Nondeterministic State Complexity of Site-Directed Operations

by

Oliver A.S. Lyon

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
April 2021

Abstract

In this thesis, we consider the nondeterministic state complexity of PCR-inspired (polymerase chain reaction) operations. Site-directed operations are used to formally describe the behavior of certain DNA (deoxyribonucleic acid) editing methods which need to identify a subsequence in a host DNA strand prior to editing. These operations can be considered as language operations acting to match patterns between two sets of strings.

The site-directed insertion and deletion operations, insert or delete into a host string based on a directing string. The directing string must have a non-empty outfix that matches a substring in the host before operating. Prefix and suffix directed insertion are similar to site-directed insertion except, instead of matching a non-empty outfix, a non-empty prefix or suffix is matched before insertion. We consider the nondeterministic state complexity of site-directed insertion and deletion. Constructing a nondeterministic finite automaton (NFA) for the operation provides an upper bound for the state complexity of the operation. Our construction improves the earlier upper bound in the literature. Existing literature did not give lower bounds for the nondeterministic state complexity of site-directed insertion and deletion. Using the fooling set method we establish lower bounds that are fairly close to the upper bound, albeit the lower bound is not tight.

The other operations considered are those that identify subsets of a language. The prefix, suffix, infix and outfix operations check to see if a word from another language is contained as a prefix, suffix, infix or outfix of the target word. We introduce NFA constructions which apply these operations over regular languages, and establish bounds for the prefix, suffix, infix and outfix operations.

Acknowledgments

I would like to thank my supervisor Dr. Kai Salomaa for the guidance and feedback he has provided along the way. Special thanks to Chris Keeler and Taylor Smith, for their patience with my eagerness to share crackpot theories and for the many successful labbies. I also want to thank my parents Dr. Lyon^2, for their sympathetic ears and unwavering support. Lastly, I want to acknowledge Kody Manastyrski and Jessica Berg, for helping bring some much needed laughter throughout the process.

These past two years have been additionally challenging for us all during the COVID-19 pandemic, no progress is individual, all support is appreciated.

Contents

List of Tables

List of Figures

Glossary

DFA Deterministic Finite Automata.

DNA Deoxyribonucleic Acid.

NFA Nondeterministic Finite Automata.

NSC Nondeterministic State Complexity.

PCR Polymerase Chain Reaction.

Chapter 1

Introduction

Reading and writing are fundamental operations of modern computers. Many computation models and algorithms rely on the ability to efficiently read and write to files or strings. Deoxyribonucleic acid (DNA) can be considered a string, with its sequence of nucleotide base pairs as its ordered symbols. The development of DNA-computing has been linked with emerging methods for reading and writing to a DNA strand. DNA manipulation has developed significantly in the last 30 years with technologies like PCR (polymerase chain reaction) and Crispr (Clustered regularly interspaced short palindromic repeats) becoming widely available. It is important to understand the costs and limitations of applying new bio-technologies to algorithms. The development of a formal model for a method allows us to quantify the limitations and costs for a given operation. Each model has an associated descriptional complexity or state complexity when operating over different languages. This thesis explores the state complexity of formal models of representing PCR and related operations.

1.1 DNA-Computing

DNA-computing is the field of study that uses DNA as a computational medium for algorithms. This field was cultivated, in large part by the work of Leonard Adleman, with his famous 1994 experiment [1]. Adleman not only showed that DNA could be used to solve an instance of the Hamiltonian Path Problem, but he also developed notation to record his algorithm. The field of natural computing has expanded on this work by formalizing many of the methods applied to manipulating DNA. Collections of these models and results can be found in surveys and textbooks such as [10, 30, 32, 36]. Of particular interest to this thesis is the PCR method for inducing site-directed mutagenesis. PCR methods can use mutagenic primers to insert or remove sections of DNA from a host strand, as visualized in Figure 1.1 [7, 34]. Mutagenic primers identify subsequences in the DNA strand that will be edited.

Contextual insertion and deletion operations have been used to model PCR. These operations were also used to construct a Turing machine and prove related closure properties by Kari and Thierrin [33]. The model of computation produced by Kari and Thierrin was expanded upon by Takahara and Yokomori [39] to generate classes of languages. Domaratzki explored trajectory based languages which use additional set of symbols to assess which section of the string is modified [11, 12]. Algorithmic applications of PCR operations have been explored by Franco and Manca [15].

Cho et al. [7, 9] defined site-directed insertion and deletion which are PCR-inspired operations and also studied their language theoretic properties. These operations share similarities to contextual insertion and deletion, but have key definitional differences. In this thesis, we study the descriptional complexity of site-directed insertion and deletion as well as other related operations.

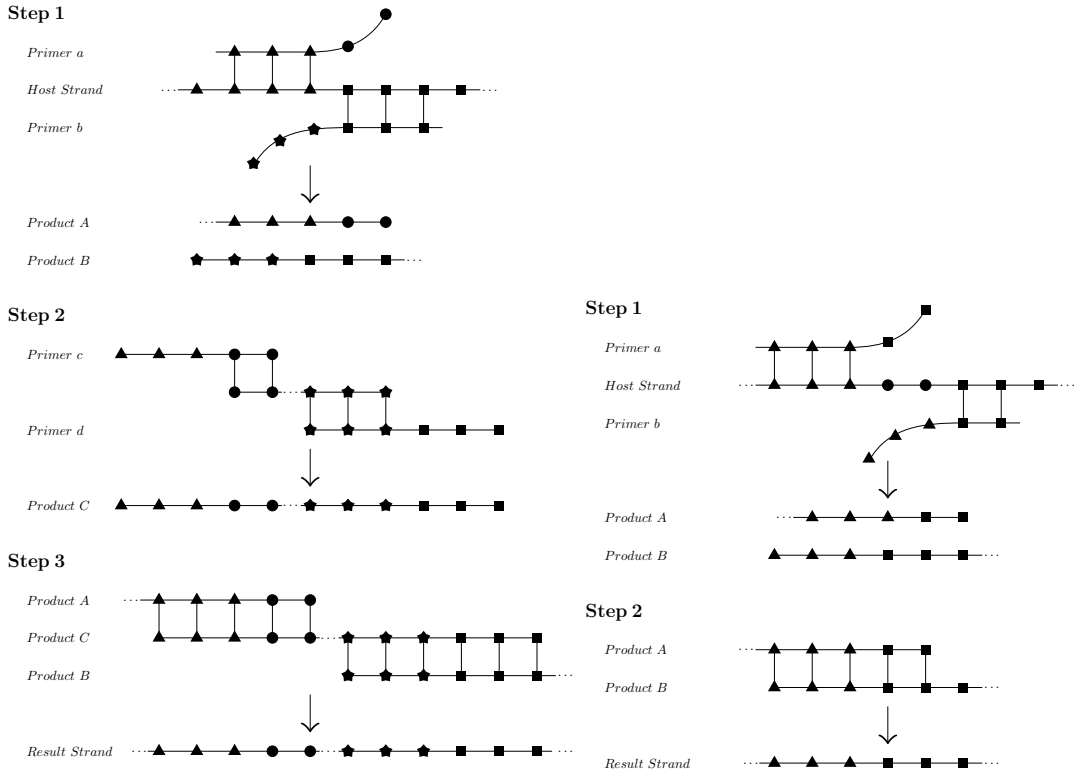

Figure 1.1: The left side of this figure depicts site-directed insertion mutagenesis: Step 1 is DNA cutting using primers a and b, Step 2 is annealing inserted sequence using primers c and d, Step 3 is ligation PCR using products A, B and C. The right side of this figure depicts site-directed deletion mutagenesis: Step 1 is DNA cutting using primers a and b, Step 2 is ligation PCR using products A and B.

1.2 State Complexity

The field of computational complexity has developed measures to quantify the amount of time and memory required by an algorithm. In contrast the field of descriptional complexity measures the size of an algorithm. When using finite automata as our model of computation, the most common descriptional complexity measure is state complexity. State complexity uses the number of states of an automaton as its size.

Automata can take many different forms and can recognize different classes of languages. In this thesis we only consider the class of regular languages. Generally, it is well known that a minimal deterministic finite automaton (DFA) defining a regular language is unique [28]. On the other hand, a language L may have more than one nondeterministic finite automaton (NFA) with a minimal number of states [24, 27]. The lack of uniqueness in an NFA's construction leads to ambiguity regarding its minimality.

Many language operations are defined in terms of a string operation that is applied to elements of the languages. An example of a language operation is string concatenation where words from L_1 are concatenated to words from L_2. To determine the *nondeterministic state complexity* of a language operation ω, we must first show that the operation preserves regularity (i.e. if L_1 and L_2 are regular languages then the language $\omega(L_1, L_2)$ is also regular). We can describe the nondeterministic state complexity of ω as a function $f_\omega : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. If we assume L_1 and L_2 are recognized by NFAs containing N and M states respectively, then $f_\omega(N, M)$ is the worst-case size of an NFA recognizing the language $\omega(L_1, L_2)$. Identifying the function $f_\omega(N, M)$ can be a difficult undertaking. Previously there has been work to understand the state complexities for various operations [5, 16, 24, 25, 26, 27, 29].

Knowing the state complexity of an operation provides a concise representation. In the words of Sheng Yu, *"State complexity problems are a fundamental part of automata theory. The study of state complexity and related problems has a long history."* [41]. State complexity can also provide insight into the properties of an operation as well as other related operations.

A construction showing that an operation ω preserves regularity of languages

also acts as an upper bound for the (nondeterministic) state complexity of ω. To know if a construction is optimal, we need to prove a consistent lower bound for the state complexity. To show that $f_\omega(N, M)$ is a lower bound for the state complexity of the operation ω, we, generally speaking, need to show that there exist NFAs A_1 and A_2 that have N and M states, respectively, such that any NFA recognizing $\omega(L(A_1), L(A_2))$ needs at least $f_\omega(N, M)$ states. In the early 90's, lower bound methods called fooling set methods were developed [2, 3, 17]. These methods are considered to be the canonical method for proving a lower bound.

There are few relevant publications establishing lower bounds on nondeterministic state complexity for PCR-related operations. The notable exception is the bound established on alphabetic site-directed insertion [8]. Alphabetic site-directed insertion is a variant of the site-directed insertion operation where only a single symbol is overlapped as the context for the insertion. Since there have been few results, it is useful to examine other operations which have similar properties. There have been several papers describing the state complexity of the shuffle operation, which acts to zipper merge two words [4, 6]. Lastly, Han et al. [19] proved tight bounds on the state complexity of the ordinary insertion operation, which places one word inside another.

Site-directed insertion is a variant of contextual insertion where a non-empty prefix and suffix from the inserted string must match a component of the target string. An upper bound on nondeterministic state complexity for this operation was provided when it was proven to preserve regularity [8, 7]. The upper bound was achieved through providing an NFA construction with a nondeterministic state complexity of $3NM+2N$. In chapter 4, we will show an improved upper bound of $3NM$, with a near

lower bound of $3NM - M$ for the nondeterministic state complexity of site-directed insertion.

Site-directed deletion is similar to site-directed insertion, but instead of inserting, it removes a section of the string. The operation had an established upper bound on nondeterministic state complexity of $2NM + 2N$, as shown by Cho et al. [9]. We were able to improve the upper bound to $2NM + N$ along with a near lower bound of $2NM$.

1.3 Organization of Thesis

This thesis contains different chapters exploring PCR inspired languages. The following is a summary of their contents:

2 Preliminaries This chapter reviews important definitions as well as nondeterministic finite automata models and important theorems for determining state complexity bounds.

3 Prefix, Suffix, Infix and Outfix This chapter presents tight bounds on nondeterministic and deterministic state complexity for the prefix operation. We also present nondeterministic state complexity bounds for suffix, infix, and outfix operations.

4 Site-Directed Insertion Operation and Variants In this chapter we present fairly closer upper bounds and lower bounds on nondeterministic state complexity for prefix-directed insertion, suffix-directed insertion and site-directed insertion.

5 Unary Site-Directed Insertion In this chapter we consider the nondeterministic state complexity of site-directed insertion restricted to languages over a one-letter alphabet.

6 Site-Directed Deletion In this chapter we show bounds on the nondeterministic state complexity for the site-directed deletion operation. Unary site-directed deletion is also considered.

7 Conclusion This chapter reviews and summarizes the results of the thesis and provides suggestions for future work.

Chapter 2

Preliminaries

In this section, we will review the definition of nondeterministic finite automata (NFA) and how computations are intrinsically linked with their languages. We will also review several useful results for discussing state complexity of nondeterministic machines. Finally, the chapter will conclude with a review of unary NFAs.

2.1 Formal Languages

An alphabet Σ is a collection of letters used to construct words w. Words are sequences of letters and the length of a word is denoted $|w|$, and, is the number of letters contained. The empty word ε is the word containing no letters which means it has a length of zero. We recursively define Σ^i to be $\{xy \mid x \in \Sigma^{i-1} \text{ and } y \in \Sigma\}$, where $1 < i$ and $\Sigma^0 = \{\varepsilon\}$. We define Σ^* to be $\bigcup_{0 \leq i}^{\infty} \Sigma^i$, and Σ^+ to be $\bigcup_{1 \leq i}^{\infty} \Sigma^i$

A set of words is called a language L. A language is called finite if the number of words contained in the language is a whole number. Cardinality of a set is the number elements contained in the set, we denote cardinality as $|L|$. We define an infinite language as having infinite cardinality.

Example 1. *For a finite language* $L = \{\epsilon, a, aa, aaa, b, bb, bbb\}$ *with alphabet* $\Sigma = \{a, b\}$, $|L| = 7, |aa| = 2$.

Many of the operations studied in this thesis are dependent on different affixes, those affixes are defined as follows:

Definition 1 (Non-empty Prefix)**.** *A word* $w \in \Sigma^*$ *has a non-empty prefix of* x *if we decompose the word* w *into* xz *provided* $x \neq \varepsilon$.

Definition 2 (Non-empty Suffix)**.** *A word* $w \in \Sigma^*$ *has a non-empty suffix of* z *if we decompose the word* w *into* xz *provided* $z \neq \varepsilon$.

Definition 3 (Non-empty Infix)**.** *A word* $w \in \Sigma^*$ *has a non-empty infix of* y *if we decompose the word* w *into* xyz *provided* $y \neq \varepsilon$.

Definition 4 (Non-empty Outfix)**.** *A word* $w \in \Sigma^*$ *has a non-empty outfix of* x, z *if we decompose the word* w *into* xyz *provided* $x, z \neq \varepsilon$.

Definition 5 (Non-empty Bifix)**.** *A word* $w \in \Sigma^*$ *has a non-empty bifix of* x, z *if we identify a non-empty prefix* x *and a non-empty suffix* z *for the word* w.

2.2 Automata and Regular Languages

The main computational model we will use is the *finite automaton* that reads input one character at a time and changes state according to a transition function.

Definition 6. *(Nondeterministic Finite Automata). An NFA* $A = (Q, \Sigma, \delta, s, F)$, *is defined as a tuple of the following:*

- Q *is the finite set of states of* A.

- Σ *is the finite alphabet of characters.*
- δ *is the partial transition function defined over* $Q \times \Sigma \to 2^Q$.
- s *is the start state,* $s \in Q$.
- F *is the set of final states,* $F \subseteq Q$.

The nondeterministic aspect of the aforementioned automata is that the transition function allows for a set of states to be returned as a result of a given state/character pairing. This implies that a computation is capable of being in many states at once. For a computation to be accepted, only one of the current states must be a final state at the end of a given word.

For every string, there is an ordered set of states that are visited during the computation, which is called a path. A path can be used to prove the correctness of an automaton recognizing a given language.

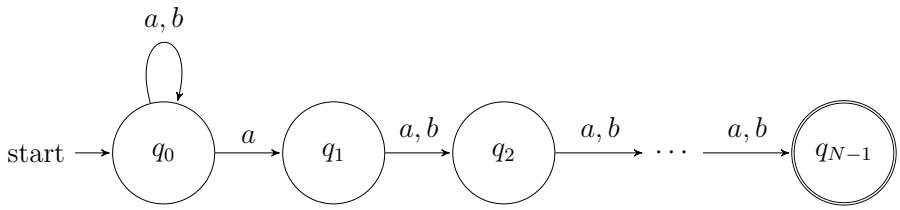

Figure 2.1: Example of a nondeterministic finite automaton.

Example 2. *The automaton in Figure 2.1 recognizes the binary language where the* $(N-1)$*th last character is an* a*. The states returned from* $\delta(q_0, a) = \{q_0, q_1\}$.

An extension of an NFA, called a ε-NFA, allows transitions labeled by the empty word. This allows for transitions between states without reading from the input string. NFAs containing ε transitions are called ε-NFAs.

Proposition 1. *For any ε-NFA with N states, an equivalent N-state NFA can be constructed without ε transitions [19, 40].*

Definition 7 (deterministic finite automaton)**.** *A deterministic finite automaton (DFA) is defined as an NFA $A = (Q, \Sigma, \delta, s, F)$, where*

$$for\ all\ q \in Q\ and\ for\ all\ a \in \Sigma, |\delta(q, a)| \leq 1$$

DFAs and NFAs recognize exactly the class of regular languages. There exists an algorithm to convert NFAs to DFAs. This algorithm has the potential to require an exponential number of states to represent the same language [28]. Non-regular languages are beyond the scope of this thesis.

2.3 Lower Bounds on Nondeterministic State Complexity

NFAs can be constructed to recognize a given language, but, it is a challenge to show that all the states are essential. The state complexity for a given automaton is the minimal number of states required to recognize its language. The state complexity to recognize a language L using an NFA is defined by the function $nsc(L)$, which is the *nondeterministic state complexity* of the language L. To prove that a given language L has a minimum state complexity, we must prove that the upper and lower bound on state complexity are the same. When bounds are the same, we call this a tight bound.

Definition 8 (Fooling set method)**.** *A fooling set for a language L is defined as $FS = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$ such that $x_i y_i \in L$, for $i = 1, 2, ..., n$ and $x_i y_j \notin L$ or $x_j y_i \notin L$ where $1 \leq i, j \leq n$ and $i \neq j$.*

The canonical method for finding lower bounds on nondeterministic state complexity is the extended fooling set method [27, 29]. The fooling set method uses a fooling set to prove the existence of unique states through a natural extension of the Myhill-Nerode Theorem [28].

The fooling set method is used to provide a lower bound on nondeterministic state complexity of a language. We visualize the use of the fooling set in Figure 2.2. The fooling set contains words from L that are divided into x and y components. The computation is halted in a particular state when transitioning from reading x_1 to reading y_1. This particular state must be unique if no other pairing (x_i, y_i) can make a word in L with the words x_1y_i or x_iy_1. Examining Figure 2.2, if both x_1y_i and x_iy_1 were in the language then it would be ambiguous whether or not q_1 is equivalent to q_i.

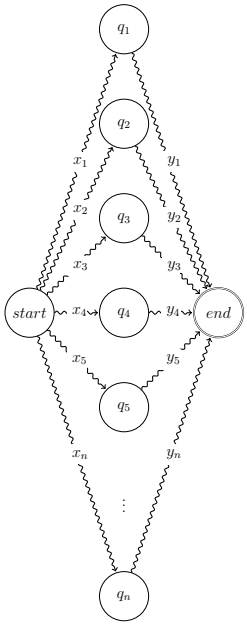

Figure 2.2: Visualization of the fooling set for an NFA with n states.

Since finding an exact lower bound using the fooling set method has been shown

to be NP-Hard[18], finding a tight lower bound requires insight into the language structure. Additionally, fooling sets can be arbitrarily bad with respect to nondeterministic state complexity [29].

2.4 Unary Languages

A unary language is a language where the alphabet Σ contains a single character. This means that every word in the language may only contain one letter repeated in varying amounts. A tally counting system is likely to be the most ubiquitous example of a unary language.

Chapter 3

Prefix, Suffix, Infix and Outfix

There has been a great deal of work conducted on prefix, suffix and infix computation when considering a single language. The majority of this work shows that a language is prefix-, suffix-, infix- or bifix-free [14, 22, 38]. Significant work has also been conducted to study DNA codes, which has focused primarily on encoding properties [31].

These particular operations have not been studied in the context of language operations between regular languages. In particular, the non-deterministic state complexity has not been described. The structures and techniques employed in the following chapter are utilized to improve upper and lower bound in chapter 4.

3.1 Prefix Operation

The prefix operation imposes that a particular prefix is within a set of recognized words. We can define the prefix restriction on strings by checking if a string $x \in \Sigma^+$ is the prefix of another string $w \in \Sigma^*$,

$$w \xleftarrow{Prefix} x = \begin{cases} \{w\} & \text{if } x \text{ a non-empty prefix of } w \\ \emptyset & \text{otherwise} \end{cases}$$

The result of this operation is to include the strings w that contain x as a proper prefix, and exclude all words that do not. This operation can be expanded to recognize languages $L_1, L_2 \subseteq \Sigma^*$ with the following,

Definition 9 (Prefix)**.** *For languages L_1 and L_2, the prefix language of L_2 on L_1 is defined as follows, $L_1 \xleftarrow{Prefix} L_2 = \{xz \mid xz \in L_1, x \in L_2, x \neq \varepsilon\}$*

The language $L_1 \xleftarrow{Prefix} L_2$ consists of words of L_1 that contains words from L_2 as non-empty prefixes. An automaton can be constructed to recognize $L_1 \xleftarrow{Prefix} L_2$ where NFAs A_1 and A_2 recognize L_1 and L_2 respectively.

Theorem 1. *For NFAs A_1 and A_2 with N and M states respectively. The language $L(A_1) \xleftarrow{Prefix} L(A_2)$ can be recognized with an NFA containing $NM + 1$ states.*

Proof. Let $A_1 = (Q, \Sigma, \delta, s_1, F_1)$ and $A_2 = (P, \Sigma, \gamma, s_2, F_2)$ recognize L_1 and L_2 with N and M states, respectively. Given these NFAs, we can construct an NFA $A_{Pre} = (Q_{Pre}, \Sigma, \Omega, (s_1', s_2'), F_{Pre})$, where

$$Q_{Pre} = \{(s_1', s_2')\} \cup (Q \times (P \backslash F_2)) \cup (Q \times \{p_f\}),$$

The state p_f is used to collapse all final states from F_2 into one. This is done since it does not matter which final state in A_2 was reached. The final states of A_{pre} are $(q_i, p_f) \in F_{Pre}$ where $q_i \in F_1$. The transitions of Ω are defined as follows:

(i) for the start state (s_1', s_2') :

$$\Omega((s_1', s_2'), \alpha) = \{(q', p') \mid q' \in \delta(s_1, \alpha) \text{ and } p' \in \gamma(s_2, \alpha), p' \notin F_2\}$$
$$\cup \{(q', p_f) \mid q' \in \delta(s_1, \alpha) \text{ and } \gamma(s_2, \alpha) \cap F_2 \neq \emptyset\}$$

(ii) for (q, p) where $q \in Q$ and $p \in P \backslash F_2$:

$$\Omega((q, p), \alpha) = \{(q', p') \mid q' \in \delta(q, \alpha) \text{ and } p' \in \gamma(p, \alpha), p' \notin F_2\}$$
$$\cup \{(q', p_f) \mid q' \in \delta(q, \alpha) \text{ and } \gamma(p, \alpha) \cap F_2 \neq \emptyset\}$$

(iii) for (q, p_f) where $q \in Q$:

$$\Omega((q, p_f), \alpha) = \{(q', p_f) \mid q' \in \delta(q, \alpha)\}$$

First we show the inclusion $L(A_1) \xleftarrow{Prefix} L(A_2) \subseteq L(A_{pre})$:

Generally speaking, A_{pre} uses the state (s_1', s_2') to enforce the non-empty prefix condition, the states (q, p) to simulate the intersection of the prefix x and the states (q, p_f) to simulate the suffix z.

We want to show that any word in $L(A_1) \xleftarrow{Prefix} L(A_2)$ is recognized by A_{pre}. To accomplish this, the states of A_{pre} can be divided into q and p components. An accepting computation on A_{pre} requires that both q and p components are in a final state after reading the input. To prove the language is a subset, we can follow the computation of any word $xz \in L(A_1) \xleftarrow{Prefix} L(A_2)$ for each state component and show that it must end in a final state. By the definition of $L(A_1) \xleftarrow{Prefix} L(A_2)$, xz is recognized by A_1 with an accepting path of states $\mathcal{P}_{A_1} = s_1, \ldots, q_x, \ldots, q_z$. Additionally,

x is recognized by A_2, with an accepting path of states $\mathcal{P}_{A_2}$=$s_2, \ldots, p_x$.

If we follow the q component first, we can see the initial state follows the transitions set in (i), which adheres to δ when transitioning. All other states also adhere to δ when transitioning the q component for both (ii) and (iii). Since A_{pre} can only be in an accepting state when the q component is an accepting state for A_1, the q component of A_{pre} "accepts" xz.

Next, we follow the p component of each state in A_{pre} as xz is read. The transitions in (i) and (ii) are consistent with γ. The only time it is inconsistent is when a transition would bring the p to an accepting state in A_2. When a transition would land in an accepting state of A_2, it is instead replaced with F in A_{pre}. Thus, since x is recognized by A_2, the p component of states in A_{pre} will result in F, which, once reached, only allows transitions from (iii). Transitions from (iii) never update the p component of the state. With the two components of a state in A_{pre} being satisfied, the accepting path $(s'_1, s'_2), \ldots, (q_x, p_f), \ldots, (q_z, p_f)$ exists for any word $xz \in L(A_1) \xleftarrow{Prefix} L(A_2)$ in $L(A_{pre})$.

Secondly we show the inclusion $L(A_{pre}) \subseteq L(A_1) \xleftarrow{Prefix} L(A_2)$:

We can show that the words in $L(A_{pre})$ are in $L(A_1) \xleftarrow{Prefix} L(A_2)$ by decomposing the words from $L(A_{pre})$ into two sub words w_1 and w_2. The prefix w_1 follows from transitions from (i) and (ii) and must be non-empty. The prefix w_1 must be a valid prefix in a word in $L(A_1)$ and a word in $L(A_2)$, this is because the transitions (i) and (ii) are consistent with δ and γ for q and p components respectively. The prefix w_1 end when the p component reaches a state p_f, signifying $w_1 \in L(A_2)$. Finally, the suffix w_2 only follows transitions defined in (iii), meaning the suffix only updates the state in terms of states from A_1. Since all transitions in A_{pre} adhere to δ, and the

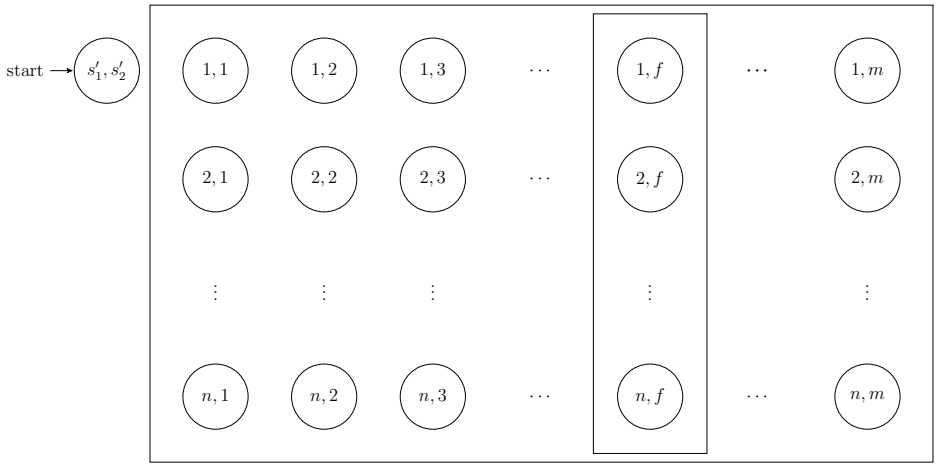

Figure 3.1: The construction of the prefix automaton.

final state in A_{pre} must have the q component in a final state, all words of A_{pre} must be in A_1. □

Figure 3.1 illustrates the construction of A_{Pre} that recognizes $L_1 \xleftarrow{Prefix} L_2$. In the figure, the boxes encapsulate different transition rules. The outer box reflects that classical intersection transitions are present and the inner box reflects the presence of A_1 transitions between the states regardless of the state in P. An example of the construction for the prefix automaton is contained in Figure 3.2.

It is interesting to note that the final states from A_2 collapse into a single state in the construction. This could lead to a linear factor of state saving for languages with many final states. But, in the worst case where, A_2 contains only one final state, there are no state savings for this construction.

It is also worth noting that the prefix operation, as defined in Theorem 1, produces a DFA if A_1 and A_2 are DFAs. This is because the transitions defined in Ω do not introduce nondeterminism.

Corollary 1. *If A_1 and A_2 are deterministic finite automata, then we can construct a DFA recognizing $L(A_1) \xleftarrow{Prefix} L(A_2)$ with $NM + 1$ states.*

To achieve a tight bound on state complexity, we need to provide a case where all states in the construction are necessary.

Theorem 2. *For M, $N \in \mathbb{N}$, there exists two regular languages L_1 and L_2 over binary alphabets with NFAs that have N and M states, respectively. Then, any NFA for $L_1 \xleftarrow{Prefix} L_2$ needs at least $MN + 1$ states.*

Proof. If we define the functions $\#_a(w)$ and $\#_b(w)$ to count the number of $a's$ and $b's$ for a given word w, we take $L_1 = \{w \in \{a, b\}^* | \#_a(w) \equiv 0 \mod N\}$ and $L_2 = \{w \in \{a, b\}^* | \#_b(w) \equiv 0 \mod M\}$ to construct a witness language $L_1 \xleftarrow{Prefix} L_2$. The witness language can be defined as follows,

$$L_1 \xleftarrow{Prefix} L_2 = \{xz \mid \#_a(xz) \equiv 0 \mod N \text{ and } \#_b(x) \equiv 0 \mod M\}$$

A fooling set FS can be constructed for $L_1 \xleftarrow{Prefix} L_2$,

$$FS = \{(b^{M-i}a^{N-j}, a^j b^i) | 0 \leq i < M, 0 \leq j < N\} \cup \{(\varepsilon, a^N b^M)\}$$

We can easily see that for any element (x, y), the word xy must contain N $a's$ and M $b's$, ensuring all the elements are in the language.

We consider the two pairs

$$(x, y) = (b^{M-i}a^{N-j}, a^j b^i) \text{ and } (x', y') = (b^{M-i'}a^{N-j'}, a^{j'} b^{i'}),$$

where $(i, j) \neq (i', j')$ and $0 \leq i, i' < M,\ 0 \leq j, j' < N$. We can show that either

(x, y') or (x', y) are not in the language $L_1 \xleftarrow{Prefix} L_2$, when $(x, y), (x', y') \in FS$.

For each word in the subset we have selected, the start of each word is a b character which imposes that an additional $M-1$ $b's$ must be in the prefix. If we assume $i < i'$ or $j < j'$ then, any word $x'y = b^{M-i'}a^{N-j'}a^j b^i$ has $(M - i' + i) \mod M \neq 0$ b's or $N - j' + j \mod N \neq 0$ a's since $i \neq i'$ or $j \neq j'$. This implies that you cannot make a word from xy'.

Thus far, the subset considered accounts for NM terms,but, we are also able to consider the additional element $(\varepsilon, a^N b^M)$. If we consider the words made by taking the first term of the additional element and concatenating it with the second term of all the elements we have already considered, then we get the following words, $\varepsilon \cdot a^j b^i$ where $0 \leq i < M,\ 0 \leq j < N$. These words are not in the witness language since they never contain a multiple of N a's or M b's except, when $i = j = 0$. When $i = j = 0$, we violate the minimum length since the word has a length of zero.

By constructing a fooling set, we have shown a lower bound on state complexity for an NFA, recognizing $L_1 \xleftarrow{Prefix} L_2$ has at least $NM + 1$ states. □

Since we have a lower bound and an upper bound of $MN + 1$, we can conclude, that this is a tight bound.

Corollary 2. *Let L_1 and L_2 be regular languages. Then in the worst case $NM + 1$ states are necessary and sufficient to recognize $L_1 \xleftarrow{Prefix} L_2$.*

3.2 Suffix Operation

Similar to the prefix operation, we can define a restriction on strings by checking if a string $z \in \Sigma^+$ is the suffix of another string $w \in \Sigma^*$,

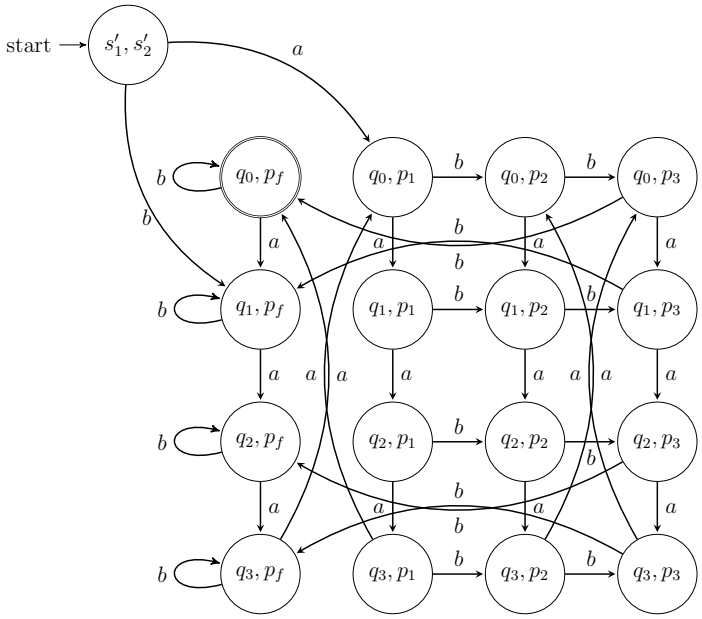

Figure 3.2: A diagram representing the automaton A_{pre} when $L_1 = \{w \in \{a,b\}^* | \#_a(w) \equiv 0 \mod 4\}$ and $L_2 = \{w \in \{a,b\}^* | \#_b(w) \equiv 0 \mod 4\}$.

$$w \xleftarrow{Suffix} z = \begin{cases} \{w\} & \text{if } z \text{ a non-empty suffix of } w \\ \emptyset & \text{otherwise} \end{cases}$$

The suffix operation recognizes strings w that contain z as a proper suffix, and excludes all words that do not. This operation can be expanded to recognize languages $L_1, L_2 \subseteq \Sigma^*$ with the following,

Definition 10 (Suffix)**.** *For languages L_1 and L_2 the suffix language of L_2 on L_1 is defined as follows, $L_1 \xleftarrow{Suffix} L_2 = \{xz \mid xz \in L_1, z \in L_2, z \neq \varepsilon\}$*

This operation recognizes words from L_1 that contain words from L_2 as a non-empty suffix. We can extend this definition to operate on NFAs A_1 and A_2 which recognize L_1 and L_2 respectively.

Theorem 3. *For NFAs A_1 and A_2 with N and M states respectively. The language $L(A_1) \xleftarrow{Suffix} L(A_2)$ can be recognized with an NFA containing $NM + 1$ states.*

Proof. Let $A_1 = (Q, \Sigma, \delta, s_1, F_1)$ and $A_2 = (P, \Sigma, \gamma, s_2, F_2)$ recognizing L_1 and L_2 with N and M states, respectively. Given these NFAs, we can construct an NFA $A_{Suf} = (Q_{Suf}, \Sigma, \Omega, (s_1, s_2), \{(F_1', F_2')\})$, where

$$Q_{Suf} = \{(F_1', F_2')\} \cup (Q \times P)$$

The final state (F_1', F_2') is added to enforce the non-empty property. The transitions of Ω are defined as follows,

(i) for (q, s_2) where $q \in Q$:

$$\Omega((q, s_2), \alpha) = \{(q', s_2) | q' \in \delta(q, \alpha)\}$$

(ii) for (q, p) where $q \in Q$ and $p \in P$:

$$\Omega((q, p), \alpha) = \{(q', p') \mid q' \in \delta(q, \alpha) \text{ and } p' \in \gamma(p, \alpha)\}$$
$$\cup \{(F_1', F_2') \mid \delta(q, \alpha) \cap F_1 \neq \emptyset \text{ and } \gamma(p, \alpha) \cap \ F_2 \neq \emptyset\}$$

First we show the inclusion $L(A_1) \xleftarrow{Suffix} L(A_2) \subseteq L(A_{suf})$:

Generally speaking, the states (q, s_2) simulate the computation of x and non-deterministically begin reading z. The states (q, p) then simulate the computation of z. Finally, the state (F_1', F_2') is used to ensure the non-empty condition on z.

It is necessary to show that if a word is in $L(A_1) \xleftarrow{Suffix} L(A_2)$ then A_{suf} recognizes it. If we take a word $xz \in L_1 \xleftarrow{Suffix} L_2$ then, let $\mathcal{P}_{A_1}(xz)$ be an accepting path

$s_1, \ldots, q_x, \ldots, q_z$ of $xz \in L(A_1)$ in A_1 and $\mathcal{P}_{A_2}(z)$ be an accepting path $s_2, \ldots, p_z$ of $z \in L(A_2)$ in A_2. If we read xz in A_{suf} then from the initial state, (s_1, s_2) of A_{suf} the first symbol of x is read using transitions from (i) and (ii). The transitions (i) allows A_{suf} to simulate the path $s_1, \ldots, q_x$ resulting in state (q_x, s_2). The transitions defined (ii) allow A_{suf} to simulate the paths $q_x, \ldots, q_z$ of $\mathcal{P}_{A_1}$ and $s_2, \ldots, p_z$ of $\mathcal{P}_{A_2}$.

The computation is complete when the state (F_1', F_2') is reached at the end of the input. This final state can only be reached by transition (ii) which enforces the minimum length of x, and requires both state components to reach a final state.
Secondly we show the inclusion $L(A_{suf}) \subseteq L(A_1) \xleftarrow{Suffix} L(A_2)$:

If A_{suf} recognizes xz, then, A_1 recognizes xz and A_2 recognizes z. We decompose an accepting path $(s_1, s_2), \ldots, (F_1', F_2')$ in A_{suf} for a string $xz \in L(A_{suf})$ into two paths $\mathcal{P}_1$ and $\mathcal{P}_2$ such that,

$$\mathcal{P}_1 = (s_1, s_2), ..., (q_x, s_2), \mathcal{P}_2 = (q_x, s_2), \ldots, (F_1', F_2')$$

Path $\mathcal{P}_1$ demonstrates the existence of a path $s_1, \ldots, q_x$ in $\mathcal{P}_{A_1}$ that recognizes a string x. Path $\mathcal{P}_2$ shows that there exists a path $q_x, \ldots, F_1$ in $\mathcal{P}_{A_1}$ and $s_2, \ldots, F_2$ in $\mathcal{P}_{A_2}$. This terminates the simulation at the final state (F_1', F_2'), which implies that, if A_{suf} recognizes xz, then there exists an accepting paths on A_1 and A_2 that recognize strings xz and z, where $z \neq \varepsilon$. □

Figure 3.3 illustrates the construction of the automaton A_{Suf} which recognizes the language $L_1 \xleftarrow{Suffix} L_2$. In the figure, the boxes encapsulate different transition rules. The outer box reflects the classical intersection transitions are present and the inner

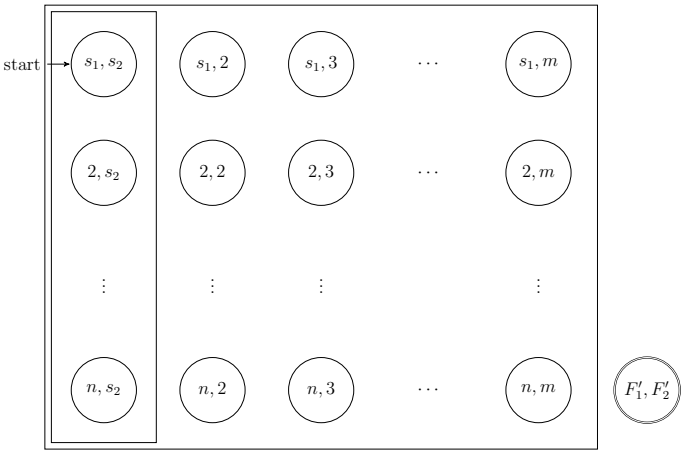

Figure 3.3: The construction of the suffix automaton.

box reflects the presence of A_1 transitions between the states inside without needing to advance the state in P. All states contained inside the boxes may transition to (F_1', F_2') provided the criteria in (ii) are met.

The proof for the lower bound for the suffix operation is essentially the same as the lower bound for prefix. We can intuit this result from the fact that if we reverse the fooling set for the prefix operation we get the fooling set for the suffix operation. Thus the two operation should have the same lower bound, and near identical lower bound proofs.

Theorem 4. *For M, $N \in \mathbb{N}$, there exists two regular languages L_1 and L_2 over binary alphabets with NFAs that have N and M states respectively. Then any NFA for $L_1 \xleftarrow{Suffix} L_2$ needs at least $MN + 1$ states.*

Proof. If we define the functions $\#_a(w)$ and $\#_b(w)$ to count the number of $a's$ and $b's$ for a given word w, we take $L_1 = \{w \in \{a, b\}^* \mid \#_a(w) \equiv 0 \mod N\}$ and $L_2 = \{w \in \{a, b\}^* \mid \#_b(w) \equiv 0 \mod M\}$ to construct a witness language $L_1 \xleftarrow{Suffix} L_2$.

The witness language can be defined as follows,

$$L_1 \xleftarrow{Suffix} L_2 = \{xz \mid \#_a(xz) \equiv 0 \mod N \text{ and } \#_b(x) \equiv 0 \mod M\}$$

A fooling set FS can be constructed for $L_1 \xleftarrow{Suffix} L_2$,

$$FS = \{(b^i a^j, a^{N-j} b^{M-i}) \mid 0 \leq i < M, 0 \leq j < N\} \cup \{(a^N b^M, \varepsilon)\}$$

We can easily see that for any element (x, y), the word $x \cdot y$ must contain N $a's$ and M $b's$, ensuring all the elements are in the language.

We consider the two pairs

$$(x, y) = (b^i a^j, a^{N-j} b^{M-i}) \text{ and } (x', y') = (b^{i'} a^{j'}, a^{N-j'} b^{M-i'}),$$

where $(i, j) \neq (i', j')$ and $0 \leq i, i' < M,\ 0 \leq j, j' < N$. We can show that either (x, y') or (x', y) are not in the language $L_1 \xleftarrow{Suffix} L_2$.

Since each word in the subset ends with a b character this imposes that an additional $M - 1$ $b's$ must be in the suffix. If we assume $i < i'$ or $j < j'$ then, any word $x'y = b^{i'} a^{j'} \cdot a^{N-j} b^{M-i}$ has $(M - i + i') \mod M \neq 0$ b's or $N - j + j' \mod N \neq 0$ a's since $i \neq i'$ or $j \neq j'$. This implies that the word xy' is not in $L_1 \xleftarrow{Suffix} L_2$.

Thus far, the subset considered accounts for NM terms, but we are also able to consider the additional element $(a^N b^M, \varepsilon)$. If we consider the words made by taking the second term of the additional element and concatenating it with the first term of all the elements we have already considered, we get the following words, $b^i a^j \cdot \varepsilon$ where $0 \leq i < M,\ 0 \leq j < N$. These words are not in the witness language since they

never contain a multiple of N a's or M b's except, when $i = j = 0$. When $i = j = 0$, we violate the minimum length since the word has a length of zero.

By constructing a fooling set, we have shown a lower bound on state complexity for an NFA, recognizing $L_1 \xleftarrow{Suffix} L_2$ has at least $NM + 1$ states. □

Since we have a lower bound and an upper bound of $MN + 1$, we can conclude, that this is a tight bound.

Corollary 3. *Let L_1 and L_2 be regular languages. Then in the worst case $NM + 1$ states are necessary and sufficient to recognize $L_1 \xleftarrow{Suffix} L_2$.*

The construction for the suffix operation, although similar to the prefix operation does not share all the same properties. The construction described in theorem 3 implies that the automaton is nondeterministic. This nondeterminism is introduced when the machine nondeterministically "guesses" when the non-empty suffix begins. Because of this nondeterminism, even if we provide DFAs A_1 and A_2 that recognize L_1 and L_2 respectively the constructed automaton will be an NFA. This means that the upper bound on deterministic state complexity of the suffix operation remains an open problem.

3.3 Infix Operation

With the infix operation, it is no longer required that the sub-word recognized is fixed at the beginning or end of the word, as was the case with prefix or suffix operations. The infix operation, as a word constraint, checks if a string $y \in \Sigma^+$ is the infix of another string $w \in \Sigma^*$,

$$w \xleftarrow{Infix} y = \begin{cases} \{w\} & \text{if } y \text{ a non-empty infix of } w \\ \emptyset & \text{otherwise} \end{cases}$$

The result of this operation is the acceptance of strings w that contain y as an infix, and a rejection of all words that do not. This operation can be expanded to apply to languages $L_1, L_2 \subseteq \Sigma^*$ with the following,

Definition 11 (Infix)**.** *For languages L_1 and L_2 the infix language of L_2 on L_1 is defined as follows, $L_1 \xleftarrow{Infix} L_2 = \{xyz \mid xyz \in L_1, y \in L_2, y \neq \varepsilon\}$*

The infix operation can be extended to operate on NFAs A_1 and A_2 which recognize L_1 and L_2 respectively. This extension provides an upper bound on nondeterministic state complexity.

Theorem 5. *For NFAs A_1 and A_2 with N and M states respectively. The language $L(A_1) \xleftarrow{Infix} L(A_2)$ can be recognized with an NFA containing $NM + N$ states.*

Proof. Let $A_1 = (Q, \Sigma, \delta, s_1, F_1)$ and $A_2 = (P, \Sigma, \gamma, s_2, F_2)$ recognizing L_1 and L_2 with N and M states, respectively. Given these NFAs, we can construct an NFA $A_{in} = (Q_{in}, \Sigma, \Omega, (s_1, \clubsuit), F_{in})$, where

$$Q_{in} = (Q \times \{\clubsuit\}) \cup (Q \times (P \backslash F_2)) \cup (Q \times \{p_f\})$$

The $Q \times \{p_f\}$ are used to represent all the final states from A_2. The final states for A_{in} are F_{in}=(q_i, p_f) where $q_i \in F_1$. The transitions of Ω are defined as follows:

(i) for $(q, \clubsuit)$ where $q \in Q$:

$$\begin{aligned}\Omega((q,\clubsuit),\alpha) = &\{(q',\clubsuit) \mid q' \in \delta(q,\alpha)\} \\ &\cup \{(q',p') \mid q' \in \delta(q,\alpha) \text{ and } p' \in \gamma(s_2,\alpha)\} \\ &\cup \{(q',p_f) \mid q' \in \delta(q,\alpha) \text{ and } \gamma(s_2,\alpha) \cap\ F_2 \neq \emptyset\}\end{aligned}$$

(ii) for (q, p) where $q \in Q$ and $p \in P \backslash F_2$:

$$\begin{aligned}\Omega((q,p),\alpha) = &\{(q',p') \mid q' \in \delta(q,\alpha) \text{ and } p' \in \gamma(p,\alpha)\} \\ &\cup \{(q',p_f) \mid q' \in \delta(q,\alpha) \text{ and } \gamma(p,\alpha) \cap\ F_2 \neq \emptyset\}\end{aligned}$$

(iii) for (q, p_f) where $q \in Q$:

$$\Omega((q,p_f),\alpha) = \{(q',p_f) \mid q' \in \delta(q,\alpha)\}$$

First we show the inclusion $L(A_1) \xleftarrow{Infix} L(A_2) \subseteq L(A_{in})$:

Generally speaking, the states $(q, \clubsuit)$ simulate the computation of x and non-deterministically start reading y. The states (q, p) are required to compute the infix y. Finally, the states (q, p_f) are used to finish the computation of z.

We need to prove that if a word in $L(A_1) \xleftarrow{Infix} L(A_2)$ then the word is recognized by A_{in}. Let $\mathcal{P}_{A_1}(xyz)$ be an accepting path $s_1, \ldots, q_x, \ldots, q_y, \ldots, q_z$ of $xyz \in L(A_1)$ on A_1 and $\mathcal{P}_{A_2}(y)$ be an accepting path $s_2, \ldots, p_y$ of $y \in L(A_2)$ on A_2. From the initial state $(s_1, \clubsuit)$ of A_{in}, the first sub-word x is read using transitions from (i) resulting in state $(q_x, \clubsuit)$. Then, the automaton A_{in} can non-deterministically decide to move to states (q, p) upon reading the first symbol of y. A_{in} then reads y using transitions

in (ii) resulting in the state (q_y, p_f). At the final symbol in y, A_{in}, then transitions to state (q_y, p_f) since p_y is an accepting state in A_2. Finally, A_{in}, then reads z and uses the transitions defined in (iii) to move from (q_y, p_f) to (q_z, p_f) which is the final state.

Secondly we show the inclusion $L(A_{in}) \subseteq L(A_1) \xleftarrow{Infix} L(A_2)$:

If A_{in} recognizes xyz then A_1 recognizes xyz and A_2 recognizes y. We can decompose an accepting path $(s'_1, \clubsuit), \ldots, (q_z, p_f)$ for a string $xyz \in L(A_{suf})$ into three paths $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P}_3$ such that,

$$\mathcal{P}_1 = (s'_1, \clubsuit), ..., (q'_x, \clubsuit), \mathcal{P}_2 = (q_x, s_2), \ldots, (q_y, p_f), \mathcal{P}_3 = (q_y, p_f), \ldots, (q_z, p_f)$$

Path $\mathcal{P}_1$ shows that there exists a path $s_1, \ldots, q_x$ in $\mathcal{P}_{A_1}$ that recognizes a string x. Path $\mathcal{P}_2$ shows that there exists a path $q_x, \ldots, q_y$ in $\mathcal{P}_{A_1}$ and $s_2, \ldots, p_y$ in $\mathcal{P}_{A_2}$. The final path $\mathcal{P}_3$ shows that there is a path $q_y, \ldots, q_z$ in $\mathcal{P}_{A_1}$. This terminates the simulation at the final state (q_z, p_f), and it implies that if A_{in} recognizes xyz, then there exists an accepting paths on A_1 and A_2 that recognize strings xyz and y. □

Figure 3.4 illustrates the construction of A_{in} that recognizes $L(A_1) \xleftarrow{Infix} L(A_2)$. In the figure, the boxes encapsulate different transition rules. The left box contains states where the out transitions are defined in (i), and the right box reflects the presence of out transitions defined by (ii). Finally, the states that only have out transitions defined by (iii) are contained in the interior box on the right. An example of the construction for the infix automaton is contained in Figure 3.5.

It is interesting to note that, as with the prefix operation, the infix operation

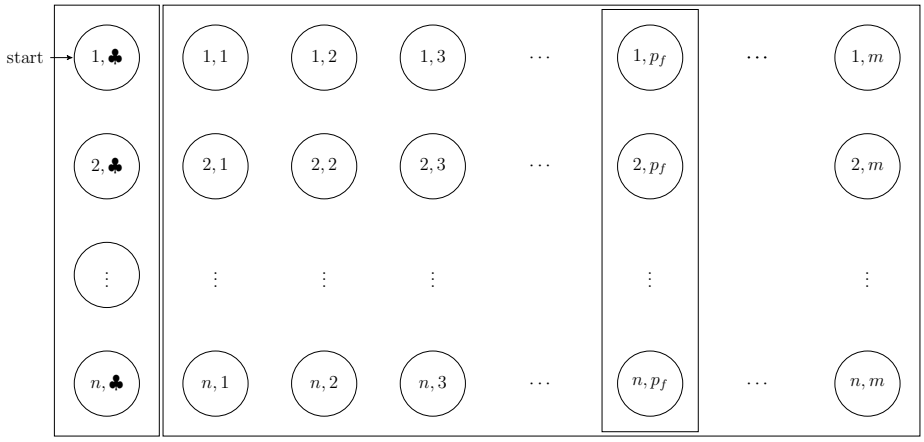

Figure 3.4: The construction of the infix automaton.

construction also collapses the final states of A_2 into a single state in the construction. This could lead to a linear factor of state savings for languages with many final states. But, in the worst case, where A_2 contains only one final state, there are no state savings for this construction.

Theorem 6. *For M, $N \in \mathbb{N}$, there exists two regular languages L_1 and L_2 over binary alphabets with NFAs that have N and M states respectively. Then any NFA for $L_1 \xleftarrow{Infix} L_2$ needs at least $MN + N$ states.*

Proof. Let $L_1 = \{w \in \{a, b\}^* | \#_a(w) \equiv 0 \mod N\}$ and $L_2 = \{w \in \{b\}^* | \#_b(w) \equiv 0 \mod M\}$. We can construct a fooling set $\text{FS} = FS_1 \cup FS_2$ for the language $L_1 \xleftarrow{Infix} L_2$ where,

$$FS_1 = \{(a^j b^i, b^{M-i} a^{N-j}) | 0 \leq i < M, 0 \leq j < N\}$$

$$FS_2 = \{(a^j b^M, a^{N-j}) | 0 \leq j < N\}$$

We need to prove the fooling properties hold for each set and between the sets. The

witness language is defined by,

$$L_1 \xleftarrow{Infix} L_2 = \{w \in ((a^*)(b^*))^*(b^M)^+((a^*)(b^*))^* | \#_a(w) \equiv 0 \mod N\}$$

It is easy to see that all words in FS_1 and FS_2 contain N $a's$ and M $b's$. For any $(x, y), (x', y') \in FS_1$ where $(x, y) \neq (x', y')$, we need to show (x, y') or (x', y) are not in the language. If we take $xy' = a^j b^i \cdot b^{M-i'} a^{N-j'}$ and assume $j < j'$, then $a^j b^i b^{M-i} a^{N-j'} \notin L_1 \xleftarrow{Infix} L_2$. This is because the number of $a's$ is less than N but greater than 0. If we assume $i < i'$ then, $xy' = a^j b^i b^{M-i'} a^{N-j} \notin L_1 \xleftarrow{Infix} L_2$. This is because the number of $b's$ is less than M but greater than 0.

For any $(x, y), (x', y') \in FS_2$ where $(x, y) \neq (x', y')$, we can take $xy' = a^j b^M a^{N-j'}$ where $j < j'$, then, this word is never in the language since the number of $a's$ is always less than N but greater than 0.

Finally, if we take $(x, y) \in FS_1$ and $(x', y') \in FS_2$, then, the word $xy' = a^j b^i a^{N-j'}$, since $0 \leq i < M$ in FS_1. We know that the number of $b's$ will always be less than M, meaning, it will never create a word in $L_1 \xleftarrow{Infix} L_2$.

By the fooling set theorem, we have shown that for any NFA recognizing $L_1 \xleftarrow{Infix} L_2$ it has at least $NM + N$ states. □

Since we have a lower bound and an upper bound of $MN + N$, we can conclude, that this is a tight bound.

Corollary 4. *Let L_1 and L_2 be regular languages. Then in the worst case, $NM + N$ states are necessary and sufficient to recognize $L_1 \xleftarrow{Infix} L_2$.*

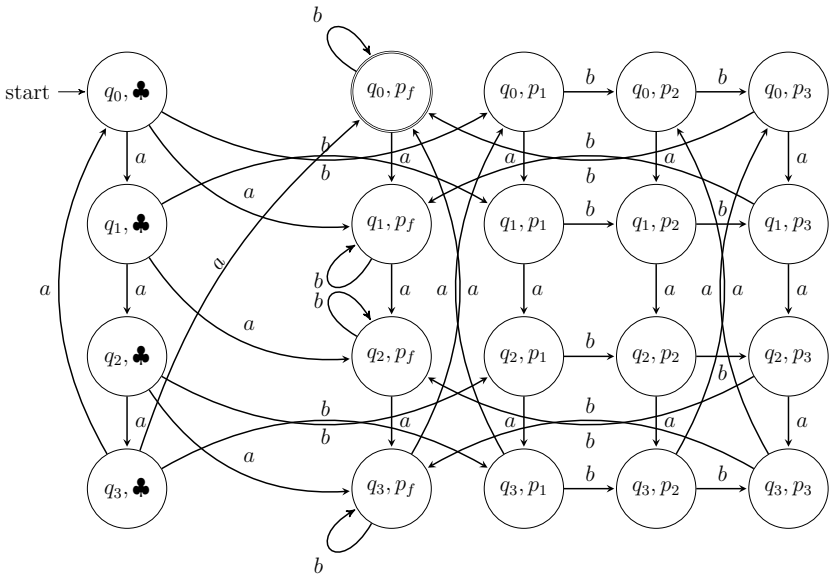

Figure 3.5: A diagram representing the automaton A_{in} when $L_1 = \{w \in \{a,b\}^* | \#_a(w) \equiv 0 \mod 4\}$ and $L_2 = \{w \in \{a,b\}^* | \#_b(w) \equiv 0 \mod 4\}$.

3.4 Outfix Operation

The final operation for this chapter is the outfix operation. The outfix operation is distinct from the other operations. The outfix operator enforces that a word can be split in two and have the first half be a non-empty prefix and the second half, a non-empty suffix, without overlapping the prefix and suffix. The outfix operation, as a word constraint, checks if a string $xz \in \Sigma^*$ and $2 \leq |xz|$ is the outfix of another string $w \in \Sigma^*$,

$$w \xleftarrow{Outfix} xz = \begin{cases} \{w\} & \text{if } xz \text{ is a non-empty outfix of } w \\ \emptyset & \text{otherwise} \end{cases}$$

The result of this operation is the acceptance of strings w that contain xz as an outfix, and a rejection of all words that do not. This operation can be expanded to

apply to languages $L_1, L_2 \subseteq \Sigma^*$ with the following,

Definition 12 (Outfix)**.** *For languages L_1 and L_2 the outfix language of L_2 on L_1 is defined as follows,*

$$L_1 \xleftarrow{Outfix} L_2 = \{xyz \mid xyz \in L_1, xz \in L_2, x \neq \varepsilon, z \neq \varepsilon\}$$

Next we consider the nondeterministic state complexity of the outfix operation. First we give an NFA construction that provides an upper bound for the state complexity.

Theorem 7. *For NFAs A_1 and A_2 with N and M states respectively. The language $L(A_1) \xleftarrow{Outfix} L(A_2)$ can be recognized with an NFA containing $3NM$ states.*

Proof. Let $A_1 = (Q, \Sigma, \delta, s_1, F_1)$ and $A_2 = (P, \Sigma, \gamma, s_2, F_2)$ recognizing L_1 and L_2 with N and M states, respectively. Given these NFAs, we can construct an NFA $A_{out} = (Q_{out}, \Sigma, \Omega, (s_1, s_2)_{\mathcal{X}_1}, F_{out})$, where

$$Q_{out} = (Q \times P)_{\mathcal{X}_1} \cup (Q \times P)_{\mathcal{X}_2} \cup (Q \times P)_{\mathcal{X}_3}$$

The final states are defined by $F_{out} = \{(q, p)_{\mathcal{X}_3} \mid q \in F_1, p \in F_2\}$. Each set of states follow their own transition rules in Ω, which are as follows:

(i) for $(q, p)_{\mathcal{X}_1}$ where $q \in Q$ and $p \in P$:

$$\begin{aligned}\Omega((q, p)_{\mathcal{X}_1}, \alpha) = &\{(q', p')_{\mathcal{X}_1} \mid q' \in \delta(q, \alpha) \text{ and } p' \in \gamma(p, \alpha)\} \\ &\cup \{(q', p')_{\mathcal{X}_2} \mid q' \in \delta(q, \alpha) \text{ and } p' \in \gamma(p, \alpha)\}\end{aligned}$$

(ii) for $(q,p)_{\mathcal{X}_2}$ where $q \in Q$ and $p \in P$:

$$\Omega((q,p)_{\mathcal{X}_2}, \alpha) = \{(q',p)_{\mathcal{X}_2} \mid q' \in \delta(q,\alpha)\} \cup \{(q',p')_{\mathcal{X}_3} \mid q' \in \delta(q,\alpha) \text{ and } p' \in \gamma(p,\alpha)\}$$

(iii) for $(q,p)_{\mathcal{X}_3}$ where $q \in Q$ and $p \in P$:

$$\Omega((q,p)_{\mathcal{X}_3}, \alpha) = \{(q',p')_{\mathcal{X}_3} \mid q' \in \delta(q,\alpha) \text{ and } p' \in \gamma(p,\alpha)\}$$

First we show the inclusion $L(A_1) \xleftarrow{Outfix} L(A_2) \subseteq L(A_{out})$:

Generally speaking, the states $(q,p)_{\mathcal{X}_1}$ simulate the computation of x and non-deterministically start reading y. The states $(q,p)_{\mathcal{X}_2}$ are required to compute the sub-string y. Finally, the states $(q,p)_{\mathcal{X}_3}$ are used to finish the computation of z.

We need to prove that if a word is in $L(A_1) \xleftarrow{Outfix} L(A_2)$, then it is recognized by A_{out}. If we take an arbitrary word $xyz \in L(A_1) \xleftarrow{Outfix} L(A_2)$, then, we can describe the accepting path for xyz and xz in A_1, A_2, respectively. Let $\mathcal{P}_{A_1}(xyz)$ be an accepting path $s_1, \ldots, q_x, \ldots, q_y, \ldots, q_z$ of $xyz \in L(A_1)$ on A_1 and $\mathcal{P}_{A_2}(xz)$ be an accepting path $s_2, \ldots, p_x, \ldots, p_z$ of $xz \in L(A_2)$ on A_2.

If we simulate the computation of xyz on A_{out}, we can get the following accepting path. Starting at the initial state $(s_1, s_2)_{\mathcal{X}_1}$ and then using the transitions defined in (i) we are able to transition to $(q_x, p_x)_{\mathcal{X}_2}$ after reading x and using the nondeterministic transition to $(q_x, p_x)_{\mathcal{X}_2}$ instead of $(q_x, p_x)_{\mathcal{X}_1}$. From $(q_x, p_x)_{\mathcal{X}_2}$, we use the transitions defined in (ii) to read y resulting in the state $(q_y, p_x)_{\mathcal{X}_2}$. Lastly, we read z and using the nondeterministic transition to the states $(Q, P)_{\mathcal{X}_3}$. Upon reading the first symbol of z, we are then confined to using the transitions defined in (iii), ending the simulation

in $(q_z, p_z)_{\mathcal{X}_3}$, which is a final state in A_{out}.

Secondly we show the inclusion $L(A_{out}) \subseteq L(A_1) \xleftarrow{Outfix} L(A_2)$:

We need to show that for any word xyz recognized by A_{out}, then the word xyz is recognize by A_1 and xz is recognized by A_2. If we take an arbitrary word $w_1w_2w_3 \in L(A_{out})$ and consider the decomposition of an accepting path through A_{out}.

$$\mathcal{P}_{w_1} = (s_1, s_2)\mathcal{X}_1, \ldots, (q_{w_1}, p_{w_1})\mathcal{X}_2, \mathcal{P}_{w_2} = (q'_{w_1}, p_{w_1})\mathcal{X}_2, \ldots, (q_{w_2}, p_{w_1})\mathcal{X}_2$$

$$\mathcal{P}_{w_3} = (q'_{w_2}, p'_{w_1})\mathcal{X}_3, \ldots, (q_{w_3}, p_{w_3})\mathcal{X}_3$$

Where the state q'_{w_1} is a state in $\delta(q_{w_1}, \beta_{w_2})$, if β_{w_2} is the first symbol of w_2. Similarly, the states q'_{w_2}, p'_{w_1} are states in $\delta(q_{w_2}, \beta_{w_3}), \gamma(p_{w_1}, \beta_{w_3})$ if β_{w_3} is the first symbol of w_3.

The path $\mathcal{P}_{w_1}$ shows that a path $s_1, \ldots, q_{w_1}$ exists in $\mathcal{P}_{A_1}$ and $s_2, \ldots, p_{w_1}$ in $\mathcal{P}_{A_2}$. Path $\mathcal{P}_{w_2}$ is optional and can be circumvented with the nondeterministic transitions defined in (ii). If a path $\mathcal{P}_{w_2}$ does exist, it implies that a path $q_{w_1}, \ldots, q_{w_2}$ exists in $\mathcal{P}_{A_1}$. Finally, the path $\mathcal{P}_{w_3}$ shows a path $q_{w_2}, \ldots, q_{w_3}$ is in path $\mathcal{P}_{A_1}$, as well as a path $p_{w_1}, \ldots, p_{w_3}$ in $\mathcal{P}_{A_2}$. The simulation ends in the final state (q_{w_3}, p_{w_3}), when the string $w_1w_2w_3$ is recognized by A_1 and w_1w_2 is recognized by A_2. □

Figure 3.6 illustrates the construction of the automaton A_{out} which recognizing $L(A_1) \xleftarrow{Outfix} L(A_2)$. In the figure, the boxes encapsulate different transition rules. The left box contains all the states with a subscript of $\mathcal{X}_1$, which means that their transitions rules are defined in (i). The middle box contains states with $\mathcal{X}_2$ as their subscript, the lengthwise rectangles inside the box indicate the inability to change

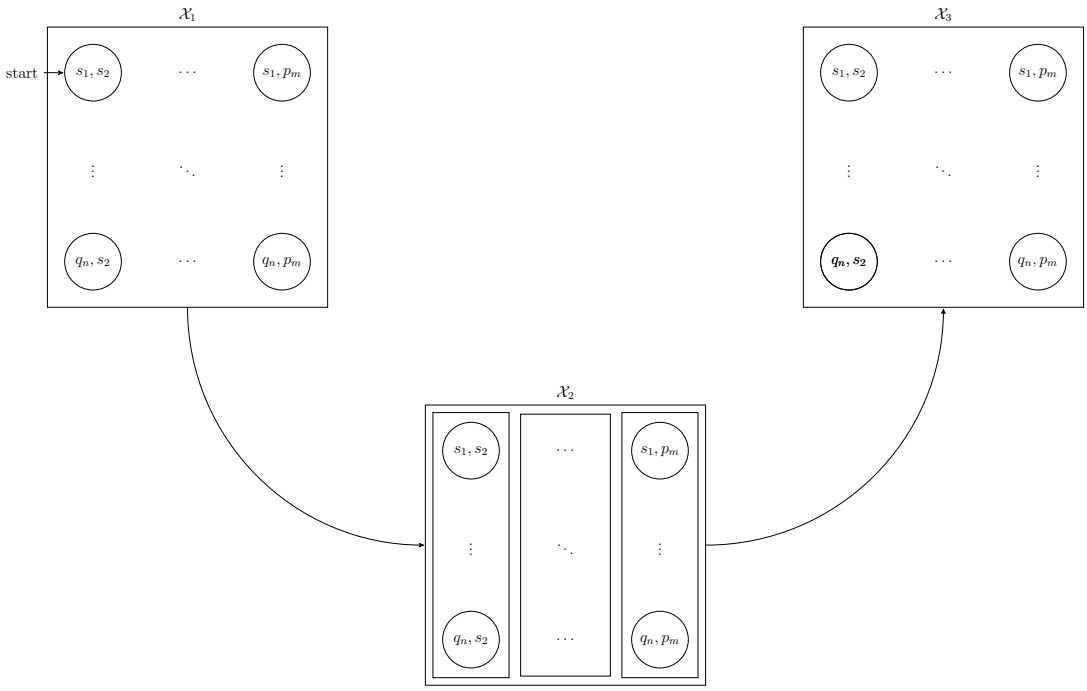

Figure 3.6: The construction of the outfix automaton.

the p component of the states using the transitions defined in (ii). Finally, the right box contains states with $\mathcal{X}_3$ as a subscript, these states use transitions defined in (iii) and contain all final states present in the machine.

Theorem 8. *For M, $N \in \mathbb{N}$, there exists two regular languages L_1 and L_2 with NFAs that have N and M states respectively. Then any NFA for $L_1 \xleftarrow{Outfix} L_2$ needs at least 2N(M-1) states.*

Proof. Let $L_1 = \{w \in \{a,b,c\}^* | \#_a(w) \equiv 0 \mod N\}$ and $L_2 = \{w \in \{a,b\}^* | \#_b(w) \equiv 0 \mod M\}$. We can construct a fooling set $FS = FS_1 \cup FS_2$ for the language $L_1 \xleftarrow{Outfix} L_2$ where,

$$FS_1 = \{(b^i a^j c b^{M-i}, b^i c a^{N-j} b^{M-i}) \mid 1 \leq i < M, 0 \leq j < N\}$$

$$FS_2 = \{(b^i a^j, b^{M-i} c a^{N-j}) \mid 1 \leq i < M, 0 \leq j < N\}$$

We need to prove the fooling properties hold for each set and between the sets. The witness language is defined by,

$$\begin{aligned} L_1 \xleftarrow{Outfix} L_2 = \{xyz \mid xz \in \{a,b\}^* \text{ and } \#_b(xz) \equiv 0 \mod M, \\ y \in \{a,b,c\}^* \text{ and } \#_a(xyz) \equiv 0 \mod N\} \end{aligned}$$

For any word in FS_1 we can see that it contains N $a's$ easily enough, but each word starts with a b, meaning that the outfix must contain additional $b's$. There must always be M $b's$ in the outfix we can verify this if we consider any sub-word left of the first c to be the prefix and any sub-word right of the second c to be the suffix. In this way, we always have an outfix with M $b's$. As with FS_1, the words in FS_2 always contains N $a's$, and if we consider the c to be the infix y, then there is always M $b's$ in the outfix.

For any $(x,y),(x',y') \in FS_1$ where $(x,y) \neq (x',y')$. We need to show (x,y') or (x',y) are not in the language. If we take $xy' = b^i a^j c b^{M-i} \cdot b^i c a^{N-j'} b^{M-i}$ and assume $j < j'$, then $b^i a^j c b^{M-i} \cdot b^i c a^{N-j'} b^{M-i} \notin L_1 \xleftarrow{Outfix} L_2$. This is because the number of $a's$ is less than N but greater than 0. If we assume $i < i'$, then, $xy' = b^i a^j c b^{M-i} \cdot b^i c a^{N-j'} b^{M-i'} \notin L_1 \xleftarrow{Outfix} L_2$. This is because the word starts with a b, implying that the outfix must contain a b. Since the outfix must be a word in L_2, then there must be an additional $M-1$ $b's$ in the outfix. Nothing between the two $c's$ in these words can be in the outfix since c is only in Σ_1, then the number of $b's$ in the outfix is $(M - i' + i) \mod M \neq 0$.

For any $(x,y),(x',y') \in FS_2$ where $(x,y) \neq (x',y')$. If we take $xy' = b^i a^j \cdot$

$b^{M-i}ca^{N-j'}$ where $j < j'$, then, this word is never in the language because the number of $a's$ is always less than N but greater than 0. If instead, we take $xy' = b^i a^j \cdot b^{M-i'}ca^{N-j}$ where $i < i'$, then, the word always starts with a b and the rest of the prefix before the c must have an additional $M-1$ $b's$. This is not true in the case where $i < i'$.

Finally, if we take $(x, y) \in FS_1$ and $(x', y') \in FS_2$, then, the word $xy' = b^i a^j cb^{M-i} \cdot b^{M-i}ca^{N-j}$ is never in $L_1 \xleftarrow{Outfix} L_2$. We know this because the word always starts with i $b's$, where $1 \leq i < M$, but all remaining $b's$ in the word are between two $c's$ forcing them into the y component of the word. Thus, the number of $b's$ will always be less than M.

By the fooling set theorem, we have shown that for any NFA recognizing $L_1 \xleftarrow{Outfix} L_2$ it has at least $2N(M-1)$ states. □

The bounds we have provided for the outfix operation contain a gap. An NFA representing the outfix operation needs at most $3NM$ states, as shown in Theorem 7. The lower bound on nondeterministic state complexity as proven in Theorem 8, is $2NM - 2N$. This leaves a gap in the nondeterministic state complexity, thus the exact bound remains an open problem.

Chapter 4

Site-Directed Insertion and Variants

In this chapter we explore insertion operations first described in 2017, by Cho et al. [7], those operations being *prefix-guided insertion*, *suffix-guided insertion* and *outfix-guided insertion*. In a follow-up paper [8], these authors focused on restricted forms of the language, mainly those of maximal and minimal insertion and their closure properties. They also included a tight bound on nondeterministic state complexity for alphabetic site-directed insertion. Additionally, these authors also changed the name of the outfix-guided insertion operation to *site-directed insertion*, which likens the operation to its biological inspiration of site-directed mutagenesis. To avoid using multiple naming systems, I have chosen to conform with the most current 2019 nomenclature of *prefix-directed insertion* (PreDI), *suffix-directed insertion* (SufDI) and *site-directed insertion* (SDI).

4.1 Prefix-Directed Insertion

Prefix-directed insertion uses a guiding string to identify possible locations for insertion into the host string. The use of prefix in the name of the operation refers to the requirement that the prefix of the insertion string match the sub-string in the host.

Definition 13 (Prefix-Directed Insertion)**.** *For languages L_1 and L_2, the prefix directed insertion language of L_2 into L_1 is defined as follows:*

$$L_1 \xleftarrow{PreDI} L_2 = \{xy_1\bar{y}z \mid xy_1z \in L_1, y_1\bar{y} \in L_2, y_1 \neq \epsilon\}$$

For regular languages specified by NFAs A_1 and A_2, we construct an ε-NFA for the prefix-directed insertion of $L(A_2)$ into $L(A_1)$. The construction gives an upper bound for the nondeterministic state complexity of prefix-guided insertion.

Theorem 9. *For NFAs A_1 and A_2 with N and M states, respectively, the language $L(A_1) \xleftarrow{PreDI} L(A_2)$ is recognized by an ε-NFA containing $2NM + N$ states.*

Proof. Let $A_1 = (Q, \Sigma, \delta, s_1, F_1)$ and $A_2 = (P, \Sigma, \gamma, s_2, F_2)$ be automata recognizing languages L_1 and L_2 with N and M states, respectively. Given these NFAs, we can construct an NFA $A_{PreDI} = (Q_{PreDI}, \Sigma, \Omega, (s_1, s_2), F_{PreDI})$, where

$$Q_{PreDI} = \{(Q \times P)\} \cup \{(\bar{Q} \times P)\} \cup \{(Q \times \{\clubsuit\})\},$$

The states $\bar{q} \in \bar{Q}$ have corresponding states $q \in Q$ although Q and $\bar{Q}$ are disjoint sets. The final states of A_{PreDI}, F_{PreDI}, are the states $(q_f, \clubsuit)$ where $q_f \in F_1$. The transition relation Ω is defined as follows:

(i) for (q, p) where $q \in Q$ and $p \in P$:

$$\begin{aligned}\Omega((q,p),\alpha) = &\{(q', p') \mid q' \in \delta(q, \alpha) \text{ and } p' \in \gamma(p, \alpha)\} \\ &\cup \{(\bar{q}', p') \mid \bar{q}' \in \delta(q, \alpha) \text{ and } p' \in \gamma(p, \alpha)\} \\ &\cup \{(q', s_2) \mid q' \in \delta(q, \alpha) \text{ and } p = s_2\}\end{aligned}$$

(ii) for $(\bar{q}, p)$ where $\bar{q} \in \bar{Q}$ and $p \in P$:

$$\Omega((\bar{q}, p), \alpha) = \{(\bar{q}, p') \mid p' \in \gamma(p, \alpha)\}$$

(iii) for $(\bar{q}, p)$ where $\bar{q} \in \bar{Q}$ and $p \in P$:

$$\Omega((\bar{q}, p), \epsilon) = \{(q, \clubsuit) \mid p \in F_2\}$$

(iv) for $(q, \clubsuit)$ where $q \in Q$:

$$\Omega((q, \clubsuit), \alpha) = \{(q', \clubsuit) \mid q' \in \delta(q, \alpha)\}$$

Firstly we show the inclusion $L(A_1) \xleftarrow{PreDI} L(A_2) \subseteq L(A_{PreDI})$:

Generally speaking, A_{PreDI} uses the states (q, s_2) to read the substring x and the states (q, p) to read the substring y_1. The states $(\bar{q}, p)$ are used to read the substring $\bar{y}$. Finally, the states $(q, \clubsuit)$ are used to complete the computation, reading z.

We need to show that, for any word in $L(A_1) \xleftarrow{PreDI} L(A_2)$, it is also in $L(A_{PreDI})$. If we take a word $xy_1\bar{y}z \in L(A_1) \xleftarrow{PreDI} L(A_2)$, then there must be a path $\mathcal{P}_{A_1} = s_1, \ldots, q_x, \ldots, q_{y_1}, \ldots, q_z$ in A_1 which recognizes the string xy_1z. Additionally, the path $\mathcal{P}_{A_2} = s_2, \ldots, p_{y_1}, \ldots, p_{\bar{y}}$ must exist in A_2 recognizing the string $y_1\bar{y}$.

If we consider the paths $\mathcal{P}_{A_1}$ and $\mathcal{P}_{A_2}$ when reading the word $xy_1\bar{y}z$ into A_{PreDI}, we get the following accepting path. From the start state (s_1, s_2), we are able to read x using transitions from (i) to arrive at (q_x, s_2). When reading y_1, we use the transitions defined in (i) to arrive at the state $(\bar{q_{y_1}}, p_{y_1})$. We are now restricted to transitions

defined in (ii) and (iii). Using the transitions defined in (ii), we arrive at the state $(q_{y_1}^-, p_{\bar{y}})$, followed by the transitions defined in (iii) to arrive in the state $(q_{y_1}, \clubsuit)$. Finally, we conclude the computation by reading z, using the transition defined in (iv) to arrive at the final state $(q_z, \clubsuit)$. Thus, any word in $L(A_1) \xleftarrow{PreDI} L(A_2)$ must also be in $L(A_{PreDI})$.

Secondly we show the inclusion$L(A_{PreDI}) \subseteq L(A_1) \xleftarrow{PreDI} L(A_2)$:

We need to show that a word recognized by A_{PreDI} is in $L(A_1) \xleftarrow{PreDI} L(A_2)$. We can take an accepting path $\mathcal{P}_{A_{PreDI}}$ and decompose it into four components:

$$\mathcal{P}_1 = (s_1, s_2), \ldots, (q_x, s_2), \mathcal{P}_2 = (q_x, s_2), \ldots, (q_{y_1}^-, p_{y_1})$$

$$\mathcal{P}_3 = (q_{y_1}^-, p_{y_1}), \ldots, (q_{y_1}^-, p_{\bar{y}}), \text{ and } \mathcal{P}_4 = (q_{y_1}, \clubsuit), \ldots, (q_z, \clubsuit)$$

If we first consider the path $\mathcal{P}_1$, we see that a path $s_1, \ldots, q_x$ exists in $\mathcal{P}_{A_1}$ recognizing the prefix x. Path $\mathcal{P}_2$ implies that paths $q_x, \ldots, q_{y_1}$ in $\mathcal{P}_{A_1}$ and $s_2, \ldots, p_{y_1}$ in $\mathcal{P}_{A_2}$ both exist. These paths recognize the sub-string y_1. The path $\mathcal{P}_3$ implies that the path $p_{y_1}, \ldots, p_{\bar{y}}$ exists in $\mathcal{P}_{A_2}$ and recognizes the string $\bar{y}$. Finally, the path $\mathcal{P}_4$ implies the existence of a path $q_{y_1}, \ldots, q_z$ recognizing string z. The simulation concludes in the final state $(q_z, \clubsuit)$. It is worth noting that the paths $\mathcal{P}_1, \mathcal{P}_2$ and $\mathcal{P}_4$ may contain a single element, implying that the sub-strings $x, \bar{y}$ and z may each be empty. The existence of these paths also implies that, if A_{PreDI} recognizes a string $xy_1\bar{y}z$, then there exist accepting paths on A_1 and A_2 that recognize the strings xy_1z and $y_1\bar{y}$, respectively. □

Figure 4.1 illustrates the construction of an automaton A_{PreDI} that recognizes $L_1 \xleftarrow{PreDI} L_2$. In the figure, the boxes encapsulate different transition rules. The box

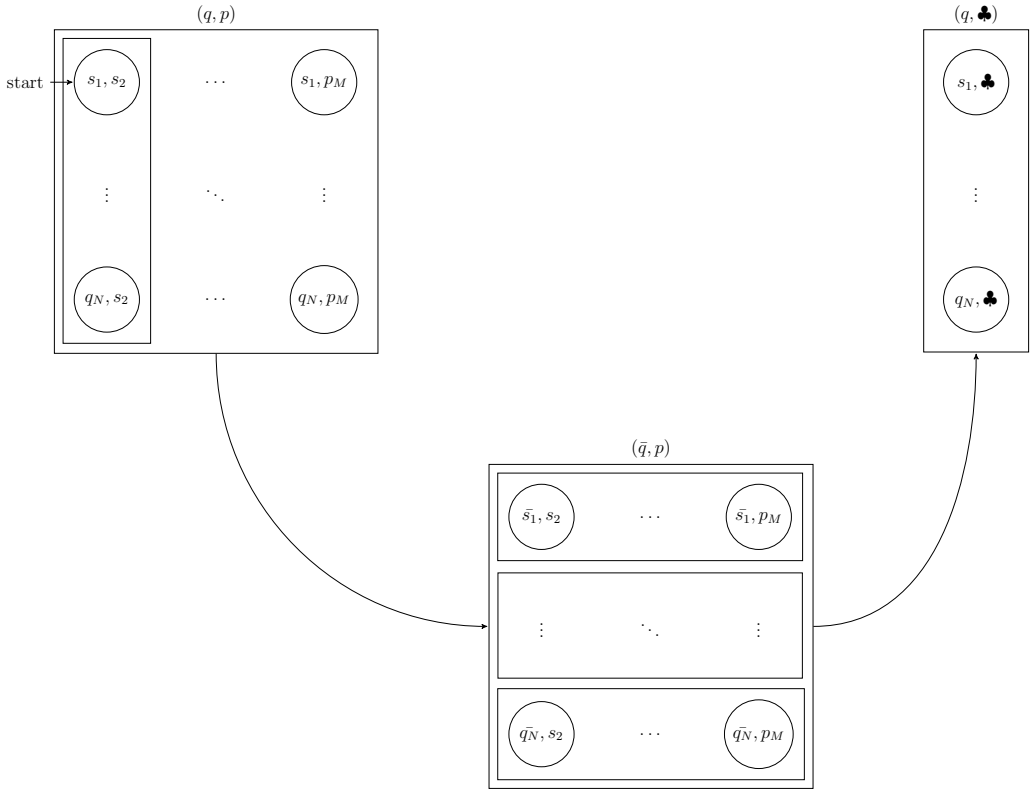

Figure 4.1: The construction of the prefix-directed insertion automaton.

labeled with (q, p) contains the states with outgoing transitions defined in (i). The rectangle contained inside the (q, p) box encapsulates the states responsible for reading the prefix x. The box labeled with $(\bar{q}, p)$ contains states with outgoing transitions defined in (ii) and (iii) and these states read the substring $\bar{y}$. The horizontal rectangles signify that transitions between the rectangles are not possible. The final set of states, in the rectangle labeled with $(q, \clubsuit)$ are the states with outgoing transitions defined in (iv) and which read the string z. An example of the construction for the prefix-directed insertion automaton is contained in Figure 4.2.

We want to provide a more useful bound on nondeterministic state complexity.

To this end, we provide a lower bound on state complexity by showing a case where many of the states in the construction are necessary.

Theorem 10. *Let M, $N \in \mathbb{N}$. There exists regular languages L_1 and L_2 over a four-letter alphabet recognized by NFAs that have N and M states, respectively, such that any NFA for $L_1 \xleftarrow{PreDI} L_2$ needs at least $2NM + N - M$ states.*

Proof. We take the following languages to construct our witness language:

$$L_1 = \{w \mid w \in \{\text{⚓}_1, a, b\}^* \text{ and } (\#_{\text{⚓}_1}(w) + \#_a(w)) \equiv 0 \mod N\} \text{ and}$$

$$L_2 = \{w \mid w \in \{\text{⚓}_2, a, b\}^* \text{ and } (\#_{\text{⚓}_2}(w) + \#_b(w)) \equiv 0 \mod M\}.$$

The language L_1 recognizes strings that contain a multiple of N total ⚓_1 and a symbols and does not contain any ⚓_2 symbols. Similarly, the language L_2 recognizes strings that contain a multiple of M total ⚓_2 and b symbols and does not contain any ⚓_1 symbols. L_1 and L_2 can be recognized with NFAs with N and M states respectively. Using L_1 and L_2 we can produce the following witness language:

$$L_1 \xleftarrow{PreDI} L_2 = \{xy_1\bar{y}z \mid \#_{\text{⚓}_1}(xy_1z) + \#_a(xy_1z) \equiv 0 \mod N \text{ and}$$

$$\#_{\text{⚓}_2}(y_1\bar{y}) + \#_b(y_1\bar{y}) \equiv 0 \mod M, \#_{\text{⚓}_1}(y_1\bar{y}) = 0, \#_{\text{⚓}_2}(xy_1z) = 0, y_1 \neq \varepsilon\}.$$

We can construct a fooling set $FS = FS_1 \cup FS_2 \cup FS_3$ that contains 2NM+N-M elements in the following way.

The first set FS_1 contains NM elements and is defined as follows:

$$FS_1 = \{(a^{N-j}\text{⚓}_2^{M}\text{⚓}_2^{M-i}, \text{⚓}_2^{i}a^j) \mid 0 \leq i < M, 0 \leq j < N\}.$$

We can show that each tuple, when concatenated, forms a word in $L_1 \xleftarrow{PreDI} L_2$. If we take all words of the form $a^{N-j}\text{⚓}_2^{M}\text{⚓}_2^{M-i}\text{⚓}_2^{i}a^j$, we can parse each word as per Definition 13 as follows: $y_1 = a^{N-j}$, $\bar{y} = \text{⚓}_2^{M}\text{⚓}_2^{M-i}\text{⚓}_2^{i}$ and $z = a^j$. The substring y_1 is never empty, since $0 \leq j < N$, and the string $y_1 z$ always contains N a's. The substring $\bar{y}$ always contains 2M ⚓$_2$'s, which is consistent with the witness language.

We need to show that if we take $(x, y), (x', y') \in FS_1$ where $(x, y) \neq (x', y')$, then either $x'y$ or xy' are not in the witness language. If we assume $i < i'$, then without loss of generality we can make a word $x'y = a^{N-j}\text{⚓}_2^{M}\text{⚓}_2^{M-i'}\text{⚓}_2^{i}a^j$, which will contain $2M + i - i'$ ⚓$_2$'s, indicating that the word does not meet the language restrictions. Similarly, if we assume $j < j'$, we can make the word $x'y = a^{N-j'}\text{⚓}_2^{M}\text{⚓}_2^{M-i}\text{⚓}_2^{i}a^j$ which will contain $N + j - j'$ a's.

The next set we consider contains $(N-1)M$ elements and is defined as follows:

$$FS_2 = \{(a^{N-j}b^{M-i}, a^j\text{⚓}_2^{i}\text{⚓}_2^{M}) \mid 0 < i \leq M, 0 < j < N\}$$

.

We can decompose the words made in FS_2 similarly to the ones in FS_1. If we take the words $a^{N-j}b^{M-i}a^j\text{⚓}_2^{i}\text{⚓}_2^{M}$, where $0 < i \leq M, 0 < j \leq N$, we can set $y_1 = a^{N-j}b^{M-i}a^j$ and $\bar{y} = \text{⚓}_2^{i}\text{⚓}_2^{M}$. y_1 always contains N $a's$, and $y_1\bar{y}$ always contains $2M$ b's and ⚓$_2$'s.

For each pair $(x, y) \in FS_2$ and $(x', y') \in FS_2$ where either $x \neq x'$ or $y \neq y'$, the word $x'y$ is not in the language $L_1 \xleftarrow{PreDI} L_2$. If $j < j'$, then the word $a^{N-j'}b^{M-i}a^j\text{⚓}_2^{i}$ does not contain enough $a's$ to meet the criteria from L_1. If we take $i < i'$, then $a^{N-j}b^{M-i'}a^j\text{⚓}_2^{i}$ does not contain enough ⚓$_2$ symbols to account for the missing b's.

We need to show that the fooling set properties are maintained between FS_1 and

FS_2. To this end, if we take $(x, y) \in FS_2$ and $(x', y') \in FS_1$, then the word $x'y$ is not in the witness language when $0 < j' < N$ and xy' is not in the witness language when $j' = 0$.

The word $x'y = a^{N-j'}\text{⚓}_2^M\text{⚓}_2^{M-i'}a^j\text{⚓}_2^i\text{⚓}_2^M$ is not in the witness language when $0 < j' < N$ because a^j cannot be counted towards the mod N requirements of L_1, since it must be in $\bar{y}$. Thus, the count of a's is always greater than 0 but less than N. When $j' = 0$, the word $xy' = a^{N-j}b^{M-i}\text{⚓}_2^{i'}$ is never in the witness language since $0 < j < N$. This also will not meet the L_1 language requirement.

The last subset of the fooling set contains N elements and is defined as follows:

$$FS_3 = \{(a^N\text{⚓}_2^M\text{⚓}_1^{N-j}, \text{⚓}_1^j) \mid 0 \leq j < N\}.$$

We can decompose each word created by the fooling set as follows: $y_1 = a^N, \bar{y} = \text{⚓}_2^M$ and $z = \text{⚓}_1^{N-j}\text{⚓}_1^j$. In this way, we can see $y_1\bar{y}$ always has M ⚓$_2$ symbols and y_1z contains N total a's and ⚓$_1$'s.

We need to show that the pairs $(x, y) \in FS_3$ maintain the fooling set properties with $(x', y') \in FS_3$ where $(x, y) \neq (x', y')$. If we take the word $x'y$ where $j < j'$, then each word $a^N\text{⚓}_2^M\text{⚓}_1^{N-j'}\text{⚓}_1^j$ is never in our witness language, since there are $N+j-j'$ total a's and ⚓$_1$ symbols.

If we next consider $(x, y) \in FS_3$ and $(x', y') \in FS_1$, we can show that xy' is not in the witness language. The word is defined as $a^N\text{⚓}_2^M\text{⚓}_1^{N-j}\text{⚓}_2^i a^j$ and this word cannot be in the witness language, since all ⚓$_2$ symbols can only be in the $\bar{y}$ sub-word. No ⚓$_1$ symbols are permitted in $\bar{y}$ since it is not a character in L_2. Thus, the fooling set property is maintained between FS_1 and FS_3.

Lastly if we consider $(x, y) \in FS_3$ and $(x', y') \in FS_2$, we can show that xy' is

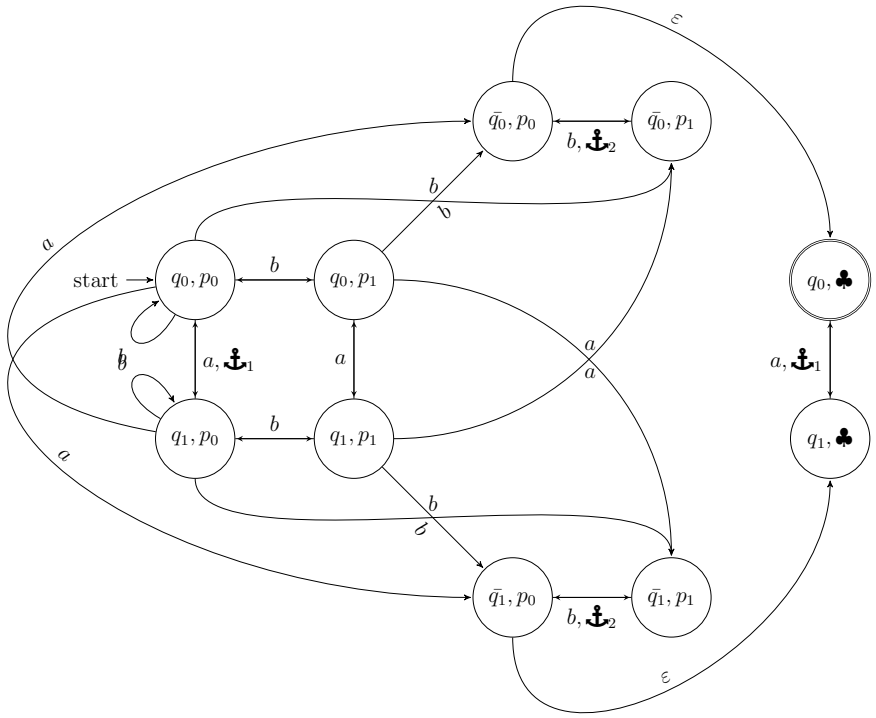

Figure 4.2: A diagram representing the automaton A_{PreDI} when $L_1 = \{w \mid w \in \{\text{⚓}_1, a, b\}^*$ and $(\#_{\text{⚓}_1}(w) + \#_a(w)) \equiv 0 \mod 2\}$ and $L_2 = \{w \mid w \in \{\text{⚓}_2, a, b\}^*$ and $(\#_{\text{⚓}_2}(w) + \#_b(w)) \equiv 0 \mod 2\}$.

not in $L_1 \overleftarrow{\ \ }^{PreDI} L_2$. The same argument used for the elements from FS_1 applies to the elements of FS_2. The word $xy' = a^N \text{⚓}_2^M \text{⚓}_1^{N-j} a^j \text{⚓}_2^i$, always has the symbol ⚓_1 appearing between two ⚓_2 symbols.

□

The construction of an ε-NFA containing $2NM + N$ states acts as an upper bound on nondeterministic state complexity, since by Proposition 1 there must exist an equivalent NFA with the same number of states. This upper bound differs by a

factor of M from the lower bound of $2NM + N - M$. The problem of closing this difference remains open. The similarity of the bounds is promising and suggests that a tight bound could yet be found.

4.2 Suffix-Directed Insertion

The similarities between the prefix and suffix operations are mirrored in prefix-directed and suffix-directed insertion. The construction of NFAs recognizing suffix-directed and prefix-directed insertion operations are similar and result in the same state complexity. Although suffix-directed insertion has some state savings properties associated with collapsing the final states.

Definition 14 (Suffix-Directed Insertion)**.** *For languages L_1 and L_2, the suffix directed insertion language of L_2 into L_1 is defined as follows:*

$$L_1 \xleftarrow{SufDI} L_2 = \{x\bar{y}y_2z \mid xy_2z \in L_1, \bar{y}y_2 \in L_2, y_2 \neq \varepsilon\}$$

For regular languages specified by NFAs A_1 and A_2, we construct an ε-NFA for the suffix-directed insertion of $L(A_2)$ into $L(A_1)$. The construction gives an upper bound for the nondeterministic state complexity of suffix-guided insertion.

Theorem 11. *For NFAs A_1 and A_2 with N and M states, respectively, the language $L(A_1) \xleftarrow{SufDI} L(A_2)$ is recognized by an ε-NFA containing $2NM + N$ states.*

Proof. Let $A_1 = (Q, \Sigma, \delta, s_1, F_1)$ and $A_2 = (P, \Sigma, \gamma, s_2, F_2)$ be automata recognizing languages L_1 and L_2 with N and M states, respectively. Given these NFAs, we can

construct an NFA $A_{SufDI} = (Q_{SufDI}, \Sigma, \Omega, (s_1, \clubsuit), F_{SufDI})$, where

$$Q_{SufDI} = \{(Q \times \{\clubsuit\})\} \cup \{(\bar{Q} \times P)\} \cup \{(Q \times P \backslash F_2)\} \cup \{(Q \times \{p_f\})\}.$$

The states $\bar{q} \in \bar{Q}$ have corresponding states $q \in Q$ although Q and $\bar{Q}$ are disjoint sets. The final states of A_{SufDI} are states (q_i, p_f) where $q_i \in F_1$. The state p_f is used to collapse all final states from F_2 into one. The transitions of Ω are defined as follows:

(i) for $(q, \clubsuit)$ where $q \in Q$:

$$\Omega((q, \clubsuit), \alpha) = \{(q', \clubsuit) \mid q' \in \delta(q, \alpha)\}$$

(ii) for $(q, \clubsuit)$ where $q \in Q$:

$$\Omega((q, \clubsuit), \varepsilon) = \{(q, s_2)\}$$

(iii) for $(\bar{q}, p)$ where $q \in Q$ and $p \in P$:

$$\begin{aligned}\Omega((\bar{q}, p), \alpha) = {} & \{(\bar{q}, p') \mid p' \in \gamma(p, \alpha)\} \\ & \cup \{(q', p') \mid q' \in \delta(q, \alpha) \text{ and } p' \in \gamma(p, \alpha)\} \\ & \cup \{(q', p_f) \mid q' \in \delta(q, \alpha) \text{ and } \gamma(p, \alpha) \cap F_2 \neq \emptyset\}\end{aligned}$$

(iv) for (q, p) where $q \in Q$ and $p \in P$:

$$\begin{aligned}\Omega((q, p), \epsilon) = {} & \{(q', p') \mid q' \in \delta(q, \alpha) \text{ and } p' \in \gamma(p, \alpha)\} \\ & \cup \{(q', p_f) \mid q' \in \delta(q, \alpha) \text{ and } \gamma(p, \alpha) \cap F_2 \neq \emptyset\}\end{aligned}$$

(v) for (q, p_f) where $q \in Q$:

$$\Omega((q, p_f), \alpha) = \{(q', p_f) \mid q' \in \delta(q, \alpha)\}$$

We first show the inclusion $L(A_1) \xleftarrow{SufDI} L(A_2) \subseteq L(A_{SufDI})$:

Generally speaking, A_{SufDI} uses the states $(q, \clubsuit)$ to compute the substring x, the states $(\bar{q}, p)$ to read the inserted string $\bar{y}$ and the states (q, p) to read the sub-string y_2. Finally, the states (q, p_f) are used to complete the computation, reading z.

If we take an arbitrary word $x\bar{y}y_2z$ from $L_1 \xleftarrow{SufDI} L_2$, the existence of this word implies that a path $\mathcal{P}_{A_1} = s_1, \ldots, q_x, \ldots, q_{y_2}, \ldots, q_z$ exists in A_1 which recognizes xy_2z and a path $\mathcal{P}_{A_2} = s_2, \ldots, p_{\bar{y}}, \ldots, p_{y_2}, \ldots, p_z$ exists in A_2 which recognizes $\bar{y}y_2z$.

We can construct an accepting path through A_{SufDI} to compute the word $x\bar{y}y_2z$. We start at $(s_1, \clubsuit)$ and, using transitions defined in (i) to read x, arrive at the state $(q_x, \clubsuit)$. Next, we use the ε transition in (ii) to transition to the state $(\bar{q_x}, s_2)$. Using the transitions defined in (iii), we read $\bar{y}$ to arrive at $(\bar{q_x}, p_{\bar{y}})$. To read y_2, we use the transition defined in (iii) to arrive at the state $(q'_x, p'_{\bar{y}})$ and then use the transitions defined in (iv) to arrive at (q_{y_2}, p_f). Finally, we use the transitions defined in (v) to read x and arrive at the final state (q_x, p_f).

Secondly we show the inclusion $L(A_{SufDI}) \subseteq L(A_1) \xleftarrow{SufDI} L(A_2)$:

To show the inclusion $L(A_{SufDI}) \subseteq L(A_1) \xleftarrow{SufDI} L(A_2)$, we need to show that, if a word is recognized by A_{SufDI}, then the word is in $L(A_1) \xleftarrow{SufDI} L(A_2)$. If we take a word $x\bar{y}y_2z$ with an accepting path $\mathcal{P}_{SufDI} = (s_1, \clubsuit), \ldots, (q_x, \clubsuit), (\bar{q_x}, s_2), \ldots, (\bar{q_x}, p_{\bar{y}}), \ldots, (q_{y_2}, p_f), \ldots, (q_z, p_f)$, we can decompose this path into four components recognizing the different sub-words:

$$\mathcal{P}_1 = (s_1, \clubsuit), \ldots, (q_x, \clubsuit), \mathcal{P}_2 = (\bar{q_x}, s_2), \ldots, (\bar{q_x}, p_{\bar{y}})$$

$$\mathcal{P}_3 = (\bar{q_x}, p_{\bar{y}}), \ldots, (q_{y_2}, F), \text{ and } \mathcal{P}_4 = (q_{y_2}, F), \ldots, (q_z, F)$$

Path $\mathcal{P}_1$ implies that a path $s_1, \ldots, q_x$ exists in $\mathcal{P}_{A_1}$ recognizing the prefix x. Path $\mathcal{P}_2$ implies that the path $s_2, \ldots, p_{\bar{y}}$ exists in $\mathcal{P}_{A_2}$ recognizing $\bar{y}$. The paths before path $\mathcal{P}_3$ could each read ε to reach $\mathcal{P}_3$. Since $y_2 \neq \varepsilon$ path $\mathcal{P}_3$ must read a non-empty word, implying that paths $q_x, \ldots, q_{y_2}$ and $p_{\bar{y}}, \ldots, p_{y_2}$ exist in $\mathcal{P}_{A_1}$ and $\mathcal{P}_{A_2}$, respectively. Finally, the path $\mathcal{P}_4$ implies that the path $q_{y_2}, \ldots, q_z$ exists in A_1 and the simulation concludes in the state (q_z, p_f). This implies that $\bar{y}y_2 \in L(A_2)$ and $xy_2z \in L(A_1)$ and since $y_2 \neq \varepsilon$, we satisfy the criteria for a word to be in $L(A_1) \xleftarrow{SufDI} L(A_2)$. □

Figure 4.3 illustrates the construction of an automaton A_{SufDI} that recognizes $L_1 \xleftarrow{SufDI} L_2$. In the figure, the boxes encapsulate different transition rules. The box labeled with $(q, \clubsuit)$ contains states with outgoing transitions defined in (i), these states are responsible for reading the prefix x. The rectangles contained inside the $(\bar{q}, p)$ box encapsulates the states responsible for reading the substring $\bar{y}$, using the transitions defined in (iii). The horizontal rectangles signify that transitions between the rectangles are not possible. The final set of states on the right in the rectangle labeled with (q, p) are the states with out transitions defined in (iv) and (v). These states read the substring y_2 and z.

To provide more insight into the nondeterministic state complexity of suffix directed insertion, a lower bound is helpful. The proof for a lower bound for suffix-directed insertion is the same as the lower bound proof for prefix-directed insertion,

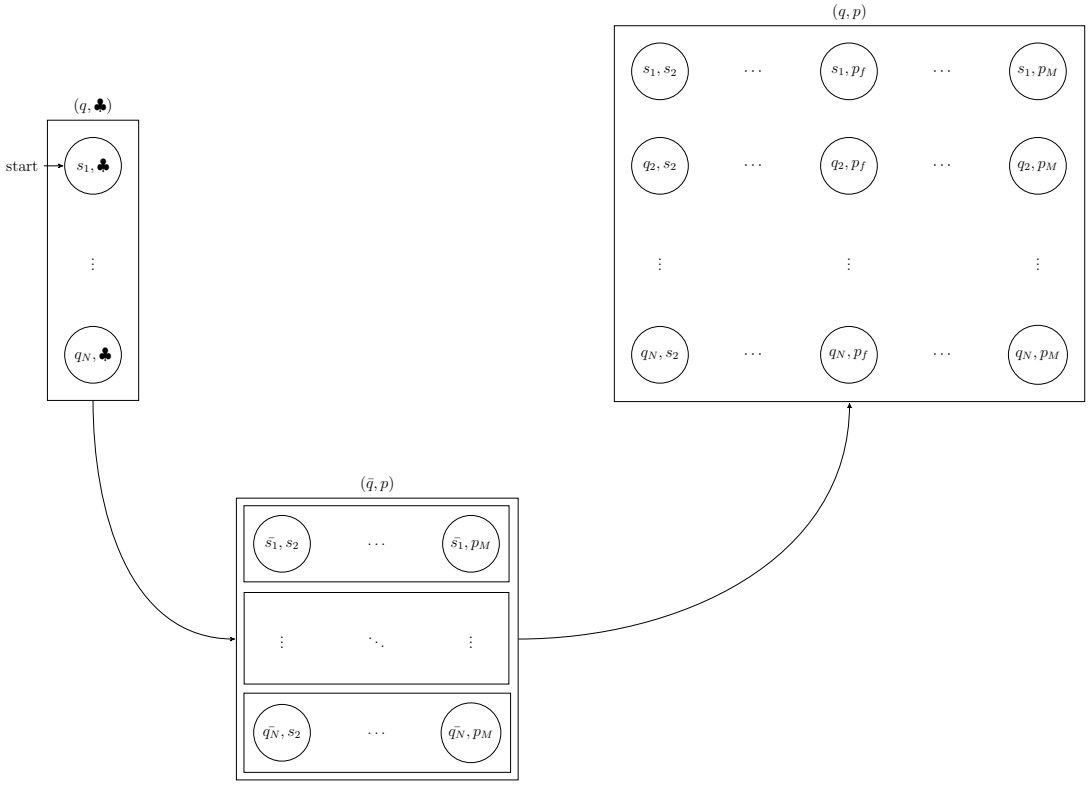

Figure 4.3: The construction of the suffix-directed insertion automaton.

except all the words are reversed.

Theorem 12. *Let M, $N \in \mathbb{N}$. There exists regular languages L_1 and L_2 over a four-letter alphabet recognized by NFAs that have N and M states, respectively, such that any NFA for $L_1 \xleftarrow{SufDI} L_2$ needs at least $2NM + N - M$ states.*

Proof. We take the following languages to construct our witness language:

$$L_1 = \{w \mid w \in \{\text{⚓}_1, a, b\}^* \text{ and } (\#_{\text{⚓}_1}(w) + \#_a(w)) \mod N \equiv 0\} \text{ and}$$

$$L_2 = \{w \mid w \in \{\text{⚓}_2, a, b\}^* \text{ and } (\#_{\text{⚓}_2}(w) + \#_b(w)) \mod M \equiv 0\}.$$

The language L_1 recognizes strings that contain a multiple of N total ⚓$_1$ and a

symbols and does not contain any ⚓_2 symbols. Similarly, the language L_2 recognizes strings that contain a multiple of M total ⚓_2 and b symbols and does not contain any ⚓_1 symbols. L_1 and L_2 can be recognized with NFAs with N and M states respectively. Using L_1 and L_2 we can produce the following witness language:

$$L_1 \xleftarrow{SufDI} L_2 = \{x\bar{y}y_2z \mid \#_{\text{⚓}_1}(xy_2z) + \#_a(xy_2z) \mod N \equiv 0 \text{ and}$$

$$\#_{\text{⚓}_2}(\bar{y}y_2) + \#_b(\bar{y}y_2) \mod M \equiv 0, \#_{\text{⚓}_1}(\bar{y}y_2) = 0, \#_{\text{⚓}_2}(xy_2z) = 0, y_2 \neq \varepsilon\}.$$

We can construct a fooling set $FS = FS_1 \cup FS_2 \cup FS_3$ that contains 2NM+N-M elements in the following way.

The first set FS_1 contains NM elements and is defined as follows:

$$FS_1 = \{(a^j \text{⚓}_2^M \text{⚓}_2^i, \text{⚓}_2^{M-i} \text{⚓}_2^M a^{N-j}) \mid 0 \leq i < M, 0 \leq j < N\}$$

The next set we consider contains (N-1)M elements and is defined as follows:

$$FS_2 = \{(\text{⚓}_2^M \text{⚓}_2^i a^j, b^{M-i}a^{N-j}) \mid 0 < i \leq M, 0 < j < N\}.$$

The last subset of the fooling set contains N elements and is defined as follows:

$$FS_3 = \{(\text{⚓}_1^j, \text{⚓}_1^{N-j} \text{⚓}_2^M a^N) \mid 0 \leq j < N\}.$$

The extended proof verifying that FS is a fooling set is contained in Appendix A. □

As expected we achieve the same bounds on nondeterministic state complexity for

suffix-directed insertion as for prefix-directed insertion. The upper bound on state complexity proved in Theorem 11 is $2NM + N$. The accompanying lower bound of $2NM + N - M$ was proved in Theorem 12.

4.3 Site-Directed Insertion

The site-directed insertion operation inserts a substring into the host string by identifying a substring in the host string that matches the outfix of the inserted string. We restate the definition of site-directed insertion for convenience.

Definition 15 (Site-Directed Insertion)**.** *For languages L_1 and L_2, the site-directed insertion language of L_2 into L_1 is defined as follows:*

$$L_1 \xleftarrow{SDI} L_2 = \{xy_1\bar{y}y_2z \mid xy_1y_2z \in L_1, y_1\bar{y}y_2 \in L_2, y_1 \neq \epsilon, y_2 \neq \epsilon\}$$

This definition can be extended to operate over NFAs A_1 and A_2 that recognize languages L_1 and L_2, respectively.

Theorem 13. *For NFAs A_1 and A_2 with N and M states, respectively, the language $L(A_1) \xleftarrow{SDI} L(A_2)$ is recognized by an NFA containing $3NM$ states.*

Proof. If we take the NFAs $A_1 = (Q, \Sigma, \delta, s_1, F_1)$ and $A_2 = (P, \Sigma, \gamma, s_2, F_2)$ recognizing languages L_1 and L_2, respectively, then we can construct an NFA $A_{SDI} = (Q_{SDI}, \Sigma, \Omega, (s_1, s_2)_{\mathcal{X}_1}, F_{SDI})$, to recognize $L_1 \xleftarrow{SDI} L_2$.

The states contained within Q_{SDI} are defined as follows:

$$Q_{SDI} = (Q \times P)_{\mathcal{X}_1} \cup (Q \times P)_{\mathcal{X}_2} \cup (Q \times \{P \backslash F_2\})_{\mathcal{X}_3} \cup (Q \times \{p_f\})_{\mathcal{X}_3}$$

Q_{SDI} contains up to 3 copies of $(Q \times P)$ with different subscripts to differentiate the states, the exception being $(q,p)_{\mathcal{X}_3}$ which may not contain as many states. The states $(q,p_f)_{\mathcal{X}_3}$ are present to replace final states $p \in F_2$ in the p components of the sates $(Q \times P)_{\mathcal{X}_3}$. The final states contained in F_{SDI} are states $(q,p_f)_{\mathcal{X}_3}$ where $q \in F_1$.

The transition function Ω is defined as follows:

(i) for $(q,s_2)_{\mathcal{X}_1}$ where $q \in Q$:

$$\begin{aligned}\Omega((q,s_2)_{\mathcal{X}_1},\alpha) = &\{(q',s_2)_{\mathcal{X}_1} \mid q' \in \delta(q,\alpha)\} \\ &\cup \{(q',p')_{\mathcal{X}_1} \mid q' \in \delta(q,\alpha) \text{ and } p' \in \gamma(s_2,\alpha)\} \\ &\cup \{(q',p')_{\mathcal{X}_2} \mid q' \in \delta(q,\alpha) \text{ and } p' \in \gamma(s_2,\alpha)\}\end{aligned}$$

(ii) for $(q,p)_{\mathcal{X}_1}$ where $q \in Q$ and $p \in P \backslash \{s_2\}$:

$$\begin{aligned}\Omega((q,p)_{\mathcal{X}_1},\alpha) = &\{(q',p')_{\mathcal{X}_1} \mid q' \in \delta(q,\alpha) \text{ and } p' \in \gamma(p,\alpha)\} \\ &\cup \{(q',p')_{\mathcal{X}_2} \mid q' \in \delta(q,\alpha) \text{ and } p' \in \gamma(p,\alpha)\}\end{aligned}$$

(iii) for $(q,p)_{\mathcal{X}_2}$ where $q \in Q$ and $p \in P$:

$$\begin{aligned}\Omega((q,p)_{\mathcal{X}_2},\alpha) = &\{(q,p')_{\mathcal{X}_2} \mid p' \in \gamma(p,\alpha)\} \\ &\cup \{(q',p')_{\mathcal{X}_3} \mid q' \in \delta(q,\alpha) \text{ and } p' \in \gamma(p,\alpha) \text{ and } p' \notin F_2\} \\ &\cup \{(q',p_f)_{\mathcal{X}_3} \mid q' \in \delta(q,\alpha) \text{ and } \gamma(p,\alpha) \cap F_2 \neq \emptyset\}\end{aligned}$$

(iv) for $(q,p)_{\mathcal{X}_3}$ where $q \in Q$ and $p \in P$:

$$\Omega((q,p)_{\mathcal{X}_3},\alpha) = \{(q',p')_{\mathcal{X}_3} \mid q' \in \delta(q,\alpha) \text{ and } p' \in \gamma(p,\alpha) \text{ and } p' \notin F_2\}$$
$$\cup \{(q',p_f)_{\mathcal{X}_3} \mid q' \in \delta(q,\alpha) \text{ and } \gamma(p,\alpha) \cap F_2 \neq \emptyset\}$$

(v) for $(q,p_f)_{\mathcal{X}_3}$ where $q \in Q$:

$$\Omega((q,p_f)_{\mathcal{X}_3},\alpha) = \{(q',p_f)_{\mathcal{X}_3} \mid q' \in \delta(q,\alpha)\}$$

First we show the inclusion $L(A_1) \xleftarrow{SDI} L(A_2) \subseteq L(A_{SDI})$.

Generally speaking, the states $(q,s_2)_{\mathcal{X}_1}$ are used to read the prefix x. Next, the states $(q,p)_{\mathcal{X}_1}$ are used to read the substring y_1 and the states $(q,p)_{\mathcal{X}_2}$ are used to read the substring $\bar{y}$. The states $(q,p)_{\mathcal{X}_3}$ read the substring y_2 and, finally, the states $(q,p_f)_{\mathcal{X}_3}$ are used to complete the computation and read z.

If we take a word $xy_1\bar{y}y_2z$ from the language $L(A_1) \xleftarrow{SDI} L(A_2)$, then there must exist an accepting path $\mathcal{P}_{A_1} = s_1,\ldots,q_x,\ldots,q_{y_1},\ldots,q_{y_2},\ldots,q_z$ in A_1 recognizing the string xy_1y_2z. Additionally, an accepting path $\mathcal{P}_{A_2} = s_2,\ldots,p_{y_1},\ldots,p_{\bar{y}},\ldots,p_{y_2}$ must exist in A_2 recognizing the string $y_1\bar{y}y_2$.

If we follow the computation of $xy_1\bar{y}y_2z$ in A_{SDI}, we get the following accepting path. The computation starts in the state $(s_1,s_2)_{\mathcal{X}_1}$, reads x and uses the transitions described in (i) to arrive at the state $(q_x,s_2)_{\mathcal{X}_1}$. To read y_1, we use the transitions defined in (ii) to arrive at the state $(q_{y_1},p_{y_1})_{\mathcal{X}_2}$. The next state we arrive at is $(q_{y_1},p_{\bar{y}})_{\mathcal{X}_3}$, reached by using the transitions defined in (iii) to read $\bar{y}$. To read y_2, we use the transitions defined in (iv), which brings us to the state $(q_{y_2},p_f)_{\mathcal{X}_3}$. Finally, to read z, we use the transitions defined in (v) and arrive at the final state (q_z,p_f).

Secondly we show the inclusion $L(A_{SDI}) \subseteq L(A_1) \overleftarrow{}^{SDI} L(A_2)$:

To show the inclusion $L(A_{SDI}) \subseteq L(A_1) \overleftarrow{}^{SDI} L(A_2)$ we consider a word $xy_1\bar{y}y_2z$ recognized by A_{SDI}. The accepting path of computation of $xy_1\bar{y}y_2z$ can be decomposed into five sub-paths, which are defined as follows:

$$\mathcal{P}_1 = (s_1, s_2)_{\mathcal{X}_1}, \ldots, (q_x, s_2)_{\mathcal{X}_1}, \mathcal{P}_2 = (q_x, s_2)_{\mathcal{X}_1}, \ldots, (q_{y_1}, p_{y_1})_{\mathcal{X}_2}$$

$$\mathcal{P}_3 = (q_{y_1}, p_{y_1})_{\mathcal{X}_2}, \ldots, (q_{y_1}, p_{\bar{y}})_{\mathcal{X}_2}, \mathcal{P}_4 = (q_{y_1}, p_{\bar{y}})_{\mathcal{X}_2}, \ldots, (q_{y_2}, p_f)_{\mathcal{X}_3}$$

$$\mathcal{P}_5 = (q_{y_2}, p_f)_{\mathcal{X}_3}, \ldots, (q_z, p_f)_{\mathcal{X}_3}$$

The path $\mathcal{P}_1$ is chosen to be as long as possible, such that the path $\mathcal{P}_2$ starts at a state $(q, s_2)_{\mathcal{X}_1}$ such that no other state $(q', s_2)_{\mathcal{X}_1}$ appears in the path. The path $\mathcal{P}_1$ implied that there exists a path $s_1, \ldots, q_x$ in A_1 recognizing the prefix string x. The path $\mathcal{P}_2$ implies the existence of paths $q_x, \ldots, q_{y_1}$ in A_1 and $s_2, \ldots, p_{y_1}$ in A_2, and these paths each recognize the sub-string y_1. The path $\mathcal{P}_3$ implies that the path $p_{y_1}, \ldots, p_{\bar{y}}$ exists in A_2 recognizing $\bar{y}$. The path $\mathcal{P}_4$ implies the paths $p_{\bar{y}}, \ldots, p_{y_2}$ exists in A_2 and $q_{y_1}, \ldots, q_{y_2}$ exists in A_1, and each of these paths recognize the substring y_2. Finally, the path $\mathcal{P}_5$ implies the existence of a path $q_{y_2}, \ldots, q_z$ in A_1 recognizing the string z. Since q_z is in F_1 and p_{y_2} is in F_2, the automaton A_1 recognizes the word xy_1y_2z and A_2 recognizes $y_1\bar{y}y_2$. □

Figure 4.4 depicts a diagram of A_{SDI}. The different boxes encapsulate states with different out going transitions. The box labeled by $\mathcal{X}_1$ contains the states with outgoing transitions defined in (i) and (ii). The center box contains states with outgoing transitions defined in (iii). The rightmost box labeled by $\mathcal{X}_3$ contains states with outgoing transitions defined in (iv), and the rectangle to the right of this box contains

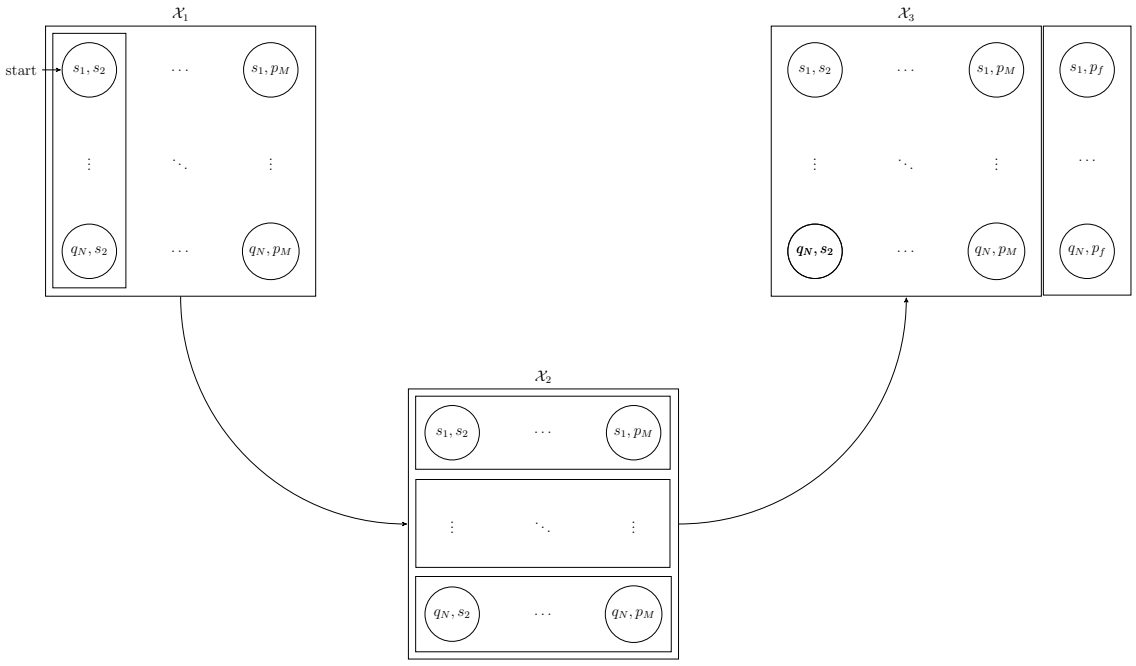

Figure 4.4: The construction of the site-directed insertion automaton.

the state with outgoing transitions defined in (v). An example of the construction for the site-directed deletion automaton is contained in Figure 4.6

Theorem 14. *Let M, $N \in \mathbb{N}$. There exist regular languages L_1, L_2 over an alphabet that depend on N and M and is recognized by NFAs with M and N states, respectively, such that any NFA for the language $L_1 \xleftarrow{SDI} L_2$ needs at least $3NM - M$ states.*

Proof. In order to define the languages L_1 and L_2, we must first define some notation for different alphabets. The alphabet Σ_a contains symbols $\{a_0, a_1, \ldots, a_{N-1}\}$ and, likewise, the alphabet Σ_b contains symbols $\{b_0, b_1, \ldots, b_{M-1}\}$. Σ_c contains the symbols $\{c_{i,j} \mid 0 \leq i < M, 0 \leq j < N\}$. The full alphabet for L_1 and L_2 is defined as follows:

$$\Sigma = \Sigma_a \cup \Sigma_b \cup \Sigma_c \cup \{\text{⚓}_2\}$$

The languages L_1 and L_2 are defined as follows:

$$L_1 = \bigcup_{\substack{0<j<N,\\ 0\leq i,i'<M}} \Big((\Sigma_b \mid a_0)^* \big((a_j \mid c_{i,j}) \Sigma_b^* (a_j \mid c_{i',j}) \big)^* \Big)^*;$$

$$L_2 = \bigcup_{\substack{0\leq j,j'<N,\\ 0<i<M}} \Big((\text{⚓}_2 \mid \Sigma_a \mid b_0)^* \big((b_i \mid c_{i,j}) (\text{⚓}_2 \mid \Sigma_a)^* (b_i \mid c_{i,j'}) \big)^* \Big)^*.$$

L_1 consists of words containing symbols a_j, b_i and $c_{i,j}$. We examine patterns in the subscripts of a_j's and $c_{i,j}$'s to determine membership. If we read a word from left to right we keep track of the subscript pattern with a variable k, which starts at 0. When a symbol a_j or $c_{i,j}$ is encountered and $k = 0$, we set $k = j$. If we encounter a symbol a_j or $c_{i,j}$ and $k \neq 0$, we subtract j from k, k must equal 0 after the subtraction for the word to be in the language. k must equal zero at the end of reading the word. Similarly, L_2 consists of words containing symbols a_j, b_i, $c_{i,j}$ and ⚓$_2$. We examine patterns in the subscripts of b_i's and $c_{i,j}$'s to determine membership. If we read a word from left to right we keep track of the subscript pattern with a variable k, which starts at 0. When a symbol b_i or $c_{i,j}$ is encountered and $k = 0$, we set $k = i$. If we encounter a symbol b_i or $c_{i,j}$ and $k \neq 0$, we subtract i from k, k must equal 0 after the subtraction for the word to be in the language. k must equal zero at the end of reading the word. Automata recognizing the languages L_1 and L_2 are depicted in Figure 4.5.

We use these languages to construct our witness language $L_1 \xleftarrow{SDI} L_2$. The fooling set for this language, FS, contains $3NM - M$ elements and is divided into three subsets: FS_1, FS_2, and FS_3.

The fooling set FS_1 is defined as follows:

$$FS_1 = \{(a_j \text{⚓}_2 b_i, b_i \text{⚓}_2 a_j) \mid 0 < j < N, 0 \leq i < M\}.$$

If we take a pair (x, y) from FS_1, then the word xy can be parsed according to Definition 15 as, $y_1 = a_j$, $\bar{y} = \text{⚓}_2 b_i b_i \text{⚓}_2$ and $y_2 = a_j$. The word $y_1 y_2$ is in L_1 and the word $y_1 \bar{y} y_2$ is in L_2.

If we take any two pairs $(x, y), (x', y') \in FS_1$ where $(x, y) \neq (x', y')$, then the word xy' is not in $L_1 \xleftarrow{SDI} L_2$. When $i < i'$ the word xy' takes the form $a_j \text{⚓}_2 b_i b_{i'} \text{⚓}_2 a_j$, which is not in the language since $b_i b_{i'}$ must be in $\bar{y}$ and the matching b_i or $b_{i'}$ is not in the string.

In the case where $j < j'$, the word xy' takes the form $a_j \text{⚓}_2 b_i b_i \text{⚓}_2 a_{j'}$. Since the string starts with a_j, it must be in either the x or y_1 substring as described in Definition 15, and since there is no matching a_j in the string it must not be in the language. Thus, the set FS_1 is consistent with the definitions of a fooling set.

The fooling set FS_2 is defined as follows:

$$FS_2 = \{(a_j b_i, a_j \text{⚓}_2 b_i) \mid 0 \leq j < N, 0 \leq i < M\}.$$

For each pair (x, y), we can concatenate x and y to obtain the word xy. This word can be parsed into sub-words to demonstrate its membership in $L_1 \xleftarrow{SDI} L_2$. The word $xy = a_j b_i a_j \text{⚓}_2 b_i$ can be parsed as per Definition 15 into $y_1 = a_j b_i a_j$, $\bar{y} = \text{⚓}_2$ and $y_2 = b_i$. The word xy is consistent with the definition of $L_1 \xleftarrow{SDI} L_2$.

If we take two pairs $(x, y), (x', y') \in FS_2$ where $(x, y) \neq (x', y')$, we can show xy' is

not in the witness language. If we assume that $i < i'$, then with out loss of generality the word xy' takes the form of $a_j b_i a_j \textcircled{\scriptsize 2} b_{i'}$. This word is not in the language, since b_i must be in y_1 and $b_{i'}$ must be in y_2. Thus, there is no corresponding b_i or $b_{i'}$ to pair with to make the substring in L_2.

If we similarly take $j < j'$, then the word xy' takes the form $a_j b_i a_{j'} \textcircled{\scriptsize 2} b_i$. This word is also not in the witness language when $j > 0$, since the first a_j must be in either the x or y_1 substring and the rest of the string does not contain a corresponding a_j symbol.

In the case where $j = 0$ but $j < j'$, then the word $x'y$ is also not in the witness language. The word $x'y$ takes the form $a_{j'} b_i a_j \textcircled{\scriptsize 2} b_i$. Since $j' > 0$, there will not be a corresponding $a_{j'}$ found in the string. Thus, the set Fs_2 is internally consistent with the properties of a fooling set, but we must show that FS_2 maintains those properties with FS_1.

If we take $(x, y) \in FS_2$ and $(x', y') \in FS_1$, then the word $x'y$ is not in the language. The word $x'y$ takes the form $a_{j'} \textcircled{\scriptsize 2} b_{i'} a_j \textcircled{\scriptsize 2} b_i$. Since $0 < j' < N$, all strings must start with $a_{j'}$ and no corresponding $a_{j'}$ is read. Since the only other symbol from Σ_a is between two $\textcircled{\scriptsize 2}$ symbols, it must be in $\bar{y}$, and therefore the word can not be in $L_1 \xleftarrow{SDI} L_2$.

We define a helper function $f(j)$ to allow us to show FS_3 contains NM elements:

$$f(j) = \begin{cases} 1 & \text{if } j = N - 1 \\ j + 1 & \text{if } 0 \leq j < N - 1. \end{cases}$$

The fooling set FS_3 is defined as follows:

$$FS_3 = \{(c_{i,f(j)}\text{⚓}_2 a_{f(j)}a_j, a_j b_i) \mid 0 \leq j < N, 0 \leq i < M\}.$$

We can show the word xy is in the witness language when $(x, y) \in FS_3$. The word xy can be parsed into the following substrings according to Definition 15, $y_1 = c_{i,f(j)}$, $\bar{y} =$ ⚓$_2$ and $y_2 = a_{f(j)}a_j a_j b_i$. In order for the word xy to be in the language, $a_{f(j)}$ must be in y_2, since $j \neq f(j)$.

If we take the elements $(x, y), (x', y') \in FS_3$, where $(x, y) \neq (x', y')$, then the word xy' is not in the language. If we suppose $j < j' < N - 1$, then the word xy' takes the form $c_{i,j'+1}$⚓$_2 a_{j'+1}a_{j'}a_j b_i$, and have the substring $a_{j'+1}a_{j'}a_j b_i$ in y_2. If not, then either $c_{i,j'+1}$ would not have a matching $a_{j'+1}$ or it would not have a matching b_i. This also implies that $a_{j'}$ does not have a corresponding $a_{j'}$ symbol. If $j = N - 1$ and $0 < j' < N - 2$, then the word $x'y$ is not in the language. The word $x'y$ takes the form $c_{i,j'+1}$⚓$_2 a_{j'+1}a_{j'}a_{N-1}b_i$, and it needs to read $a_{j'+1}$ in y_2. However, this also implies that $a_{j'}$ will be in y_2, but another $a_{j'}$ or $c_{i,j'}$ cannot be found. In the case where $j' = N - 2$, the word $x'y$ is in the language, but xy' is not. The word xy' takes the form $c_{i,1}$⚓$_2 a_1 a_{N-1}a_{N-2}b_1$, which is not in the language since a_{N-1} must be in y_2 or z, and no other a_{N-1} or $c_{i,N-1}$ is in the word.If we consider when $j' = 0$ and $j = N - 1$,we can show that the word $x'y$ is not in the language. The word $x'y$ takes the form $c_{i,1}$⚓$_2 a_1 a_0 a_{N-1}b_i$, this word is not in the language since $c_{i,1}$ must be in y_1, to read the corresponding a_1 and b_i the suffix $a_1 a_0 a_{N-1}b_i$ must be in y_2. $y_1 y_2$ is not a word in L_1 since it must read a second a_{N-1} or $c_{i,N-1}$, but another cannot be found in the string.

If we also examine the case when $i < i'$, the word xy' is not in the language since $c_{i,f(j)}$ or $b_{i'}$ would not be matched by the corresponding b or c symbol. Thus, the set

FS_3 maintains the fooling set properties internally; we now show that the properties hold with FS_1 and FS_2.

If we take $(x, y) \in FS_3$ and $(x', y') \in FS_2$, then the word xy' is not in the language. The word xy' takes the form $c_{i,f(j)}\textcircled{\scriptsize 2}a_{f(j)}a_j a_{j'}\textcircled{\scriptsize 2}b_{i'}$. This word is not in the language since $f(j) \neq 0$ and there are no additional $a_{f(j)}$ or $c_{i,f(j)}$ symbols that are not between two $\textcircled{\scriptsize 2}$ symbols to match the first symbol.

If we take (x, y) from FS_3 and (x', y') from FS_1, we get the word, $xy' = c_{i,f(j)}\textcircled{\scriptsize 2}a_{f(j)}a_j b_{i'}\textcircled{\scriptsize 2}a_{j'}$, which is not in the language unless $j' = f(j)$. This is because there is no $a_{f(j)}$ to match that is not between two $\textcircled{\scriptsize 2}$ symbols. If $j' = f(j)$, then the word $x'y$ is not in the language. The word $x'y$ is of the form $a_{f(j)}\textcircled{\scriptsize 2}b_{i'}a_j b_i$, since $j \neq f(j)$ there is no $a_{f(j)}$ or $c_{i,f(j)}$ to pair with in the word. Thus, the set FS_3 maintains the fooling set properties.

□

Although the nondeterministic state complexity bound presented is not a tight bound for site-directed insertion, this result is still a promising improvement from previous bounds. The previous known upper bound was $3NM + 2N$ [7]; the results presented in Theorem 13 mark an improvement by an additive factor of $2N$. Although the lower bound is not tight and is over an arbitrarily large alphabet, I believe that a new lower bound could be found which contains an additional M terms, thus forming a tight bound. It is important to note that the choice of languages for constructing the witness language do have large alphabet sizes. Ideally, a similar lower bound could be achieved while using a smaller alphabet, preferably binary. The problem of constructing a tight lower bound for site-directed insertion, as well as replicating the lower bound of $3NM - M$ over a smaller alphabet, remain open.

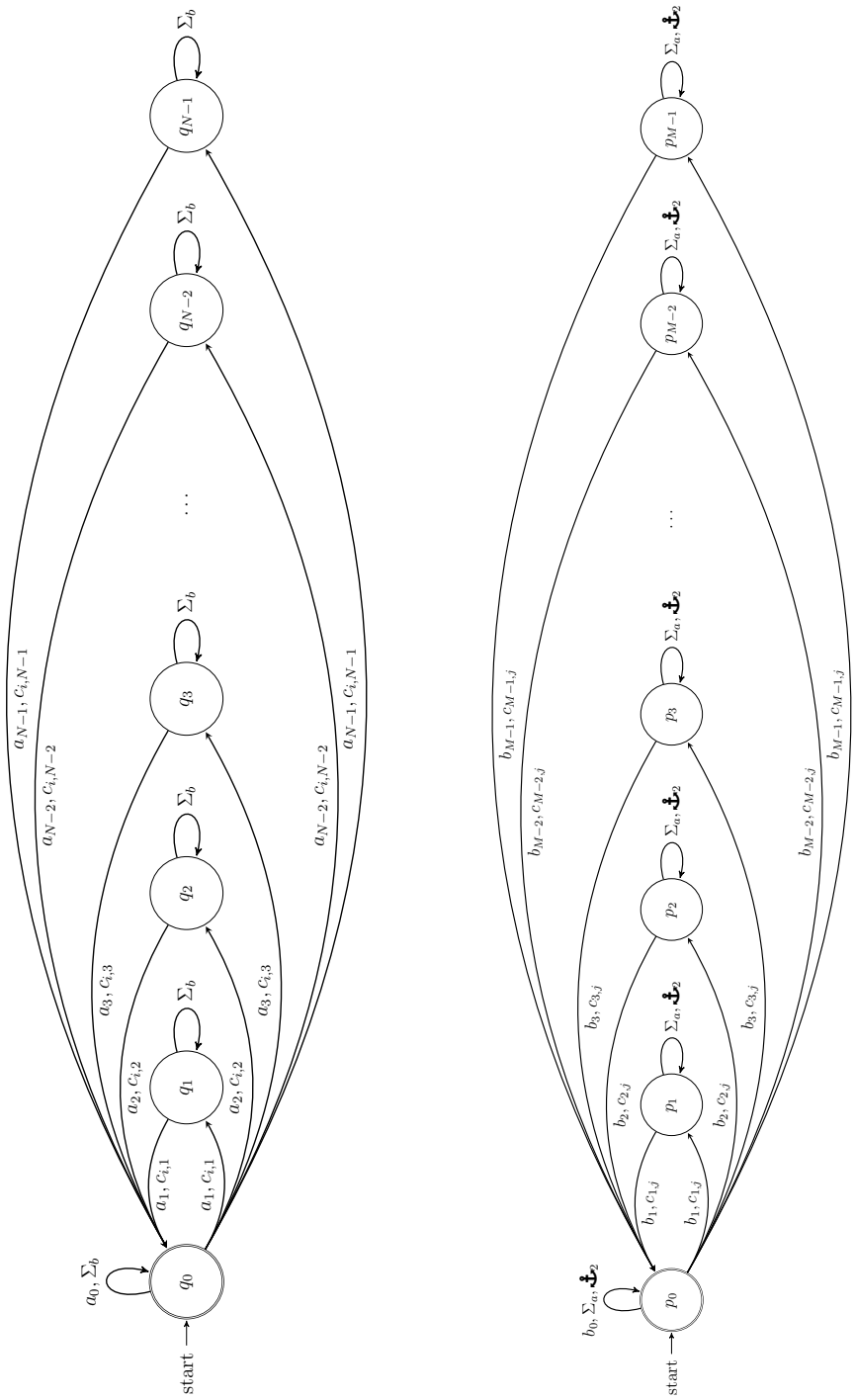

Figure 4.5: The automata used for constructing a witness language for site-directed insertion. Transitions labeled with $c_{i,j}$ where either i or j are not specified, allow for $0 \leq i < M$ and $0 \leq j < N$.

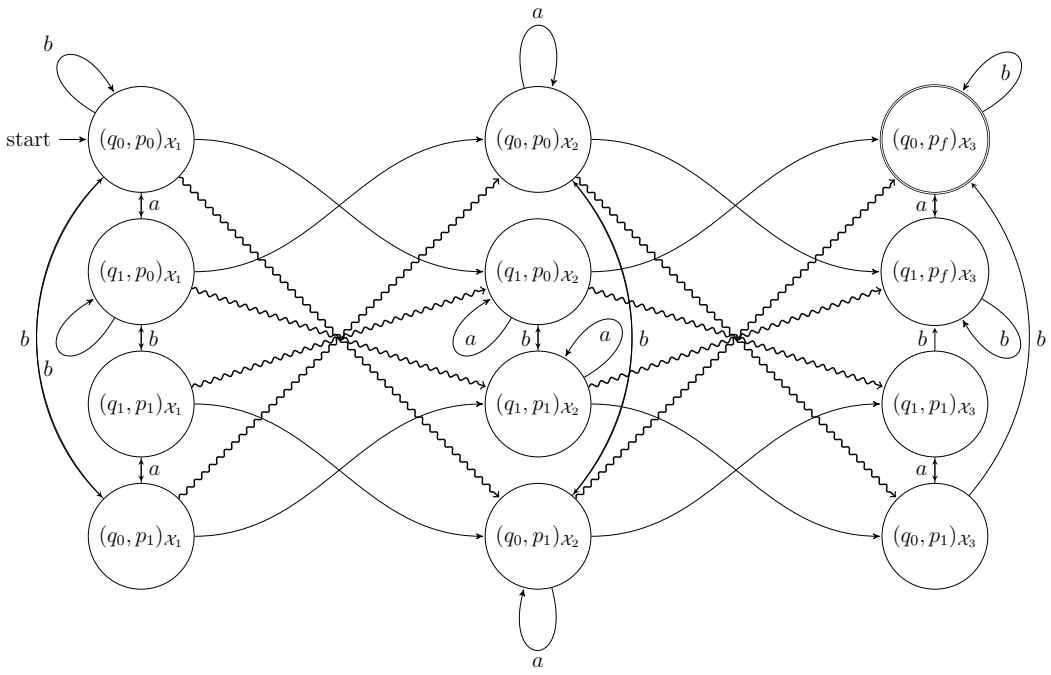

Figure 4.6: A diagram representing the automaton A_{SDI} when $L_1 = \{w \mid w \in \{a, b\}^* \text{ and } \#_a(w) \equiv 0 \mod 2\}$ and $L_2 = \{w \mid w \in \{, a, b\}^* \text{ and } \#_b(w) \equiv 0 \mod 2\}$. Unlabeled transitions with straight lines are a transitions and with squiggly lines are b transitions.

Chapter 5

Unary Site-Directed Insertion

Unary operations are often studied alongside their general counterparts. This allows for a comparison of the general operation and occasionally it will provide insight into general cases. We recall the definition of site-directed insertion of language L_2 into language L_1 is $\{xy_1\bar{y}y_2z \mid xy_1y_2z \in L_1, y_1\bar{y}y_2 \in L_2, y_1 \neq \epsilon, y_2 \neq \epsilon\}$.

When working with unary languages, the order of the substrings $x, y_1, \bar{y}, y_2$ and z is not significant. When site-directed insertion is operating over unary languages the operation can be described as unary concatenation with, at least, the last 2 symbols coalesced. This description of the operation is similar to the chop operation [23] which can be described as concatenation with a single character coalesced. The nondeterministic state complexity of fundamental unary operations has been surveyed in [26]. Table 5.1 contains nondeterministic state complexity results for related operations. We can see that site-directed insertion has similar state complexity as operations with similar string properties.

Table 5.1: Nondeterministic state complexity of operations when L_1,L_2 are represented by NFAs with N and M states respectively.

Operation	Unary	Finite unary
Concatenation[a]	$N+M-1 \leq \cdot \leq N+M$	$N+M-1$
Chop[b]	$N+M$	$N+M-2$
Site-directed insertion[c]	$N+M-3 \leq \overleftarrow{}^{SDI} \leq NM+N+M$	$N+M-3$
Intersection	NM	$min(N,M)$[d]

[a] The nondeterministic state complexity of concatenation was proven by Pighzzini and Shallit [37].
[b] The unary chop operation was studied by Holzer and Jakobi [23].
[c] These results are proven in Theorems 15,16,17.
[d] This result is a consequence of the work done by Mandl [35].

5.1 Finite Unary Languages

We explore the nondeterministic state complexity of the site-directed insertion operation over finite unary languages. We observe that unary concatenation is commutative and this has an impact on the definition for unary site-directed insertion.

Proposition 2. *For unary languages L_1 and L_2,*

$$L_1 \overleftarrow{}^{SDI} L_2 = \{y_1 y_2 \bar{y} x z \mid x y_1 y_2 z \in L_1, y_1 \bar{y} y_2 \in L_2, y_1 \neq \varepsilon, y_2 \neq \varepsilon\}$$

We are able to define an NFA to recognize the language $L(A_1) \overleftarrow{}^{SDI} L(A_2)$, when A_1 and A_2 are NFAs recognizing finite unary languages.

Theorem 15. *Let A_1 and A_2 be unary NFAs recognizing finite languages, where A_1 has N states and A_2 has M states. The language $L(A_1) \overleftarrow{}^{SDI} L(A_2)$ can be recognized with an NFA containing $N+M-3$ states.*

Proof. Let $A_1 = (Q, \Sigma, \delta, q_0, F_1)$ and $A_2 = (P, \Sigma, \gamma, p_0, F_2)$ recognizing L_1 and L_2 respectively. If we label each state in Q and P with a subscript, such that the

subscript indicates the distance of the state from there respective start states, then we get, $Q = \{q_0, q_1, \ldots, q_{N-1}\}$ and $P = \{p_0, p_1, \ldots, p_{M-1}\}$. Given these NFAs, we can construct an NFA $A_{uf-SDI} = (Q_{uf-SDI}, \Sigma, \Omega, q_0, F_2)$, where

$$Q_{uf-SDI} = \{q_0, q_1, \ldots, q_{N-2}\} \cup \{p_2, p_3, \ldots, p_{M-1}\}$$

The transition function Ω is defined as follows:

(i) for the state $q_i \in Q_{uf-SDI}$:

$$\Omega(q_i, \alpha) = \{q' \mid q' \in \delta(q_i, \alpha)\}$$
$$\cup \{\delta(q_i, \alpha) \cap F_1 \neq \emptyset, \text{ and } 2 \leq k \leq i+1\}$$

(ii) for the state $p_i \in Q_{uf-SDI}$:

$$\Omega(p_i, \alpha) = \{p' \mid p' \in \gamma(p, \alpha)\}$$

First we show the inclusion $L(A_1) \xleftarrow{SDI} L(A_2) \subseteq L(A_{uf-SDI})$:

Generally speaking, the states q_i are used to read xzy_1y_2, and the state states p_j read $\bar{y}$. For a word $xy_1\bar{y}y_2z \in L(A_1) \xleftarrow{SDI} L(A_2)$, there must exist an accepting path $\mathcal{P}_{A_1} = s_1, \ldots, q_x, \ldots, q_z, \ldots, q_{y_1}, \ldots, q_{y_2}$ in A_1 recognizing xzy_1y_2. Additionally the accepting path $\mathcal{P}_{A_2} = s_2, \ldots, p_{y_1}, \ldots, p_{y_2}, \ldots, p_{\bar{y}}$ must be in A_2 recognizing $y_1y_2\bar{y}$.

An accepting path exists through A_{uf-SDI} recognizing the word $xzy_1y_2\bar{y}$. Starting in q_0, using the transitions defined in (i) we can read xzy_1, reaching the states q_x, q_z and q_{y_1}. When reading y_2 we use the transitions defined in (i) to arrive at the state

p_{y_2}. We are able to nondeterministically select the state p_{y_2} from the states in P. Lastly, the transitions defined in (ii) are used to read $\bar{y}$ arriving at the accepting state $p_{\bar{y}}$.

Secondly we show the inclusion $L(A_{uf-SDI}) \subseteq L(A_1) \overleftarrow{SDI} L(A_2)$:

If we consider $w_1 w_2' \in L(A_{uf-SDI})$, where $w_1 \in L(A_1)$, then an accepting path can be defined as, $\mathcal{P}_{uf-SDI} = q_0, \ldots, p_h, \ldots, p_{w_2'}$ recognizing $w_1 w_2'$ such that $h \in \{2, \ldots, |w_1|\}$. The suffix w_2' can have the length a word in $L(A_2)$ minus h. This means that words recognized by A_{uf-SDI} are of length $\{max(w_1, w_2), \ldots, w_1 + w_2 - 2\}$ where $w_2 \in L(A_2)$. These words are consistent with the definition of unary site-directed insertion. □

The construction of A_{uf-SDI} is visualized in Figure 5.1. The top automaton displays a construction where $N < M$. In contrast, the bottom automaton is a construction where $M \leq N$. We can see in both cases the state complexity is unchanged.

Theorem 16. *For M, $N \in \mathbb{N}$, there exists two finite languages L_1 and L_2 over unary alphabets with NFAs that have N and M states, respectively, such that any NFA recognizing the language $L_1 \overleftarrow{SDI} L_2$ needs at least $N + M - 3$ states.*

Proof. We take the following languages to construct our witness language:

$$L_1 = \{a^{N-1}\} \text{ and } L_2 = \{a^{M-1}\}$$

We are able to construct the following witness language:

$$L_1 \overleftarrow{SDI} L_2 = \{a^k \mid \max(N, M) \leq k \leq N + M - 4\}$$

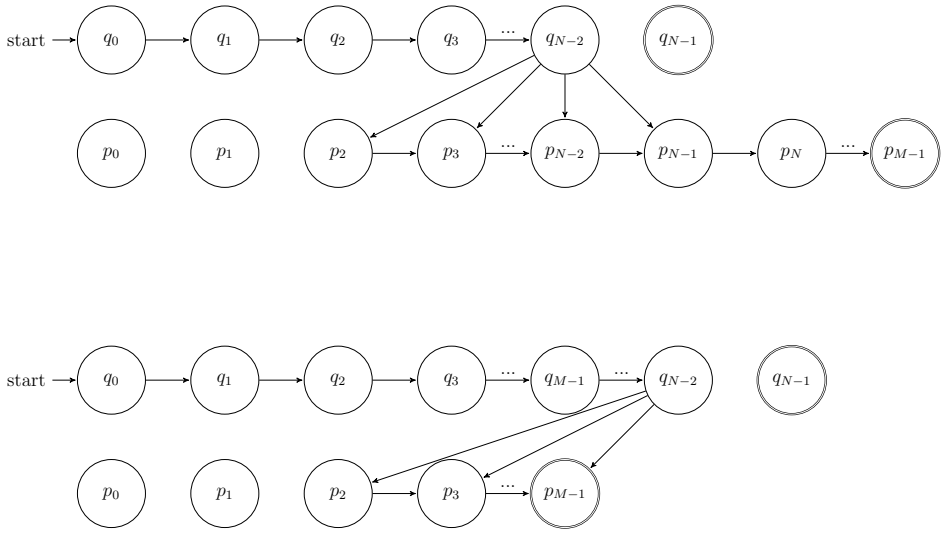

Figure 5.1: Construction of NFAs recognizing finite unary site-directed insertion. The top automaton is the case when $N < M$, where as the bottom automaton is when $M \leq N$ The diagrams intentionally retains the unused states without edges to visually illustrate the state savings.

We can construct a fooling set FS as follows:

$$FS = \{a^r, a^{N+M-4-r}) \mid r \in [0, N+M-4]\}$$

For each pair $(x, y) \in FS$ we can see xy takes the form $a^r a^{N+M-4-r}$, which creates the word a^{N+M-4}, which is in the witness language.

If we instead take $(x, y), (x', y') \in FS$ where $(x, y) \neq (x', y')$, then the word $x'y$ is not in the language when $r < r'$. The word $x'y$ is of the form $a^{r'} a^{N+M-4-r}$, which is not in the language since $N + M - 4 < N + M - 4 - r + r'$. The properties of the fooling set are maintained, thus a lower bound of $N + M - 3$ is achieved. □

Since the upper and lower bounds on nondeterministic state complexity are consistent at $N + M - 3$, we can conclude a tight bound.

Corollary 5. *Let L_1 and L_2 be finite unary languages with NFAs containing N and M states respectively. Then, in the worst case $N + M - 3$ states are necessary and sufficient to recognize $L_1 \xleftarrow{SDI} L_2$.*

5.2 Regular Unary Languages

When working with regular unary languages, it is desirable to produce another intuitive construction similar to the finite case.

Conjecture 1. *For NFAs A_1 and A_2 with N and M states respectively. The languages $L(A_1)$ and $L(A_2)$ are unary. The language $L(A_1) \xleftarrow{SDI} L(A_2)$ can be recognized with an NFA containing $N + M$ states.*

We can attempt an intuitive construction based on the idea that site-directed insertion over unary languages can be achieved through overlap and concatenation of words between the two languages. We propose the following construction:

> Given NFAs $A_1 = (Q, \Sigma, \delta, s_1, F_1)$ and $A_2 = (P, \Sigma, \gamma, s_2, F_2)$. We can construct $A'_{u-SDI} = (Q_{u-SDI}, \Sigma, \Omega, s_1, F_2)$. If we let $Q_{u-SDI} = Q \cup P$, and let Ω maintain the transitions defined in δ and γ. Unlike the finite case, we cannot add all transitions from the final state (or one state prior) since we are uncertain how much of the string has been read prior to arriving at a state. If we iteratively add transitions from A_1 to A_2 using $k = 2, 3, \ldots$, where a state $q \in Q$ is k steps from s_1 and $p \in P$ is $k + 1$ steps from s_2, we add a transition from q to p. The iterative transitions from Q to P are added to behave as the points of transition from xzy_1y_2 to $\bar{y}$ and account for an arbitrary sized y_1y_2.

The problem with a construction using the structures of A_1 and A_2 is that states are potentially reachable multiple times. A counterexample is presented in Example 3. Words that are recognized by using a cycle in A_1, can use the transition to A_2 without passing through the cycle.

Example 3. *We consider the example where* $L_1 = L_2 = (a^5)^*$. *If* $A_1 = (Q, \Sigma, \delta, q_0, \{q_0\})$ *and* $A_2 = (P, \Sigma, \delta, p_0, \{p_0\})$ *are NFAs recognizing* L_1. *Then if we define the NFA* $A'_{u-SDI} = (Q \cup P, \Sigma, \Omega, q_0, \{p_0\})$, *we can define the transitions in* Ω *as the transitions contained in* δ *and* γ, *as well as the following:* $\Omega(q_3, a) = p_3$ $(k = 2)$, $\Omega(q_2, a) = p_4$ $(k = 3)$, $\Omega(q_1, a) = p_0$ $(k = 4)$, $\Omega(q_0, a) = p_1$ $(k = 5)$, *and* $\Omega(q_4, a) = p_2$ $(k = 6)$.

Then there exists a path $\mathcal{P} = q_0, q_1, p_0$ *which is an accepting path recognizing* a^2 *which is not in the language* $L_1 \xleftarrow{SDI} L_2$.

We are able to obtain a much higher upper bound for unary site- directed insertion.

Theorem 17. *Let* A_1 *and* A_2 *be unary NFAs with* N *and* M *states respectively. The language* $L(A_1) \xleftarrow{SDI} L(A_2)$ *can be recognized with an NFA containing* $NM + N + M$ *states.*

Proof. Let $A_1 = (Q, \Sigma, \delta, q_0, F_1)$ and $A_2 = (P, \Sigma, \gamma, p_0, F_2)$ be NFAs recognizing L_1 and L_2 respectively. Given these NFAs, we can construct an NFA $A_{u-SDI} = (Q_{u-SDI}, \Sigma, \Omega, (s_1, \clubsuit), F_{u-SDI})$, where

$$Q_{u-SDI} = (Q \times \clubsuit) \cup (Q \times P) \cup (\clubsuit \times P)$$

The transition function Ω is defined as follows:

(i) for the state $(q, \clubsuit) \in Q_{u-SDI}$:

$$\Omega((q,\clubsuit),\alpha) = \{(q',\clubsuit) \mid q' \in \delta(q,\alpha)\} \cup \{(q',p') \mid q' \in \delta(q,\alpha) \text{ and } p' \in \gamma(s_2,\alpha)\}$$

(ii) for the state $(q, p) \in Q_{u-SDI}$:

$$\Omega((q,p),\alpha) = \{(q',p') \mid q' \in \delta(q,\alpha) \text{ and } p' \in \gamma(p,\alpha)\} \cup \{(\clubsuit,p') \mid \delta(q,\alpha) \cap F_1 \neq \emptyset \text{ and } p' \in \gamma(p,\alpha)\}$$

(iii) for the state $(\clubsuit, p) \in Q_{u-SDI}$:

$$\Omega((\clubsuit,p),\alpha) = \{(\clubsuit,p') \mid p' \in \gamma(p,\alpha)\}$$

First we show the inclusion $L(A_1) \xleftarrow{SDI} L(A_2) \subseteq L(A_{u-SDI})$:
Generally speaking, the states $(q, \clubsuit)$ are used to read the prefix xz. The states (q, p) read the infix y_1y_2 and enforce the length requirements. The states $(\clubsuit, p)$ are used to read the suffix $\bar{y}$.

If we take a word $xzy_1y_2\bar{y}$ from $L(A_1) \xleftarrow{SDI} L(A_2)$ then there must exist accepting paths $\mathcal{P}_{A_1} = s_1, \ldots, q_{xz}, \ldots, q_{y_1y_2}$ in A_1 recognizing xzy_1y_2 and $\mathcal{P}_{A_2} = s_2, \ldots, p_{y_1y_2}, \ldots, p_{\bar{y}}$ in A_2 recognizing $y_1y_2\bar{y}$. From the initial state $(s_1, \clubsuit)$ we can read the prefix xz using the transitions defined in (i) to arrive at the state $(q_{xz}, \clubsuit)$. To read the infix y_1y_2 we read the first symbol using the transition defined in (i) to arrive at a state (q, p), then using the transitions defined in (ii) to arrive at the state $(\clubsuit, p_{y_1y_2})$. Lastly, the transitions defined in (iii) are used to read $\bar{y}$ to arrive at the

accepting state $(\clubsuit, p_{\bar{y}})$.

Secondly we show the inclusion $L(A_{u-SDI}) \subseteq L(A_1) \overleftarrow{\cdot}^{SDI} L(A_2)$:

If we consider a word $xzy_1y_2\bar{y}$ has an accepting path through A_{u-SDI}, then the word must be in $L(A_1) \overleftarrow{\cdot}^{SDI} L(A_2)$. An accepting path $\mathcal{P}_{u-SDI}$ in A_{u-SDI} recognizing $xzy_1y_2\bar{y}$ can be decomposed into three paths:

$$\mathcal{P}_1 = (s_1, \clubsuit), \ldots, (q_{xz}, \clubsuit), \mathcal{P}_2 = (q_{xz}, \clubsuit), \ldots, (\clubsuit, p_{y_1y_2})$$

$$p_3 = (\clubsuit, p_{y_1y_2}), \ldots, (\clubsuit, p_{\bar{y}}).$$

The path $\mathcal{P}_1$ implies that there exists a path $s_1, \ldots, q_{xz}$ in $\mathcal{P}_{A_1}$ recognizing the prefix xz. Path $\mathcal{P}_2$ shows the existence of paths $q_{xz}, \ldots, q_{y_1y_2}$ in $\mathcal{P}_{A_1}$ and $s_2, \ldots, p_{y_1y_2}$ in $\mathcal{P}_{A_2}$ that recognize a string y_1y_2. The path $\mathcal{P}_2$ starts by non-deterministically choosing to move to a state (q, p). Finally, the path $\mathcal{P}_3$ implies that the path $q_{y_1y_2}, \ldots, q_{\bar{y}}$ exists in $\mathcal{P}_{A_2}$ recognizing the string $\bar{y}$. This terminates the simulation at final state $(\clubsuit, p_{\bar{y}})$, and implies that if A_{u-SDI} recognizes $xzy_1y_2\bar{y}$, then there exists accepting paths on A_1 and A_2 recognizing the strings xzy_1y_2 and $y_1y_2\bar{y}$. □

In Figure 5.2 we can see the construction of the unary site-directed insertion automaton. The states $(q, \clubsuit)$ have outgoing transitions defined in (i). The states (q, p) have their outgoing transitions defined in (ii). Lastly, the states $(\clubsuit, p)$ have outgoing transitions defined in (iii).

As shown earlier in Table 5.1 the nondeterministic state complexity for the unary site-directed insertion operation has a large gap between upper and lower bounds. The lower bound for unary site directed insertion is provided in Theorem 16. The gap in state complexity is $N + M - 3 \leq \overleftarrow{\cdot}^{SDI} \leq NM + N + M$. It remains open to

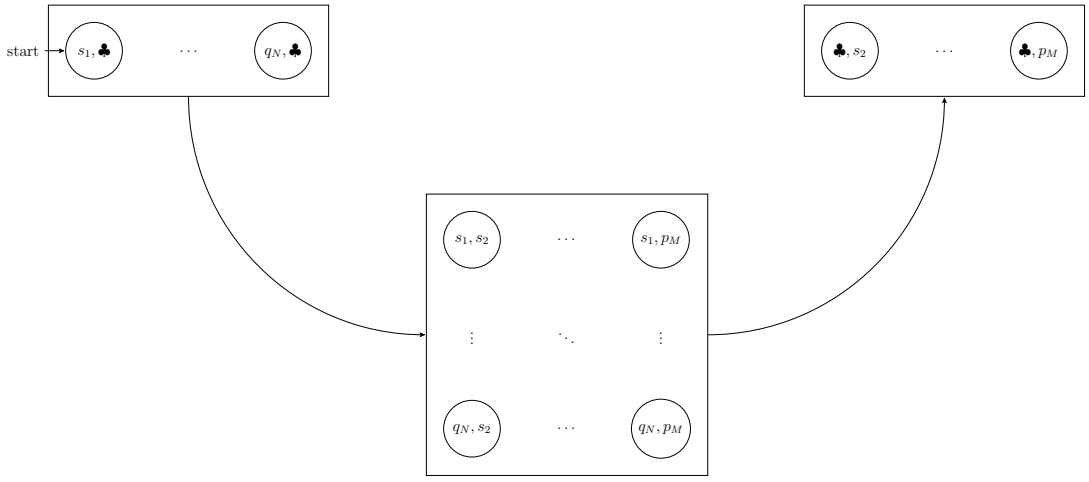

Figure 5.2: The construction of the unary site-directed insertion automaton.

close the state complexity gap and achieve a tight bound.

Chapter 6

Site-Directed Deletion

The natural companion operation to site-directed insertion is site-directed deletion. As with site-directed insertion, site-directed deletion has several related operations that have been studied. The most similar operation previously considered is the contextual deletion operation studied by Kari and Thierrin [33]. The other operation studied was deletion along trajectories [11]. The majority of the work published on related operations were focused on proving decidability, closures, and other language properties. Site-directed deletion was first proposed by Cho et al., [9]. We are currently interested in quantifying the non-deterministic state complexity of site-directed deletion.

Definition 16. *(Site-Directed Deletion) For languages L_1 and L_2, the site-directed deletion language of L_2 on L_1 is defined as follows:*

$$L_1 \xleftarrow{SDD} L_2 = \{xy_1y_2z \mid xy_1\bar{y}y_2z \in L_1, y_1y_2 \in L_2, y_1 \neq \varepsilon, y_2 \neq \varepsilon\}$$

6.1 Unary Site-Directed Deletion

We observe that unary concatenation is commutative and this has an impact on the definition for unary site-directed deletion.

Proposition 3. *For unary languages L_1 and L_2,*

$$L_1 \xleftarrow{SDD} L_2 = \{y_1y_2xz \mid xy_1\bar{y}y_2z \in L_1, y_1y_2 \in L_2, y_1 \neq \varepsilon, y_2 \neq \varepsilon\}$$

When working with unary languages L_1 and L_2, the language $L_1 \xleftarrow{SDD} L_2$ is equivalent to the language $L_1 \xleftarrow{SDD} \{w\}$, where w is the shortest word in L_2 that has a length greater than 1. This is proven by showing that any word recognized with a longer word w' can also be recognized by the shorter word w.

Lemma 1. *For the unary language L_1 and the two words w and w' where $2 \leq |w| < |w'|$. The languages $L_1 \xleftarrow{SDD} \{w'\} \subseteq L_1 \xleftarrow{SDD} \{w\}$.*

Proof. If we consider $w = y_1y_2$ and $w' = y_1y_2'$ where $y_2 < y_2'$, then the words produced by SDD with y_1y_2' will be the subset of those recognized by y_1y_2. If a word y_1y_2' is in $L_1 \xleftarrow{SDD} \{w'\}$, then it must also be in $L_1 \xleftarrow{SDD} \{w\}$ and can be parsed with respect to Proposition 3 as y_1y_2xz with w where $|xz| = |y_1y_2'| - |y_1y_2|$. □

If L_1 has infinite cardinality, the language $L_1 \xleftarrow{SDD} L_2$ contains all words longer than the shortest word in $w_2 \in L_2$. This occurs because a word of arbitrary length from $w_1 \in L_1$ can be selected and words of length between $|w_2|$ and $|w_1|$ are generated.

Lemma 2. *Let L_1 and L_2 be unary languages and L_1's cardinality be infinite. All words of length of at least k will be in $L_1 \xleftarrow{SDD} L_2$ where k is the length of the shortest word in L_2 of length at least 2.*

Proof. Let y_1y_2 be the shortest word in L_2 of length at least 2, where $y_1 \neq \varepsilon$ and $y_2 \neq \varepsilon$. Consider an arbitrary $p \in \mathbb{N}$, $p \geq \mid y_1y_2 \mid$.

Choose $w \in L_1$ such that $\mid w \mid \geq p$ and write $w = w_1w_2$ where $\mid w_1 \mid = p - \mid y_1y_2 \mid$.

Now $w_1y_1w_2y_2 \in L_1$, $y_1y_2 \in L_2$ and, by the definition of site-directed deletion, $w_1y_1y_2 \in L_1 \overleftarrow{\ }^{SDD} L_2$. Thus $L_1 \overleftarrow{\ }^{SDD} L_2$ contains the word of length p. □

6.2 Unary Site-Directed Deletion Bounds

Site-directed deletion over unary languages has interesting state complexity properties. An NFA with either N or M states recognizes the language $L(A_1) \overleftarrow{\ }^{SDD} L(A_2)$, depending on whether or not $L(A_1)$ is a finite language.

Theorem 18. *For unary NFAs A_1 and A_2 with N and M states, respectively, the language $L(A_1) \overleftarrow{\ }^{SDD} L(A_2)$ is recognized by an NFA with N states, when $L(A_1)$ is finite.*

Proof. Let $A_1 = (Q, \Sigma, \delta, q_0, F_1)$ and $A_2 = (P, \Sigma, \gamma, p_0, F_2)$ be automata recognizing the unary languages L_1 and L_2 with N and M states respectively. Given these NFAs, we can construct the NFA $A_{u-SDD} = (Q_{u-SDD}, \Sigma, \Omega, p_0, F_{u-SDD})$.

If we label the states in Q and P with a subscript i, such that the states p_i and q_i are i steps from their respective start states in A_1 and A_2. If we also define k to be the shortest word in L_2 with length greater than 1. We then define the states in Q_{u-SDD} as follows:

$$Q_{u-SDD} = \{p_0, p_1, \ldots, p_{k-1}\} \cup \{q_k, \ldots, q_{N-1}\}$$

A state q_i is in F_{u-SDD} when $q_i \in F_1$ and $k \leq i < N$.

The transitions of Ω are defined as follows:

(i) for the state $p \in P$:

$$\Omega(p_i, \alpha) = \{p' \mid p' \in \gamma(p_i, \alpha) \text{ and } p' \notin F_2\}$$
$$\cup \{q_j \mid 1 < i < j < N \text{ and } \gamma(p_i, \alpha) \cap F_2 \neq \emptyset\}$$

(ii) for the state $q \in Q$:

$$\Omega(q, \alpha) = \{q' \mid q' \in \delta(q, \alpha)\}$$

It is important to note that the start state of A_{u-SDD} is s_2. A_{u-SDD} starts computation in A_2 and then transitions to A_1.

First we show the inclusion $L(A_1) \xleftarrow{SDD} L(A_2) \subseteq L(A_{u-SDD})$:

Generally speaking, the automaton A_{u-SDD} uses the states $\{p_0, p_1, \ldots, p_{k-1}\}$ to compute the substring y_1y_2, and the states $\{q_k, \ldots, q_{N-1}\}$ to compute xz.

If we take an arbitrary word y_1y_2xz from $L_1 \xleftarrow{SDD} L_2$, there exists paths, $\mathcal{P}_{A_1} = q_0, \ldots, q_{y_1}, \ldots, q_{y_2}, \ldots, q_{\bar{y}}, \ldots, q_x, \ldots, q_z$ recognizing $y_1y_2\bar{y}xz$ in A_1 and $\mathcal{P}_{A_2} = p_0, \ldots, p_{y_1}, \ldots, p_{y_2}$ recognizing y_1y_2 in A_2.

We can construct an accepting path through A_{u-SDD} to read the word y_1y_2xz. Starting at p_0, we next read the prefix y_1 using the transition defined in (i) to arrive at the state p_{y_1}. Again, using the transitions defined in (i) we read y_2 to arrive at the state $q_{\bar{y}}$. It is important to note that the transition (i) nondeterministicly chooses $q_{\bar{y}}$ since $q_{\bar{y}}$ has a subscript larger than $k-1$ by definition. Finally, the suffix xz is read using the transitions defined in (ii) to arrive at the accepting state q_z.

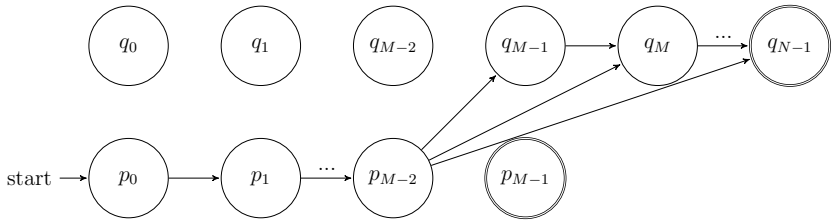

Figure 6.1: NFA construction of the case where L_1 is finite for unary SDD. The diagram intentionally retains the unused states without transitions to illustrate the state savings visually.

Secondly we show the inclusion $L(A_{u-SDD}) \subseteq L(A_1) \overleftarrow{^{SDD}} L(A_2)$:

For a word y_1y_2xz recognized by an accepting path $\mathcal{P}_{u-SDD} = p_0, \ldots, q_{\bar{y}}, \ldots, q_z$ in A_{u-SDD}, we can decompose this path into $\mathcal{P}_1 = p_0, \ldots, q_{\bar{y}}$ and $\mathcal{P}_2 = q_{\bar{y}}, \ldots, q_z$.

The path $\mathcal{P}_1$ implies that the prefix y_1y_2 is recognized by A_2. The second path $\mathcal{P}_2$ nondeterministicly guesses how long $\bar{y}$ is and implies that a word $y_1y_2\bar{y}xz$ is recognized by L_1. □

Figure 6.1 illustrates the construction of A_{u-SDD} when L_1 is a finite language. We can visually confirm that the construction does contain N states.

Theorem 19. *For M, $N \in \mathbb{N}$. There exists a finite unary language L_1 with an NFA with N states and L_2 with an NFA with M states. Then any NFA recognizing the language $L_1 \overleftarrow{^{SDD}} L_2$ needs at least N states.*

Proof. For the languages $L_1 = \{a^{N-1}\}$ and $L_2 = \{a^{M-1}\}$, where $M \leq N$. We can construct the following witness language:

$$L_1 \overleftarrow{^{SDD}} L_2 = \{a^i \mid i \in [M-1, \ldots, N-1]\}$$

The fooling set $FS = \{(a^{N-1-i}, a^i) \mid 0 \leq i \leq N-1\}$ exists for the witness language.

If we consider an element $(x, y) \in FS$, the word xy takes the form $a^{N-1-i+i}$, which is always in the witness language. If instead we take two elements $(x, y), (x', y') \in FS$ where $(x, y) \neq (x', y')$, then word xy' is not in the witness language when $i < i'$. The word xy' will take the form $a^{N-1-i}a^{i'}$ which has a length longer than $N - 1$. □

We next consider when L_1 is an infinite unary language.

Theorem 20. *Let A_1 and A_2 be unary NFAs with with N and M states, respectively, where $L(A_1)$ is infinite. Then $L(A_1) \xleftarrow{SDD} L(A_2)$ is recognized by an NFA with M states.*

Proof. Given the NFAs $A_1 = (Q, \Sigma, \delta, q_0, F_1)$ and $A_2 = (P, \Sigma, \gamma, p_0, F_2)$, we can construct an automaton $A_{u-SDD} = (P, \Sigma, \Omega, s_2, F_{u-SDD})$. If we label each state in P with a subscript i, where i is the minimum number of steps to reach state p_i from s_2. We define the transition function Ω as follows:

(i) for the state $p \in P$:

$$\Omega(p_i, \alpha) = \{p' \mid p' \in \gamma(p_i, \alpha)\}$$

(ii) for the state $q \in Q$:

$$\Omega(p_f, \alpha) = \{p_f \mid p_f \in F_2 \text{ and } 2 \leq 2\}$$

First we show the inclusion $L(A_1) \xleftarrow{SDD} L(A_2) \subseteq L(A_{u-SDD})$:

Generally speaking, the states $P \backslash F_2$ read the prefix y_1y_2, and the states F_2 read the suffix xz.

Since $L(A_1)$ is an infinite language we know by Lemma 2 that any word longer than y_1y_2 is in the language $L(A_1)\xleftarrow{SDD}L(A_2)$. We also know that there exists for the word y_1y_2 an accepting path $\mathcal{P}_{A_2} = p_0, \ldots, p_{y_1}, \ldots, p_{y_2}$ in A_2, as well the accepting path $\mathcal{P}_{A_1} = q_0, \ldots, q_{y_1}, \ldots, q_{y_2}, \ldots, q_{\bar{y}}, \ldots, q_{xz}$ recognizing the word $y_1y_2\bar{y}xz$.

We can construct an accepting path in A_{u-SDD} for the word y_1y_2xz. Starting at the state p_0, we can use the transitions defined in (i) to read y_1y_2 to arrive at the state p_{y_2}. We can then read the suffix xz using the transition defined in (ii) to stay in the state p_{y_2}. Thus the word y_1y_2xz is recognized by A_{u-SDD}.
Secondly we show the inclusion $L(A_{u-SDD}) \subseteq L(A_1)\xleftarrow{SDD}L(A_2)$:

Let y_1y_2xz be a word in $L(A_{u-SDD})$, then y_1y_2xz is a word in $L(A_1)\xleftarrow{SDD}L(A_2)$. We can decompose the accepting path $\mathcal{P}_{u-SDD} = p_0, \ldots, p_{y_1}, \ldots, p_{y_2}, \ldots, p_{y_2}$, in to the following paths:

$$\mathcal{P}_1 = p_0, \ldots, p_{y_2} \text{ and } \mathcal{P}_2 = p_{y_2}, \ldots, p_{y_2}.$$

The path $\mathcal{P}_1$ reads the string y_1y_2 and the path $\mathcal{P}_2$ reads the string xz. If we define $l_1 = |y_1y_2xz|$, then since $L(A_1)$ is an infinite language there must be some word $W \in L(A_2)$ such that $l_1 \leq |W|$. We can define W as $y_1y_2xz\bar{y}$, where $|\bar{y}| = |W| - |y_1y_2xz|$.

The path $\mathcal{P}_2$ implies that the prefix y_1y_2 is recognized by A_2. The fact that $L(A_1)$ is infinite implies the existence of a word W. Thus the word y_1y_2xz must be in $L(A_1)\xleftarrow{SDD}L(A_2)$. □

In Figure 6.2, we can see the construction of the unary NFA when L_1 is an infinite language.

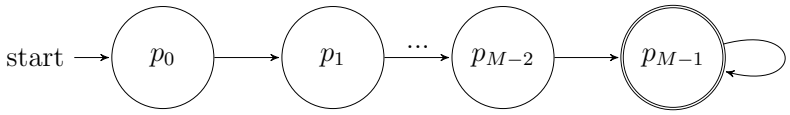

Figure 6.2: NFA construction of the case where L_1 is infinite for unary SDD. The diagram intentionally retains the unused states without transitions to illustrate the state savings visually.

Theorem 21. *Let M, $N \in \mathbb{N}$. There exists an infinite unary language L_1 with an NFA containing N states and L_2 with an NFA containing M states such that, any NFA recognizing the language $L_1 \xleftarrow{SDD} L_2$ needs at least M states.*

Proof. If we take $L_1 = (a^N)^*$ and $L_2 = \{a^{M-1}\}$, we can construct the following witness language:

$$L_1 \xleftarrow{SDD} L_2 = \{a^{M-1+i} \mid i = 0, 1, 2, \ldots\}$$

The fooling set $FS = \{(a^{M-1-i}, a^i) \mid 0 \leq i \leq M-1\}$ exists for the witness language. If we consider any element $(x, y) \in FS$, then the word xy takes the form a^{M-1}, which is in the witness language. If we consider the elements $(x, y), (x', y') \in FS$ where $(x, y) \neq (x', y')$, then one of the words xy' of $x'y$ has length less than $M-1$ and is not in the language. The word xy' takes the form $a^{M-1-i+i'}$ and if we assume $i < i'$ then $M - 1 - i + i' < M - 1$, which implies the word is not in the language. Thus the set FS is consistent with the properties of a fooling set. □

When the cardinality of $L(A_1)$ is known, then a tight bound on the non-deterministic state complexity can be found. When $L(A_1)$ is finite, a tight bound of N states is required. In the general case of site-directed deletion over unary languages we can conclude that the state complexity is bound between $min(N, M)$ and $max(N, M)$.

6.3 Site-Directed Deletion

The general results have a significantly larger bound than the unary cases. To prove an upper bound on state complexity we introduce a function which returns the reachable states for an automaton from a given state. Consider $A = (Q, \Sigma, \delta, s, F)$ and define,

$$Closure_A(q) = \{q' \mid \exists\ path\ \mathcal{P}\ from\ q\ to\ q'\ in\ A\}$$

Theorem 22. *For NFAs A_1 and A_2 with N and M states, respectively, the language $L(A_1)\xleftarrow{SDD}L(A_2)$ is recognized by an NFA with $2NM + N$ states.*

Proof. For NFAs $A_1 = (Q, \Sigma, \delta, s_1, F_1)$ and $A_2 = (P, \Sigma, \gamma, s_2, F_2)$. We can define the automaton $A_{SDD} = (Q_{SDD}, \Sigma, \Omega, (s_1, s_2)_{\mathcal{X}_1}, F_{SDD})$ which recognizes the language $L(A_1)\xleftarrow{SDD}L(A_2)$. We define the states in Q_{SDD} as follows:

$$Q_{SDD} = (Q \times P)_{\mathcal{X}_1} \cup (Q \times P)_{\mathcal{X}_2} \cup (Q \times \{p_f\})$$

Where the subscript $\mathcal{X}_i$ indicates copies of the Cartesian product of the sets Q and P. The states $(q, p_f) \in F_{SDD}$, when $q \in F_1$. We define the transition function Ω as follows:

(i) for $(q, s_2)_{\mathcal{X}_1}$ where $q \in Q$:

$$\begin{aligned}\Omega((q, s_2)_{\mathcal{X}_1}, \alpha) = &\{(q', s_2)_{\mathcal{X}_1} \mid q' \in \delta(q, \alpha)\} \\ &\cup \{(q', p')_{\mathcal{X}_1} \mid q' \in \delta(q, \alpha) \text{ and } p' \in \gamma(s_2, \alpha)\} \\ &\cup \{(q'', p')_{\mathcal{X}_2} \mid q'' \in Closure_{A_1}(q'), q' \in \delta(q, \alpha) \text{ and } p' \in \gamma(s_2, \alpha)\}\end{aligned}$$

(ii) for $(q,p)_{\mathcal{X}_1}$ where $q \in Q$ and $p \in P$:

$$\Omega((q,p)_{\mathcal{X}_1},\alpha) = \{(q',p')_{\mathcal{X}_1} \mid q' \in \delta(q,\alpha) \text{ and } p' \in \gamma(p,\alpha)\}$$
$$\cup \{(q'',p')_{\mathcal{X}_2} \mid q'' \in Closure_{A_1}(q'), q' \in \delta(q,\alpha) \text{ and } p' \in \gamma(p,\alpha)\}$$

(iii) for $(q,p)_{\mathcal{X}_2}$ where $q \in Q$ and $p \in P$:

$$\Omega((q,p)_{\mathcal{X}_2},\alpha) = \{(q',p')_{\mathcal{X}_2} \mid q' \in \delta(q,\alpha) \text{ and } p' \in \gamma(p,\alpha)\}$$
$$\cup \{(q',p_f) \mid q' \in \delta(q,\alpha) \text{ and } \gamma(p,\alpha) \cap F_2 \neq \emptyset\}$$

(iv) for (q,p_f) where $q \in Q$:

$$\Omega((q,p_f),\alpha) = \{(q',p_f) \mid q' \in \delta(q,\alpha)\}$$

First we show the inclusion $L(A_1) \xleftarrow{SDD} L(A_2) \subseteq L(A_{SDD})$:

Generally speaking, the states $(q,s_2)_{\mathcal{X}_1}$ read the prefix x. The states $(q,p)_{\mathcal{X}_1}$ are used to read the substring y_1 and insure that it is non-empty. The states $(q,p)_{\mathcal{X}_2}$ read the substring y_2 and also insure that it is non-empty. Lastly, the state (q,p_f) are used to read any suffix z.

For a word $xy_1y_2z \in L(A_1) \xleftarrow{SDD} L(A_2)$, there exists accepting paths $\mathcal{P}_{A_1} = s_1,\ldots,q_x,\ldots,q_{y_1},\ldots,q_{\bar{y}},\ldots,q_{y_2},\ldots,q_z$ in A_1 recognizing $xy_1\bar{y}y_2z$ and $\mathcal{P}_{A_2} = s_2,\ldots,p_{y_1},\ldots,p_{y_2}$ in A_2 recognizing y_1y_2.

From the initial state $(s_1,s_2)_{\mathcal{X}_1}$, the prefix x can be read using the transitions defined in (i) to arrive at the state $(q_x,s_2)_{\mathcal{X}_1}$. The substring y_1 can be read using transitions defined in (ii) to arrive at the state $(q_{\bar{y}},p_{y_1})_{\mathcal{X}_2}$. The last symbol of y_1

can nondeterministicly choose to transition to $(q_{\bar{y}}, p_{y_1})_{\mathcal{X}_2}$ instead of $(q_{y_1}, p_{y_1})_{\mathcal{X}_1}$. The substring y_2 is read using transitions defined in (iii) to arrive at the state (q_{y_2}, p_f). Lastly, the suffix z is read using the transitions defined in (iv) to arrive at the final state (q_z, p_f)

We next show the inclusion $L(A_{SDD}) \subseteq L(A_1) \xleftarrow{SDD} L(A_2)$:

Let xy_1y_2z be a word with an accepting path in A_{SDD}. We can decompose the words accepting path into four paths as follows:

$$\mathcal{P}_1 = (s_1, s_2)_{\mathcal{X}_1}, \ldots, (q_x, s_2)_{\mathcal{X}_1}, \mathcal{P}_2 = (q_x, s_2)_{\mathcal{X}_1}, \ldots, (q_{\bar{y}}, p_{y_1})_{\mathcal{X}_2}$$

$$\mathcal{P}_3 = (q_{\bar{y}}, p_{y_1})_{\mathcal{X}_2}, \ldots, (q_{y_2}, p_f) \text{ and } \mathcal{P}_4 = (q_{y_2}, p_f), \ldots, (q_z, p_f)$$

The path $\mathcal{P}_1$ implies that a path $s_2, \ldots, q_x$ exists in A_1 recognizing x. The path $\mathcal{P}_2$ implies the existence of the paths $q_x, \ldots, q_{y_1}$ in A_1 and $s_2, \ldots, p_{y_1}$ in A_2 recognizing the substring y_1. There is a nondeterministic transition from state $(q_{y_1}, p_{y_1})_{\mathcal{X}_1}$ to $(q_{\bar{y}}, p_{y_1})_{\mathcal{X}_2}$. This transitions exists because there is a path between q_{y_1} and $q_{\bar{y}}$ in A_1 recognizing some substring $\bar{y}$. The path $\mathcal{P}_3$ implies the existence of paths $q_{\bar{y}}, \ldots, q_{y_2}$ in A_2 and $p_{y_1}, \ldots, p_{y_2}$ recognizing the substring y_2 where p_{y_2} is a final state in A_2. Finally, the path $\mathcal{P}_4$ implies that the path $q_{y_2}, \ldots, q_z$ exists in A_1 recognizing the suffix z. This implies that the words $xy_1\bar{y}y_2z \in L(A_1)$ and $y_1y_2 \in L(A_2)$. □

Figure 6.3 depicts the automaton A_{SDD}. The different boxes identify states that have different outgoing transitions. The box labeled with $\mathcal{X}_1$ contains a rectangle, this rectangle contains states with outgoing transitions defined in (i), the remainder of the states in box $\mathcal{X}_1$ have outgoing transitions defined in (ii). The middle box labeled with an $\mathcal{X}_2$ contains states with outgoing transitions defined in (iii). Lastly,

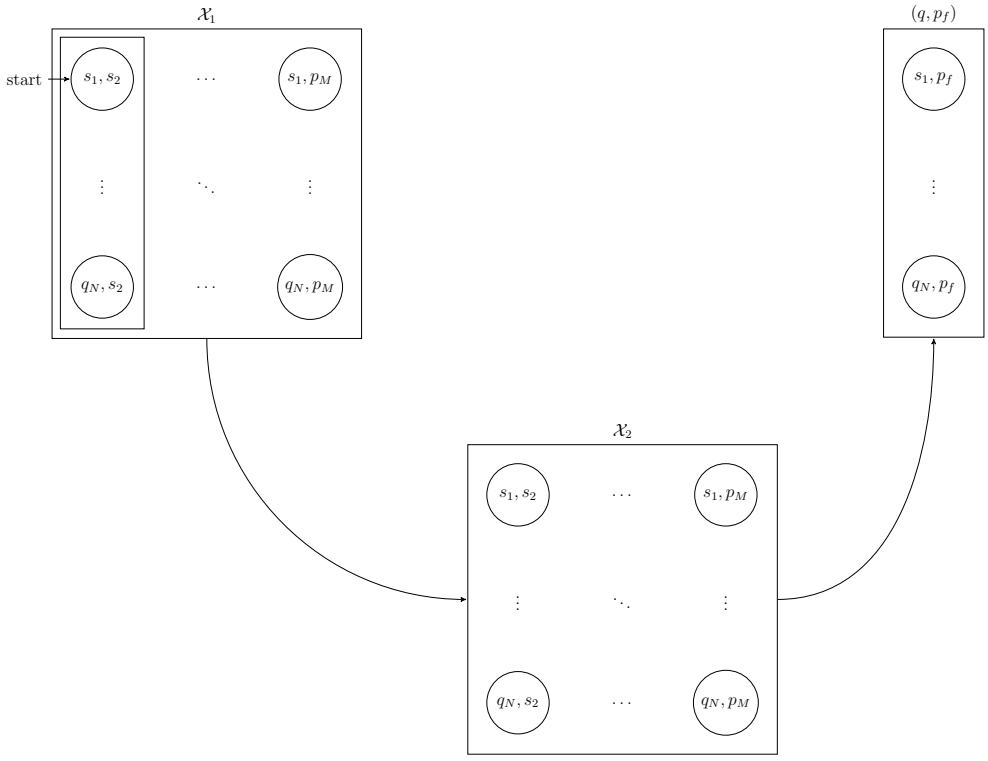

Figure 6.3: The construction of the site-directed deletion automaton.

the states labeled with (q, p_f), have outgoing transitions defined in (iv). An example of the construction for the site-directed deletion automaton is contained in Figure 6.4.

To prove the lower bound on nondeterministic state complexity of site-directed deletion, we introduce notation to simplify the following proof. If we use an alphabet $\Sigma_a = \{a_0, a_1, \ldots, a_N\}$, we can define the operation $\bullet_i^j$ for $i \leq j$ to be the repeated concatenation of the symbols $a_i a_{i+1} \ldots a_{j-1} a_j$ (i.e. $\bullet_1^4 = a_1 a_2 a_3 a_4$).

Theorem 23. *For $2 \leq M, N \in \mathbb{N}$, there exists two regular languages L_1 and L_2 over arbitrarily large alphabets with NFAs that have N and M states, respectively. Then any NFA recognizing the language $L_1 \xleftarrow{SDD} L_2$ needs at least $2NM$ states.*

Proof. We use the languages L_1 and L_2 to construct our witness language.

$$L_1 = (b^*a_0b^*a_1b^* \dots b^*a_{N-2}b^*a_{N-1}b^*)^*$$

$$L_2 = \{w \in \{a_0, \dots, a_{N-1}, b\}^* \mid \#_b(w) \mod M \equiv 0\}$$

L_1 can be represented with an NFA containing N states and can the words in L_1 can be described as words containing a sequence of a_i symbols, with arbitrary numbers of $b's$ between each a_i. L_2 can be represented as an NFA containing M states, the language L_2 can described as the set of strings containing $\mod M$ b's. We can construct the witness language $L_1 \xleftarrow{SDD} L_2$.

A fooling set $FS = FS_1 \cup FS_2$ can defined for the witness language.

The fooling set FS_1 is defined as follows:

$$FS_1 = \{(\bullet_0^{j-1}b^ia_{j+1 \mod N}, b^{M-i}\bullet_{j+2 \mod N}^{N-1}) \mid 1 \leq i \leq M, 1 \leq j \leq N\}$$

For each pair $(x, y) \in FS_1$, we find the word xy takes the form, $\bullet_0^{j-1}b^ia_{j+1 \mod N}b^{M-i}\bullet_{j+2 \mod N}^{N-1}$. The word xy is always in the witness language since we can parse it into the substrings described in Definition 16. Let $x = \bullet_0^{j-1}$, $y_1 = b^i$, $\bar{y} = a_j$, $y_2 = a_{j+1 \mod N}b^{M-i}$ and $z = \bullet_{j+2 \mod N}^{N-1}$. We can see that $xy_1\bar{y}y_2z$ is in L_1 since it has a contiguous subscripts for the a symbols. We also see that the word y_1y_2 contains M $b's$.

If we take $(x, y), (x', y') \in FS_1$, then we can show that xy' is not in the witness language when $(x, y) \neq (x', y')$. If we assume $i < i'$, then the word xy' takes the form, $\bullet_0^{j-1}b^ia_{j+1 \mod N}b^{M-i'}\bullet_{j+2 \mod N}^{N-1}$. Since $0 < i$, there is always at least one b symbol between a_{j-1} and a_{j+1}, which implies it is either in the substring y_1 or y_2. The word

xy' does not contain M b symbols which implies there is no way to parse the word xy' to meet the criteria set by L_2.

If we also assume without loss of generality that $j < j'$, the word xy' is not in the language. The word xy' takes the form, $\bullet_0^{j-1} b^i a_{j+1 \mod N} b^{M-i} \bullet_{j'+2 \mod N}^{N-1}$. The word xy' is not in the language since there must have been two deletions conducted in this word to have the substring $a_{j-1} b^i a_{j+1 \mod N} b^{M-i} a_{j'+2 \mod N}$. The of properties a fooling set are upheld by FS_1

The fooling set FS_2 is defined as follows:

$$FS_2 = \{(\bullet_0^{j-2} b^i, a_{j-1 \mod N} b^{M-i} \bullet_{j+1 \mod N}^{N-1}) \mid 0 \leq i \leq M-1, 2 \leq j \leq N+1\}$$

Similarly to the pairs of FS_1, each pair $(x, y) \in FS_2$, forms the word xy which is in the witness language. The word xy takes the form, $\bullet_0^{j-2} b^i a_{j-1 \mod N} b^{M-i} \bullet_{j+1 \mod N}^{N-1}$, and can be parsed with accordance to Definition 16. We can take $x = \bullet_0^{j-2}$, $y_1 = b^i a_{j-1 \mod N}$, $\bar{y} = a_{j \mod N}$, $y_2 = b^{M-i}$, and $z = \bullet_{j+1 \mod N}^{N-1}$.

If we take $(x, y), (x', y') \in FS_2$, then we can show that $x'y$ is not in the witness language when $(x, y) \neq (x', y')$. Let $i < i'$, then the word $x'y$ takes the form, $\bullet_0^{j-2} b^{i'} a_{j-1 \mod N} b^{M-i} \bullet_{j+1 \mod N}^{N-1}$. This word is not in the language since there is at least one b symbol in gap between $a_{j-1 \mod N}$ and $a_{j+1 \mod N}$ which implies that the $b's$ are in the substring y_1 or y_2. Since there are not M $b's$ in the word $x'y$, it is not possible for y_1 and y_2 to form a word $y_1 y_2 \in L_2$.

We can also assume $j < j'$ without loss of generality to show that $x'y$ is not in the witness language. The word $x'y$ takes the form, $\bullet_0^{j'-2} b^i a_{j-1 \mod N} b^{M-i} \bullet_{j+1 \mod N}^{N-1}$. The word $x'y$ is not in the language since there must have been at least two deletion operations to contain the substring $a_{j'-2} b^i a_{j-1 \mod N} b^{M-i} a_{j+1 \mod N}$. This can be

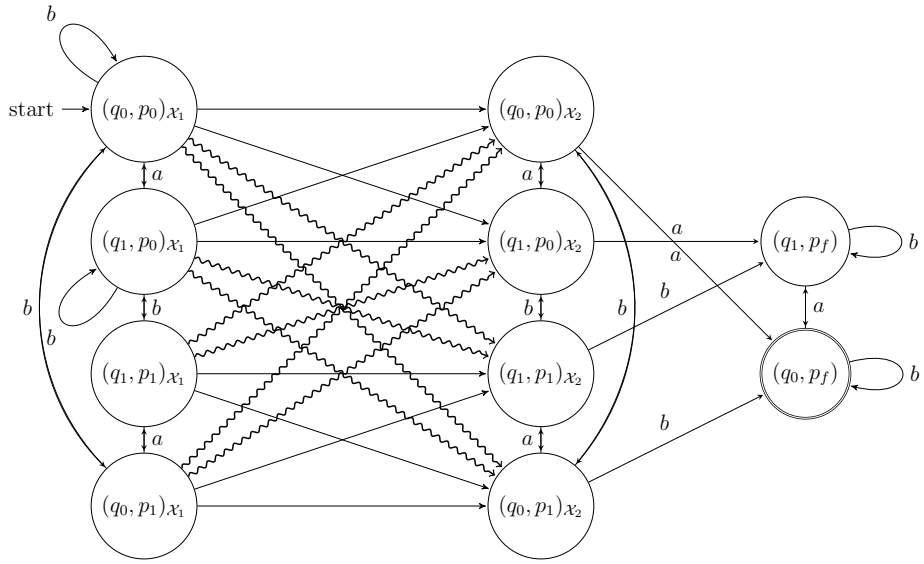

Figure 6.4: A diagram representing the automaton A_{SDD} when $L_1 = \{w \mid w \in \{a,b\}^* \text{ and } \#_a(w) \equiv 0 \mod 2\}$, and $L_2 = \{w \mid w \in \{a,b\}^* \text{ and } \#_b(w) \equiv 0 \mod 2\}$. Unlabeled transitions with straight lines are a transitions and with squiggly lines are b transitions.

seen since for $j-1 \mod N$ to be the sequential after $j'-2$, j would have to equal j'. Thus FS_2 is internally consistent with the properties of a fooling set.

If we take $(x,y) \in FS_1$ and $(x'y') \in FS_2$, then the word xy' is not in the witness language. The word xy' takes the form, $\bullet_0^{j-1} b^i a_{j+1 \mod N} a_{j-1 \mod N} b^{M-i} \bullet_{j+1 \mod N}^{N-1}$. This word is not in the witness language since for any value of j or j', there are always two deletions present in the word. Thus, the set FS_2 maintains the fooling set properties. □

In conclusion, unary site-directed deletion when operating over finite languages has a tight non-deterministic state complexity bound at N. The general unary bounds on state complexity are bound between $min(N,M)$ and $max(N,M)$.

For the general language bounds of site-directed deletion, tight bounds were not

obtained. An upper bound of $2NM + N$ was shown to be sufficient in Theorem 22. This marks an improvement of N states from the bound $2NM + 2N$ which was proposed by Cho et al. [9]. We were also able to obtain a lower bound of $2NM$ in Theorem 23. The lower bound required alphabets to be arbitrarily large. It remains an open problem to reduce the alphabet size, and to produce a tight bound for site-directed deletion.

Chapter 7

Conclusion

In this thesis, we have studied the nondeterministic state complexity of different formal language operations that are used to formalize PCR methods.

In Chapter 3, we introduced the operations of Prefix, Suffix, Infix and Outfix as language operations. For these operations we considered their nondeterministic state complexity bounds and have provided constructions to act as their upper bounds. These results are summarized in Table 7.1. Although the notions of prefix and suffix are fundamental in language theory, the state complexity of binary prefix- and suffix-operations has not been considered earlier. Note that the state complexity of prefix- and suffix-free languages [21, 13] has been extensively studied, but this is different from the binary operations considered here. These results were instrumental in developing improved bounds for site-directed insertion.

In Chapter 4, we considered the nondeterministic state complexity of site-directed insertion, and of related operations prefix- and suffix-directed insertion. An upper bound for site-directed insertion was given by Cho et al. [7]. We improve the upper bound and also prove a lower bound. The results are summarized in Table 7.2.

Table 7.1: Nondeterministic state complexities for prefix, suffix, infix and outfix operations. When the operation does not have a tight bound, we denote this by placing the operation in a inequality expression.

Operation	NSC
Prefix	$NM + 1$
Suffix	$NM + 1$
Infix	$NM + N$
Outfix	$2NM - 2N \leq \overleftarrow{^{Outfix}} \leq 3NM$

Table 7.2: Nondeterministic state complexity bounds for site-directed insertion, prefix-directed insertion and suffix directed insertion. When the operation does not have a tight bound, we denote this by placing the operation in a inequality expression.

Operation	NSC
Prefix-Directed Insertion	$2NM + N - M \leq \overleftarrow{^{PreDI}} \leq 2NM + N$
Suffix-Directed Insertion	$2NM + N - M \leq \overleftarrow{^{SufDI}} \leq 2NM + N$
Site-Directed Insertion	$3NM - M \leq \overleftarrow{^{SDI}} \leq 3NM$

In Chapter 5, we considered the state complexity of site-directed insertion in the special cases of unary languages and finite unary languages. The state complexity bound for unary site-directed insertion does leave a large gap between the upper and lower bounds. We have conjectured that the real upper bound for unary site-directed insertion is at $N + M$. These results are summarized in the Table 7.3.

In Chapter 6, we examined the unary variant of the site-directed deletion operation. The bounds on state complexity for this operation were established and are also found in Table 7.3. A general NFA construction was presented for the site-directed deletion operation containing $2NM + N$ states. This marked a state savings improvement from the construction presented by Cho et al. [9]. Furthermore, a fooling set was constructed to provide a lower bound of $2NM$ for the operation. These results

Table 7.3: Nondeterministic state complexity bounds for unary site-directed insertion and deletion. When the operation does not have a tight bound, we denote this by placing the operation in a inequality expression.

Operation	Finite Unary	Unary
Site-Directed Insertion	$N + M - 3$	$N + M - 3 \leq \overleftarrow{^{SDI}} \leq NM + N + M$
Site-Directed Deletion	N	$min(N, M) \leq \overleftarrow{^{SDD}} \leq max(N, M)$

contribute to the literature by producing a more concise representation of site-directed deletion.

7.1 Future Work

Future research could develop tight bounds for many of the operations described. The exact nondeterministic state complexities for site-directed insertion and site-directed deletion are of particular interest. The deterministic state complexity of site-directed insertion, site-directed deletion and most other operations considered here remains open. Commonly an optimal DFA construction is more involved than the corresponding NFA construction. Tight bounds for the deterministic state complexity of ordinary insertion and deletion have been established by Han et. al [19, 20].

It would be valuable to find an exact state complexity for unary site-directed insertion, as discussed in Chapter 5. It appears that unary site-directed insertion converges to accept or reject all words beyond a certain length, in certain cases convergence does not occur but a stable pattern emerges. These properties could be further explored from a number theory perspective and could potentially link generating functions to state complexity.

Bibliography

[1] LM Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, 1994.

[2] Jean-Camille Birget. Intersection and union of regular languages and state complexity. *Information Processing Letters*, 43(4):185 – 190, 1992.

[3] Jean-Camille Birget. Partial orders on words, minimal elements of regular languages, and state complexity. *Theoretical Computer Science*, 119(2):267 – 291, 1993.

[4] Janusz Brzozowski, Galina Jirásková, Bo Liu, Aayush Rajasekaran, and Marek Szykuła. On the state complexity of the shuffle of regular languages. In Cezar Câmpeanu, Florin Manea, and Jeffrey Shallit, editors, *Descriptional Complexity of Formal Systems*, pages 73–86, Cham, 2016. Springer International Publishing.

[5] C. Câmpeanu, K. Culik, Kai Salomaa, and Sheng Yu. State complexity of basic operations on finite languages. In Oliver Boldt and Helmut Jürgensen, editors, *Automata Implementation*, pages 60–70, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[6] Cezar Câmpeanu, Kai Salomaa, and Sheng Yu. Tight lower bound for the state complexity of shuffle of regular languages. *J. Autom. Lang. Comb.*, 7(3):303–310, January 2002.

[7] Da-Jung Cho, Yo-Sub Han, Timothy Ng, and Kai Salomaa. Outfix-guided insertion. *Theoretical Computer Science*, 701:70 – 84, 2017.

[8] Da Jung Cho, Yo Sub Han, Kai Salomaa, and Taylor J. Smith. Site-directed insertion: Language equations and decision problems. *Theoretical Computer Science*, 798:40–51, Dec 2019.

[9] Cho Da-Jung, Yo-Sub Han, Hwee Kim, and Kai Salomaa. Site-Directed Deletion. In *22nd International Conference on Developments in Language Theory (DLT 2018)*, Tokyo, Japan, Sep 2018.

[10] Mark J. Daley and Lila Kari. Dna computing: Models and implementations. *Comments on Theoretical Biology*, 7(3):177, 2002.

[11] Michael Domaratzki. Deletion along trajectories. *Theoretical Computer Science*, 320(2):293–313, 2004.

[12] Michael Domaratzki. Minimality in template-guided recombination. *Information and Computation*, 207(11):1209 – 1220, 2009. Special Issue: 2nd International Conference on Language and Automata Theory and Applications (LATA 2008).

[13] Hae-Sung Eom, Yo-Sub Han, Kai Salomaa, and Sheng Yu. State complexity of combined operations for prefix-free regular languages. In Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Discrete Mathematics and Computer Science. In Memoriam Alexandru Mateescu (1952-2005)*, pages 137–151.

The Publishing House of the Romanian Academy, 2014.

[14] Robert Ferens and Marek Szykuła. Complexity of bifix-free regular languages. *Theoretical Computer Science*, 787:14 – 27, 2019. Implementation and Application of Automata (CIAA 2017).

[15] Giuditta Franco and Vincenzo Manca. Algorithmic applications of xpcr. *Natural Computing: An International Journal*, 10(2):805–819, jun 2011.

[16] Yuan Gao, Nelma Moreira, Rogério Reis, and Sheng Yu. A survey on operational state complexity. *J. Autom. Lang. Comb.*, 21(4):251–310, June 2016.

[17] Ian Glaister and Jeffrey Shallit. A lower bound technique for the size of nondeterministic finite automata. *Information Processing Letters*, 59(2):75 – 77, 1996.

[18] Hermann Gruber and Markus Holzer. Finding lower bounds for nondeterministic state complexity is hard. In *Developments in Language Theory*, 2006.

[19] Yo-Sub Han, Sang-Ki Ko, Timothy Ng, and Kai Salomaa. State complexity of insertion. *International Journal of Foundations of Computer Science*, 27(07):863–878, 2016.

[20] Yo-Sub Han, Sang-Ki Ko, and Kai Salomaa. State complexity of deletion and bipolar deletion. *Acta Informatica*, 53(1):67–85, February 2016.

[21] Yo-Sub Han and Kai Salomaa. State complexity of basic operations on suffix-free regular languages. *Theoretical Computer Science*, 410(27):2537–2548, 2009.

[22] Yo-Sub Han, Yajun Wang, and Derick Wood. Infix-free regular expressions and languages. *International Journal of Foundations of Computer Science*, 17(02):379–393, 2006.

[23] Markus Holzer and Sebastian Jakobi. State complexity of chop operations on unary and finite languages. In Martin Kutrib, Nelma Moreira, and Rogério Reis, editors, *Descriptional Complexity of Formal Systems*, pages 169–182, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[24] Markus Holzer and Martin Kutrib. Nondeterministic descriptional complexity of regular languages. *International Journal of Foundations of Computer Science*, 14(06):1087–1102, 2003.

[25] Markus Holzer and Martin Kutrib. Unary language operations and their nondeterministic state complexity. In Masami Ito and Masafumi Toyama, editors, *Developments in Language Theory*, pages 162–172, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[26] Markus Holzer and Martin Kutrib. Nondeterministic finite automata—recent results on the descriptional and computational complexity. In Oscar H. Ibarra and Bala Ravikumar, editors, *Implementation and Applications of Automata*, pages 1–16, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[27] Markus Holzer and Martin Kutrib. Descriptional and computational complexity of finite automata—a survey. *Information and Computation*, 209(3):456 – 470, 2011. Special Issue: 3rd International Conference on Language and Automata Theory and Applications (LATA 2009).

[28] John E. Hopcroft and Jeff D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.

[29] Michal Hospodár, Galina Jirásková, and Peter Mlynárčik. *A Survey on Fooling Sets as Effective Tools for Lower Bounds on Nondeterministic Complexity*. 2018.

[30] Zoya Ignatova, Israel Marck Martínez Pérez, and Karl-Heinz Zimmermann. *DNA computing models*. Springer, Beijing, 2008.

[31] Lila Kari and Kalpana Mahalingam. Dna codes and their properties. In *DNA Computing*, pages 127–142, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[32] Lila Kari, Shinnosuke Seki, and Petr Sosík. *DNA Computing — Foundations and Implications*, pages 1073–1127. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[33] Lila Kari and Gabriel Thierrin. Contextual insertions/deletions and computability. *Information and Computation*, 131(1):47 – 61, 1996.

[34] Jehan Lee, Hye-Jin Lee, Myeong-Kyun Shin, and Wang-Shick Ryu. Versatile pcr-mediated insertion or deletion mutagenesis. *BioTechniques*, 36(3):398–400, 2004. PMID: 15038153.

[35] Robert Mandl. Precise bounds associated with the subset construction on various classes of nondeterministic finite automata. *7th Princeton Confrence on Information and Systems Sciences*, pages 263 – 267.

[36] Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa. *DNA Computing: New Computing Paradigms (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[37] Giovanni Pighizzini and Jeffery Shallit. Unary language operations, state complexity and jacobsthal's function. *International Journal of Foundations of Computer Science*, 13(01):145–159, 2002.

[38] Elena Pribavkina and Emanuele Rodaro. State complexity of code operators. *International Journal of Foundations of Computer Science*, 22(07):1669–1681, 2011.

[39] Akihiro Takahara and Takashi Yokomori. On the computational power of insertion-deletion systems. In Masami Hagiya and Azuma Ohuchi, editors, *DNA Computing*, pages 269–280, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[40] Derick Wood. *Theory of Computation: A Primer*. Addison-Wesley Longman Publishing Co., Inc., USA, 1987.

[41] Sheng Yu. State complexity: Recent results and open problems. *Fundam. Inform.*, 64:471–480, 01 2005.

Appendix A

Suffix-directed insertion lower bound

The extended proof for Theorem 12.

Proof. We take the following languages to construct our witness language:

$$L_1 = \{w \mid w \in \{\text{⚓}_1, a, b\}^* \text{ and } (\#_{\text{⚓}_1}(w) + \#_a(w)) \mod N \equiv 0\}, \text{ and}$$

$$L_2 = \{w \mid w \in \{\text{⚓}_2, a, b\}^* \text{ and } (\#_{\text{⚓}_2}(w) + \#_b(w)) \mod M \equiv 0\}.$$

The language L_1 recognizes strings that contain a multiple of N total ⚓_1 and a symbols and does not contain any ⚓_2 symbols. Similarly, the language L_2 recognizes strings that contain a multiple of M total ⚓_2 and b symbols and does not contain any ⚓_1 symbols. L_1 and L_2 can be recognized with NFAs with N and M states respectively. Using L_1 and L_2 we can produce the following witness language:

$$L_1 \xleftarrow{SufDI} L_2 = \{x\bar{y}y_2z \mid \#_{\text{⚓}_1}(xy_2z) + \#_a(xy_2z) \mod N \equiv 0 \text{ and}$$

$$\#_{\text{⚓}_2}(\bar{y}y_2) + \#_b(\bar{y}y_2) \mod M \equiv 0, y_2 \neq \varepsilon\}.$$

The fooling set $FS = FS_1 \cup FS_2 \cup FS_3$ contains 2NM+N-M elements in the following way.

The first set FS_1 contains NM elements and is defined as follows:

$$FS_1 = \{(a^j \anchor_2^M \anchor_2^i, \anchor_2^{M-i} \anchor_2^M a^{N-j}) \mid 0 \leq i < M, 0 \leq j < N\}$$

We can show that each tuple, when concatenated, forms a word in $L_1 \xleftarrow{SufDI} L_2$. If we take all words of the form $a^j \anchor_2^M \anchor_2^i \anchor_2^{M-i} \anchor_2^M a^{N-j}$, we can parse each word as per Definition 14 as follows: $x = a^j$, $\bar{y} = \anchor_2^M \anchor_2^{M-i} \anchor_2^i \anchor_2^M$ and $y_2 = a^{N-j}$. The substring y_2 is never empty, since $0 \leq j < N$, and the string xy_2 always contains N $a's$. The substring $\bar{y}$ always contains 3M $\anchor_2's$, which is consistent with the witness language.

We need to show that if we take $(x, y), (x', y') \in FS_1$ where $(x, y) \neq (x', y')$, then either x'y or xy' are not in the witness language. If we assume $i < i'$, then without loss of generality we can make a word $xy' = a^j \anchor_2^M \anchor_2^i \anchor_2^{M-i'} \anchor_2^M a^{N-j}$, will contain 3M+i-i' $\anchor_2's$, implying these words does not meet the language restrictions. Similarly, if we assume $j < j'$, then the word $xy' = a^j \anchor_2^M \anchor_2^i, \anchor_2^{M-i} \anchor_2^M a^{N-j'}$ will contain N+j-j' $a's$, which is greater than 0 but less than N.

The next set we consider contains (N-1)M elements and is defined as follows:

$$FS_2 = \{(\anchor_2^M \anchor_2^i a^j, b^{M-i} a^{N-j}) \mid 0 < i \leq M, 0 < j < N\}.$$

We can decompose the words made in FS_2 similarly to the ones in FS_1. If we take the words $\anchor_2^M \anchor_2^i a^j b^{M-i} a^{N-j}$, where $0 < i \leq M, 0 < j \leq N$, we can parse the words as $\bar{y} = \anchor_2^M \anchor_2^i$ and $y_2 = a^j b^{M-i} a^{N-j}$ where y_2 always contains N total $a's$ and

$\text{⚓}_1's$, and $\bar{y}y_2$ always contains $2M$ total $b's$ and $\text{⚓}_2's$.

For each pair $(x, y) \in FS_2$ and $(x', y') \in FS_2$ where either $x \neq x'$ or $y \neq y'$, the word xy' is not in the language $L_1 \xleftarrow{SufDI} L_2$. If $j < j'$, then the word $\text{⚓}_2^M \text{⚓}_2^i a^j b^{M-i} a^{N-j'}$ does not contain enough $a's$ to met the criteria from L_1. If $i < i'$, then $\text{⚓}_2^M \text{⚓}_2^i a^j b^{M-i'} a^{N-j}$ does not contain enough ⚓_2 symbols to account for the missing $b's$.

We next need to show that the fooling set properties are maintained between FS_1 and FS_2. To this end, if we take (x,y)$\in FS_2$ and (x',y')$\in FS_1$, then the word xy' is not in the witness language when $0 < j' < N$ and x'y is not in the witness language when $j' = 0$.

The word $xy' = \text{⚓}_2^M \text{⚓}_2^i a^j \text{⚓}_2^{M-i'} \text{⚓}_2^M a^{N-j'}$ is not in the witness language when $0 < j' < N$ because a^j cannot be counted towards the mod N requirements of L_1, since it must be in $\bar{y}$. Thus, the count of $a's$ is always greater than 0 but less than N. When $j' = 0$, the word $x'y = \text{⚓}_2^M \text{⚓}_2^i b^{M-i} a^{N-j}$ is never in the witness language since $0 < j < N$; also not meeting the L_1 language requirement.

The last subset of the fooling set contains N elements and is defined as follows:

$$FS_3 = \{(\text{⚓}_1^j, \text{⚓}_1^{N-j} \text{⚓}_2^M a^N) \mid 0 \leq j < N\}.$$

We can decompose each word created by the fooling set as follows: $x = \text{⚓}_1^j \text{⚓}_1^{N-j}$, $\bar{y} = \text{⚓}_2^M$ and $y_2 = a^N$. In this way, we can see $\bar{y}y_2$ always has M ⚓_2 symbols and xy_2 contains N total $a's$ and $\text{⚓}_1's$.

We next need to show that the pairs, $(x, y) \in FS_3$, maintain the fooling set properties with $(x', y') \in FS_3$ where $(x, y) \neq (x', y')$. If we take the word xy' where $j < j'$, then, each word is $\text{⚓}_1^j \text{⚓}_1^{N-j'} \text{⚓}_2^M a^N$ is never in our witness language since

there are N+j-j' total a's and ⚓_1 symbols.

If we next consider $(x, y) \in FS_3$ and $(x', y') \in FS_1$, we can show that x'y is not in the witness language. The word is defined as $a^j \text{⚓}_2^M \text{⚓}_2^i \text{⚓}_1^{N-j} \text{⚓}_2^M a^N$. This word cannot be in the witness language, since all ⚓_1 symbols can only be in the $\bar{y}$ sub-word. No ⚓_1 symbols are permitted since it is not a character in L_2. Thus, the fooling set property is maintained between FS_1 and FS_3.

Lastly, we consider $(x, y) \in FS_3$ and $(x', y') \in FS_2$. We can show that x'y is not in $L_1 \xleftarrow{SufDI} L_2$. The same argument used with the elements from FS_1 applies to the elements of FS_2. The word $x'y = \text{⚓}_2^M \text{⚓}_2^{i'} a^{j'} \text{⚓}_1^{N-j} \text{⚓}_2^M a^N$, has the symbol ⚓_1 appearing between two ⚓_2 symbols, which implies that it can't be in the witness language. □