whose property vector is $(000\ldots)$, the smallest of all languages over Σ, and Σ^*, whose property vector is $(111\ldots)$, the

largest. Also note that all languages that contain the empty string ϵ are larger than those that do not.

The canonical order on languages also induces a canonical order on the states of an NFA, by ordering states based on their prophecies, assuming that no two states have the same prophecy:

$$q < q' \text{ iff } \mathit{Proph}(q) < \mathit{Proph}(q').$$

Finally, we can define a canonical ordering on NFAs over Σ. Let NFAs M and M' have canonically-ordered states $\{q_1, \ldots, q_m\}$ and $\{q'_1, \ldots, q'_n\}$, respectively. First, we say that $M < M'$ if $m < n$. And if $m = n$, we say that $M < M'$ if, letting i be the first index such that $\mathit{Proph}(q_i) \neq \mathit{Proph}(q'_i)$, we have $\mathit{Proph}(q_i) < \mathit{Proph}(q'_i)$. Notice however that this order is partial in the sense that two different NFAs with the same number of states and the same prophecies for all states are not comparable.

CHAPTER 3

INDUCTIVE BASES AND FUNCTIONAL BASES

As we saw in Chapter 2, an NFA M gives rise to two collections of languages, namely the histories of its states and the prophecies of its states. In this chapter, we study the collection of prophecies more closely. First we establish an important interrelationship among the prophecies of an NFA. Going in the opposite direction, we explore the question of when a collection of languages can be the set of prophecies of some NFA. We introduce the concept of an *inductive basis*, and show that a collection of languages can be the prophecies of some NFA iff it is an inductive basis. This result will allow us to search for minimal NFAs by searching for minimal inductive bases. This characterization will serve as the foundation of our work on algorithms for finding minimal NFAs. In this chapter, we also develop a related concept of *functional basis*, and establish some of its interesting properties.

3.1 From Prophecies to Inductive Bases

We begin with some useful terminology:

Definition 3.1.1 *A* basis $\mathcal{B}$ *is a finite multiset of languages over some alphabet* Σ. *We say that a basis* $\mathcal{B}$ generates *a language* L *if there is some subcollection of* $\mathcal{B}$ *whose union is* L.

Usually our bases will in fact be *sets* rather than multisets; after all, with respect to the question of whether $\mathcal{B}$ generates L, there is no reason for $\mathcal{B}$ to include elements of multiplicity greater than 1. However, we will be particularly interested in the basis consisting of the prophecies of the states of an NFA, and it is of course possible for two states to have the same prophecy—for this reason we allow bases to be multisets.

Definition 3.1.2 *Given NFA $M = (Q, \Sigma, \delta, I, F)$, $Prophecies(M)$ denotes the basis consisting of the prophecies of the states of M:*

$$Prophecies(M) = \{Proph(q) \mid q \in Q\}.$$

It turns out that $Prophecies(M)$ has some interesting properties as a basis. First, it generates $L(M)$, since $L(M) = \bigcup_{q \in I} Proph(q)$. More interestingly, it generates each of its *left quotients*:

Theorem 3.1.3 *For all $q \in Q$ and $a \in \Sigma$, $Prophecies(M)$ generates $a\backslash Proph(q)$.*

Proof. We show that $a\backslash Proph(q) = \bigcup_{r \in \delta(q,a)} Proph(r)$. For $w \in a\backslash Proph(q)$ iff $aw \in Proph(q)$ iff there is a path labeled aw from q to some final state iff there is there is a path labeled w from some state in $\delta(q, a)$ to some final state iff $w \in \bigcup_{r \in \delta(q,a)} Proph(r)$. □

Abstracting from this property, we are led to introduce what we call an *inductive basis*:

Definition 3.1.4 *A basis $\mathcal{B}$ over alphabet Σ is an* inductive basis *if for each $B \in \mathcal{B}$ and $a \in \Sigma$, $\mathcal{B}$ generates $a\backslash B$.*

(The name "inductive basis" is chosen to reflect the fact that if $\mathcal{B}$ generates a language C, then it also generates $a\backslash C$.) The theorem above can now be restated:

Theorem 3.1.5 *For any NFA M, $Prophecies(M)$ is an inductive basis.*

Conversely, given a basis $\mathcal{B}$ we may wonder whether there is any NFA that has $\mathcal{B}$ as its prophecies; we say that such an NFA *realizes* $\mathcal{B}$.

Definition 3.1.6 *We say that NFA M* realizes *basis $\mathcal{B}$ if $Prophecies(M) = \mathcal{B}$.*

Interestingly, it turns out that *any* inductive basis can be realized; thus we have a perfect characterization of when a basis can be the prophecies of some NFA. (This result appeared earlier in [Smi06].)

Theorem 3.1.7 *A basis $\mathcal{B}$ can be realized iff it is inductive.*

Proof. The "only if" direction was proved above.

For the "if" direction, let $\mathcal{B}$ be an inductive basis over alphabet Σ and let the elements of $\mathcal{B}$ be enumerated in some order:

$$\mathcal{B} = \{L_1, L_2, \ldots, L_n\}$$

where $n \geq 0$. (Note that the L_i's need not be distinct.)

Let $\mathbb{N}_n$ denote the set of natural numbers from 1 to n. We construct an NFA that realizes $\mathcal{B}$, using $\mathbb{N}_n$ as the set of states.[1] Define $M = (\mathbb{N}_n, \Sigma, \delta, I, F)$, where

- $\delta(i, a)$ is any subset of $\mathbb{N}_n$ such that $\bigcup_{j \in \delta(i,a)} L_j = a \backslash L_i$,
- I is arbitrary, and
- $F = \{i \in \mathbb{N}_n \mid \epsilon \in L_i\}$.

The fact that $\mathcal{B}$ is an inductive basis exactly ensures that δ can be constructed; notice however that δ is not necessarily uniquely determined. Next we prove that $\mathit{Prophecies}(M) = \mathcal{B}$ by showing that each state i has L_i as its prophecy:

Lemma 3.1.8 *For all $i \in \mathbb{N}_n$, $\mathit{Proph}(i) = L_i$.*

Proof. We show that $w \in \mathit{Proph}(i)$ iff $w \in L_i$ by induction on $|w|$:

Basis:

We have $\epsilon \in \mathit{Proph}(i)$ iff $i \in F$ iff $\epsilon \in L_i$.

[1] In the case where $\mathcal{B}$ is a *set*, rather than a *multiset*, a more elegant option is to use $\mathcal{B}$ itself as the set of states; this is how the proof is done in [Smi06].

Induction:

Given $a \in \Sigma$, $u \in \Sigma^*$, we have

$au \in \mathit{Proph}(i)$

iff $\exists k \in F$ such that there is a path from i to k labeled au

iff $\exists j \in \delta(i,a), \exists k \in F$ such that there is a path from j to k labeled u

iff $\exists j \in \delta(i,a)$ such that $u \in \mathit{Proph}(j)$

iff $\exists j \in \delta(i,a)$ such that $u \in L_j$ (by the induction hypothesis)

iff $u \in a\backslash L_i$ (by the definition of δ)

iff $au \in L_i$. □

□

We remark that the construction of M in the above proof can be seen as a generalization of a construction given years ago by Conway: theorem 2 of chapter 5 of [Con71] builds the minimal DFA for a regular language L using the repeated left quotients of L as states. This corresponds to an inductive basis $\mathcal{B}$ in which the subcollections used in generating the left quotients all have size 1.

Notice that the *language* recognized by the NFA M constructed in the above proof is not fixed, since its set I of initial states is arbitrary. In fact, by choosing I appropriately, we can make $L(M)$ be *any* language that is generated by the inductive basis $\mathcal{B}$. Thus we have the following corollary:

Corollary 3.1.9 *A language L can be recognized by an n-state NFA iff L can be generated by an n-element inductive basis.*

We now explore some simple consequences of Theorem 3.1.7.

Let $M = (Q, \Sigma, \delta, I, F)$ be an NFA. We know that $\mathit{Prophecies}(M)$ is an inductive basis. Now, $\mathit{Prophecies}(M)$ may contain some languages more than once, since two

states of M might have the same prophecy. But then M is not minimal: we can delete all duplicates from $\mathit{Prophecies}(M)$ to get a smaller inductive basis that still generates $L(M)$; this in turn gives us a smaller NFA that recognizes $L(M)$. Thus, in a minimal NFA, all of the state prophecies must be distinct.

More generally, no state in a minimal NFA M can have a prophecy that is equal to the union of the prophecies of some other states of M. For if a language L in an inductive basis $\mathcal{B}$ is the union of some other languages in $\mathcal{B}$, then $\mathcal{B} - \{L\}$ is still an inductive basis. (This follows from the fact that $a\backslash(L_1 \cup L_2) = (a\backslash L_1) \cup (a\backslash L_2)$.)

Theorem 3.1.7 also implies that all the elements of an inductive basis are regular languages.

Corollary 3.1.10 *If $\mathcal{B}$ is an inductive basis, then each element of $\mathcal{B}$ is regular.*

Proof. By Theorem 3.1.7, there is an NFA M such that $\mathit{Prophecies}(M) = \mathcal{B}$, where the set of initial states I of M can be chosen arbitrarily. If $L_i \in \mathcal{B}$, then we can choose $I = \{i\}$, making $L(M) = \mathit{Proph}(i) = L_i$. Hence L_i is regular. □

We can also define a notion of equivalence on NFAs:

Definition 3.1.11 *We say that NFAs M_1 and M_2 are equivalent iff $L(M_1) = L(M_2)$ and $\mathit{Prophecies}(M_1) = \mathit{Prophecies}(M_2)$.*

The number of equivalent NFAs can be exponential in the number of states. For example, consider the set of 6-state NFAs recognizing a^* and such that each state has prophecy a^*. Figure 3.1 gives two such NFAs. The number of such NFAs is greater than 2^{21}, but they all share the same inductive basis. This suggests that searching for inductive bases may be far easier than searching for NFAs.

Going forward, our plan is to search for minimal NFAs recognizing a given language L by searching for minimal inductive bases generating L. However, to make

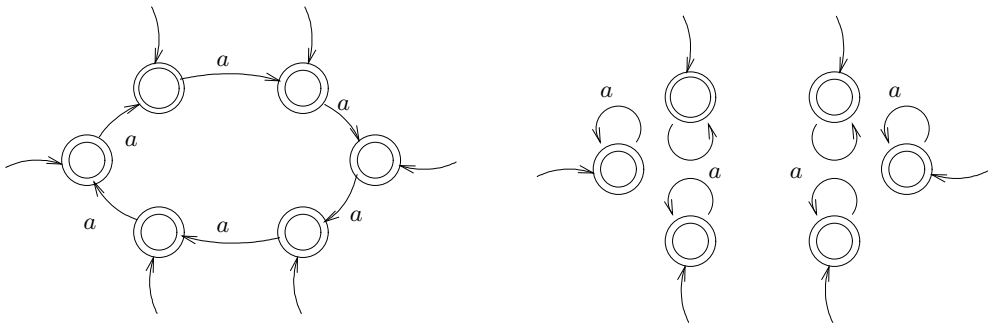

Figure 3.1: Two equivalent 6-state NFAs recognizing a^*

this plan algorithmically feasible, we will need a finite representation of the languages in the inductive basis. In the next chapter, we will introduce what we call *partials* to allow us to use finite approximations of potentially infinite languages.

But first we explore further the properties of the possible prophecies of an NFA.

3.2 Functional Bases

Given a basis $\mathcal{B}$, we are interested in knowing whether $\mathcal{B}$ can serve as the set of prophecies of an NFA. In the last section, we showed that this is possible iff $\mathcal{B}$ is an inductive basis. Here we approach the question in a different way.

Given a basis, let us pick some order for its elements: $\mathcal{B} = \{L_1, L_2, \ldots, L_n\}$. We can now represent $\mathcal{B}$ concretely as an $(n \times \infty)$ (0,1) matrix whose rows are indexed by $\mathbb{N}_n = \{1, 2, \ldots, n\}$ and whose columns are indexed by Σ^*. Note that the ith row then is the (0,1) property vector for language L_i. For example, Figure 3.2 shows a portion of this matrix for a three-element basis which we call $\mathcal{B}_{248}$:

$$\mathcal{B}_{248} = \{\{a^j \mid j \bmod 2 = 1\}, \{a^j \mid j \bmod 4 = 2 \text{ or } 3\}, \{a^j \mid j \bmod 8 = 4, 5, 6, \text{ or } 7\}\}.$$

(The portion shown repeats forever to the right.)

	ϵ	a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	$\ldots$
1	0	1	0	1	0	1	0	1	0	$\ldots$
2	0	0	1	1	0	0	1	1	0	$\ldots$
3	0	0	0	0	1	1	1	1	0	$\ldots$

Figure 3.2: A matrix representation of functional basis $\mathcal{B}_{248}$

The inductive basis characterization is based on the matrix's *rows*; here we consider what we can say about the matrix's *columns*. To this end, for $w \in \Sigma^*$, we define $Col(w)$ to be the set of indices of the rows that contain w:

Definition 3.2.1 *Given a basis in some chosen ordering,* $\mathcal{B} = \{L_1, L_2, \ldots, L_n\}$, *define* $Col : \Sigma^* \to 2^{\mathbb{N}_n}$ *by* $Col(w) = \{i \in \mathbb{N}_n \mid w \in L_i\}$.

For example, with respect to $\mathcal{B}_{248}$ we have $Col(a^5) = \{1, 3\}$.

Now we can wonder about the relationship between $Col(w)$ and $Col(aw)$. To do this, we can define a binary relation $\xrightarrow{a}$ on $2^{\mathbb{N}_n}$ for each $a \in \Sigma^*$:

$$\xrightarrow{a} = \{(Col(w), Col(aw)) \mid w \in \Sigma^*\}.$$

Using infix notation, what we get for basis $\mathcal{B}_{248}$ is

$$\emptyset \xrightarrow{a} \{1\} \xrightarrow{a} \{2\} \xrightarrow{a} \{1,2\} \xrightarrow{a} \{3\} \xrightarrow{a} \{1,3\} \xrightarrow{a} \{2,3\} \xrightarrow{a} \{1,2,3\} \xrightarrow{a} \emptyset \xrightarrow{a} \cdots$$

Notice that in this case $\xrightarrow{a}$ turns out to be a *function*. We will then call $\mathcal{B}_{248}$ a *functional basis*:

Definition 3.2.2 *A basis over* Σ *is* functional *if for all* $a \in \Sigma$, $\xrightarrow{a}$ *is a function.*

As an example of a non-functional basis, consider

$$\mathcal{B}_{23} = \{\{a^i \mid i \bmod 2 = 1\}, \{a^i \mid i \bmod 3 = 2\}\}$$

whose matrix representation is shown in Figure 3.3. This basis is not functional,

	ϵ	a	a^2	a^3	a^4	a^5	...
1	0	1	0	1	0	1	...
2	0	0	1	0	0	1	...

Figure 3.3: A matrix representation of non-functional basis $\mathcal{B}_{23}$

because if we look at $Col(\epsilon)$ and $Col(a^4)$, we see that we have $\emptyset \xrightarrow{a} \{1\}$ as well as $\emptyset \xrightarrow{a} \{1,2\}$. So $\xrightarrow{a}$ is not a function.

How do functional bases relate to inductive bases? We have the following result:

Theorem 3.2.3 *Every inductive basis is functional.*

Proof. Suppose that $\mathcal{B} = \{L_1, L_2, \ldots, L_n\}$ is an inductive basis over alphabet Σ. Consider arbitrary strings x and y such that $Col(x) = Col(y)$. To show that $\mathcal{B}$ is a functional basis, we must show that $Col(ax) = Col(ay)$ for every $a \in \Sigma$.

Let $i \in \mathbb{N}_n$. We will argue that $i \in Col(ax)$ iff $i \in Col(ay)$. Before beginning, we note the following two facts. First, since $\mathcal{B}$ is an inductive basis, we have that there exists $S_i \subseteq \mathbb{N}_n$ such that $a\backslash L_i = \bigcup_{j \in S_i} L_j$. Second, since $Col(x) = Col(y)$, we have for any $j \in \mathbb{N}_n$ that $x \in L_j$ iff $j \in Col(x)$ iff $j \in Col(y)$ iff $y \in L_j$. Now we argue as follows:

$i \in Col(ax)$

iff $ax \in L_i$

iff $x \in a\backslash L_i$

iff $x \in \bigcup_{j \in S_i} L_j$ (by the first fact above)

iff $\exists j \in S_i$ such that $x \in L_j$

iff $\exists j \in S_i$ such that $y \in L_j$ (by the second fact above)

iff $y \in \bigcup_{j \in S_i} L_j$

iff $y \in a\backslash L_i$

iff $ay \in L_i$

iff $i \in Col(ay)$.

□

Here is an alternative proof of this theorem; this one makes use of Theorem 3.1.7:

Proof. Suppose that $\mathcal{B} = \{L_1, L_2, \ldots, L_n\}$ is an inductive basis. Then, by the proof of Theorem 3.1.7, there exists an NFA $M = (\mathbb{N}_n, \Sigma, \delta, I, F)$ such that for all $i \in \mathbb{N}_n$, $Proph(i) = L_i$. In that case, we have

$$Col(w) = \{i \in \mathbb{N}_n \mid w \in Proph(i)\} = \widehat{\delta^R}(F, w^R),$$

since $w \in Proph(i)$ iff $i \in \widehat{\delta^R}(F, w^R)$. From this, we can now express $Col(aw)$ in terms of $Col(w)$:

$$\begin{aligned} Col(aw) &= \widehat{\delta^R}(F, (aw)^R) \\ &= \widehat{\delta^R}(F, w^R a) \\ &= \bigcup_{i \in \widehat{\delta^R}(F, w^R)} \delta^R(i, a) \\ &= \bigcup_{i \in Col(w)} \delta^R(i, a) \end{aligned}$$

□

Thus being functional is a *necessary* condition for a basis to be inductive. However, it is not a *sufficient* condition: for instance $\mathcal{B}_{248}$ is a functional basis that is not inductive. (Notice that the left quotient of the first row is $(10101010\ldots)$, which is not generated by $\mathcal{B}_{248}$.) This means that a functional basis need not be realizable by any NFA.

However, we can establish a weaker realizability result for functional bases. Specifically, we show that if $\mathcal{B}$ is a functional basis with n elements, then there

exists an NFA M with at most $2^n + n$ states such that the elements of $\mathcal{B}$ are the prophecies of n of the states of M. Thus the elements of $\mathcal{B}$ can be prophecies of states of an NFA, but the NFA may require additional states.

To establish this result, we begin by observing that the columns of any basis $\mathcal{B}$ give rise to an equivalence relation on Σ^*:

Definition 3.2.4 *Given $x, y \in \Sigma^*$, we say that $x \sim y$ if $Col(x) = Col(y)$.*

It is easy to see that $\sim$ is an equivalence relation; it is called the *kernel of Col*. (Note also that, unlike the *Col* function, the $\sim$ relation does not depend on the *order* in which we choose to write the elements of $\mathcal{B}$.) As usual we write $[x]$ to denote the *equivalence class* of x:

$$[x] = \{y \in \Sigma^* | x \sim y\} = \{y \in \Sigma^* | Col(x) = Col(y)\}.$$

Thus $[x]$ is just the *inverse image* (with respect to *Col*) of $Col(x)$.

Note that each equivalence class of $\sim$ is a language over Σ; thus the set of all the equivalence classes forms a basis, which we call the *kernel basis of* $\mathcal{B}$, denoted $K(\mathcal{B})$. Notice that if $\mathcal{B}$ has n elements, then $K(\mathcal{B})$ has at most 2^n elements, since there are 2^n possible columns, but not all need actually be present in $\mathcal{B}$.

As an example, $K(\mathcal{B}_{248})$ contains 8 elements:

$$\{a^i \mid i \bmod 8 = 0\}, \{a^i \mid i \bmod 8 = 1\}, \{a^i \mid i \bmod 8 = 2\}, \ldots, \{a^i \mid i \bmod 8 = 7\}$$

Now we come to the key theorem:

Theorem 3.2.5 *If basis $\mathcal{B}$ is functional, then its kernel basis $K(\mathcal{B})$ is inductive.*

Proof. Suppose that $\mathcal{B}$ is a functional basis. Given $x \in \Sigma^*$ and $a \in \Sigma$, we must show that $K(\mathcal{B})$ generates $a\backslash[x]$.

To do this, we argue that

$$a\backslash[x] = \bigcup_{y \in a\backslash[x]} [y]$$

The "$\subseteq$" direction of this equation follows immediately from the fact that $y \in [y]$. To show the "$\supseteq$" direction, we show that if $y \in a\backslash[x]$, then $[y] \subseteq a\backslash[x]$; this depends crucially on the assumption that $\mathcal{B}$ is functional. For if $y \in a\backslash[x]$, then $ay \in [x]$ and hence $ay \sim x$. And if $y' \sim y$, then $Col(y') = Col(y)$, which implies (since $\mathcal{B}$ is functional) that $Col(ay') = Col(ay)$, which gives $ay' \sim ay$. So, by the transitivity of $\sim$ we have $ay' \sim x$, which implies $y' \in a\backslash[x]$. □

Now we need one more fact, which holds whether or not $\mathcal{B}$ is functional:

Theorem 3.2.6 *$K(\mathcal{B})$ generates every element of $\mathcal{B}$.*

Proof. Let L_i be an element of $\mathcal{B}$ and suppose that $x \sim y$. Then $Col(x) = Col(y)$. Since $Col(x) = \{i \in \mathbb{N}_n \mid x \in L_i\}$ and $Col(y) = \{i \in \mathbb{N}_n \mid y \in L_i\}$, it follows that $x \in L_i$ iff $y \in L_i$. Using also the fact that $x \in [x]$, we conclude that $x \in L_i$ iff $[x] \subseteq L_i$, which implies that $L_i = \bigcup_{x \in L_i} [x]$. □

Now we can show the realizability result for functional bases that was mentioned above.

Corollary 3.2.7 *If $\mathcal{B}$ is a functional basis of size n, then there exists an NFA M with at most $2^n + n$ states such that $\mathcal{B} \subseteq Prophecies(M)$.*

Proof. Since $\mathcal{B}$ is functional, we know that $K(\mathcal{B})$ is inductive. And, since $K(\mathcal{B})$ generates every element of $\mathcal{B}$, we have that $K(\mathcal{B}) \cup \mathcal{B}$ is also inductive. (This relies on the fact that $a\backslash(L_1 \cup L_2) = (a\backslash L_1) \cup (a\backslash L_2)$.) Also note that $K(\mathcal{B}) \cup \mathcal{B}$ has size at most $2^n + n$. So, by Theorem 3.1.7, it can be realized by an NFA with at most $2^n + n$ states. □

As a final corollary, we note that Corollary 3.1.10 carries over to functional bases:

Corollary 3.2.8 *If $\mathcal{B}$ is a functional basis, then every element of $\mathcal{B}$ is regular.*

We do not consider functional bases further in this dissertation, since (unlike inductive bases) they do not correspond closely to NFAs. We do remark, however, that it might be possible to apply the results of this section to search more efficiently for inductive bases. (For example, since every inductive basis is functional, we know that if an inductive basis over a unary alphabet ever repeats a column, then the whole basis must repeat from that point on.) Next chapter, we introduce *partials*, which allow us to work with finite approximations of languages.

CHAPTER 4

THEORY OF PARTIALS

In this chapter, we introduce the theory of *partially-specified languages*, which we call *partials*. Partials are a central concept of the present work; they are used at all conceptual levels, including the *theoretical*, *algorithmic*, and *implementation* levels.

Partials generalize languages by allowing the status of some strings in Σ^* to be unspecified. For example, over $\{a\}^*$ we could have a partial P that specifies that ϵ and a^3 are *in* P, that a^2 is *not in* P, and that all other strings are *unspecified*. P could be described using the infinite property vector (1?01????...), or by the finite property vector (1?01) if we assume that all strings beyond the end of the vector are unspecified.

The concept of *recognition* by an NFA can be straightforwardly generalized from languages to partials—we will say that an NFA recognizes a partial P if it correctly accepts or rejects each of the strings that P specifies; its behavior on unspecified strings is unconstrained. An important consequence of this definition is that if an NFA M is minimal for some partial P, then M must also be minimal for its language $L(M)$; the reason is that a smaller NFA recognizing $L(M)$ would also be a smaller NFA recognizing P. We will also see that the theory of inductive bases from Chapter 3, including Theorem 3.1.7, can be generalized to partials.

4.1 Fundamental Definitions

We formalize partials as mappings from Σ^* to $\{\mathit{True}, \mathit{False}, \perp\}$. The strings mapped to *True* are *in* the partial, the strings mapped to *False* are *not in* the partial, and the strings mapped to $\perp$ are *unspecified*.

Definition 4.1.1 *$Bool_\perp$ is the flat domain $\{True, False, \perp\}$ partially ordered by $\sqsubseteq$, so that $\perp \sqsubseteq True$, $\perp \sqsubseteq False$, and for all $a \in Bool_\perp$, $a \sqsubseteq a$. (Note that True and False are incomparable.)*

Definition 4.1.2 *We extend the logical operators $\wedge$ ("and"), $\vee$ ("or"), $\neg$ ("not"), and $\rightarrow$ ("implies") to $Bool_\perp$ as follows:*

- $\perp \wedge \perp = \perp$
- $\perp \wedge True = True \wedge \perp = \perp$
- $\perp \wedge False = False \wedge \perp = False$
- $\perp \vee \perp = \perp$
- $\perp \vee True = True \vee \perp = True$
- $\perp \vee False = False \vee \perp = \perp$
- $\neg\perp = \perp$
- $\perp \rightarrow \perp = \perp$
- $False \rightarrow \perp = True$
- $True \rightarrow \perp = \perp$
- $\perp \rightarrow False = \perp$
- $\perp \rightarrow True = True$

Note that these definitions are *non-strict*.

Now we are ready to define partials formally.

Definition 4.1.3 *A* partial *P (over Σ) is a function from Σ^* to $Bool_\perp$.*

Note that a *language* L can be viewed as a function from Σ^* to $\{True, False\}$; thus a language is simply a partial that maps no strings to $\perp$.

An equivalent, and sometimes convenient, way of defining a partial P is as a pair (A, R) of disjoint languages, where

$$A = \{w \in \Sigma^* \mid P(w) = True\}$$

and

$$R = \{w \in \Sigma^* \mid P(w) = False\}.$$

Here we refer to A as the *accept set* of P and to R as the *reject set* of P. We also refer to the set $D = A \cup R$ as the *defined set* of P, and to the set $U = \overline{D}$ as the *undefined set* of P. Note that we have $A \cup R \cup U = \Sigma^*$.

An important special case is when the defined set of a partial is finite:

Definition 4.1.4 *We say that a partial is* finite *when its defined set $A \cup R$ is finite.*

The main point of using partials is to have a finite representation of regular languages, a representation that can be extended by incorporating the specification of new strings. Our plan is to generate automata that recognize finite partials of the desired language; we keep extending the partials until these automata recognize the desired language. In fact we will use a restricted kind of finite partial, namely one in which all the specified strings come lexicographically before all the unspecified strings; we refer to these as *initial partials*.

Definition 4.1.5 *A finite partial P is an* initial partial *if there exists $k \geq 0$ such that, for every $w \in \Sigma^*$, $P(w) \neq \perp$ iff $index_\Sigma(w) < k$. In this case we say that k is the* width *of P.*

(Recall that $index_\Sigma(w)$ is the 0-based index of w within Σ^*.) We sometimes use the notation $P^{(k)}$ to denote an initial partial of width k.

An initial partial of width k can be conveniently represented by a (0,1) property vector of length k. For example, recall that $\{a, b\}^*$ is lexicographically ordered as follows:

$$\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, aaaa, \ldots$$

Consider the initial partial of width 10 whose accept set is $\{a, aa, ab, aaa, aab, aba\}$ and whose reject set is $\{\epsilon, b, ba, bb\}$. This partial is represented by a property vector of length 10 that uses 1's to indicate strings in the accept set and 0's to indicate strings in the reject set: (0101100111).

Now we turn to the fundamentally important *extends* relation on partials.

Definition 4.1.6 *Partial* P_2 extends *partial* P_1, *denoted* $P_1 \sqsubseteq P_2$ *if for all* $w \in \Sigma^*$, $P_1(w) \sqsubseteq P_2(w)$.

If P maps some string w to $\perp$, then we can form a proper extension P' by mapping w to either *True* or *False*, and mapping all other strings to the same value that P does. More generally, it is easy to see that $P_1 \sqsubseteq P_2$ iff P_2's accept and reject sets are supersets of P_1's accept and reject sets, respectively.

4.2 Simple Properties of Partials and Extension

The extends relation is a partial order:

Theorem 4.2.1 *The extends relation* $\sqsubseteq$ *is reflexive, transitive, and antisymmetric:*

- *For all* P, $P \sqsubseteq P$.
- *For all* P_1, P_2, *and* P_3, *if* $P_1 \sqsubseteq P_2$ *and* $P_2 \sqsubseteq P_3$, *then* $P_1 \sqsubseteq P_3$.
- *For all* P_1 *and* P_2, *if* $P_1 \sqsubseteq P_2$ *and* $P_2 \sqsubseteq P_1$, *then* $P_1 = P_2$.

Proof. These are direct consequences of the definition of extends and the fact that $Bool_\perp$ is a partial order. □

Next we consider *maximal* and *minimal* partials. As mentioned earlier, a *language* L can be viewed as a partial that maps every string to *True* or *False*, or as a partial whose defined set is Σ^*. Notice that every language is *maximal* with respect to $\sqsubseteq$, since the only partial that extends a language is the language itself. (Also notice that a language is never a finite partial or an initial partial. Especially note that a finite language is not a finite partial, because a finite language has a finite accept set but an infinite reject set.)

Next we consider *minimal* partials.

Definition 4.2.2 *The* bottom partial $P_\perp$ *is given by* $P_\perp(w) = \perp$*, for all* $w \in \Sigma^*$.

Notice that the bottom partial is the unique initial partial of width 0; it can be represented with an empty property vector: (). It is immediate that every partial extends the bottom partial $P_\perp$, so it is the unique minimal partial.

In fact partials form a *complete partial order (CPO)*, because every chain

$$P_0 \sqsubseteq P_1 \sqsubseteq P_2 \sqsubseteq \cdots$$

has a *least upper bound* U given by

$$U(w) = \begin{cases} \mathit{True}, & \text{if, for some } i,\ P_i(w) = \mathit{True} \\ \mathit{False}, & \text{if, for some } i,\ P_i(w) = \mathit{False} \\ \perp, & \text{otherwise.} \end{cases}$$

(The fact that U is well defined follows from the fact that the P_i's form a chain.)

Now we consider extensions of initial partials. Consider a width-k initial partial $P^{(k)}$; it maps the first k strings of Σ^* (ordered lexicographically) to either *True* or *False*, and all other strings of Σ^* to $\perp$. Notice that there are exactly two initial

partials of width-$(k+1)$ that extend $P^{(k)}$—both are identical to $P^{(k)}$ except that one maps the string whose index is k to *True*, while the other maps it to *False*. With respect to the property vector representation, we can form the representations of the two width-$(k+1)$ extensions by simply appending either a 0 or a 1. For example, the partial (011) can be extended either to (0110) or to (0111). Thus we see that the extensions of initial partials can be described precisely using an infinite binary tree.

We next observe that any language L extends a unique initial partial of width k, for any $k \geq 0$:

Definition 4.2.3 *Given a language $L \subseteq \Sigma^*$, we define $Init^{(k)}(L)$, the* width-k initial partial of L, *by*

$$Init^{(k)}(L)(w) = \begin{cases} L(w), & \text{if } index_\Sigma(w) < k \\ \bot, & \text{otherwise.} \end{cases}$$

Thus the $Init^{(k)}$ function maps languages into width-k initial partials. It is easy to see that this gives us an infinite chain of partials, all extended by L:

$$Init^{(0)}(L) \sqsubseteq Init^{(1)}(L) \sqsubseteq Init^{(2)}(L) \sqsubseteq Init^{(3)}(L) \sqsubseteq \cdots \sqsubseteq L$$

Furthermore, $Init^{(k)}$ can be used to define an equivalence relation on languages: we define $L_1 \equiv_k L_2$ if $Init^{(k)}(L_1) = Init^{(k)}(L_2)$. Note that $\equiv_k$ has 2^k equivalence classes, since there are 2^k initial partials of width k. There are infinitely many regular languages in each of these equivalent classes.

We can also define a notion of separator:

Definition 4.2.4 *Given distinct languages L_1 and L_2 over Σ, separator(L_1, L_2) is the least k such that $Init^{(k)}(L_1) \neq Init^{(k)}(L_2)$.*

It is easy to see that $separator(L_1, L_2)$ is the smallest position at which L_1 and L_2 differ.

It is convenient to extend $Init^{(k)}$ from languages to *partials*:

Definition 4.2.5 *Given a partial P over Σ, we define $Init^{(k)}(P)$ by*

$$Init^{(k)}(P)(w) = \begin{cases} P(w), & \text{if } index_\Sigma(w) < k \\ \bot, & \text{otherwise.} \end{cases}$$

Notice here that $Init^{(k)}(P)$ need not be an initial partial of width k; indeed it need not be an initial partial at all. But if P is an initial partial of width at least k, then $Init^{(k)}(P)$ will be an initial partial of width k.

We now develop some results relating $\sqsubseteq$ and $\subseteq$.

Definition 4.2.6 *Given a partial P over Σ, let $L(P)$ denote the family of languages extending P:*

$$L(P) = \{L \subseteq \Sigma^* \mid P \sqsubseteq L\}.$$

Note that extended partials have smaller families: if $P_1 \sqsubseteq P_2$, then $L(P_2) \subseteq L(P_1)$.

Also, $L(P)$ contains minimum and maximum elements with respect to $\subseteq$:

Definition 4.2.7 *Let $minL(P)$ be defined by*

$$minL(P)(w) = \begin{cases} False, & \text{if } P(w) = \bot \\ P(w), & \text{otherwise} \end{cases}$$

and $maxL(P)$ by

$$maxL(P)(w) = \begin{cases} True, & \text{if } P(w) = \bot \\ P(w), & \text{otherwise} \end{cases}$$

Then, for every $L \in L(P)$, we have $minL(P) \subseteq L \subseteq maxL(P)$.

In fact, $L(P)$ is an interval:

$$L(P) = \{L \mid minL(P) \subseteq L \subseteq maxL(P)\}.$$

Finally, we note that incomparable partials have disjoint families: if $P_1 \not\sqsubseteq P_2$ and $P_2 \not\sqsubseteq P_1$, then $L(P_1) \cap L(P_2) = \emptyset$.

4.3 Operations on Partials

Because partials are a generalization of languages, it is natural to want to extend operations on languages, such as union and intersection, to partials. We define these extended operations in this section.

There are several properties that our extended operations will satisfy. For example, suppose that we generalize a binary operation $\square$ to partials. Then we want the following properties:

- *Generalization*: if P_1 and P_2 are languages, then $P_1 \square P_2$ should coincide with $\square$ on languages.
- *Monotonicity*: if $P_1 \sqsubseteq P_1'$ and $P_2 \sqsubseteq P_2'$, then $P_1 \square P_2 \sqsubseteq P_1' \square P_2'$.
- *Maximality*: $\square$ leaves unspecified as little as possible. More precisely, if $P_1 \square P_2 \sqsubset P$, then there exist languages L_1 and L_2 such that $P_1 \sqsubseteq L_1$, $P_2 \sqsubseteq L_2$, but $P \not\sqsubseteq L_1 \square L_2$.
- *Algebraic properties* like associativity and commutativity are preserved.

We prove some of these properties for some of our extended operations; the omitted proofs are routine.

The operations we define are union, intersection, complement, reverse, concatenation, and left quotient by a letter. This is not just a theoretical exercise; many of

these operations are used in our `ibas` algorithm and are fundamental to its correctness. In our definitions, we sometimes use the functional characterization of partials and sometimes the set characterization, depending on the clarity of the definition.

Definition 4.3.1 (Union) *Given partials P_1 and P_2, $P_1 \cup P_2$ is given by*

$$(P_1 \cup P_2)(w) = P_1(w) \vee P_2(w).$$

Alternatively, we could have used the set characterization. If $P_1 = (A_1, R_1)$ and $P_2 = (A_2, R_2)$, then $P_1 \cup P_2 = (A_1 \cup A_2, R_1 \cap R_2)$.

Here is the proof that our extended $\cup$ satisfies monotonicity:

Theorem 4.3.2 *If $P_1 \sqsubseteq P_1'$ and $P_2 \sqsubseteq P_2'$, then $P_1 \cup P_2 \sqsubseteq P_1' \cup P_2'$.*

Proof. For any $w \in \Sigma^*$ we have $P_1(w) \sqsubseteq P_1'(w)$ and $P_2(w) \sqsubseteq P_2'(w)$. Hence, because $\vee$ is monotone, we have $(P_1 \cup P_2)(w) = P_1(w) \vee P_2(w) \sqsubseteq P_1'(w) \vee P_2'(w) = (P_1' \cup P_2')(w)$. □

It is also easy to see that union satisfies generalization and maximality. Also, union is associative, justifying more general unions like $\bigcup_i P_i$.

Also, $Init^{(k)}$ commutes with union:

Theorem 4.3.3 $Init^{(k)}(P_1 \cup P_2) = Init^{(k)}(P_1) \cup Init^{(k)}(P_2)$.

Proof. Given $w \in \Sigma^*$, we distinguish two cases. If $index_\Sigma(w) < k$, then

$$\begin{aligned} Init^{(k)}(P_1 \cup P_2)(w) &= (P_1 \cup P_2)(w) \\ &= P_1(w) \vee P_2(w) \\ &= Init^{(k)}(P_1)(w) \vee Init^{(k)}(P_2)(w) \\ &= (Init^{(k)}(P_1) \cup Init^{(k)}(P_2))(w). \end{aligned}$$

And if $index_\Sigma(w) \geq k$, then

$$Init^{(k)}(P_1 \cup P_2)(w) = \perp = (Init^{(k)}(P_1) \cup Init^{(k)}(P_2))(w).$$

□

Notice that the union of initial partials need not be an initial partial. But the union of initial partials of the same width k is an initial partial of width k. Notice that if initial partials of width k are represented as property vectors of length k, then union is just bitwise logical *or*. We use this implementation in `ibas`.

Definition 4.3.4 (Intersection) *Given partials P_1 and P_2, $P_1 \cap P_2$ is given by*

$$(P_1 \cap P_2)(w) = P_1(w) \wedge P_2(w).$$

On initial partials of width k, this is just bitwise logical *and*.

Definition 4.3.5 (Complement) *Given partial P, $\overline{P}$ is given by $\overline{P}(w) = \neg(P(w))$.*

On initial partials, this is just bitwise logical *not*.

The complement operation satisfies monotonicity: if $P \sqsubseteq P'$, then $\overline{P} \sqsubseteq \overline{P'}$.

Definition 4.3.6 (Reverse) *Given partial P, P^R is given by $P^R(w) = P(w^R)$.*

Definition 4.3.7 (Concatenation) *Given partials P_1 and P_2, P_1P_2 is given by*

$$(P_1P_2)(w) = \bigvee_{x,y \text{ s.t. } w=xy} (P_1(x) \wedge P_2(y)).$$

Finally, we turn to left quotients of partials. This operation turns out to be very important for `ibas`, which is based on a generalization of inductive bases to partials.

Definition 4.3.8 (Left Quotient) *Given partial P and $a \in \Sigma$, $a\backslash P$ is defined by $(a\backslash P)(w) = P(aw)$.*

Alternatively, we can use the set characterization: if $P = (A, R)$, then $a\backslash P = (a\backslash A, a\backslash R)$. It is easy to check that both forms are equivalent since $w \in (a\backslash A)$ iff $aw \in A$ iff $P(aw) = \mathit{True}$, and similarly for R.

Here we show that our extended left quotient operation satisfies monotonicity:

Theorem 4.3.9 *If $P \sqsubseteq P'$, then $a\backslash P \sqsubseteq a\backslash P'$.*

Proof. We have $(a\backslash P)(w) = P(aw) \sqsubseteq P'(aw) = (a\backslash P')(w)$. □

We next explore the left quotients of initial partials in more detail.

Theorem 4.3.10 *If P is an initial partial, then $a\backslash P$ is an initial partial.*

Proof. An initial partial P is a finite partial such that for all $x, y \in \Sigma^*$, if x is less than y lexicographically, then $P(x) \sqsupseteq P(y)$.

First, note that $a\backslash P$ is a finite partial, because $(a\backslash P)(w) \neq \perp$ iff $P(aw) \neq \perp$, which holds for only finitely many choices of aw, and hence for only finitely many choices of w. Next, suppose that x is less than y lexicographically. Then $(a\backslash P)(x) = P(ax) \sqsupseteq P(ay) = (a\backslash P)(y)$, since P is an initial partial and ax is less than ay lexicographically. □

Theorem 4.3.11 *Let L be a language over Σ. For every $a \in \Sigma$ and $k \geq 0$, we have $a\backslash \mathit{Init}^{(k)}(L) \sqsubset \mathit{Init}^{(k)}(a\backslash L)$.*

Proof. By definition of $\mathit{Init}^{(k)}$ and left quotient, for every $w \in \Sigma^*$ we have

$$(a\backslash \mathit{Init}^{(k)}(L))(w) = \mathit{Init}^{(k)}(L)(aw) = \begin{cases} L(aw), & \text{if } \mathit{index}_\Sigma(aw) < k \\ \perp, & \text{otherwise.} \end{cases}$$

And we have

$$\mathit{Init}^{(k)}(a\backslash L))(w) = \begin{cases} L(aw), & \text{if } \mathit{index}_\Sigma(w) < k \\ \perp, & \text{otherwise.} \end{cases}$$

The desired result follows from the fact that $index_\Sigma(w) < index_\Sigma(aw)$, which holds because w comes before aw lexicographically. □

Finally, we discuss the computation of the left quotient of an initial partial in more detail:

Unary Alphabet

A particularly easy case is the left quotient of an initial partial over a unary alphabet: if P is an initial partial of width k over a unary alphabet, then $a\backslash P$ is an initial partial of width $k-1$, whose property vector is formed by deleting the first bit of the property vector for P. This can be implemented using a *left shift* bit operation.

For example, if $P = Init^{(7)}(\{a^i \mid i \bmod 3 = 0\})$, whose property vector is (1001001), then $a\backslash P$ has property vector (001001).

Binary Alphabet

To take the left quotient language of an initial partial over alphabet $\{a, b\}$, we need to take pieces of the partial and put them together into a new property vector. We first eliminate the ϵ and then take half of the elements of length 1, half of the elements of length 2, half of the element of length 3, and so on. Assuming that the width of P is of the form $2^k - 1$, the resulting quotient partials will be of width $2^{k-1} - 1$. For example,

	ϵ	a	b	aa	ab	ba	bb	aaa	aab	aba	abb	baa	bab	bba	bbb
P	1	0	1	1	1	0	1	1	1	1	0	1	0	0	1
$a\backslash P$	0	1	1	1	1	1	0								
$b\backslash P$	1	0	1	1	0	0	1								

CHAPTER 5

PARTIALS AND NFAS

In this chapter, we explore the relationship between partials and NFAs. We will say that an NFA M recognizes a partial P if $L(M)$ is an extension of P. This implies that lower bounds on the size of NFAs recognizing a partial P are also lower bounds on the size of NFAs recognizing any language that extends P. We also show that every regular language L has a *canonical partial*, which has the property that every minimal NFA recognizing the canonical partial also recognizes L.

5.1 Recognition of Partials

First, we extend the concept of *recognition* from languages to partials:

Definition 5.1.1 *NFA M recognizes partial P iff $P \sqsubseteq L(M)$.*

Under this definition, M must accept all strings in P's accept set, must reject all strings in P's reject set, but is unconstrained on strings in P's undefined set.

It is important to see that an NFA M recognizes only one language, $L(M)$, but recognizes many different partials. However, for each k, an NFA M recognizes exactly one initial partial of width k, namely $Init^{(k)}(L(M))$.

This leads us to a notion of *regular partials*.

Definition 5.1.2 *A partial is* regular *iff it is recognized by some NFA.*

Equivalently, a partial is regular iff some regular language extends it. Note that all finite partials (and hence all initial partials) are regular.

5.2 Partials and Minimal NFAs

Here we study the relationship between NFAs and the partials they recognize.

If an NFA M recognizes a partial P, then all we know is that $L(M)$ is a regular language that extends P. This does not tell us very much about $L(M)$. But suppose we know that M is a *minimal* machine recognizing P, in that no NFA with fewer states recognizes P. Then we can get more interesting conclusions. First, we show that a minimal NFA for a partial is minimal for its language:

Theorem 5.2.1 *If NFA M is minimal for partial P, then M is also minimal for its language $L(M)$.*

Proof. By contradiction. Since M recognizes P, we have by definition that $P \sqsubseteq L(M)$. So any NFA that recognizes $L(M)$ also recognizes P. Hence if M is not minimal for $L(M)$, it cannot be minimal for P either. □

However the converse is not true; minimal NFAs for languages need not be minimal for some of the partials they recognize. For example, *every* NFA recognizes the bottom partial $P_\perp$, so the minimal machine for $P_\perp$ has 0 states.

The set of all minimal NFAs that recognize an initial partial can be found by `ibas`. The set of all the languages recognized by any of these minimal machines is finite, and many times very small. Indeed, some partials are such that all of their minimal machines recognize the same language:

Definition 5.2.2 *A partial P is* unambiguous *if all minimal NFAs that recognize it recognize the same language.*

For example, over the unary alphabet $\{a\}$ we have that $Init^{(2)}(\{a\}^*)$ and $Init^{(2)}(\{\epsilon\})$ are unambiguous partials recognized by 1-state NFAs. (Their property vectors are (11) and (10), respectively.) On the other hand, $Init^{(1)}(\{a\}^*)$, whose property vector is (1), is ambiguous; as shown in Figure 5.1 there are two minimal 1-state NFAs recognizing this partial but recognizing different languages.

Figure 5.1: Two minimal NFAs for ambiguous partial (1)

Definition 5.2.3 *An unambiguous partial P is* complete for L *if all the minimal NFAs that recognize P recognize L. The initial partial of minimum width that is complete for L is called the* canonical partial for L*, denoted $Canon(L)$.*

Of course, it is not obvious that canonical partials necessarily exist. But we will show later that every regular language has a canonical partial, thereby giving a one-to-one correspondence between regular languages and canonical partials. The practical result will be that partials and complete partials can be used to study the characteristics of of minimal NFAs recognizing a given language.

5.3 Theorems on Partials and NFAs

Theorem 5.3.1 *If an NFA does not recognize $Init^{(k)}(L)$, for some $k \geq 0$, then it does not recognizes L.*

Proof. Immediate from the fact that $Init^{(k)}(L) \sqsubseteq L$. □

In searches, this theorem allows us to discard any NFA that does not recognize an initial partial P of the language we want to recognize, since it cannot recognize the language. (Indeed, such an NFA cannot recognize *any* extension of P.) Of course, discarding just one NFA is not very powerful. But, in principle, we could discard

many NFAs. For if NFA M wrongly accepts w, then all NFAs formed by *adding* arrows to M will also wrongly accept w. Similarly, if M wrongly rejects w, then all NFAs formed by *deleting* arrows from M will also wrongly reject w. The search techniques that we develop later use similar ideas to prune the depth-first-search tree. But our search does not work directly with NFAs, but rather with inductive bases, which are built incrementally. We develop this approach in Chapter 6.

We can strengthen the above theorem:

Theorem 5.3.2 *An NFA recognizes $Init^{(k)}(L)$, for all k, iff it recognizes L.*

Proof. The "if" direction was shown above.

For the "only if" direction, consider any string $w \in \Sigma^*$ and suppose that the index of w is m. By assumption, M recognizes $Init^{(m+1)}(L)$, but $Init^{(m+1)}(L)(w) = L(w)$, so M must correctly accept or reject w, depending on L. Since w was arbitrary, we see that M recognizes L. □

These results give us interesting conclusions about minimality. For instance, to know that NFA M is minimal for its language, we do not even need to know what $L(M)$ is; it suffices to know that M is minimal for some partial P. Moreover M is then guaranteed to be minimal for any partial extending P that it recognizes.

Theorem 5.3.3 *The size of a minimal machine for a partial P is a lower bound on the size of a minimal machine for any partial P' such that $P \sqsubseteq P'$.*

Proof. If NFA M recognizes P', then (by the transitivity of $\sqsubseteq$) also M recognizes P. Hence if P cannot be recognized with fewer than n states, then also P' cannot be recognized with fewer than n states. □

Hence expanding to wider partials of L will give a nondecreasing sequence of lower bounds on the size of a minimal NFA recognizing L.

Theorem 5.3.4 *Let n_i be the minimum number of states in any NFA that recognizes $Init^{(i)}(L)$. Then $n_0 \leq n_1 \leq n_2 \leq \ldots$, and the sequence is bounded iff L is regular.*

Proof. The "if" direction is obvious. For the "only if" direction, notice that there is some maximum value n in the sequence. Hence, for all i, $Init^{(i)}(L)$ can be recognized by some n-state NFA M_i. But there are only finitely many n-state NFAs, so some M_k must occur on the list infinitely often. Hence M_k must recognizes L. □

As we will show, we are able to generate all minimal machines for initial partials in an efficient way; a machine for $Init^{(k+1)}(L)$ can be made from some machine for $Init^{(k)}(L)$. Just as partials form a lattice-like structure induced by the extends relation, we will see in Chapter 6 that when NFAs are represented using partial inductive bases, they too inherit a lattice-like structure from the extends relation.

5.4 Existence of Canonical Partials

Now we show the key result that every regular language has a canonical partial.

Theorem 5.4.1 *For any regular language L over alphabet Σ, there is an integer k such that all minimal NFAs that recognize $Init^{(k)}(L)$ also recognize L.*

Proof. Let s be the minimum number of states required to recognize L. Now consider the set of all s-state NFAs over Σ that do *not* recognize L; this set is typically huge, but it is finite, because there are only finitely many s-state NFAs over Σ. For each NFA M in this set, note that there is a *smallest index* at which $L(M)$ differs from L. Now choose k to be one more than the maximum of these indices. It follows that none of the NFAs in the set can recognize $Init^{(k)}(L)$. Hence any s-state NFA that does recognize $Init^{(k)}(L)$ also recognizes L. □

Corollary 5.4.2 *Every regular language L has a canonical partial $Canon(L)$.*

Proof. We just proved that every regular language has a complete initial partial, hence there must be one of minimum width. (In fact the k we constructed is minimal.) □

Corollary 5.4.3 *There is a one-to-one correspondence between regular languages and canonical partials.*

Proof. We just showed that each regular language has a canonical partial. And two distinct languages cannot have the same canonical partial. □

5.5 Bounds on the Width of the Canonical Partial

An important question is whether there are good bounds on the width of the canonical partial for a regular language L. One might hope to achieve bounds in terms of the size of the minimal NFA or DFA recognizing L.

Here are a couple of rather negative results:

Theorem 5.5.1 *There can be no polynomial bound on the width of the canonical partial for L in terms of the size of a minimal NFA recognizing L.*

Proof. If $p_1, p_2, \ldots, p_n$ are distinct primes, $n \geq 1$, then the canonical partial for the language

$$L = \{a^i \mid i \bmod p_1 p_2 \cdots p_n \neq 0\}$$

has width greater than $p_1 p_2 \cdots p_n$. (The reason is that if $k \leq p_1 p_2 \cdots p_n$, then $Init^{(k)}(L) = Init^{(k)}(aa^*)$.) But L can be recognized by an NFA with $p_1 + p_2 + \cdots + p_n$ states. And $p_1 p_2 \cdots p_n$ is exponentially bigger than $p_1 + p_2 + \cdots + p_n$. □

Theorem 5.5.2 *The width of the canonical partial for L may need to be twice the size of the minimal DFA recognizing L.*

Proof. For $m \geq 1$, the canonical partial for the singleton language $\{a^m\}$ has width greater than $2m + 1$. (The reason is that if $k \leq 2m + 1$, then $Init^{(k)}(\{a^m\}) = Init^{(k)}((a^m)^+)$.) But the language can be recognized by a DFA with $m + 1$ states. □

CHAPTER 6

PARTIAL INDUCTIVE BASES

In Chapter 5, we showed that if we want to find all the minimal NFAs recognizing a language L, it is useful to search for the minimal NFAs recognizing *partials* of L. Specifically, we may choose a $k \geq 0$ and search for minimal NFAs recognizing the partial $Init^{(k)}(L)$. Such NFAs may not recognize L, of course, but as we saw in Chapter 5 this approach is useful in several ways. First, if n is the minimum number of states needed to recognize $Init^{(k)}(L)$, then n is a *lower bound* on the number of states needed to recognize L. This follows that the fact that any NFA that recognizes L also recognizes $Init^{(k)}(L)$, for every k. Second, if k is large enough, then any minimal NFA recognizing $Init^{(k)}(L)$ must in fact recognize L. (However, it does not seem easy to know how large k must be.) Finally, even if k is not large enough, we may find that the set of all minimal NFAs recognizing $Init^{(k)}(L)$ is small enough that we can feasibly check each such NFA to see whether it accepts L.

In this chapter, we develop techniques for finding NFAs recognizing a given partial. The idea is to generalize inductive bases to *partial inductive bases* while preserving Theorem 3.1.7. This will allow us to find NFAs recognizing a partial P by finding partial inductive bases generating P.

6.1 Partial Bases and Partial Inductive Bases

We first generalize from bases to *partial bases*:

Definition 6.1.1 *A* partial basis $\mathcal{B}$ *is a finite multiset of partials over some alphabet* Σ. *We say that a partial basis* $\mathcal{B}$ generates *a partial* P *if there is some subcollection of* $\mathcal{B}$ *whose union extends* P.

Notice that this definition does not require that the union of the subcollection *equals* P, but only that it *extends* P. (This will be important when we generalize to partial inductive bases, because $a\backslash P$ is less wide than P.) Also notice that we have defined a partial basis to be a *multiset*, rather than a set.

We will usually work with a restricted kind of partial basis:

Definition 6.1.2 *We say that $\mathcal{B}$ is a partial basis* of width k *if each element of $\mathcal{B}$ is an initial partial of width k.*

A natural way to create a partial basis of width k is by applying $Init^{(k)}$ to each language in a basis. More generally, we can define $Init^{(k)}(\mathcal{B})$ for any partial basis:

Definition 6.1.3 *Given a partial basis $\mathcal{B}$, we define $Init^{(k)}(\mathcal{B})$ to be the multiset*

$$\{Init^{(k)}(P) \mid P \in \mathcal{B}\}.$$

Of course, if $\mathcal{B}$ is a partial basis, then in general $Init^{(k)}(\mathcal{B})$ need not be a partial basis of width k. But it will be whenever the elements of $\mathcal{B}$ are wide enough.

Notice that even if the elements of $\mathcal{B}$ are distinct, the elements of $Init^{(k)}(\mathcal{B})$ need not be, since initial partials of distinct partials could be equal. For example, for any P_1, P_2, and P_3, $Init^{(0)}(\mathcal{B}) = \{P_\perp, P_\perp, P_\perp\}$. This is the main reason why we have chosen to define partial bases to be multisets.

Also notice that if the elements of a basis $\mathcal{B}$ are in nondecreasing order lexicographically, then the same will be true of the elements of $Init^{(k)}(\mathcal{B})$.

It is natural to generalize the notion of *extension* to partial bases:

Definition 6.1.4 *If $\mathcal{B}$ and $\mathcal{B}'$ are partial bases with the same number of elements, then we say that $\mathcal{B}'$* extends *$\mathcal{B}$, denoted $\mathcal{B} \sqsubseteq \mathcal{B}'$, if $\mathcal{B}$ and $\mathcal{B}'$ can be ordered as $\mathcal{B} = \{P_1, \ldots, P_n\}$ and $\mathcal{B}' = \{P'_1, \ldots, P'_n\}$ such that $P_i \sqsubseteq P'_i$, for all i.*

For example, we have for any basis $\mathcal{B}$ that $Init^{(k)}(\mathcal{B}) \sqsubseteq Init^{(l)}(\mathcal{B})$, if $k \leq l$.

The definition of "extends" on partial bases satisfies two key properties: it preserves the "generates" relation and (with some restrictions) it preserves the "does not generate" relation as well:

Theorem 6.1.5 *If partial basis $\mathcal{B}$ generates partial P, and $\mathcal{B} \sqsubseteq \mathcal{B}'$, then $\mathcal{B}'$ also generates P.*

Proof. By definition, there is a subcollection of $\mathcal{B}$ whose union U extends P. Each element of that subcollection is extended by an element of $\mathcal{B}'$. Hence the union of those elements of $\mathcal{B}'$ extends U, and therefore extends P, by the transitivity of $\sqsupseteq$. □

Theorem 6.1.6 *Let $\mathcal{B}$ be a partial basis of width k and let P be an initial partial of width less than or equal to k. If $\mathcal{B}$ does not generate P, then no extension of $\mathcal{B}$ generates any extension of P.*

Proof. Suppose to the contrary that $\mathcal{B} \sqsubseteq \mathcal{B}'$, $P \sqsubseteq P'$, and $\mathcal{B}'$ generates P'. Then there is a subcollection $P'_1, P'_2, \ldots, P'_n$ of $\mathcal{B}'$ whose union U' extends P'. Because $\mathcal{B} \sqsubseteq \mathcal{B}'$, we know that each P'_i extends some element of $\mathcal{B}$. But this implies that each $\mathit{Init}^{(k)}(P'_i)$ is an element of $\mathcal{B}$, since $\mathcal{B}$ has width k. And the union of these elements is $\mathit{Init}^{(k)}(U')$, which must extend P, giving a contradiction. □

Theorem 6.1.6 is useful in allowing us to search incrementally for partial bases. For if we are searching for partial bases generating an initial partial P, and we find that a partial basis $\mathcal{B}$ of width i does *not* generate $\mathit{Init}^{(i)}(P)$, then we know that $\mathcal{B}$ cannot possibly be extended to a partial basis generating P.

	ϵ	a	a^2	a^3	a^4	a^5	a^6	a^7
1	0	0	1	1	0	1	1	1
2	0	1	0	0	1	1	0	1
3	0	1	1	0	1	1	1	1
4	1	0	0	1	1	0	1	1

Figure 6.1: A matrix representation of a partial inductive basis

Now we are ready to generalize from inductive bases to *partial inductive bases*:

Definition 6.1.7 *A partial basis $\mathcal{B}$ over alphabet Σ is a* partial inductive basis *if for each $B \in \mathcal{B}$ and $a \in \Sigma$, $\mathcal{B}$ generates $a\backslash B$.*

If a partial basis $\mathcal{B}$ (inductive or not) is of width k, then $\mathcal{B}$ is conveniently represented as a $(0,1)$ matrix of size $n \times k$, each row of which is the property vector of one of the partials. For example, over the unary alphabet $\{a\}$, the 4×8 matrix shown in Figure 6.1 represents a 4-element partial inductive basis of width 8. To see that this is a partial inductive basis, note for instance that the left quotient of the third row with a is the width-7 initial partial (1101111); this is generated by taking the union of the second and fourth rows:

$$(01001101) \cup (10011011) = (11011111).$$

Notice that the union properly *extends* the left quotient; this is why we defined "generates" as we did in Definition 6.1.1.

Next we show that any inductive basis gives rise to partial inductive bases of every width:

Theorem 6.1.8 *If $\mathcal{B}$ is an inductive basis, then $Init^{(k)}(\mathcal{B})$ is a partial inductive basis, for every $k \geq 0$.*

Proof. Let $B \in \mathcal{B}$. Since $\mathcal{B}$ is an inductive basis, we know that $\mathcal{B}$ generates $a\backslash B$ for every a. And we must show that $Init^{(k)}(\mathcal{B})$ generates $a\backslash Init^{(k)}(B)$. Suppose that

$B_1, B_2, \ldots, B_n$ is the subcollection of $\mathcal{B}$ whose union is $a\backslash B$. So we have

$$B_1 \cup B_2 \cup \cdots \cup B_n = a\backslash B$$

Now we have

$$\begin{aligned} \mathit{Init}^{(k)}(B_1) \cup \mathit{Init}^{(k)}(B_2) \cup \cdots \cup \mathit{Init}^{(k)}(B_n) &= \mathit{Init}^{(k)}(B_1 \cup B_2 \cup \cdots \cup B_n) \\ &= \mathit{Init}^{(k)}(a\backslash B) \\ &\sqsupseteq a\backslash \mathit{Init}^{(k)}(B) \end{aligned}$$

where the first and last steps follow from Theorems 4.3.3 and 4.3.11, respectively. □

6.2 Partial Inductive Bases and NFAs

In this section we show that the results from Chapter 3 can be generalized from inductive bases to partial inductive bases.

First, we observe that any NFA gives rise to partial inductive bases of every width:

Theorem 6.2.1 *If M is an NFA, then for every $k \geq 0$, $\mathit{Init}^{(k)}(\mathit{Prophecies}(M))$ is a partial inductive basis.*

Proof. By Theorem 3.1.5, $\mathit{Prophecies}(M)$ is an inductive basis. So by Theorem 6.1.8, $\mathit{Init}^{(k)}(\mathit{Prophecies}(M))$ is a partial inductive basis. □

Conversely, we can generalize Theorem 3.1.7 by showing that any partial inductive basis $\mathcal{B}$ gives rise to an NFA whose prophecies *extend* $\mathcal{B}$.

Definition 6.2.2 *We say that NFA M* realizes *partial inductive basis $\mathcal{B}$ if $\mathcal{B} \sqsubseteq \mathit{Prophecies}(M)$.*

Theorem 6.2.3 *For every partial inductive basis $\mathcal{B}$ over Σ, there exists an NFA M that realizes $\mathcal{B}$.*

Proof. Given partial inductive basis $\mathcal{B}$ over alphabet Σ, let the elements of $\mathcal{B}$ be enumerated in some order:

$$\mathcal{B} = \{P_1, P_2, \ldots, P_n\}$$

where $n \geq 0$. (Note that the P_i's need not be distinct.)

Let $\mathbb{N}_n$ denote the set of natural numbers from 1 to n. Define NFA $M = (\mathbb{N}_n, \Sigma, \delta, I, F)$, where

- $\delta(i, a)$ is any subset of $\mathbb{N}_n$ such that $\bigcup_{j \in \delta(i,a)} P_j \sqsupseteq a \backslash P_i$,
- I is arbitrary, and
- $F = \{i \in \mathbb{N}_n \mid P_i(\epsilon) = \mathit{True}\}$.

The fact that $\mathcal{B}$ is a partial inductive basis exactly ensures that δ can be constructed.

Now we show that $\mathcal{B} \sqsubseteq \mathit{Prophecies}(M)$ by proving that for each i, P_i is extended by $\mathit{Proph}(i)$:

Lemma 6.2.4 *For all $i \in \mathbb{N}_n$, $P_i \sqsubseteq \mathit{Proph}(i)$.*

Proof. We must show that for every $w \in \Sigma^*$ and every $i \in \mathbb{N}_n$, $P_i(w) \sqsubseteq \mathit{Proph}(i)(w)$. We do this by induction on the length of w.

Basis:

If $P_i(\epsilon) = \mathit{True}$, then $P_i \in F$, so $\epsilon \in \mathit{Proph}(i)$, so $\mathit{Proph}(i)(\epsilon) = \mathit{True}$. If $P_i(\epsilon) = \mathit{False}$ or $P_i(\epsilon) = \bot$, then $i \notin F$, so $\epsilon \notin \mathit{Proph}(i)$, so $\mathit{Proph}(i)(\epsilon) = \mathit{False}$.

Induction:

Let $k \geq 0$. The induction hypothesis is that for all strings u such that

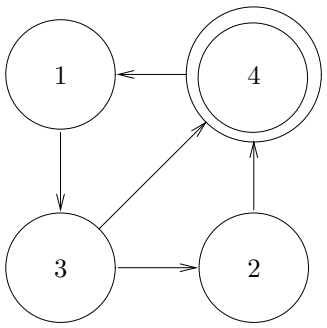

Figure 6.2: An NFA that realizes a partial inductive basis

$|u| \leq k$ and all $j \in \mathbb{N}_n$, $P_j(u) \sqsubseteq \mathit{Proph}(j)(u)$. Now consider the string au, where $a \in \Sigma$.

If $P_i(au) = \mathit{True}$ then, by the definition of left quotient, we have that $(a\backslash P_i)(u) = \mathit{True}$. So, by the definition of δ, we have that there exists $j \in \delta(i, a)$ such that $P_j(u) = \mathit{True}$. By the induction hypothesis, this gives $\mathit{Proph}(j)(u) = \mathit{True}$. Hence $u \in \mathit{Proph}(j)$ and (since $j \in \delta(i, a)$) therefore $au \in \mathit{Proph}(i)$, which implies that $\mathit{Proph}(i)(au) = \mathit{True}$.

If $P_i(au) = \mathit{False}$, then $(a\backslash P_i)(u) = \mathit{False}$. Hence, by the definition of δ, we have $(\bigcup_{j \in \delta(i,a)} P_j)(u) = \mathit{False}$. Hence for every $j \in \delta(i, a)$, we have $P_j(u) = \mathit{False}$, which (by the induction hypothesis) gives $u \notin \mathit{Proph}(j)$. Hence $au \notin \mathit{Proph}(i)$, which gives $\mathit{Proph}(i)(au) = \mathit{False}$.

Finally, if $P_i(au) = \bot$, then trivially $P_i(au) \sqsubseteq \mathit{Proph}(i)(au)$. □

□

As an illustration of this theorem, Figure 6.2 shows an NFA realizing the partial inductive basis from Figure 6.1. Notice that the NFA is shown with no initial states. But because I is arbitrary in the construction, any or all of its states can freely be made initial.

Now we turn our attention to the main goal of this chapter: to develop techniques for finding NFAs recognizing a given partial. So far we have shown that NFAs are, in some sense, equivalent to partial inductive bases. We now show that if P is an initial partial, then NFAs *recognizing* P are, in some sense, equivalent to partial inductive bases *generating* P. This equivalence is made precise in the following two theorems.

Theorem 6.2.5 *Let P be an initial partial of width $k \geq 0$. If NFA M recognizes P, then $Init^{(k)}(Prophecies(M))$ is a partial inductive basis that generates P.*

Proof. By Theorem 6.2.1, $Init^{(k)}(Prophecies(M))$ is a partial inductive basis. And if P is an initial partial of width k and NFA $M = (Q, \Sigma, \delta, I, F)$ recognizes P, then

$$\begin{aligned} P &= Init^{(k)}(L(M)) \\ &= Init^{(k)}(\bigcup_{q \in I} Proph(q)) \\ &= \bigcup_{q \in I} Init^{(k)}(Proph(q)) \end{aligned}$$

□

Theorem 6.2.6 *If $\mathcal{B}$ is an n-element partial inductive basis that generates a partial P, then there exists an n-state NFA M that recognizes P.*

Proof. If $\mathcal{B} = \{P_1, P_2, \ldots, P_n\}$, then by Theorem 6.2.3, there exists an NFA M with state set $\mathbb{N}_n$ such that for all i, $P_i \sqsubseteq Proph(i)$. Since $\mathcal{B}$ generates P, there is a subcollection of $\mathcal{B}$ whose union extends P. Choose I to be the set of indices of that subcollection. Then we have

$$L(M) = \bigcup_{i \in I} Proph(i) \sqsupseteq \bigcup_{i \in I} P_i \sqsupseteq P.$$

□

Hence the problem of finding NFAs recognizing an initial partial P is essentially equivalent to the problem of finding partial inductive bases generating P.

6.3 Extending Partial Inductive Bases

Let P be an initial partial of width $k \geq 0$. If we wish to find partial inductive bases generating P, it is natural to try to proceed iteratively. For note that if $\mathcal{B}$ is a partial inductive basis of width k that generates P, then for any i with $0 \leq i < k$ we have that $Init^{(i)}(\mathcal{B})$ is a partial inductive basis of width i that generates $Init^{(i)}(P)$. Thus any of the bases that we seek can be built by "growing" narrower ones.

This leads us to consider the problem of widening a partial inductive basis of width i to one of width $i + 1$. We have the following corollary that says that this can always be done:

Corollary 6.3.1 *Let $\mathcal{B}$ be a partial inductive basis of width i, for some $i \geq 0$. There exists a partial inductive basis $\mathcal{B}'$ of width $i + 1$ such that $\mathcal{B} \sqsubseteq \mathcal{B}'$.*

Proof. If $\mathcal{B}$ is a partial inductive basis of width i, then by Theorem 6.2.3 there exists an NFA M such that $\mathcal{B} \sqsubseteq Prophecies(M)$. So $Init^{(i+1)}(Prophecies(M))$ is a partial inductive basis of width $i + 1$, by Theorem 6.1.8. And this basis is easily seen to extend $\mathcal{B}$. □

When $\mathcal{B}$ is a partial inductive basis of size n and width i, recall that $\mathcal{B}$ can be represented as an $n \times i$ matrix. We can then understand "widening" $\mathcal{B}$ concretely as a matter of adding a new column to this matrix. The question is whether we can do this efficiently, given that there are 2^n possible columns.

Must we try all 2^n possibilities for the new column? This was what we did in our first version of the `ibas` program (described in [Smi06]), but fortunately we can do much better. The key idea is that we can consider the extension of each element of $\mathcal{B}$ *independently* of how we extend the other elements. Since we know that *some* new column is possible, this means that for each position in the new column there are exactly three possibilities: a fixed 0, a fixed 1, or a free choice.

Let's look at an example before developing this idea formally. Consider the following partial inductive basis over a 1-symbol alphabet:

a 0010

b 0011

c 0100

d 1001

When we try to widen to a partial inductive basis of width 5, we must be sure that the widened basis can generate the left quotient of each newly-widened row. But notice that these left quotients have *width 4*, which means that the new column is irrelevant in determining whether or not they can be generated. For example, if we extend row a with 0, its left quotient will be 0100; this is generated by row c, regardless of how we extend row c. If we extend row a with 1, its left quotient will be 0101, which cannot be generated, regardless of how the other rows are extended. Thus we see that to widen this inductive basis, we *must* extend row a with 0. Now consider row b. If we extend it with 0, its left quotient will be 0110, which is generated as $a \cup c$. And if we extend it with 1, its left quotient will be 0111, which is generated as $b \cup c$. Hence we can extend row b with *either* 0 or 1, and we will still have a partial inductive basis. The reader can similarly verify that row c must be extended with 1, and row d can be extended with either 0 or 1. We can therefore exactly characterize the possible width-5 extensions of our partial inductive basis as follows:

a 00100

b 0011?

c 01001

d 1001?

The two "?"s can be independently replaced by either 0 or 1, giving a total of 4 width-5 extensions.

The general situation is that we have a partial inductive basis $\mathcal{B}$ of width i, over some alphabet, which we are trying to extend to a partial inductive basis $\mathcal{B}'$ of width $i+1$. For each $P \in \mathcal{B}$, we want to determine how P can be extended from width i to width $i+1$ while preserving the property that each of the left quotients can be generated. Now the key is to notice that, by Theorem 4.3.11, the left quotient of an initial partial of width $i+1$ has width *at most* i. This means that $\mathcal{B}'$ will be able to generate the left quotient of the extension of P iff $\mathcal{B}$ can. Hence, as in the example above, we can determine for each $P \in \mathcal{B}$ one of three possibilities: P must be extended with 0, P must be extended with 1, or P can be extended freely with either 0 or 1.

We now make some remarks about implementing these calculations efficiently. There are two main concerns: first, how to compute the left quotients of the extensions; second, how to check whether these can be generated. We can represent our partial inductive basis as an array of bit-vectors. As discussed in Section 4.3, computing the left quotients is then easy in the case of a unary alphabet, since left quotient is then just a left shift. In the case of a non-unary alphabet, computing the left quotients is harder—in that case it seems better to store the left quotients of P so that the left quotients of the extension P' can be computed incrementally from them. Turning to the second concern, we can efficiently check whether $\mathcal{B}$ generates a left quotient using bit operations. First, we can see whether an element P of $\mathcal{B}$ is useful by bitwise ANDing it with the bitwise negation of the left quotient; the result is 0 iff P is useful. Then we just union together all the useful elements of $\mathcal{B}$, to see whether the quotient is generated.

Many other optimizations are possible. For example, the set of elements of $\mathcal{B}$ that are useful in making a left quotient can only *shrink* as we widen the partial inductive basis. It is therefore beneficial to remember and update this set of useful elements as we iteratively widen $\mathcal{B}$.

Also, as will be discussed in the next chapter, the iterative search for partial inductive bases involves both widening and narrowing. It is therefore necessary to maintain a stack-like representation of our knowledge of the 0's, 1's, and ?'s in each new column.

Unique Extendibility

Sometimes it turns out that a partial inductive basis can be extended to an inductive basis in a unique way. Here is an example, over unary alphabet $\{a\}$:

1	10000
2	11000
3	11100
4	11110

Because the last column contains only 0's, we cannot extend any of the rows with 1 without breaking the inductive basis condition. Hence the next column must contain only 0's. The argument then repeats, leading to the result that all subsequent columns must contain only 0's. One of the NFAs that realizes this basis is shown in Figure 6.3.

One might wonder whether a partial inductive basis is uniquely extendible whenever its final column has no "?"s. This turns out not to be true. In fact, experiments show that there can be several consecutive columns with no "?"s followed by a column with a "?" in it.

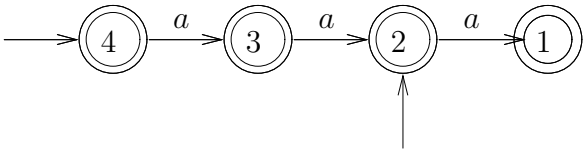

Figure 6.3: A minimal NFA for $\{\epsilon, a, aa, aaa\}$

Generating the Target Partial

As we widen our partial inductive basis, we are not only concerned with keeping it inductive; we also want it to be able to generate the target partial $Init^{(k)}(L)$. We deal with this concern iteratively. During our search, we have a partial inductive basis $\mathcal{B}$ of width i, where $0 \leq i < k$, which generates $Init^{(i)}(L)$. When we try to widen $\mathcal{B}$ to width $i+1$, we also widen the target partial to $Init^{(i+1)}(L)$ and see whether the widened $\mathcal{B}$ can generate the widened target.

If not, then no extension of $\mathcal{B}$ can generate the desired language L, and $\mathcal{B}$ need not be considered further. We then proceed to look at the "next" partial inductive basis by the use of the *increment* operation, which advances to the next possible the last column of $\mathcal{B}$.

Incrementing a Partial Inductive Basis

Sometime a partial inductive basis can be extended in a unique way, but many times it can be extended in more than one way; there are potentially 2^n extensions of a partial inductive basis. In order to consider all feasible extensions we must do it in some order. The obvious choice is to increment the binary numerical value of the extended column; this process may be interrupted by a further extension and resumed later on.

But many of the 2^n potential extensions are not partial inductive bases, since fixed 0's and fixed 1's cannot be changed. Hence the actual number of extensions is

2^q, where q is the number of "?"s in the column. In our implementation, we use a template to indicate which elements are fixed and cannot be incremented.

The increment operation preserves the inductive property if a template for fixed values is used, but we run the risk of breaking the lexicographical order and therefore generating the same partial inductive basis more than once in different permutations. (Note that this can arise only if the width-i basis has repeated elements, which is rare unless i is less than 10.) If the next partial inductive basis is not in lexicographical order, we skip to the next; this guaranties that the same partial inductive basis is not considered more than once.

Shrinking a Partial Inductive Basis

Once we have exhausted all possibilities for the new column, we need to see whether there are any more possibilities for the previous column. We decrease the width of the basis and reinstate the previous fixed values template (they are kept in a stack) then continue the increment operation at the place it was left off. When we shrink back to width 0, we know that all partial inductive bases of that size have been considered. If any were found that generate the target partial, then we terminate the search, since we only want minimal partial inductive bases. If none were found, then increment the number of elements in the basis and begin the search again.

Now that we have developed the theory of partial inductive bases, we are ready to present the `ibas` algorithm. We do this in the next chapter.

CHAPTER 7

THE IBAS ALGORITHM

In this chapter, we use the theory developed in the last several chapters as the foundation of our algorithm for finding all minimal NFAs recognizing a given initial partial P of width k.

Our approach is to find minimal partial inductive bases generating P iteratively, by widening the partial inductive bases generating $Init^{(i)}(P)$, for $0 \leq i < k$.

The actual implementation of `ibas` is written in C; for maximum efficiency we use low-level operations such as bitwise "or" (`|`), bitwise "and" (`&`), and left shift (`<<`). In this chapter we describe the algorithm using object-oriented terminology and recursion in the hopes of making it more understandable.

7.1 `ibas` Algorithm Description

The algorithm takes as input a $0, 1$ string of length k, which represents the initial partial P. The input is stored in an object of the class `Partial`, which holds an array of $0, 1$ values.

The class `Partial` is used to represent the input partial P, and also to represent the elements of the partial inductive basis being constructed.

The class `Basis` holds an array of `Partial`s. We can view a basis as a $0, 1$ matrix with `size` rows and `width` columns or as a collection of `Partial`s, all with the same `width`. Once an object of the class is created, it never changes its `size` and it also never changes its `width`.

We are looking for partial inductive bases of minimum `size`. We have an outer loop (main) that increments the `size` and creates a basis for the inner loop (DFS) to work with, until at least one basis of width k is found.

The inner loop does not need to check that the partial basis is inductive since the algorithm never generates a basis that is not inductive. It only has to test two conditions:

1. The order of the rows of the basis is non-decreasing.
2. The basis generates the first `width` bits of the input.

If at least one of these conditions is not satisfied—note that the efficiency of the algorithm depends on which condition we check first—the inner loop calls `increment()` on the basis.

`increment()` is the only method that changes the attributes of the basis. The increment method will use a `filter` to generate the "next" inductive basis; the `filter` records which bits are fixed and which are free choices. When all possibilities are exhausted, `increment()` will raise an overflow flag that will terminate the loop.

On the other hand if both conditions are satisfied, the inner loop will do a recursive call with a new basis that is an extension of its basis. The method `extends` returns a new wider basis with a correct `filter`.

Thus our search invariant is as follows:

1. The rows form a partial inductive basis possibly containing duplicates; all the rows have the same width `width`.
2. The rows are nondecreasing lexicographically.
3. The rows generate $Init^{(\texttt{width})}(P)$.

Note that this approach cannot miss any of the desired bases:

1. The partial basis is inductive. By Theorem 6.1.8, if a basis is not inductive then no extension is inductive.

2. The rows are nondecreasing lexicographically. If a basis is *not* in nondecreasing order, then no extension is nondecreasing.
3. The basis generates $Init^{(\texttt{width})}(P)$. By Theorem 6.1.6, if a basis does not generate a partial, then no extension will generate the partial.

The analysis of the invariant implies that all relevant bases are considered, proving the correctness of the algorithm and that no basis is considered more than once, proving optimality. Note however that we must allow duplicates among the rows.

We only need to reason that all bases are minimal and this is proved by the fact that the size of the basis is increased only when all bases of the previous size have been tried and rejected.

Notice that eventually we will overflow when we increment the very first column. At that point we will have found all partial inductive bases of size `size`. If any were found, then we are done. Otherwise we increment `size` and start again.

So we can describe our algorithm as a sequence of depth first searches on trees of increasing branching, until some solution leaves are reached. The outer loop of the algorithm keeps increasing `size`, starting with 0, until solutions are found by the depth-first search. As a result, all the partial inductive bases that we output are minimal, and we are guaranteed to find all of them.

Notice, incidentally, that we will never output a partial inductive basis with duplicate elements. Such a basis can always be made smaller by eliminating the duplicates, which means that the search would have succeeded already with a smaller value of `size`. In terms of NFAs, as noted in Chapter 3, minimal NFAs do not have repeated prophecies; of course the converse is not true, as there are many non-minimal NFAs with no repeated prophecies.

What is the significance of the partial inductive bases that we find? Recalling the results of Chapters 5 and 6, we know that these partial inductive bases are

essentially equivalent to the set of all minimal NFAs recognizing some *extension* of the input partial P. The size of these bases is thus a lower bound on the size of the minimal NFAs recognizing any *extension* of P. Even more, if P is *unambiguous* (Definition 5.2.2) then all NFAs that realize any of the bases produced by the algorithm recognize the same language. And if P is the *canonical partial of* L (Definition 5.2.3) then all the NFAs recognize L.

7.2 Classes of `Ibas`

An object of class `Basis` holds an array of `size` objects of class `Partial`, each of equal `width`. We now describe the classes of `ibas` in more detail:

1. Class `ibas`

   Attributes:

   `Partial language` holds a partial of the language to be recognized; Integer `basesFound` counts the inductive bases that generate the partial `language`.

   Methods:

   `main(String[])` creates objects of class `Basis` in increasing order of `size`, starting at 0, and calls recursive method `DFS(Basis)` until a solution is found.

   `DFS(Basis)` checks that the invariant is satisfied; if so, it makes a recursive call with an extended basis; if not, it increments the last column. When increment overflows, it returns the number of bases found.

2. Class `Basis`

   Attributes:

   Integer constants `width` and `size`, Boolean `overflow`, Array [`size`] of `Partial`, Array [`width`] of bit.

Methods:

`increment()` will generate the next node in the DFS tree by incrementing the last column; if all combinations have already been generated, it sets `overflow` to true. This is the only method that changes attributes of the class. It uses the array `filter` to avoid generating non-inductive bases. It proceeds by looking at the first non-filtered bit; if is 0, it will change it to 1 and return; if it is 1, it will change it to 0 and continue to the next bit.

`extend()` returns a new `Basis` of width `width+1` and generates a new filter.

`nondecreasing()` and `makeLanguage()` check that the invariant still holds.

3. Class `Partial`

   Holds a $0, 1$ array and contains all the functionalities of Partial that are used by the algorithm.

We present pseudo-Java code for `ibas`. Several implementations of this algorithm have been written in C; we do not here describe the use of low-level bit operations and other techniques that improve running time, as our goal here is to present a readable description of the algorithm without considering performance issues.

The code presented next works for unary and binary implementations, the only method that must be changed is `extend`. Our algorithm does not deal directly with strings in Σ^*, but indirectly by considering only their lexicographical positions.

7.3 Ibas Algorithm

```
public class ibas {
  private Language language = new Language(args[0]);
  int basesFound = 0;
```

```
  public void main(String[] args) {
    Basis basis = new Basis(0);  // new basis of size 0, width 0
    while (basesFound == 0) {
      basesFound = DFS(basis);
      if (basesFound == 0)
        basis = new Basis(basis.size+1);
    }
    print(basesFound);
  }
  int DFS(Basis basis) {
    int basesFound = 0;
    while (! basis.overflow) {
      if (basis.nondecreasing() && basis.makeLanguage(language)) {
        // All conditions are met
        if (basis.width() == language.width()) {
          basesFound++;
          basis.print();
        }
        else basesFound += DFS(basis.extend());
      }
      // Next basis of the same width
      basis.increment();
    }
    return basesFound;
  }
}
```

7.4 Some Methods of Class `Basis`

We now proceed to give a less formal discussion of the methods of the class `Basis`.

The `nondecreasing` method is just a comparison of the elements of the basis. In our C language implementation each element of the basis is just an integer, so the ordering of the basis reduces to a simple integer comparison.

The `makeLanguage` method is also easy to implement; first cut the partial of `language` to the same width as `basis` and then select all the elements that are subsets of the language, take their union, and compare with the language.

The `increment` method is little more complex. The first time it is called, all the free choices are set to 0 and all the fixed values are set; we then look at the first free choice and set it to 1, and are done; the second time we look at the first free choice and since is a 1 we set it to 0 and look for the second free choice. In general we look through the free choices in order; if we find a 1 then we set it to 0 and go on to the next; if we find a 0 we just set it to 1 and return. When all the free choices are back to 0 we then set `overflow` to true.

The `extend` method is implemented differently for unary and binary alphabets. This method is actually a constructor—it uses a partial inductive basis to build a wider partial inductive basis. It needs to generate the next column of the basis in a way that preserves inductivity; this means that it needs to know the quotient partial of each of the rows of the new basis. Each row will be first extended with a 0. We check whether the quotient partial is the union of some rows; if not, we know that this value is a fixed 1 and we set it; we also mark it fixed in the `filter` so that `increment` will know not to change it. If so, (the row could be extended by 0) we then try to extend it by 1; if that fails than the row is a fixed 0, so we set it and mark it fixed in the `filter`. Finally, if it can be extended with both 0 and 1, we

then know that it is a free choice; we set as 0 and we do not mark it as fixed in the `filter` so that `increment` is free to change it.

The only difficult part is to calculate the quotient partial for binary languages. We note that in the lexicographical order we first have ϵ, which does not begins with a or b, and then we have 1 string that begins with a and 1 string that begins with b, and then we have 2 with a and 2 with b, and then 4 and 4, and then 8 and 8, and so on. We use a counter twice and then we double the upper bound; in this way we always know whether the string begins with a or with b. We keep two auxiliary matrices; one for the left quotient $a\backslash P$ and another for $b\backslash P$, extending both matrices as we extend the basis and setting the `filter`; once the new `Basis` is built with all its attributes (`basis`, `aQuotient`, `bQuotient`, and `filter`) we return it to be used by `DFS`.

CHAPTER 8

EXPERIMENTAL RESULTS WITH IBAS

In this chapter, we present some experimental results to show the effectiveness of searching for minimal NFAs recognizing a given partial by searching for minimal partial inductive bases generating the partial.

8.1 Limits on Exhaustive Search

In this section, we explore what we are able to achieve through exhaustive search for unary NFAs.

The first question is how many n-state NFAs need to be enumerated in an exhaustive search. Given a unary alphabet, notice that we have for each state a total of $n + 2$ independent yes/no choices. (*Is it initial? Is it final? Does it have an arrow to* q_1*? To* q_2*? ... To* q_n*?*) Hence there are 2^{n+2} possibilities for each state. This implies that it suffices to enumerate at most $(2^{n+2})^n = 2^{n^2+2n}$ NFAs. But of course this number is too large, since many of those NFAs will be isomorphic to one another.

In fact, determining the number of non-isomorphic n-state NFAs is a long-standing open problem. However, the answer to that question is of only incidental interest to us here, because what we really want is a way of efficiently *enumerating* the non-isomorphic NFAs. Since we do not know how to do that, we instead use an enumeration that tries to generate as few isomorphic NFAs as possible.

The best general enumeration that we have found works by observing that *three* of each state's yes/no choices are "local" in the sense that they do not require distinguishing among the states. The local choices for a state are: *Is it initial?*, *Is it final?*, and *Does it have a self loop?*. (In contrast, the choice of whether the state has an arrow to some state q is not local, because it requires distinguishing

state q.) The combination of these three choices leads to $2^3 = 8$ kinds of states, and we can assume without loss of generality that the NFA's states are ordered into 8 consecutive groups:

1. initial, final, self loop
2. initial, final, no self loop
3. initial, nonfinal, self loop
4. initial, nonfinal, no self loop
5. noninitial, final, self loop
6. noninitial, final, no self loop
7. noninitial, nonfinal, self loop
8. noninitial, nonfinal, no self loop

In how many ways can this "local" data be chosen? We notice that our situation can be viewed as an "Occupancy Problem" as discussed in Section II.5 of Feller [Fel68]. We want to know how in many ways we can place n indistinguishable "balls" (the states) into 8 "bins" (the possibilities for each state).

In general, the number of ways of putting n indistinguishable balls into k bins turns out to be the binomial coefficient

$$\left(\begin{array}{c} k+n-1 \\ n \end{array} \right).$$

To see this, note that each such placement can be represented as a string of n stars (representing the balls) with $k-1$ bars inserted (representing the boundaries between the bins). For example, with $n = 5$ and $k = 4$, the string

$$**|*||**$$

represents the case when we put 2 balls in the first bin, 1 ball in the second bin, 0 balls in the third bin, and 2 balls in the fourth bin. If the symbols were all distinguishable, then the number of such strings would be $(n + k - 1)!$. But since the n stars and $k - 1$ bars are indistinguishable, then the total number of strings is

$$\frac{(n + k - 1)!}{n!(k - 1)!},$$

which is equal to the above binomial coefficient.

Applying this result, we see that the number of ways of choosing the "local" data of an n-state unary NFAs is

$$\left(\begin{array}{c} n + 7 \\ n \end{array} \right),$$

as compared with 2^{3n}, which is what pure brute force would give.

Having chosen the "local" data for the NFA, we then must add the arrows *between* states; we cannot see a way to avoid doing this in all possible ways. So, since each state may or may not have an arrow to each of the other $n - 1$ states, we get $2^{n(n-1)}$ ways of drawing the arrows.

In total, the number of n-state unary NFAs generated is

$$\left(\begin{array}{c} n + 7 \\ n \end{array} \right) 2^{n(n-1)}.$$

The value of this formula for n up to 10 is given in Figure 8.1. We remark that we have been able to compute the number of non-isomorphic NFAs of size 0, 1, 2, and 3; the number is 1, 8, 138, and 5,728, respectively. Hence our enumeration is quite close to optimal on these tiny cases; but it must get farther and farther from optimal as the number of states increases.

n	NFAs enumerated
0	1
1	8
2	144
3	7,680
4	1,351,680
5	830,472,192
6	1,842,540,969,984
7	15,094,095,626,108,928
8	463,690,617,634,066,268,160
9	54,023,872,564,028,741,244,682,240
10	24,075,457,884,022,075,586,238,167,908,352

Figure 8.1: Number of n-state unary NFAs under our best enumeration

We experimented with carefully optimized C programs to search for 5-state and 6-state NFAs recognizing the (more or less arbitrarily chosen) partial

$$(01111111101111111101)$$

which corresponds to the set of strings of a's whose length is not a multiple of 9. Running on a 2.66 GHz Pentium 4 processor, we were able to complete the search for 5-state NFAs in 7.4 seconds, and the search for 6-state NFAs in 4.2 hours. Referring to Figure 8.1, the 5-state search space has size 830,472,192; hence we were able to check 112 million NFAs per second. The 6-state search space has size 1,842,540,969,984; hence that search checked 122 million NFAs per second.

Extrapolating to the case of 7-state NFAs, if we assume that we could check 122 million NFAs per second, then the search would require about 4 years. Under the same assumptions, the search for NFAs with 8, 9, and 10 states would respectively take 120,000 years, 14 billion years, and 6 quadrillion years.

We remark that we can actually reduce these times by a factor of 2 or so by choosing the initial and final states *prior* to choosing the arrows. For if the partial we are trying to recognize contains ϵ, then some initial state must be final. Hence if

I and F are disjoint, then the NFA cannot possibly recognize the partial, and we can discard it before ever choosing the arrows. Similarly, if the partial does not contain ϵ, then we can immediately discard all NFAs in which I and F are not disjoint.

8.2 Results with `ibas` on Unary Languages

In this section, we report on some experimental results obtained with `ibas.c`, a carefully optimized C implementation of the unary version of the `ibas` algorithm. The implementation limits the width of all partials to at most 31; as a result, operations like union and left quotient can be done in just one or two machine instructions. Our experiments were run on a 2.66 GHz Pentium 4 processor. Note that memory should not be an issue at all, as our implementation maintains only a few thousand bytes of data.

Suppose we wish to find the minimal NFAs recognizing the language $a(a^4)^* \cup (a^6)^*$. To do this, we run `ibas` with a suitable partial:

```
ibas 1100011001001100011001000
```

In about 37 minutes, the run terminates, producing the output shown in Figure 8.2. Because `ibas` determined that the smallest partial inductive basis generating partial (1100011001001100011001000) has size 10, we know that the minimal NFAs recognizing $a(a^4)^* \cup (a^6)^*$ must have at least 10 states. In addition, `ibas` prints a description of the NFAs realizing each partial inductive basis that it finds. In the description, the notation "`--> *8 | 0`" indicates that state 8 is both initial and final, and $\delta(8, a) = \{0\}$. Notice that the NFA in Figure 8.2 consists of two rings of states, one of size 6 and the other of size 4. Also notice that in this case `ibas` prints `NFA completely determined`; this means that there is only *one* NFA realizing that basis. Of course, the NFAs found by `ibas` might not recognize the language that we

```
% ibas 11000110010011000110100
No inductive bases of size 0.
No inductive bases of size 1.
No inductive bases of size 2.
No inductive bases of size 3.
No inductive bases of size 4.
No inductive bases of size 5.
No inductive bases of size 6.
No inductive bases of size 7.
No inductive bases of size 8.
No inductive bases of size 9.
Found an inductive basis of size 10:
  000001000001000001000001
  000010000010000010000010
  000100000100000100000100
  000100010001000100010001
  001000001000001000001000
  001000100010001000100010
  010000010000010000010000
  010001000100010001000100
  100000100000100000100000
  100010001000100010001000
Corresponding NFA:
       0 | 1
       1 | 2
       2 | 4
       3 | 5
       4 | 6
       5 | 7
       6 | 8
  -->  7 | 9
  --> *8 | 0
      *9 | 3
  NFA completely determined.

There are 1 inductive bases of size 10.
```

Figure 8.2: Sample run of `ibas`

want—we know only that their languages will *extend* the partial that we input to `ibas`. In this case, however, it is easy to see that the NFA does recognize the desired language—hence we now know that $a(a^4)^* \cup (a^6)^*$ has a unique minimal NFA, which has 10 states.

In general, of course, there could be more than one NFA realizing the partial inductive bases that `ibas` finds. In such cases, `ibas` displays "optional" arrows with parentheses. Here is an example:

```
Found an inductive basis of size 2:
  0111111
  1111111
Corresponding NFA:
  -->  0 |(0) 1
      *1 |(0) 1
  Arrows not completely determined.
```

Notice that the left quotient of (0111111) is (111111). To generate this, row 1 is *necessary*, while row 0 is *optional*. For this reason, `ibas` prints "`0 |(0) 1`", indicating that state 0 *must* have an arrow to state 1 and *can* have an arrow to state 0. Note however that "optional" arrows may not be completely optional. Consider the following example:

```
Found an inductive basis of size 4:
  001111
  011111
  100111
  110011
```

```
Corresponding NFA:
       0 |(0) 1
       1 |(0)(1)(2)(3)
      *2 | 0
  --> *3 | 2
  Arrows not completely determined.
```

Look at the arrows out of state 1. The left quotient of (011111) is (11111), which can be generated in many ways using rows 0, 1, 2, and 3: $0 \cup 3$, $1 \cup 2$, $0 \cup 3$, etc. None of the rows are *necessary*, which is why `ibas` puts parentheses around each of them. But in fact at least two are them are needed to generate the quotient (and only certain pairs work).

Initial states can be optional in exactly the same way; `ibas` notes such states with "`(-->)`". For example, on `ibas 1110` we get the following basis:

```
Found an inductive basis of size 3:
  0100
  0110
  1000
Corresponding NFA:
 (-->) 0 | 2
  -->  1 | 0  2
  --> *2 |
  Initial states not completely determined.
```

Here we see that to generate the input partial (1110) we need to use rows 1 and 2, but row 0 is optional.

r	Minimum Basis Size	Number of Bases	Run Time
1	0	1	0 sec
2	2	1	0 sec
3	3	2	0 sec
4	4	3	0 sec
5	5	6	0.002 sec
6	5	2	0.003 sec
7	7	18	0.5 sec
8	8	30	13 sec
9	9	56	6.5 min
10	7	6	1.1 sec
11	11	≥ 1	$>$ 24 hrs
12	7	6	1.5 sec
14	9	18	16 min
15	8	12	42 sec
21	10	36	15.5 hrs

Figure 8.3: Results of `ibas` on $L_r = \{a^i \mid i \bmod r \neq 0\}$

One family of languages that we have explored rather thoroughly is

$$L_r = \{a^i \mid i \bmod r \neq 0\},$$

the set of strings of a's whose length is not a multiple of r, for $r \geq 1$. (Note that $L_1 = \emptyset$.) L_r can of course be recognized using a ring of r states, but fewer states are sufficient if r has at least 2 distinct prime factors; for example, Figure 2.1 shows a 5-state NFA for L_6. In general, if $r = pq$ where p and q are relatively prime, then L_r can be recognized by an NFA with $p + q$ states. We ran `ibas` using the width-30 initial partial of L_r, for various values of r. For example, the run for L_9 was

```
ibas 011111111011111111011111111011
```

The results of our experiments are shown in Figure 8.3. (The run when $r = 11$ did not finish—after 24 hours, `ibas` had determined that there are no inductive bases of size 10 or smaller, and it had found 1 inductive basis of size 11.) We suspect that these languages are relatively difficult cases for `ibas`, because they contain so

many more 1's than 0's, which may make `ibas`'s pruning less effective. In contrast, the run shown in Figure 8.2, involving a more "balanced" partial, found all 10-state inductive bases in just 37 minutes.

On $(a^3 \cup a^5)(a^{10})^*$, `ibas` finds 2 minimal inductive bases of size 10, taking 16 minutes.

Another interesting language to try is $(a^5 \cup a^9)^*$. The minimal DFA for this language has 33 states, because the longest string of a's that is *not* in the language is a^{31}. On the width-30 initial partial for this language, `ibas` finds 27 minimal inductive bases of size 9, taking 58 seconds. The first of these inductive bases is

000000001000010001100011001100

000000010000100011000110011001

000000100001000110001100111001

000001000010001100011001110011

000010000100011000110011100111

000010000100011000110011100110

000100000000100001000110001100

001000000001000010001100011001

010000000010000100011000110011

100000000100001000110001100111

The unique NFA realizing it is shown in Figure 8.4.

On $(a^3 \cup a^{11})^*$, `ibas` finds 2,225 minimal inductive bases of size 11, taking about 19 hours. So `ibas` can sometimes find minimal 11-state NFAs, if we are patient.

Curiously, on the apparently similar language $(a^2 \cup a^{11})^*$, `ibas` finds that 6 states are enough! One of the NFAs that it finds is shown in Figure 8.5.

On the finite language $\{a, a^2, a^6, a^8, a^9\}$, `ibas` finds 443 minimal inductive bases of size 10, taking 22 minutes. Notice that the minimal DFA for this language contains 10 states (if we omit the "dead" state); hence nondeterminism does not let

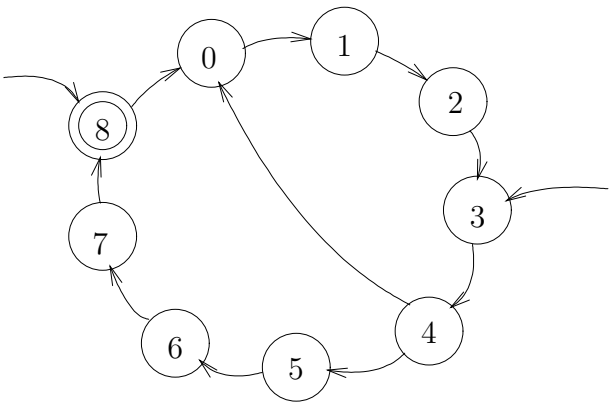

Figure 8.4: A minimal NFA for $(a^5 \cup a^9)^*$

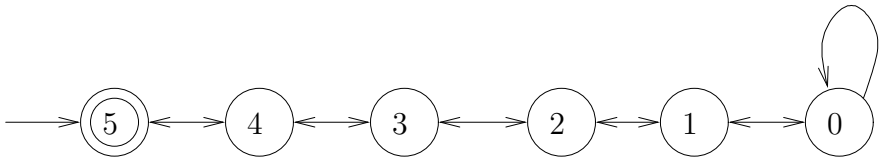

Figure 8.5: A minimal NFA for $(a^2 \cup a^{11})^*$

us construct a smaller machine for this language. Looking ahead, in Section 9.8 we will show that this holds in general—the minimal DFA for a finite unary language is also a minimal NFA.

As a final topic, it is interesting to compare the size of `ibas`'s search space with that of the exhaustive search considered in Section 8.1. On the language

$$L_8 = \{a^i \mid i \bmod 8 \neq 0\},$$

profiling reveals that in 13 seconds `ibas` considers about 50 million partial inductive bases of size 8, checking about 4 million bases per second. In contrast, exhaustive search can check about 120 million NFAs per second. But the search space of 8-state NFAs (using the enumeration in Section 8.1) has size about $4.6 \cdot 10^{20}$, which is 9 trillion times as large.

size \ width	5	10	15	20	30
0	1	1	1	1	1
1	3	3	3	3	3
2	12	12	12	12	12
3	65	65	65	65	65
4	472	483	483	483	483
5	3,914	4,745	4,745	4,745	4,745
6	31,913	58,788	58,791	58,791	58,791
7	237,128	896,526	896,999	896,999	896,999
8	1,566,659	16,784,652	16,827,226	16,827,241	16,827,241
9	9,200,281	388,630,823	391,949,222	391,952,601	391,952,607

Figure 8.6: Number of Unary Partial Inductive Bases of Various Sizes and Widths

More systematically, we can modify `ibas` so that it generates partial inductive bases *without* checking that they generate a specified partial; in this way, we can get a count of how many unary partial inductive bases of each size and width there are in total. Some experimental results are shown in Figure 8.6; it is striking how small these numbers are as compared with the numbers in Figure 8.1. It is also interesting to note how little the number of bases grows as the *width* increases beyond 10—even on bases of size 9, increasing the width from 10 to 30 increases the count by less than one percent. This suggests that almost all partial inductive bases of size 9 and width 10 are uniquely extendible.

CHAPTER 9

THE CHARACTERISTIC MATRIX AND LOWER BOUNDS

In this chapter, we discuss some work that we did prior to the development of inductive bases. We explore minimal NFAs from an algebraic standpoint, using a variant of the *characteristic matrix* C_L of a language L, considered earlier by Condon, Hellerstein, Pottle, and Wigderson [CHPW98]. It turns out that the structure of C_L reveals interesting bounds on finite automata recognizing L. For example, we can recast the Myhill-Nerode Theorem to this setting, showing that the number of states in the minimal DFA recognizing L is given by the number of distinct nonzero rows in C_L. More interestingly, we show that any NFA recognizing L gives rise to a factorization of C_L; as a result we are able to deduce lower bounds on the size of any NFA recognizing L. For instance, the number of states in any UFA (unambiguous finite automaton) recognizing L must be at least the *rank* of C_L.

It should be noted that most of the results in this chapter have appeared previously in the literature, in works such as [Sch78, GS96, CHPW98, HS01, HSK$^+$02]. In particular, Condon, Hellerstein, Pottle, and Wigderson [CHPW98] seem to use the factorization of C_L implicitly in some of their arguments, although they do not state it explicitly. Hromkovič et al. [HS01, HSK$^+$02] also consider the same characteristic matrix; they use methods from communication complexity to obtain bounds similar to those described here. We believe, however, that our factorization-based approach has the advantage of giving simpler and more self-contained derivations of the lower bounds. We will discuss related work more fully as we proceed.

The chapter is organized as follows. Section 9.1 develops the characteristic matrix and its factorization. The following sections develop a number of applications, including a proof of the Myhill-Nerode theorem, lower bounds based on the rank of the computational matrix, and theorems about bideterminism.

9.1 The Characteristic Matrix and its Factorization

In this section, we define the characteristic matrix C_L and explore its properties.

Definition 9.1.1 *Given a language L over alphabet Σ, the* characteristic matrix of *L is an $\infty \times \infty$ binary matrix C_L, indexed by Σ^*, such that*

$$C_L[x, y] = \begin{cases} 1 & \text{if } xy^R \in L \\ 0 & \text{otherwise.} \end{cases}$$

Let us begin with some examples. Figure 9.1 shows the upper left-hand portion of the characteristic matrix for $\{a^i \mid i \bmod 6 \neq 0\}$, the language recognized by the NFA in Figure 2.1. And Figure 9.2 shows the upper left-hand portion of the characteristic matrix for

$$\{w \in \{a, b\}^* \mid \text{the 3rd-from-last symbol of } w \text{ is } b\},$$

the language recognized by the UFA in Figure 2.2.

Note that the characteristic matrix has a very restricted structure:

Theorem 9.1.2 *For all $x, y \in \Sigma^*$ and $a \in \Sigma$, $C_L[xa, y] = C_L[x, ya]$.*

Proof. $C_L[xa, y] = 1$ iff $xay^R \in L$ iff $x(ya)^R \in L$ iff $C_L[x, ya] = 1$. □

Our characteristic matrix is a variant of the one considered by Condon et al. [CHPW98] and Hromkovič et al. [HS01, HSK+02], who define $M_L[x, y] = 1$ iff $xy \in L$. In contrast, we use xy^R instead, which just permutes the columns. An advantage of our definition of C_L is that the characteristic matrix for the reverse of L is just the transpose of the characteristic matrix for L:

Theorem 9.1.3 *For any L, $(C_L)^T = C_{L^R}$.*

	ϵ	a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9
ϵ	0	1	1	1	1	1	0	1	1	1
a	1	1	1	1	1	0	1	1	1	1
a^2	1	1	1	1	0	1	1	1	1	1
a^3	1	1	1	0	1	1	1	1	1	0
a^4	1	1	0	1	1	1	1	1	0	1
a^5	1	0	1	1	1	1	1	0	1	1
a^6	0	1	1	1	1	1	0	1	1	1
a^7	1	1	1	1	1	0	1	1	1	1
a^8	1	1	1	1	0	1	1	1	1	1
a^9	1	1	1	0	1	1	1	1	1	0

Figure 9.1: Characteristic matrix for $\{a^i \mid i \bmod 6 \neq 0\}$

	ϵ	a	b	aa	ab	ba	bb	aaa	aab	aba	abb	baa	bab	bba	bbb
ϵ	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1
a	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1
b	0	0	0	1	1	1	1	0	1	0	1	0	1	0	1
aa	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1
ab	0	0	0	1	1	1	1	0	1	0	1	0	1	0	1
ba	0	1	1	0	0	0	0	0	1	0	1	0	1	0	1
bb	0	1	1	1	1	1	1	0	1	0	1	0	1	0	1
aaa	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1
aab	0	0	0	1	1	1	1	0	1	0	1	0	1	0	1
aba	0	1	1	0	0	0	0	0	1	0	1	0	1	0	1
abb	0	1	1	1	1	1	1	0	1	0	1	0	1	0	1
baa	1	0	0	0	0	0	0	0	1	0	1	0	1	0	1
bab	1	0	0	1	1	1	1	0	1	0	1	0	1	0	1
bba	1	1	1	0	0	0	0	0	1	0	1	0	1	0	1
bbb	1	1	1	1	1	1	1	0	1	0	1	0	1	0	1

Figure 9.2: Characteristic matrix for the "third from last" language

Proof. We have $(C_L)^T[x, y] = 1$ iff $C_L[y, x] = 1$ iff $yx^R \in L$ iff $(yx^R)^R \in L^R$ iff $xy^R \in L^R$. $\square$

Before we can prove our factorization theorem, we first need an auxiliary result. Intuitively, it says that running an NFA M on a string w gives the same result as splitting w into two pieces arbitrarily, running M on the first piece, running M^R on the reverse of the second piece, and seeing whether any common states can be reached:

Theorem 9.1.4 (Split Computation) *For any NFA M and strings $x, y \in \Sigma^*$, we have that M accepts xy^R iff $\widehat{\delta}(I, x) \cap \widehat{\delta^R}(F, y) \neq \emptyset$. Moreover, if M is a UFA, then the intersection contains at most one state.*

Proof. By Corollary 2.1.4, NFA M accepts xy^R iff M has a path labeled xy^R from some state in I to some state in F. This holds iff M has a path labeled x from some state in I to some intermediate state q and a path labeled y^R from q to some state in F. Equivalently, M has a path labeled x from some state in I to some state q and M^R has a path labeled y from some state in F to q. Equivalently (by Lemma 2.1.3), there is a q such that $q \in \widehat{\delta}(I, x) \cap \widehat{\delta^R}(F, y)$. Finally, note that if M is a UFA, then the intersection contains at most one state, because xy^R has at most one accepting path. $\square$

Now we are ready to show that an NFA recognizing L gives rise to a factorization of C_L.

Definition 9.1.5 *Given an NFA $M = (Q, \Sigma, \delta, I, F)$, where $|Q| = n$, define Δ_M to be an $\infty \times n$ matrix, indexed by Σ^* and Q, as follows:*

$$\Delta_M[x, q] = \begin{cases} 1 & \text{if } q \in \widehat{\delta}(I, x) \\ 0 & \text{otherwise.} \end{cases}$$

Also, define Δ_M^R *to be an* $n \times \infty$ *matrix, indexed by* Q *and* Σ^**, as follows:*

$$\Delta_M^R[q, y] = \begin{cases} 1 & \text{if } q \in \widehat{\delta^R}(F, y) \\ 0 & \text{otherwise.} \end{cases}$$

For example, Figure 9.3 gives the upper portion of Δ_M for the UFA of Figure 2.2, which recognizes $\{w \in \{a, b\}^* \mid \text{the 3rd-from-last symbol of } w \text{ is } b\}$. And Figure 9.4 gives the left-hand portion of Δ_M^R for the same UFA.

Notice that the rows and columns of Δ_M and Δ_M^R are the property vectors of a number of interesting sets:

- Row w of Δ_M, which we denote by $\Delta_M[w, -]$, is the property vector of $\widehat{\delta}(I, w)$.
- Column q of Δ_M, denoted $\Delta_M[-, q]$, is the property vector of $Hist(q)$.
- Column w of Δ_M^R, denoted $\Delta_M^R[-, w]$, is the property vector of $\widehat{\delta^R}(F, w)$.
- Row q of Δ_M^R, denoted $\Delta_M^R[q, -]$, is the property vector of $Proph(q)^R$.

For example, referring to Figure 9.3, $\Delta_M[bab, -] = (1, 1, 0, 1)$, which represents the set $\{0, 1, 3\}$, indicating that after scanning *bab*, the UFA in Figure 2.2 can be in states 0, 1, or 3. And $\Delta_M[-, 2] = (0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, \ldots)$, indicating that $Hist(2) = \{ba, bb, aba, abb, bba, bbb, \ldots\}$.

More interestingly, if we compute the matrix product of Δ_M and Δ_M^R in Figures 9.3 and 9.4, we find that the result is precisely the characteristic matrix for $L(M)$, which is given in Figure 9.2. This is the subject of our main theorem. But first we need a definition:

Definition 9.1.6 *Nonnegative integer matrices* U *and* V *are* congruent*, written* $U \cong V$*, if* $U[x, y] = 0$ *iff* $V[x, y] = 0$*.*

It is easy to see that congruence is an equivalence relation. Now we present our factorization theorem.

	0	1	2	3
ϵ	1	0	0	0
a	1	0	0	0
b	1	1	0	0
aa	1	0	0	0
ab	1	1	0	0
ba	1	0	1	0
bb	1	1	1	0
aaa	1	0	0	0
aab	1	1	0	0
aba	1	0	1	0
abb	1	1	1	0
baa	1	0	0	1
bab	1	1	0	1
bba	1	0	1	1
bbb	1	1	1	1

Figure 9.3: Δ_M for the UFA of Figure 2.2

	ϵ	a	b	aa	ab	ba	bb	aaa	aab	aba	abb	baa	bab	bba	bbb
0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1
1	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9.4: Δ_M^R for the UFA of Figure 2.2

Theorem 9.1.7 (Factorization) *For any NFA M, $\Delta_M \times \Delta_M^R \cong C_{L(M)}$. Moreover, $\Delta_M \times \Delta_M^R = C_{L(M)}$ iff M is a UFA.*

Proof. Let $M = (Q, \Sigma, \delta, I, F)$ be an NFA. By the definition of matrix multiplication, $(\Delta_M \times \Delta_M^R)[x, y]$ is the inner product of $\Delta_M[x, -]$ and $\Delta_M^R[-, y]$.

As observed above, $\Delta_M[x, -]$ is the property vector of $\widehat{\delta}(I, x)$, and $\Delta_M^R[-, y]$ is the property vector $\widehat{\delta^R}(F, y)$. Hence the inner product of $\Delta_M[x, -]$ and $\Delta_M^R[-, y]$ gives the cardinality of $\widehat{\delta}(I, x) \cap \widehat{\delta^R}(F, y)$.

And by Theorem 9.1.4 (Split Computation), $\widehat{\delta}(I, x) \cap \widehat{\delta^R}(F, y) \neq \emptyset$ iff $xy^R \in L(M)$. Hence $(\Delta_M \times \Delta_M^R)[x, y] \neq 0$ iff $xy^R \in L(M)$, which implies that $\Delta_M \times \Delta_M^R \cong C_{L(M)}$.

Moreover, if M is a UFA then the cardinality of the intersection is at most 1, and therefore $\Delta_M \times \Delta_M^R = C_{L(M)}$. And if M is not a UFA then there exists a string $w \in L(M)$ with at least two accepting paths. Then there must exist distinct states q and q' and a splitting of w into xy^R such that there is a path from some state in I to q labeled x, a path from q to some state in F labeled y^R, a path from some state in I to q' labeled x, and a path from q' to some state in F labeled y^R. Hence both q and q' are elements of $\widehat{\delta}(I, x) \cap \widehat{\delta^R}(F, y)$, which implies that $(\Delta_M \times \Delta_M^R)[x, y] \geq 2$. □

As far as we know, the Factorization theorem has not appeared explicitly in prior work. However, it seems to be used implicitly in some of the arguments in Condon et al.—see for example the discussion after Theorem 4.2 [CHPW98, p. 750].

Many results in automata theory can be expressed in a uniform and concise form in terms of the characteristic matrix and its factorization. In the following sections, we develop some of these.

9.2 A Proof of the Myhill-Nerode Theorem

The Myhill-Nerode Theorem [HU79, Theorem 3.9] is based on an equivalence relation R_L on Σ^*, defined by xR_Ly iff for all z, $(xz \in L \leftrightarrow yz \in L)$. It is easy to see that xR_Ly iff rows x and y of C_L are the same. It follows that the number of states in the minimal DFA recognizing L is the number of distinct rows in C_L. This concrete reformulation of the Myhill-Nerode Theorem was previously noted by Hromkovič and Schnitger [HS01], but here we observe that Theorem 9.1.7 can be applied to give a nice proof of the theorem:

Theorem 9.2.1 *For any language L over alphabet Σ, let r be the number of distinct nonzero rows in C_L. If r is infinite, then L is not regular. And if r is finite, then L is regular and r is the number of states in the minimal DFA recognizing L; moreover, this minimal DFA is unique up to isomorphism.*

Proof. Suppose that L is recognized by some DFA M. By Theorem 9.1.7, $\Delta_M \times \Delta_M^R = C_L$. Now, because M is deterministic, each row of Δ_M contains at most one 1. Hence each row of the product matrix $\Delta_M \times \Delta_M^R$ is either all zero or else it is simply one of the rows of Δ_M^R. Therefore, every nonzero row of C_L must appear at least once as a row of Δ_M^R. This implies that M must have at least r states. In particular, if r is infinite then no such M can exist.

But if r is finite, then we can construct an r-state DFA $M_0 = (Q_0, \Sigma, \delta_0, I_0, F_0)$ that recognizes L, where

- Q_0 is the set of distinct nonzero rows of C_L
- $\delta_0(C_L[w,-],a) = \begin{cases} \{C_L[wa,-]\} & \text{if } C_L[wa,-] \text{ is nonzero} \\ \emptyset & \text{otherwise} \end{cases}$

- $I_0 = \begin{cases} \{C_L[\epsilon, -]\} & \text{if } C_L[\epsilon, -] \text{ is nonzero} \\ \emptyset & \text{otherwise} \end{cases}$
- $F_0 = \{C_L[w, -] \mid C_L[w, \epsilon] = 1\}$.

Notice that δ_0 is well defined: if $C_L[w, -] = C_L[w', -]$, then it follows from Theorem 9.1.2 that $C_L[wa, -] = C_L[w'a, -]$. Also notice that an all-zero row of C_L would be a "dead" state; our construction of M_0 takes care to eliminate such states. It is easy to see that M_0 is a DFA and that for all w,

$$\widehat{\delta_0}(I_0, w) = \begin{cases} \{C_L[w, -]\} & \text{if } C_L[w, -] \text{ is nonzero} \\ \emptyset & \text{otherwise.} \end{cases}$$

Hence M_0 accepts w iff $C_L[w, -]$ is nonzero and $C_L[w, -] \in F_0$ iff $C_L[w, \epsilon] = 1$ iff $w\epsilon^R \in L$ iff $w \in L$. Hence $L(M_0) = L$.

Finally, we can argue that any r-state DFA $M = (Q, \Sigma, \delta, I, F)$ recognizing L must be isomorphic to M_0. By the reasoning above, the r rows of Δ_M^R must simply be the r nonzero rows of C_L; hence an isomorphism can map state q of M to state $\Delta_M^R[q, -]$ of M_0. This mapping preserves the set of final states, because $q \in F$ iff $\Delta_M^R[q, \epsilon] = 1$ iff $\Delta_M^R[q, -] \in F_0$. To see that it also preserves the set of initial states and the transitions, first note that for any $w \in \Sigma^*$,

$$C_L[w, -] = \sum_{s \in Q} \Delta_M[w, s] \cdot \Delta_M^R[s, -]$$

but this sum contains at most one term, because each row of Δ_M contains at most one 1. Hence for any $w \in \Sigma^*$ and for any $s \in Q$, we have

$$C_L[w, -] = \Delta_M^R[s, -] \text{ iff } \Delta_M[w, s] = 1.$$

Hence the set of initial states is preserved: $q \in I$ iff $\Delta_M[\epsilon, q] = 1$ iff $C_L[\epsilon, -] = \Delta_M^R[q, -]$ iff $\Delta_M^R[q, -] \in I_0$. To argue that the set of transitions is preserved, let $q \in Q$

be arbitrary. We know that $\Delta_M^R[q,-] = C_L[w,-]$ for some $w \in \Sigma^*$, which implies that $\Delta_M[w,q] = 1$. Hence we have $q' \in \delta(q,a)$ iff $\Delta_M[wa,q'] = 1$ iff $C_L[wa,-] = \Delta_M^R[q',-]$ iff $\Delta_M^R[q',-] \in \delta_0(\Delta_M^R[q,-],a)$. □

As an application, consider the characteristic matrix for $\{a^i \mid i \bmod 6 \neq 0\}$ in Figure 9.1. It has 6 distinct nonzero rows, which implies that this language requires a 6-state DFA. And the characteristic matrix for

$$\{w \in \{a,b\}^* \mid \text{the 3rd-from-last symbol of } w \text{ is } b\},$$

given in Figure 9.2, has 8 distinct nonzero rows, implying that this language requires an 8-state DFA. Interestingly, it has only 4 distinct nonzero columns, implying that the reverse language requires a 4-state DFA; indeed, the reverse of the UFA in Figure 2.2 is a DFA.

9.3 Lower Bounds Based on the Rank of C_L

While minimal DFAs are well understood, minimal UFAs and NFAs are more elusive. Here we show lower bounds on the number of states in UFAs and NFAs recognizing a language; our bounds are based on the *rank* of the characteristic matrix. (The *rank* of a matrix is the maximum number of linearly independent rows; see [CLRS01, Chapter 28] for further discussion.)

Theorem 9.3.1 *For any L, rank(C_L) is a lower bound on the number of states in any UFA recognizing L.*

Proof. If $M = (Q,\Sigma,\delta,I,F)$ is a UFA recognizing L, then by Theorem 9.1.7, $\Delta_M \times \Delta_M^R = C_L$. It follows that the rows of C_L are linear combinations of the rows of

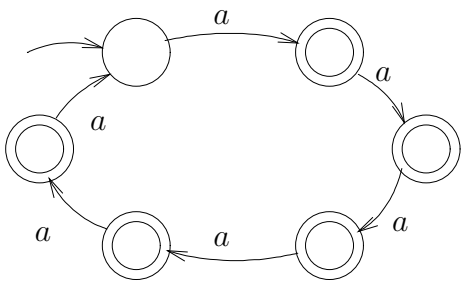

Figure 9.5: The minimal DFA for $\{a^i \mid i \bmod 6 \neq 0\}$

Δ_M^R, and the columns of C_L are linear combinations of the columns of Δ_M. More precisely,

$$C_L[w, -] = \sum_{q \in Q} \Delta_M[w, q] \cdot \Delta_M^R[q, -]$$

and

$$C_L[-, w] = \sum_{q \in Q} \Delta_M^R[q, w] \cdot \Delta_M[-, q].$$

Hence $rank(C_L)$, the maximum number of linearly independent rows (or columns) of C_L, is at most $|Q|$. □

The above bound was shown previously in [HSK+02] using results from communication complexity; here we get it as an immediate corollary to the factorization theorem. We can apply Theorem 9.3.1 to $\{a^i \mid i \bmod 6 \neq 0\}$, whose characteristic matrix is given in Figure 9.1. The rank of this matrix can be calculated to be 6. (Note that it suffices to consider the 6×6 submatrix in the upper left-hand corner.) This implies that the obvious 6-state DFA shown in Figure 9.5 is a minimal UFA for this language. (Note that the 5-state NFA in Figure 2.1 is not a UFA.) Interestingly, the *reverse* of the DFA in Figure 9.5 gives *another* minimal UFA for the language; it has five initial states and just one final state. This shows that minimal UFAs need not be unique.

Long ago, Schmidt used a similar technique to argue that, for each n, the minimal UFA recognizing the language

$$L(n) = \{xy \mid x, y \in \{0,1\}^n, x \neq y\}$$

requires at least 2^n states [Sch78, Theorem 3.9]. Restated in our framework, his argument is based on the observation that the submatrix of $C_{L(n)}$ indexed by strings of length n has rank 2^n, since it consists of 2^n distinct rows, each of which has the form $(1, 1, \ldots, 1, 0, 1, \ldots, 1)$.

As another application, the characteristic matrix in Figure 9.2 can be calculated to have rank 4. Therefore the UFA in Figure 2.2 is minimal.

Now let us consider general NFAs, rather than UFAs. If M is an NFA recognizing a language L, then by Theorem 9.1.7 we have $\Delta_M \times \Delta_M^R \cong C_L$. Because we have congruence rather than equality, the number of states in M can be smaller than $rank(C_L)$, because there may be a matrix congruent to C_L with lower rank. For example, the matrix consisting of repetitions of

$$\begin{matrix} 0 & 2 & 1 & 1 & 1 & 2 \\ 2 & 1 & 1 & 1 & 2 & 0 \\ 1 & 1 & 1 & 2 & 0 & 2 \\ 1 & 1 & 2 & 0 & 2 & 1 \\ 1 & 2 & 0 & 2 & 1 & 1 \\ 2 & 0 & 2 & 1 & 1 & 1 \end{matrix}$$

is congruent to the characteristic matrix in Figure 9.1, but its rank is at most 5, since it factors as $\Delta_M \times \Delta_M^R$ for the 5-state NFA given in Figure 2.1. (Actually, its rank turns out to be 4.)

Definition 9.3.2 *The* structural rank *of a matrix C is the smallest rank of any matrix congruent to C.*

Using the notion of structural rank, we can get a lower bound on NFAs:

Theorem 9.3.3 *For any L, the structural rank of C_L is a lower bound on the number of states in any NFA recognizing L.*

Proof. Similar to the proof of Theorem 9.3.1. □

Unfortunately, the structural rank of C_L is usually difficult to calculate. But sometimes it can be determined easily, or at least bounded from below. In these cases we can get bounds on minimal NFAs. Here are some examples.

Consider first the language $\{a^i \mid i \bmod 6 = 0\}$. Its characteristic matrix is the boolean complement of the matrix in Figure 9.1; it consists of repetitions of

$$\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{array}$$

Clearly the structural rank of this matrix is 6. Hence nondeterminism is useless in recognizing this language; the obvious 6-state DFA is a minimal NFA.

A special case of Theorem 9.3.3 has been previously noted. Glaister and Shallit [GS96] show that for any language L, if there exist strings $x_1, \ldots, x_n$ and $w_1, \ldots, w_n$ such that for all i, j, $x_i w_j \in L$ iff $i = j$, then any NFA recognizing L must have a least n states. Notice that under the hypotheses, we can permute the columns of C_L to form an $n \times n$ identity submatrix. This implies that the structural rank of C_L is at least n.

Here is a more general bound. Recall that a matrix is *unit lower triangular* if it has 1's on the main diagonal and 0's above.

Corollary 9.3.4 *Suppose that by permuting the rows and columns of C_L we can form an $n \times n$ unit lower triangular submatrix. Then any NFA recognizing L has at least n states.*

Proof. An $n \times n$ unit lower triangular matrix has structural rank n. □

As an application, consider the characteristic matrix in Figure 9.2. Form the 4×4 submatrix consisting of rows ϵ, b, ba, and bbb, and columns aab, aa, a, and ϵ:

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{matrix}$$

It follows that any NFA for $\{w \in \{a,b\}^* \mid \text{the 3rd-from-last symbol of } w \text{ is } b\}$ requires at least 4 states. Hence the UFA in Figure 2.2 is not just a minimal UFA for this language; it is also a minimal NFA.

We can also easily derive as corollaries Goldstine's quotient language theorems for minimal NFAs [GLW92]. Recall that the *quotient of L by L'*, written L/L', is $\{x \mid \text{there exists } y \in L' \text{ such that } xy \in L\}$.

Corollary 9.3.5 *If a language L has n pairwise-disjoint, nonempty quotients L/L_1, L/L_2, ..., L/L_n, then the minimal NFA recognizing L must have at least n states.*

Proof. First observe that $L/L' = \bigcup_{w \in L'} L/\{w\}$. Note also that column w of the characteristic matrix, $C_L[-, w]$, is a bit-vector representation of $L/\{w^R\}$. Hence, under the hypotheses, it is easy to see that there must be n pairwise-disjoint, nonzero columns in C_L. Hence the structural rank of C_L is at least n. □

Corollary 9.3.6 *If a language L has n nonempty quotients L/L_1, L/L_2, ..., L/L_n satisfying $L/L_1 \subset L/L_2 \subset \ldots \subset L/L_n$, then the minimal NFA recognizing L must have at least n states.*

	ϵ	a	a^2	a^3	a^4	a^5	a^6	a^7
ϵ	1	0	0	1	1	0	0	1
a	0	0	1	1	0	0	1	1
a^2	0	1	1	0	0	1	1	0
a^3	1	1	0	0	1	1	0	0
a^4	1	0	0	1	1	0	0	1
a^5	0	0	1	1	0	0	1	1
a^6	0	1	1	0	0	1	1	0
a^7	1	1	0	0	1	1	0	0

Figure 9.6: Characteristic matrix for $\{a^i \mid i \bmod 4 = 0 \text{ or } i \bmod 4 = 3\}$

Proof. By similar reasoning, we can form an $n \times n$ unit lower triangular submatrix of C_L. (Interestingly, in the case when each L_i is a singleton, we can also form an $(n-1) \times (n-1)$ unit lower triangular submatrix of $C_{\overline{L}}$, which implies that the minimal NFA for $\overline{L}$ must have at least $n-1$ states.) □

9.4 Lower Bounds Based on Set Bases

The rank-based lower bounds developed in Section 9.3 are not always tight. For example, consider $\{a^i \mid i \bmod 4 = 0 \text{ or } i \bmod 4 = 3\}$, whose characteristic matrix is shown in Figure 9.6. It turns out that the rank of this matrix is 3. (Note that row 1 = row 2 + row 4 − row 3.) But an exhaustive search shows that the minimal UFA for this language requires 4 states. We can explain this discrepancy by observing that a UFA M expresses each row of the characteristic matrix as a linear combination of the rows of Δ_M^R using only *nonnegative* coefficients. In contrast, matrix rank allows negative coefficients (as in the example above). We can improve our lower bounds by considering *set bases* instead.

Definition 9.4.1 *Given a collection $\mathcal{C}$ of target sets, a* set basis for $\mathcal{C}$ *is a collection of sets such that each set in $\mathcal{C}$ can be expressed as a union of sets from $\mathcal{B}$. If each*

set in $\mathcal{C}$ can be expressed as a union of pairwise disjoint *sets from $\mathcal{B}$, then $\mathcal{B}$ is said to be a* normal set basis *for $\mathcal{C}$.*

We explain the relevance of set bases by observing that if a language L is recognized by a UFA M, then the rows of C_L are linear combinations of the rows of Δ_M^R, and *each coefficient is either 0 or 1.* Hence, viewing the rows of C_L and Δ_M^R as property vectors, we see that the rows of Δ_M^R are a *normal set basis* for the rows of C_L. Similarly, if L is recognized by an NFA M, then the rows of Δ_M^R are a *set basis* for the rows of C_L. Thus we have the following theorems:

Theorem 9.4.2 *For any L, the number of states in any UFA recognizing L is at least the size of the smallest normal set basis for the rows of C_L.*

Theorem 9.4.3 *For any L, the number of states in any NFA recognizing L is at least the size of the smallest set basis for the rows of C_L.*

A similar bound is shown by Condon et al. [CHPW98]. They define a *1-tile* of C_L to be a submatrix (specified by a nonempty set of rows and a nonempty set of columns) all of whose entries are 1. Next, they say that a set of 1-tiles is a *1-tiling* of C_L if every 1 entry of C_L is covered by at least one tile in the set. Finally, they let $T^1(C_L)$ denote the minimum size of a 1-tiling of C_L, and $T_L^1(n)$ denote the minimum size of a 1-tiling of the finite submatrix of C_L whose rows and columns are indexed by the strings of length at most n. They then show [CHPW98, p. 750] that $T_L^1(n)$ is a lower bound on the number of states in any NFA N that recognizes L "up to length $2n$", which means that N recognizes a language that agrees with L on strings of length at most $2n$. This result is, in fact, closely related to ours, because 1-tilings are really the same thing as set bases! Indeed, it is easy to see that C_L has a 1-tiling of size k iff it has a set basis of size k.

	ϵ	a	a^2	a^3	a^4	a^5	a^6
ϵ	1	0	0	1	1	0	1
a	0	0	1	1	0	1	1
a^2	0	1	1	0	1	1	1
a^3	1	1	0	1	1	1	1
a^4	1	0	1	1	1	1	1
a^5	0	1	1	1	1	1	1
a^6	1	1	1	1	1	1	1

Figure 9.7: Characteristic matrix for $(a^3 \cup a^4)^*$

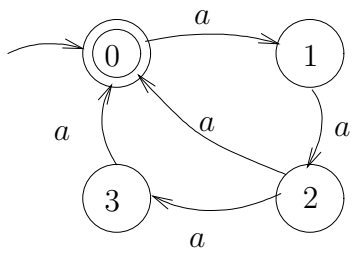

Figure 9.8: A Minimal NFA for $(a^3 \cup a^4)^*$

Unfortunately, determining the size of the smallest set basis or normal set basis is known to be NP-complete [JR93]. But in some cases these sizes can be determined by exhaustive search. For example, it can be checked that the characteristic matrix of $\{a^i \mid i \bmod 4 = 0 \text{ or } i \bmod 4 = 3\}$ (see Figure 9.6) requires a set basis of size 4. Hence the obvious 4-state DFA is a minimal NFA for this language.

As another example, consider the language generated by the regular expression $(a^3 \cup a^4)^*$. A portion of its characteristic matrix is shown in Figure 9.7; the remaining entries are all 1. The characteristic matrix has 7 distinct nonzero rows and its rank is 7. Hence the minimal DFA has 7 states, and it is also a minimal UFA. An exhaustive search reveals that the smallest set basis has size 4. In fact there is a 4-state NFA for this language; one is shown in Figure 9.8.

Unfortunately, the lower bounds based on set bases are not tight. For example, the characteristic matrix in Figure 9.1 turns out to have set bases of size 4, as can be seen from the following:

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 2 \\ 2 & 1 & 1 & 1 & 1 & 0 \\ 1 & 2 & 1 & 1 & 0 & 1 \\ 1 & 1 & 2 & 0 & 1 & 1 \\ 1 & 1 & 0 & 2 & 1 & 1 \\ 1 & 0 & 1 & 1 & 2 & 1 \end{bmatrix}$$

But an exhaustive search shows that this language actually requires a 5-state NFA. The problem is that some factorizations of the characteristic matrix that do not correspond to any NFA. Of course, the theory of inductive bases explains this—note that the second matrix in the above factorization is not an *inductive basis*, since it cannot generate the left quotient of its first row.

9.5 Factoring the Subset DFA of an NFA

Given an n-state NFA N, the well-known subset construction [HU79] gives an equivalent DFA M with at most 2^n states. Here we observe that the histories and prophecies of the states of M can be expressed in terms of the histories and prophecies of the states of N. Each state A of the subset DFA M is a set of states of the NFA N. We have

$$Hist_M(A) = \bigcap_{q \in A} Hist_N(q) - \bigcup_{q \notin A} Hist_N(q)$$

and

$$Proph_M(A) = \bigcup_{q \in A} Proph_N(q).$$

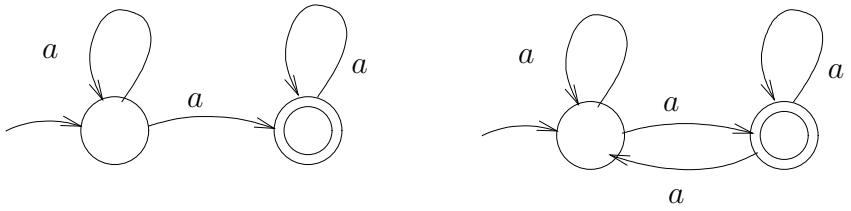

Figure 9.9: Two strongly equivalent NFAs

(It is interesting to note the asymmetry between these two equations.)

Recall that the columns of Δ represent histories while the rows of Δ^R represent prophecies (in reverse). As a consequence, Δ_M and Δ_M^R can be computed directly from Δ_N and Δ_N^R by using these equations.

9.6 A Notion of Strong Equivalence on NFAs

Let us say that two NFAs are *strongly equivalent* if they give the same factorization of the characteristic matrix. Of course this implies that they recognize the same language and have the same set of states, but some of the arrows may be different. For example, the two NFAs shown in Figure 9.9 are strongly equivalent.

9.7 Bideterminism

Before discussing bidetermism, we first recall the notion of a *trim* NFA, as defined in [ADN92]:

Definition 9.7.1 *NFA M is* trim *if for every state q, $Hist(q)$ and $Proph(q)$ are both nonempty.*

A state whose history is empty can never be reached, and a state whose prophecy is empty is a "dead" state; such states may as well be deleted.

Definition 9.7.2 *An NFA M is said to be a* bideterministic DFA *(BDFA) if both M and M^R are DFAs. A language L is said to be* bideterministic *if it is recognized by some BDFA.*

In [TU03], it is shown that any trim BDFA M is a minimal NFA recognizing $L(M)$. Here we explore bideterministic languages in terms of their characteristic matrices.

Theorem 9.7.3 *A language L is bideterministic iff the distinct rows of C_L are pairwise disjoint.*

Proof. If L is recognized by a BDFA M then $C_L = \Delta_M \times \Delta_M^R$ and, since M is a DFA, each nonzero row of C_L appears as one of the rows of Δ_M^R. Also, since M^R is a DFA, each column of Δ_M^R contains at most one 1. Now, we want to prove that if two rows of C_L have a 1 in the same column, then they are equal. So suppose that $C_L[v, x] = 1$ and $C_L[w, x] = 1$. Then $C_L[v, -]$ and $C_L[w, -]$ are nonzero rows, which implies by the above discussion that there exist states p and q such that $C_L[v, -] = \Delta_M^R[p, -]$ and $C_L[w, -] = \Delta_M^R[q, -]$. Hence $\Delta_M^R[p, x] = \Delta_M^R[q, x] = 1$. But since each column of Δ_M^R contains at most one 1, it follows that $p = q$. Hence $C_L[v, -] = C_L[w, -]$.

Conversely, suppose that the distinct rows of C_L are pairwise disjoint. Recall from the proof of the Myhill-Nerode Theorem that if M is the minimal DFA recognizing L, then the rows of Δ_M^R are the distinct nonzero rows of C_L. Since these are pairwise disjoint, it follows that each column of Δ_M^R contains at most one 1. Hence M^R is a DFA, which implies that L is bideterministic. □

Corollary 9.7.4 *If M is a trim BDFA, then M is a minimal NFA recognizing $L(M)$.*

Proof. Suppose that M is a trim BDFA recognizing a language L. Because $C_L = \Delta_M \times \Delta_M^R$ and M is a BDFA, each nonzero row of C_L appears exactly once as a row in Δ_M^R. Furthermore, because M is trim, each row of Δ_M^R is nonzero. Also, each column of Δ_M is nonzero, which implies that each row of Δ_M^R appears at least once as a row in C_L. We conclude that that the rows of Δ_M^R are exactly the distinct nonzero rows of C_L. It follows that M is the minimal DFA recognizing L.

Also, by the above theorem, the distinct rows of C_L are pairwise disjoint. Hence if C_L contains r distinct nonzero rows, then its structural rank is clearly r. So any NFA recognizing L must have at least r states. Thus M is also a minimal NFA recognizing L. □

9.8 Finite Languages and One-Character Alphabets

In this section, we explore the usefulness of nondeterminism in recognizing some restricted kinds of languages.

First we show that nondeterminism can be useful in recognizing finite languages. Consider the set of non-palindromes of length $2n$, for some $n \geq 1$:

$$L_n = \{w \in \{a, b\}^* \mid w \neq w^R \text{ and } |w| = 2n\}$$

It is easy to see that a DFA recognizing L_n requires at least 2^n states, since the first n symbols of the input string need to be remembered. But an NFA can recognize L_n using only about $4n^2$ states; for each k, $1 \leq k \leq n$, two chains of $2n + 1$ states each can verify that positions k and $2n - k + 1$ of the input string differ; and hence n pairs of chains can verify that the input is not a palindrome. Schmidt [Sch78] has similar observations.

Next consider languages over a one-character alphabet. Generalizing the NFA of Figure 2.1, let p and q be relatively prime and consider

$$\{a^i \mid i \bmod pq \neq 0\}.$$

The characteristic matrix has pq distinct nonzero rows, so a DFA requires pq states. But an NFA can recognize this language using using a ring of size p and a ring of size q, for a total of $p + q$ states.

Of course this NFA is not a UFA, since it has two accepting paths for any string whose length is a multiple of neither p nor q. So we may wonder whether unambiguous nondeterminism is useful over a one-character alphabet. It turns out that it can be. Let p and q be *not* relatively prime and consider

$$\{a^i \mid i \bmod p = 0 \text{ or } i \bmod q = 1\}.$$

The number of states in the minimal DFA is the least common multiple of p and q. As above, an NFA can be built using a ring of p states and a ring of q states. And this NFA turns out to be a UFA. For if d is the greatest common divisor of p and q, then the first ring accepts only strings whose length is a multiple of d, while the second ring accepts only strings whose length is not a multiple of d.

Finally consider finite languages over a one-character alphabet. Here we can argue that nondeterminism is not useful. For if $L \subseteq \{a\}^*$ and a^n is the longest string in L, then C_L has $n+1$ distinct nonzero rows, and the $(n+1) \times (n+1)$ submatrix in the upper left-hand corner is (if its rows are reversed) unit lower triangular. Hence it has structural rank $n + 1$, and the minimal DFA (which has $n + 1$ states) is also a minimal NFA.

To conclude this chapter, we have seen that the structure of the computation matrix of a language allows us to establish interesting lower bounds on the size of any NFA or UFA recognizing the language. In some cases, these lower bounds are

precise enough to show the minimality of an NFA or UFA; in other cases, the lower bounds are not tight.

CHAPTER 10

CONCLUSION

The theoretical nature of this work and the originality of its methods should provide a wide range of future developments. Even though we have been working for many years, most of the good results are very recent and it is not easy to foresee practical applications at such an early stage. It seems to us that the theory is very complete and elegant; therefore it does not seem to need theoretical extensions. We think that the use of the inductive basis representation as a tool to solve specific instances is probably a natural continuation to our study.

We now discuss a few specific future directions:

Enumeration of Non-isomorphic NFAs

Enumerating non-isomorphic NFAs or minimal NFAs is a problem of some difficulty. The brute-force approach is feasible only for a very small number of states; this is due to the size of the search space and the fact that isomorphic NFAs are costly to detect in the traditional directed graph representation.

Our inductive basis characterization is better ordered and avoids the problem of generating the "same" NFA in multiple forms. It seems very possible that an inductive basis algorithm, similar to `ibas` but aimed at enumerating non-isomorphic NFAs, could enumerate non-isomorphic NFAs whose size is several states larger than what can be handled with pure combinatorial graph-based algorithms.

Pattern Recognition

The way `ibas` works—by transforming a nonprocedural representation into a procedural representation in a incremental, exhaustive, yet efficient way—makes us think that it could be used to recognize patterns whose nature and structure is unknown.

An application of greater size (ours only works for very small sequences) involving longer strings and maybe clusters of NFAs might be able to decipher complex patterns, when fed with enough examples.

Complexity

The problem of good lower or upper bounds has been a major concern of our study from the beginning. Upper bounds for worst-case complexity are theoretically important, and also in order to assess time and space resources for open-ended searching algorithms like the ones we have presented. Questions that require future study include:

- How can we know that some of the NFAs obtained recognize the language that we initially wanted?
- How long should we wait until giving up in an unsuccessful search?

Algorithms

We have implemented and tested many variations to `ibas` with encouraging results; the optimized implementations and new theoretical insights have resulted in significant improvements in speed.

We have wondered whether looking at only one side of the factorization (namely the prophecy) was not a symmetrical approach, and might lead to extra work in some particular cases. But the idea of working simultaneously with histories and prophecies was not implemented due to the complexity of the interaction among the sets of elements used to build the quotient languages in the process of extending partial inductive bases.

A *double ibas* algorithm might be developed and implemented to work simultaneously with two inductive basis (the histories and the prophecies) that should be

related by the use of an auxiliary matrix in order to control the interaction between the two bases. Such an algorithm could have some advantages, but it is clear that it presents many implementation challenges.

Combinatorial Models

One of the most promising directions that we foresee is associated with the use of non-linear optimization algorithms, like gradient-based feasible direction algorithms. The algebraic nature of the inductive basis representation seems ideal for producing a box-based restriction $0, 1$ model that could be solved using a relaxation to the real numbers.

Starting from a feasible solution given by a non-minimal NFA represented in a matrix form, the model could force one of the rows to take all zero values, doing a de facto elimination of the row by means of a cost function.

It seems that the proximity of the minimal solution to the non-minimal initial solution could avoid local minima and obtain a global minimum when a return to a $0, 1$ matrix is forced by a penalty, most probably a quadratic function that will increase value in each iteration.

The model will consist of

- a fixed target vector, the partial of the language;
- an inductive basis that will be relaxed, allowing it to take values on the (0,1) interval;
- a set of restrictions of equality with a penalty that will prevent the basis from distancing too far from the inductive property; and
- a set of inequality restrictions, that will be implemented by the use of a test of ratio where all negative components of the direction will be limited to a step

that does not make them negative and all positive components of the direction will be limited to a value less than or equal to 1.

The "box" type inequality restrictions are relatively easy to implement and the model could be solved using any of the many non-linear optimization techniques.

We think that such a development could actually for the first time succeed in minimizing NFAs in many practical contexts, remembering of course that the theoretical problem is NP-hard.

BIBLIOGRAPHY

[ADN92] André Arnold, Anne Dicky, and Maurice Nivat. A note about minimal non-deterministic automata. *Bulletin of the EATCS*, 47:166–169, 1992.

[Blu94] Norbert Blum. An $O(n \log n)$ implementation of the standard method for minimizing n-state finite automata. *Information Processing Letters*, April 1994.

[CCS06] Andrei Paun Cezar Campeanu and Jason R. Smith. Incremental construction of minimal deterministic finite cover automata. *Theoretical Computer Science*, 363(2):135–148, October 2006.

[CCY01] N Santean Cezar Campeanu and S Yu. Minimal cover automata for finite languages. *Theoretical Computer Science*, 267(1-2):3–16, September 2001.

[CHPW98] Anne Condon, Lisa Hellerstein, Samuel Pottle, and Avi Wigderson. On the power of finite automata with both nondeterministic and probabilistic states. *SIAM Journal on Computing*, 27(3):739–762, June 1998.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. MIT Press, 2001.

[CM04] Wei Cheng and Zhi-Wen Mo. Minimization algorithm of fuzzy finite automata. *Fuzzy sets and Systems*, 141, February 2004. recieved in 10 2000.

[Con71] John H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall Ltd, 1971.

[Fel68] William Feller. *An Introduction to Probability Theory and Its Applications*, volume I. John Wiley & Sons, Inc., Third edition, 1968.

[GLW92] Jonathan Goldstine, Hing Leung, and Detlef Wotschke. On the relation between ambiguity and nondeterminism in finite automata. *Information and Computation*, 100(2):261–270, October 1992.

[GM99] Dora Giammarresi and Rosa Moltanbano. Deterministic generalized automata. *Theoretical Computer Science*, 215(1-2):191–208, February 1999.

[Gra03] Gregor Gramlich. Probabilistic and nondeterministic unary automata. In *Proceedings of Mathematical Foundations of Computer Science*, volume 2747 of *Lecture Notes in Computer Science*, pages 460–469. Springer-Verlag, August 2003.

[GS96] Ian Glaister and Jeffrey Shallit. A lower bound technique for the size of nondeterministic finite automata. *Information Processing Letters*, 59:75–77, 1996.

[GS05] Gregor Gramlich and Georg Schnitger. Minimizing NFA's and regular expressions. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science*, volume 3404 of *Lecture Notes in Computer Science*, pages 399–411. Springer-Verlag, February 2005.

[HK05] Markus Holzer and Martin Kutrib. On the descriptional complexity of finite automata with modified acceptance conditions. *Theoretical Computer Science*, 330(2):267–285, February 2005.

[Hop71] John E. Hopcroft. An $n \log n$ algorithm for minimizing the states in a finite automaton. In Z. Kohavi, editor, *The Theory of Machines and Computations*, pages 189–196. Academic Press, New York, 1971.

[HS01] Juraj Hromkovič and Georg Schnitger. On the power of Las Vegas for one-way communication complexity, OBDDs, and finite automata. *Information and Computation*, 169:284–296, 2001.

[HSK+02] Juraj Hromkovič, Sebastian Seibert, Juhani Karhumäki, Hartmut Klauck, and Georg Schnitger. Communication complexity method for measuring nondeterminism in finite automata. *Information and Computation*, 172:202–217, 2002.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[HW04] Yo-Sub Han and Derik Wood. The generalization of generalized automata: Expression automata. In *Implementation and Applications in Automata: 9th International Conference*, pages 156–167, July 2004.

[IY02] Lucian Ilie and Sheng Yu. Algorithms for computing small NFAs. In *MFCS '02: Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science*, pages 328–340, 2002.

[JJZ04] Galina Jirascova Joseph Jirasek and Alexander Zabari. State complexity of concatenation and complementation of regular languages. In *Implementation and Application of Automata: 9th Conference*, pages 145–156, July 2004.

[JR93] Tao Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM Journal on Computing*, 22(6):1117–1141, December 1993.

[Knu01] Timo Knuutla. Re-describing an algorithm by Hopcroft. *Theoretical Computer Science*, 250(1-2):333–363, January 2001.

[Koz97] Dexter C. Kozen. *Automata and Computability*. Springer-Verlag, 1997.

[KW06] T. Kameda and P. Weiner. On the state minimization of nondeterministic finite automata. *Transaction on Computers*, 19(7):617–627, October 2006.

[LL07] Hongxuan Lei and Yongmin Li. Minimization of states in automata theory based on finite lattice-ordered monoids. *Information Sciencies*, 177(6):1413–1421, March 2007.

[LP07] Yongming Li and Witold Pedrycz. Minimization of lattice finite automata and its application to the decomposition of lattice languages. *Fuzzy Sets and Systems*, 158(13):1423–1436, July 2007.

[Mal04] Andreias Malcher. Minimizing finite automata is computationally hard. *Theoretical Computer Science*, 327(3):375–390, November 2004.

[Mat] Oliver Matz. Canonical nondeterministic finite automata and equivalent subautomata.

[MMS99] D. S. Malik, John N. Mordeson, and M. K. Sen. Minimization of fuzzy finite automata. *Information Science*, 113(3):323–330, February 1999.

[MP95] Oliver Matz and Andreas Potthoff. Computing small nondeterministic finita automata. In *Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, pages 74–88, 1995.

[Ner58] Anil Nerode. Linear automaton transformations. *Proc. AMS*, 9:541–544, 1958.

[Pee88] Ketty G Peeva. Behavoir, reduction and minimization of finite automata. *Fuzzy Sets and Systems*, 28(2):171–181, November 1988.

[Per90] Dominique Perrin. Finite automata. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science. Volume B: Formal Models and Semantics*, chapter 1, pages 1–57. Elsevier Science Publishers, 1990.

[Pet06] Tatjana Petkovic. Congruences and homomorphisms of fuzzy automata. *Fuzzy sets and Systems*, 157(3):444–458, February 2006. posted in the ScienceDirect site in 06 06.

[Rev92] Dominique Revus. Minimization of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92(1):181–189, January 1992.

[RS59] Michael Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

[Sch78] Erik Meineche Schmidt. *Succinctness of Descriptions of Context-Free, Regular and Finite Languages*. PhD thesis, Cornell University, January 1978.

[SM73] Larry Stockmeyer and Albert Meyer. Word problems requiring exponential time. In *Proceedings 5th ACM Symposium on Theory of Computing*, pages 1–9, 1973.

[Smi06] Geoffrey Smith. Inductive bases and their application to searches for minimal unary NFAs. In *Proceedings ACMSE 2006: 44th ACM Southeast Conference*, pages 470–475, Melbourne, FL, March 2006.

[Sta83] Ludwig Staiger. Finite states ω languages. *Journal of Computer and System Science*, 27(3):434–448, December 1983.

[T94] Van Le T. Fuzzy finite automata and their application to speech recognition. In *AI: Simulation, and Planning in High Autonomy Systems: 5th annual conference*, pages 269–272, December 1994.

[TU03] Hellis Tamm and Esko Ukkonen. Bideterministic automata and minimal representations of regular languages. In *Proceedings Eighth International Conference on Implementation and Application of Automata*, vol-

ume 2759 of *Lecture Notes in Computer Science*, pages 61–71. Springer-Verlag, July 2003.

[vZ04] Linette van Zijl. Magic numbers and symetric difference. In *Implementation and Application of Automata: 9th Conference*, pages 333–335, July 2004.

[Wat03] Bruce W. Watson. A new algorithm for the construction of minimal acyclic NFA. *Science of Computer Programming*, 48(2-3):81–97, September 2003.

VITA

DANIEL CAZALIS

October 4, 1948	Born, Havana, Cuba
1978	B.A., Architecture Central University of Venezuela Also completed required courses for Mathematics
1991	M.S., Computer Science Pennsylvania State University
1994	Ph.D., Applied Mathematics COPPE Universidade Federal do Rio de Janeiro
1994–2007	Full-time or part-time professor: Universidad Metropolitana, Caracas, Venezuela Universidad Simon Bolivar, Caracas, Venezuela Barry University, Miami, Florida Florida International University, Miami, Florida