修 士 学 位 論 文

Master's Thesis

論文題目

Thesis Title

Learning of Symbolic Weighted Finite Automata

(重み付きシンボリックオートマトンの

学習に関する研究)

提 出 者

東 北 大 学 大 学 院 情 報 科 学 研 究 科 システム情報科学 専攻

Department of System Information Sciences

Graduate School of Information Sciences, Tohoku University

氏名（Name）鈴木　海渡（Kaito Suzuki）

指　導　教　員	篠原　歩　教授
学位論文指導教員	吉仲　亮　准教授
審　査　委　員 （○印は主査）	○篠原　歩　　　教授 1 周　暁　　　　教授　　2 乾　健太郎　　　教授 3 吉仲　亮　　准教授　　4　　　　　　　教授 5　　　　　　　教授　　6　　　　　　　教授

Learning of Symbolic Weighted Finite Automata
(重み付きシンボリックオートマトンの
学習に関する研究)

Kaito Suzuki
(鈴木　海渡)

Graduate School of Information Sciences
Tohoku University, Japan

January 28, 2022

Contents

Chapter 1

Introduction

In this paper, we present learning algorithms for symbolic weighted finite automata (SWFAs) with the investigation of some essential properties of SWFAs. SWFAs can be seen as the unification of two notable extensions of classical finite automata (FAs), weighted finite automata (WFAs) and symbolic finite automata (SFAs). WFAs are finite automata to represent *formal power series* on a semiring, which are functions from a set of strings over a finite alphabet to a semiring. SFAs are an extension of FAs, where each transition edge is labeled with a *predicate*, which represents a possibly infinite subset of the alphabet compactly. By combining them, SWFAs can represent power series as with WFAs while dealing with a large or infinite alphabet efficiently as well as SFAs. Transition edges of an SWFA are labeled by functions from the alphabet to a semiring instead of predicates.

Because of their generality, SWFAs have been considered in several recent studies. Herrmann and Vogler [16] introduced SWFAs with data storage and investigated their expressiveness for some types of data storage. Alur et al. [1] considered SWFAs with nesting operations and parallel execution, and proposed an efficient evaluation algorithm for them. Jaksic et al. [18, 19] and Waga [26] used SWFAs to develop underlying algorithms for runtime verification of cyber-physical systems. However, none of the prior works explore essential properties of SWFAs in detail. In addition, the learnability of SWFAs has not been studied yet.

The problem of learning FAs has been studied for a long time. In this field, one of the central research topics is *exact learning*. One of the most famous exact learning algorithm of FAs is L_* algorithm proposed by Angluin [2]. L_* algorithm learns deterministic FAs with an oracle, called a *minimally adequate teacher (MAT)*, which answers *membership queries (MQs)* and *equivalence queries (EQs)* from the learner. Bergadano and Varricchio [8] extended L^* for learning WFAs and Bisht et al. [9] improved the algorithm. The learning of SFAs under the MAT model has also been studied. Drews and D'Antoni [13] showed that deterministic SFAs can be learned if partitions of Σ are inferable from finite examples. Argyros and D'Antoni [3] proposed a more powerful algorithm that learns deterministic SFAs under the assumption that predicates on transition edges are also MAT learnable. This work was extended for learning non-deterministic SFAs by Chubachi et al. [11].

We propose a query learning algorithm for SWFAs, which is a combination of the ones for WFAs [9] and SFAs [3]. The query complexity of our algorithm is polynomial in the minimal number of states to represent the target power series, the length of the longest counterexample against EQs, and the number of queries required to learn functions on transition edges. By proposing this algorithm, we prove that SWFAs are efficiently MAT-learnable if functions on transition edges are also efficiently MAT-learnable. In addition, our experiments show that the practical performance of our algorithm is much more efficient than the theoretical worst-case complexity.

Another central research topic of learning FAs is *approximate learning*. In particular, there are many studies on approximate learning for WFAs, which output weights instead of binary outputs like classical FAs. The most famous approximate learning algorithm of WFAs is the *spectral learning* algorithm [4, 5]. This algorithm utilizes the *singular value decomposition (SVD)* to efficiently restore a WFA from a dataset. We extend this algorithm to the case of SWFAs and evaluate the performance by experiments. Our algorithm reveals that SWFAs can achieve almost the same performance compared to

WFAs, with much fewer parameters and data.

This paper also investigates some essential properties of SWFAs. We give a novel, clear definition of SWFAs considering a class of functions $\mathscr{G}$ on transition edges of SWFAs. We find that some theoretical properties and learnability hold under an essential assumption — *$\mathscr{G}$ is closed under linear combination.* We call the representation class of SWFAs $\mathscr{G}$-recognizable. To clarify the representation range of $\mathscr{G}$-recognizable, we define a set of operations called $\mathscr{G}$-rational and prove the *Kleene–Schützenberger theorem for SWFAs*, the equality of $\mathscr{G}$-recognizable and $\mathscr{G}$-rational. This theorem corresponds to the equality of the representation class of FAs and regular expression, which is a well-known result shown by Kleene [20], and later extended for WFAs by Schützenberger [25]. We also show that minimization and equivalence checking of SWFAs can be achieved efficiently. Our proposed algorithm for minimization and equivalence checking works in cubic time to the number of states of given SWFAs.

Chapter 2

Preliminaries

2.1 Semiring and Field

Symbolic weighted finite automata carry weights defined over a *semiring* while reading an input string. When we consider learning of SWFAs, we assume the weights are actually in a *field* because this restriction allows helpful manipulations for learning. In this section, we introduce the definitions of semiring and field.

For a set S, a binary operation $\cdot$ over S, and $e \in S$, the tuple $(S, \cdot, e)$ is a *monoid* when it satisfies the following conditions.

$$\begin{aligned} \text{associative law} : (a \cdot b) \cdot c = a \cdot (b \cdot c),\ \text{ for all } a, b, c \in S \\ \text{identity element} : e \cdot a = a \cdot e = a,\ \text{ for all } a \in S \end{aligned}$$

When $\cdot$ is *commutative*, that is, $a \cdot b = b \cdot a$ for all $a, b \in S$, such a monoid is called a *commutative monoid.*

A tuple $(\mathbb{S}, +, \star, 0, 1)$ is a *semiring* if $(\mathbb{S}, +, 0)$ is a commutative monoid, $(\mathbb{S}, \star, 1)$ is a monoid, and the tuple satisfies the following conditions.

$$\begin{aligned}\text{distributive law} : a * (b + c) &= (a * b) + (a * c),\\ (a + b) * c &= (a * c) + (b * c), \text{ for all } a, b, c \in \mathbb{S}\\ \text{annihilator} : 0 * a &= a * 0 = 0, \text{ for all } a \in \mathbb{S}\end{aligned}$$

Intuitively, a semiring is a set that allows multiplication and commutative addition. The symbol $*$ is often omitted like $ab = a * b$. The set of natural numbers $\mathbb{N}$ (including 0) is a simple example of semiring. We sometimes use $\mathbb{S}$ to represent a semiring by omitting other tuple elements if there is no risk of confusion.

A tuple $(\mathbb{F}, +, *, 0, 1)$ is a *field* if $(\mathbb{F}, +, 0)$ is a commutative monoid and any $a \in \mathbb{F}$ allows an additive inverse $-a$ which satisfies $a + (-a) = 0$, $(\mathbb{F}, *, 1)$ is a commutative monoid and any $a \in \mathbb{F} \setminus \{0\}$ allows a multiplicative inverse a^{-1} which satisfies $a * a^{-1} = 1$, and distributive law as with semiring. Following the custom, we add the condition $1 \neq 0$ to exclude the trivial field. Intuitively, a field is a set that allows four arithmetic operations. We can interpret field as an extension of semiring introducing subtraction and division by adding inverse elements. The set of rational numbers $\mathbb{Q}$ and real numbers $\mathbb{R}$ are typical examples of field. We also use $\mathbb{F}$ to represent a field as shorthand.

2.2 Notation

The set of all strings over an alphabet Σ is denoted by Σ^*. The empty string is denoted by ϵ. The set of all functions from X to Y is denoted by Y^X.

We consider possibly infinite matrices throughout this paper. A matrix over Y whose rows and columns are indexed by sets P and S, respectively, is identified with a function from $P \times S$ to Y. The element of a matrix $A \in Y^{P \times S}$ indexed by $p \in P$ and $s \in S$ is denoted by $A[p, s]$. A vector may be seen as a special case of a matrix whose column index set is a singleton $S = \{s\}$, where we often drop s and simply refer to the element $A[p, s]$ by

$A[p]$ regarding $A \in Y^P$. A pair (P', S') of subsets $P' \subseteq P$ and $S' \subseteq S$ is called a *mask* and the (P', S')-sub-block of $A \in Y^{P \times S}$ with a row index set P' and a column index set S' is denoted as $A_{(P',S')} \in Y^{P' \times S'}$. Particularly when $P' = P$ and S' is a singleton set $\{s\}$, the matrix $A_{(P,\{s\})}$ is often denoted by $A[\cdot, s]$ and is called the *column vector* of A indexed by $s \in S$. The *row vector* of A indexed by $p \in P$ is symmetrically defined and denoted by $A[p, \cdot]$. We also use $\vec{a}$ to clarify that a is a column vector. The transpose of $A \in Y^{P \times S}$, denoted by $A^\intercal \in Y^{S \times P}$, exchanges the rows and columns of A. We often express a matrix as a table ordering the row and column indices arbitrarily. We use I to represent an identity matrix, O to represent a zero matrix, and $\vec{0}$ to represent a zero vector.

When Y is a semiring, the multiplication $AB \in Y^{P \times S}$ of two matrices $A \in Y^{P \times Q}$ and $B \in Y^{Q \times S}$ is defined by $(AB)[p, s] = \sum_{q \in Q} A[p, q]B[q, s]$ for all $p \in P$ and $s \in S$ provided that Q is finite. When Y is a field, the rank of $A \in Y^{P \times S}$ is defined in the usual way and denoted by $\mathrm{rank}(A)$. A mask (P', S') is called a *basis* if $\mathrm{rank}(A_{(P',S')}) = \mathrm{rank}(A)$. The basis (P', S') is *minimal* if $|P'| = |S'| = \mathrm{rank}(A)$. A mask (P', S') is *non-singular* when $A_{(P',S')}$ is non-singular. Then, there is a unique matrix called the inverse $A^{-1}_{(P',S')}$ of $A_{(P',S')}$ such that $A_{(P',S')}A^{-1}_{(P',S')} = I$. A minimal basis is always non-singular since a matrix $A_{(P',S')}$ is non-singular if and only if $|P'| = |S'| = \mathrm{rank}(A_{(P',S')})$. However, the reverse is not always true. That is, there may be a mask that is non-singular but not a basis.

Chapter 3

Symbolic Weighted Finite Automata

Symbolic weighted finite automata (SWFAs) combine symbolic finite automata (SFAs) and weighted finite automata (WFAs) to represent formal power series over a possibly infinite alphabet. SFAs are defined over a Boolean algebra, which has *predicates* (or *guards*), finite descriptions to represent possibly infinite subsets of the alphabet. SFAs use predicates as labels of transition edges. Likewise, SWFAs have edges labeled by finitely describable functions, called *guard functions*, in a fixed class $\mathscr{G}$ from the alphabet Σ to a semiring $\mathbb{S}$.

3.1 Definition

Definition 3.1 (Symbolic Weighted Finite Automata, SWFAs)**.** A *symbolic weighted finite automaton (SWFA)* $\mathcal{A}$ over $\mathscr{G}$ is a tuple $(\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta)$, where $\mathscr{G} \subseteq \mathbb{S}^{\Sigma}$ is a class of guard functions from Σ to $\mathbb{S}$, Q is a finite set of states, $\vec{\alpha} \in \mathbb{S}^{Q}$ is a vector of initial weights, $\vec{\beta} \in \mathbb{S}^{Q}$ is a vector of final weights, and $\Delta \in \mathscr{G}^{Q \times Q}$ is a transition relation. For any $x \in \Sigma$, the transition relation for x is represented by a matrix $\Delta_x \in \mathbb{S}^{Q \times Q}$, i.e., $\Delta_x[q, q'] = \Delta[q, q'](x)$ for all $(q, q') \in Q^2$. Δ_x can be extended for strings $w \in \Sigma^*$ as $\Delta_w = \Delta_{x_1}\Delta_{x_2}\ldots\Delta_{x_l}$, where $w = x_1 x_2 \ldots x_l$ with $x_i \in \Sigma$. The formal power series $f_{\mathcal{A}}$ represented by $\mathcal{A}$ is defined by $f_{\mathcal{A}}(w) = \vec{\alpha}^{\intercal}\Delta_w\vec{\beta}$ for all $w \in \Sigma^*$ and called *$\mathscr{G}$-recognizable.* The set of all $\mathscr{G}$-recognizable power series is denoted by *$\mathscr{G}$-Rec*. We allow a special SWFA with no states, which represents the zero constant series, called the *zero SWFA*.

If $\mathbb{S}$ is the two-element Boolean algebra, SWFAs coincide with SFAs. For the class $\mathscr{G}^{\mathrm{wfa}}$ of all guard functions from a *finite* alphabet Σ to a semiring $\mathbb{S}$, SWFAs over $\mathscr{G}^{\mathrm{wfa}}$ are exactly WFAs, except that usually the transition relations of WFAs are described by enumeration. That is, the transition relation of a WFA is $|\Sigma|$ matrices in $\mathbb{S}^{Q\times Q}$.

Throughout this paper, we make the following assumption.

Assumption 1. *Any guard function class $\mathscr{G} \subseteq \mathbb{S}^{\Sigma^*}$ is closed under linear combination. That is, for any $a, b \in \mathbb{S}$ and $g_1, g_2 \in \mathscr{G}$, there is $g \in \mathscr{G}$ such that*

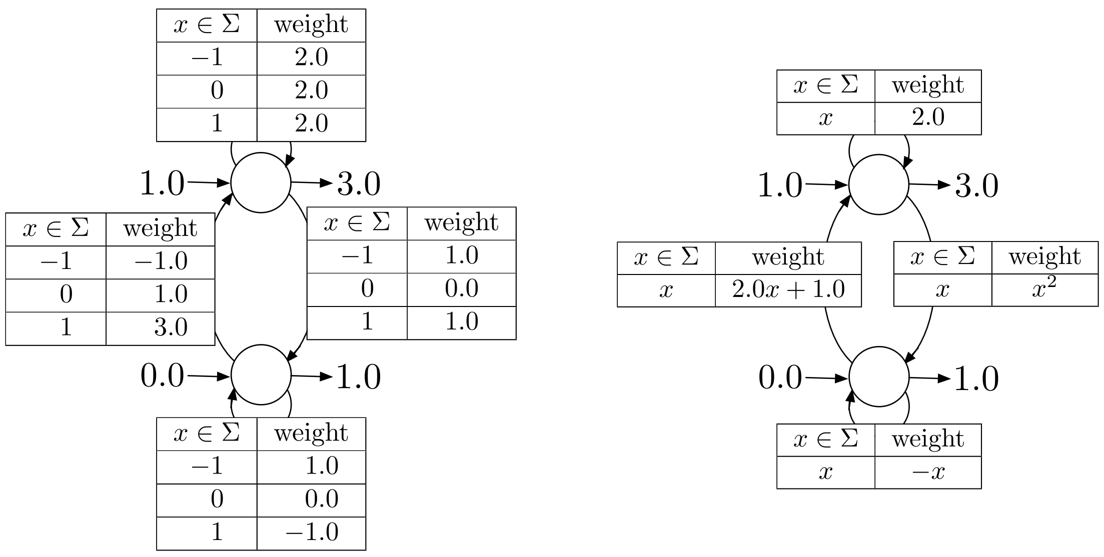
$$g(w) = ag_1(w) + bg_2(w) \quad \text{for all } w \in \Sigma^*.$$

This assumption may be seen as the weighted counterpart of the Boolean closure property of predicates in SFAs. We can prove many fruitful results under the assumption, including the Kleene–Schützenberger theorem for SWFAs, decidability of equivalence, and learnability.

Figure 3.1 shows toy examples of a WFA and an SWFA representing the same power series $f\colon \{-1, 0, 1\}^* \to \mathbb{R}$, where $\mathbb{R}$ is the set of real numbers. The SWFA is readily applicable for extending the alphabet to the integer set with no change.

3.2 The Kleene–Schützenberger Theorem for SWFAs

Kleene [20] shows that regular expression and the class of languages recognized by a classical finite automata are equivalent. Schützenberger [25] extended this result to the case of WFAs. In this section, we derive the compatible result for SWFAs. That is, we define $\mathscr{G}$-rational power series, which is symbolic weighted version of regular expression, and prove the equivalence to $\mathscr{G}$-recognizable power series. We call this equivalence the *Kleene–Schützenberger theorem for SWFAs.*



$x \in \Sigma$	weight
-1	2.0
0	2.0
1	2.0

$x \in \Sigma$	weight
-1	-1.0
0	1.0
1	3.0

$x \in \Sigma$	weight
-1	1.0
0	0.0
1	1.0

$x \in \Sigma$	weight
-1	1.0
0	0.0
1	-1.0

(a) WFA

$x \in \Sigma$	weight
x	2.0

$x \in \Sigma$	weight
x	$2.0x + 1.0$

$x \in \Sigma$	weight
x	x^2

$x \in \Sigma$	weight
x	$-x$

(b) SWFA

Figure 3.1: (a) An example WFA on $\Sigma = \{-1, 0, 1\}$. Numbers attached to the input arrow shows the initial vector $\vec{\alpha}$. Numbers attached to the output arrow shows the final vector $\vec{\beta}$. The table labeling each edge from a state q to other state q' shows $\Delta_x[q, q']$ for each $x \in \Sigma$. (b) An example SWFA representing the same power series as (a). The label on each edge is the assigned guard function from Σ to $\mathbb{R}$.

As the counterpart of singleton language, first we introduce *monomials*. A monomial r_g for $g \in \mathscr{G}$ is defined by

$$r_g(x) = g(x) \text{ for all } x \in \Sigma, r_g(w) = 0 \text{ for all } w \in \Sigma^* \setminus \Sigma.$$

Note that $r_g(\epsilon) = 0$.

Next we introduce the following operations on $\mathbb{S}^{\Sigma^*}$.

For $r \in \mathbb{S}^{\Sigma^*}$ and $a \in \mathbb{S}$, the (left and right) *scalar multiplication* is defined by

$$(ar)(w) = ar(w) \text{ for all } w \in \Sigma^*,$$

$$(ra)(w) = r(w)a \text{ for all } w \in \Sigma^*.$$

For $r_1, r_2 \in \mathbb{S}^{\Sigma^*}$, the *addition* is defined by

$$(r_1 + r_2)(w) = r_1(w) + r_2(w) \text{ for all } w \in \Sigma^*.$$

For $r_1, r_2 \in \mathbb{S}^{\Sigma^*}$, the *Cauchy product* is defined by

$$(r_1 \cdot r_2)(w) = \sum_{w=uv} r_1(u) r_2(v) \text{ for all } w \in \Sigma^*.$$

Since any finite word w has only finitely many factorizations into u and v, the right-hand of this equation is the sum of finitely many elements. Therefore, this operation is *well-defined* in the sense that it does not diverge.

For $r \in \mathbb{S}^{\Sigma^*}$, let $r^n = r^{n-1} \cdot r$ and $r^0 \in \mathbb{S}^{\Sigma^*}$ where $r^0(w) = 1$ if $w = \epsilon$ otherwise 0. the *Kleene star* is defined only for r such that $r(\epsilon) = 0$. We call such r *proper.*

$$r^* = \sum_{i \geq 0} r^i$$

Since r is proper, for $i > |w|$, $r^i(w) = 0$. Therefore, this operation is sum of finitely many elements and well-defined.

Using these operations, we can define $\mathcal{G}$-rational power series.

Definition 3.2. A power series from $\mathbb{S}^{\Sigma^*}$ is *$\mathcal{G}$-rational* if it can be constructed from the monomials r_g for $g \in \mathcal{G}$ by (left and right) scalar multiplication, addition, Cauchy-product, and Kleene star. The set of all $\mathcal{G}$-rational power series is denoted by *$\mathcal{G}$-Rat.*

Theorem 3.3 (The Kleene–Schützenberger theorem for SWFAs)**.**

$$\mathcal{G}\text{-}Rec = \mathcal{G}\text{-}Rat.$$

To prove the theorem, we first show $\mathcal{G}$-Rat $\subseteq$ $\mathcal{G}$-Rec by introducing the following lemmas, which define an SWFA corresponding to each $\mathcal{G}$-rational operation.

Lemma 3.4 (monomials)**.** *For any $g \in \mathscr{G}$, there is an SWFA $\mathcal{A}$ such that $f_{\mathcal{A}}(x) = g(x)$ for all $x \in \Sigma, f_{\mathcal{A}}(w) = 0$ for all $w \in \Sigma^* \setminus \Sigma$.*

Proof: Let an SWFA $\mathcal{A} = (\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta)$ where

$$\begin{aligned} Q &= (q_1, q_2), \\ \vec{\alpha} &= \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \\ \vec{\beta} &= \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \\ \Delta &= \begin{pmatrix} 0 & g \\ 0 & 0 \end{pmatrix}. \end{aligned}$$

Note that guard functions output constants are in $\mathscr{G}$ by Assumption 1.

This SWFA satisfies the definition of monomials since

$$\begin{aligned} &f_{\mathcal{A}}(\epsilon) = \vec{\alpha}^\top \vec{\beta} = 0, \\ &f_{\mathcal{A}}(x) = \vec{\alpha}^\top \Delta_x \vec{\beta} = g(x), \text{ where } x \in \Sigma, \\ &f_{\mathcal{A}}(w) = \vec{\alpha}^\top \Delta_{x_1} \cdots \Delta_{x_l} \vec{\beta} = \vec{\alpha}^\top O \vec{\beta} = 0, \text{ where } w = x_1 \cdots x_l \in \Sigma^*, l \geq 2. \end{aligned}$$

■

Lemma 3.5 ((left and right) scalar multiplication)**.** *For any SWFA $\mathcal{A}$ and $a \in \mathbb{S}$, there is an SWFA $\mathcal{A}'$ such that $f_{\mathcal{A}'} = a f_{\mathcal{A}}$ and $\mathcal{A}''$ such that $f_{\mathcal{A}''} = f_{\mathcal{A}} a$.*

Proof: For an SWFA $\mathcal{A} = (\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta)$, let SWFAs $\mathcal{A}' = (\mathscr{G}, Q, a\vec{\alpha}, \vec{\beta}, \Delta)$ and $\mathcal{A}'' = (\mathscr{G}, Q, \vec{\alpha}, \vec{\beta} a, \Delta)$. Here, $a\vec{\alpha}$ and $\vec{\beta} a$ are pointwise scalar multiplications. Then,

$$\begin{aligned} f_{\mathcal{A}'}(w) &= a\vec{\alpha}^\top \Delta_w \vec{\beta} = a f_{\mathcal{A}}(w) \text{ for all } w \in \Sigma^*, \\ f_{\mathcal{A}''}(w) &= \vec{\alpha}^\top \Delta_w \vec{\beta} a = f_{\mathcal{A}}(w) a \text{ for all } w \in \Sigma^*. \end{aligned}$$

■

Lemma 3.6 (addition)**.** *For any SWFAs $\mathcal{A}'$ and $\mathcal{A}''$, there is an SWFA $\mathcal{A}$ such that $f_{\mathcal{A}} = f_{\mathcal{A}'} + f_{\mathcal{A}''}$.*

Proof: For SWFAs $\mathcal{A}' = (\mathscr{G}, Q', \vec{\alpha}', \vec{\beta}', \Delta')$ and $\mathcal{A}'' = (\mathscr{G}, Q'', \vec{\alpha}'', \vec{\beta}'', \Delta'')$, we can assume $Q' \cap Q'' = \varnothing$ without loss of generality. Let an SWFA $\mathcal{A} = (\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta)$ where

$$\begin{aligned} Q &= Q' \cup Q'', \\ \vec{\alpha} &= \begin{pmatrix} \vec{\alpha}' \\ \vec{\alpha}'' \end{pmatrix}, \\ \vec{\beta} &= \begin{pmatrix} \vec{\beta}' \\ \vec{\beta}'' \end{pmatrix}, \\ \Delta &= \begin{pmatrix} \Delta' & O \\ O & \Delta'' \end{pmatrix}. \end{aligned}$$

Then,

$$\begin{aligned} f_{\mathcal{A}}(w) &= \begin{pmatrix} \vec{\alpha}'^{\top} \vec{\alpha}''^{\top} \end{pmatrix} \begin{pmatrix} \Delta'_{x_1} & O \\ O & \Delta''_{x_1} \end{pmatrix} \begin{pmatrix} \Delta'_{x_2} & O \\ O & \Delta''_{x_2} \end{pmatrix} \cdots \begin{pmatrix} \Delta'_{x_l} & O \\ O & \Delta''_{x_l} \end{pmatrix} \begin{pmatrix} \vec{\beta}' \\ \vec{\beta}'' \end{pmatrix} \\ &= \begin{pmatrix} \vec{\alpha}'^{\top} \vec{\alpha}''^{\top} \end{pmatrix} \begin{pmatrix} \Delta'_{x_1 x_2 \dots x_l} & O \\ O & \Delta''_{x_1 x_2 \dots x_l} \end{pmatrix} \begin{pmatrix} \vec{\beta}' \\ \vec{\beta}'' \end{pmatrix} \\ &= \vec{\alpha}'^{\top} \Delta'_w \vec{\beta}' + \vec{\alpha}''^{\top} \Delta''_w \vec{\beta}'' \\ &= f_{\mathcal{A}'}(w) + f_{\mathcal{A}''}(w) \end{aligned}$$

for all $w = x_1 x_2 \dots x_l \in \Sigma^*$. ■

Lemma 3.7 (Cauchy product)**.** *For any SWFAs $\mathcal{A}'$ and $\mathcal{A}''$, there is an SWFA $\mathcal{A}$ such that $f_{\mathcal{A}} = f_{\mathcal{A}'} \cdot f_{\mathcal{A}''}$.*

Proof: For SWFAs $\mathcal{A}' = (\mathscr{G}, Q', \vec{\alpha}', \vec{\beta}', \Delta')$ and $\mathcal{A}'' = (\mathscr{G}, Q'', \vec{\alpha}'', \vec{\beta}'', \Delta'')$, we can assume $Q' \cap Q'' = \varnothing$ without loss of generality. Let an SWFA $\mathcal{A} = (\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta)$ where

$$\begin{aligned}
Q &= Q' \cup Q'', \\
\vec{\alpha} &= \begin{pmatrix} \vec{\alpha}' \\ \vec{0} \end{pmatrix}, \\
\vec{\beta} &= \begin{pmatrix} \vec{\beta}'\vec{\alpha}''^{\mathsf{T}}\vec{\beta}'' \\ \vec{\beta}'' \end{pmatrix}, \\
\Delta &= \begin{pmatrix} \Delta' & \vec{\beta}'\vec{\alpha}''^{\mathsf{T}}\Delta'' \\ O & \Delta'' \end{pmatrix}.
\end{aligned}$$

Note that guard functions of Δ is in $\mathscr{G}$ by Assumption 1.

Then,

$$\begin{aligned}
f_{\mathcal{A}}(w) &= \begin{pmatrix} \vec{\alpha}'^{\mathsf{T}} & \vec{0}^{\mathsf{T}} \end{pmatrix} \begin{pmatrix} \Delta'_{x_1} & \vec{\beta}'\vec{\alpha}''^{\mathsf{T}}\Delta''_{x_1} \\ O & \Delta''_{x_1} \end{pmatrix} \begin{pmatrix} \Delta'_{x_2} & \vec{\beta}'\vec{\alpha}''^{\mathsf{T}}\Delta''_{x_2} \\ O & \Delta''_{x_2} \end{pmatrix} \cdots \begin{pmatrix} \Delta'_{x_l} & \vec{\beta}'\vec{\alpha}''^{\mathsf{T}}\Delta''_{x_l} \\ O & \Delta''_{x_l} \end{pmatrix} \begin{pmatrix} \vec{\beta}'\vec{\alpha}''^{\mathsf{T}}\vec{\beta}'' \\ \vec{\beta}'' \end{pmatrix} \\
&= \begin{pmatrix} \vec{\alpha}'^{\mathsf{T}} & \vec{0}^{\mathsf{T}} \end{pmatrix} \begin{pmatrix} \Delta'_{x_1 x_2 \dots x_l} & \vec{\beta}'\vec{\alpha}''^{\mathsf{T}}\Delta''_{x_1 x_2 \dots x_l} + \sum_{1 \le j \le l-1} \Delta'_{x_1 \dots x_j}\vec{\beta}'\vec{\alpha}''^{\mathsf{T}}\Delta''_{x_{j+1} \dots x_l} \\ O & \Delta''_{x_1 x_2 \dots x_l} \end{pmatrix} \begin{pmatrix} \vec{\beta}'\vec{\alpha}''^{\mathsf{T}}\vec{\beta}'' \\ \vec{\beta}'' \end{pmatrix} \\
&= \begin{pmatrix} \vec{\alpha}'^{\mathsf{T}}\Delta'_w & \vec{\alpha}'^{\mathsf{T}} \sum_{w=uv, u \ne w} \Delta'_u \vec{\beta}'\vec{\alpha}''^{\mathsf{T}}\Delta''_v \end{pmatrix} \begin{pmatrix} \vec{\beta}'\vec{\alpha}''^{\mathsf{T}}\vec{\beta}'' \\ \vec{\beta}'' \end{pmatrix} \\
&= \sum_{w=uv} \vec{\alpha}'^{\mathsf{T}}\Delta'_u \vec{\beta}'\vec{\alpha}''^{\mathsf{T}}\Delta''_v \vec{\beta}'' \\
&= \sum_{w=uv} f_{\mathcal{A}'}(u) f_{\mathcal{A}''}(v)
\end{aligned}$$

for all $w = x_1 x_2 \dots x_l \in \Sigma^*$. ■

Lemma 3.8 (Kleene star). *For any SWFA $\mathcal{A}$, there is an SWFA $\mathcal{A}^*$ such that $f_{\mathcal{A}^*} = f^*_{\mathcal{A}}$.*

Proof: For SWFA $\mathcal{A} = (\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta)$, using an additional state $q^* \notin Q$, let an SWFA

$\mathcal{A}^* = (\mathscr{G}, Q^*, \vec{\alpha}^*, \vec{\beta}^*, \Delta^*)$ where

$$\begin{aligned}
Q^* &= \{q^*\} \cup Q, \\
\vec{\alpha}^* &= \begin{pmatrix} 1 \\ \vec{0} \end{pmatrix}, \\
\vec{\beta}^* &= \begin{pmatrix} 1 \\ \vec{0} \end{pmatrix}, \\
\Delta^* &= \begin{pmatrix} \vec{\alpha}^\top \Delta \vec{\beta} & \vec{\alpha}^\top \Delta \\ \Delta \vec{\beta} & \Delta \end{pmatrix}.
\end{aligned}$$

Note that guard functions of Δ is in $\mathscr{G}$ by Assumption 1.

Then,

$$\begin{aligned}
f_{\mathcal{A}^*}(w) &= \begin{pmatrix} 1 & \vec{0}^\top \end{pmatrix} \begin{pmatrix} \vec{\alpha}^\top \Delta_{x_1} \vec{\beta} & \vec{\alpha}^\top \Delta_{x_1} \\ \Delta_{x_1} \vec{\beta} & \Delta_{x_1} \end{pmatrix} \begin{pmatrix} \vec{\alpha}^\top \Delta_{x_2} \vec{\beta} & \vec{\alpha}^\top \Delta_{x_2} \\ \Delta_{x_2} \vec{\beta} & \Delta_{x_2} \end{pmatrix} \cdots \begin{pmatrix} \vec{\alpha}^\top \Delta_{x_l} \vec{\beta} & \vec{\alpha}^\top \Delta_{x_l} \\ \Delta_{x_l} \vec{\beta} & \Delta_{x_l} \end{pmatrix} \begin{pmatrix} 1 \\ \vec{0} \end{pmatrix} \\
&= \begin{pmatrix} 1 & \vec{0}^\top \end{pmatrix} \left(\begin{array}{c|c} \begin{matrix} \vec{\alpha}^\top \Delta_{x_1 x_2 \dots x_l} \vec{\beta} \\ + \vec{\alpha}^\top \Delta_{x_1} \vec{\beta} \vec{\alpha}^\top \Delta_{x_2 \dots x_l} \vec{\beta} + \vec{\alpha}^\top \Delta_{x_1 x_2} \vec{\beta} \vec{\alpha}^\top \Delta_{x_3 \dots x_l} \vec{\beta} + \cdots + \vec{\alpha}^\top \Delta_{x_1 \dots x_{l-1}} \vec{\beta} \vec{\alpha}^\top \Delta_{x_l} \vec{\beta} \\ \vdots \\ + \sum_{w = u_1 u_2 \dots u_i, u_j \neq \epsilon, 1 \le j \le i} \prod_{1 \le k \le i} \vec{\alpha}^\top \Delta_{u_k} \vec{\beta} \\ \vdots \\ + \vec{\alpha}^\top \Delta_{x_1} \vec{\beta} \vec{\alpha}^\top \Delta_{x_2} \vec{\beta} \cdots \vec{\alpha}^\top \Delta_{x_l} \vec{\beta} \end{matrix} & - \\ \hline - & - \end{array} \right) \begin{pmatrix} 1 \\ \vec{0} \end{pmatrix} \\
&= \begin{pmatrix} 1 & \vec{0}^\top \end{pmatrix} \left(\begin{array}{c|c} \begin{matrix} f_{\mathcal{A}}(w) \\ + f_{\mathcal{A}}^2(w) \\ \vdots \\ + f_{\mathcal{A}}^i(w) \\ \vdots \\ + f_{\mathcal{A}}^l(w) \end{matrix} & - \\ \hline - & - \end{array} \right) \begin{pmatrix} 1 \\ \vec{0} \end{pmatrix} \\
&= \sum_{i \ge 1} f_{\mathcal{A}}^i(w)
\end{aligned}$$

for all $w = x_1 x_2 \ldots x_l \in \Sigma^* \setminus \{\epsilon\}$. "–" in the matrix means omitted formulas since they are irrelevant to the final result. Considering $f_{\mathcal{A}^*}(\epsilon) = \vec{\alpha}^\top \vec{\beta} = 1 = f^0_{\mathcal{A}}(\epsilon)$ leads

$$f_{\mathcal{A}^*} = f^0_{\mathcal{A}} + \sum_{i \geq 1} f^i_{\mathcal{A}} = \sum_{i \geq 0} f^i_{\mathcal{A}} = f^*_{\mathcal{A}}.$$

■

Next we show $\mathscr{G}$-Rec $\subseteq$ $\mathscr{G}$-Rat. First we define some useful notation. For a matrix of power series $R \in (\mathbb{S}^{\Sigma^*})^{P \times S}$ and $w \in \Sigma^*$, let $R(w)$ be the matrix of the semiring such that $R(w)[p,s] = R[p,s](w)$ for all $(p,s) \in P \times S$. The Cauchy product $R \in (\mathbb{S}^{\Sigma^*})^{P \times S}$ of $R_1 \in (\mathbb{S}^{\Sigma^*})^{P \times Q}$ and $R_2 \in (\mathbb{S}^{\Sigma^*})^{Q \times S}$ is $R(w) = \sum_{w=uv} R_1(u) R_2(v)$ for all $w \in \Sigma^*$ and written by $R = R_1 \cdot R_2$. The Kleene star $R^* \in (\mathbb{S}^{\Sigma^*})^{Q \times Q}$ of $R \in (\mathbb{S}^{\Sigma^*})^{Q \times Q}$ is defined only for proper R, that is, $R(\epsilon) = O$. Let $R^n = R \cdot R^{n-1}$ and $R^0(w) = I$ when $w = \epsilon$, otherwise O. Then $R^* = \sum_{i \geq 0} R^i$.

The following lemmas and corollary are useful.

Lemma 3.9. *Let $R_1 \in (\mathbb{S}^{\Sigma^*})^{P \times Q}$ and $R_2 \in (\mathbb{S}^{\Sigma^*})^{Q \times S}$ be matrices of $\mathscr{G}$-rational power series. Then, $R = R_1 \cdot R_2$ is also a matrix of $\mathscr{G}$-rational power series.*

Proof:

$$\begin{aligned} R(w) &= (R_1 \cdot R_2)(w) \\ &= \sum_{w=uv} R_1(u) R_2(v) \end{aligned} \tag{3.1}$$

for all $w \in \Sigma^*$.

That is,

$$\begin{aligned} R[p,s](w) &= \sum_{w=uv} \sum_{q \in Q} R_1[p,q](u) R_2[q,s](v) \\ &= \sum_{q \in Q} (R_1[p,q] \cdot R_2[q,s])(w) \end{aligned} \tag{3.2}$$

for all $p \in P$, $s \in S$ and $w \in \Sigma^*$.

Note that $\cdot$ in Equation (3.1) means the Cauchy product between matrices of power series and $\cdot$ in Equation (3.2) means the Cauchy product between power series.

Since the Cauchy product between power series is in $\mathscr{G}$-rational operations, Equation (3.2) shows that each element of R can be represented by sum of $\mathscr{G}$-rational power series. Thus, R is a matrix of $\mathscr{G}$-rational power series. ∎

Lemma 3.10. *Let $R \in (\mathbb{S}^{\Sigma^*})^{Q \times Q}$ be a matrix of power series and $\vec{u} \in (\mathbb{S}^{\Sigma^*})^{Q}$ be a vector of power series. Let us consider the equation about an unknown vector of power series $\vec{X}$,*

$$\vec{X} = R \cdot \vec{X} + \vec{u}.$$

If R is proper, the equation has a unique solution $\vec{X} = R^* \cdot \vec{u}$.

Proof: Substituting $R^* \cdot \vec{u}$ into the equation shows $R^* \cdot \vec{u}$ is a solution.

Conversely, if $\vec{v}$ is a solution of the equation $\vec{X} = R \cdot \vec{X} + \vec{u}$, we have

$$\vec{v} = \vec{u} + R \cdot \vec{v} = \vec{u} + R \cdot \vec{u} + R^2 \cdot \vec{v} = \cdots = \sum_{0 \le i < n} R^i \cdot \vec{u} + R^n \cdot \vec{v},$$

for all integers n. Since $\lim_{n \to \infty} R^n = O$ and $\lim_{n \to \infty} \sum_{0 \le i < n} R^i = R^*$, $\vec{v} = R^* \cdot \vec{u}$. ∎

Corollary 3.11. *Let r and u be a power series of $\mathbb{S}^{\Sigma^*}$. Let us consider the equation about an unknown power series X,*

$$X = r \cdot X + u.$$

If r is proper, the equation has a unique solution $X = r^* \cdot u$.

Using these lemmas and corollary, we can prove the following lemmas.

Lemma 3.12. *Let Δ be an $n \times n$ square matrix consisting of $\mathscr{G}$-rational power series. Let us extend a length n constant vector $\vec{\beta}$ to a vector of $\mathscr{G}$-rational power series such that $\vec{\beta}(w) = \vec{\beta}$ when $w = \epsilon$ otherwise $\vec{0}$. Then, $\vec{v} = \Delta^* \cdot \vec{\beta}$ is a vector of $\mathscr{G}$-rational power series.*

Proof: By Lemma 3.10, $\vec{v}$ satisfies

$$\vec{v} = \Delta \cdot \vec{v} + \vec{\beta}. \tag{3.3}$$

That is,

$$\vec{v}[i] = \sum_{1 \le j \le n} \Delta[i,j] \cdot \vec{v}[j] + \vec{\beta}[i] \tag{3.4}$$

for all $1 \le i \le n$.

When $i = n$ in Equation (3.4), by Corollary 3.11,

$$\begin{aligned} \vec{v}[n] &= \sum_{1 \le j \le n} \Delta[n,j] \cdot \vec{v}[j] + \vec{\beta}[n] \\ &= \Delta[n,n] \cdot \vec{v}[n] + \sum_{1 \le j \le n-1} \Delta[n,j] \cdot \vec{v}[j] + \vec{\beta}[n] \\ &= \Delta[n,n]^* \cdot \left(\sum_{1 \le j \le n-1} \Delta[n,j] \cdot \vec{v}[j] + \vec{\beta}[n] \right). \end{aligned} \tag{3.5}$$

When $1 \le i \le n-1$ in Equation (3.4), substituting $\vec{v}[n]$ into the equation (3.4) gives

$$\begin{aligned} \vec{v}[i] &= \sum_{1 \le j \le n} \Delta[i,j] \cdot \vec{v}[j] + \vec{\beta}[i] \\ &= \Delta[i,n] \cdot \vec{v}[n] + \sum_{1 \le j \le n-1} \Delta[i,j] \cdot \vec{v}[j] + \vec{\beta}[i] \\ &= \Delta[i,n] \cdot \Delta[n,n]^* \cdot \left(\sum_{1 \le j \le n-1} \Delta[n,j] \cdot \vec{v}[j] + \vec{\beta}[n] \right) + \sum_{1 \le j \le n-1} \Delta[i,j] \cdot \vec{v}[j] + \vec{\beta}[i] \\ &= \sum_{1 \le j \le n-1} \Big((\Delta[i,n] \cdot \Delta[n,n]^* \cdot \Delta[n,j] + \Delta[i,j]) \cdot \vec{v}[j] \Big) + \Delta[i,n] \cdot \Delta[n,n]^* \cdot \vec{\beta}[n] + \vec{\beta}[i]. \end{aligned} \tag{3.6}$$

Based on these equations, we can prove the lemma by induction about n.

When $n = 1$, $\vec{v}$ satisfies

$$\vec{v}[1] = \Delta[1,1] \cdot \vec{v}[1] + \vec{\beta}[1].$$

Corollary 3.11 shows this equation has a unique solution $\vec{v}$ such that

$$\begin{aligned}\vec{v}[1](w) &= (\Delta[1,1]^* \cdot \vec{\beta}[1])(w)\\ &= \sum_{w=uv} \Delta[1,1]^*(u)\vec{\beta}[1](v)\\ &= \Delta[1,1]^*(w)\vec{\beta}[1](\epsilon)\\ &= \Delta[1,1]^*(w)\vec{b}[1]\end{aligned}$$

since $\vec{\beta}(w) = \vec{0}$ for all $w \in \Sigma^* \setminus \{\epsilon\}$. We can construct this by $\mathscr{G}$-rational operations.

When $n > 1$, we define a $(n-1) \times (n-1)$ matrix of $\mathscr{G}$-rational power series Δ' and a length $n-1$ vector of power series $\vec{\beta}'$ as below:

$$\begin{aligned}\Delta'[i,j] &= (\Delta[i,n] \cdot \Delta[n,n]^* \cdot \Delta[n,j] + \Delta[i,j]),\\ \vec{\beta}'[i] &= \Delta[i,n] \cdot \Delta[n,n]^* \cdot \vec{\beta}[n] + \vec{\beta}[i]\end{aligned}$$

for all $1 \le i,j \le n-1$.

Using this, we can transform the equation (3.6).

$$\vec{v}[i] = \Sigma_{1 \le j \le n-1}\Delta'[i,j] \cdot \vec{v}[j] + \vec{\beta}'[i]$$

When we write the elements of $\vec{v}$ from 1 to $n-1$ by $\vec{v}[1..n-1]$, this equation means

$$\vec{v}[1..n-1] = \Delta' \cdot \vec{v}[1..n-1] + \vec{\beta}'.$$

By Lemma 3.10,

$$\vec{v}[1..n-1] = \Delta'^* \cdot \vec{\beta}'.$$

From an induction by assuming that Lemma 3.12 holds for $n-1$, we can confirm that $\vec{v}[1..n-1]$ is a vector of $\mathscr{G}$-rational power series. The final element $\vec{v}[n]$ of $\vec{v}$ is $\mathscr{G}$-rational power series since Equation (3.5) only requires the elements of $\vec{v}[1..n-1]$ and $\mathscr{G}$-rational operations. ■

Lemma 3.13. *Let an SWFA $\mathcal{A} = (\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta)$. $f_{\mathcal{A}}$ can be represented by $\mathscr{G}$-rational operations.*

Proof: Let us extend $\vec{\alpha}$ and $\vec{\beta}$ to vectors of $\mathscr{G}$-rational power series as with Lemma 3.12. Δ can also be extended to a matrix of $\mathscr{G}$-rational power series by defining $\Delta(x) = \Delta_x$ for all $x \in \Sigma$ and $\Delta(w) = O$ for all $w \in \Sigma^* \setminus \Sigma$. Then,

$$f_{\mathcal{A}} = \vec{\alpha} \cdot \Delta^* \cdot \vec{\beta}.$$

By Lemma 3.12, $\Delta^* \cdot \vec{\beta}$ is a vector of $\mathscr{G}$-rational power series. Combining with Lemma 3.9 shows that $\vec{\alpha} \cdot \Delta^* \cdot \vec{\beta}$ is $\mathscr{G}$-rational power series. Therefore, $f_{\mathcal{A}}$ can be represented by $\mathscr{G}$-rational operations. ■

This lemma proves $\mathscr{G}$-Rec $\subseteq$ $\mathscr{G}$-Rat.

3.3 Basic Properties

In the following of this paper, we assume the codomain of the power series in concern is a field $\mathbb{F}$. This restriction has brought fruitful positive results on WFAs including their efficient learnability [8, 9]. Under the assumption, we observe that some basic properties of WFAs over fields also hold for SWFAs.

Definition 3.14. The matrix $\mathbf{H}_f$ associated to a formal power series $f : \Sigma^* \to \mathbb{F}$ is an infinite matrix with rows and columns indexed by strings in Σ^* such that $\mathbf{H}_f[p, s] = f(ps)$ for all $p, s \in \Sigma^*$.

Theorem introduced by Fliess [15] can be seen as a weighted counterpart of the Myhill-Nerode theorem [22], which also holds for SWFAs.

Theorem 3.15. *The rank of the matrix $\mathbf{H}_f$ associated to $f : \Sigma^* \to \mathbb{F}$ is finite if and only if f is $\mathscr{G}$-recognizable for some $\mathscr{G}$. In that case, there exists an SWFA $\mathcal{A}$ over $\mathscr{G}$ with* $\mathrm{rank}(\mathbf{H}_f)$ *states representing f and no SWFA with fewer states can represent f.*

The proof in [6] works regardless of whether Σ is finite or infinite.

Hereafter, we fix the $\mathscr{G}$-recognizable power series f in concern and we often drop the subscript f from $\mathbf{H}_f$ if there is no risk of confusion.

When (P, S) is a minimal basis of $\mathbf{H}$ and the alphabet is finite, one can construct a WFA representing f from $\mathbf{H}_{(P,S)}$ and $\mathbf{H}_{(P,x,S)}$ for all $x \in \Sigma$, where $\mathbf{H}_{(P,x,S)}[p, s] = f(pxs)$ for all $p \in P$ and $s \in S$ [9]. We will establish the parallel result on SWFAs in Proposition 3.17.

Definition 3.16. For a non-singular mask (P, S), the SWFA based on (P, S) is defined to be $\mathcal{A}_{(P,S)} = (\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta)$, where $Q = P$, $\vec{\alpha}^\top = \mathbf{H}_{(\{\epsilon\},S)}\mathbf{H}^{-1}_{(P,S)}$, $\vec{\beta} = \mathbf{H}_{(P,\{\epsilon\})}$ and $\Delta_x = \mathbf{H}_{(P,x,S)}\mathbf{H}^{-1}_{(P,S)}$ for all $x \in \Sigma$.

One may wonder whether functions in Δ defined above belong to $\mathscr{G}$. We will show it in the following propositions.

Proposition 3.17. *Let (P, S) be a minimal basis. Then, $\mathcal{A}_{(P,S)} = (\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta)$ correctly represents f. Moreover, all guard functions in Δ of $\mathcal{A}_{(P,S)}$ are in $\mathscr{G}$.*

Proof: By Theorem 3.15, there is an SWFA $\mathcal{A}' = (\mathscr{G}, Q', \vec{\alpha}', \vec{\beta}', \Delta')$ with $|P|$ $(= |S|)$ states representing f, i.e., $f(w) = \vec{\alpha}'^\top \Delta'_w \vec{\beta}'$ for all $w \in \Sigma^*$. Define matrices $U_P \in \mathbb{F}^{P \times Q'}$ and $V_S \in \mathbb{F}^{Q' \times S}$ by $U_P[p, \cdot] = \vec{\alpha}'^\top \Delta'_p$ for all $p \in P$ and $V_S[\cdot, s] = \Delta'_s \vec{\beta}'$ for all $s \in S$, respectively. Then we have $\mathbf{H}_{(P,S)} = U_P V_S$ and $\mathbf{H}_{(P,x,S)} = U_P \Delta'_x V_S$ for $x \in \Sigma$. Since $\mathbf{H}_{(P,S)}$ is non-singular, U_P and V_S are also non-singular. From the definition of $\mathcal{A}_{(P,S)}$, $\mathbf{H}_{(P,x,S)} = \Delta_x \mathbf{H}_{(P,S)} = \Delta_x U_P V_S$. Thus, $\Delta'_x = U_P^{-1} \Delta_x U_P$. This fact can be extended to strings $w \in \Sigma^*$ as $\Delta'_w = U_P^{-1} \Delta_w U_P$. Since $\mathbf{H}_{(\{\epsilon\},S)} = \vec{\alpha}'^\top V_S$ and $\mathbf{H}_{(P,\{\epsilon\})} = U_P \vec{\beta}'$, we have $\vec{\alpha}'^\top = \vec{\alpha}^\top U_P$ and $\vec{\beta}' = U_P^{-1} \vec{\beta}$. Therefore, $f(w) = \vec{\alpha}'^\top \Delta'_w \vec{\beta}' = \vec{\alpha}^\top U_P U_P^{-1} \Delta_w U_P U_P^{-1} \vec{\beta} = \vec{\alpha}^\top \Delta_w \vec{\beta}$. Additionally, $\Delta'_x = U_P^{-1} \Delta_x U_P$ shows that every guard function in Δ can be represented by a linear transformation of those in Δ'. Thus, all guard functions in Δ are in $\mathscr{G}$ by Assumption 1. ■

Proposition 3.18 guarantees that the guard functions in the transition relation constructed in Definition 3.16 indeed belong to $\mathscr{G}$, by extending the property shown in Proposition 3.17 to the case that (P, S) is a non-singular mask.

Proposition 3.18. *If (P, S) is non-singular, all guard functions of $\mathcal{A}_{(P,S)} = (\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta)$ are in $\mathscr{G}$.*

Proof: Let (P', S') be a minimal basis that expands (P, S), i.e., $P' \supseteq P$ and $S' \supseteq S$. For the SWFA $\mathcal{A}_{(P',S')} = (\mathscr{G}, Q', \vec{\alpha}', \vec{\beta}', \Delta')$, we have for all $x \in \Sigma$,

$$\Delta'_x \mathbf{H}_{(P',S')} = \mathbf{H}_{(P',x,S')} \, ,$$

which can be rewritten as

$$\begin{pmatrix} \Delta'_{(P,P)_x} & \Delta'_{(P,\tilde{P})_x} \\ \Delta'_{(\tilde{P},P)_x} & \Delta'_{(\tilde{P},\tilde{P})_x} \end{pmatrix} \begin{pmatrix} \mathbf{H}_{(P,S)} & \mathbf{H}_{(P,\tilde{S})} \\ \mathbf{H}_{(\tilde{P},S)} & \mathbf{H}_{(\tilde{P},\tilde{S})} \end{pmatrix} = \begin{pmatrix} \mathbf{H}_{(P,x,S)} & \mathbf{H}_{(P,x,\tilde{S})} \\ \mathbf{H}_{(\tilde{P},x,S)} & \mathbf{H}_{(\tilde{P},x,\tilde{S})} \end{pmatrix}$$

where $\tilde{P} = P' \setminus P$ and $\tilde{S} = S' \setminus S$. Thus $\Delta'_{(P,P)_x} \mathbf{H}_{(P,S)} + \Delta'_{(P,\tilde{P})_x} \mathbf{H}_{(\tilde{P},S)} = \mathbf{H}_{(P,x,S)}$ and

$$\Delta_x = \mathbf{H}_{(P,x,S)} \mathbf{H}^{-1}_{(P,S)} = (\Delta'_{(P,P)_x} \mathbf{H}_{(P,S)} + \Delta'_{(P,\tilde{P})_x} \mathbf{H}_{(\tilde{P},S)}) \mathbf{H}^{-1}_{(P,S)}$$

for all $x \in \Sigma$. Therefore, all guard functions in Δ can be represented by a linear transformation of those of Δ'. That is, they belong to $\mathscr{G}$ by Assumption 1. ■

3.4 Minimization and Equivalence Checking

In this section, we consider *minimization* and *equivalence checking* for SWFAs. Minimization (also called *standardization*) is the problem to minimize the number of states of

$$\underset{U_P[p,\cdot]\Delta}{\overset{U_P}{\begin{pmatrix}1&2&2&1\\0&1&3&0\\g_1&g_2&g_3&g_4\end{pmatrix}}}\longrightarrow\begin{pmatrix}1&2&2&1\\0&1&3&0\\0&g_2-2g_1&g_3-2g_1&g_4-g_1\end{pmatrix}\longrightarrow\left(\overset{\text{triangular matrix}}{\boxed{\begin{matrix}1&2&2\\0&1&3\\0&0&g_3-2g_1-3(g_2-2g_1)\end{matrix}}}\;\begin{matrix}1\\0\\g_4-g_1\end{matrix}\right)$$

Figure 3.2: Gaussian elimination dealing with guard functions

a given SWFA. Equivalence checking is the problem to check whether given two SWFAs are equivalent or not. Our algorithms assume that $\mathscr{G}$ admits a zero-checking procedure.

Assumption 2. *For a given $g \in \mathscr{G}$, there is a procedure that decides whether $g(x) = 0$ for all $x \in \Sigma$. If not, it finds a witness $x \in \Sigma$ such that $g(x) \neq 0$.*

This requirement corresponds to the emptiness checking in the context of SFAs [3, 11]. Under the assumption, we present a minimization procedure for SWFAs inspired by Schutzenberger's algorithm for WFAs [25]. The minimization procedure for SWFAs consists of two phases: First we obtain a minimal basis (P, S) for a given SWFA; Second, we construct a minimized SWFA based on (P, S) and $\mathbf{H}_{(P,S)}$.

Suppose an SWFA $\mathcal{A} = (\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta)$ representing $f_{\mathcal{A}}$ is given as an input. The first phase is shown in Algorithm 1. For two sets P and S of strings, define matrices $U_P \in \mathbb{F}^{P\times Q}$ and $V_S \in \mathbb{F}^{Q\times S}$ by $U_P[p,\cdot] = \vec{\alpha}^{\top}\Delta_p$ for all $p \in P$ and $V_S[\cdot, s] = \Delta_s\vec{\beta}$ for all $s \in S$, respectively. When U_P and V_S are bases of U_{Σ^*} and V_{Σ^*}, respectively, (P, S) will be a basis of $\mathbf{H}_{f_{\mathcal{A}}}$, since $U_{\Sigma^*}V_{\Sigma^*} = \mathbf{H}_{f_{\mathcal{A}}}$. Starting from $P = \{\epsilon\}$, our procedure constructs such P by expanding P and increasing the rank of U_P in the loop of Line 2. Of course when $|P| = |Q|$, we cannot expand P any further. Otherwise, to execute Line 4, we first consider the matrix $[U_P^{\top}\ (U_P[p,\cdot]\Delta)^{\top}]^{\top}$, which concatenates U_P and $U_P[p,\cdot]\Delta$, for each $p \in P$. As shown in the example in Figure 3.2, Gaussian elimination process dealing with guard functions makes the first $|P| + 1$ columns of the concatenated matrix a triangular matrix, which includes a guard function only at the lower-right corner. Then, we check whether the guard function g at the lower-right corner of the triangular matrix always outputs 0 or not. If $g(x) = 0$ for all $x \in \Sigma$, the finding in Line 4 fails. If not, we can get $x \in \Sigma$ such that $g(x) \neq 0$ by Assumption 2. It means we have succeeded in finding $p \in P$ and $x \in \Sigma$ such that $U_P[p,\cdot]\Delta_x$ is linearly independent of U_P. Then, we extend P by px. We construct S and V_S with the desired property by the symmetric procedure. The obtained mask (P,S) is a basis, but not necessarily minimal. By removing elements of P and S as much as possible while keeping the rank of the matrix $\mathbf{H}_{(P,S)}$, we obtain a minimal basis.

The second phase constructs a minimized SWFA from the minimal basis. By Proposition 3.17, it is enough to show that the SWFA $\mathcal{A}_{(P,S)}$ in Definition 3.16 is computable. The only possible difficulty is in the construction of the transition relation, named $\Delta^{\min}$ here, for which $\Delta^{\min}_x = \mathbf{H}_{(P,x,S)}\mathbf{H}^{-1}_{(P,S)}$ must hold for all $x \in \Sigma$. One can compute the matrix $\mathbf{H}_{(P,\Delta,S)} \in \mathscr{G}^{P\times S}$ defined by $\mathbf{H}_{(P,\Delta,S)}[p,s] = \vec{\alpha}^{\top}\Delta_p\Delta\Delta_s\vec{\beta}$. We then have $\mathbf{H}_{(P,\Delta,S)}[p,s](x) = \mathbf{H}_{(P,x,S)}$ for all $x \in \Sigma$, $p \in P$ and $s \in S$. Thus, $\Delta^{\min} = \mathbf{H}_{(P,\Delta,S)}\mathbf{H}^{-1}_{(P,S)}$ has the desired property[1].

Algorithm 1: Computing a minimal basis from an SWFA

```
   Input: an SWFA $\mathcal{A} = (\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta)$
   Output: a minimal basis $(P,S)$
1  initialize $P \leftarrow \{\epsilon\}$, $S \leftarrow \{\epsilon\}$;
2  repeat
3      $U_P[p,\cdot] \leftarrow \vec{\alpha}^{\top}\Delta_p$ for all $p \in P$;
4      find $p \in P$ and $x \in \Sigma$ such that $U_P[p,\cdot]\Delta_x$ is linearly independent of $U_P$;
5      $P \leftarrow P \cup \{px\}$;
6  until finding in Line 4 fails;
7  repeat
8      $V_S[\cdot,s] \leftarrow \Delta_s\vec{\beta}$ for all $s \in S$;
9      find $s \in S$ and $x \in \Sigma$ such that $\Delta_x V_S[\cdot,s]$ is linearly independent of $V_S$;
10     $S \leftarrow S \cup \{xs\}$;
11 until finding in Line 9 fails;
12 $\mathbf{H}_{(P,S)} \leftarrow U_P V_S$;
13 $P \leftarrow P \setminus P'$ for a maximal $P' \subseteq P$ such that $\mathrm{rank}(\mathbf{H}_{(P,S)}) = \mathrm{rank}(\mathbf{H}_{(P\setminus P',S)})$;
14 $S \leftarrow S \setminus S'$ for a maximal $S' \subseteq S$ such that $\mathrm{rank}(\mathbf{H}_{(P,S)}) = \mathrm{rank}(\mathbf{H}_{(P,S\setminus S')})$;
15 return $(P,S)$;
```

[1]The matrices Δ and $\mathbf{H}_{(P,\Delta,S)}$ are over $\mathscr{G}$, which is not necessarily a semiring, whereas the others are over $\mathbb{F}$. We can naturally apply the definition of multiplication of two matrices over a semiring to these cases, since $\mathscr{G}$ is closed under linear combination.

Theorem 3.19. *Let L and E be the time complexities of computing a linear combination and zero-checking of a guard function, respectively. One can minimize an arbitrary SWFA in $O(|Q|(L|Q|^2 + E))$ time, where $|Q|$ is the number of states of the SWFA.*

Proof: The finding at Lines 4 and 9 of Algorithm 1 involves Gaussian elimination dealing with guard functions and a zero-checking on $\mathscr{G}$. The former requires $O(L|Q|^2)$ and the latter requires E time. Thus, the entire procedure requires $O(|Q|(L|Q|^2 + E))$ time since these lines are executed at most $|Q|$ times. ■

When $\mathscr{G}$ is $\mathscr{G}^{\mathrm{wfa}}$, we have $L, E \in O(|\Sigma|)$ and the time complexity $O(|\Sigma||Q|^3)$ coincides with that of Schutzenberger's algorithm for minimizing WFAs [25].

The equivalence checking between SWFAs is immediately derived.

Corollary 3.20. *One can decide whether two SWFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ represent the same power series in $O((|Q_{\mathcal{A}_1}| + |Q_{\mathcal{A}_2}|)(L(|Q_{\mathcal{A}_1}| + |Q_{\mathcal{A}_2}|)^2 + E))$ time.*

Proof: One can construct in linear time the "difference SWFA" $\mathcal{A}$ with $|Q_{\mathcal{A}_1}| + |Q_{\mathcal{A}_2}|$ states that represents $f_{\mathcal{A}} = f_{\mathcal{A}_1} - f_{\mathcal{A}_2}$. Two SWFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are equivalent if and only if the basis of $\mathbf{H}_{f_{\mathcal{A}}}$ obtained by Algorithm 1 is empty. ■

Again, when $\mathscr{G}$ is $\mathscr{G}^{\mathrm{wfa}}$, the time complexity becomes $O(|\Sigma|(|Q_{\mathcal{A}_1}| + |Q_{\mathcal{A}_2}|)^3)$, which coincides with the time complexity of Cortes et al.'s algorithm for equivalence checking of WFAs [12]. Note that if $\mathcal{A}_1$ and $\mathcal{A}_2$ are not equivalent, there must be strings $p \in P$ and $s \in S$ of the obtained minimal basis (P, S) such that $\mathbf{H}_{(P,S)}[p, s] \neq 0$, for which ps witnesses the difference of $f_{\mathcal{A}_1}$ and $f_{\mathcal{A}_2}$. This gives a counterexample to an EQ on SWFAs.

Chapter 4

Query Learning of Symbolic Weighted Finite Automata

4.1 Problem Setup

Query learning is an active learning model, where an algorithm actively asks queries to the teacher and constructs a representation of the target function from a domain X to a codomain Y using the teacher's answers. The most famous setting of query learning is learning under a *minimally adequate teacher (MAT)*, proposed by Angluin [2]. In this model, the MAT can answer two types of queries concerning an unknown function f_*: *membership queries (MQs)* and *equivalence queries (EQs)*. For an MQ, the MAT receives $w \in X$ and returns $f_*(w)$. For an EQ, the MAT receives a hypothesis $\mathcal{H}$ that represents a function $f_{\mathcal{H}} \in Y^X$. The MAT returns "yes" if $f_* = f_{\mathcal{H}}$. Otherwise, the MAT returns $c \in X$ such that $f_*(c) \neq f_{\mathcal{H}}(c)$ as a counterexample.

4.2 Query Learning Algorithm

This section presents a learning algorithm under the MAT model for $\mathscr{G}$-recognizable power series using SWFAs when $\mathscr{G}$ satisfies Assumption 1. We extend the existing learning algorithm for WFAs [9] by embedding the key idea of the SFA learning algorithm of Argyros and D'Antoni [3].

Argyros and D'Antoni's algorithm takes as input a MAT learning algorithm Λ for predicates and uses instances $\Lambda^{(q,q')}$ to infer a predicate on the edge between two states q and q'. The learning algorithm plays the role of a MAT for those instances and answers queries from them. Through communication with predicate learners, the learning algorithm constructs its hypothesis. Following this idea, our algorithm also assumes that the class of guard functions $\mathscr{G}$ admits a MAT learner Λ and uses its instances to construct the transition relation of a hypothesis SWFA.

The pseudo-code of our algorithm is shown in Algorithms 2–5. Now, our learning target $\mathscr{G}$-recognizable power series is $f_*: \Sigma^* \to \mathbb{F}$. Our goal is to find a minimal basis (P, S) of $\mathbf{H}_{f_*}$ and to construct the SWFA $\mathcal{A}_{(P,S)}$, for which $f_{\mathcal{A}_{(P,S)}} = f_*$ holds by Proposition 3.17. We call a triple $\mathcal{T} = (P, S, \mathbf{H}_{(P,S)})$ an *observation table.*

Algorithm 2 shows the overall picture of our algorithm. First, our algorithm asks an EQ to the MAT on the zero SWFA. If the MAT answers with a counterexample $c \in \Sigma^*$, our algorithm initializes the observation table $\mathcal{T}$ with $P = \{c\}$ and $S = \{\epsilon\}$. We will expand the mask to obtain a minimal basis, while keeping it non-singular.

To build a hypothesis, we follow the construction shown in Definition 3.16. Among components of $\mathcal{A}_{(P,S)} = (\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta)$, one can easily construct $Q = P$, $\vec{\alpha}^\top = \mathbf{H}_{(\{\epsilon\},S)}\mathbf{H}^{-1}_{(P,S)}$, and $\vec{\beta} = \mathbf{H}_{(P,\{\epsilon\})}$, since any finite sub-block of $\mathbf{H}_{f_*}$ can be obtained by an appropriate number of MQs. The remaining issue is how to compute Δ. We can get $\Delta_x = \mathbf{H}_{(P,x,S)}\mathbf{H}^{-1}_{(P,S)}$ for concrete $x \in \Sigma$ using MQs, but computing guard functions in Δ is not trivial. For this sake, we need the help of the guard function learner Λ. Algorithm 3 initializes all entries of the transition relation $\Delta^{\mathcal{H}}$ of our hypothesis $\mathcal{H}$ by null and creates $|Q|^2$ instances $\Lambda^{(q,q')}$ of Λ for all state pairs $(q, q') \in Q^2$, and then calls Algorithm 4.

To bring $\Delta^{\mathcal{H}}$ closer to Δ, Algorithm 4 lets $\Lambda^{(q,q')}$ learn $\Delta[q, q']$ by pretending to be a MAT for them, though we do not know the exact goal Δ itself. Proposition 3.18 guarantees that Λ is capable of learning $\Delta[q,q'] \in \mathscr{G}$. To avoid confusion between queries by $\Lambda^{(q,q')}$ to our algorithm and those by our algorithm to the MAT, we use $\mathscr{G}$-EQs and $\mathscr{G}$-MQs for equivalence queries and membership queries from the instances of Λ, respectively. The answer to a $\mathscr{G}$-MQ on $x \in \Sigma$ from $\Lambda^{(q,q')}$ is just $\Delta_x[q,q']$. Since $\Delta_x[q,\cdot] = \mathbf{H}_{(\{q\},x,S)}\mathbf{H}^{-1}_{(P,S)}$, we require only $|S|$ extra MQs given the observation table.

When $\Lambda^{(q,q')}$ asks a $\mathscr{G}$-EQ on a hypothesis guard function g, our algorithm takes g as the (q,q')-element of $\Delta^{\mathcal{H}}$. To answer the $\mathscr{G}$-EQ from $\Lambda^{(q,q')}$, we need to find a counterexample. This will be done by processing the MAT's answer to our EQ by Algorithm 5. Until then, answering the $\mathscr{G}$-EQ of $\Lambda^{(q,q')}$ is suspended.

After building a hypothesis SWFA, our algorithm asks an EQ to the MAT on the hypothesis. When the MAT replies with a counterexample c, Algorithm 5 starts a procedure to fix the hypothesis. Following prior works [24, 9], our algorithm finds a prefix ux of c, with $u \in \Sigma^*$ and $x \in \Sigma$, that satisfies the following equations, where $S = \{s_1, s_2, \ldots, s_k\}$ (Line 1 of Algorithm 5):

$$(\mathrm{MQ}(us_1), \mathrm{MQ}(us_2), \ldots, \mathrm{MQ}(us_k)) = \vec{\alpha}^\top \Delta^{\mathcal{H}}_u \mathbf{H}_{(P,S)}\,, \tag{4.1}$$

$$(\mathrm{MQ}(uxs_1), \mathrm{MQ}(uxs_2), \ldots, \mathrm{MQ}(uxs_k)) \neq \vec{\alpha}^\top \Delta^{\mathcal{H}}_{ux} \mathbf{H}_{(P,S)}\,, \tag{4.2}$$

where $\mathrm{MQ}(w)$ is just $f_*(w)$, but we emphasize the learner obtains the values by MQs. Such a prefix ux always exists and can be found by binary search about the length $|c|$ of c, which requires $O(|S| \log |c|)$ MQs. This suggests that the transition relation $\Delta^{\mathcal{H}}$ of our current hypothesis has something wrong with the transition by x.

So, our algorithm computes $\Delta_x = \mathbf{H}_{(P,x,S)}\mathbf{H}^{-1}_{(P,S)}$ and compares it with $\Delta^{\mathcal{H}}_x$. If Δ_x and $\Delta^{\mathcal{H}}_x$ disagree on the (q,q')-element, we give x to $\Lambda^{(q,q')}$ as a counterexample to the $\mathscr{G}$-EQ from $\Lambda^{(q,q')}$. With the counterexample, our algorithm calls Algorithm 4 and restarts learning the guard function of $\Delta[q,q']$ using $\Lambda^{(q,q')}$. If $\Delta^{\mathcal{H}}_x = \Delta_x$, this means that the target power series f_* requires an SWFA with more states (Lemma 4.1). By processing the counterexample properly, our algorithm expands the observation table $\mathcal{T}$ (Lines 9-11 of Algorithm 5). Now, our algorithm decides to reconstruct the hypothesis from scratch because the goal hypothesis $\mathcal{A}_{(P,S)}$ has changed. Along with this, all Λ instances are discarded.

Algorithm 2: SWFA Learning Algorithm

```
   Input: a MAT learning algorithm $\Lambda$
   Output: an SWFA representing the target power series
1  $P \leftarrow \{c\}$ and $S \leftarrow \{\epsilon\}$ for the counterexample $c$ to the EQ on the zero SWFA;
2  initialize the observation table $\mathcal{T} = (P, S, \mathbf{H}_{(P,S)})$ using the MQ on $c$;
3  $\mathcal{H} \leftarrow$ null;
4  loop
5      if $\mathcal{H}$ is null then $\mathcal{H} \leftarrow$ build_hypothesis($\mathcal{T}$);   // Algorithm 3
6      ask an EQ on $\mathcal{H}$;
7      if the MAT replies with a counterexample $c$ then
8          $\mathcal{H}, \mathcal{T} \leftarrow$ process_counterexample($\mathcal{H}$, $\mathcal{T}$, $c$);   // Algorithm 5
9      else return $\mathcal{H}$ and terminate;
```

Algorithm 3: build_hypothesis

```
   Input: $\mathcal{T} = (P, S, \mathbf{H}_{(P,S)})$
   Output: a hypothesis SWFA
1  $Q \leftarrow P$; $\vec{\alpha}^\top \leftarrow \mathbf{H}_{(\{\epsilon\},S)}\mathbf{H}^{-1}_{(P,S)}$; $\vec{\beta} \leftarrow \mathbf{H}_{(P,\{\epsilon\})}$;
2  $\mathcal{H} \leftarrow (\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta^{\mathcal{H}})$, where $\Delta^{\mathcal{H}}[q,q'] =$ null for all $(q,q') \in Q^2$;
3  for $(q,q') \in Q^2$ do
4      initialize the algorithm $\Lambda^{(q,q')}$;
5      $\mathcal{H} \leftarrow$ update_transition($q, q', \Lambda^{(q,q')}, \mathcal{H}, \mathcal{T}$);   // Algorithm 4
6  return $\mathcal{H}$;
```

4.3 Correctness and Query Complexity

The following lemma ensures that our mask (P,S) is always kept non-singular and $\mathrm{rank}(\mathbf{H}_{(P,S)})$ is strictly increasing at Line 11 of Algorithm 5.

Lemma 4.1 (Bisht et al. [9]). *Suppose ux with $u \in \Sigma^*$ and $x \in \Sigma$ satisfies Eqs. (4.1) and (4.2). If $\Delta^{\mathcal{H}}_x = \Delta_x$, then $\mathrm{rank}(\mathbf{H}_{(P',S')}) = \mathrm{rank}(\mathbf{H}_{(P,S)}) + 1$ for $P' = P \cup \{u\}$ and $S' = S \cup \{xs_i\}$, where $\mathrm{MQ}(uxs_i) \neq \vec{\alpha}^\top \Delta^{\mathcal{H}}_{ux} \mathbf{H}_{(P,S)}[s_i]$.*

Algorithm 4: update_transition

```
Input: $q, q', \Lambda^{(q,q')}, \mathcal{H} = (\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta^{\mathcal{H}}), \mathcal{T} = (P, S, \mathbf{H}_{(P,S)})$
Output: the updated hypothesis
1  repeat
2      $\Lambda^{(q,q')}$ asks a $\mathscr{G}$-MQ on $x \in \Sigma$;
3      get $\mathbf{H}_{(\{q\},x,S)}[q,\cdot]$ by MQs on $qxs$ for all $s \in S$;
4      $\Delta_x[q,\cdot] \leftarrow \mathbf{H}_{(\{q\},x,S)}\mathbf{H}^{-1}_{(P,S)}$;
5      answer the $\mathscr{G}$-MQ by $\Delta_x[q,q']$;
6  until $\Lambda^{(q,q')}$ asks a $\mathscr{G}$-EQ on a hypothesis guard function $g$;
7  $\Delta^{\mathcal{H}}[q,q'] \leftarrow g$;
8  return $(\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta^{\mathcal{H}})$;
```

Algorithm 5: process_counterexample

```
Input: $\mathcal{H} = (\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta^{\mathcal{H}}), \mathcal{T} = (P, S, \mathbf{H}_{(P,S)})$, a counterexample $c$
Output: a hypothesis SWFA (or null if $\mathcal{T}$ is updated), the observation table
1  find a prefix $ux$ of $c$, where $u \in \Sigma^*, x \in \Sigma$, that satisfies Eqs. (4.1) and (4.2);
2  get $\mathbf{H}_{(P,x,S)}$ by MQs on $pxs$ for all $p \in P$ and $s \in S$;
3  $\Delta_x \leftarrow \mathbf{H}_{(P,x,S)}\mathbf{H}^{-1}_{(P,S)}$;
4  for $(q,q') \in Q^2$ do
5      if $\Delta_x[q,q'] \neq \Delta^{\mathcal{H}}_x[q,q']$ then
6          give $x$ to $\Lambda^{(q,q')}$ as a counterexample to the $\mathscr{G}$-EQ;
7          $\mathcal{H} \leftarrow$update_transition$(q, q', \Lambda^{(q,q')}, \mathcal{H}, \mathcal{T})$;   // Algorithm 4
8  if $\mathcal{H}$ is updated then return $\mathcal{H}$, $\mathcal{T}$;
9  $P' \leftarrow P \cup \{u\}$;
10 $S' \leftarrow S \cup \{xs_i\}$ where $s_i \in S$ satisfies $\mathrm{MQ}(uxs_i) \neq \vec{\alpha}^\top \Delta^{\mathcal{H}}_{ux}\mathbf{H}_{(P,S)}[s_i]$;
11 expand $\mathbf{H}_{(P,S)}$ to $\mathbf{H}_{(P',S')}$;
12 return null, $(P', S', \mathbf{H}_{(P',S')})$;
```

The proof by [9] requires no change for the case of SWFAs, since the lemma involves only finitely many elements of Σ.

By Lemma 4.1, our mask will finally converge to a minimal basis (P,S), for which we have $f_{\mathcal{A}_{(P,S)}} = f_*$. Then each $\Lambda^{(q,q')}$ will finally output the correct guard function.

In order to evaluate the query complexity of our algorithm, we first discuss how many $\mathscr{G}$-MQs and $\mathscr{G}$-EQs each MAT learner $\Lambda^{(q,q')}$ may make. We write the class of all linear combinations of guard functions in the transition relation Δ of a minimal SWFA representing f_* by

$$\mathscr{G}_{f_*} = \left\{ \sum_{(q,q')\in Q^2} a_{qq'} \Delta[q,q'] \;\middle|\; a_{qq'} \in \mathbb{F} \text{ for each } (q,q') \in Q^2 \right\} \subseteq \mathscr{G}\,.$$

Note that the class $\mathscr{G}_{f_*}$ does not depend on the choice of the minimal SWFA, since any minimal SWFAs representing the same power series can be converted into each other by linear transformation as shown in the proof of Proposition 3.17. Let $\mathcal{M}$ and $\mathcal{E}$ be the number of $\mathscr{G}$-MQs and $\mathscr{G}$-EQs that Λ makes to learn guard functions in $\mathscr{G}_{f_*}$, respectively.

Theorem 4.2. *Let $n = \operatorname{rank}(\mathbf{H}_{f_*})$ and m be the length of the longest counterexample to EQs returned by the MAT. Then, the proposed algorithm returns an SWFA representing f_* using Λ after raising at most $O(n^3\mathcal{E})$ EQs and $O(n^4\mathcal{M} + n^4\mathcal{E}(n + \log m))$ MQs.*

Proof: At first, we claim that the observation table $\mathcal{T} = (P, S, \mathbf{H}_{(P,S)})$ cannot be extended beyond n times. The table is extended on Line 11 of Algorithm 5 only when the condition of Lemma 4.1 is satisfied, and hence the rank of the sub-block $\mathbf{H}_{(P,S)}$ is definitely increased. By $\mathbf{H}_{(P,S)} \leq \operatorname{rank}(\mathbf{H}_{f_*}) = n$, the claim holds.

Whenever a counterexample to an EQ is given, we call Algorithm 5, except the first EQ for initializing $\mathcal{T}$. To count the number of EQs, we count how many times Algorithm 5 is performed. Each call of Algorithm 5 ends in either Line 8 or Line 12. The former happens only when a counterexample to an instance of Λ is found, which can happen at most $|Q|^2\mathcal{E} \leq n^2\mathcal{E}$ times without extending $\mathcal{T}$. The latter extends $\mathcal{T}$ and it is performed at most n times as shown before. Therefore, the algorithm builds a correct hypothesis after making at most $O(n^3\mathcal{E})$ EQs.

After the first MQ for initializing $\mathcal{T}$, our algorithm makes MQs for (a) constructing $\vec{\alpha}$ at Line 1 of Algorithm 3, (b) answering $\mathscr{G}$-MQs from instances of Λ at Line 3 of Algorithm 4, (c) finding a critical prefix of a counterexample at Line 1 of Algorithm 5, (d) answering $\mathscr{G}$-EQs by instances of Λ at Line 2 of Algorithm 5. Note that computing $\vec{\beta} = \mathbf{H}_{(P,\{\epsilon\})}$ at Line 1 of Algorithm 3 requires no extra MQs, since $\mathbf{H}_{(P,\{\epsilon\})}$ is a sub-block of $\mathbf{H}_{(P,S)}$. We can construct $\mathbf{H}_{(P',S')}$ at Line 11 of Algorithm 5 using results of MQs in

Table 4.1: Query complexities of representative learning algorithms for relevant automata

		Deterministic	Weighted
FA	EQ	$O(n)$	$O(n)$
	MQ	$O(n^2\|\Sigma\| + n \log m)$	$O(n^2\|\Sigma\| + n^2 \log m)$
		Rivest and Schapire [24]	Bisht et al. [9]
SFA	EQ	$O(n^3\hat{\mathcal{E}})$	$O(n^3\mathcal{E})$
	MQ	$O(n^4\hat{\mathcal{M}} + n^4\hat{\mathcal{E}} \log m)$	$O(n^4\mathcal{M} + n^4\mathcal{E}(n + \log m))$
		Argyros and D'Antoni [3]	(Ours)

(c) and (d).

The total number of MQs for (a) is at most n. Concerning (b), we make $|S|$ MQs for each $\mathscr{G}$-MQ from an instance of Λ. Thus, the number of MQs per an instance of Λ is at most $O(n\mathcal{M})$ by $|S| \leq n$. Each time $\mathcal{T}$ is extended, we make $|Q|^2 \leq n^2$ new instances of Λ, which happens n times. Thus, $O(n^4\mathcal{M})$ MQs are asked for (b) till the algorithm outputs a correct SWFA. For (c), we use $O(n \log m)$ MQs to find a critical prefix of each counterexample with binary search in Algorithm 5, which is called at most $O(n^3\mathcal{E})$ times, as we have argued for counting EQs. Thus, in total $O(n^4\mathcal{E} \log m)$ MQs are asked for (c). The total number of MQs for (d) is at most $O(n^5\mathcal{E})$, since we need $|P| \times |S| \leq n^2$ MQs for constructing $\mathbf{H}_{(P,x,S)}$ each time Algorithm 5 is called. All in all, the total number of MQs that our algorithm makes is at most $O(n^4\mathcal{M} + n^4\mathcal{E}(n + \log m))$. ■

We compare the query complexities of representative algorithms to learn relevant classes of automata under the MAT model in Table 4.1[1]. The query complexity of the proposed algorithm is higher than that of the one for learning WFAs with respect to n, which is also true when extending FA to SFA. The important point is that the if $\mathcal{M}$ and $\mathcal{E}$ do not depend on alphabet size $|\Sigma|$, the number of MQs of our algorithm also does not depend $|\Sigma|$ unlike that of WFAs. Therefore, our algorithm is suitable for learning power series over an extremely large or infinite alphabet.

We show some examples of $\mathscr{G}$ and guard function learners.

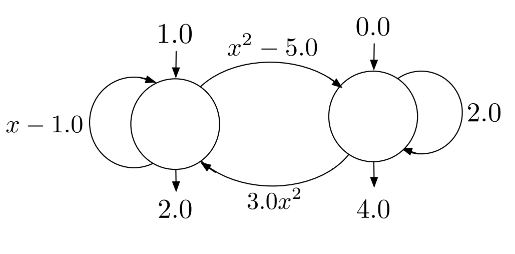
[1] $\hat{\mathcal{M}}$ and $\hat{\mathcal{E}}$ denote the numbers of $\mathscr{G}$-MQs and $\mathscr{G}$-EQs that predicate learners make in [3]'s algorithm, respectively.

Example 1. Guard functions in $\mathscr{G}^{\mathrm{wfa}}$ can be learned by $|\Sigma|$ $\mathscr{G}$-MQs and a $\mathscr{G}$-EQ, i.e., $\mathcal{M} = |\Sigma|$ and $\mathcal{E} = 1$. When learning SWFAs over $\mathscr{G}^{\mathrm{wfa}}$ with such a guard function learner, Algorithm 5 never calls Algorithm 4, because guard learners always raise the correct hypotheses. Then the number of required EQs goes down to $O(n)$. Still our algorithm makes more MQs than the WFA learner, where the same MQs are repeatedly made many times. We can remove such redundant MQs using memorization and then the number of MQs of our algorithm becomes as few as the WFA learner's.

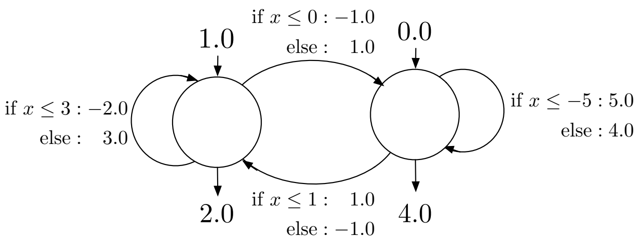
Example 2. Let $\mathscr{G}^{\mathrm{poly}}$ be the class of all polynomials. Then a trivial learning strategy for $\mathscr{G}^{\mathrm{poly}}$ requires $k+1$ $\mathscr{G}$-MQs and $k+1$ $\mathscr{G}$-EQs for a target polynomial of degree k. Suppose that an SWFA representing f_* has guard functions of polynomials of degree at most k. Since the maximum degree of polynomials does not increase by linear transformation, we have $\mathcal{M} = \mathcal{E} = k + 1$.

Example 3. Let $\mathscr{G}^{\mathrm{div}}$ be the class of all guard functions $g{:}\mathbb{Z} \to \mathbb{F}$ which partition the integer set $\mathbb{Z}$ into finitely many intervals and assign an entity of $\mathbb{F}$ to each interval. That is, $g \in \mathscr{G}^{\mathrm{div}}$ has a partition number $l \in \mathbb{N}$, l borders $a_1, a_2, \dots, a_l \in \mathbb{Z}$ and $l + 1$ values $b_0, b_1, b_2, \dots, b_l \in \mathbb{F}$ such that $a_1 < a_2 < \cdots < a_l$ and $g(x) = b_i$ if $a_i \le x < a_{i+1}$, assuming that $a_0 = -\infty$ and $a_{l+1} = \infty$. Since identifying each border a_i requires $O(\log |a_i|)$ $\mathscr{G}$-MQs, to learn g requires in total $O(\sum_{i=1}^{l} \log |a_i|)$ $\mathscr{G}$-MQs and $O(l)$ $\mathscr{G}$-EQs. Let L be the set of all borders a_i used in any of guard functions of an SWFA representing the target power series. Then a linear combination of those guard functions produces the function $h \in \mathscr{G}^{\mathrm{div}}$ with at most $|L|$ borders. Such h requires $\mathcal{M} = O(|L| \log \hat{l})$ $\mathscr{G}$-MQs and $\mathcal{E} = O(|L|)$ $\mathscr{G}$-EQs to learn, where $\hat{l} = \max\{\, |l| \mid l \in L \,\}$.

Example 4. We have shown that if $\mathscr{G} \subseteq \mathbb{F}^{\Sigma}$ is learnable, the class $\mathscr{G}'$ of $\mathscr{G}$-recognizable series is also learnable. Here, we can consider SWFAs over $\mathscr{G}' \subseteq \mathbb{F}^{\Sigma^*}$, whose edges have SWFAs over $\mathscr{G}$ as representations of guard functions. In this way, one can recursively obtain learnable classes of more complex formal power series.



(a) $\mathscr{G}^{poly}$



(b) $\mathscr{G}^{div}$

Figure 4.1: Examples of target SWFAs for experiments (a) An SWFA of $\mathscr{G}^{poly}$ with $k = 2$ and $n = 2$. (b) An SWFA of $\mathscr{G}^{div}$ with the partition number $p = 1$ and $n = 2$.

4.4 Experiments

Theoretical worst-case query complexity and empirical performance are often different. In this section, we conduct some experiments to evaluate the empirical performance of our proposed algorithm.

4.4.1 Setting

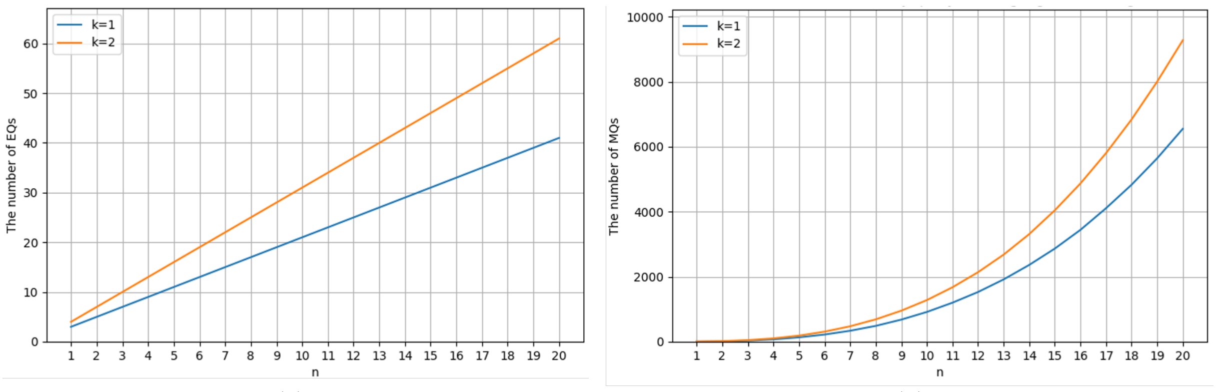
We chose $\mathscr{G}^{poly}$ shown in Example 2 and $\mathscr{G}^{div}$ in Example 3.

We generated learning target $\mathscr{G}$-recognizable power series using SWFAs with $\mathscr{G}$. Figure 4.1 shows examples of such target SWFAs. We randomly chose initial vector, final vector, polynomial coefficients and weight in each partition of the target SWFA from integers in $[-100, 100]$.

The answer to a membership query is the output value of the target SWFA. The answer to an equivalence query is computed as in Section 3.4. To find counterexample, we input randomly generated strings up to length 20 to both the target SWFA and the hypothesis. The counterexample is the first string that the target SWFA and the hypothesis output different values.

We used memorization to suppress to ask same membership queries. We used linear search to find a prefix that satisfies equations (4.1) and (4.2). This makes worse the worst query complexity but practically works better than binary search.

For $\mathscr{G}^{poly}$, we conducted experiments for degrees of polynomials $k \in \{1, 2\}$. We used



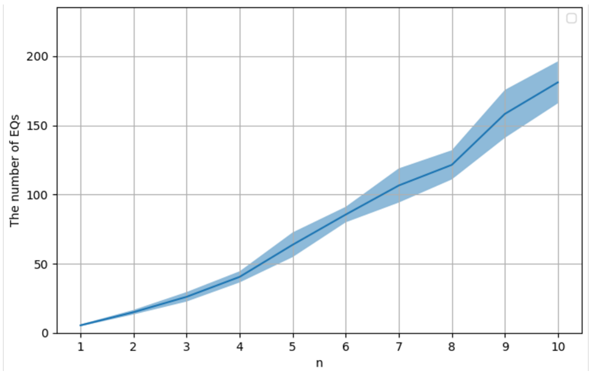
(a) EQ (b) MQ

Figure 4.2: The practical performance of the proposed algorithm for $\mathscr{G}^{poly}$. The number of (a) EQs and (b) MQs.

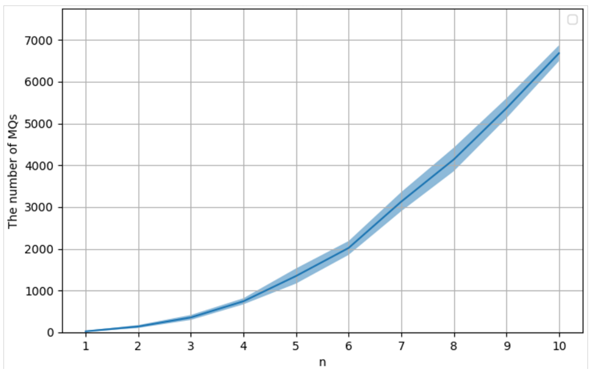
integers in $[-32768, 32767]$ as Σ. We measured the number of queries asked by our algorithm for the number of states n in $[1, 20]$. To learn guard functions, we used a simple polynomial curve fitting algorithm as Λ. The query complexity is $k + 1$ for both of $\mathscr{G}$-MQs and $\mathscr{G}$-EQs.

For $\mathscr{G}^{div}$, we fixed the partition number $l = 1$. That is, the input space is split by a boundary and has two divisions with different weights. We used integers in $[-32, 31]$ as Σ. We used n in $[1, 10]$ since $n > 10$ requires too long computation time. To learn guard functions, we used a dicision-tree based algorithm. The query complexity is $O(|L| \log \hat{l})$ for both of $\mathscr{G}$-MQs and $\mathscr{G}$-EQs. Because of implementation reasons, this is worse than the one shown in Example 3 by $\log \hat{l}$ for $\mathscr{G}$-EQs.

We used rational numbers as the weight space. This allows us to avoid the error accumulation problem that frequently appears in the context of learning sequential data. To maintain this feature while learning, we used fractions for weights and polynomial coefficients in the hypothesis. We implemented our algorithm in Python, and such a rational computation is realized by the python library SymPy [21].



(a) EQ



(b) MQ

Figure 4.3: The practical performance of the proposed algorithm for $\mathscr{G}^{div}$. The number of (a) EQs and (b) MQs.

4.4.2 Results

We conducted each experiment 10 times. Figures 4.2 and Figure 4.3 show the average numbers of EQs and MQs raised by our algorithm for each n with standard deviation for $\mathscr{G}^{poly}$ and $\mathscr{G}^{div}$, respectively. We observed that the number of queries of our algorithm is quite stable for $\mathscr{G}^{poly}$, thus standard deviation is too small to be apparent. Both results shows that when compared to the theoretical query complexity in Table 4.1, the practical performances are much better. Especially, the number of EQs looks almost linear in n, despite that the worst-case complexity is cubic in n. The result of $\mathscr{G}^{poly}$ in Figure 4.2 shows that our algorithm works well for large alphabet size.

4.4.3 Discussion

In this section, we discuss the reason of good empirical performance of our algorithm. We monitored the behavior of the proposed algorithm during experiments and observed a kind of "counterexample sharing" phenomenon. Our algorithm finds some input symbol x in Line 1 of Algorithm 5 as a candidate of a counterexample for instances of Λ. In the worst case, this x is a counterexample for only one instance of Λ. However, in practice, we observed that picked symbol x worked as counterexamples for many instances of Λ. In such a case, our algorithm returns this x to all instances of Λ as in Line 7 of Algorithm

5. We call this phenomenon "counterexample sharing".

"Counterexample sharing" can explain the reason for linear dependence in n of EQ complexity. In the worst case complexity, EQ complexity depends on cubic in n. n^2 in this EQ complexity is raised from the number of instances of Λ. That is, our algorithm requires n^2 EQs to answer a $\mathscr{G}$-EQ from each instance of Λ since the number of instances is n^2. When "counterexample sharing" occurs, our algorithm can answer $\mathscr{G}$-EQs from many instances of Λ by only one EQ. This may suppress the EQ complexity by at most n^2 scale and bring the linear dependency in experiments.

Chapter 5

Spectral Learning of Symbolic Weighted Finite Automata

5.1 Problem Setup

In the previous chapter, we focused on query learning of SWFAs, which can be regarded as a kind of *exact learning*. On the other hand, *approximate learning* of SWFAs is also an interesting research direction. Actually, in the field of WFAs, approximate learning is one of the central research areas [4, 5], and there is a well-known algorithm called *spectral learning* algorithm. In the typical setting of spectral learning, we have a dataset $Z = \{(w_1, y_1), (w_2, y_2), \dots, (w_m, y_m)\}$, where $w_i \in \Sigma^*$ and $y_i \in \mathbb{R}$, for all $i \in [1, m]$. Note that we only consider the set of real numbers $\mathbb{R}$ as output spaces for the singular value decomposition (SVD), which will be required later. Our goal is to learn a power series $f \in \mathbb{R}^{\Sigma^*}$ that explains relationship between w_i and y_i. That is, we aim to learn f which minimizes the error $\sum_{1 \le i \le m} (f(w_i) - y_i)^2$. As before, we assume this relationship can be represented by a $\mathscr{G}$-recognizable power series for some known $\mathscr{G}$. We assume that a mask (P, S) where $P, S \subseteq \Sigma^*$ and a set of symbols $\sigma \subseteq \Sigma$ are given as hyperparameters. In addition, we assume $\epsilon \in P$ and $\epsilon \in S$. Let $\sigma_\epsilon = \sigma \cup \{\epsilon\}$. Let $Px = \{px \mid p \in P\}$ for some

$x \in \sigma_\epsilon$ and $P\sigma_\epsilon = \bigcup_{x \in \sigma_\epsilon} Px$. Given the dataset, we can set the values of a matrix $\mathbf{H}_{(P\sigma_\epsilon, S)}$ by $\mathbf{H}_{(P\sigma_\epsilon, S)}[p, s] = y_i$ if $ps = w_i$. Obviously, this matrix may contain missing values if $ps \neq w_i$ for any $i \in [1, m]$. How to deal with this missing value problem will be discussed later. After properly processing missing value, we can construct an SWFA representing f from this matrix by the spectral learning algorithm as in the next section.

5.2 Spectral Learning Algorithm

In this section, we describe an algorithm to construct an SWFA from $\mathbf{H}_{(P\sigma_\epsilon, S)}$. At this time, we assume that this matrix does not contain any missing value. Although we can use the construction in Definition 3.16, this construction have some problems in the setting of spectral learning. First, Definition 3.16 is well-defined only if (P, S) is a non-singular mask. In the query learning algorithm, it is not a problem since our proposed algorithm can extend (P, S) while keeping it non-singular. However, for spectral learning, this assumption introduces an additional limitation for given hyperparameters. The second problem of Definition 3.16 is that this construction may produce an SWFA with too many states. Definition 3.16 constructs an SWFA with $|P| = \mathrm{rank}(\mathbf{H}_{(P,S)})$ states. However, in the setting of approximate learning, noise in the dataset may often make a matrix full-rank regardless of the actual rank of the matrix. Therefore, the rank information of a matrix may be unhelpful. Because of these problems, we need a new way to construct an SWFA from $\mathbf{H}_{(P\sigma_\epsilon, S)}$. We specify the number of states n and define the constructed SWFA as the following definition.

Definition 5.1. For a mask (P, S) and an integer n, let $U \in \mathbb{R}^{P \times n}$ and $V \in \mathbb{R}^{S \times n}$ be orthogonal matrices, respectively. Let $D \in \mathbb{R}^{n \times n}$ be a diagonal matrix containing positive values for diagonal elements. The SWFA based on these matrices is defined to be $\mathcal{A}_{(P,S)} = (\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta)$, where $Q = \{1, 2, \dots, n\}$, $\vec{\alpha}^\top = \mathbf{H}_{(\{\epsilon\}, S)} V$, $\vec{\beta} = D^{-1} U^\top \mathbf{H}_{(P, \{\epsilon\})}$ and $\Delta_x = D^{-1} U^\top \mathbf{H}_{(P,x,S)} V$ for all $x \in \Sigma$.

The famous *spectral learning* algorithm utilizes the *Singular Value Decomposition (SVD)* such as $\mathbf{H}_{(P,S)} \approx UDV^\top$ as matrices used in Definition 5.1 [4, 5]. When the given mask (P,S) is a basis and this decomposition reconstructs $\mathbf{H}_{(P,S)}$ perfectly, the constructed SWFA $\mathcal{A}_{(P,S)}$ correctly represents $\mathscr{G}$-recognizable power series f. In addition, all guard functions in Δ of $\mathcal{A}_{(P,S)}$ are in $\mathscr{G}$.

Proposition 5.2. *Let (P,S) be a basis. Let the SVD perfectly reconstruct $\mathbf{H}_{(P,S)}$ as $\mathbf{H}_{(P,S)} = UDV^\top$. Then, $\mathcal{A}_{(P,S)} = (\mathscr{G}, Q, \vec{\alpha}, \vec{\beta}, \Delta)$ correctly represents f. Moreover, all guard functions in Δ of $\mathcal{A}_{(P,S)}$ are in $\mathscr{G}$.*

Proof: By Theorem 3.15, there is an SWFA $\mathcal{A}' = (\mathscr{G}, Q', \vec{\alpha}', \vec{\beta}', \Delta')$ with n states representing f, i.e., $f(w) = \vec{\alpha}'^\top \Delta'_w \vec{\beta}'$ for all $w \in \Sigma^*$. Define matrices $U_P \in \mathbb{F}^{P \times Q'}$ and $V_S \in \mathbb{F}^{Q' \times S}$ by $U_P[p,\cdot] = \vec{\alpha}'^\top \Delta'_p$ for all $p \in P$ and $V_S[\cdot,s] = \Delta'_s \vec{\beta}'$ for all $s \in S$, respectively. Then we have $\mathbf{H}_{(P,S)} = U_P V_S$ and $\mathbf{H}_{(P,x,S)} = U_P \Delta'_x V_S$ for $x \in \Sigma$. From the definition of $\mathcal{A}_{(P,S)}$, $\Delta_x = D^{-1} U^\top \mathbf{H}_{(P,x,S)} V = D^{-1} U^\top U_P \Delta'_x V_S V$. Note that $\mathbf{H}_{(P,S)}$ is non-singular since we assume $\mathbf{H}_{(P,S)} = UDV^\top$. Using this fact, we find $V_S V D^{-1} U^\top U_P = V_S \mathbf{H}^{-1}_{(P,S)} U_P = I$. Thus, we can extend $\Delta_x = D^{-1} U^\top U_P \Delta'_x V_S V$ to strings $w \in \Sigma^*$ as $\Delta_w = D^{-1} U^\top U_P \Delta'_w V_S V$. Since $\mathbf{H}_{(\{\epsilon\},S)} = \vec{\alpha}'^\top V_S$ and $\mathbf{H}_{(P,\{\epsilon\})} = U_P \vec{\beta}'$, we have $\vec{\alpha}^\top = \vec{\alpha}'^\top V_S V$ and $\vec{\beta} = D^{-1} U^\top U_P \vec{\beta}'$. Therefore, $\vec{\alpha}^\top \Delta_w \vec{\beta} = (\vec{\alpha}'^\top V_S V)(D^{-1} U^\top U_P \Delta'_w V_S V)(D^{-1} U^\top U_P \vec{\beta}') = \vec{\alpha}'^\top I \Delta'_w I \vec{\beta}' = \vec{\alpha}'^\top \Delta'_w \vec{\beta}' = f(w)$. Additionally, $\Delta_x = D^{-1} U^\top U_P \Delta'_x V_S V$ shows that every guard function in Δ can be represented by a linear transformation of those in Δ'. Thus, all guard functions in Δ are in $\mathscr{G}$ by Assumption 1. ■

Definition 5.1 requires the transition relation Δ satisfies $\Delta_x = D^{-1} U^\top \mathbf{H}_{(P,x,S)} V$ for all $x \in \Sigma$. In the spectral learning of WFAs, all we need is to compute matrices Δ_x for all $x \in \Sigma$ following the definition. On the other hand, for SWFAs, we need to prepare guard functions in Δ. That is, the additional procedure is required to learn such guard functions. Let $\Delta^t_x = D^{-1} U^\top \mathbf{H}_{(P,x,S)} V$ for $x \in \sigma$. We assume a supervised learning algorithm λ to learn guard functions in $\mathscr{G}$ from a dataset. The set of pairs $\{(x, \Delta^t_x[i,j]) \mid x \in \sigma\}$ can be regarded as the dataset for a guard function at $i,j \in [1,n]$. Then, for all $i,j \in [1,n]$, we utilize λ and learn a guard function which minimizes $\sum_{x\in\sigma}(\Delta[i,j](x)-\Delta^t_x[i,j])^2$. By this modification, the computation cost of this algorithm no longer depends on the alphabet size. However, sufficient generalization of learned guard functions is required to work well for all $x \in \Sigma$, beyond $x \in \sigma$.

The overall procedure of spectral learning of SWFAs is shown below. Missing value completion at Lines 2-5 will be explained in the next section.

Algorithm 6: spectral learning of SWFAs

```
  Input: a dataset $Z$, a mask $(P, S)$, a set of symbols $\sigma$, an integer $n$, a supervised
         learning algorithm $\lambda$
  Output: an SWFA $\mathcal{A}$
1 initialize $\mathbf{H}_{(P\sigma_\epsilon,S)} \leftarrow O$ ;
2 if the target power series is a probability distribution then
3   $\mathbf{H}_{(P\sigma_\epsilon,S)}[w] = \frac{1}{|Z|}\sum_{w\in Z}\mathbb{I}[w_i = w]$
4 else
5   $\mathbf{H}_{(P\sigma_\epsilon,S)} = \underset{\mathbf{H}_{(P\sigma_\epsilon,S)}\in\mathbb{H}_{(P\sigma_\epsilon,S)}}{\operatorname{argmin}} L_{l,\rho,\tau}(\mathbf{H}_{(P\sigma_\epsilon,S)})$;
6 compute the best rank $n$ approximation of $\mathbf{H}_{(P,S)} \approx UDV^\top$ by the SVD ;
7 $\vec{\alpha}^\top = \mathbf{H}_{(\{\epsilon\},S)}V$; $\vec{\beta} = D^{-1}U^\top\mathbf{H}_{(P,\{\epsilon\})}$; $\Delta^t_x = D^{-1}U^\top\mathbf{H}_{(P,x,S)}V$ for all $x \in \sigma$ ;
8 determine functions in $\Delta$ by $\lambda$ with a dataset $\{(x, \Delta^t_x[i,j]) \mid x \in \sigma\}$, $\forall i,j \in [1,n]$;
9 return $\mathcal{A} = (\mathscr{G}, Q = \{1,2,\ldots,n\}, \vec{\alpha}, \vec{\beta}, \Delta)$;
```

5.3 Missing Value Completion

When a matrix $\mathbf{H}_{(P\sigma_\epsilon,S)}$ contains missing values, we need to complement them before applying the construction in Definition 5.1.

5.3.1 Learning Probability Distribution

There is a trivial completion strategy for the problem learning probability distribution over strings. Let $p(w)$ be a probability distribution over strings and a dataset $Z = (w_1, w_2, \ldots, w_m)$, where $w_i \sim p(w)$ for all $i \in [1,m]$. Here, we aim to learn an SWFA representing $p(w)$. For this, the natural estimate of output value y_i for the input string w_i is the sample mean $y_i = \frac{1}{|Z|}\sum_{w \in Z} \mathbb{I}[w_i = w]$, where $\mathbb{I}$ is the identity function. For a given mask (P, S) and a set of symbols σ, we substitute values in $\mathbf{H}_{(P\sigma_\epsilon, S)}$ by this esimated probability (Line 3 of Algorithm 6). Most importantly, the sample mean $\frac{1}{|Z|}\sum_{w \in Z} \mathbb{I}[\hat{w} = w]$ is 0 for $\hat{w} \notin Z$, and this is a natural estimate for strings that never appeared in the dataset. Thus, we do not need an explicit algorithm to complement missing values.

5.3.2 Learning General Power Series

In general, we need to use a matrix completion algorithm to deal with the missing value problem. Then, we consider to apply a matrix completion algorithm to $\mathbf{H}_{(P\sigma_\epsilon, S)}$. Now, we introduce a simple algorithm adopted in [5]. Note that $\mathbf{H}_{(P\sigma_\epsilon, S)}[p, s] = \mathbf{H}_{(P\sigma_\epsilon, S)}[p', s']$ if $ps = p's'$ because of its definition. Therefore, this is a constrained matrix completion problem. Let

$$\mathbb{H}_{(P\sigma_\epsilon, S)} = \{\mathbf{H}_{(P\sigma_\epsilon, S)} \in \mathbb{R}^{P\sigma_\epsilon \times S} \mid \forall p, p' \in P\sigma_\epsilon, \forall s, s' \in S, ps = p's' \Rightarrow \mathbf{H}_{(P\sigma_\epsilon, S)}[p, s] = \mathbf{H}_{(P\sigma_\epsilon, S)}[p', s']\}.$$

$\mathbb{H}_{(P\sigma_\epsilon, S)}$ represents the set of matrices which satisfies the constraint explained above. This set is *convex* because it is a subset of a convex space defined by equality constraints. One typical method for such a problem is to define a convex loss function and optimize it by the gradient descent method.

Let $\tilde{Z}$ be the subsample of Z by examples (w, y) where $w = ps$ for some $p \in P\sigma_\epsilon$ and $s \in S$. Let $\mathbf{H}_{(P\sigma_\epsilon, S)}[w]$ be $\mathbf{H}_{(P\sigma_\epsilon, S)}[p, s]$ such that $w = ps$. Balle and Mohri [5] adopts loss function with a norm regularization as below,

$$L_{l,\rho,\tau}(\mathbf{H}_{(P\sigma_\epsilon, S)}) = \frac{1}{\tilde{Z}} \sum_{(w,y) \in \tilde{Z}} l(\mathbf{H}_{(P\sigma_\epsilon, S)}[w] - y) + \tau\rho(\mathbf{H}_{(P\sigma_\epsilon, S)}) \tag{5.1}$$

where l is some distance function (*e.g.* mean squared error), ρ is some matrix norm, and τ is the weight for the term of the matrix norm. The first term of Equation (5.1) brings the values of the matrix in which the data exists closer to the target values in the dataset.

When combined with the second term regularizing the matrix norm, optimization over this loss function is expected to complement missing values by plausible values. One typical choice of ρ is the Frobenius norm, defined as the square root of the sum of square values in the matrix.

$$\|A\|_F = \sqrt{\sum_{i,j} A[i,j]^2}.$$

This norm suppresses large values in the matrix, which is typical regularization in the machine learning context. In addition, this norm brings missing values closer to 0.

Another choice of ρ is the nuclear norm, defined as the sum of singular values of the matrix.

$$\|A\|_* = \sum_i d_i(A)$$

where $d_i(A)$ is the i-th singular value of the matrix A. The minimization of this norm is known to reduce the rank of the matrix [14]. This is conceptually plausible since a smaller rank means an SWFA with fewer states because of Theorem 3.15. That is, this regularization finally leads a simpler hypothesis.

We specifiy l, ρ, and τ as hyperparameters and complement missing values by finding $\mathbf{H}_{(P\sigma_\epsilon,S)}$ which minimizes $L_{l,\rho,\tau}$ in $\mathbb{H}_{(P\sigma_\epsilon,S)}$ (Line 5 of Algorithm 6).

5.4 Experiments

We conducted experiments to evaluate the performance of our spectral learning algorithm for SWFAs. The purpose of the experiments is to answer the following questions.

- **Q1**: Do SWFAs trained by our algorithm perform well compared to WFAs? Especially, does the guard function class $\mathscr{G}$ of SWFAs limit performance?
- **Q2**: Does our algorithm work well with small $\sigma \subseteq \Sigma$, that is, fewer data?
- **Q3**: Does our algorithm work well when there are missing values?

We designed toy problems to answer these questions. We prepared randomly initialized recurrent neural networks (RNN) as teachers and approximated them using the spectral learning algorithm. This is an idealized problem setup since we can get a response to any input by asking the teacher, but the algorithm we experimented here can be used for more difficult setting, such as only given information is limited amount of data.

5.4.1 Common Setting

We adopted long short-term memory (LSTM) [17] and gated recurrent unit (GRU) [10] as the teacher RNNs. We fixed the number of hidden states to 100 and the number of layers to 1. All parameters in the teacher RNNs are randomly initialized by the normal distribution $\mathcal{N}(0,1)$.

Throughout the experiments, we used $\mathscr{G}^{poly}$ as the guard function class. We used the simple least squares method as the supervised learning algorithm λ. We created alphabet Σ such as $|\Sigma| = 1024$ by dividing $[-10, 10]$ at regular intervals. We fixed the sizes of hyperparameters P and S to 10, and determined them by randomly sampling from strings up to length 2. For n that specifies the number of states of the finally generated automata, we tried $n \in \{1, 2, \dots, 8\}$ and chose the best parameter for each experiment.

We randomly generated 1000 strings up to length 10 and computed the output of the teacher RNN for them. We used this set of pairs as the test set to evaluate and compare the performance of the proposed algorithm. We adopted root mean squared error (RMSE) $\sqrt{\frac{1}{\hat{Z}} \sum_{(w,y)\in\hat{Z}} (f_{\mathcal{A}}(w) - y)^2}$ as the evaluation metric, where $\hat{Z}$ is the test set and $f_{\mathcal{A}}$ is the power series represented by the learned automaton $\mathcal{A}$. We conducted each experiment 10 times with different teacher RNNs and (P, S), and compute the mean of the results.

5.4.2 Q1: Comparison to WFAs

This experiment examines the performance of our spectral learning algorithm for SWFAs compared to the one for WFAs. Here, we consider the setting with no missing values. For

Table 5.1: RMSE of the learned WFA and SWFA.

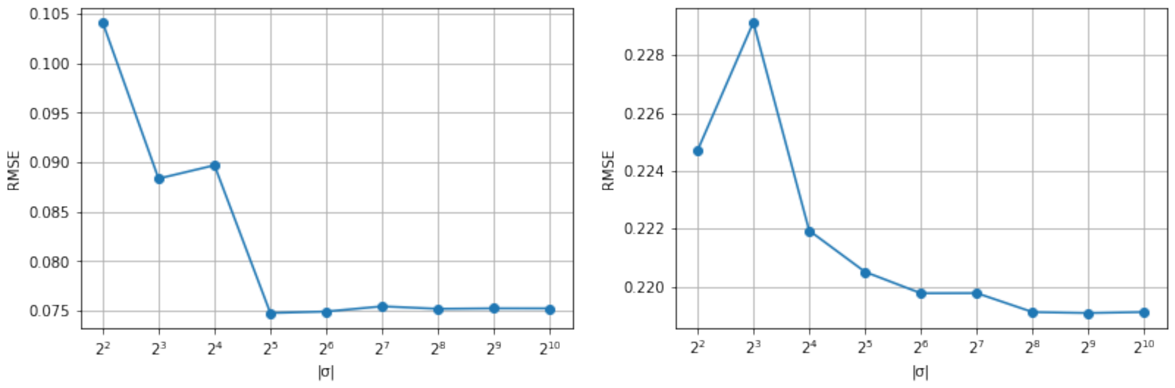
	LSTM	GRU	#Params
WFA	0.0655	0.1187	36876
SWFA ($k = 1$)	0.1600	0.2380	84
SWFA ($k = 2$)	0.0791	0.1380	120
SWFA ($k = 3$)	0.0669	0.1196	156
SWFA ($k = 4$)	0.0657	0.1188	192
SWFA ($k = 5$)	0.0655	0.1187	228

a fair comparison to WFAs, we set $\sigma = \Sigma$. We filled the values in the matrix $\mathbf{H}_{(P\Sigma,S)}$ by asking to the teacher RNNs and constructed WFAs and SWFAs by the spectral learning algorithm. Note that the spectral learning algorithm for WFAs is almost the same as for SWFAs except for skipping on Line 8 of Algorithm 6. To examine the effect of the limitation by the guard function class, we conducted experiments with the different degree of polynomial k of $\mathscr{G}^{poly}$.

We show results of WFAs and SWFAs with $k \in \{1, 2, 3, 4, 5\}$ for LSTM and GRU in Table 5.1. In this experiment, the best number of states is $n = 6$ for all cases. The results of SWFAs are steadily improving as k increases and expressiveness improves. When $k = 5$, SWFAs achieve almost the same results as WFAs for both teacher RNNs. We also show the number of total parameters in Table 5.1. It can be seen that SWFAs achieve almost the same RMSE with a much smaller number of parameters. This difference comes from the fact that the general form of the number of parameters of SWFAs is $2n + (k + 1)n^2$ and that of WFAs is $2n + |\Sigma|n^2$.

5.4.3 Q2: Smaller σ

This experiment examines the performance of our spectral learning algorithm for SWFAs with smaller σ, that is, fewer data. As before, we consider the setting with no missing values. We conducted experiments with $|\sigma| = 4, 8, \ldots, 1024$. We determined σ by randomly sampling a specified number of symbols from Σ. From the findings of the previous experiment, we used SWFAs with $\mathscr{G}^{poly}$ with $k = 3$, which showed enough results with



(a) LSTM (b) GRU

Figure 5.1: RMSE with different size of σ

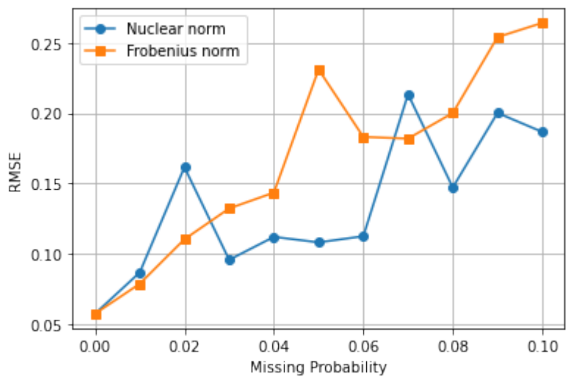
fewer parameters.

We show results for (a) LSTM and (b) GRU in Figure 5.1. In this experiment, the best number of states is $n = 6$ for all cases. The results show that our algorithm works well until $|\sigma| = 2^5 = 32$ or $|\sigma| = 2^6 = 64$. This is much data efficient compared to WFAs which always requires $\sigma = \Sigma$, that is, $|\sigma| = 1024$ to avoid additional matrix completion procedure. Note that actual required amount of data is $|P| \times |S| = 100$ times to fill the matrix $\mathbf{H}_{(P\sigma_\epsilon, S)}$.

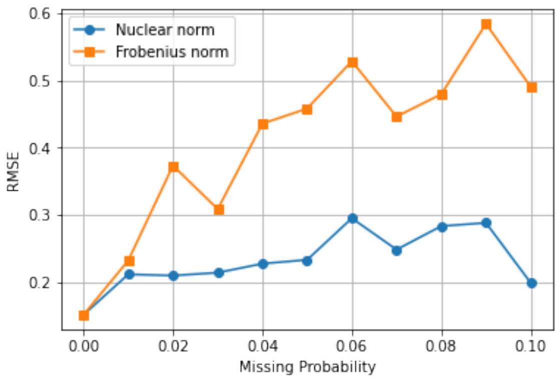
5.4.4 Q3: Experiments with Missing Values

This experiment examines the performance of our algorithm under the problem setup containing missing values. We consider the setting that we lost information from the teacher RNN with some "missing probability". We changed the missing probability from 0 to 0.1 in 0.01 increments. In this experiment, we fixed $|\sigma| = 32$. We used the matrix completion procedure explained in Section 5.4. For optimization, we adopted the gradient descent algorithm with learning rate 0.1 and $\tau = 0.0001$. We determined these values from the results of some preliminary experiments. We tried both of the Frobenius norm and the nuclear norm to compare results.

We show results for (a) LSTM and (b) GRU in Figure 5.2. In this experiment, the



(a) LSTM



(b) GRU

Figure 5.2: RMSE with the different missing probability

best number of states is $n = 5$ for all cases. It can be seen that RMSE is considerably larger than when there are no missing values, but the completion procedure works at some extent. Both norms show similar results for LSTM, but the nuclear norm works better for GRU. Combining the results in the previous section, we can see that our algorithm can learn SWFAs with a considerably limited amount of data. However, we guess that some additional devices will be required to work well for the problem that only a static dataset is available.

Chapter 6

Conclusion and Future Work

We considered symbolic weighted finite automata (SWFAs), which unify SFAs and WFAs. SWFAs can deal with a possibly infinite alphabet efficiently taking advantage of the structure of the alphabet. To clarify the representation range of SWFAs, we defined $\mathscr{G}$-rational power series and prove the Kleene–Schützenberger theorem for SWFAs. We also confirmed that the minimization and equivalence checking for SWFAs can be achieved efficiently. We proposed a new query learning algorithm for SWFAs by combining the ones for SFAs and WFAs with a correctness proof and an upper bound for the number of queries. We conducted some experiments to evaluate the practical performance of our algorithm and observed much better performance than the worst-case complexity. As another learning manner, we also proposed a spectral learning algorithm for SWFAs by extending an existing one for WFAs. We conducted some experiments and observed that our algorithm learns well-worked SWFAs with much fewer parameters from fewer data compared to WFAs.

There are many potential applications of our algorithms as extensions of applications using WFAs. As we covered in the experiments in Section 5.4, an interesting application is extracting an SWFA from a recurrent neural network (RNN) to obtain a faster surrogate. A prior work [23] uses WFAs for this purpose, but the range of applicable RNNs is limited by the size of the alphabet. SWFAs and our algorithms are suitable to extract automata from RNNs over a large or infinite alphabet. It is useful since RNNs with real-valued inputs are very common for time series analysis.

Since our experiments are currently limited to the range of toy problems, one of the important future works is to conduct more realistic experiments with a view of applications. As another point of view, some paper investigates theoretical nature of WFAs and research about the upper bound of generalization error from a point of stability analysis [5] or Rademacher complexity [7]. Extending such theoretical results to the case of SWFAs is a hopeful research direction.

References

[1] Rajeev Alur, Konstantinos Mamouras, and Caleb Stanford. Automata-based stream processing. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, volume 80 of *LIPIcs*, pages 112:1–112:15, 2017.

[2] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[3] George Argyros and Loris D'Antoni. The learnability of symbolic automata. In *30th International Conference on Computer Aided Verification, CAV 2018*, pages 427–445, 2018.

[4] Borja Balle, Xavier Carreras, Franco M. Luque, and Ariadna Quattoni. Spectral learning of weighted automata - A forward-backward perspective. *Machine Learning*, 96(1-2):33–63, 2014.

[5] Borja Balle and Mehryar Mohri. Spectral learning of general weighted automata via constrained matrix completion. In *26th Annual Conference on Neural Information Processing Systems, NeurIPS 2012*, pages 2168–2176, 2012.

[6] Borja Balle and Mehryar Mohri. Learning weighted automata. In *6th International Conference on Algebraic Informatics, CAI 2015*, pages 1–21, 2015.

[7] Borja Balle and Mehryar Mohri. On the rademacher complexity of weighted automata. In *26th International Conference of Algorithmic Learning Theory, ALT 2015*, pages 179–193, 2015.

[8] Francesco Bergadano and Stefano Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. *SIAM Journal on Computing*, 25(6):1268–1280, 1996.

[9] Laurence Bisht, Nader H. Bshouty, and Hanna Mazzawi. On optimal learning algorithms for multiplicity automata. In *19th Annual Conference on Learning Theory, COLT 2006*, pages 184–198, 2006.

[10] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014*, pages 1724–1734, 2014.

[11] Kaizaburo Chubachi, Diptarama Hendrian, Ryo Yoshinaka, and Ayumi Shinohara. Query learning algorithm for residual symbolic finite automata. In *Proceedings Tenth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2019*, pages 140–153, 2019.

[12] Corinna Cortes, Mehryar Mohri, and Ashish Rastogi. L_p distance and equivalence of probabilistic automata. *International Journal of Foundations of Computer Science*, 18(4):761–779, 2007.

[13] Samuel Drews and Loris D'Antoni. Learning symbolic automata. In *23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2017*, pages 173–189, 2017.

[14] Maryam Fazel. *Matrix rank minimization with applications.* PhD thesis, Stanford University, 2002.

[15] Michel Fliess. Matrices de Hankel. *Journal de Mathématiques Pures et Appliquées*, 53, 1974.

[16] Luisa Herrmann and Heiko Vogler. Weighted symbolic automata with data storage. In *20th International Conference on Developments in Language Theory , DLT 2016*, volume 9840 of *Lecture Notes in Computer Science*, pages 203–215, 2016.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[18] Stefan Jaksic, Ezio Bartocci, Radu Grosu, Thang Nguyen, and Dejan Nickovic. Quantitative monitoring of STL with edit distance. *Formal Methods in System Design*, 53(1):83–112, 2018.

[19] Stefan Jaksic, Ezio Bartocci, Radu Grosu, and Dejan Nickovic. An algebraic framework for runtime verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2233–2243, 2018.

[20] Stephen Cole Kleene. Representation of events in nerve nets and finite automata. *Automata Studies, Annals of Mathematics Studies*, 34:3–42, 1956.

[21] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[22] Anil Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9:541–544, 1958.

[23] Takamasa Okudono, Masaki Waga, Taro Sekiyama, and Ichiro Hasuo. Weighted automata extraction from recurrent neural networks via regression on state spaces. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, pages 5306–5314, 2020.

[24] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, 1993.

[25] Marcel Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, 1961.

[26] Masaki Waga. Online quantitative timed pattern matching with semiring-valued weighted automata. In *17th International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS 2019*, volume 11750 of *Lecture Notes in Computer Science*, pages 3–22, 2019.

Acknowledgements

I would like to express my sincere gratitude to Prof. Ayumi Shinohara, Prof. Ryo Yoshinaka, and Prof. Diptarama Hendrian for their strong support throughout my research.

I wish to thank the members of my thesis committee Prof. Kentaro Inui and Prof. Xiao Zhou for their valuable advice.

I am also grateful to my fellow students and family for supporting me throughout writing this thesis and my life in general.

Publications

1. Kaito Suzuki, Diptarama Hendrian, Ryo Yoshinaka, and Ayumi Shinohara. Query Learning Algorithm for Symbolic Weighted Finite Automata. In the *15th International Conference on Grammatical Inference*, *ICGI 2020/21*, Volume 153 of *Proceedings of Machine Learning Research*, pp.202-216, 2021.

2. Kaito Suzuki, Diptarama Hendrian, Ryo Yoshinaka, and Ayumi Shinohara. Query Learning of Symbolic Weighted Finite Automata. In the *14th Annual Meeting of the Asian Association for Algorithms and Computation*, *AAAC 2021*, 2021.