FAKULTÄT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Lehrstuhl für Sicherheit in der Informatik

Learning Probabilistic Subsequential Transducers

Hasan Ibne Akram

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ-Prof. Dr. Helmut Seidl

Prüfer der Dissertation:

1. Univ-Prof. Dr. Claudia Eckert
2. Univ-Prof. Dr. Dr. h.c. Javier Esparza
3. Prof. Dr. Colin de la Higuera,
   Universit de Nantes, Frankreich

Die Dissertation wurde am 29.10.2012 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 11.03.2013 angenommen.

Abstract (English)

Probabilistic transducers or weighted transducers are widely applied in the fields of natural language processing, machine translation, IT security, bioinformatics and many other areas. The generality of probabilistic transducers allow them capturing other existing probabilistic models such as hidden Markov models. Due to the widespread use and applicability of probabilistic transducers, the learnability issue of such models is an important problem.

In this thesis we investigate inference of probabilistic subsequential transducers in an active learning environment. First, we propose a novel inference algorithm where the learner interacts with an oracle by asking probabilistic queries on the observed data. We prove our algorithm in an identification in the limit model. We also provide experimental evidence to show the correctness and to analyze the learnability of the proposed algorithm. Second, we propose another learning algorithm where the oracle can be replaced by statistical tests over observed data. We report some experiments with synthetic datasets. Finally, related work concerning the implementation of grammatical inference algorithms (optimisation, parallelisation, distribution) is reported in this thesis.

Kurzfassung (Deutsch)

Probabilistische bzw. gewichtete Transducer finden verbreitet Anwendung in den verschiedenen Gebieten der maschinellen bersetzung, Verarbeitung natrlicher Sprache, IT-Sicherheit und Bioinformatik. Probabilistische Transducer sind in der Lage existierende probabilistische Modelle, wie z.B. Hidden-Markov-Modelle, abzudecken. Den Schwerpunkt dieser Arbeit bildet die Erlernbarkeit von probabilistischen Transducern. Eine Problemstellung, die aufgrund der weiten Verbreitung und der vielfltigen Anwendungsmglichkeiten relevant ist.

In der vorliegenden Arbeit untersuchen wir probabilistische subsequentielle Transducer in einer aktiven Lernumgebung. Zunchst stellen wir einen neuen Lernalgorithmus vor, der probabilistische Anfragen ber beobachtete Daten an ein Orakel stellt. Wir beweisen unseren Algorithmus durch eine Identifikation im Limit-Modell. Um die Korrektheit nachzuweisen und die Erlernbarkeit des vorgeschlagenen Algorithmus zu analysieren, liefern wir darber hinaus experimentelle Belege. Im Anschluss stellen wir einen weiteren Lernalgorithmus vor, bei dem das Orakel durch statistische Tests ber beobachteten Daten ersetzt wird. Danach fhren wir mit diesem Algorithmus einige Experimente mit synthetischen Datenstzen durch. Schließlich diskutieren wir verwandte wissenschaftliche Arbeiten im Hinblick auf die Implementierung von grammatikalischen Lernalgorithmen. Hierbei gehen wir auf Aspekte wie Optimierung, Parallelisierung und Verteilung ein.

Acknowledgments

This thesis is the result of three years of hard work together with encouragements and supports I have received from a number of people during these three years. First of all, I would like to thank two people: Prof. Claudia Eckert and Prof. Colin de la Higuera for their tremendous support during the last three years. I am greatly thankful to Prof. Claudia Eckert for giving me the opportunity to pursue my PhD in her group and providing me with all kinds of support to reach my objective. I am also immensely thankful to Prof. Colin de la Higuera for his scientific guidance to carry out my work and allowing me the opportunity to have research visits in his group at the University of Nantes. I owe both of them debt of gratitudes that I can never pay back. I would also like to sincerely thank Prof. Javier Esparza for accepting to be my advisor in a short notice and also for his valuable feedback. This thesis would not have been possible without the supports from these people.

I would like to take the opportunity to thank my colleague Thomas Stibor for bringing me enormous courage, harsh and honest criticism, and motivation. I was heavily inspired by his earnest dedication to science. I would also like to thank my other colleagues: Fatih Kilic, Huang Xiao, Krista Grothoff, Jonas Pfoh, Christian Schneider, Daniel Angermeier, Alexander Ldtke and Petra Lorenz for allowing a wonderful working environment and their encouragements. Last but not the least I would like to thank my parents, my sisters, my brother-in-law and my niece for being a part of my motivation and courage.

Contents

Chapter 1

Introduction

> Mathematics must be written so that it is impossible to misunderstand, not merely so that it is possible to understand.
>
> Herman Rubin

TRANSDUCERS, also known as finite state transducers (FSTs), can be viewed as finite state machines where whenever an input is read, a state transition takes place and an output is emitted. In a weighted or probabilistic setting, transducers have weights or probabilities assigned to each transition. The title of the thesis suggests that the thesis focuses on the problem of learning subsequential probabilistic transducers (PSTs). PSTs are probabilistic transducers where a probability is assigned to each transition and the machine generates no ambiguous input-output pairs. In other words, PSTs are FSTs deterministic *w.r.t.* the inputs with transition probabilities. In this chapter we will first motivate the learning of probabilistic transducers as a formal model by discussing some areas of applications of transducers and exhibiting a few motivating real world examples. Second, we discuss the term *learning* in general to develop the idea of how the term should be interpreted in the context of the thesis. Third, we will discuss learning in terms of mathematical modeling and learning environments where learning tasks can be achieved by means of computation. And finally, we will define the objectives and the organization of the thesis.

1.1. Motivation

The formal model we want to learn is a probabilistic subsequential transducer. The natural questions that arise at this point are: why do we want to learn transducers? What are the practical implications of learning such a model?

Transducers have been known in the computer science literature since as early as the 1950s (Mealy, 1955) and have been used in many different areas ever since. In this section a brief overview on diverse applicability of transducers is presented.

1.1.1. Natural Language Processing

In a specialized domain of natural language, namely in phonology, it was discovered that the morphological rules could be described as finite state models (Johnson, 1972). Therefore, a major application of FSTs in natural language processing is *finite state morphology* (Karttunen, 2000; Roark and Sproat, 2007).

Morphology is the area of computational linguistics where the formation of the words are studied. Morphological rules are the rules for alteration of morphology. For instance, the German word `schreiben` is written as `geschrieben` in its past participle form and the rule for rewriting such a verb in its past participle form is a morphological rule.

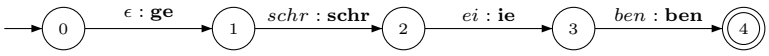


Figure 1.1.: An example of morphological rule expressed by an FST. The transition labels should be interpreted as *input* : **output**.

It has been advocated by Roark and Sproat in (2007) that transducers can be used to represent morphological rules. Figure 1.1 shows how the rule of forming `geschrieben` from the word `schreiben` can be expressed by an FST.

Clark proposed to learn morphology with stochastic transducers in (2001a; 2001b; 2002). In his work he showed that morphological rules such as formation of singular to plural in different languages, *e.g.*, German (*e.g.*, `kraft` → `krfte`) can be learnt using stochastic transducers.

1.1.2. Machine Translation

Machine translation is a problem of mapping an input language to an output language. It has been argued that the input-output relationship between most of the usual natural languages can be expressed by finite state devices (Casacuberta and Vidal, 2007). FSTs being capable of modeling input-output relations, these have been extensively investigated for accomplishing machine translation and transliteration tasks. Figure 1.2 depicts a toy example of English to French translation scheme represented by an FST. In this section we briefly discuss some relevant work where FSTs have been used for machine translation tasks.

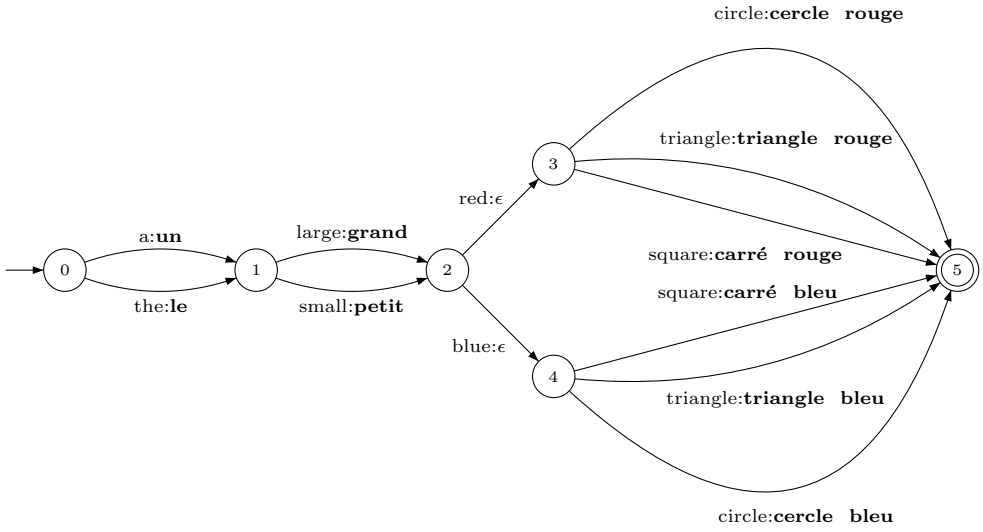


Figure 1.2.: An example of an FST representing an English-French translation scheme.

In an early attempt Vilar *et al.* (1996) implemented an English-Spanish speech translation system by means of learning a deterministic transducer from an aligned training corpus. The problem of alignment is a difficult issue while learning transducers, therefore, when heuristics based techniques from machine translation is used for the alignment task, it improves the prediction accuracy. Other attempts for the use of finite state models are (Casacuberta and Vidal, 2004; Vilar et al., 1996; Casacuberta and Vidal, 2007). In their work the idea was to combine statistical machine translation and transducer learning. Bangalore and Riccardi employed finite state models for spoken language machine

translation (2002; 2001).

Syntactically, natural languages are far more complex than finite state models (Chomsky, 1957). Syntactic structures of natural languages, such as English, are often modeled by more powerful formalisms. Therefore, there has been work done to model more powerful classes, for example, modeling with *context free grammars*. A generalization of a context free grammar to represent a pair of strings (similar to a transduction model) is known as *synchronous grammar*, originally developed for programming language compilation (Aho and Ullman, 1969). The tree transducer model has been extensively applied in various aspects (Koehn, 2010) including in different classes of synchronous grammars (Graehl et al., 2008; Knight and Graehl, 2005; Maletti, 2010). Maletti presented a tree transducer model for *synchronous tree-adjoining grammars* (2010), a class of synchronous grammar represented by a tree transducer model.

As far as the scope of this thesis is concerned we will mainly focus on regular *bi-grammar* or regular synchronous grammar, a subclass of synchronous grammar, that can be modeled by FSTs.

1.1.3. IT Security

One of the newly emerging application areas of grammatical inference is the domain of IT security. Within the recent years several papers have been published in the top tier security conferences where grammatical inference algorithms have been leveraged; *e.g.*, (Cho et al., 2010a; Babić et al., 2011, 2012). In this section we will discuss a motivating example of application of transducer learning in IT security.

Botnet Protocol Inference

Botnets are one of the major mechanisms of conducting cybercrimes such as spamming, denial of service attacks, theft of personal data *etc.*, which results in an annual damage of billions of dollars[1] world wide. Botnets are collections of infected or compromised computers that are connected to the Internet and perform certain tasks while the owners of the computers are unaware of it (for details see (Eckert, 2009; Anderson, 2008)).

The combating procedure of botnets can be broadly categorized into three interrelated sub-problems: 1) understanding and analyzing the botnet protocol, 2) detecting the botnet, and 3) taking measures to bring down the botnet. As far as detection of botnets and taking measures are concerned, a lot of work

[1]The economic impact of viruses, spyware, adware, botnets, and other malicious code: Technical report, Computer Economics Inc., 2007, `http://www.computereconomics.com`

has been done by the researchers, whereas relatively less work has been done in understanding and analyzing botnet protocols. Traditionally, for analyzing the botnet, protocol inference (*a.k.a.* protocol reverse engineering) is done manually, which is time consuming, error prone, and expensive.

Automated botnet protocol inference can ease a lot of manual work and be a handy tool to take rapid measure against botnets. It also helps analyzing the botnet to make effective detection mechanism and cost effective measure to bring down the botnets. Cho *et al.* attempted automated botnet protocol inference (2010a) where they have worked on the MegaD botnet protocols.

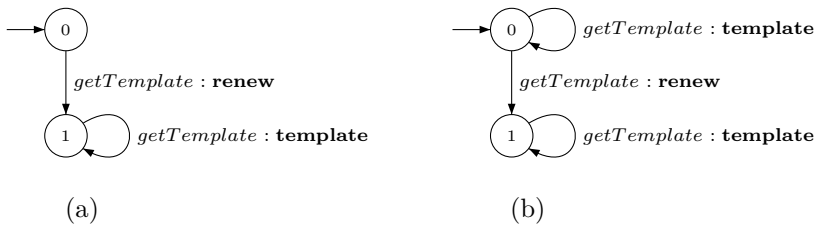


Figure 1.3.: The two figures are taken from (Cho et al., 2010a) where the protocol state machines of the MegaD template server are presented. Figure 1.3(a) is the inferred machine by using the template server as an oracle. By analyzing the inferred protocol state machines of other servers, they discovered that the actual protocol state machine for the template server is nondeterministic as depicted in Figure 1.3(b).

MegaD is a botnet that was first discovered in 2007. At its peak, MegaD was responsible for generating two thirds of the total spam all over the world in 2007[2]. The MegaD botnet involves a Master server used as the Command & Control (C&C) server, and other auxiliary servers: SMTP server, template server, and drop server. A spamming MegaD bot contacts only the template and SMTP servers (Cho et al., 2010b). Cho *et al.* presented in (2010a) a mechanism to infer protocol state machines of MegaD botnet servers by employing transducer learning by queries. Figure 1.3 depicts the protocol state machine of the MegaD botnet template server. Since the protocol state machine has input-output (request-response) for each transition, it can be formally modeled as a transducer.

In the inference model, the botnet server itself acts as an informant, *i.e.*, it answers by giving response to a particular request sent to it. The inference algorithm asks queries by communicating with the server and after a certain number

[2] `http://www.m86security.com/trace/i/Mega-D,spambot.896~.asp`

of queries is able to learn the protocol state machine of the server. Cho *et al.* (2010a) used an algorithm derived from the $L*$ algorithm (Angluin, 1987a,b), which is a well known algorithm for learning deterministic finite state automata by asking queries to an oracle, to learn the target transducer. The learning algorithm was able to infer a deterministic transducer shown in Figure 1.3(a) as the protocol state machine for the template server of MegaD. However, their analysis showed that the actual protocol state machine is nondeterministic as shown in Figure 1.3(b).

1.1.4. Bioinformatics

Bioinformatics is a research area devoted to modeling and analyzing biological sequences (Durbin et al., 1998). *Pairwise alignment* of DNA, RNA, or protein sequences and gene finding is an important and extensively studied problem in bioinformatics. It has been investigated that pair hidden Markov models (PHMMs) (Durbin et al., 1998), which are extension of regular hidden Markov models (HMMs) (Rabiner, 1990), can generate aligned pairs of sequences (Sakakibara, 2003; Pachter et al., 2002). PHMMs can be shown that they are equivalent to weighted or probabilistic transducers. Another early attempt that shows automata theoretic approach for biological sequence alignment is (Searls and Murphy, 1995). Several surveys can be found related to applications of weighted transducer or PHMM learning such as (Sakakibara, 2005; Bradley and Holmes, 2007; Coste, 2010; Searls, 2010).

Besides pairwise alignment, in another biological sequence prediction task, namely transmembrane domain prediction task, Peris and López have attempted learning deterministic transducers representing a regular language (Peris and López, 2010).

1.1.5. Other Application Areas

Finite state transducers (FSTs) have also been applied extensively in the field of speech processing (Mohri et al., 1996; Allauzen et al., 2007; Mohri et al., 2002), speech synthesis (Mohri et al., 2004; Pereira et al., 1994; Sproat, 1995), pattern recognition (Oncina et al., 1993), language understanding (Angluin and Becerra-Bonache, 2009; Castellanos et al., 1998), and machine learning (Cortes et al., 2002; Cortes and Mohri, 2008). One of the major attributes that makes FSTs very useful is that in weighted or probabilistic setting they are powerful models and are able of capturing the modeling capacities of hidden Markov models (HMMs) (Rabiner, 1990; Vidal et al., 2005b) or pair hidden Markov models (PHMMs) (Durbin et al., 1998).

1.2. Learning

According to Merriam Webster's online dictionary[3] the word learning is defined as follows:

> *Learning:* 1) the act or experience of one that learns 2) knowledge or skill acquired by instruction or study 3) modification of a behavioral tendency by experience (as exposure to conditioning).

The first two definitions above can be interpreted as learning being merely about acquiring information and reproducing the same information. This can simply be achieved by memorization techniques or rote learning. For example, a child is taught five sentences in a language and the child is only able to recall exactly these five instances, *i.e.*, the child is only able to reproduce the information it has been given. The third definition goes somewhat beyond mere rote learning and talks about modification of behavior. The third definition can be interpreted as generalization from the observed instances over unseen instances. For example, after been taught five sentences in a language, the child is able to formulate new sentences in that language, *i.e.*, the child is able to generalize.

Learning in the context of the thesis will imply generalization ability of the learner. By the term learner we will imply the learning algorithm that is undergoing the learning process and the objective is to generalize in a meaningful manner.

1.3. Learning Environments

The fields of machine learning, computational learning theory, and algorithmic learning theory aim to formulate mathematical models for learning by means of computation. Often real world scenarios are modeled formally, *e.g.*, a language learning scenario of a child. In the process of formalizing the scenario mathematically, certain assumptions, simplification, and idealization have to be incorporated in the model. This is required due to the fact that the real world learning environment and learning capabilities of human are far too complex to be taken into consideration in a mathematical model. Therefore, in this section we talk about different environments or settings of learning. We will discuss two learning models that are relevant to our work.

[3] http://www.merriam-webster.com/dictionary

1.3.1. Identification in the Limit

Identification in the limit model was introduced by Gold in his seminal paper (1967). In his learning model, Gold formulated learning as an infinite process. The learner will be given examples of the language, and from the given example the learner will deduce a hypothesis H. As the learner is given more examples, the learner is allowed to modify his hypothesis H during the learning process. This process continues forever. If after observing a finite number of examples, the learner does not require to modify his hypothesis anymore, the learner is said to have learnt the language.

It has been argued that this learning model is similar to how a human learns languages (Clark and Lappin, 2011). A child receives instances of a language from his childhood, learns a part of the language and incrementally his hypothesis model of the language is modified. The process of receiving instances of the language continues even during the rest of his lifetime.

Gold defined the term *learnable* as the following: a class of language is learnable *w.r.t.* a formal device representing that language, if there exists a learning algorithm that has the following property - from the presentation of any languages of the class, after observing a finite number of examples the learner will deduce a hypothesis of the formal device that will recognize every example of that language correctly. When a new example is observed by the algorithm, it updates the hypothesis if required. As this process of receiving examples continues forever, and the hypothesis makes no mistake, the algorithm is said to have *converged*.

Modifications and extensions have been done to adapt Gold's definition of identification in the limit model to specific learning tasks. We will present formal definitions of such extensions in the forthcoming chapters of the thesis. Details about identification in the limit can be found in (de la Higuera, 2010; Jain et al., 1999; Clark and Lappin, 2011).

1.3.2. Active Learning

Active learning (*a.k.a.* query learning) is a subfield of machine learning where the learner can interact with the learning environment. In the learning process, the learner has got access to an expert or a teacher, often termed as the *oracle* or the *minimally adequate teacher* (MAT). It is advocated by the active learning proponents that instead of learning from a large pool of labeled examples, if the learner can have labeled examples on demand then the learner will learn with fewer labeled examples. Thus, the labeling cost, *i.e.*, the cost of designating the correct class labels to the training data, can be minimized.

The first active learning scenario was presented by Angluin in (Angluin, 1987b) where she presented an algorithm to learn a deterministic finite state automaton using membership queries and equivalence queries. Membership queries are the queries to an oracle *w.r.t.* an example sentence or string where the oracle answers if the example belongs to a language or not. For equivalence queries, the oracle will answer if the hypothesis is equivalent to the actual learning target or not. If the hypothesis is not equivalent to the target machine, the oracle returns a counter example. After Angluin's algorithm, a number of different kinds of queries have been used in different learning tasks. In Chapter 3 we will discuss in details about different types of queries.

The paradigm of query learning was introduced to obtain negative learnability results. Angluin (1987b) showed that it is not possible to learn a deterministic finite automaton (DFA) from membership queries only. This also meant that learning DFAs from given data is also not possible because if it is not possible to learn when the learner can choose the training data by queries, logically it is also not possible when the training data is given randomly to the learner. Moreover, since equivalence queries are not practicable, the query learning paradigm was restricted as theoretical results than practical usage.

However, today, query learning is one of the most successful applications of grammatical inference. It is widely applied in model checking (Bréhélin et al., 2001; Raffelt and Steffen, 2006), software engineering (Carme et al., 2007), robotics (Rivest and Schapire, 1993; Dean et al., 1995) *etc.* In practice, the oracle can be a human, the web or a corpus and queries can be simulated. One of the main objectives of query learning is to minimize the number of queries or the labeling costs. Recent surveys on active learning can be found in (Settles, 2009, 2011).

1.4. The Problem of Transducer Learning

The problem of learning transducers from examples can be viewed as a generalization of the problem of learning finite state automata. In case of learning a deterministic finite automaton (DFA), the following decision problem is $\mathcal{NP}-$complete: *Is there a* DFA *consistent to the given examples?* The problem of finding minimal consistent DFA from given examples has been proven to be $\mathcal{NP}-$hard (Pitt and Warmuth, 1993). As a generalization of the learning problem of finite state automata from examples, the problem of learning transducers from examples is also nontrivial.

However, in certain restrictive forms, such as subsequential or deterministic, number of solutions for learning transducer has been presented where the training

data must contain a minimum number of examples with some predefined criteria, *e.g.*, Oncina *et al.* presented an algorithm for learning subsequential transducers from positive sample in an identification in the limit model (Oncina et al., 1993; Oncina and García, 1991), Vilar showed learning of subsequential transducers in active learning setting (Vilar, 1996), and several others include (Wakatsuki and Tomita, 2010; Clark, 2011; Oncina, 2008). Some of these algorithms and other related work will be discussed in details in Chapter 3.

The problem of learning a probabilistic transducer from positive examples remains an open problem. This thesis aims to investigate the problem of learning probabilistic subsequential transducers.

1.5. Contributions of the Thesis

The primary focus of the thesis is the problem of learning probabilistic subsequential transducers. To investigate the focal problem, we develop five research questions (**RQ**s) that are essentially either decompositions of the broader problem statement or closely related to the actual problem statement. In this section we will discuss the **RQ**s.

Research Question 1 *How powerful are different types of probabilistic transducers in terms of expressive power?*

Formal languages modeled by different types of syntactic machines can be categorized hierarchically in terms of the expressive power of the respective machine type. This is known as the famous Chomsky's Hierarchy (Chomsky, 1956). Similar to Chomsky's hierarchy, it is also important to have an hierarchical view of stochastic bi-languages or transduction schemes expressed by different types of probabilistic transducers. By **RQ**1, we address this issue in order to show the expressive power and limitations of our model.

Within the scope of **RQ**1 we show three hierarchical views: the hierarchy of bi-languages, hierarchy of stochastic bi-languages and the hierarchy syntactic machines that generates stochastic bi-languages.

Research Question 2 *What algorithms can we use to solve typical finite state machine tasks over different types of probabilistic transducers?*

Typical tasks using a probabilistic transducer may involve computing the most probable translation of a given string, the joint probability of a translation pairs and other similar tasks. Computational complexity to compute translations by different types of probabilistic transducers differ, and in some cases there exists

no polynomial solution. If it is too difficult to parse using a type of probabilistic transducer, learning that type of model is poorly justified. Therefore, it is important to investigate the complexities in terms of parsing using the machines. By **RQ**2, we address the parsing issue by different types of probabilistic transducers. Here, the objectives are to present algorithms for accomplishing different tasks using different types of machines and analyze the runtime complexities.

Research Question 3 *How are our models (the syntactic machines) compared with other existing models?*

Automata theoretic approach has been adopted in diverse disciplines and even has been named differently in different domain, *e.g.*, probabilistic finite state automaton (Pfa), Hmm, Phmm, and, weighted finite state transducer (Wfst). It is important to study the similarity, differences and equivalences of the different models in order to argue the generality of our work. Within the context of this **RQ** we perform such a study.

Research Question 4 *Are probabilistic subsequential transducers learnable in the limit?*

This **RQ** addresses the learnability problem in an identification in the limit model. The main objective is to present a novel algorithm for learning probabilistic subsequential transducers in the limit. The algorithm uses positive example and queries to learn the target machine. This **RQ** also encompasses the issues of correctness, runtime complexity, and query complexity of the presented algorithm.

Research Question 5 *Is it possible to learn probabilistic subsequential transducers from a positive sample?*

Unlike **RQ**4, here the learning algorithm has no access to an oracle. The learning algorithm is given only an empirical joint distribution of translation pairs. The objectives are to investigate learning from joint distribution of positive examples only, analysis of the runtime complexity of the proposed algorithm, analysis of the algorithms in terms of the amount of training data required and experimentation with synthetic data.

Other Contributions

Besides the five **RQ**s, we also investigate the parallelization of grammatical inference algorithms. We present a novel parallel algorithm for Dfa learning to gain better runtime (see Appendix A). Moreover, we present an open source library of grammatical inference algorithms (see Appendix B).

1.6. Organization of the Manuscript

Chapter 2: In this chapter we start by defining the preliminaries and mathematical background. We present a study of different classes of transduction schemes or bi-languages represented by respective syntactic machines. We also present a Venn diagram of these classes of bi-languages to show relative expressive power of each class. We analyze the complexities *w.r.t.* parsing using the syntactic machines. Finally, we discuss equivalence of our model *w.r.t.* other existing models.

Chapter 3: Here, we discuss learnability and previous work in automata and transducer learning. We discuss different existing learning algorithms to develop the ideas that have been used in the forthcoming chapters.

Chapter 4: This chapter focuses on the identification in the limit problem of probabilistic subsequential transducers. We present a novel algorithm that learns probabilistic subsequential transducer in the limit from positive presentation and by asking probabilistic queries. We present an analysis of our algorithm in terms of correctness, runtime complexity, and query complexity. We also provide experimental evidence to show the correctness of our algorithm.

Chapter 5: In this chapter we address the problem of learning probabilistic subsequential transducers from a randomly drawn sample. We leverage the empirical distribution of the observed sample to accomplish the learning task. We show experimental results where we compare the learnability of our proposed learning algorithm to other existing ones.

Chapter 6: This is the epilogue chapter where the main results are summarized, limitations, open issues and future work are discussed.

We do not claim that the survey of the related work presented in this thesis is absolutely complete. Our objective was to cover all existing work related to transducer learning in this thesis. However, if we have unintentionally missed any work that relates directly or indirectly to transducer learning, we sincerely apologize to those authors. Our intent was to make the literature review as complete as possible.

Chapter 2

Probabilistic Finite State Transducers

2.1. Introduction

In this chapter we define the preliminaries and work on **RQ**1, **RQ**2 and **RQ**3 formulated in Chapter 1. We start by discussing the mathematical background, notations, and definitions used throughout the thesis. In order to address **RQ**1, we present hierarchical views on bi-languages modeled by finite state machines. As an attempt to attack **RQ**2, we present a study on complexities of parsing *w.r.t.* probabilistic transducers. Finally, within the context of **RQ**3, we present a discussion on equivalence of probabilistic transducers *w.r.t.* PHMMs and WFSTs.

2.2. Mathematical Background

2.2.1. Semirings

Semirings have been used in automata theory to define the *weights* of an automaton. Semirings are useful mathematical objects in automata theory, since they allow a general way for computation of the weights. In this section we define the algebraic structure semiring, elaborate various types of semirings and some of their properties. The most fundamental structures, namely, monoid and semirings will be defined in this section. The definitions and results presented in this section are primarily from (Berstel and Reutenauer, 1988; Droste, 2009; Droste and Kuich, 2009; Kuich, 1997; Kuich and Salomaa, 1986; Mohri, 2002b, 2009; Rosenfeld, 1968; Salomaa and Soittola, 1978).

A *monoid* consists of a set $\mathbb{M}$ that is closed under an associative binary opera-

tion $\circ$ on $\mathbb{M}$ and an identity element $\overline{1}$ such that: $\forall a \in \mathbb{M}, \overline{1} \circ a = a \circ \overline{1} = a$ (Kuich and Salomaa, 1986; Kuich, 1997). A monoid is called commutative if $\forall a, b \in \mathbb{M}$, $a \circ b = b \circ a$. The binary operation is usually denoted multiplicatively.

Definition 1 (Semiring) *A semiring is a set $\mathbb{K}$ having two binary operations $\oplus$ and $\otimes$ such that:*

- $\left\langle \mathbb{K}, \oplus, \overline{0} \right\rangle$ *is a commutative monoid where $\overline{0}$ is the identity element,*
- $\left\langle \mathbb{K}, \otimes, \overline{1} \right\rangle$ *is a monoid where $\overline{1}$ is the identity element,*
- $\otimes$ *distributes over $\oplus$, i.e., $\forall a, b, c \in \mathbb{K}$:*
  - $a \otimes (b \oplus c) = a \otimes b \oplus a \otimes c$
  - $(a \oplus b) \otimes c = a \otimes c \oplus b \otimes c$,
- $\forall a \in \mathbb{K}, \overline{0} \otimes a = a \otimes \overline{0} = \overline{0}$.

Intuitively, a semiring is a ring without the inverse elements for addition. Note that in the definition, $\otimes$ is not a commutative operation. If however, $\otimes$ is commutative, the semiring is said to be *commutative*. A semiring is *idempotent* if $\forall a \in \mathbb{K}$, $a \oplus a = a$.

The most common and widely used semiring in automata theory is the *boolean semiring*. The boolean semiring is denoted as $\langle \mathbb{B} = \{0, 1\}, \vee, \wedge, 0, 1 \rangle$. A weighted automaton defined under a boolean semiring is essentially a classical finite automaton or an acceptor.

Another important type of semiring is the *probability semiring* $\langle \mathbb{R}_+, +, \times, 0, 1 \rangle$ which is the set of all positive real numbers ($\mathbb{R}_+$) with two binary operations $+$ and $\times$ having their conventional meanings in arithmetic. In practice, the *log semiring* $\langle \mathbb{R} \cup \{-\infty, +\infty\}, \oplus_{log}, +, +\infty, 0 \rangle$, the isomorphic counterpart of the probability semiring through negative log transformation, is often used instead of the probability semiring. With the probability semiring, there is a practical problem that can occur easily while implementing; to compute probability, if we multiply floating point numbers a large number of times, we may encounter an *underflow* error. In order to overcome this practical problem, the log semiring is often used for achieving numerical stability. The addition $\oplus_{log}$ operation is defined as $x \oplus_{log} y = -log(e^{-x} + e^{-y})$.

Table 2.1 summarizes some useful types of semirings.

Table 2.1.: Types of useful semirings.

Semiring type	set	$\oplus$	$\otimes$	$\overline{0}$	$\overline{1}$	purpose
Boolean	$\mathbb{B} = \{1, 0\}$	$\vee$	$\wedge$	0	1	recognition
Probability	$\mathbb{R}_+$	$+$	$\times$	0	1	probability
Log	$\mathbb{R} \cup \{-\infty, +\infty\}$	$\oplus_{log}$	$+$	$+\infty$	0	log probability
Tropical	$\mathbb{R} \cup \{-\infty, +\infty\}$	min	$+$	$+\infty$	0	distance problem

2.2.2. Definitions and Notations

Let $[n]$ denote the set $\{1, \ldots, n\}$ for each $n \in \mathbb{N}$. An *alphabet* Σ is a non-empty set of symbols and the symbols are called *letters*. Σ^* is a *free-monoid* over Σ. Subsets of Σ^* are known as (*formal*) *languages* over Σ. A *string* w over Σ is a finite sequence $w = a_1 \ldots a_n$ of letters. Let $|w|$ denote the length of the string w. In this case we have $|w| = |a_1 \ldots a_n| = n$. The empty string is denoted by ϵ. For every $w_1, w_2 \in \Sigma^*$, $w_1 \cdot w_2$ is the concatenation of w_1 and w_2. The concatenation of ϵ and a string w is given by $\epsilon \cdot w = w$ and $w \cdot \epsilon = w$. When decomposing a string into substrings, we will write $w = w_1, \ldots, w_n$ where $\forall i \in [n], w_i \in \Sigma^*$. If $w = w_1 w_2$ is a string, then w_1 is a *prefix* and w_2 is a *suffix* of the string w. Given a language $\mathcal{L} \subseteq \Sigma^*$, the *prefix set* of $\mathcal{L}$ is defined as:

$$\mathit{Pref}(\mathcal{L}) = \{u \in \Sigma^* : \exists v \in \Sigma^*, uv \in \mathcal{L}\}$$

and the *suffix set* of $\mathcal{L}$ is defined as:

$$\mathit{Suff}(\mathcal{L}) = \{v \in \Sigma^* : \exists u \in \Sigma^*, uv \in \mathcal{L}\}.$$

$\mathit{Pref}(w)$ and $\mathit{Suff}(w)$ are defined as the sets of all the substrings of w that are prefixes and suffixes of w correspondingly. The *longest common prefix* of $\mathcal{L}$ is denoted as $lcp(\mathcal{L})$, where:

$$lcp(\mathcal{L}) = w \iff w \in \bigcap_{x \in \mathcal{L}} \mathit{Pref}(x) \wedge \forall w' \in \bigcap_{x \in \mathcal{L}} \mathit{Pref}(x) \Rightarrow |w'| \leq |w|.$$

Less formally, *lcp* is a function that returns the longest possible string which is the prefix of all the strings in a given set of strings. For example, for $\mathcal{L} = \{aabb, aab, aababa, aaa\}$, the $lcp(\mathcal{L})$ is aa.

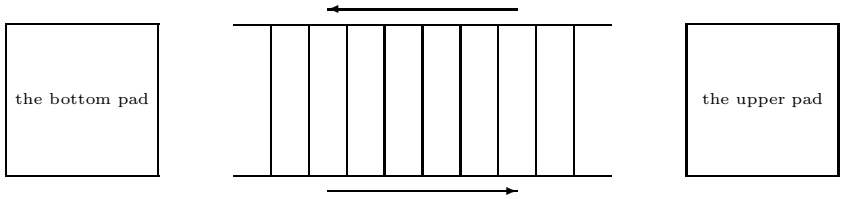


Figure 2.1.: Top view of a bi-directional elevator.

2.2.3. Finite State Automata

A *finite state automaton* is a basic and severely restricted mathematical model for computation. The word "automaton" is derived from the Greek word "automatos". According to Merriam Webster's online dictionary[1] the word automaton is defined as follows:

> *Automaton:* a machine or control mechanism designed to follow automatically a predetermined sequence of operations or respond to encoded instructions.

To elaborate the above definition, we will take a look at a simplified real life example. The controller used for a bi-directional elevator, designed for energy saving, is an example of such an automaton. Figure 2.1 shows the top view of a bi-directional elevator.

Figure 2.2 shows the state diagram of the controller. There can be three states of the elevator at any given time: stopped, moving up, and moving down. The controller is designed to execute some predetermined actions whenever it encounters any of the following events: 1) the bottom pad is occupied (denoted as `B`), 2) the upper pad is occupied (denoted as `U`), 3) the elevator is occupied (denoted as `E`), and 4) if none of the pads or the elevator is occupied (denoted as `NONE`). The action performed by the controller is to switch from one state to another based on the input.

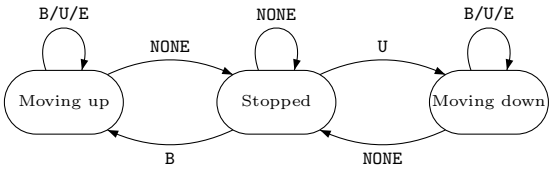


Figure 2.2.: State diagram for an automatic controller of a bi-directional elevator.

[1] http://www.merriam-webster.com/dictionary

The controller receives input and based on the input, it either stays at the current state or moves to a different state. When it is at the *stopped* state, it goes to either *moving up* or *moving down* state depending on the input. When it is at the *moving up* state or at the *moving down* state, the controller stays at the same state unless it receives the input `NONE`.

The state diagram shown in Figure 2.2 is the automaton model of the scenario described. In the following we will provide formal definitions of automata.

Definition 2 (Non-deterministic Finite State Automaton) *A non-deterministic finite state automaton (*NFA*) over the boolean semiring* $\mathbb{B}$ *is a sextuple* $A = \langle Q, I, F, R, \Sigma, E \rangle$ *where:*

- Q *is a finite set of states,*
- Σ *is a finite set of symbols called the alphabet,*
- $I \subseteq Q$ *is the set of initial states,*
- F *is the partial function defined as* $F : Q \to \mathbb{B}$ *for the accept states or the final states,*
- R *is the partial function defined as* $R : Q \to \mathbb{B}$ *for the reject states,*
- E *is a finite set of transitions defined as* $E \subseteq Q \times \Sigma \cup \{\epsilon\} \times \mathbb{B} \times Q$.

For a state $q \in Q$, if $F(q) = 1$, the state q is said to be a final state or an accept state. Similarly, if $R(q) = 1$ the state is said to be a reject state. However, a state q cannot be a final and a reject state at the same time, *i.e.*, $\forall q \in Q, F(q) = \neg R(q)$. A transition $e \in E$ only exists, if the weight of e is 1. Which means all the edges in an NFA will have a weight of 1. Essentially, automata that are defined under boolean semiring are the unweighted automata. For simplicity we will omit the weights.

For any transition $e \in E$, let $prev\,[e] \in Q$ be the origin state of the transition, $next\,[e] \in Q$ be the destination state, and $i\,[e] \in \Sigma \cup \{\epsilon\}$ is the input symbol associated with e. For any state $q \in Q$, $E\,[q]$ is the set of all the transitions going out of the state q. A *path* $\pi = e_1, e_2, e_3, \ldots, e_k$ is a sequence of k consecutive transitions such that $next\,[e_{i-1}] = prev\,[e_i]$, where $i = 2, \ldots, k$. In terms of the definition of paths, $next\,[\pi] = next\,[e_k]$ and $prev\,[\pi] = prev\,[e_1]$. We define $i\,[\pi] = x \in \Sigma^*$ such that $x = i\,[e_1] \cdot i\,[e_2] \cdot \ldots \cdot i\,[e_k]$. We denote Π_A as the set of all the paths in A. We define $\pi_\epsilon \in \Pi_A$ as the empty path, *i.e.*, a path that consists of no transitions.

Definition 3 *Let $x = a_1 \dots a_n$ be an arbitrary string where $a_i \in \Sigma \cup \{\epsilon\}$. A language $\mathcal{L}$ is recognized by an* NFA *A, if $\forall x \in \mathcal{L}$:*

- $\exists \pi = e_1, \dots, e_n \in \Pi_A : prev\,[\pi] \in I, F(next\,[\pi]) = 1$,
- $\forall e_i \in \pi, i\,[e_i] = a_i$.

Definition 4 (Deterministic Finite State Automaton) *A deterministic finite state automaton (*DFA*) over the boolean semiring $\mathbb{B}$ is a sextuple $A = \langle Q, I, F, R, \Sigma, E \rangle$ where:*

- *Q is a finite set of states,*
- *$I = \{q_0\}$ is the unique initial state,*
- *Σ is the alphabet,*
- *F is the partial function defined as $F : Q \to \mathbb{B}$ for the accept states or the final states,*
- *R is the partial function defined as $R : Q \to \mathbb{B}$ for the reject states,*
- *E is a finite set of transitions defined as $E \subseteq Q \times \Sigma \times \{1\} \times Q$ and subject to the following condition:*

$$\forall (q, a, 1, q'), (q, a, 1, q'') \in E \Rightarrow q' = q''.$$

It is well known that an NFA can be determinized to an equivalent DFA. Determinization algorithm for NFA to DFA can be found in any standard textbook of automata theory, *e.g.*, see (Lewis and Papadimitriou, 1997; Sipser, 1996).

Languages that are recognized by DFAs are known as *regular languages*. We denote the set of all regular languages as $\boldsymbol{\mathcal{RL}(\Sigma)}$.

2.3. Bi-languages

In the previous section, we have discussed languages that consist of a set of strings. There are cases where we need to work with a pair of strings or a *bi-string* (Kornai, 2007). For example, machine translation tasks often require a model having input strings and output strings. In this section we present the definition of a *bi-language*, which consists of a set of pairs of strings. The definitions presented in this section are inspired by a number of work including (Berger and Pair, 1978; Kornai, 1995, 2007).

At this point we will use two alphabets: Σ to denote the set of input symbols and Ω to denote the set of output symbols. Σ and Ω are not necessarily identical.

Definition 5 *A bi-language $\mathcal{BL}$ is a relation defined as $\mathcal{BL} \subseteq \Sigma^* \times \Omega^*$.*

The syntactic machine that recognizes a bi-language is known as a *bi-automaton* or a *transducer*. In the next section we define such machines.

2.4. Transducers

In this section we provide definitions and examples of different types of transducers. The definitions presented in this section are primarily from (Berstel and Reutenauer, 1988; Castellanos et al., 1998; de la Higuera, 2010; Droste, 2009; Kuich and Salomaa, 1986; Kuich, 1997; Oncina et al., 1993; Reutenauer and Schützenberger, 1991, 1995; Salomaa and Soittola, 1978).

2.4.1. Rational Transducers

Definition 6 *A **rational transducer** is a quintuple $T = \langle Q, \Sigma, \Omega, I, E \rangle$ over the boolean semiring $\mathbb{B}$ where:*

- *Q is a finite set of states,*
- *Σ is the input alphabet,*
- *Ω is the output alphabet,*
- *$I \subseteq Q$ is the set of initial states,*
- *$E \subseteq Q \times \Sigma^* \times \Omega^* \times \mathbb{B} \times Q$ is a finite set of transitions.*

Since the input label of a rational transducer is a string $x \in \Sigma^*$, the input string associated with e is $i[e] \in \Sigma^*$. The output label associated with an edge e is denoted as $o[e] \in \Omega^*$. We define $o[\pi] = \mathbf{y} \in \Omega^*$ such that $\mathbf{y} = o[e_1] \cdot o[e_2] \cdot \ldots \cdot o[e_k]$. Similar to the definition of an NFA, an edge $e \in E$ only exists if the weight of the edge is 1. For simplicity we will omit the boolean weights in the illustrations.

Definition 7 *Let $x = x_1 \cdot x_2 \cdot \ldots \cdot x_n$ be an arbitrary string where $x_i \in \Sigma^*$ and $\mathbf{y} = \mathbf{y}_1 \cdot \mathbf{y}_2 \cdot \ldots \cdot \mathbf{y}_n$ be an arbitrary string where $\mathbf{y}_i \in \Omega^*$. A bi-language $\mathcal{BL} = \{(x, \mathbf{y}) : x \in \Sigma^*, \mathbf{y} \in \Omega^*\}$ is recognized by a rational transducer T, if $\forall (x, \mathbf{y}) \in \mathcal{BL}$:*

- *$\exists \pi = e_1, \ldots, e_n \in \Pi_T : prev[\pi] \in I$,*

- $i\,[e_i] = x_i$ *and* $o\,[e_i] = \mathbf{y}_i$.

Note that in Definition 7 x_i and $\mathbf{y}_i$ are both substrings of arbitrary length. Therefore, although x and $\mathbf{y}$ are both decomposed to n substrings, the actual length of x and $\mathbf{y}$ can differ.

For an input-output pair $(x, \mathbf{y}), x \in \Sigma^*$ and $\mathbf{y} \in \Omega^*$, the set of paths in a transducer T is given by:

$$\Pi_T(x, \mathbf{y}) = \{\pi \in \Pi_T : i\,[\pi] = x \wedge o\,[\pi] = \mathbf{y}\}\,.$$

We define the set of valid paths in a *rational transducer* as:

$$\Pi_T(I) = \{\pi \in \Pi_T : prev\,[\pi] \in I\}\,.$$

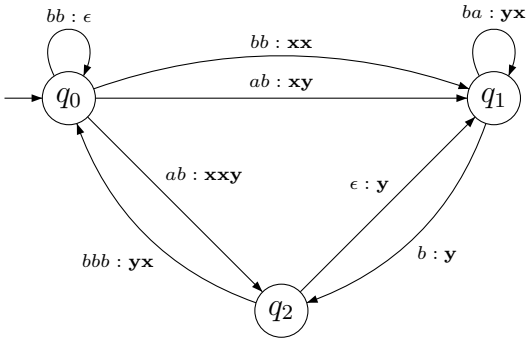


Figure 2.3.: An example of a *rational transducer.*

Figure 2.3 shows an example of a rational transducer. Basically, it is a string to string transducer without any final state. It is possible to halt at any state of the machine.

A bi-language which is recognized by a rational transducer is known as a *rational bi-language*. We will denote the set of all the rational bi-languages as $\boldsymbol{\mathcal{RBL}(\Sigma, \Omega)}$.

Notice that in Figure 2.3 the transitions (q_0, bb, ϵ, q_0) and $(q_0, bb, \mathbf{xx}, q_1)$ are one of the causes for non-determinism. Such cases are known as *ambiguity* of rational transducers. There are three ways ambiguity can be defined *w.r.t.* rational transducers:

Ambiguity of input

A rational transducer T admits *ambiguity of input* when the following property holds:

$$\exists \pi_1, \pi_2 \in \Pi_T(I) \colon i\,[\pi_1] \neq i\,[\pi_2] \wedge o\,[\pi_1] = o\,[\pi_2]\,.$$

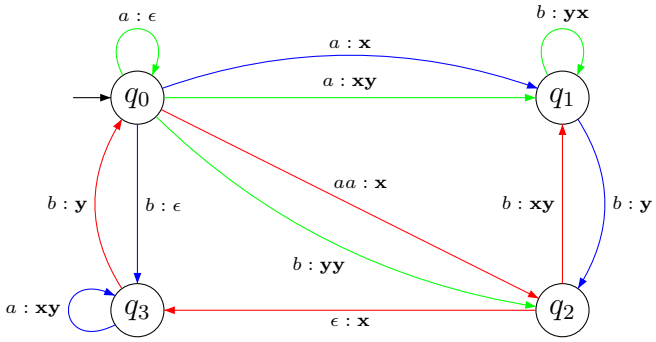


Figure 2.4.: An example of a *rational transducer* with ambiguities *w.r.t.* input, output, and path. Blue, green, and red edges show the causes of ambiguities of input, output, and path respectively.

In Figure 2.4 the blue edges exemplify the input ambiguity. The input-output pairs $(ab, \mathbf{xy})$ and $(ba, \mathbf{xy})$ have identical output strings but different input strings, hence, cause input ambiguity.

Ambiguity of output

A rational transducer T admits *ambiguity of output* when the following property holds:

$$\exists \pi_1, \pi_2 \in \Pi_T(I) \colon i[\pi_1] = i[\pi_2] \wedge o[\pi_1] \neq o[\pi_2].$$

In Figure 2.4 the green edges are shown to give an example of the output ambiguity. The input-output pairs $(ab, \mathbf{xyx})$ and $(ab, \mathbf{yy})$ have identical input strings and different output strings, and thus, cause output ambiguity.

Ambiguity of path

A rational transducer T admits *ambiguity of path* when the following property holds:

$$\begin{aligned} \exists \pi_1, \pi_2 \in \Pi_T(I) \quad &: i[\pi_1] = i[\pi_2] \wedge o[\pi_1] = o[\pi_2] \\ &\wedge \pi_1 \neq \pi_2. \end{aligned}$$

In Figure 2.4 the red edges are shown to exemplify the ambiguity of path. The input-output pair (aab,$\mathbf{xxy}$) can have two different paths and consequently causes ambiguity of path.

When we want to map from the input language to the output language, the input ambiguity does not cause any ambiguity in terms of parsing using the

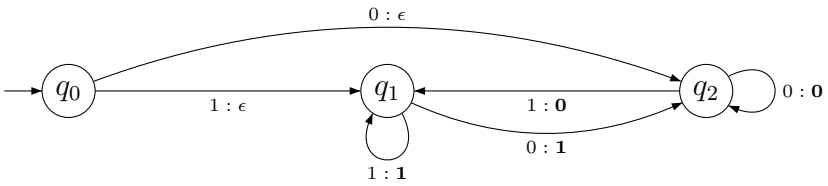


Figure 2.5.: An example of a *sequential transducer.*

syntactic machine, and hence, computation of translation *w.r.t.* the input string is not ambiguous. Since in a transduction scheme, we are interested in computing translation from the input language to the output language, we will not consider the case of input ambiguity. We will call a rational transducer *un-ambiguous* if the ambiguity of output and the ambiguity of path do not hold and *ambiguous* otherwise. Rational transducers are in general non-deterministic and hence why, ambiguous. Next, we define some deterministic versions of rational transducers.

2.4.2. Sequential Transducers

Definition 8 (Sequential Transducer) *A sequential transducer is a quintuple* $T = \langle Q, \Sigma, \Omega, I, E \rangle$*, where* T *is a rational transducer such that:*

- $E \subseteq Q \times \Sigma \times \Omega^* \times \mathbb{B} \times Q$*,*
- $I = \{q_0\}$ *where* $q_0 \in Q$*,*
- $\forall q \in Q, \forall e, e' \in E\,[q]\,, i\,[e] = i\,[e'] \Rightarrow o\,[e] = o\,[e'] \wedge next\,[e] = next\,[e']\,.$

Notice that due to the determinism condition, a bi-language $\mathcal{BL}$ recognized by a sequential transducer is functional. Therefore, in the context of sequential transducers we will define a bi-language $\mathcal{BL}$ recognized by a sequential transducer T as the function $t_T : \Sigma^* \to \Omega^*$. For an input-output pair or a bi-string $(x, \mathbf{y}) \in \mathcal{BL}$ we will write $t_T(x) = \mathbf{y}$.

As an example, let us consider a toy task of transduction t_T, where $\Sigma = \{0, 1\}$ and $\Omega = \{\mathbf{0}, \mathbf{1}\}$. By means of the transduction t_T we want to compute arithmetic shift right operation over a binary input, *i.e.*, we want to perform division by 2 in base 2, *e.g.*, $t_T(10010) = \mathbf{1001}$. The sequential transducer shown in Figure 2.5 is the machine that represents the transduction t_T.

We will call a bi-language that is recognized by a sequential transducer: a *sequential bi-language*. The set of all the sequential bi-languages is denoted as $\boldsymbol{\mathcal{SBL}(\Sigma, \Omega)}$.

Proposition 1 *Sequential transducers are un-ambiguous.*

Sequential transducers admit prefix preserving property. Formally, $t_T(\epsilon) = \epsilon$ and $\forall x, y \in \Sigma^*$, if $t_T(xy)$ exists, then $t_T(x) \in \mathit{Pref}(t_T(xy))$ (Oncina et al., 1993). The prefix preserving property brings certain limitations to sequential transducers (Castellanos et al., 1998; de la Higuera, 2010). Due to the prefix preserving property, not all transductions are possible using a sequential transducer. This argument can be best explained using an example of a transduction task where we want to compute a rewrite operation of arithmetic shift left, *i.e.*, we want to compute multiplication by 2 in base 2, *e.g.*, $t_T(011) = \mathbf{110}$. Notice that in this example the prefix preserving property does not hold. Moreover, there are two more interesting aspects of this transduction function. Firstly, if we read 0 as the first symbol of the input string, the output should be an empty string ϵ. Secondly, at the end of reading the whole string there has to be a suffix **0** added to the output string. Again we observe that the second property of the given transduction cannot be satisfied by a sequential transducer. In order to overcome this limitation, we will introduce a modified machine called *subsequential transducer*.

2.4.3. Subsequential Transducers

Definition 9 (Subsequential Transducer) *A subsequential transducer (*ST*) is a sextuple $T = \langle Q, \Sigma, \Omega, I, E, \sigma \rangle$, where $\psi(T) = \langle Q, \Sigma, \Omega, I, E \rangle$ is a sequential transducer and $\sigma : Q \to \Omega^*$ is a partial function that assigns output strings to states $q \in Q$.*

The bi-language recognized by an ST T is defined as $t_T : \Sigma^* \to \Omega^*$ such that, $\forall x \in \Sigma^*, t_T(x) = t_{\psi(T)}(x) \cdot \sigma(q)$, where $t_{\psi(T)}$ is the sequential transduction provided by $\psi(T)$, q is the last state reached with the input string x and $\sigma(q)$ is the state output function. Figure 2.6 shows an example of a subsequential transducer. In this example the subsequential transducer computes the rewrite operation of arithmetic shift left in base 2, *e.g.*, $t_T(1010) = \mathbf{10100}$. Note that the bi-language represented by the ST in Figure 2.6 cannot be modeled by a sequential transducer without any additional assumptions.

Proposition 2 *Subsequential transducers are un-ambiguous.*

Bi-languages modeled by subsequential transducers can also be modeled by sequential transducers by means of an extra stop symbol $\sharp \notin \Sigma$. The state outputs can be replaced by special transitions from that state with input symbol $\sharp$ having the transition outputs same as the state outputs. The destination of

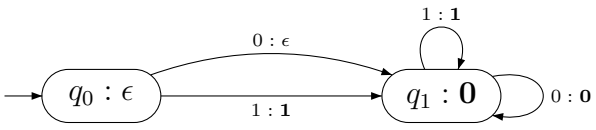


Figure 2.6.: An example of a *subsequential transducer*.

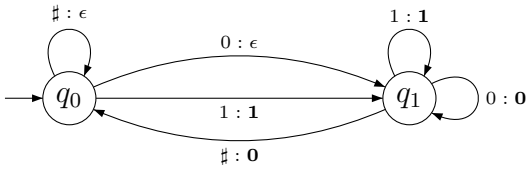


Figure 2.7.: A sequential representation of the arithmetic shift left transduction task by using the stop symbol ♯.

such transitions are for technical convenience always the initial state. Figure 2.7 shows a sequential representation of the subsequential transducer in Figure 2.6. For technical convenience we will use ♯ edges instead of the state output function.

We will call a bi-language that is recognized by a subsequential transducer: a *regular bi-language*. The set of all the regular bi-languages is denoted as $\boldsymbol{\mathcal{REGBL}(\Sigma, \Omega)}$.

Property 1 $\forall \mathcal{BL} \in \boldsymbol{\mathcal{REGBL}(\Sigma, \Omega)}$*, for a given input string* $x \in \Sigma^*$*, there is either* 0 *or* 1 *translation for* x*.*

Proof The proof is a direct consequence of Proposition 2. □

Some more auxiliary properties of sequential and subsequential transducers are discussed in Appendix C.

2.4.4. p-Subsequential Transducers

In this section, we will introduce p-subsequential transducers, transducers which are deterministic *w.r.t.* input symbols and non-deterministic *w.r.t.* state outputs. p-Subsequential transducers were introduced by Allauzen and Mohri in (Allauzen and Mohri, 2002, 2003a; Mohri, 2000b). They showed that rational transducers (non-deterministic) having the so-called twins property can be determinized to

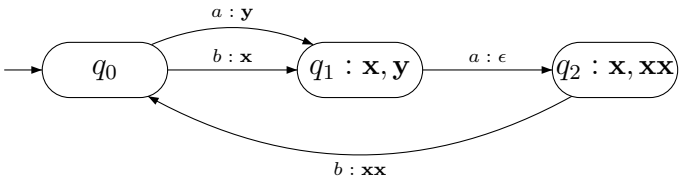


Figure 2.8.: An example of a 2-*subsequential transducer.*

p-subsequential transducers, and hence, p-subsequential transducers have more expressive power than subsequential transducers.

Definition 10 (p-Subsequential Transducer) *A p-subsequential transducer (p-ST) is a sextuple $T = \langle Q, \Sigma, \Omega, I, E, \sigma \rangle$, where $\psi(T) = \langle Q, \Sigma, \Omega, I, E \rangle$ is a sequential transducer and $\sigma : Q \rightarrow (\Omega^*)^p$ is a partial function that assigns exactly p number of output strings to states $q \in Q$.*

Figure 2.8 illustrates an example of 2−subsequential transducer. Here, every state output function (partial) outputs 2 output strings. In case of $p = 1$, a p-ST is essentially an ST.

We will call a bi-language that is recognized by a p-subsequential transducer: a *p-regular bi-language*. The set of all the p-regular bi-languages is denoted as $\boldsymbol{\mathcal{PREGBL}(\Sigma, \Omega)}$.

Property 2 *Let T be a p-subsequential transducer and $x \in \Sigma^*$ such that $x = x_1 \dots x_{|x|}$. T produces p number of different outputs for x denoted as $\mathbf{y}_1, \dots, \mathbf{y}_p$ where $\mathbf{y}_i \in \Omega^*$. Then the following statement is true:*

$$o\left[e_1\right] \cdot \ldots \cdot o\left[e_{|x|}\right] \in \mathit{Pref}(\mathit{lcp}(\mathbf{y}_1, \dots, \mathbf{y}_p)) \text{ where } e_i \in E \text{ and } i\left[e_i\right] = x_i.$$

Proof The proof of the property can be constructed as the following:

$$\mathbf{y}_1 = o\left[e_1\right] \cdot \ldots \cdot o\left[e_{|x|}\right] \cdot \sigma(q) \text{ where } \mathit{next}\left[e_{|x|}\right] = q$$

$$\dots$$

$$\mathbf{y}_p = o\left[e_1\right] \cdot \ldots \cdot o\left[e_{|x|}\right] \cdot \sigma(q) \text{ where } \mathit{next}\left[e_{|x|}\right] = q$$

It is easy to see that $o\left[e_1\right] \cdot \ldots \cdot o\left[e_{|x|}\right]$ is always a prefix of the outputs, and therefore the property holds. □

Property 3 *$\forall \mathcal{BL} \in \boldsymbol{\mathcal{PREGBL}(\Sigma, \Omega)}$, for a given input string $x \in \Sigma^*$, there is either 0 or p translations for x.*

Proof The proof follows from the definition of p-subsequential transducers (Definition 10), the syntactic machine that recognizes $\mathcal{BL}$. □

2.4.5. Semi Deterministic Transducers

In this section we will define a new object called semi deterministic transducer (SDT), which is a machine whose expressive power and complexity is in between rational transducers and subsequential transducers. SDTs have some deterministic properties, yet not totally deterministic as subsequential transducers. The formal definition of an SDT is the following:

Definition 11 (Semi Deterministic Transducer) *A semi deterministic transducer (*SDT*) is a sextuple* $T = \langle Q, \Sigma, \Omega, I, E, \rho \rangle$ *where* $\psi(T) = \langle Q, \Sigma, \Omega, I, E \rangle$ *is a rational transducer such that:*

- $I = \{q_0\}$,
- $E \subseteq Q \times \Sigma \times \Omega^* \times \mathbb{B} \times Q$,
- $\forall q \in Q, \forall e, e' \in E[q], i[e] = i[e'] \Rightarrow next[e] = next[e']$,
- *a finite set of state outputs* $\rho \subseteq Q \times \Omega^*$,
- *and subject to the following condition:* $\forall q \in Q, \forall e', e \in E[q]$, *if* $i[e] = i[e'] \Rightarrow o[e] \notin Pref(o[e']) \wedge o[e'] \notin Pref(o[e])$.

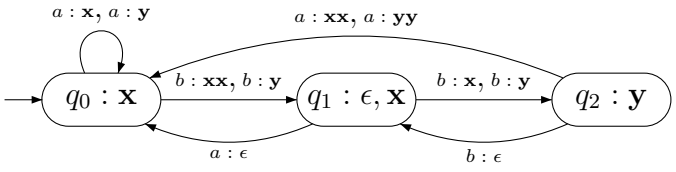


Figure 2.9.: An example of an SDT.

Figure 2.9 shows an example of an SDT. Notice that the state to state transitions are deterministic *w.r.t.* input symbols. However, there are different output strings possible for transitions with the same input string. Moreover, there can be multiple state outputs for a given state.

The bi-languages that are recognized by SDTs are called: *semi regular bi-languages*. The set of all the semi regular bi-languages is denoted as $\boldsymbol{\mathcal{SREGBL}(\Sigma, \Omega)}$.

Property 4 $\forall \mathcal{BL} \in \boldsymbol{\mathcal{SREGBL}(\Sigma, \Omega)}$, *for a given input string* $x \in \Sigma^*$, *the set* $\{\mathbf{y} : (x, \mathbf{y}) \in \mathcal{BL}\}$ *is finite.*

Proof We will show the proof by contradiction. Let us assume to the contrary that the set $\{\mathbf{y} : (x, \mathbf{y}) \in \mathcal{BL}\}$ is infinite and let T be an SDT that recognizes $\mathcal{BL}$. First, that means, we will have an infinite number of paths $\pi \in \Pi_T(I)$ such that $i[\pi] = x$. Since, SDTs by definition have no ϵ transition *w.r.t.* input, we must have an infinite number of edges $e \in E$ in order to have an infinite number of paths with the same input string. This leads to a contradiction since E is by definition a finite set. Second, for an infinite number of translations for a given string x with finite number of paths, we must have infinite state outputs. This leads to another contradiction since the set of state outputs is by definition finite. Therefore, the property holds. □

2.5. Hierarchy of Bi-Languages

In this section we examine the relative expressive power of different types of transducers. In the previous section we have categorized the different sets of bi-languages that can be recognized by different types of syntactic machines. Here, our objective is to have a hierarchical view of sets of bi-languages that can be modeled by finite state machines, which is precisely the research question formulated in **RQ**1 in Chapter 1.

Lemma 1 $\forall \mathcal{BL} \in \boldsymbol{\mathcal{SREGBL}(\Sigma, \Omega)}, \mathcal{BL} \in \boldsymbol{\mathcal{RBL}(\Sigma, \Omega)}$.

Proof The proof trivially follows from the definitions of the type of transducers that recognizes the bi-languages in the two sets. SDTs are by definition rational transducers having some deterministic conditions and state outputs. Therefore, the lemma holds. □

Lemma 2 $\exists \mathcal{BL} \in \boldsymbol{\mathcal{RBL}(\Sigma, \Omega)}$ *such that* $\mathcal{BL} \notin \boldsymbol{\mathcal{SREGBL}(\Sigma, \Omega)}$.

Proof We will prove the lemma by providing a counter example. Figure 2.10(a) precisely shows a counter example where the illustrated transducer is a rational transducer T and there exists no equivalent SDT that represents the bi-language modeled by T. Figure 2.10(a) is a counter example because it violates Property Definition 11. Moreover, the bi-languages represented by the counter example can have infinite number of translations for a given input string, *e.g.*, $(a, \mathbf{y}), (a, \mathbf{yx}), (a, \mathbf{yxx}) \ldots$. Therefore, it violates the Property 4 of a $\boldsymbol{\mathcal{SREGBL}(\Sigma, \Omega)}$. □

Lemma 3 $\forall \mathcal{BL} \in \boldsymbol{\mathcal{PREGBL}(\Sigma, \Omega)}, \mathcal{BL} \in \boldsymbol{\mathcal{SREGBL}(\Sigma, \Omega)}$.

Proof We will show the proof by construction. Given an arbitrary p-ST $T = \langle Q, \Sigma, \Omega, I, E, \rho \rangle$ that recognizes a p-regular bi-language, we will construct an equivalent SDT $T' = \langle Q', \Sigma, \Omega, I', E', \rho' \rangle$ that recognizes a semi regular bi-language. The construction is the following:

- $Q' \leftarrow Q$;
- $I' \leftarrow I$;
- $E' \leftarrow E$;
- $\rho' \leftarrow \{(q, \mathbf{y}) : \rho(q) = \mathbf{y}\}$.

Trivially T' is a SDT that recognizes the same bi-language. Therefore, the lemma holds. □

Lemma 4 $\exists \mathcal{BL} \in \boldsymbol{\mathcal{SREGBL}(\Sigma, \Omega)}$ *such that* $\mathcal{BL} \notin \boldsymbol{\mathcal{PREGBL}(\Sigma, \Omega)}$.

Proof We will show the proof by providing a counter example of syntactic machine, namely an SDT, representing a semi regular bi-language for which there exists no equivalent p-ST that represents a p-regular bi-language. The SDT illustrated in Figure 2.10(b) is precisely the counter example. Figure 2.10(b) is a counter example because it violates Property 2. Moreover, in this example for for an input $a^n \in \Sigma^*, n > 1$, there are 2^n number of translations: this violates Property 3. Therefore, the lemma holds. □

Lemma 5 $\forall \mathcal{BL} \in \boldsymbol{\mathcal{REGBL}(\Sigma, \Omega)}, \mathcal{BL} \in \boldsymbol{\mathcal{PREGBL}(\Sigma, \Omega)}$.

Proof The proof is trivial since STs are essentially p-STs with $p = 1$. □

Lemma 6 $\exists \mathcal{BL} \in \boldsymbol{\mathcal{PREGBL}(\Sigma, \Omega)}$ *such that* $\mathcal{BL} \notin \boldsymbol{\mathcal{REGBL}(\Sigma, \Omega)}$.

Proof Here, we show a counter example of a p-ST that recognizes a p-regular bi-language where there is no equivalent ST. Figure 2.10(c) depicts such a p-ST which violates the unambiguous property of an ST (Proposition 2). Moreover, it violates Property 1. Therefore, the lemma holds. □

Lemma 7 $\forall \mathcal{BL} \in \boldsymbol{\mathcal{SBL}(\Sigma, \Omega)}, \mathcal{BL} \in \boldsymbol{\mathcal{REGBL}(\Sigma, \Omega)}$.

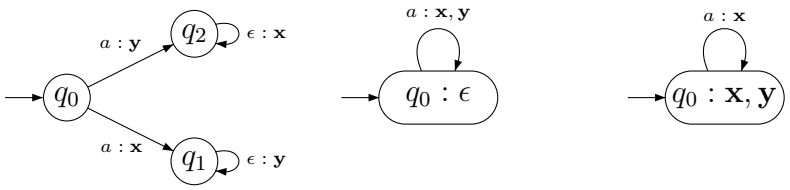
Proof We will show the proof by construction. Given an arbitrary sequential transducer $T = \langle Q, \Sigma, \Omega, I, E \rangle$, we will construct an equivalent ST $T' = \langle Q', \Sigma, \Omega, I', E', \rho \rangle$. The construction is the following:

- $Q' \leftarrow Q$;
- $I' \leftarrow I$;
- $E' \leftarrow E$;
- $\forall q' \in Q', \rho(q') \leftarrow \epsilon$.

It is easy to see that T and T' recognize the same bi-language. Therefore, the lemma holds. □

Lemma 8 $\exists \mathcal{BL} \in \boldsymbol{\mathcal{REGBL}}(\boldsymbol{\Sigma}, \boldsymbol{\Omega})$ *such that* $\mathcal{BL} \notin \boldsymbol{\mathcal{SBL}}(\boldsymbol{\Sigma}, \boldsymbol{\Omega})$.

Proof To prove the lemma, we refer back to the example of an ST given Figure 2.6 which cannot be modeled by a sequential transducer. Therefore, the lemma holds. □



(a) (b) (c)

Figure 2.10.: Counter examples for Lemma 2 (Figure 2.10(a)), Lemma 4 (Figure 2.10(b)), and Lemma 6 (Figure 2.10(c)).

Theorem 1 *Theorem of strict inclusion:*

1. $\boldsymbol{\mathcal{SREGBL}}(\boldsymbol{\Sigma}, \boldsymbol{\Omega}) \subsetneq \boldsymbol{\mathcal{RBL}}(\boldsymbol{\Sigma}, \boldsymbol{\Omega})$
2. $\boldsymbol{\mathcal{PREGBL}}(\boldsymbol{\Sigma}, \boldsymbol{\Omega}) \subsetneq \boldsymbol{\mathcal{SREGBL}}(\boldsymbol{\Sigma}, \boldsymbol{\Omega})$
3. $\boldsymbol{\mathcal{REGBL}}(\boldsymbol{\Sigma}, \boldsymbol{\Omega}) \subsetneq \boldsymbol{\mathcal{PREGBL}}(\boldsymbol{\Sigma}, \boldsymbol{\Omega})$
4. $\boldsymbol{\mathcal{SBL}}(\boldsymbol{\Sigma}, \boldsymbol{\Omega}) \subsetneq \boldsymbol{\mathcal{REGBL}}(\boldsymbol{\Sigma}, \boldsymbol{\Omega})$

Proof The proof of 1 is a direct consequence of the lemmata 1 and 2. The proof of 2 is a direct consequence of the lemmata 3 and 4. The proof of 3 is a direct consequence of the lemmata 5 and 6. Finally, the proof of 4 is a direct consequence of the lemmata 7 and 8. □

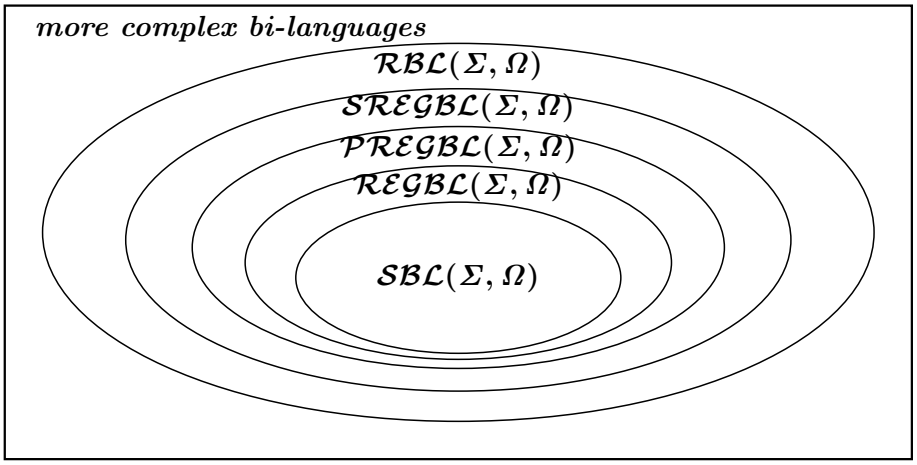


Figure 2.11.: A Venn diagram of the sets of bi-languages modeled by finite state transducers.

Figure 2.11 depicts the hierarchical view of the sets of bi-languages modeled by finite state machines. In essence, the figure is a pictorial representation of the theorem of strict inclusion (Theorem 1). As it shows that $\mathcal{RBL}(\Sigma, \Omega)$ is the most complex set of bi-languages in this hierarchy and $\mathcal{SBL}(\Sigma, \Omega)$ is the least complex one. Note that there are even more complex sets of bi-languages than $\mathcal{RBL}(\Sigma, \Omega)$ which cannot be modeled by finite state machines, *e.g.*, synchronous grammar (Aho and Ullman, 1969).

At this point, we are only half way toward answering **RQ**1, as the research question involves investigation of hierarchy of stochastic bi-languages and we have only talked about bi-languages without considering any underlying distribution. In the next sections we will elaborate bi-languages, syntactic machines that model bi-languages, and investigate their properties.

2.6. Stochastic Languages

A *stochastic language* $\mathcal{D}$ is a probability distribution over Σ^*. The probability of a string $x \in \Sigma^*$ under the distribution $\mathcal{D}$ is denoted as $Pr_{\mathcal{D}}(x)$ and must verify $\sum_{x \in \Sigma^*} Pr_{\mathcal{D}}(x) = 1$. If the distribution is modeled by some syntactic machine M, the probability of x according to the probability distribution defined by M is denoted as $Pr_M(x)$. The distribution modeled by a machine M will be denoted as $\mathcal{D}_M$ and simplified to $\mathcal{D}$ if the context is not ambiguous.

2.7. Probabilistic Finite Automata

The finite state machine that can be used to model a stochastic language is called a probabilistic finite automaton (PFA) (Paz, 1971). PFAs are generative devices. The formal definition of a PFA is the following:

Definition 12 (Probabilistic Finite Automaton) *A probabilistic finite automaton (PFA) over the probability semiring $\mathbb{R}_+$ is a quintuple $A = \langle Q, \Sigma, \lambda, F, E \rangle$ where:*

- *Q is a finite set of states,*
- *Σ is the alphabet,*
- *λ is the initial state probability,*
- *F is the final state probability,*
- *E is a finite set of transitions defined as $E \subseteq Q \times \Sigma \cup \{\epsilon\} \times \mathbb{R}_+ \times Q$,*
- *and subject to the following conditions:*

$$\sum_{q \in Q} \lambda(q) = 1,$$

$$F(q) + \forall q \in Q, \sum_{e \in E[q]} prob\,[e] = 1.$$

Figure 2.12 depicts an example of a PFA. A PFA defines a distribution over Σ^*. If a distribution over Σ^* or a stochastic language $\mathcal{D}$ is modeled by a PFA then $\mathcal{D}$ is known as a *stochastic regular language*. The set of all the stochastic regular languages is denoted as $\boldsymbol{\mathcal{SRL}(\Sigma)}$.

2.8. Stochastic Bi-Languages

Here, in order to represent a *stochastic bi-language* we will use the two alphabets Σ and Ω in a way similar as used previously for bi-languages in the non-stochastic setting. For technical reasons, to denote the end of an input string we use a special symbol $\sharp \notin \Sigma$ as an end marker.

A stochastic bi-language $\mathcal{R}$ is given by a function $Pr_{\mathcal{R}} : \Sigma^*\sharp \times \Omega^* \to \mathbb{R}_+$, such that:

$$\sum_{u \in \Sigma^*\sharp} \sum_{\mathbf{v} \in \Omega^*} Pr_{\mathcal{R}}(u, \mathbf{v}) = 1,$$

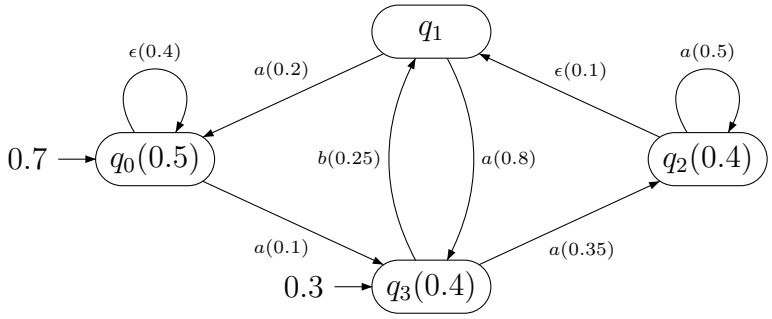


Figure 2.12.: A graphical representation of a PFA.

where $Pr_{\mathcal{R}}(u, \mathbf{v})$ is the joint probability of u and $\mathbf{v}$. Let $\mathcal{L} \subset \Sigma^* \sharp$ and $\mathcal{L}' \subset \Omega$;

$$Pr_{\mathcal{R}}(\mathcal{L}, \mathcal{L}') = \sum_{u \in \Sigma^* \sharp} \sum_{\mathbf{v} \in \Omega^*} Pr_{\mathcal{R}}(u, \mathbf{v}).$$

Example 1 *The stochastic bi-language* $\mathcal{R}$ *where* $Pr_{\mathcal{R}}(a^n \sharp, \mathbf{1}^n) = \frac{1}{2^n}, \forall n > 0$, *and* $Pr_{\mathcal{R}}(u, \mathbf{v}) = 0$ *for every other pair.*

In the sequel we will use $\mathcal{R}$ to denote a stochastic bi-language and T to denote a transducer. Note that the end marker $\sharp$ is needed for technical reasons only. The probability of generating the $\sharp$ symbol is equivalent to the stopping probability of an input string.

2.9. Probabilistic Transducers

So far, we have considered transducers defined over the boolean semiring, *i.e.*, the weights of the edges are either 1 or 0. Sometimes, it is necessary to model scenarios where the weights have to be real numbers within $[0, 1]$, *e.g.*, in machine translation, speech processing, and bioinformatics. If the weights of a rational transducer are defined under the probability semiring, we call it a *probabilistic finite state transducer* (PFST). The formal definition of a PFST is given in the following sub-section.

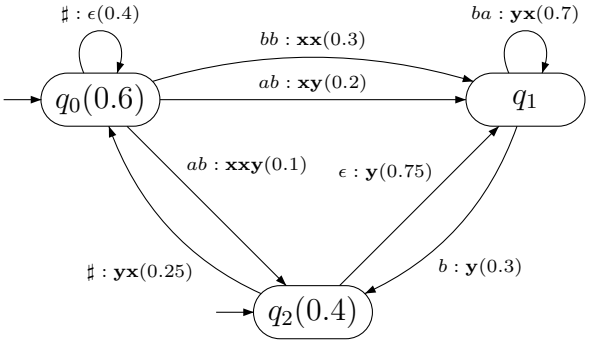


Figure 2.13.: An example of a PFST. The transition labels should be interpreted as: *input*:**output**(*transition probability*) and the state labels as *state*:(*initial state probability*).

2.9.1. Probabilistic Finite State Transducers

Definition 13 (Probabilistic Finite State Transducer) *A probabilistic finite state transducer (*PFST*) is a quintuple* $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, E, \lambda \rangle$ *over the probabilistic semiring* $\mathbb{R}_+$ *where:*

- Q *is a finite set of states,*
- Σ *and* Ω *are the input and the output alphabets,*
- $\sharp$ *is a special input symbol,*
- $E \subseteq Q \times \Sigma^* \cup \{\sharp\} \times \Omega^* \times \mathbb{R}_+ \times Q$ *and* $\forall e \in E, i[e] = \sharp \Rightarrow next[e] = q_0$,
- λ *is the initial state probability,*
- *subject to the following normalization conditions:*

$$\sum_{q \in Q} \lambda(q) = 1,$$

$$\forall q \in Q, \sum_{e \in E[q]} prob[e] = 1.$$

A PFST T defines a joint probability distribution or a stochastic bi-language $\mathcal{R}_T$ over $\Sigma^*\sharp \times \Omega^*$. We denote the probability of an edge $e \in E$ as $prob[e]$ and

the probability of a path $\pi \in \Pi_T$ as $prob\,[\pi]$. The probability of a path $\pi \in \Pi_T$ is given by:

$$Pr_T(\pi) = \lambda(prev\,[\pi]) \left(\prod_{i=1}^{|\pi|} prob\,[e_i] \right).$$

The probability of a translation pair $(x, \mathbf{y}) \in \mathcal{R}_T$ is denoted as $Pr_T(x, \mathbf{y})$ and can be computed as:

$$Pr_T(x, \mathbf{y}) = \sum_{\pi \in \Pi_T(x, \mathbf{y})} Pr_T(\pi).$$

PFSTs are generative models. The stochastic bi-languages that are generated by PFSTs are called *stochastic rational bi-languages*. The set of all stochastic rational bi-languages is denoted as $\boldsymbol{\mathcal{SRBL}(\Sigma, \Omega)}$.

Figure 2.13 depicts an example of a PFST. Here, the stopping probability of a string is given by the probability of the edges with the input symbol $\sharp$. This is equivalent to the stopping probability or the final probability of a given state. For technical convenience, the destination state of the edges with the $\sharp$ symbol is one particular initial state, which is in case of the PFST in Figure 2.13, the state q_0.

Similar to its non-probabilistic counterpart, rational transducers (Sub-section 2.4.1), PFSTs can also be ambiguous. We will call a PFST *un-ambiguous in terms of translation* if the following holds:

$$\forall x \in \Sigma^*, \forall \mathbf{y}, \mathbf{y}' \in \Omega^*, Pr_T(x\sharp, \mathbf{y}) > 0 \text{ and } Pr_T(x\sharp, \mathbf{y}') > 0 \Rightarrow \mathbf{y} = \mathbf{y}'.$$

Notice that by definition a PFST is a non-deterministic machine. Unlike a regular finite-state automaton, where for every Non-deterministic Finite Automaton (NFA) there exists an equivalent Deterministic Finite Automaton (DFA), a transducer does not always have an equivalent *deterministic* or *subsequential* counterpart. Determinization is only possible for transducers obeying the twins property (see Appendix C) and equivalent deterministic machines can be obtained in *p*-subsequential form (Allauzen and Mohri, 2003b). Therefore, it logically follows that the expressive power of a PFST is more than that of a deterministic form of probabilistic transducers. In the following sub-sections we will examine PFSTs with some deterministic properties.

2.9.2. Probabilistic Semi Deterministic Transducers

Here, we define an object having some *weak* deterministic characteristics. Essentially we extend the definition of PFST and impose some conditions of determinism on it.

Definition 14 *A probabilistic semi deterministic transducer (*PSDT*) is a quintuple* $T = \langle \Sigma \cup \{\sharp\}, \Omega, Q, E, \lambda \rangle$ *over the probability semiring* $\mathbb{R}_+$ *where* $\psi(T) = \langle \Sigma \cup \{\sharp\}, \Omega, Q, E, \lambda \rangle$ *is a* PFST *such that the following conditions hold:*

- $\exists! q_0 \in Q : \lambda(q_0) = 1$,
- $\forall q \in Q, \forall e, e' \in E[q], i[e] = i[e'] \Rightarrow next[e] = next[e']$,
- *and subject to the following condition:* $\forall q \in Q, \forall e', e \in E[q]$, *if* $i[e] = i[e'] \Rightarrow o[e] \notin Pref(o[e']) \wedge o[e'] \notin Pref(o[e])$.

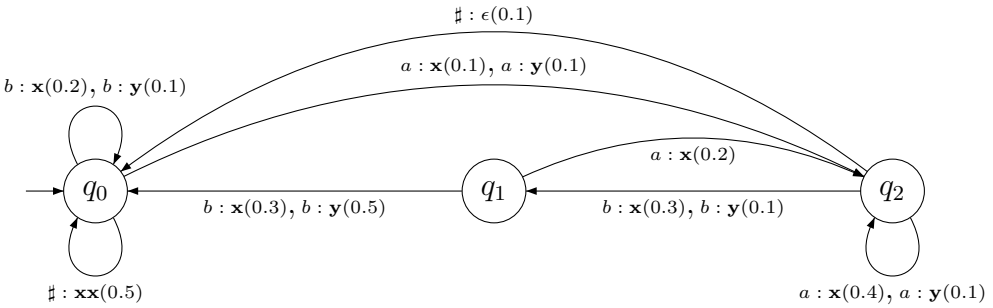


Figure 2.14.: An example of a PSDT (weak deterministic).

Figure 2.14 depicts an example of a PSDT. Note that as per the definition (Definition 14) there is exactly one initial state q_0 where $\lambda(q_0) = 1$. The initial state probability (always equal to 1) has not been shown for simplicity.

We will call the bi-languages that are generated by PSDTs: *stochastic semi regular bi-languages*. The set of all the stochastic semi regular bi-languages is denoted as $\boldsymbol{\mathcal{SSREGBL}(\Sigma, \Omega)}$.

Corollary 1 $\forall \mathcal{R} \in \boldsymbol{\mathcal{SSREGBL}(\Sigma, \Omega)}$, *for a given input string* $x \in \Sigma^*\sharp$, *the set* $\{\mathbf{y} : Pr_{\mathcal{R}}(x, \mathbf{y}) \neq 0\}$ *is finite.*

Lemma 9 $\exists \mathcal{R} \in \boldsymbol{\mathcal{SRBL}(\Sigma, \Omega)}$ *such that* $\mathcal{R} \notin \boldsymbol{\mathcal{SSREGBL}(\Sigma, \Omega)}$.

Proof Figure 2.15 depicts a PFST for which there exists no equivalent PSDT, hence, this is precisely a counter example. Because of the input ϵ loops, the number of translations for the input string $a\sharp$ is infinite. Therefore, it violates Corollary 1. □

Lemma 10 $\forall \mathcal{R} \in \boldsymbol{\mathcal{SSREGBL}(\Sigma, \Omega)}$, $\exists \mathcal{R} \in \boldsymbol{\mathcal{SRBL}(\Sigma, \Omega)}$.

Proof The proof is trivial and follows from Definition 13 and Definition 14. □

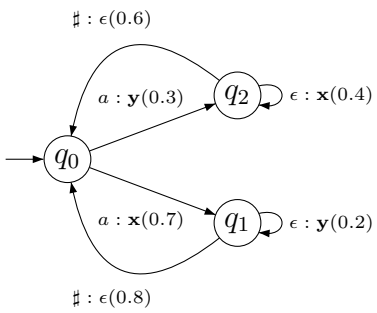


Figure 2.15.: A counter example for Lemma 9.

2.9.3. Probabilistic p-Subsequential Transducers

Definition 15 *A **probabilistic p-subsequential transducer** (*PPST*) is a quintuple $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E\rangle$ defined over the probability semiring $\mathbb{R}_+$ where:*

- *Q is a non-empty finite set of states,*
- *$q_0 \in Q$ is the unique initial state,*
- *Σ and Ω are the input and the output alphabets,*
- *$E \subseteq Q \times \Sigma \cup \{\sharp\} \times \Omega^* \times \mathbb{R}_+ \times Q$,*
- *subject to the following conditions:*
  - $\forall q \in Q$
    $\forall (q, u, \mathbf{v}, \alpha, q'), (q, u', \mathbf{v}', \beta, q'') \in E,\ u = u' \neq \sharp \Rightarrow \mathbf{v} = \mathbf{v}',\ \alpha = \beta,\ q' = q''$
  - $\forall q \in Q, \sum_{a \in \Sigma \cup \{\sharp\}, q' \in Q} Pr(q, a, q') = 1,$
  - $\forall (q, u, \mathbf{v}, \alpha, q') \in E, u = \sharp \Rightarrow q' = q_0,$
  - $\forall q \in Q, |\{e \colon e \in E\,[q]\,, i\,[e] = \sharp\}| = p.$

Figure 2.16 depicts an example of a PPST where $p = 2$. We will call the bi-languages that are generates by PPSTs: *stochastic p-regular bi-languages*. The set of all the stochastic p-regular bi-languages is denoted as $\boldsymbol{\mathcal{SPREGBL}(\Sigma, \Omega)}$.

Corollary 2 *Let T be a probabilistic p-subsequential transducer and $x \in \Sigma^*\sharp$ such that $x = x_1 \dots x_{|x|}$. T produces p number of different outputs for x denoted as $\mathbf{y}_1, \dots, \mathbf{y}_p$ where $\mathbf{y}_i \in \Omega^*$. Then the following statement is true:*

$$o\,[e_1] \cdot \dots \cdot o\left[e_{|x|}\right] \in \mathit{Pref}(\mathit{lcp}(\mathbf{y}_1, \dots, \mathbf{y}_p)) \text{ where } e_i \in E \text{ and } i\,[e_i] = x_i.$$

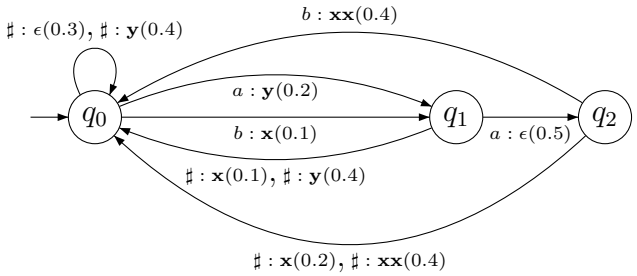


Figure 2.16.: An example of a *probabilistic* 2*-subsequential transducer.*

Corollary 3 $\forall \mathcal{R} \in \boldsymbol{\mathcal{SPREGBL}(\Sigma, \Omega)}$*, for a given input string* $x \in \Sigma^* \sharp$*,* $|\{\mathbf{y} : Pr_{\mathcal{R}}(x, \mathbf{y}) \neq 0\}| = p$*.*

Lemma 11 $\exists \mathcal{R} \in \boldsymbol{\mathcal{SSREGBL}(\Sigma, \Omega)}$ *such that* $\mathcal{R} \notin \boldsymbol{\mathcal{SPREGBL}(\Sigma, \Omega)}$*.*

Proof We show a counter example of an $\mathcal{R} \in \boldsymbol{\mathcal{SSREGBL}(\Sigma, \Omega)}$ which is represented by the PSDT depicted in Figure 2.17. Figure 2.17 is a counter example because it violates Corollary 2. Moreover, the stochastic bi-language represented by the counter example violates Corollary 3. Hence, the lemma holds. □

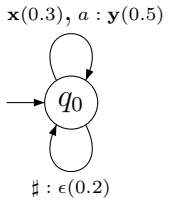


Figure 2.17.: A counter example for Lemma 11.

Lemma 12 $\forall \mathcal{R} \in \boldsymbol{\mathcal{SPREGBL}(\Sigma, \Omega)}$*,* $\exists \mathcal{R} \in \boldsymbol{\mathcal{SSREGBL}(\Sigma, \Omega)}$*.*

Proof The proof of the lemma is trivial and can be shown by construction, because probabilistic p-subsequential transducers are essentially the special case for a PSDT where many transitions are possible only for the ♯ transitions. □

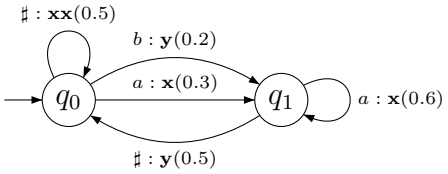


Figure 2.18.: Graphical representation of a PST.

2.9.4. Probabilistic Subsequential Transducers

Definition 16 *A **Probabilistic Subsequential Transducer** (PST) is a quintuple* $T = \langle \Sigma, \Omega, Q, E, \lambda \rangle$ *over the probability semiring* $\mathbb{R}_+$ *where* $\psi(T) = \langle \Sigma, \Omega, Q, E, \lambda \rangle$ *is a* PFST *such that:*

- *the following deterministic conditions hold:*
  - $\exists! q_0 \in Q : \lambda(q_0) = 1$,
  - $\forall q \in Q, \forall e, e' \in E[q], i[e] = i[e'] \Rightarrow o[e] = o[e'] \wedge prob[e] = prob[e'] \wedge next[e] = next[e']$.

Figure 2.18 depicts an example of a PST.

Given a PST T, for any $x \in \Sigma^*$, there exists a translation $\mathbf{y} \in \Omega^*$ if $Pr_T(x\sharp, \mathbf{y}) \neq 0$. The set of all the input strings for which there is a valid translation generated by T is defined as:

$$\mathfrak{Dom}(T) = \{x : x \in \Sigma^* \wedge Pr_T(x\sharp, \mathbf{y}) \neq 0\}.$$

Property 5 *Let* T *be a* PST. *For a string* $x \in \mathfrak{Dom}(T)$, *there exists a unique path* $\pi \in \Pi_T$ *such that* $i[\pi] = x$.

Proof Let $\pi_1, \pi_2 \in \Pi_T$ be two paths such that $i[\pi_1] = i[\pi_2] = x$. We will show that $\pi_1 = \pi_2$.

For $1 \leq i \leq |x|$, we can write: $x = x_1 \dots x_{|x|}$ where $x_i \in \Sigma$, $\pi_1 = e_1 \dots e_{|x|}$, and $\pi_2 = e'_1 \dots e'_{|x|}$. In order to prove $\pi_1 = \pi_2$, it suffices to show that for $1 \leq i \leq |x|$, $e_i = e'_i$. We will show $e_i = e'_i$ by induction.

Induction basis: Since $x \in \mathfrak{Dom}(T)$, for $i = 1$,

$$q_0 = prev[e_1] = prev[e'_1] \wedge x_1 = i[e_1] = i[e'_1] \Rightarrow o[e_1] = o[e'_1] \wedge next[e_1] = next[e'_1].$$

Therefore, $e_1 = e'_1$.

Induction hypothesis: For $i = n$, $e_n = e'_n$.

Inductive step: From induction hypothesis we can write:

$$next\left[e_n\right] = next\left[e'_n\right] = prev\left[e_{n+1}\right] = prev\left[e'_{n+1}\right].$$

Moreover, since $i\left[e_n\right] = i\left[e'_n\right] = x_n$, it follows that, $i\left[e_{n+1}\right] = x_{n+1} = i\left[e'_{n+1}\right]$. Which implies that,

$$o\left[e_{n+1}\right] = o\left[e'_{n+1}\right] \wedge next\left[e_{n+1}\right] = next\left[e'_{n+1}\right].$$

Therefore, for $1 \leq i \leq |x|$, $e_i = e'_i$. □

Proposition 3 PSTs *are un-ambiguous.*

Proof The proof of the proposition is the direct consequence of Property 5. □

We call the bi-languages that are recognized by PSTs: *stochastic regular bi-languages*. The set of all the stochastic regular bi-languages is denoted as $\boldsymbol{\mathcal{SREGBL}(\Sigma, \Omega)}$.

Corollary 4 $\forall \mathcal{R} \in \boldsymbol{\mathcal{SREGBL}(\Sigma, \Omega)}$, *for a given input string* $x \in \Sigma^*\sharp$, $|\{\mathbf{y} : Pr_{\mathcal{R}}(x, \mathbf{y}) \neq 0\}| = 1$.

Lemma 13 $\exists \mathcal{R} \in \boldsymbol{\mathcal{SPREGBL}(\Sigma, \Omega)}$ *such that* $\mathcal{R} \notin \boldsymbol{\mathcal{SREGBL}(\Sigma, \Omega)}$.

Proof We provide a counter example to show the proof. Figure 2.19 is precisely such an example. The stochastic bi-language represented by Figure 2.19 is a counter example because it violates Proposition 3 and Corollary 4. Therefore, the lemma holds. □

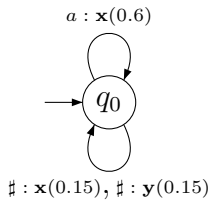


Figure 2.19.: A counter example for Lemma 13.

Lemma 14 $\forall \mathcal{R} \in \boldsymbol{\mathcal{SREGBL}(\Sigma, \Omega)}$, $\exists \mathcal{R} \in \boldsymbol{\mathcal{SPREGBL}(\Sigma, \Omega)}$.

Proof The proof of the lemma is trivial and can be shown by construction. □

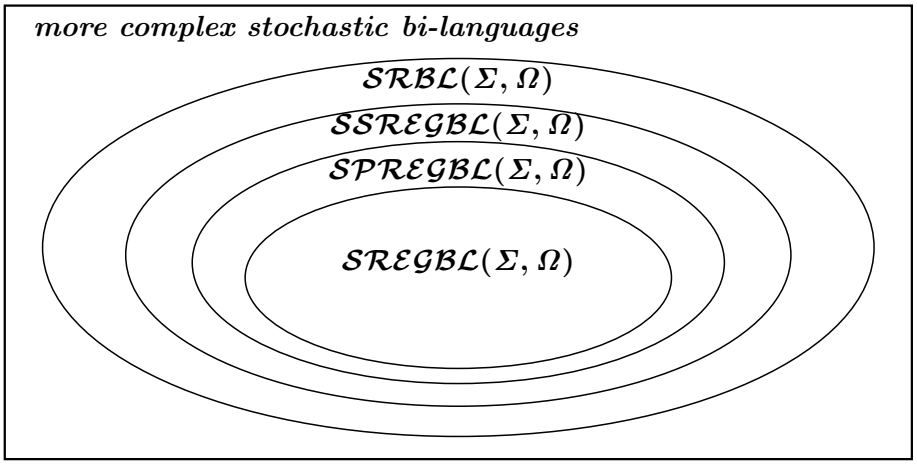


Figure 2.20.: A Venn diagram of the sets of the stochastic bi-languages modeled by probabilistic transducers.

2.10. Hierarchy of Stochastic Bi-Languages

In this section we show the hierarchy of stochastic bi-languages modeled by finite state machines. First, we show a theorem of strict inclusion as a proof of the hierarchy and then present the result by means of a Venn diagram.

Theorem 2 *Theorem of strict inclusion:*

1. $\mathcal{SREGBL}(\Sigma, \Omega) \subsetneq \mathcal{SPREGBL}(\Sigma, \Omega)$;
2. $\mathcal{SPREGBL}(\Sigma, \Omega) \subsetneq \mathcal{SSREGBL}(\Sigma, \Omega)$;
3. $\mathcal{SSREGBL}(\Sigma, \Omega) \subsetneq \mathcal{SRBL}(\Sigma, \Omega)$.

Proof The proof of 1 is a consequence of the lemmata 9 and 10. The proof of 2 is a consequence of the lemmata 11 and 12. And finally, the proof of 3 is a consequence of the lemmata 13 and 14. □

The hierarchy of stochastic bi-languages is summarized in Figure 2.20.

2.11. Parsing

Parsing in case of probabilistic transducers can have different meanings depending on the context. In different application areas, there are specific requirements

of computations using the same formal device. For instance, in statistical machine translation it is often required to compute the most probable translation of a given string using a probabilistic transducer. In this section we will look at different types of computation problems from a generic viewpoint and independent of its application areas with the types of probabilistic transducers we have defined in the previous section.

In our study, we consider PFSTs having no transitions with input and output as empty strings at the same time. We call them (ϵ, ϵ)-*free* PFSTs. Formally, an (ϵ, ϵ)-*free* PFST T is a PFST with the following conditions:

$$\begin{aligned} \forall e \in E, \quad & i[e] = \epsilon \Rightarrow o[e] \neq \epsilon, \\ & o[e] = \epsilon \Rightarrow i[e] \neq \epsilon. \end{aligned}$$

If the transducer contains such transitions, it is required to compute the probability of moving freely (*i.e.*, without having to generate any input and output) from one state to another, which is itself very expensive. There are algorithms to remove (ϵ, ϵ) transitions from PFSTs, such as (Hanneforth, 2008; Hanneforth and de la Higuera, 2010; Mohri, 2000a, 2002a). The mentioned algorithms are for removing ϵ-transitions in PFAs, but can be adapted to this case. By means of such algorithms, one can obtain an (ϵ, ϵ)-free PFST and apply the parsing algorithms we present here. Therefore, in our study we omit such cases and only focus on the (ϵ, ϵ)-free probabilistic transducers.

An (ϵ, ϵ)-*free* PFST is said to be in a *normal form* if the following condition holds:

$$\forall e \in E, (i[e] \in \Sigma \cup \{\sharp\} \wedge o[e] = \epsilon) \vee (o[e] \in \Omega \wedge i[e] = \epsilon).$$

Less formally, for each transition, the input-output pair can either be in the form of $(a, \epsilon), a \in \Sigma \cup \{\sharp\}$ or in the form of $(\epsilon, \mathbf{b}), \mathbf{b} \in \Omega$. Normalization is, in all cases, polynomial. Next, we show that every (ϵ, ϵ)-free PFST has a normal form.

Proposition 4 *Let T be an (ϵ, ϵ)-free* PFST *that represents a stochastic rational bi-language $\mathcal{BL}_T$. There exists an (ϵ, ϵ)-free* PFST *T' in the normal form that represents a stochastic rational bi-language $\mathcal{BL}_{T'}$ such that $\mathcal{BL}_T = \mathcal{BL}_{T'}$.*

Proof We will show the proof of the proposition by construction. Algorithm 1 constructs an equivalent PFST in the normal form. The correctness of the construction can be argued as the following: the algorithm iterates over each edge of the input PFST. If a particular edge is already in the normal form, its adds a copy of the edge to the PFST under construction (line 5). In other cases it adds symbol by symbol for each input strings with an ϵ output (the

first innermost for loop) and symbol by symbol for each output string with an ϵ output. The actual probability of the edge of the input PFST is only added once (line 8) and the for the rest of the new edges, the probability is 1. Therefore, it is easy to see that the constructed PFST in the normal form is equivalent to the input PFST. □

2.11.1. Computing the Joint Probability of a String Pair

The problem definition: Given a pair of strings $(x, \mathbf{y})$ where $x \in \Sigma^*$ and $\mathbf{y} \in \Omega^*$, and a PFST $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, E, \lambda \rangle$, compute $Pr_T(x, \mathbf{y})$.

First, as a syntactic machine, we consider a PFST T given in the normal form. Here we present a forward algorithm to compute $Pr_T(x, \mathbf{y})$.

Let $\mathbf{T}$ be a three dimensional matrix of size $|x| \times |\mathbf{v}| \times |Q|$, $x = x_1 \dots x_{|x|}$ where $x_i \in \Sigma$, and $\mathbf{y} = \mathbf{y}_1 \dots \mathbf{y}_{|\mathbf{y}|}$ where $\mathbf{y}_i \in \Omega$. The first steps to compute $Pr_T(x, \mathbf{y})$ are the following:

For $i = 0$ to $|Q| - 1$,

$$\mathbf{T}[0, 0, k] \leftarrow \lambda(q_k);$$

For $i = 0$ to $|x| - 1$, for $j = 0$ to $|\mathbf{y}| - 1$, for $k = 0$ to $|Q| - 1$,

$$\begin{aligned} \mathbf{T}[i+1, j+1, k] \leftarrow \quad & \textstyle\sum_{l=0}^{|Q-1|} \mathbf{T}[i, j+1, l]\, Pr_T(q_l, x_{i+1}, \epsilon, q_k) \\ & + \textstyle\sum_{l=0}^{|Q-1|} \mathbf{T}[i+1, j, l]\, Pr_T(q_l, \epsilon, \mathbf{y}_{j+1}, q_k); \end{aligned}$$

After executing the above steps, $\mathbf{T}[n, m, k]$ contains the joint probability of the prefix $x_1 \dots x_n$ with $n \leq |x|$, $\mathbf{y}_1 \dots \mathbf{y}_m$ with $m \leq |\mathbf{y}|$, and being in the state q_k with $k \leq |Q| - 1$. The probability of the string pair $(x, \mathbf{y})$ ending in the state q_k can be computed by multiplying the $\sharp$ transition probabilities of q_k using the following formula:

$$\mathbf{T}[|x| - 1, |\mathbf{y}| - 1, k] \cdot \sum_{e \in \{e : e \in E[q_k], i[e] = \sharp\}} prob[e].$$

Finally, if we sum up all the probabilities of the string pair $(x, \mathbf{y})$, we obtain $Pr_T(x, \mathbf{y})$. Formally,

$$Pr_T(x, \mathbf{y}) = \sum_{k=0}^{|Q|-1} \mathbf{T}[|x| - 1, |\mathbf{y}| - 1, k] \sum_{e \in \{e : e \in E[q_k], i[e] = \sharp\}} prob[e].$$

The runtime complexity of the presented algorithm is $\mathcal{O}\big((|x| + |\mathbf{y}|)\,|Q|^2\big)$.

Algorithm 1: NORMALIZE

Input: a PFST $T = \langle \Sigma, \Omega, Q, E, \lambda \rangle$ where $E \subseteq Q \times \Sigma^* \times \Omega^* \times \mathbb{R}_+ \times Q$
Output: an equivalent PFST $T' = \langle \Sigma, \Omega, Q', E', \lambda' \rangle$ in the normal form.

for $q \in Q$ **do**
  $Q' \leftarrow \{q\} \cup Q'$;
  $\lambda'(q) \leftarrow \lambda(q)$;
  for $e \in E[q]$ **do**
    if $|i[e]| = 1 \wedge |o[e]| = 0$ ***or*** $|i[e]| = 0 \wedge |o[e]| = 1$ **then**
      $Q' \leftarrow \{next[e]\} \cup Q'$;
      $\lambda'(next[e]) \leftarrow \lambda(next[e])$;
      $E' \leftarrow E' \cup (q, i[e], o[e], prob[e], next[e])$;
      exit;
    Let s_i be the i^{th} symbol of $i[e]$;
    for $i = 1$ *to* $|i[e]|$ **do**
      switch *value of* i **do**
        case $i = 1$
          $Q' \leftarrow \{q_{s_i}\} \cup Q'$;
          $\lambda'(q_{s_i}) \leftarrow 0$;
          $E' \leftarrow E' \cup (q, s_i, \epsilon, prob[e], q_{s_i})$;
        case $i = |i[e]|$
          $Q' \leftarrow \{next[e]\} \cup Q'$;
          $\lambda'(next[e]) \leftarrow \lambda(next[e])$;
          $E' \leftarrow E' \cup (q_{s_{i-1}r_{i-1}}, s_i, \epsilon, 1, q_{s_i})$;
        otherwise
          $Q' \leftarrow \{q_{s_i r_i}\} \cup Q'$;
          $\lambda'(q_{s_i}) \leftarrow 0$;
          $E' \leftarrow E' \cup (q_{s_{i-1}}, s_i, \epsilon, 1, q_{s_i})$;
    Let r_i be the i^{th} symbol of $o[e]$;
    for $i = 1$ *to* $|o[e]|$ **do**
      switch *value of* i **do**
        case $i = 1$
          $Q' \leftarrow \{q_{r_i}\} \cup Q'$;
          $\lambda'(q_{r_i}) \leftarrow 0$;
          $E' \leftarrow E' \cup (q_{s_{|i[e]|}}, \epsilon, r_i, 1, q_{r_i})$;
        case $i = |o[e]|$
          $Q' \leftarrow \{next[e]\} \cup Q'$;
          $\lambda'(next[e]) \leftarrow \lambda(next[e])$;
          $E' \leftarrow E' \cup (q_{r_{i-1}}, \epsilon, r_i, 1, next[e])$;
        otherwise
          $Q' \leftarrow \{q_{s_i r_i}\} \cup Q'$;
          $\lambda'(q_{s_i r_i}) \leftarrow 0$;
          $E' \leftarrow E' \cup (q_{r_{i-1}}, \epsilon, r_i, 1, q_{r_i})$;

A similar algorithm can be used for computing the joint probability of a string pair using a PSDT. The algorithm is the following: let $\mathbf{T}$ be a three dimensional matrix of size $|x| \times |\mathbf{v}| \times |Q|$, $x = x_1 \dots x_{|x|}$ where $x_i \in \Sigma$, and $\mathbf{y} = \mathbf{y}_1 \dots \mathbf{y}_{|\mathbf{y}|}$ where $\mathbf{y}_i \in \Omega$. Since PSDT has exactly one initial state, we initialize $\mathbf{T}$ as the following:

$$\mathbf{T}[0,0,0] \leftarrow 1;$$

The recurrence relation is the following:

$$\mathbf{T}[i+1,j,k] \leftarrow \sum_{r \leq j} \mathbf{T}[i,j,k'] \cdot Pr_T(q_{k'}, x_i, y_{r+1} \dots y_j, q_k),$$

where $\mathbf{T}[i,j,k]$ holds the probability of being in state q_k after having parsed $x_1 \dots x_i$ and $\mathbf{y}_1 \dots \mathbf{y}_j$; $q_{k'}$ is the unique state reached by parsing $x_1 \dots x_i$ from the initial state q_0. The complexity will therefore be $\mathcal{O}(|x|\,|y|^2)$.

For a PPST the runtime complexity remains the same and the same algorithm can be used to compute the joint probability.

If the syntactic machine T is a PST, computing $Pr_T(x, \mathbf{y})$ is rather straight forward and can be done in linear time *w.r.t.* the length of x. Since PSTs are deterministic *w.r.t.* the input, for $x = x_1 \dots x_{|x|}$, there exists a unique path $\pi \in \Pi_T$, such that $\pi = e_1 \dots e_{|x|}$ and for $1 \leq i \leq |x|$, $i[e] = x_i$ (Property 5). The algorithm is the following:

$$Pr_T(x, \mathbf{y}) = \Big(\prod_{i=1}^{|x|} prob[e_i] \Big) prob[e] \text{ where } prev[e] = next\left[e_{|x|}\right], i[e] = \sharp.$$

The runtime complexity of the above algorithm is $\mathcal{O}(|x|)$.

2.11.2. Computing the Probability of an Input String

The problem definition: Given a string $x \in \Sigma^*$ and a PFST $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, E, \lambda \rangle$, compute $Pr_T(x, \Omega^*)$, which is known as a *marginal*.

Computing the probability of an input string x in a PFST, is essentially similar to computing the probability of a string in a PFA. This can be done using the classical forward algorithm described in (Baum et al., 1970). If the PFST is by definition (ϵ, ϵ)-free and is in the normal form, the runtime complexity is $\mathcal{O}(|x|\,|Q|^2)$. Details about the forward algorithm can be found in (de la Higuera, 2010).

In case of a PSDT, the problem of computing the probability of an input string can be achieved in linear time. Let T be a PSDT. There exists a unique

sequence of states $q_0 \dots q_{|x|}$ that forms the paths for generating $x = x_1 \dots x_{|x|}$. The algorithm for computing the probability of x using T is the following:

$$Pr_T(x, \Omega^*) = \Big(\prod_{i=0}^{|x|-1} \sum_{e \in E[q_i]: i[e] = x_{i+1}} prob\,[e] \Big) \sum_{e \in E\left[q_{|x|}\right]: i[e] = \sharp} prob\,[e].$$

The runtime complexity of the above algorithm is bounded by $\mathcal{O}(|x|\, b + s)$ where b is the *branching factor* defined as:

$$b = max\{j : j = |\{e : e \in E\,[q_{k-1}]\,, i\,[e] = x_k\}|\,, 1 \leq k \leq |x|\}$$

and s is the *stopping factor* defined as:

$$s = \left|\{e : e \in E\left[q_{|x|}\right], i\,[e] = \sharp\}\right|.$$

Less formally, branching factor is the maximum of the number of outgoing edges per state used to generate x and stopping factor is the number of outgoing edges with the stopping symbol $\sharp$ at the last state while generating x.

If the syntactic machine is a PPST or a PST, the computation of $Pr_T(x, \Omega^*)$ is also linear and the runtime complexities are $\mathcal{O}(|x| + p)$ and $\mathcal{O}(|x|)$ respectively. For PPST, the algorithm is the following:

$$Pr_T(x, \Omega^*) = \Big(\prod_{i=1}^{|x|} prob\,[e_i] \Big) \sum_{e \in \{e : e \in E\left[next\left[e_{|x|}\right]\right], i[e] = \sharp\}} prob\,[e],$$

and for PST the algorithm is the following:

$$Pr_T(x, \Omega^*) = \Big(\prod_{i=1}^{|x|} prob\,[e_i] \Big) prob\,[e] \;\text{ where } prev\,[e] = next\left[e_{|x|}\right], i\,[e] = \sharp.$$

2.11.3. Computation of the Conditional Probability of a Translation

The problem definition: Given a pair of strings $(x, \mathbf{y})$ where $x \in \Sigma^*$ and $\mathbf{y} \in \Omega^*$, and a PFST $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, E, \lambda \rangle$, compute $Pr_T(\mathbf{y}|x)$.

The conditional probability of $\mathbf{y}$ given an input string x can be written as:

$$Pr_T(\mathbf{y}|x) = \frac{Pr_T(x, \mathbf{y})}{Pr_T(x, \Omega^*)}.$$

Therefore, by using the solutions presented in the previous two sub-sections, it is possible to compute $Pr_T(\mathbf{y}|x)$. If the syntactic machine is a PFST, then the runtime complexity for computing $Pr_T(\mathbf{y}|x)$ is $\mathcal{O}\big((|x| + |\mathbf{y}|)Q|^2\big)$.

If the syntactic machine is a PSDT, or a PPST, then the runtime complexity for computing $Pr_T(\mathbf{y}|x)$ is $\mathcal{O}(|x| + |y|^2)$.

In case of a PST, the runtime complexity of computing $Pr_T(\mathbf{y}|x)$ is $\mathcal{O}(|x|)$.

2.11.4. The Most Probable Translation of a String

The problem definition: Given an input string $x \in \Sigma^*$, and a PFST $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, E, \lambda \rangle$, compute:

$$\underset{\mathbf{y}}{\operatorname{argmax}}\, Pr_T(x, \mathbf{y}).$$

Otherwise stated, find $\mathbf{y}$ such that:

$$\forall \mathbf{z} \in \Omega^*, Pr_T(x\sharp, \mathbf{z}) \leq Pr_T(x\sharp, \mathbf{y}).$$

In case of a PFST, the problem of finding the most probable translation is proven to be $\mathcal{NP}$-Hard by de la Higuera and Casacuberta (2000). An approach to find an approximate solution has been presented in (Casacuberta, 1995; Picó and Casacuberta, 2001).

However, with PSDTs, PPSTs, and PSTs, it is possible to obtain the most probable translation of a given input string in linear time.

For PSDTs, an algorithm for computing the most probable translation of a given string is presented by Algorithm 2. The runtime complexity of Algorithm 2 is bounded by $\mathcal{O}(|x|\, b + s)$.

Algorithm 3 is an algorithm for finding the most probable translation when the syntactic machine is a PPST. The runtime complexity of Algorithm 3 is $\mathcal{O}(|x| + p)$.

Finally, Algorithm 4 is an algorithm when the transducer is a PST and the runtime complexity is $\mathcal{O}(|x|)$.

2.11.5. Computation of all Translations of x such that $\mathbf{Pr}(\mathbf{y}|x) > \delta$

The problem definition: Given an input string $x \in \Sigma^*$, and a PFST $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, E, \lambda \rangle$, and $0 \leq \delta \leq 1$, compute:

$$\{\mathbf{y} : \mathbf{y} \in \Omega^*, Pr_T(\mathbf{y}|x) > \delta\}.$$

Algorithm 2: MostProbableTranslation1

Input: a PSDT $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, E, \lambda \rangle$, a string $x = x_1 \dots x_{|x|}$ where $x \in \Sigma^*$ and $x_i \in \Sigma$

Output: $\mathbf{y} \in \Omega^*$ such that $\forall \mathbf{z} \in \Omega^*, Pr_T(x\sharp, \mathbf{z}) \leq Pr_T(x\sharp, \mathbf{y})$, *false* if there is no such $\mathbf{y}$

```
1  q ← q_0 ∈ Q such that λ(q_0) = 1;
2  for i = 1 to |x| do
3      p ← 0;
4      for e ∈ E[q] : i[e] = x_1 do
5          if prob[e] > p then
6              p ← prob[e];
7              y' ← o[e];
8      if p = 0 then return false;
9      y ← y.y';
10     q ← next[e];
11 p ← 0;
12 for e ∈ E[q] : i[e] = ♯ do
13     if prob[e] > p then
14         p ← prob[e];
15         y' ← o[e];
16 if p > 0 then
17     return y.y';
18 else
19     return false;
```

Algorithm 3: MostProbableTranslation2

Input: a Ppst $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, E, \lambda \rangle$, a string $x = x_1 \dots x_{|x|}$ where $x \in \Sigma^*$ and $x_i \in \Sigma$

Output: $\mathbf{y} \in \Omega^*$ such that $\forall \mathbf{z} \in \Omega^*, Pr_T(x\sharp, \mathbf{z}) \leq Pr_T(x\sharp, \mathbf{y})$, *false* if there is no such $\mathbf{y}$

```
1  q ← q_0 ∈ Q such that λ(q_0) = 1;
2  for i = 1 to |x| do
3      if ∃e ∈ E[q] : i[e] = x_1 then
4          y ← y.o[e];
5          q ← next[e];
6      else
7          return false;
8  p_r ← 0;
9  for e ∈ E[q] : i[e] = ♯ do
10     if prob[e] > p_r then
11         p_r ← prob[e];
12         y' ← o[e];
13 if p_r > 0 then
14     return y.y';
15 else
16     return false;
```

Algorithm 4: MostProbableTranslation3

Input: a Pst $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, E, \lambda \rangle$, a string $x = x_1 \dots x_{|x|}$ where $x \in \Sigma^*$ and $x_i \in \Sigma$

Output: $\mathbf{y} \in \Omega^*$ such that $\forall \mathbf{z} \in \Omega^*, Pr_T(x\sharp, \mathbf{z}) \leq Pr_T(x\sharp, \mathbf{y})$, *false* if there is no such $\mathbf{y}$

```
1  q ← q_0 ∈ Q such that λ(q_0) = 1;
2  for i = 1 to |x| do
3      if ∃e ∈ E[q] : i[e] = x_1 then
4          y ← y.o[e];
5          q ← next[e];
6      else
7          return false;
8  if ∃e ∈ E[q] : i[e] = ♯ then
9      return y.o[e];
10 else
11     return false;
```

When the syntactic machine is a Pfst, this problem is known to be $\mathcal{NP}$-Hard (Casacuberta and de la Higuera, 2000). However, in a recent work, de la Higuera and Oncina (2011) have proposed a pseudo-polynomial solution where the length of $\mathbf{y}$ must be bounded by $l \in [n] \cup \{0\}$, *i.e.*, $\mathbf{y} \in \Omega^{\leq l}$. The proposed solution works is two steps: first, they have shown that all translations of a given string can be represented by a Pfa; second, as a consequence of the first step, the problem can be reduced to finding the bounded most probable strings of a Pfa and they have presented a pseudo-polynomial algorithm for finding such strings in a Pfa. The runtime complexity of the algorithm to compute the bounded most probable strings of a Pfa is reported as in $\mathcal{O}(\frac{l|\Sigma||Q_A|^2}{\delta})$ where Q_A is the set of states in the Pfa and all the arithmetic operations are assumed to be in constant time. This result implies that, computing the set of translations of x with probability greater than δ where the lengths of the output strings are bounded, can be computed in pseudo-polynomial time: the runtime required for constructing the corresponding Pfa and the runtime required for computing the bounded most probable strings of the Pfa.

When the probabilistic transducer is one of the deterministic kinds, *i.e.*, Psdt, Ppst, and Pst, the problem can be solved in linear time and without the restriction of the output string length to be bounded. This can be achieved by similar algorithms as computing the most probable translation presented in the

previous section and the runtime complexities are also the same.

2.11.6. A Conclusion about these Parsing Questions

The summary of the runtime complexities of parsing depicted in Table 2.2 shows that PSTs are the only type of probabilistic transducer where parsing in linear *w.r.t.* the length of the input string in all cases. For PSDTs and PPSTs some of the computation problems can be done in linear time, however, computing the joint probability and conditional probability are still quadratic. In case of PFSTs the computation time for each case is either quadratic or $\mathcal{NP}$-Hard. Therefore, PSTs are computationally the most inexpensive devices although they are the most limited onces in terms of expressive power.

Table 2.2.: Runtime complexities of different computations using different types of probabilistic transducers.

	$Pr(x, \mathbf{y})$	$Pr(x, \Omega^*)$	$Pr(\mathbf{y}\|x)$	$\{\mathbf{y} : \forall \mathbf{z} \in \Omega^*, Pr(x\sharp, \mathbf{z}) \leq Pr(x\sharp, \mathbf{y})\}$	$\{\mathbf{y} : Pr(\mathbf{y}\|x) > \delta\}$
PFST	$\mathcal{O}\big((\|x\| + \|\mathbf{y}\|)\,\|Q\|^2\big)$	$\mathcal{O}(\|x\|\,\|Q\|^2)$	$\mathcal{O}\big((\|x\| + \|\mathbf{y}\|)\,\|Q\|^2\big)$	$\mathcal{NP}$-Hard	$\mathcal{NP}$-Hard, pseudo-polynomial if $\mathbf{y} \in \Omega^{\leq l}$
PSDT	$\mathcal{O}(\|x\|\,\|\mathbf{y}\|^2)$	$\mathcal{O}(\|x\|\,b + s)$	$\mathcal{O}(\|x\|\,\|\mathbf{y}\|^2)$	$\mathcal{O}(\|x\|\,b + s)$	$\mathcal{O}(\|x\|\,b + s)$
PPST	$\mathcal{O}(\|x\|\,\|\mathbf{y}\|^2)$	$\mathcal{O}(\|x\| + p)$	$\mathcal{O}(\|x\|\,\|\mathbf{y}\|^2)$	$\mathcal{O}(\|x\| + p)$	$\mathcal{O}(\|x\| + p)$
PST	$\mathcal{O}(\|x\|)$	$\mathcal{O}(\|x\|)$	$\mathcal{O}(\|x\|)$	$\mathcal{O}(\|x\|)$	$\mathcal{O}(\|x\|)$

2.12. Equivalence with Other Models

2.12.1. Pair Hidden Markov Models

The relation between PFAs and hidden Markov models (HMMs) has been examined in (Casacuberta, 1990; Dupont et al., 2005; Vidal et al., 2005a). It has been proved that HMMs and PFAs are equivalent in terms of modeling distributions over stochastic languages. In this section, we investigate the relation between PFSTs and PHMMs.

PHMM was first introduced by Durbin *et al.* in 1998 in the context of *bioinformatics* (Durbin et al., 1998). Since then, PHMM has also been used in various tasks in the domain of natural language processing, *e.g.*, (Clark, 2001a,b, 2002; Mackay and Kondrak, 2005; Nabende et al., 2008; Nabende, 2009). Intuitively, PHMM is a similar object as classical HMM where, instead of emitting a single symbol at each state, a pair of symbols is emitted at each state. A formal definition of PHMM is the following:

Definition 17 *A* PHMM *is a sextuple* $M = \langle Q, \Sigma, \Omega, I, \tau, \xi \rangle$ *where:*

- Q *is a finite set of states,*
- $q_f \in Q$ *is a distinguished final state,*
- Σ *and* Ω *are the finite alphabets of symbols,*
- $I \colon Q \backslash \{q_f\} \to \mathbb{R}_+$ *is a partial function for initial state probability,*
- $\tau \colon (Q \backslash \{q_f\}) \times Q \to \mathbb{R}_+$ *is a state to state transition probability function,*
- $\xi \colon (Q \backslash \{q_f\}) \times \Sigma \cup \{\epsilon\} \times \Omega \cup \{\epsilon\} \to \mathbb{R}_+$ *is a state based probability function for emission of symbol pairs,*

subject to the following normalization conditions:

$$\sum_{q \in Q \backslash \{q_f\}} I(q) = 1,$$

$$\forall q \in Q \backslash \{q_f\}, \sum_{q' \in Q} \tau(q, q') = 1,$$

$$\forall q \in Q \backslash \{q_f\}, \sum_{a \in \Sigma \cup \{\epsilon\}} \sum_{\mathbf{b} \in \Omega \cup \{\epsilon\}} \xi(q, a, \mathbf{b}) = 1.$$

PHMMs define joint distributions over $\Sigma^* \times \Omega^*$. Unlike HMMs where the distribution is over a fixed length of string Σ^n, in case of PHMM, ϵ transitions are allowed and therefore they define distribution over $\Sigma^* \times \Omega^*$. The halting of PHMM is managed by transition probabilities to the final state (labeled as `END`). Once the final state is reached, the parsing is halted. We denote the joint distribution defined by the PHMM M as $\mathcal{R}_M$. An example of a PHMM is illustrated in Figure 2.21.

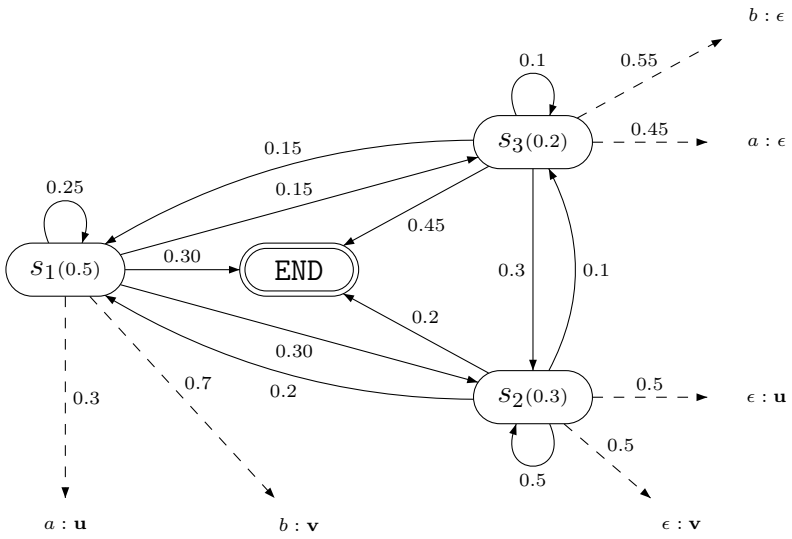


Figure 2.21.: An example of a PHMM. This example has been adapted from (Durbin et al., 1998). The state labels should be interpreted as *state*(initial state probability). The un-dotted lines are transitions labeled with transition probability and the dotted lines emissions labeled with emission probability. Each emission is a pair of input-output symbols.

We will say that the model generates or emits a pair of strings $(x, \mathbf{y})$ where x and $\mathbf{y}$ are sequences given by $x = x_1 \cdot \ldots \cdot x_k$ and $\mathbf{y} = \mathbf{y}_1 \cdot \ldots \cdot \mathbf{y}_k$ such that $x_i \in \Sigma \cup \{\epsilon\}$ and $\mathbf{y}_i \in \Omega \cup \{\epsilon\}$ and the probability of generating $(x, \mathbf{y})$ is given by $Pr_M(x, \mathbf{y})$. We will define $Pr_M(x, \mathbf{y})$ in two steps. First, let a valid path $\pi \in \Pi_M(I)$ such that $next[\pi] = q_f$. We can write $\pi = e_1 \ldots e_k$ and $e_i \in (Q \backslash \{q_f\}) \times Q$ such that $\tau(e_i) \neq 0$. Moreover, we can write $prev[e_i] = next[e_{i-1}]$ with $i \geq 2$ and $next[e_k] = q_f$. The probability of the path π is given by:

$$Pr_M(\pi) = I(prev\,[e_1]) \prod_{i=1}^{k} \tau(prev\,[e_i], next\,[e_i])$$

and the probability of generating $(x, \mathbf{y})$ through π is:

$$Pr_M((x, \mathbf{y})|\pi) = \prod_{i=1}^{k} \xi(prev\,[e_i], x_i, \mathbf{y}_i).$$

Let $\Pi_M(x, \mathbf{y})$ be the set of all valid paths for $(x, \mathbf{y})$ such that $\forall \pi \in \Pi_M(x, \mathbf{y}), \pi \in \Pi_M(I) \wedge next\,[\pi] = q_f$. Then, the probability that M generates or emits $(x, \mathbf{y})$ is:

$$Pr_M(x, \mathbf{y}) = \sum_{\pi \in \Pi_M(x, \mathbf{y})} Pr_M((x, \mathbf{y})|\pi) Pr_M(\pi).$$

Lemma 15 *Let $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, E, \lambda \rangle$ be a* PFST *where $E \subseteq Q \times \Sigma^* \times \Omega^* \times \mathbb{R}_+ \times Q$ and $\mathcal{R}_T$ be the stochastic bi-language modeled by T. There exists a* PFST *$T' = \langle Q', \Sigma \cup \{\sharp\}, \Omega, E', \lambda' \rangle$ where $E' \subseteq Q' \times \Sigma \cup \{\epsilon\} \times \Omega \cup \{\epsilon\} \times \mathbb{R}_+ \times Q'$ that models a stochastic bi-language $\mathcal{R}_{T'}$ such that $\mathcal{R}_T = \mathcal{R}_{T'}$. The number of states in T' is given by:*

$$|Q'| = \sum_{q \in Q} \sum_{e \in E[q]} max\{|i\,[e]|, |o\,[e]|\},$$

and the number of edges in T' is given by:

$$|E'| = \sum_{e \in E} max\{|i\,[e]|, |o\,[e]|\}.$$

Proof We will show the proof by construction. The algorithm CONSTRUCT1 (Algorithm 5) takes the PFST T as an input and constructs the equivalent PFST T'.

Let $(x, \mathbf{y}) \in \Sigma^* \times \Omega^*$ be an arbitrary pair of strings with $Pr_T(x, \mathbf{y}) \neq 0$ where $x = x_1 \cdot \ldots \cdot x_n$ and $\mathbf{y} = \mathbf{y}_1 \cdot \ldots \cdot \mathbf{y}_n$ such that $x_i \in \Sigma^*$ and $\mathbf{y}_i \in \Omega^*$ with $1 \leq i \leq n$. $\exists \pi \in \Pi_T(I)$ such that $\pi = e_1 \ldots e_{n+1}$ and $o\,[e_{n+1}] = \sharp$. The probability of generating $(x, \mathbf{y})$ via π is:

$$\lambda(prev\,[e_1]) \quad \Big(prob\,[e_1] \ldots prob\,[e_{n+1}]\Big).$$

Similarly, for any pair of strings $(x, \mathbf{y}) \in \Sigma^* \times \Omega^*$ with $Pr_{T'}(x, \mathbf{y}) \neq 0$, $\exists \pi \in \Pi_{T'}(I)$ such that $\pi = e'_1 \ldots e'_{m+1}$ and $o\left[e'_{m+1}\right] = \sharp$ where $m = max\{|x|, |\mathbf{y}|\}$. The probability of generating $(x, \mathbf{y})$ via π is:

$$\lambda(prev\,[e'_1]) \quad \Big(prob\,[e'_1] \ldots prob\left[e'_{m+1}\right]\Big).$$

By construction:

$$\lambda(prev\,[e_1]) = \lambda(prev\,[e'_1]),$$

$$prob\,[e_1]\ldots prob\,[e_{n+1}] = prob\,[e'_1]\ldots prob\left[e'_{m+1}\right].$$

Therefore, $\mathcal{R}_T = \mathcal{R}_{T'}$. Moreover, there are three nested for loops in the algorithm CONSTRUCT1 (Algorithm 5) where the states and the edges are added to T', therefore it is easy to see that $|Q'| = \sum_{q\in Q}\sum_{e\in E[q]} max\{|i\,[e]|\,,|o\,[e]|\}$ and $|E'| = \sum_{e\in E} max\{|i\,[e]|\,,|o\,[e]|\}$ hold. □

Theorem 3 *Given a* PFST T *with* $|E|$ *transitions, there exists a* PHMM M *with at most* m *number of states where:*

$$m = 1 + \sum_{e\in E^T} max\{|i\,[e]|\,,|o\,[e]|\}$$

such that $\mathcal{R}_T = \mathcal{R}_M$ *and* E^T *is the set of transitions in* T*.*

Proof We will show the proof of the theorem by construction in two steps. First, we will construct an equivalent PFST of T as $T' = \langle Q, \Sigma\cup\{\sharp\}, \Omega, E, \lambda\rangle$ using the construction in Lemma 15 (Algorithm 5) where $E \subseteq Q\times\Sigma\cup\{\epsilon\}\times\Omega\cup\{\epsilon\}\times\mathbb{R}_+\times Q$. Second, using T', we will create an equivalent PHMM $M = \left\langle Q^M, \Sigma, \Omega, I, \delta, \xi\right\rangle$ as follows:

- $\forall e \in E', Q^M \leftarrow Q^M \cup \{(prev\,[e]\,, next\,[e])\}$;
- $Q^M \leftarrow Q^M \cup \{q_f\}$;
- $\forall (q, q') \in Q^M\backslash\{q_f\}$,

$$I(q, q') \leftarrow \lambda(q) \sum_{e\in\{e:e\in E[q], next[e]=q', i[e]\neq\sharp\}} prob\,[e];$$

- $\forall (q, q'), (q', q'') \in Q^M\backslash\{q_f\}$,

$$\tau((q, q'), (q', q'')) \leftarrow \sum_{e\in\{e:e\in E[q'], next[e]=q'', i[e]\neq\sharp\}} prob\,[e];$$

- $\forall (q, q') \in Q^M\backslash\{q_f\}$,

$$\tau((q, q'), q_f) \leftarrow \sum_{e\in\{e:e\in E[q'], i[e]=\sharp\}} prob\,[e];$$

Algorithm 5: CONSTRUCT1

Input: a PFST $T = \langle \Sigma, \Omega, Q, E, \lambda \rangle$ where $E \subseteq Q \times \Sigma^* \times \Omega^* \times \mathbb{R}_+ \times Q$

Output: an equivalent PFST $T' = \langle \Sigma, \Omega, Q', E', \lambda' \rangle$ where $E' \subseteq Q' \times \Sigma \cup \{\epsilon\} \times \Omega \cup \{\epsilon\} \times \mathbb{R}_+ \times Q'$

```
1  for q ∈ Q do
2      Q' ← {q} ∪ Q';
3      λ'(q) ← λ(q);
4      for e ∈ E[q] do
5          if max{|i[e]|, |o[e]|} = 0 or max{|i[e]|, |o[e]|} = 1 then
6              Q' ← {next[e]} ∪ Q';
7              λ'(next[e]) ← λ(next[e]);
8              E' ← E' ∪ (q, i[e], o[e], prob[e], next[e]);
9              exit;
10         Let s_i be the i^th symbol of i[e] and r_i be the i^th symbol of o[e];
11         if i > max{|i[e]|, |o[e]|} then s_i = ε and r_i = ε;
12         for i = 1 to max{|i[e]|, |o[e]|} do
13             switch value of i do
14                 case i = 1
15                     Q' ← {q_{s_i r_i}} ∪ Q';
16                     λ'(q_{s_i}) ← 0;
17                     E' ← E' ∪ (q, s_i, r_i, prob[e], q_{s_i r_i});
18                 case i = max{|i[e]|, |o[e]|}
19                     Q' ← {next[e]} ∪ Q';
20                     λ'(next[e]) ← λ(next[e]);
21                     E' ← E' ∪ (q_{s_{i-1} r_{i-1}}, s_i, r_i, 1, next[e]);
22                 otherwise
23                     Q' ← {q_{s_i r_i}} ∪ Q';
24                     λ'(q_{s_i r_i}) ← 0;
25                     E' ← E' ∪ (q_{s_{i-1} r_{i-1}}, s_i, r_i, 1, q_{s_i r_i});
```

- $\forall (q, q') \in Q^M \backslash \{q_f\}, \forall e \in E\,[q]$,

$$\xi((q, q'), i\,[e]\,, o\,[e]) \leftarrow \frac{prob\,[e]}{\sum_{e \in \{e: e \in E[q], next[q'], i[e] \neq \sharp\}} prob\,[e]}.$$

For any pair of strings $(x, \mathbf{y}) \in \Sigma^* \times \Omega^*$ with $Pr_{T'}(x, \mathbf{y}) \neq 0$ where $x = x_1 \cdot \ldots \cdot x_n$ and $\mathbf{y} = \mathbf{y}_1 \cdot \ldots \cdot \mathbf{y}_n$ such that $x_i \in \Sigma \cup \{\epsilon\}$ and $\mathbf{y}_i \in \Omega \cup \{\epsilon\}$, $\exists \pi \in \Pi_{T'}(I)$ such that $\pi = e_1 \ldots e_{n+1}$ and $o\,[e_{n+1}] = \sharp$. The probability of generating $(x, \mathbf{y})$ through the path π is:

$$\lambda(prev\,[e_1]) \quad \Big(prob\,[e_1] \ldots prob\,[e_{n+1}]\Big).$$

For any pair of strings $(x, \mathbf{y}) \in \Sigma^* \times \Omega^*$ with $Pr_M(x, \mathbf{y}) \neq 0$, $\exists \pi \in \Pi_M(I)$ such that $\pi = e_1 \ldots e_n$ and $next\,[e_n] = q_f$. The probability of generating $(x, \mathbf{y})$ through the path π is:

$$I(prev\,[e_1]) \quad \Big(\xi(prev\,[e_1]\,, x_1, \mathbf{y}_1)\tau(prev\,[e_1]\,, next\,[e_1]) \ldots \\ \xi(prev\,[e_n]\,, x_n, \mathbf{y}_n)\tau(prev\,[e_n]\,, q_f)\Big).$$

For each path in T', we have exactly one path in M. Moreover, by construction:

$$\begin{aligned} I(prev\,[e_1])\xi(prev\,[e_1]\,, x_1, \mathbf{y}_1) &= \lambda(prev\,[e_1])\Big(\sum_{e \in \{e: e \in E[q], next[e] = q', i[e] \neq \sharp\}} prob\,[e]\Big) \\ &\quad \Big(\frac{prob\,[e_1]}{\sum_{e \in \{e: e \in E[q], next[e] = q', i[e] \neq \sharp\}} prob\,[e]}\Big) \\ &= \lambda(prev\,[e_1])prob\,[e_1]\,, \end{aligned}$$

where $q = prev\,[e_1]$ and $q' = next\,[e_1]$. Similarly, for $1 \leq i \leq n-1$,

$$\tau(prev\,[e_i]\,, next\,[e_i])\xi(prev\,[e_{i+1}]\,, x_{i+1}, \mathbf{y}_{i+1}) = prob\,[e_i]\,.$$

Finally,

$$\tau(prev\,[e_n]\,, q_f) = prob\,[e_{n+1}]\,.$$

Hence, $\mathcal{R}_{T'} = \mathcal{R}_M$ and using Lemma 15, $\mathcal{R}_T = \mathcal{R}_M$. The number of states in M is $1 + |E|$. According to Lemma 15, we can write, the number of states in M is $1 + \sum_{e \in E^T} max\{|i\,[e]|\,, |o\,[e]|\}$. Therefore, the theorem holds.

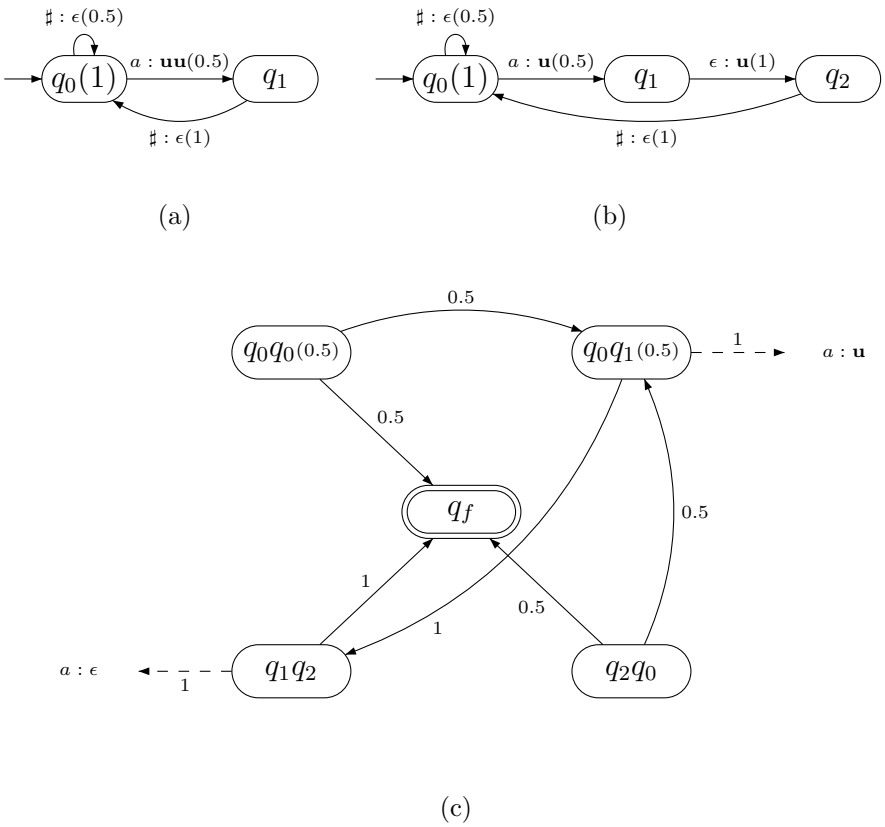


Figure 2.22.: The figure shows a PFST (Figure 2.22(a)), an equivalent intermediate PFST (Figure 2.22(b)), and an equivalent PHMM (Figure 2.22(c)).

Theorem 4 *Given a* PHMM M *with* n *states, there exists a* PFST T *with at most* n *states such that* $\mathcal{R}_M = \mathcal{R}_T$.

Proof We will show the proof of the theorem by construction. Let $M = \langle Q, \Sigma, \Omega, I, \delta, \xi \rangle$ be a PHMM. We will construct an equivalent PFST $T' = \langle Q', \Sigma \cup \{\sharp\}, \Omega, E, \lambda \rangle$ as follows:

- $Q' \leftarrow Q$;
- $\forall q \in Q \backslash \{q_f\}, \lambda(q) \leftarrow I(q)$;
- $\forall q, q' \in Q$ such that $\tau(q, q') \neq 0$, $\forall (q, a, \mathbf{b})$ such that $\xi(q, a, \mathbf{b}) \neq 0, a \in \Sigma \cup \{\epsilon\}, \mathbf{b} \in \Omega \cup \{\epsilon\}$,

- $E \leftarrow E \cup \{e : prob\,[e] = \tau(q, q')\xi(q, a, \mathbf{b}), i\,[e] = a, o\,[e] = \mathbf{b}, prev\,[e] = q, next\,[e] = q'\}$;

- $E \leftarrow E \cup \{e : prob\,[e] = 1, i\,[e] = \sharp, o\,[e] = \epsilon, prev\,[e] = q_f, next\,[e] = q_0$ where $q_0 \in I\}$;

For any pair of strings $(x, \mathbf{y}) \in \Sigma^* \times \Omega^*$ with $Pr_M(x, \mathbf{y}) \neq 0$ where $x = x_1 \cdot \ldots \cdot x_n$ and $\mathbf{y} = \mathbf{y}_1 \cdot \ldots \cdot \mathbf{y}_n$ such that $x_i \in \Sigma \cup \{\epsilon\}$ and $\mathbf{y}_i \in \Omega \cup \{\epsilon\}$, $\exists \pi \in \Pi_M(I)$ such that $\pi = e_1 \ldots e_n$ and $next\,[e_n] = q_f$. The probability of generating $(x, \mathbf{y})$ via π is:

$$I(prev\,[e_1]) \quad \Big(\xi(prev\,[e_1], x_1, \mathbf{y}_1)\tau(prev\,[e_1], next\,[e_1]) \ldots \\ \xi(prev\,[e_n], x_n, \mathbf{y}_n)\tau(prev\,[e_n], q_f)\Big).$$

Similarly, for any pair of strings $(x, \mathbf{y}) \in \Sigma^* \times \Omega^*$ with $Pr_{T'}(x, \mathbf{y}) \neq 0$, $\exists \pi \in \Pi_{T'}(I)$ such that $\pi = e_1 \ldots e_{n+1}$ and $o\,[e_{n+1}] = \sharp$. The probability of generating $(x, \mathbf{y})$ via π is:

$$\lambda(prev\,[e_1]) \quad \Big(prob\,[e_1] \ldots prob\,[e_{n+1}]\Big).$$

By construction, $I(q) = \lambda(q)$, $prob\,[e_i] = \xi(prev\,[e_i], x_i, \mathbf{y}_1)\tau(prev\,[e_i], next\,[e_i])$, and $prob\,[e_{n+1}] = 1$. Hence, $\mathcal{R}_M = \mathcal{R}'_T$. Moreover, we can build a PFST T with at most $|Q| = n$ states such that $T = T'$. □

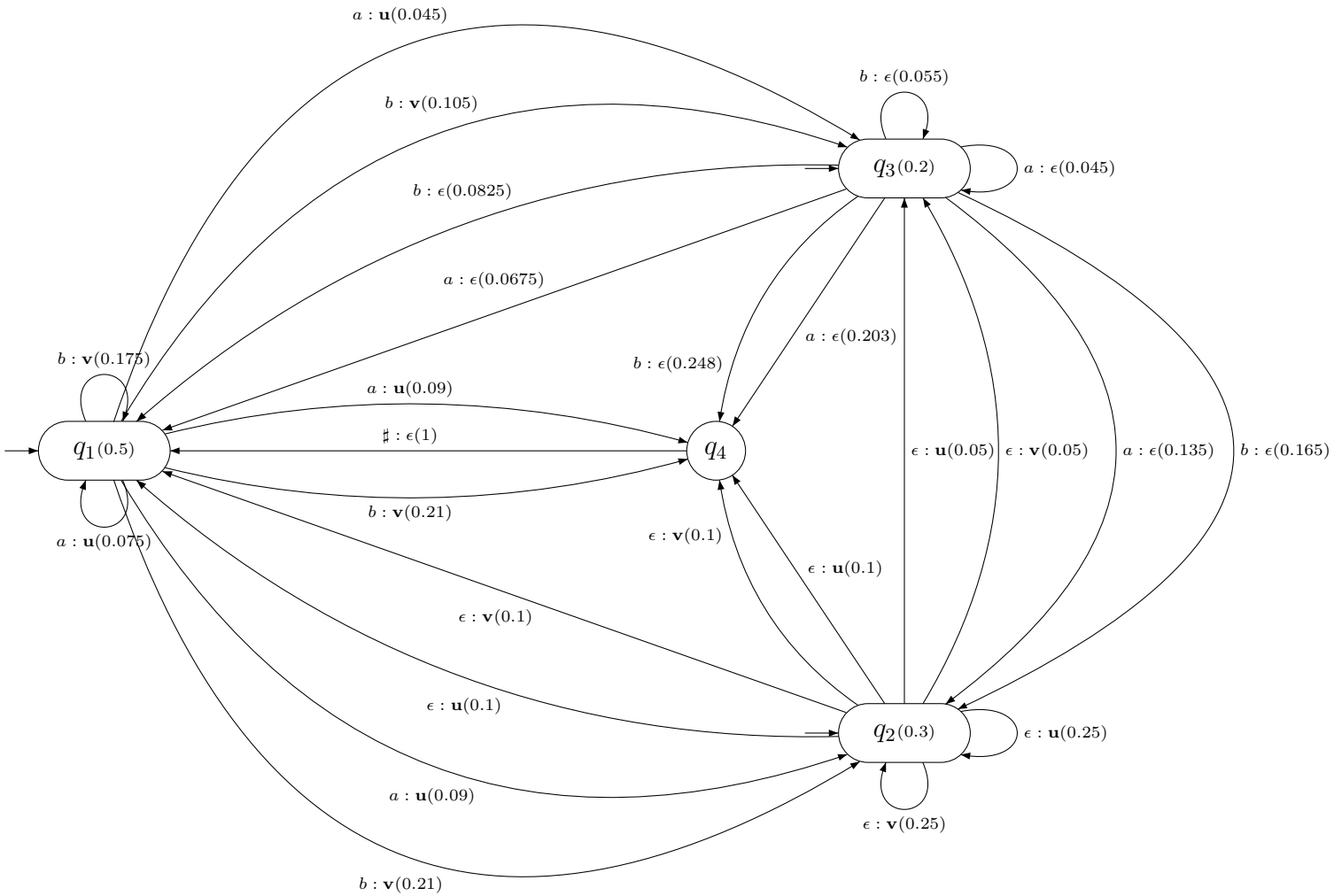


Figure 2.23.: An equivalent PFST of the PHMM given in Figure 2.21.

2.12.2. Weighted Finite State Transducers

So far we have talked about transducers over the boolean semiring or the probability semiring. A generalized formalism of finite state transducers defined under an arbitrary semiring is known as *weighted finite state transducers* (WFSTs). In this section we formulate the definition of WFST defined over an arbitrary semiring. The definitions presented in this section are based on (Roark and Sproat, 2007; Allauzen et al., 2007; Mohri, 2005).

Definition 18 *A* ***Weighted Finite State Transducer*** *is a sextuple* $T = \langle Q, \Sigma, \Omega, E, \lambda, \rho \rangle$ *over a semiring* $\mathbb{K}$ *where:*

- Q *is a finite set of states,*
- Σ *and* Ω *are the input and the output alphabets,*
- E *is a finite set of transitions* $E \subseteq Q \times \Sigma \cup \{\epsilon\} \times \Omega^* \times \mathbb{K} \times Q$*,*
- λ *and* ρ *are the initial and the final weight functions respectively.*

The set of initial states and final states in a WFST are defined as $I = \{q \in Q : \lambda(q) \neq \overline{0}\}$ and $F = \{q \in Q : \rho(q) \neq \overline{0}\}$.

Proposition 5 *A* WFST *is equivalent to a* PFST *under the following conditions:*

1. *The* WFST *is defined under probability semiring.*
2. *The following normalization condition holds:*

$$\bigoplus_{q \in Q} \lambda(q) = \overline{1},$$

$$\forall q \in Q, \rho(q) \oplus \bigoplus_{e \in E[q]} prob\,[e] = \overline{1}.$$

Proof The proof follows from the definitions of WFST and PFST.

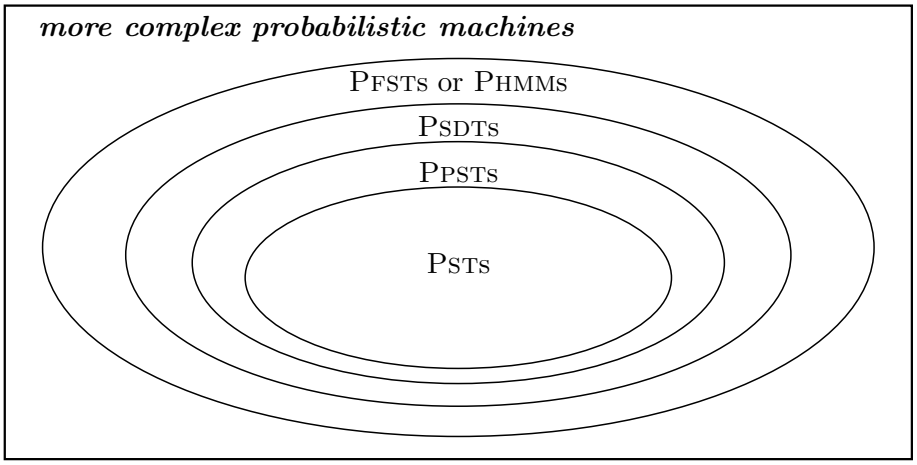


Figure 2.24.: A Venn diagram of the expressive power of probabilistic finite state machines.

2.13. Chapter Summary

In this chapter we have attempted to pursue three of the research questions defined in Chapter 1, referred as the **RQ**1, the **RQ**2, and the **RQ**3. Here we will summarize the results presented in this chapter under each of these research questions.

The **RQ**1 is regarding the investigation of the hierarchies of the stochastic bi-languages represented by finite state machines. We have conducted the study in two steps: first, we have shown the hierarchy of bi-languages in a non-probabilistic setting. In order to elaborate the hierarchy of bi-languages, we have presented the definitions and properties of several types of rational transducers. Second, we have shown the hierarchy of stochastic bi-languages represented by finite state machines. We have also presented the definitions and properties of different types of probabilistic transducers. Graphical views of our results are presented in Figure 2.11 and 2.20. Moreover, Figure 2.24 depicts the hierarchy in terms of the syntactic machines for the respective sets of the stochastic bi-languages.

The **RQ**2 is about parsing using probabilistic transducers. The objective of this research question is to find out the computational complexities for the different types of probabilistic transducers. We have presented five different types of computation problems and how they can be solved using four different types of probabilistic transducers defined in Section 2.9. The summary of our findings is presented in Table 2.2 (page 51).

Finally, the **RQ**3 is regarding comparison of probabilistic transducers with other existing finite state models. First, we have conducted a study *w.r.t.* a well known finite state model called the PHMM. We have proved that PFSTs and PHMMs are equivalent in terms of modeling joint distributions over translation pairs of input and output languages. Otherwise stated, stochastic bi-languages modeled by PFSTs can also be modeled by PHMMs and vice versa. Second, we have shown that, PFSTs are essentially equivalent to another well known model called WFST when they are defined under the probability semiring.

2.14. Discussion

The three research questions tackled in this chapter give us important and useful insights for the rest of the thesis work. The answer to the **RQ**1 tells us that the PSTs are the least powerful in terms of the stochastic bi-languages that the PSTs represent. The natural question that one may ask at this point is: if PSTs are the least powerful, why are we learning them? If we analyze the studies *w.r.t.* the **RQ**2 and the **RQ**3, we will be able to give a justified answer to this question.

The results of the **RQ**2 show, in terms of computational complexities, PSTs are the most efficient devices (Table 2.2). Otherwise stated, using the other types of machines will be more computationally complex than the PSTs.

Moreover, if we analyze the study *w.r.t.* the **RQ**3 we have the following: we have shown that the PFSTs are equivalent to another well known model called the PHMM. As the set of all the PFSTs is essentially a superset of the set of all the PSTs, it logically follows that the PSTs can also be converted to its equivalent PHMM, which is a widely used model.

Therefore, the justification to learn PSTs are: 1) inexpensive to perform computation and 2) can be converted to other widely used models.

Chapter 3

Related Work

3.1. Introduction

The history of learning finite state transducers is related to the history of learning finite state automata. The problem of learning finite state automata was first addressed by Gold (1978), where he showed an algorithm to learn DFAs in an identification in the limit model (Gold, 1967). Gold proposed an algorithm for learning DFA that converges to the minimal form of the target DFA from a complete presentation. However, the limitation of Gold's algorithm was that the inferred DFA is guaranteed to be interesting only when a characteristic sample is available. If the hypothesis DFA is not consistent to the training data, the algorithm returns the initial hypothesis DFA, which is a tree like DFA representing the training data (Gold, 1978). In 1992, Oncina and García proposed the RPNI (Regular Positive and Negative Induction) algorithm which provides guarantee to be consistent with the training data. If the training data includes the characteristic sample, the algorithm converges to the minimal DFA for the target language.

The problem of learning subsequential transducers from a given set of positive examples in the limit was first addressed by Oncina *et al.* in their seminal papers (Oncina and García, 1991; Oncina et al., 1993) where they presented their now well known OSTIA algorithm, which is based on the state merging strategies similar to those used in the DFA learning algorithms. Before the work of Oncina *et al.*, the problem of transducer learning had rarely been addressed. Only a handful quantity of work, mostly based on heuristics had been carried out such as (Veelenturf, 1978; Luneau et al., 1983; Takada, 1988; Vidal et al., 1990).

Besides state merging approaches, there has been work done with regards to transducer learning in an active learning setting (Vilar, 1996; Oncina, 2008)

which are also inspired by Angluin's DFA learning algorithm by means of queries (1987a).

In this chapter we will present a survey on transducer learning. First, we will briefly highlight the work related to transducer learning before OSTIA was proposed and their limitations. Second, we will describe the basic principles of the OSTIA algorithm. Third, we will discuss other approaches which are primarily based on OSTIA with some added features or heuristics. Fourth, we will concisely present a sketch on how transducer learning evolved in the active learning setting. Finally, we will discuss a few recent works on transducer learning.

3.2. Early Work

One of the early work with regards to transducer learning is the attempt from Veelenturf (1978). In his paper, he proposed a non-incremental algorithm for learning Mealy machines. A Mealy machine is essentially a sequential transducer where the transition outputs can only be symbols (Mealy, 1955). The algorithm obtains a Mealy machine which is compatible to the training data. If the target transducer is an n-state Mealy machine and the training data contains all the input-output pairs with lengths below $2n-1$, then the algorithm is guaranteed to infer an exactly equivalent machine to the target transducer. Which means, for an arbitrary number of input symbols $|\Sigma|$, and an arbitrary number of states n, the algorithm will require exactly the following number of examples:

$$|\Sigma|^{(2n-1)} + |\Sigma|^{(2(n-1)-1)} + \ldots + |\Sigma|.$$

This number is exponential *w.r.t.* the number of states n.

Another related work is the result presented by Luneau *et al.* in (1983). In this paper they presented a heuristic based incremental algorithm to infer Moore machines. A Moore machine is a finite state machine where the output are only determined by the current states (Moore, 1956). Unlike Mealy machines, Moore machines have no transition outputs. The main drawback of their work is: it is not guaranteed that the inferred machine will be consistent to the training data.

In another early attempt, Takada worked on transducer inference problem in (1988). In his work he presented an inference algorithm for learning DFAs extended to have an output transition. Essentially, the algorithm learns a sequential transducer which is allowed to have only one symbol as an output. This model is a highly restricted model in the sense of being deterministic *w.r.t.* the inputs and having the prefix preserving property (see Chapter 2, Section 2.4.2).

Yet another early attempt is the work done by Vidal *et al.* on inference of transducer presented in (1990). They conducted this work within the context of

pattern recognition tasks and it was based on heuristics. No theoretical guarantee of identification in the limit was given.

The early work with regards to transducer learning we have mentioned in this section are essentially the work done before the OSTIA algorithm was proposed. From this brief discussion it appears that before OSTIA, the work was restricted to sequential models, based on heuristics and had lack of theoretical guarantee for identification in the limit. Next, we explain the OSTIA approach for learning subsequential transducers.

3.3. OSTIA: The State Merging Approach

The OSTIA (Onward Subsequential Transducer Inference Algorithm) (Oncina and García, 1991) algorithm can be found in three different versions in the literature having the basic principle and the result being the same, formalized using a slightly different data structure and combinatorics. Oncina *et al.* showed the basic version of OSTIA in (Oncina et al., 1993) where they basically worked with the sequential representation of the subsequential transducer to be learnt. In (Castellanos et al., 1998) Oncina *et al.* have presented the same algorithm, however, working with subsequential transducers rather than working with sequential representation of it. De la Higuera showed another presentation of the same algorithm in (de la Higuera, 2010) where a recursive framework for state merging is adopted. All these versions of OSTIA essentially perform the same task with slightly different formalisms and implementations. Here, we explain the basic principle of the algorithm which is fundamentally the same in all its versions.

As mentioned previously, the problem of identification of subsequential transducers from a given set of transduction examples in the limit is inspired by the state merging algorithm for identification of DFA in the limit (Oncina and García, 1992). The RPNI algorithm by Oncina and García presented in (1992) is the first algorithm where the state merging approach is introduced for learning DFA. The algorithm is initialized by building a *prefix tree acceptor* (PTA) from the training data labeled as positive, and state merging is iteratively applied making sure that the hypothesis DFA is consistent with the training sample. This is achieved by rejecting a merge between two candidate states whenever a negative example is accepted from the given sample. Therefore, the RPNI algorithm requires positive and negative data for identifying the DFA in the limit. Based on modified state merging strategies, a number of variants of DFA learning algorithms have been proposed (Trakhtenbrot and Barzdin, 1973; Lang et al., 1998; García et al., 2010; Heule and Verwer, 2010).

OSTIA is very much analogous to RPNI; OSTIA is also based on state merging approaches similar to the DFA learning algorithms mentioned, except that it infers subsequential transducers. Unlike RPNI, OSTIA is only given positive examples. Therefore, the state merging strategy in OSTIA has to rely on two properties to compensate the lack of negative data; one of them is ensuring the subsequentiality (see page 23) of the hypothesis and the other is making sure the consistency of the outputs with regards to the given data.

The OSTIA algorithm essentially executes in three phases. Here we describe the three phases in details:

Initialization: Similar to RPNI, OSTIA also starts with a tree like transducer built from the training pairs, known as the *tree subsequential transducer* (TST), which exactly represents translations given by the training pairs. Basically a tree is built from the prefix set of the input strings of the given sample. The outputs of the transitions are labeled as ϵ. The output strings in the given sample are then assigned to the leaf states of the tree corresponding the respective input strings as the state outputs. Thus, the TST only produces the translations given in the sample.

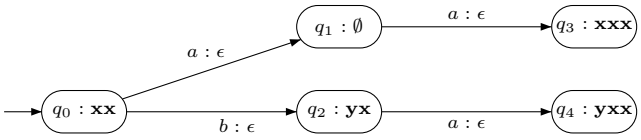


Figure 3.1.: A TST built from the positive sample $S = \{(\epsilon, \mathbf{xx}), (b, \mathbf{yx}), (aa, \mathbf{xxx}), (ba, \mathbf{yxx})\}$.

Onwarding: The TST is then converted to an *onward* (Oncina and García, 1991; Oncina et al., 1993; Castellanos et al., 1998) form, known as the *onward tree subsequential transducer* (OTST). Intuitively, a subsequential transducer in the onward form means, the output strings are always as close as possible to the initial state. Technically, this is done by advancing the longest common prefixes of the output strings of the edges starting from the leaves of the tree. This is done level by level from the bottom of the tree to the initial state. Note that since only the longest common prefixes are being moved toward the initial state, the onward tree remains fully consistent with the given data. In other words,

the initial TST and the OTST produce exactly the same translations. Till this phase of the algorithm no generalization takes place.

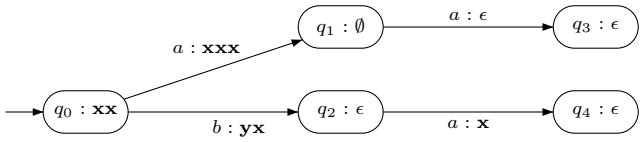


Figure 3.2.: Making the TST given in Figure 3.1 onward.

State Merging: The state merging phase of OSTIA is the phase where the algorithm attempts to generalize by state merging. Given an OTST T, the OSTIA algorithm iteratively merges the compatible states, *i.e.*, it merges the states having identical state outputs or if the output of one of the candidate states is still unknown. Formally, the merge acceptance conditions for the two candidate states q and q' can be expressed as:

$$\big(\sigma(q) = \emptyset\big) \vee \big(\sigma(q') = \emptyset\big) \vee \big(\sigma(q) = \sigma(q')\big).$$

If the subsequential condition (see Chapter 2, Section 2.4.3, Definition 9) is violated, OSTIA performs a cascade of forced merges until the OTST is reached in a subsequential form.

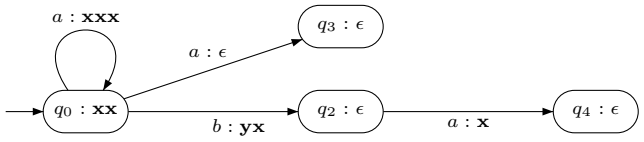


Figure 3.3.: Example of a merge attempt between q_0 and q_1. This merge is rejected since the merge acceptance condition is violated.

Sometimes due to the onwarding process, some outputs are too close to the initial state and therefore merging is not possible due to the subsequentiality condition. In order to make merges to happen in such cases, whenever necessary, OSTIA pushes back (reverse of making onward) suffixes of the output strings toward the leaves of T; this is known as the *pushback* operation. Intuitively,

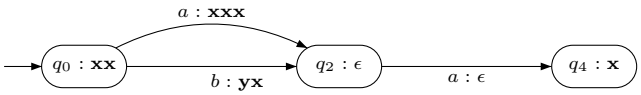


Figure 3.4.: Example of a merge attempt between q_1 and q_2 which is accepted. To preserve subsequential property q_3 is merged to q_4. For this merge to happen, pushback takes place which pushes the output $\mathbf{x}$ toward the leaf q_4.

pushback is a process of propagating the outputs toward the leaves of the Tst (see Figure). The process of iterative state merging is repeated until no more states are mergeable.

Notice that throughout the state merging phase the algorithm makes sure that the hypothesis transducer is consistent with the training data. The Ostia algorithm has been proven to identify the subsequential transducer in the limit from its characteristic sample if the corresponding subsequential function is *total* (Oncina and García, 1991; Oncina et al., 1993). More details about Ostia *e.g.*, an example run, formal proofs, experimental results *etc.* can be found in (Oncina and García, 1991; Oncina et al., 1993; Castellanos et al., 1998; de la Higuera, 2010).

3.4. Variants of OSTIA

Different variants of the Ostia algorithm have been proposed ever since the algorithm was published. Essentially, the variants are successors of the Ostia algorithm with some added features incorporated to the basic version of Ostia. One of the objectives of modifying Ostia is to gain better prediction accuracy. Moreover, another objective is: with help of additional information, to be able to learn subsequential functions which are *partial functions*, while Ostia can only learn subsequential functions which are *total functions*.

One of such variants is the Ostia-N algorithm (Oncina and Varó, 1996). In this algorithm, negative information with regards to the domain language is used for making merge decisions. A negative example *w.r.t.* the domain language means a string that is rejected by the automaton that represents the domain language. Ostia-N requires positive presentation of input-output pairs similar to Ostia and additionally, it requires negative examples of the input language

or the domain language. For taking a merge decision of a candidate pair of states, OSTIA-N makes supplementary checks as to whether a negative instance of the domain language is accepted or not. The merge is rejected if it accepts a negative instance. The prediction accuracy of OSTIA-N has been reported to be better than OSTIA; this is due to the fact that OSTIA-N has got access to more information than OSTIA. Identification in the limit is guaranteed whenever a characteristic sample is included in the training data. OSTIA-N can learn subsequential functions which are partial functions.

Another successor of OSTIA is the OSTIA-D algorithm (Oncina and Varó, 1996). This algorithm relies on the information with regards to the domain language. Here the motivation is to utilize domain specific knowledge to accomplish the transducer learning task. In various applications, it often the case that domain specific information is available and using OSTIA-D methodology, such information can be used to gain better prediction accuracy. Similar to OSTIA, a positive sample is presented and additionally it is assumed that the learner has knowledge about the domain language in terms of the syntactic machine that represents the domain language. Since the bi-language is regular, the syntactic machine in case of the domain language is a DFA. The initial TST is augmented with such domain information. The distinct states of the DFA representing the input language are labeled in the TST and only the states having the same labels are allowed to be merged. This approach has been shown to improve the prediction accuracy drastically in comparison to the basic OSTIA algorithm. Again, the reason for better prediction accuracy is OSTIA-D having access to more information than OSTIA. OSTIA-D also guarantees identification in the limit if characteristic sample is included in the training data. OSTIA-D can also learn subsequential functions which are partial functions.

3.5. Heuristics Based Approaches

Although OSTIA provides formal guarantee for asymptotic convergence, experiments showed that for complex tasks the amount of required data is large enough to be prohibitive. Based on the basic version of OSTIA, several attempts have been made to achieve better prediction accuracy with lesser amount of data by applying some heuristics. In this section, we discuss some of the heuristics based successors of OSTIA.

Oncina presented a heuristics based modification of OSTIA known as OSTIA-DD where a data driven approach has been applied (1998). The basic OSTIA algorithm follows a greedy approach in the sense that the first valid merge computed is taken into consideration. There is a risk of a wrong merge getting accepted

due to lack of enough training data in this approach. The idea is to avoid wrong merges by giving score to number of merges and considering the best merge from a set of valid merges. This approach has been inspired by the data driven approach from de la Higuera *et al.* (1996). The metric used for scoring the merges in OSTIA-DD is chosen on the basis of reduction of the number of output symbols in the tree as a result of a merge. Experiments reported in (Oncina, 1998) showed significant improvement in prediction accuracy of OSTIA-DD in comparison with OSTIA.

In the context of a machine translation task, Vilar used dictionaries and alignments along with OSTIA to improve prediction rate in (2000). One of the demanding criteria for OSTIA to infer the correct machine is the requirement of having enough examples so that the correct alignment of the output strings *w.r.t.* the input symbols can take place. If the alignments are precomputed by means of some known heuristics from statistical machine translation, the amount of data required reduces drastically to obtain an acceptable prediction accuracy. This algorithm is known as the OMEGA (OSTIA Modified for Guarantees and Alignments) algorithm. The OMEGA algorithm has been reported to outperform the OSTIA and the OSTIA-D algorithms.

In the context of the Tenjinno[1] competition (Starkie et al., 2006), Clark proposed a heuristics based algorithm in (2006). This algorithm turned out to be the winner of the competition. The main idea of the algorithm is to combine DFA learning by means of state merging, expectation maximization algorithm for alignment and the idea of state splitting. The algorithm infers the DFA representing the input language by means of some heuristics based state merging. Then alignment of the outputs *w.r.t.* the input symbols is carried out by an expectation maximization algorithm. The inferred DFA is then augmented by the aligned outputs *w.r.t.* the input symbols. Whenever necessary state splitting is done on the DFA to incorporated the aligned outputs.

Besides the community of grammatical inference, the problem of learning finite state transducers has also been explored by the *evolutionary computing* community. Evolutionary algorithms appeared to be robust in presence of noise in the context of DFA learning (Dupont, 1994; Lucas and Reynolds, 2003, 2005). Lucas and Reynolds worked on learning FSTs using an evolutionary algorithm in (2003; 2007). Evolutionary approaches to learn transducers also showed to exhibit better prediction accuracy in certain tasks in comparison to the OSTIA algorithm. However, these kinds of approaches provide no guarantee of learning the target machine.

Moreover, heuristics have been applied to adopt FST learning to machine trans-

[1]Tenjinno is a machine translation competition held in 2006.

lation (Casacuberta, 2001; Picó and Casacuberta, 2001; Vidal and Casacuberta, 2004), natural language processing (Casacuberta, 2000; González et al., 2008), speech processing (Vilar et al., 1996). All these approaches use some domain specific heuristics to adjust the learning task to their specific needs.

3.6. Active Learning

So far we have discussed learning settings where the learner is given examples to accomplish the learning tasks. The learner makes no interaction with the learning environments. As discussed in Chapter 1, active learning settings provide an environment for the learner to interact with the learning environment. The learner has access to a *minimally adequate teacher* (MAT) or an *oracle* to ask queries with regards to examples. For instance, the learner may ask the oracle if a string belongs to a language or not, and the oracle answers with either *yes* or *no*. The learning model from queries is due to Angluin; in her seminal paper (1987b) she introduced such learning model. In another paper, she presented her famous $L*$ algorithm (Angluin, 1987a): an algorithm for DFA learning by queries.

Transducer learning in an active learning setting was first addressed by Vilar (1996). The algorithm he presented in this paper is essentially an extension of Angluin's $L*$ algorithm to learn subsequential transducers. Vilar's algorithm uses two kinds of queries: translation queries and equivalence queries. In a translation query, the learner is allowed to ask the translation of a particular string and the oracle answers with its translation. If the translation is not defined, the oracle returns a signal indicating that the translation does not exist. In an equivalence query the learner can as the oracle if a hypothesis is equivalent to the target and the oracle answers with either *yes* or *no*. The learning algorithm halts when the hypothesis is equivalent to the target.

Another work done in Angluin's active learning model to infer transducers is the relatively recent work from Oncina (2008). Oncina presented an algorithm for learning multiplicity automaton to represent transducer relations. Multiplicity automata are devices that can implement functions from strings to *fields*. Oncina formalized a restriction to represent string to string relations instead of string to field relations; multiplicity automata were used to represent string to *divisive ring* relations. He showed an algorithm for learning such devices in Angluin's exact learning model. The algorithm used the same set of queries that was used in the Vilar's algorithm. Oncina mentioned in his paper that since the proposed algorithm works only in theoretical settings, a number of open questions remained to be answered and the practical implications of the proposed algorithm are still

to be investigated.

3.7. Learning Probabilistic Models

As far as learning probabilistic transducers are concerned, to the best of our knowledge there has been no work done till date where learning the structure of the machine is a primary objective. It is worth mentioning that there has been work done on probabilistic transducer models such as (Clark, 2001a, 2002; Eisner, 2001, 2002; Li and Eisner, 2009; Balle et al., 2011), primarily focused on training the weights or probabilities based on the observed data of a predefined structure of a transducer. This line of research is fundamentally different in the context of the thesis since our objective is to infer the structure of the target machine as well as the underlying probability distribution represented by the target machine.

However, algorithms exist for learning Probabilistic Finite-State Automata (Pfa). The learning problem of Pfa from positive sample is a well explored problem. The first state merging algorithm for learning deterministic Pfa from positive examples, Alergia, was introduced by Carrasco and Oncina in (1994). Having Alergia as the pioneer, a number of modified versions has been presented, *e.g.*, (Thollard et al., 2000; Carrasco and Oncina, 1999). They are state merging algorithms similar to Rpni based on statistical tests to preserve the property of the distribution of the training data. A recent survey about Pfa learning can be found in (Verwer et al., 2012).

3.8. Recent Work

Wakatsuki and Tomita presented a polynomial time algorithm for learning *strict prefix deterministic finite state transducers* from positive presentation (2010). Strict prefix deterministic finite state transducers are proper subclass of rational transducers. However, it has not been reported in their paper if strict prefix deterministic finite state transducers properly include all subsequential transducers. They proved their algorithm in Gold's identification in the limit paradigm.

In another recent work, Clark presented an algorithm for learning inversion transduction grammar in an identification in the limit model (2011). Inversion transduction grammar is a set of bi-languages that are context free bi-languages and more powerful than rational bi-languages. Hence, it properly includes all regular bi-languages recognized by subsequential transducers. The algorithm

presented by Clark is generalization of distributional learning model presented in (Clark, 2010). In his paper he reports that the theoretical results that he presented are far from actually being applied in practice, and there was no experimental results reported.

3.9. Chapter Summary

In this chapter we have presented a bibliographical study on transducer learning. We started by discussing about the early attempts of transducer learning starting back in the late seventies where the models were restricted to sequential transducers. As the pioneering work for subsequential transducer learning and also as the baseline of the thesis work, we have explained the fundamentals of the OSTIA algorithm. We also presented the ideas of several variants of OSTIA. Results with regards to the learning problem of transducers in the active learning settings have also been discussed. From the study it evidently appears that the problem of learning probabilistic transducers is still an open problem. Finally, we also presented some recent work that endeavors the learning problem of more complex classes than regular bi-languages.

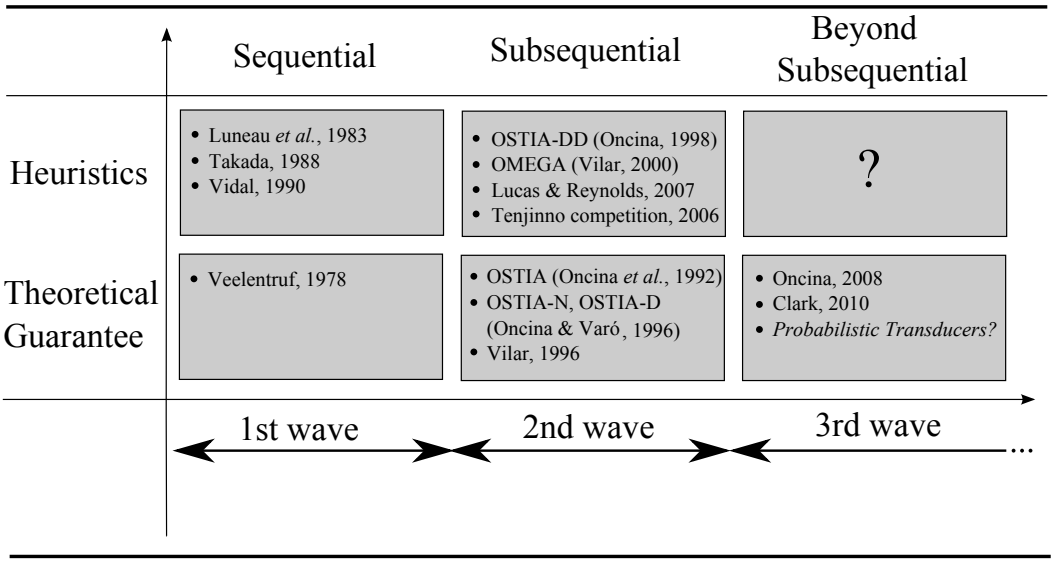


Figure 3.5.: Development of transducer learning.

3.10. Discussion

The history of transducer learning can be categorized into three waves in terms of the expressive power of the learning model. The first wave was clearly restricted to a sequential model; the work done before OSTIA was proposed, falls under this category. The second wave is the post OSTIA wave, where the learning model is subsequential. Most of the work done within this wave is based on OSTIA. The third wave includes the attempts to go beyond subsequential model and trying to learn more complex classes. For instance, the work that falls under this category includes Oncina (2008) and the work of Clark (2011).

Another way of categorizing the development of transducer learning is from the view point of the theoretical and pragmatic approaches. We clearly see that there has been work done which is backed up by some sort of theoretical guarantee and at the same time work that is solely based on heuristics and engineering. It is interesting to see that developments in these two categories have been parallel. However, the number of work done with formal guarantee is noticeably fewer that heuristics attempts.

In Figure 3.5, a visual representation of the development of transducer learning is presented. The horizontal axis of the figure shows the three waves in terms of the complexity of the learning models. The vertical axis of the figure categorizes the developments of transducer learning into the work with theoretical guarantee and the work based on heuristics. The first wave started in the late seventies of the last century and continued till the early nineties. In the early nineties, OSTIA was introduced as a stepping stone. In Figure 3.5, we can see that the second wave of transducer learning has been so far the biggest wave and most of the work done within this wave are centered around OSTIA as the baseline. The second wave continued till the middle of the first decade of this century. It was clear by then that in order to make transducer learning more practicable, there is a need of learning more powerful models and learning probabilistic transducers. The third wave started with Oncina's attempt in (2008) and continues till today (*e.g.*, (Clark, 2011)).

The bibliographical study shows that the problem of learning probabilistic subsequential transducer is evidently an open problem. There is a need of providing theoretical guarantee of learning in the limit for probabilistic transducers. At the same time the theoretical assumptions should be realistic so that the approaches are practicable.

Chapter 4

Identification of Probabilistic Subsequential Transducers in the Limit

4.1. Introduction

In Chapter 1 we have stated learnability of PSTs as one of our research questions (**RQ**4) of the thesis. In this chapter, in the context of **RQ**4, we investigate the problem of learning PSTs in an active learning setting. As a result of **RQ**3, where we have shown equivalence of bi-languages modeled by other widely used syntactic machines (Chapter 2), the outcome of **RQ**4 can also be applied to learn other models such as PHMMs.

In the active learning paradigm used for language learning tasks, the learner has access to an *oracle* or a *minimally adequate teacher* (MAT) (which can in practice be a corpus, a human expert or the Web) and the learner is able to interact with this oracle. The learner generates strings and queries the oracle about the strings (Angluin and Smith, 1983; Angluin, 1987a). In the field of grammatical inference, traditionally, the learner generates the data it needs and makes a membership query, asking if the string is or isn't in the language: the learner asks queries about data which has not been observed. In practice, querying about unseen data or data artificially generated by the learner can sometimes lead to problems: the oracle may find it difficult to classify this *nonsense* data. This has been described and analyzed in (Lang and Baum, 1992); a recent survey in active learning and a discussion can be found in (Settles, 2011).

To mitigate such practical issues, as conversed to the $L*$ approach (Angluin, 1987a) where the learner can ask questions about any data, in this chapter we work on a scenario where positive data is given to the learner and the learner is only allowed to ask queries about the observed data. Furthermore, we make

use of the extra information that exists when the data has been randomly drawn following an unknown distribution, distribution itself generated by a finite state machine. We present a novel learning algorithm for learning probabilistic subsequential transducers (PSTs), which are deterministic transducers with probabilities. The algorithm learns from positive examples by making probabilistic queries only regarding the data present in the training sample.

The new proposed algorithm is essentially the hybridization of the state merging and active learning paradigms. In our proposed algorithm we build a tree transducer from the observed positive data, which is an exact representation of the training data and ask probabilistic queries only regarding the observed data. In our algorithm, instead of asking queries about the data that are not present in the training set, we utilize the lack of information to make state merging decisions. This brings an improvement over the OSTIA described in Chapter 3: while OSTIA can only learn transduction schemes which are *total functions*, the proposed algorithm is also capable of learning transduction schemes which are *partial functions*. We prove the correctness of our algorithm in an identification in the limit model. Moreover, we report experimental evidence that shows that our algorithm converges with relatively few training examples and produces an acceptable translation accuracy.

4.2. A Canonical Normal Form

In this section, we recall the definition of PSTs presented in Chapter 2. The objective of this section is to define a canonical normal form of PSTs. The canonical normal form of PSTs defined here will be used as the normal form of the target machine in our learning algorithm.

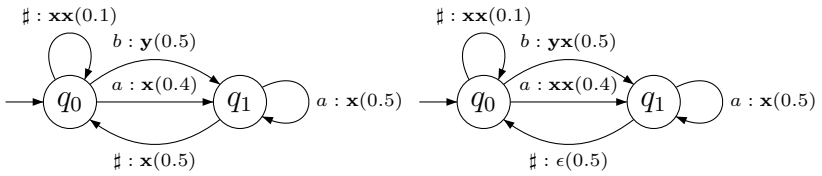
At this point we introduce the concept of *onward* (Oncina et al., 1993) PSTs, which is required to define the minimal canonical form of PST that our learning algorithm infers.

Definition 19 *A* PST $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E\rangle$ *is said to be* in onward form *if the following property holds:*

$$\forall q \in Q \backslash \{q_0\}, lcp\left(\bigcup_{e \in E[q]} \{o\,[e]\}\right) = \epsilon.$$

The onward form makes sure that a translation is given by the PST as early as possible.

Let $\mathcal{R}$ is a stochastic regular bi-language, *i.e.*, $\mathcal{R} \in \boldsymbol{\mathcal{SREGBL}(\Sigma, \Omega)}$. Therefore, $\mathcal{R}$ can be modeled by a PST (see Chapter 2, Section 2.9.4).



(a) (b)

Figure 4.1.: Figure 4.1(a) shows a PST which is not in the canonic normal form. Figure 4.1(b) is the equivalent PST in the canonic normal form (onward). The labels of the edges should be interpreted as *input*:**output**(probability).

The quotient $(u, \mathbf{v})^{-1}\mathcal{R}$ where $u \in \Sigma^*$ and $\mathbf{v} \in \Omega^*$ is the stochastic set that obeys the following properties:

1. $Pr_{(u,\mathbf{v})^{-1}\mathcal{R}}(w\sharp, \mathbf{w}') = \frac{Pr_{\mathcal{R}}(uw\sharp, \mathbf{v}\mathbf{w}')}{Pr_{\mathcal{R}}(u\Sigma^*\sharp, \mathbf{v}\Omega^*)}$,
2. $(u, \mathbf{v})^{-1}\mathcal{R} = \{(w, \mathbf{w}')|(uw, \mathbf{v}\mathbf{w}') \in \mathcal{R}, \mathbf{v} = lcp(\{\mathbf{q}|(u\Sigma^*\sharp, \mathbf{q}) \in \mathcal{R}\})\}$.

If $Pr(u\Sigma^*\sharp, \mathbf{v}\Omega^*) = 0$, then by convention $(u, \mathbf{v})^{-1}\mathcal{R} = \emptyset$ and $Pr_{(u,\mathbf{v})^{-1}\mathcal{R}}(w\sharp, \mathbf{w}') = 0$. If $\mathcal{R}$ is SDRT, the number of different stochastic sets of $(u, \mathbf{v})^{-1}\mathcal{R}$ is finite. For example, there are two such nonempty stochastic sets for the example given in Figure 4.1(b) and they are:

- $(\epsilon, \epsilon)^{-1}\mathcal{R} = \{(\sharp, \mathbf{xx})\} \cup \{(a^n\sharp, \mathbf{x}^{n+1})|n \geq 1\} \cup \{(ba^n\sharp, \mathbf{yx}^n)|n \geq 1\}$,
- for $m \geq 1$, $(a^m, \mathbf{x}^{m+1})^{-1}\mathcal{R} = (ba^m, \mathbf{yx}^m)^{-1}\mathcal{R} = \{(\sharp, \mathbf{x})\} \cup \{(a^n\sharp, \mathbf{x}^{n+1})|n \geq 1\}$.

We construct the minimal canonical PST $M = \langle Q^M, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}^M, E^M \rangle$ in onward form as the following:

$$
\begin{array}{ll}
Q^M & = \{(u, \mathbf{v})^{-1}\mathcal{R} \neq \emptyset, u \in \Sigma^*, \mathbf{v} \in \Omega^*\} \\
\{q_0\}^M & = \{(\epsilon, \mathbf{v})^{-1}\mathcal{R}\} \\
E^M & = \{(q, a, \mathbf{w}, \alpha, q')| \\
 & \quad q = (u, \mathbf{v})^{-1}\mathcal{R} \in Q^M, \\
 & \quad q' = (ua, \mathbf{v}\mathbf{v}')^{-1}\mathcal{R} \in Q^M \text{ where}, \\
 & \qquad a \in \Sigma \cup \{\sharp\}, \mathbf{v}' \in \Omega^*, \\
 & \quad \mathbf{w} = lcp(\{\mathbf{v}|(u\Sigma^*, \mathbf{v}\Omega^*) \in \mathcal{R}\})^{-1} \\
 & \qquad lcp(\{\mathbf{v}\mathbf{v}'|(ua\Sigma^*, \mathbf{v}\mathbf{v}'\Omega^*) \in \mathcal{R}\}), \\
 & \quad \alpha = \frac{Pr_{\mathcal{R}}(ua\Sigma^*, \mathbf{v}\mathbf{v}'\Omega^*)}{Pr_{\mathcal{R}}(u\Sigma^*, \mathbf{v}\Omega^*)}\}
\end{array}
$$

The canonical PST generates stochastic bi-languages that are in $\mathcal{R}$ and have non-zero probabilities. There have been extensions of the Myhill-Nerode theorem (Hopcroft et al., 2006) presented in (Carrasco and Oncina, 1999; Oncina and García, 1991) and based on the extended theorems we can say that the canonical PST is the minimal form and unique upto state isomorphism.

4.3. The Stochastic Sample

When learning, the algorithm will be given a randomly drawn sample: the pairs of strings will be drawn following the joint distribution defined by the target PST. Therefore, such a sample is a multiset, since more frequent translation pairs may occur more than once and is called a *stochastic sample*. The formal definition of s stochastic sample is the following:

Definition 20 *A* ***stochastic sample*** *is a multiset* $S_n \langle X, f \rangle$ *where* $X = \{(u, \mathbf{v}) : u \in \Sigma^* \sharp, \mathbf{v} \in \Omega^*\}$, $f : (u, \mathbf{v}) \rightarrow [n]$, *and* $f(u, \mathbf{v})$ *is the multiplicity or number of occurrence of* $(u, \mathbf{v})$ *in* X.

For simplicity, for a given $S_n \langle X, f \rangle$, if $(u\sharp, \mathbf{v}) \in X$, we will write $(u\sharp, \mathbf{v}) \in S_n$ unless the context requires to be more specific.

The training data or the stochastic S_n for the proposed algorithm are *non-conflicting* or *ub-ambiguous*. Formally, S_n obeys the following condition:

$$\forall (u\sharp, \mathbf{v}), (u\sharp, \mathbf{v}') \in S_n \Rightarrow \mathbf{v} = \mathbf{v}'.$$

4.4. Queries Used

In the domain of grammatical inference, queries (Angluin, 1981, 1987a,b) have been used to learn different types of automata including transducers (Vilar, 1996). There are various types of queries that have been used including membership queries (Angluin, 1987a,b), equivalence queries (Angluin, 1987a,b), extended membership queries (Bergadano and Varricchio, 1996), translation queries (Vilar, 1996) etc. In our proposed algorithm we will use *extended prefix language queries*. Extended prefix language queries were introduced by de la Higuera and Oncina in (de la Higuera and Oncina, 2004) where such queries have been used for identification of *probabilistic finite state automata* (PFAS) in the limit.

Definition 21 ***Extended prefix language queries*** *(*EXPQ*) are made by submitting a string* w *to an oracle. Then the oracle returns the probability* $Pr_{\mathcal{D}}(w\Sigma^*)$*, i.e., the probability of* w *being a prefix of the stochastic language* $\mathcal{D}$.

For example, if we ask queries *w.r.t.* the input language of the PST given in Figure 4.1(b): EXPQ(aa), the oracle should return the probability 0.2.

4.5. Probabilistic Prefix Tree Transducers

Similar to other state merging algorithms described in Chapter 3 (*e.g.*, RPNI, OSTIA), the proposed algorithm also starts with a tree-like finite state machine, called a *probabilistic prefix tree transducer*.

During the run of the learning algorithm, we may have transitions for which at a given time the outputs are still unknown. In order to denote the outputs of the transitions that are still unknown, we introduce a new symbol $\perp$ such that, $\forall a \in \Omega^*, lcp(\{a\} \cup \{\perp\}) = a$ and $\forall a \in \Omega^*, \perp \cdot u = u \cdot \perp = \perp$.

The formal construction of a PTST and assignment of the probabilities by means of EXPQ *w.r.t.* the input strings are given in the following definition:

Definition 22 *A **probabilistic tree subsequential transducer** (*PTST*) is a 5-tuple* $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E\rangle$ *where* $\psi(T) = \langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E\rangle$ *is a* PST *and* T *is built from a training sample* S_n *such that:*

- $Q = \bigcup_{(u,\mathbf{v})\in S_n} \{q_x : x \in Pref(u)\}$,
- $\{q_0\} = \{q_\epsilon\}$,
- $E = \{e \mid prev\,[e] = q_u, next\,[e] = q_v \Rightarrow v = ua, a \in \Sigma, i\,[e] = a, o\,[e] = \epsilon\}$,
- $\forall q_u \in Q, \forall e \in E\,[q]\,,\ i\,[e] = \sharp, o\,[e] = \mathbf{v}$ *if* $(u, \mathbf{v}) \in S_n, \perp$ *otherwise,*
- $\forall e \in E\,[q_0]\,,\ prob\,[e] = \text{EXPQ}(i\,[e]\,\Sigma^*)$,
- $\forall q_u \in Q \setminus \{q_0\}, \forall e \in E\,[q_u]\,,\ prob\,[e] = \frac{\text{EXPQ}(ui[e]\Sigma^*)}{\text{EXPQ}(u\Sigma^*)}$.

Figure 4.2 shows a toy example of constructing a PTST. Here, the training examples are generated from the PST given in Figure 4.1(b).

Notice that the states in Figure 4.2 are numbered from $q_0 \ldots q_k$ where $k \in [n]$. It is important to note that the states in the PTST are numbered in a special order called the *length-lexicographic order* (*a.k.a.* the *hierarchical* or the *length-lex* order) (see Appendix D) of the prefix set of the input strings in the sample S_n starting from ϵ as q_0. We will denote the number of states of a PST or a PTST T as $|T|$. Notice that in case of a PTST T with states numbered from $q_0 \ldots q_k$, $|T| = k + 1$.

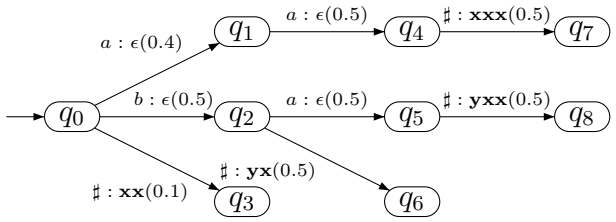


Figure 4.2.: A PTST built from $S_4 = \{(\sharp, \mathbf{xx}), (b\sharp, \mathbf{yx}), (aa\sharp, \mathbf{xxx}), (ba\sharp, \mathbf{yxx})\}$ and by asking EXPQ considering the PST given in Figure 4.1(b) as the target machine.

4.6. The Onward PTST

The PTST depicted in Figure 4.2 is still not in the onward form (Definition 19). An onward form can be obtained by the following algorithm:

- Let T be a PTST and $\pi_q \in \Pi_T(I)$ be a path in T such that $next[\pi_q] = q$;
- $\forall e \in E[q_0], o[e] \leftarrow lcp(\{\mathbf{y} : (i[e] \cdot (\Sigma^* \cup \{\Sigma^*\sharp\}), \mathbf{y}) \in S_n\})$;
- $\forall q \in Q \backslash \{q_0\}, \forall e \in E[q]$,

$$\begin{aligned} o[e] \quad \leftarrow \quad & lcp(\{\mathbf{y} : (i[\pi_q] \cdot (\Sigma^* \cup \{\Sigma^*\sharp\}), \mathbf{y}) \in S_n\})^{-1} \\ & lcp(\{\mathbf{y} : (i[\pi_q] \cdot i[e] \cdot (\Sigma^* \cup \{\Sigma^*\sharp\}), \mathbf{y}) \in S_n\}). \end{aligned}$$

Figure 4.3 shows the onward form of the PTST given in Figure 4.2.

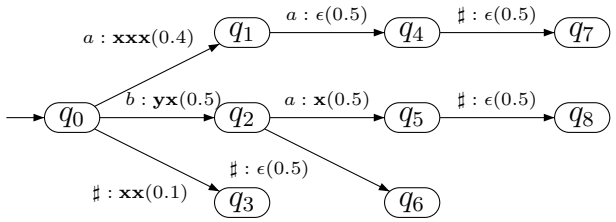


Figure 4.3.: The onward form of the PTST shown in Figure 4.2.

Intuitively, the onwarding process is advancing the output as close to the initial state as possible without causing any inconsistency *w.r.t.* the training data.

In practice an onward PTST can be built from the data very efficiently. The pseudo code for building PTST from stochastic sample and by asking EXPQ is presented in Algorithm 6.

Algorithm 6: ONWARDPTST

Input: a sample S_n
Output: a PTST T

$Q \leftarrow \{q_\epsilon\}$;
$q_x \leftarrow q_\epsilon$;
for $(u, \mathbf{v}) \in S_n$ **do**
  Let $u_i \in \{u_i | u_1 \cdot \ldots \cdot u_{|u|} = u, u_i \in \Sigma \cup \{\sharp\}, 1 \geq i \geq |u|\}$;
  for $i = 1$ *to* $i = |u|$ **do**
    if $\exists e \in E[q_x]$ *such that* $i[e] = u_i$ **then**
      for $d \in E[next[e]]$ **do**
        $o[d] \leftarrow lcp(o[e], \mathbf{v})^{-1} o[e] \cdot o[d]$;
      $o[e] \leftarrow lcp(o[e], \mathbf{v})$;
      $\mathbf{v} \leftarrow lcp(o[e], \mathbf{v})^{-1}\mathbf{v}$;
    else
      $Q \leftarrow Q \cup \{q_{x \cdot u_i}\}$;
      Let e' be an edge such that:
      $prev[e'] \leftarrow q_x$;
      $next[e'] \leftarrow q_{x \cdot u_i}$;
      $i[e'] \leftarrow u_i$;
      $o[e'] \leftarrow \mathbf{v}$;
      if $i = 1$ **then** $prob[e'] \leftarrow \mathrm{EXPQ}(u_i)$;
      else $prob[e'] \leftarrow \frac{\mathrm{EXPQ}(u_1 \cdot \ldots \cdot u_i)}{\mathrm{EXPQ}(u_1 \cdot \ldots \cdot u_{i-1})}$;
      $E \leftarrow E \cup \{e'\}$;
return T;

4.7. The Phantoms

Another new type of object that is used in our proposed algorithm, is the notion of a *phantom*. A phantom is an imaginary arc in a PST with zero probability. Since phantoms have zero probabilities, adding phantoms to a PST does not

change the stochastic bi-language represented by the PST. The formal construction of a phantom *w.r.t.* a PTST is the following:

Definition 23 (Phantoms) *A phantom φ w.r.t. a* PTST $T\,\langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E\rangle$ *is a triple $\varphi = \langle q_u, e, q'_u\rangle$ where, $q_u \in Q$, $e \notin E$ and $q'_u \notin Q$. The construction of each of these edges e is the following:*

- $prev\,[e] = q_u$
- $i\,[e] = a,$ *such that* $a \in \{\{\Sigma \cup \{\sharp\}\} \backslash \{b | \forall e' \in E\,[q_u]\,, i\,[e'] = b\}\},$
- $o\,[e] = \perp,$
- $prob\,[e] = 0,$
- $next\,[e] = q'_u = q_{|T|}.$

A phantom is only added to a given state of a PTST if the following condition holds:

$$\sum_{e \in E[q]} prob\,[e] = 1 \;\wedge\; |E\,[q]| < |\Sigma \cup \{\sharp\}|\,. \tag{4.1}$$

Figure 4.4 depicts examples of phantoms with dotted lines. Essentially phantoms are edges and states that in the PTST do not exist in the target PST.

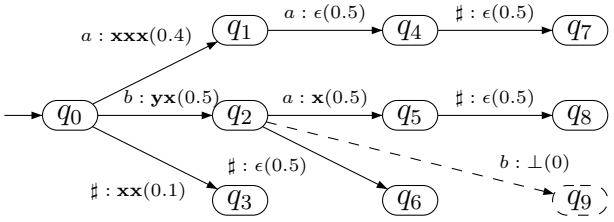


Figure 4.4.: The onward PTST with phantoms (dotted lines). Notice, that the condition 4.1 holds only in case of state q_2 and therefore only one phantom is added in this particular PTST.

Algorithm 7: PUSHBACK

Input: a PST T, two edges $e, e' \in E$
Output: a PST T updated

```
1 for d ∈ E[next[e]] do
2     o[d] ← lcp({o[e], o[e']})^{-1} o[d];
3 for d ∈ E[next[e']] do
4     o[d] ← lcp({o[e], o[e']})^{-1} o[d];
5 o[e] ← lcp({o[e], o[e']});
6 o[e'] ← lcp({o[e], o[e']});
7 return T;
```

4.8. The Pushback Operation

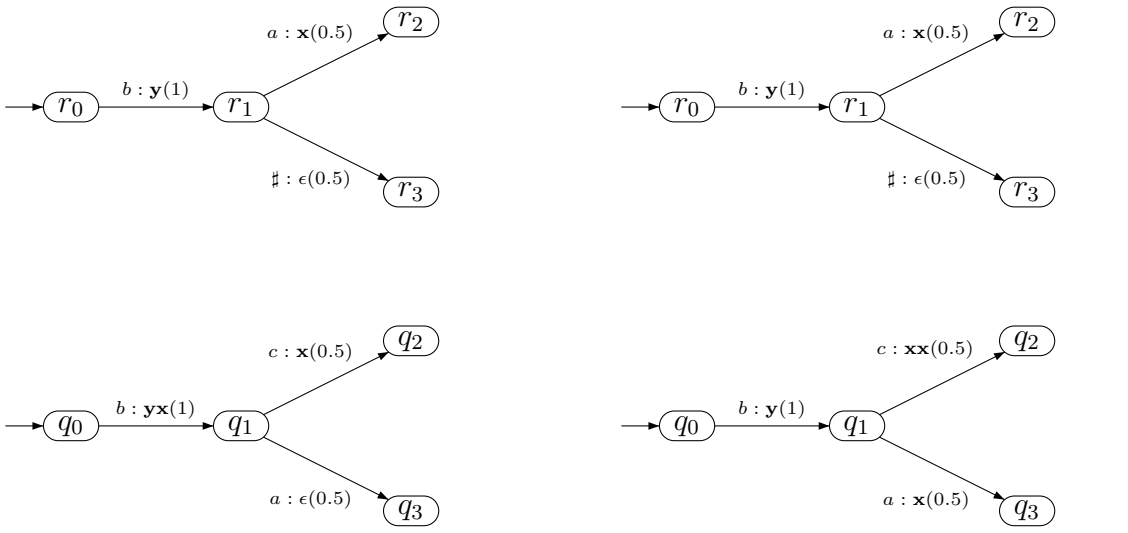
The *pushback* operation used in the proposed algorithm is same as OSTIA (see Chapter 3). For the sake of completeness we will provide pseudo code (Algorithm 7) for the pushback operation used in the proposed algorithm.

Due to the onwarding process some outputs are advanced to close to the initial state of the PTST. Therefore, sometimes it is required to move these outputs back toward the leaves of the tree. Pushback operation essentially performs this task.

Algorithm 7 is the pseudo code for pushback operation. This routine will be used as an auxiliary function of our proposed algorithm. Figure 4.5 shows an illustration of the pushback operation.

4.9. The RED & BLUE States

The proposed algorithm works with state merging using the RED-BLUE notion given in (de la Higuera, 2010). Basically, the algorithm maintains two sets of states: RED and BLUE. The RED states are the states which will be contained in the resulting PST of the algorithm. The BLUE states are the states which are under going experimentations to figure out if they can be merged to a RED state or not. If a BLUE state can not be merged to any of the current RED stated, then the BLUE state is *promoted* to a RED state. Whenever a BLUE state becomes a RED state, it becomes permanent and is a part of the inferred machine by the proposed algorithm.



(a) Before performing pushback on the edges $(q_0, b, \mathbf{yx}, q_1)$ and $(r_0, b, \mathbf{y}, r_1)$.

(b) After performing pushback.

Figure 4.5.: An example of the pushback operation.

4.10. The Merging and Folding Operations

State merging is one of the core operations of state merging algorithms such as RPNI, OSTIA (see Chapter 3) and also used in our proposed algorithm. In our proposed algorithm we adapt the recursive framework of merge and fold technique introduced by de la Higuera in (2010).

Algorithm 8: MERGE

Input: a PST T, two states $q \in$ RED and $q' \in$ BLUE
Output: a tuple of updated T and a boolean value

1 Let $e \in E$ and $q_f \in Q$ such that $next\,[e] = q'$, $prev\,[e] = q_f$;
2 $next\,[e] \leftarrow q$;
3 **return** AWSTIFOLD(T, q, q');

Algorithm 8 shows the pseudo code for the merge operation. Essentially, the merge operation between two states $q_0 \in$ RED and $q_1 \in$ BLUE is redirecting all the incoming edges of the state q_1 to q_0 and then calling the FOLD routine.

Finally, it returns the updated PTST and a boolean value indicating if the merge is valid or not returned by FOLD routine.

Algorithm 9: FOLD

Input: a transducer T, two states $q \in$ RED and $q' \in$ BLUE
Output: a tuple of updated T and a boolean value

```
1  for a ∈ Σ do
2      if ∃e' ∈ E[q'] : i[e'] = a then
3          if ∃e ∈ E[q] : i[e] = a then
4              if (next[e] ∈ RED ∧ o[e'] ∉ Pref(o[e])) or
               (o[e] ≠ o[e'] ∨ prob[e] ≠ prob[e']) then
5                  return (T, false);
6              else
7                  PUSHBACK(T, e, e');
8                  FOLD(T, next[e], next[e']);
9          else
10             prev[e'] ← q;
11 return (T, true);
```

The fold operation is described formally in Algorithm 9. Figure 4.6 shows an example of merge and fold operations.

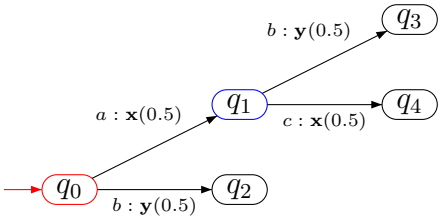
4.11. The Inference Algorithm

The proposed algorithm APTI (Algorithm for Probabilistic Transducer Inference) (Algorithm 18) consists of four phases:

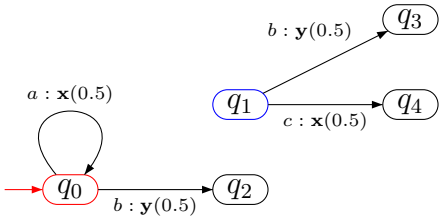
Initialization: In this phase the algorithm builds a tree transducer (Definition 22) in onward form, which is the exact representation of the training data.

Query: In this phase the algorithm populates the probabilities of the edges of the tree transducer by means of EXPQ (Definition 21) on the observed data.

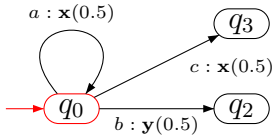
Phantomization: This is the phase when phantoms are added to the tree transducer with zero probability for those states $q \in Q$, where the condition 4.1 holds.



(a) Before merging q_0 and q_1.



(b) After merging q_0 and q_1 and before folding



(c) After folding.

Figure 4.6.: An example of merge and fold operation.

State Merging: This is the phase where we iteratively keep on merging states keeping the hypothesis transducer consistent to the training sample till the algorithm terminates.

In the following sub-sections we describe the details of each phase of APTI.

4.11.1. The Initialization Phase & the Query Phase

The APTI (Algorithm 18) algorithm starts with an overfitted model of the training data. It builds the PTST from the given stochastic sample S_n $\langle X, f \rangle$ (Definition 20) which is fundamentally the exact representation of X. The PTST built from the training data is still incomplete since we do not have the probabilities of the edges of the PTST. In the query phase of the algorithm the PTST is made complete by assigning the probabilities to its edges by asking EXPQ.

The function ONWARDPTST used in Algorithm 18 performs these tasks. The formal construction of the PTST from S_n and by EXPQ is shown in Definition 22.

4.11.2. The Phantomization Phase

After the initialization phase and the query phase, the algorithm APTI adds phantoms to the PTST.

After having built the onward PTST, in the third phase of the algorithm, phantoms are added to the PTST the routine ADDPHANTOMS (Algorithm 10). We add such edges for every state in PTST where the conditions in (4.1) hold.

Figure 4.4 shows an example of PTST where new edges with zero probability have been added. Notice, in this particular example, that this is only possible at the state q_2. After populating the tree with probabilistic queries (EXPQ) we know that the probability of the edge from q_2 with input symbols b is zero, and hence why we add a phantom with probability 0. Syntactically the phantoms are of no use. However, this extra bit of information in the PTST improves the learning capacities of the algorithm.

Finally, the state merging phase of APTI is essentially similar to the OSTIA algorithm, with modified state merging strategy. The details of OSTIA can be found in (Oncina et al., 1993; Castellanos et al., 1998; de la Higuera, 2010). Here we follow the recursive formalism given in (de la Higuera, 2010).

Algorithm 10: ADDPHANTOMS

Input: a PTST T
Output: a PTST T with phantoms added

```
for q ∈ Q do
    if Σ_{e∈E[q]} prob[e] = 1 ∧ |E[q]| < |Σ ∪ {♯}| then
        for a ∈ {Σ ∪ {♯}}\{b|b = i[e] ∧ e ∈ E[q]} do
            E ← {(q, a, ⊥, 0, q_a)} ∪ E;
return T;
```

Algorithm 11: REMOVEPHANTOMS

Input: a PTST T
Output: a PTST T without phantoms

```
for e ∈ E do
    if prob[e] = 0 then
        Q ← Q\{next[e]};
        E ← E\{e};
return T;
```

Algorithm 12: APTI

Input: a sample S_n
Output: a PST T

$T \leftarrow \text{ONWARDPTST}(S_n)$;
$T \leftarrow \text{ADDPHANTOMS}(T)$;
$\text{RED} \leftarrow \{q_\epsilon\}$;
$\text{BLUE} \leftarrow \{q_a \colon a \in \Sigma \cap \mathit{Pref}(\{u | (u, \Omega^*) \in S_n\})\}$;
while $\text{BLUE} \neq \emptyset$ **do**
  $q = \text{CHOOSE}_{<_{lex}}(\text{BLUE})$;
  for $p \in \text{RED}$ *in lex-length order* **do**
    $(T', \mathit{IsAccept}) \leftarrow \text{MERGE}(T, p, q)$;
    if *IsAccept* **then**
      $T \leftarrow T'$;
    else
      $\text{RED} \leftarrow \text{RED} \cup \{q\}$;
      $\text{BLUE} \leftarrow \{q \colon \forall p \in \text{RED}, \forall e \in E[p], q = next[e] \wedge q \notin \text{RED}\}$;
$T \leftarrow \text{REMOVEPHANTOMS}(T)$;
return T;

4.11.3. The State Merging Phase

The proposed algorithm APTI (Algorithm 18) selects a candidate pair of RED and BLUE states in lex-length order using the CHOOSE$_{<_{lex}}$ function. The MERGE function merges the two selected states and recursively performs a cascade of folding of a number of edges (see (de la Higuera, 2010) for details). Some pushback operations (Algorithm 7), which are reverse of onwarding, also take place to be able to fold those edges where output strings are advanced too close to the initial state due to the onwarding process. During the recursive fold operation, it is actually decided that whether a merge is accepted or not. A merge is rejected if any one of the following holds: 1) if there is a conflict *w.r.t.* the outputs of any two edges having the input symbol $\sharp$ 2) if there is a conflict *w.r.t.* the probabilities of any two edges. This process is repeated till the BLUE set is empty. Finally before termination, the phantoms are removed from the inferred machine using the routine REMOVEPHANTOMS (Algorithm 11).

4.12. A Run of the APTI Algorithm

As a running example, we will consider the PST shown in Figure 4.1(b) as the target PST. The given training data is $S_4 = \{(\sharp, \mathbf{xx}), (a\sharp, \mathbf{xx}), (b\sharp, \mathbf{yx}), (aa\sharp, \mathbf{xxx}), (baa\sharp, \mathbf{yxxx})\}$. For simplicity, we have not shown the number of occurrence of each translation pairs.

The initial PTST built from S_4 is shown in Figure 4.7 before the queries are made. Notice that the PTST is incomplete since the probabilities of the edges are still missing. Then, the PTST is then populated with the probabilities of the edges by asking EXPQ (Figure 4.8). Next, we convert the PTST into the onward form (Figure 4.9). Building the PTST in onward form by asking EXPQ can be done very efficiently by using the routine ONWARDPTST (Algorithm 6).

In the onward PTST, the phantoms are then added and the RED and BLUE states are initialized (Figure 4.10).

Then the algorithm tries to merge q_0 and q_1. This merge fails because the phantom $(q_1, b, \perp, 0, q_{11})$ can not be folded on the edge $(q_0, b, \mathbf{yx}, 0.5, q_2)$. After that q_1 is promoted to RED and consequently q_4, q_5, and q_{11} are added to the BLUE set. The algorithm selects the lowest BLUE state, which is q_2. The merge between the RED state q_0 and the lowest BLUE state q_2 is then tried and this merge is also rejected (Figure 4.12). Then the merge between q_1 and q_2 is tried and this merge is successful. The resulting machine is shown in Figure 4.13.

Now the candidate merge pair is q_0 and q_3. This merge is also accepted and the resulting machine is depicted in Figure 4.14. At this point the candidate

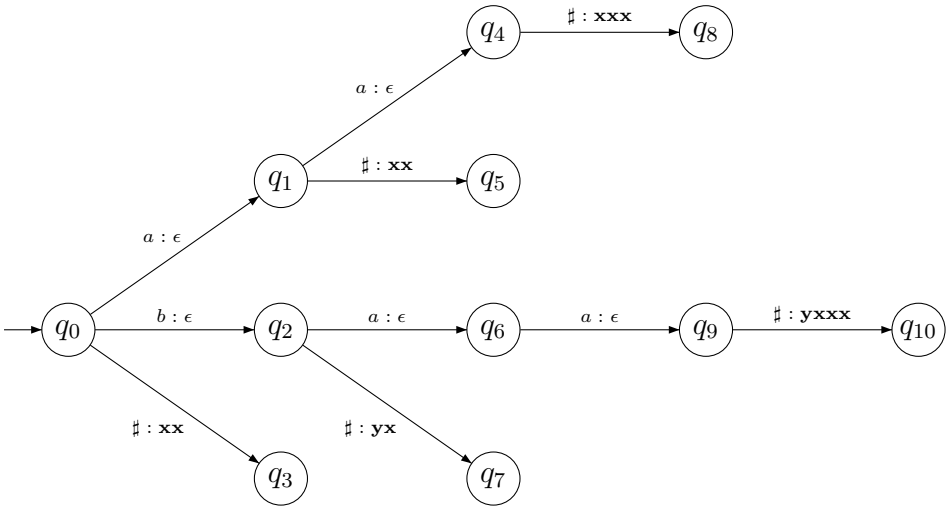


Figure 4.7.: A PTST built from $S_4 = \{(\sharp, \mathbf{xx}), (a\sharp, \mathbf{xx}), (b\sharp, \mathbf{yx}), (aa\sharp, \mathbf{xxx}), (baa\sharp, \mathbf{yxxx})\}$ before the queries are made.

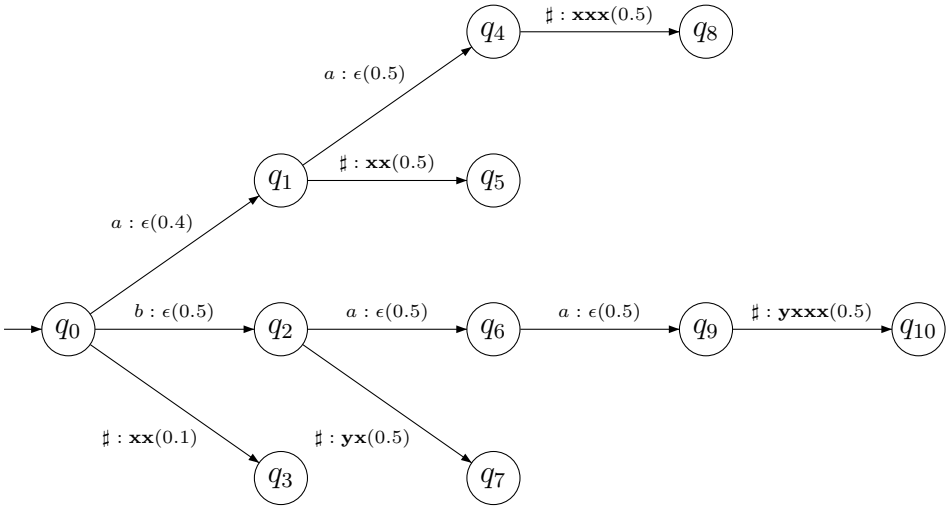


Figure 4.8.: A PTST built from $S_4 = \{(\sharp, \mathbf{xx}), (a\sharp, \mathbf{xx}), (b\sharp, \mathbf{yx}), (aa\sharp, \mathbf{xxx}), (baa\sharp, \mathbf{yxxx})\}$ and by asking EXPQ.

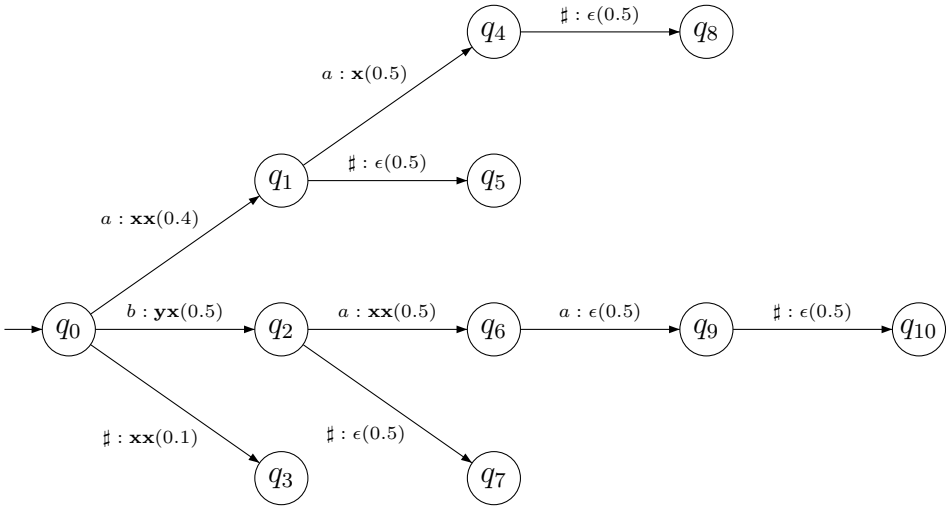


Figure 4.9.: Onward form of the PTST shown in Figure 4.8.

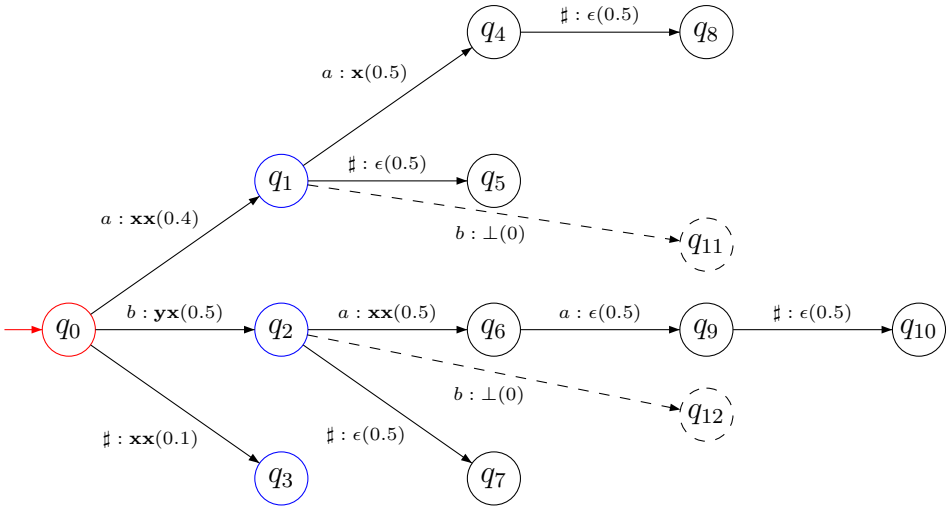


Figure 4.10.: Phantoms are added to the onward PTST and the RED and the BLUE sets are initiated.

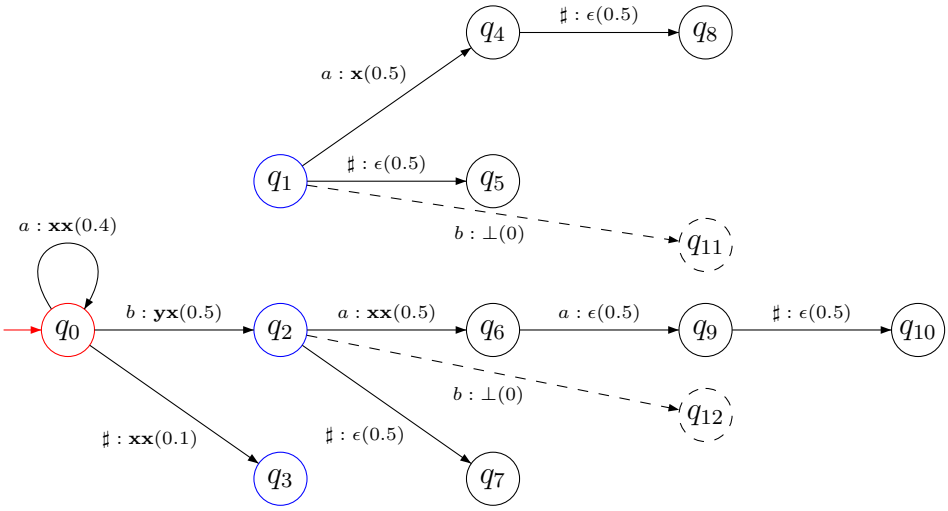


Figure 4.11.: The merge between the state q_0 and q_1 is tried.

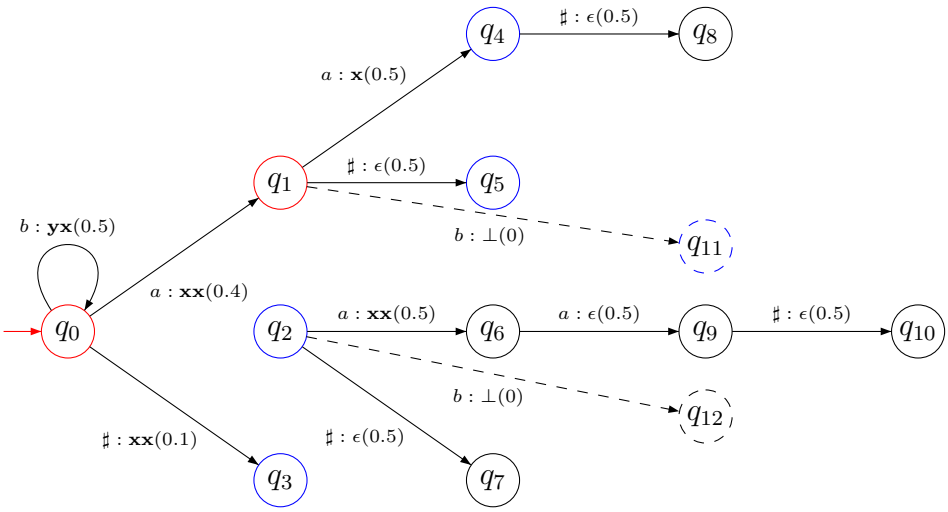


Figure 4.12.: The merge attempt between q_0 and q_2.

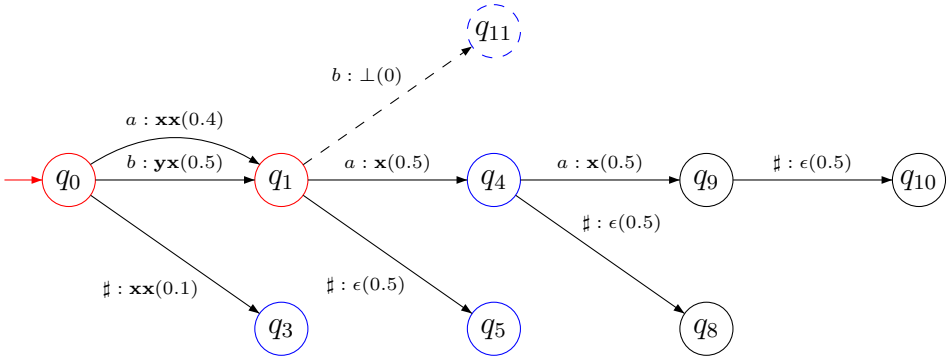


Figure 4.13.: After merging and folding q_1 and q_2.

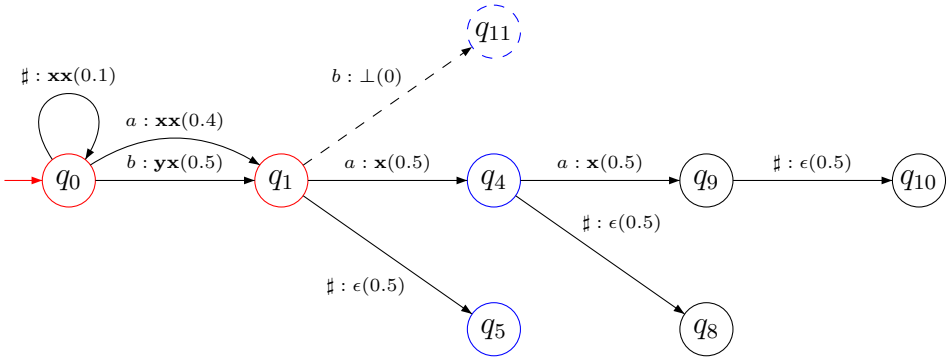


Figure 4.14.: After merging and folding q_0 and q_3.

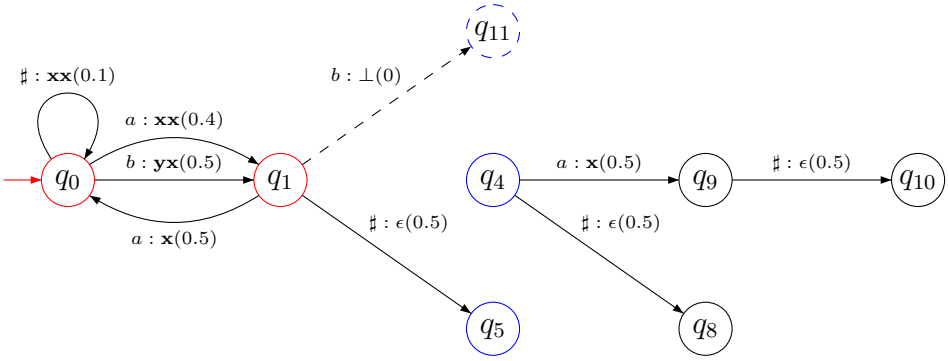


Figure 4.15.: The merge attempt between q_0 and q_4.

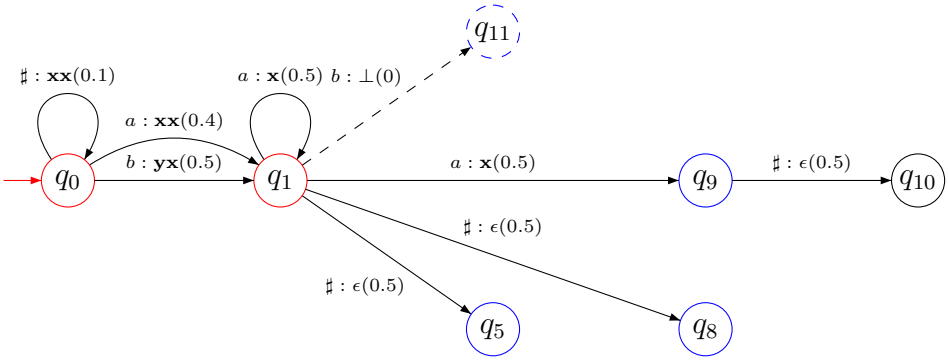


Figure 4.16.: After merging and folding q_1 and q_4.

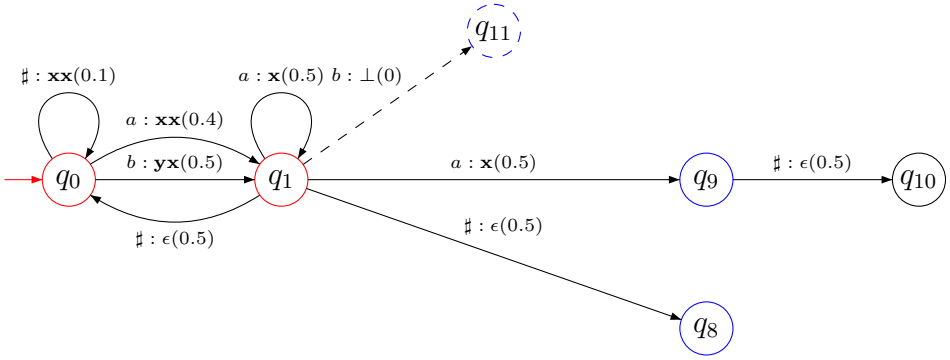


Figure 4.17.: After merging and folding q_0 and q_5.

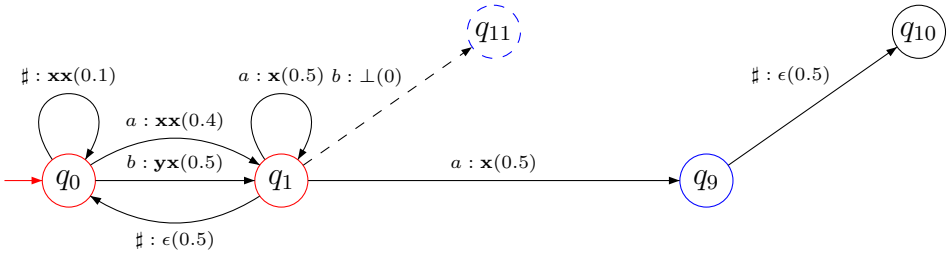


Figure 4.18.: After merging and folding q_0 and q_8.

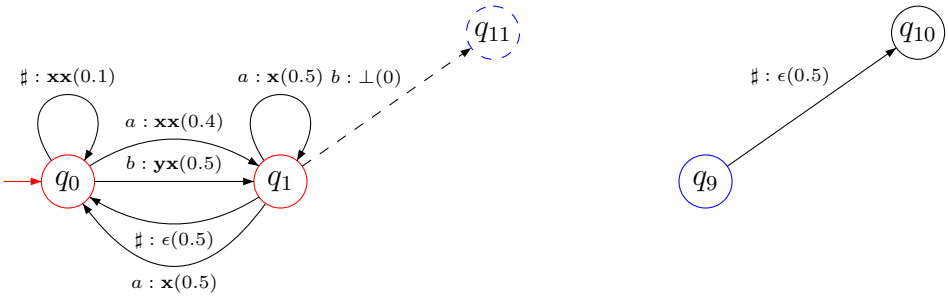


Figure 4.19.: The merge attempt between q_0 and q_9.

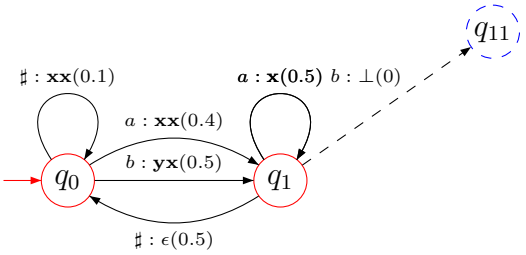


Figure 4.20.: After merging and folding q_1 and q_9.

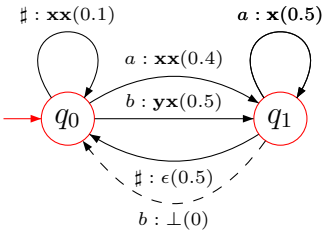


Figure 4.21.: After merging and folding q_0 and q_{11}.

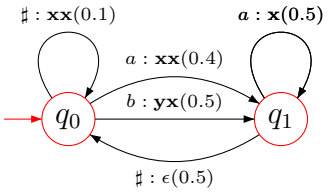


Figure 4.22.: After removing the phantoms.

merge pair is q_0 and q_4 (Figure 4.15). This merge is rejected.

The next candidate merge pair is q_1 and q_4. This merge is accepted and the resulting transducer after merging and folding is shown in Figure 4.16. As a consequence of this merge, q_8 and q_9 are added to the BLUE set. The next candidate states are q_0 and q_5. This merge is accepted and the resulting machine is shown in Figure 4.18. Then the merge between q_0 and q_8 is tried and this merge is accepted (Figure 4.18). Then the next merge attempt between q_0 and q_9 is rejected (Figure 4.20). The algorithm APTI tries to merge the blue state q_9 with another RED state q_1. This merge is successful. Next, the algorithm merges the states q_0 and q_{11} and this merge is accepted (Figure 4.21). Finally, the phantoms are removed from the hypothesis (Figure 4.22). As we can see, the hypothesis machine is exactly identical to the target machine shown in Figure 4.1(b).

4.13. Analysis of the Algorithm

In this section we will present an analysis of the proposed algorithm. Here, we prove the correctness of APTI in an identification in the limit model. Moreover, we show the runtime complexity and query complexity of APTI.

We define the *prefix set* ($\mathbb{PR}$) and the *short prefix set* ($\mathbb{SP}$) with respect to a stochastic transduction $\mathcal{R}$ as the following:

$$\begin{aligned}\mathbb{PR}(\mathcal{R}) = &\quad \{u \in \Sigma^* | (u, \mathbf{v})^{-1}\mathcal{R} \neq \emptyset, \mathbf{v} \in \Omega^*\} \\ \mathbb{SP}(\mathcal{R}) = &\quad \{u \in \mathbb{PR}(\mathcal{R}) | (u, \mathbf{v})^{-1}\mathcal{R} = (w, \mathbf{x})^{-1} \\ &\quad \mathcal{R} \Rightarrow |u| \leq |w|\}\end{aligned}$$

The *kernel set* ($\mathbb{K}$) of $\mathcal{R}$ is defined as follows:

$$\mathbb{K}(\mathcal{R}) = \{\epsilon\} \cup \{ua \in \mathbb{PR}(\mathcal{R}) | u \in \mathbb{SP}(\mathcal{R}) \wedge a \in \Sigma\}$$

Note that $\mathbb{SP}$ is included in $\mathbb{K}$.

Identification with probability one will be achieved if given any possible target transduction $\mathcal{R}$ defined by a PST, given any infinite presentation of translation pairs drawn according to the joint distribution, and, denoting by S_n the multi-set consisting of the n first pairs, with probability one, APTI returns a correct hypothesis from S_n for all but a finite number of values of n.

The characteristic sample for the given algorithm essentially requires to meet three conditions: firstly, it should include all the states and edges. Secondly, some properties should hold that disallow wrong merges to take place. Finally, it should have enough examples that ensure the alignments of the input and output during the learning phase. For the first condition we only need the kernel of the stochastic transduction or strings that contain the elements of the kernel as prefixes to be included in the sample. For the second condition, for states to be declared non mergeable we need any one of the following: 1) output mismatch 2) probability mismatch. Since we are adding zero probability edges for the non existing edges in the PTST, at least one of these conditions is guaranteed to be eventually met by the stochastic sample. Formally, we can define the characteristic sample as:

Definition 24 *A stochastic sample S_n is said to be characteristic ($\mathbb{CS}$) w.r.t. a* SDRT $\mathcal{R}$ *if it satisfies the following conditions:*

1. $\forall u \in \mathbb{K}, \exists (uw, \mathbf{vx}) \in S_n$ *such that* $w \in \Sigma^*, \mathbf{vx} \in \Omega^*, (w, \mathbf{x}) \in (u, \mathbf{v})^{-1}\mathcal{R}$,
2. $\forall u \in \mathbb{SP}, u' \in \mathbb{K}$, *if* $(u, \mathbf{v})^{-1}\mathcal{R} \neq (u', \mathbf{v}')^{-1}\mathcal{R}$ *where* $\mathbf{v}, \mathbf{v}' \in \Omega^*$, *then any one of the following holds:*
   - $\exists (uw, \mathbf{vx}), (u'w, \mathbf{v}'\mathbf{x}'), (ur, \mathbf{vy}),$ $(u'r, \mathbf{v}'\mathbf{y}') \in S_n$ *such that:* $\frac{Pr_{\mathcal{R}}(uw, \mathbf{vx})}{Pr_{\mathcal{R}}(ur, \mathbf{vy})} \neq \frac{Pr_{\mathcal{R}}(u'w, \mathbf{v}'\mathbf{x}')}{Pr_{\mathcal{R}}(u'r, \mathbf{v}'\mathbf{y}')}$
   - $\exists (uw, \mathbf{vx}), (u'w, \mathbf{v}'\mathbf{x}'), \in S_n$ *such that:* $(w, \mathbf{x}) \in (u, \mathbf{v})^{-1}\mathcal{R}, (w, \mathbf{x}') \in (u', \mathbf{v}')^{-1}\mathcal{R}, x \neq x'$
3. $\forall u \in \mathbb{K}, \exists (uw, \mathbf{vx})(uw', \mathbf{vx}') \in S_n$ *such that* $lcp(\mathbf{x}, \mathbf{x}') = \epsilon \wedge w \neq w'$.

Lemma 16 *For any $i^{th}(i \geq 1)$ call of the subroutine* MERGE, *it recursively computes the i^{th} hypothesis* PST H_i *such that if the merge is successful, $|H_i| < |H_{i-1}|$ and H_i is consistent with the training sample S_n.*

Proof It is easy to see that $|H_i| < |H_{i-1}|$ after each merge is accepted. H_i is consistent with the training sample S_n because of the following: initially the hypothesis H_0 is the PTST which is the exact representation of S_n. A merge

between the two states q and q' is accepted only when the recursive fold operation is successful. The recursive fold operation returns a negative result if at any time during the fold there are two candidate edges e and e' such that $i[e] = i[e']$ are incompatible *w.r.t.* the probabilities of e and e' or if there is incompatibility between the output strings when $i[e] = i[e'] = \sharp$. Therefore, at anytime during the algorithm H_i will be consistent to the training sample. □

Lemma 17 *The algorithm* APTI *terminates after a finite number of steps.*

Proof The initial PTST consists of a finite number of states. Therefore, there can be only finite number of RED and BLUE states. At each iteration of the outer most loop of the algorithm (Algorithm 18) either a BLUE state will disappear (if the merge is accepted) or will be promoted to a RED state and at the same time the offspring of the new RED states will become BLUE states. Therefore, it can be seen that eventually the set BLUE will be empty, which is precisely the terminating condition for the outermost loop. □

Theorem 5 *The algorithm* APTI *identifies* SDRT *in the limit with probability one from positive presentation and* EXPQ*.*

Proof The condition 1 of the $\mathbb{CS}$ ensures that there will be at least as many states in the initial hypothesis H_0 (the PTST) as the target T. The condition 2 of the $\mathbb{CS}$ prevents merges of the non-equivalent states during the run. The weights of the edges are populated using EXPQ *w.r.t.* the input language. Even if the transduction scheme is not a total function, the phantoms added to the PTST will prevent merges of non-equivalent states. The condition 3 of $\mathbb{CS}$ ensures factorization of the output strings and correct alignment *w.r.t.* the input symbols. Moreover, any positive presentation of the target T will eventually include $\mathbb{CS}$. Thus, from lemmata 16 and 17, APTI converges to a PST which represents the same SDRT as the target machine T. Therefore, APTI satisfies the conditions of identification in the limit with probability one. □

4.14. Complexity Analysis

In this section we present a complexity analysis of the proposed algorithm. We are primarily interested in the runtime complexity and the query complexity.

Let $\|S_n\| = \sum_{(u,\mathbf{v})\in S_n} |u|$ and $m = max\{|u| : (u, \mathbf{v}) \in S_n\}$.

Theorem 6 *The worst case run time complexity of the algorithm* APTI *is polynomially bounded by* $\mathcal{O}((\|S_n\|\,|\Sigma \cup \{\sharp\}|)^3(m+n) + \|S_n\|\,mn)$.

Proof The worst case run time complexity of APTI is essentially similar to OSTIA with some additional computation costs. The run time complexity of OSTIA has been shown $\mathcal{O}(\|S_n\|^3 (m+n) + \|S_n\| mn)$ in (Oncina et al., 1993). In APTI we are populating the probabilities of the PTST by asking queries and adding phantoms. The computation cost for populating the probabilities of the PTST is linear *w.r.t.* $\|S_n\|$. The computation cost for adding the phantoms is in the worst case $\|S_n\| mn$. Due to the phantoms, in the worst case the total number of states in the PTST will be $\|S_n\| \, |\Sigma \cup \{\sharp\}|$. Therefore, the worst case run time complexity of APTI is $\mathcal{O}((\|S_n\| \, |\Sigma \cup \{\sharp\}|)^3 (m+n) + \|S_n\| mn)$. □

The worst case analysis shown is pessimistic and hardly occurs in practice. Therefore, although the worst case run time complexity is shown to be cubic, practically the runtime is much lower. We report some empirical evidence of APTI's run time in Section 4.15.

Proposition 6 *The algorithm* APTI *requires at most* $\|S_n\| \cdot |\Sigma \cup \{\sharp\}|$ *extended prefix language queries for a given training sample* S_n.

Proof The total number of states in the PTST will be $\|S_n\| \, |\Sigma \cup \{\sharp\}|$ in the worst case. APTI only asks queries to populate the PTST and therefore will ask at most $\|S_n\| \, |\Sigma \cup \{\sharp\}|$ number of EXPQ. □

4.15. Experiments

4.15.1. Data Sets

We conduct our experiments with two types of data sets: 1) artificial data sets generated from random transducers 2) data generated from the so-called Miniature Language Acquisition (MLA) task (Feldman et al., 1990) adapted to English-French translations. Details of the data generation protocol follows.

Data Generation Protocol

For the artificial data sets, we first generate a random PST with m states. The states are numbered from q_0 to q_{m-1} where state q_0 is the initial state. The states are connected randomly; labels on transitions preserve the deterministic property. Then the unreachable states are removed. The outputs are assigned as random strings drawn from a uniform distribution over $\Omega^{\leq k}$, for an arbitrary

value of k. The probabilities of the edges are randomly assigned making sure the following condition holds:

$$\forall q_i \in Q, \sum_{e \in E[q_i]} prob\,[e] = 1 \tag{4.2}$$

Using the random PST T, we generate a stochastic training sample S_n where each translation pair is drawn i.i.d. from the joint distribution $\mathcal{R}_T$. The test set S'_p is also drawn i.i.d. from the joint distribution $\mathcal{R}_T$, restricted to $\Sigma^* \sharp \times \Omega^* \backslash S_n$. Therefore $S'_p \cap S_n = \emptyset$.

The second type of data set which is adapted from the MLA task, essentially consists of pseudo-natural sentences of English and French describing visual scenes within a restricted conceptual domain. The target transducer built under the MLA task is depicted in Figure 4.23. Random probabilities are assigned to the edges of the target PST (Figure 4.23) satisfying condition (4.2). The generation procedure of S_n and S'_p for the MLA task is as above.

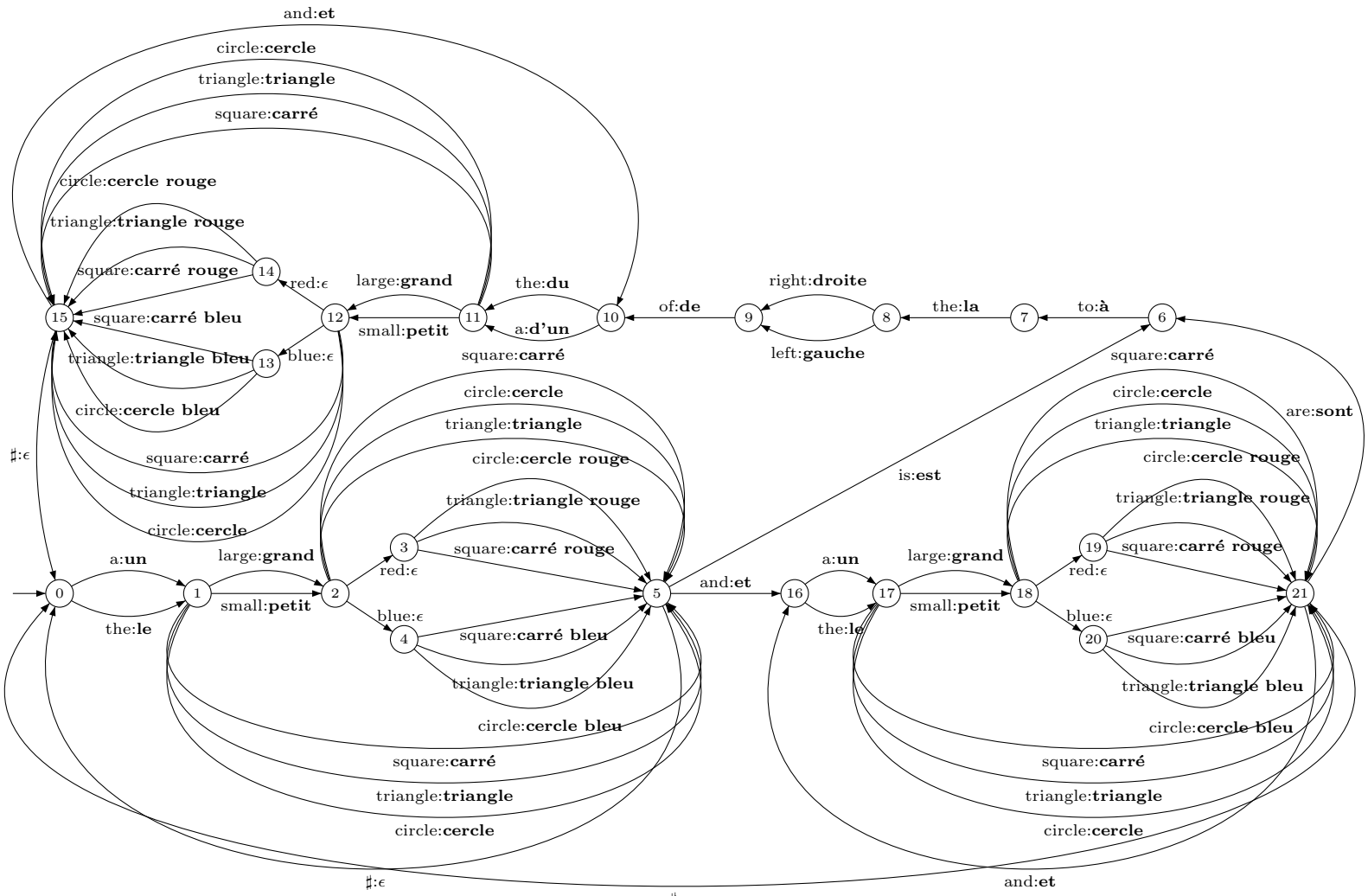


Figure 4.23.: The target PST for an English-French translation under the MLA framework. The probabilities of the edges are not shown for clarity.

Evaluation

In our experiments we measure the Word Error Rate (WER) (Koehn, 2010) as a metric of correctness. Other types of evaluations, such as BLEU (Papineni et al., 2002), NIST (Doddington, 2002) *etc* are beyond the scope of this chapter because they are primarily for evaluating translation quality. Our objective is to analyze the correctness and learnability of APTI for which WER is more appropriate.

The results of the experiment conducted with the artificial data sets are shown in Figures 4.24(a), 4.24(b), and 4.24(c). We have generated a random PST of size 10 and $|\Sigma| = |\Omega| = 5$. APTI is executed with different sizes of inputs, starting from 500 training pairs to 20,000 training pairs, each time incrementing the training set size by 500. Test sets of 1,000 pairs are also being generated for each run. For the sake of statistical significance, the procedure is repeated 10 times for each data point.

The results in Figure 4.24(a) show that the WER gets close to 0 only with 5,000 training pairs and with 8,000 training pairs onwards the WER converges to 0. The objective of this experiment is to show the correctness of the algorithm.

It is important to note that the translation pairs of each training sample have been drawn i.i.d. with replacements. In order to analyze the generalization capabilities of the algorithms *w.r.t.* the percentage of the stochastic transduction $\mathcal{R}$ we define the following. Let U be the set of unique training pairs in S_n defined as:

$$U = \bigcup_{(u,\mathbf{v})\in S_n} \{(u, \mathbf{v})\}.$$

We define the *sample density* d as:

$$d = \sum_{(u,\mathbf{v})\in U} Pr_{\mathcal{R}}(u, \mathbf{v}).$$

Figure 4.24(b) depicts how WER varies with the sample density of the training set. It shows that even for less than 40% of the stochastic transduction $\mathcal{R}$, the WER is as low as 25%. With 45% of $\mathcal{R}$, the WER almost converges to zero.

Figure 4.24(c) shows the *learning rate* of APTI in comparison with rote learner. The learning rate (LR) is defined as:

$$\text{LR} = d + (1 - d) * (1 - \text{WER}).$$

Intuitively, LR tells us the percentage of $\mathcal{R}$ the learner has learnt. In other words, a rote learner can only translate the strings it has seen during training and will therefore have a learning rate of d.

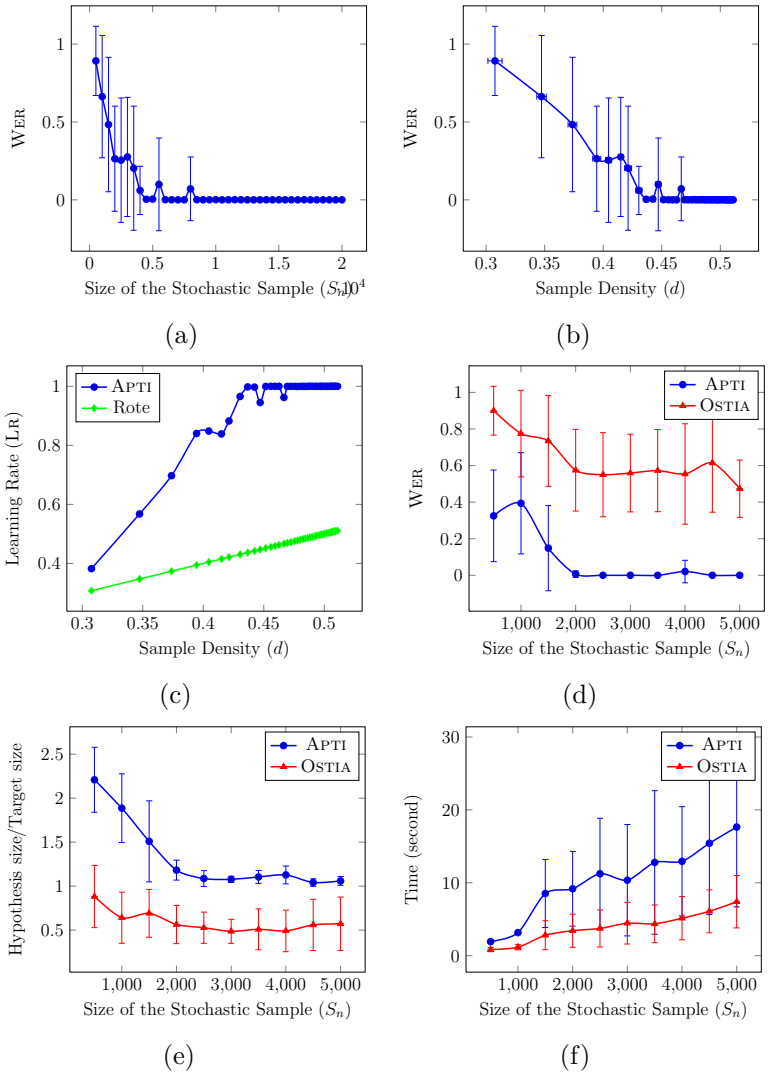


Figure 4.24.: Performance of APTI reported for the artificial data set (Figures 4.24(a), 4.24(b), and 4.24(c)) and the data set for the MLA task (Figures 4.24(d), 4.24(e), and 4.24(f)).

As expected, in Figure 4.24(c) APTI performs better than a rote learner. The slope of the learning rate - density curve is high when the training density is low. The slope decreases as the density gets higher and eventually approaches to 0 after a certain transition phase.

The results shown in Figures 4.24(d), 4.24(e), and 4.24(f) are based on data sets for the MLA task. The experiments were conducted on 5000 training pairs

and 1000 test pairs. Both the algorithms APTI and OSTIA were executed with input size starting from 500 training pairs to 5000 training pairs, incremented by 500 at each step. Similar to the previous experiment, each run is repeated 10 times.

Figure 4.24(d) shows the WER for APTI and OSTIA for different sizes of training data. The WER for APTI almost converges to zero from 2000 training pairs onward, whereas OSTIA continues to result more than 50% of WER.

APTI outperforms OSTIA for a number of reasons. Firstly, in APTI we are exploiting probabilistic information by asking EXPQ to a MAT. Secondly, theoretically OSTIA has got the limitation that it is guaranteed to converge only in case of subsequential transductions that are total functions (Oncina et al., 1993). Our target transducer for the MLA task (Figure 4.23) represents a transduction which is a partial function. Therefore, OSTIA is not capable of rejecting some of the merges between states that should not be merged. As a result, OSTIA over-generalizes the hypothesis and leads to transducers with fewer number of states than the target PST. Figure 4.24(e) shows the ratio between the hypothesis and the target size for each training size. Here, we see that in the case of OSTIA, the hypothesis-target ratio always remains below 1. On the other hand, by introducing the notion of phantoms, we are able to overcome such limitations in APTI. Figure 4.24(e) shows that the ratio of the hypothesis and the target size in case of APTI always remain above 1, *i.e.*, APTI does not over generalize.

For our experiments, we have implemented OSTIA and APTI using OpenFST (Allauzen et al., 2007), an open source C++ library for FSTs. Figure 4.24(f) shows some timing results of OpenFST implementation of OSTIA and APTI. It shows that the execution time of APTI is more than the execution time of OSTIA. In case of APTI additional edges and states are added to the initial tree as phantoms. Therefore, in most of the cases the size of the initial tree for APTI is bigger than the size of the initial tree for OSTIA which is the reason for longer execution time for APTI. Nevertheless, the execution time of APTI is reasonably low as shown in Figure 4.24(f): this always remained below 30 seconds.

4.16. Conclusion

In this chapter we have addressed **RQ**4, which was one of the objectives of the thesis. We have presented an algorithm for learning PSTs from positive training sample and by asking probabilistic queries about the observed data. We have shown that our algorithm APTI is capable of learning partial functions, which were not guaranteed to be learnt using OSTIA. We have experimentally shown that our algorithm outperforms OSTIA. We have used EXPQ *w.r.t.* the given

data, and thus avoided peculiar problems as reported in (Lang and Baum, 1992). This kind of queries can easily be simulated in real life using relative frequencies of the observed data.

Moreover, the generality of our approach can be argued based on the results of **RQ**3 given in Chapter 2. The PST inferred by our proposed algorithm can be converted into an equivalent PHMM and be used where PHMM is a more suitable model.

The expressive power of our model is limited to the class of SDRT, which surely does not include a huge part of natural translation schemes. How to learn more complex classes of transductions remains an open question.

Chapter 5

Learning Probabilistic Subsequential Transducers from Positive Data

> The brain is nothing but statistical decision organ.
>
> Barlow

5.1. Introduction

In the previous chapter (Chapter 4) we have presented an algorithm for learning probabilistic subsequential transducers (PSTs) in the limit using probabilistic queries and positive data. From a practical point of view it is necessary to simulate the probabilistic queries using statistics over the observed positive data. In this chapter we address this problem, which is one of the research questions formulated in Chapter 1. This particular research question is referred as **RQ**5.

The learning problem of PFA from a positive sample is a well explored problem. The first state merging algorithm for learning deterministic PFA from positive examples, ALERGIA, was introduced by Carrasco and Oncina in (1994). Having ALERGIA as the pioneer, a number of modified versions have been presented, *e.g.*, (Thollard et al., 2000; Carrasco and Oncina, 1999). A recent survey about PFA learning can be found in (Verwer et al., 2012).

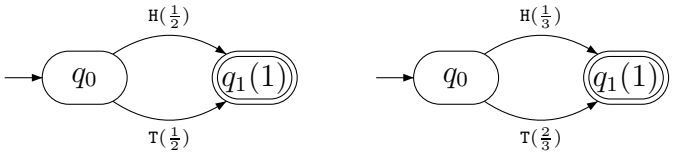
There have been lessons learnt from subsequential transducer learning algorithms in non-probabilistic setting and also from PFA learning algorithms. This chapter aims to utilize the lessons learnt from both of these research streams and attempts to solve the problem of learning PSTs from positive examples.

In this chapter we present a novel algorithm for learning PSTs from a positive sample. We present an analysis of the algorithm to illustrate the theoretical boundaries. Finally we present experimental results based on artificially generated datasets.

5.2. Distributions

PFAs represent distributions over Σ^* (Vidal et al., 2005b). Analogously, probabilistic transducers represent join distributions over $\Sigma^* \times \Omega^*$ (Vidal et al., 2005a). In a state merging algorithm for probabilistic automata the following issue is important: given two PFAs how close are they *w.r.t.* the distributions represented by the PFAs? In this section we will discuss the issues of distributions represented PFAs and their distinguishability.

We will start with the classic example of probability theory, the coin tossing example. The distribution given by tossing a coin (fair or biased) a number of times, is the well known *Binomial distribution*. Details about Binomial distribution can be found in any standard textbook of probability theory, see for example (Feller, 1971; Mitzenmacher and Upfal, 2005; Dubhashi and Panconesi, 2009).



(a) (b)

Figure 5.1.: PFAs modeling the coin tossing scenario. Transition symbol H represents heads and T represents tails. The PFA shown in Figure 5.1(a) models a fair coin and the PFA in Figure 5.1(b) models a biased coin.

The coin tossing scenario can be modeled by a PFA; Figure 5.1 shows two such examples: a fair coin (Figure 5.1(a)) and a biased coin (Figure 5.1(b)). Notice that, in both the cases, during each parse from the initial state there will be exactly one transition, either heads (represented by H) or tails (represented by T) since, both the transitions go to q_1 and the final state probability of q_1 is 1.

We will call the outcome of each parse success, if the symbol H is generated, *i.e.*, if the outcome is heads. Which means a random variable $\mathcal{X}_n$ is the number of times H is generated by one of the automata in Figure 5.1 out of n number of

trials. Let p be the probability of generating H, *i.e.*, in Figure 5.1(a) $p = \frac{1}{2}$ and in Figure 5.1(b) $p = \frac{1}{3}$. Note that each of these trials is an independent event.

The Binomial distribution with parameters n and p is given by:

$$Pr(\mathcal{X} = \mathcal{X}_i) = \binom{n}{i} p^i (p-1)^{n-i}, 0 \leq i \leq n.$$

The random variable $\mathcal{X}_i$ associated with a binomial distribution is known as a *Bernoulli random variable*.

In case of the automaton given for a fair coin (Figure 5.1(a)), for 100 parses, ideally we will expect approximately 50 times success. On the other hand, for the automaton given for a biased coin (Figure 5.1(b)) for 100 parses, we will expect approximately 33 times success. So, the expected relative frequencies are $\frac{50}{100}$ and $\frac{33}{100}$ respectively. The difference between the relative frequencies gives us a strong evidence that they are outcomes of two different coins. The question remains, how close the relative frequencies should be so that they can be treated as equivalent? This can be solved by using a bound on the probability that the random variable does not deviate too much from the expected value. There exist a number of bounds for concentration of measure, *e.g.*, Chernoff bound, Hoeffding bound *etc.* In this chapter we will briefly talk about Hoeffding bounds for concentration of measure.

If we treat the outcome of each parse in Figure 5.1(a) as a Bernoulli variable and the outcome of each parse in Figure 5.1(b) as another Bernoulli variable, intuitively, the Hoeffding bound can tell us if two such random variables follow the same distribution or not.

5.3. Hoeffding Bound

The Hoeffding bound was introduced by Wassilij Hoeffding in (1963). The proof and other details of Hoeffding bound can also be found in (Casella and Berger, 2001; Feller, 1971; Dubhashi and Panconesi, 2009). In this section we will use only one special form of Hoeffding bound.

One of the useful forms of Hoeffding bound is as follows. Let $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n$ be Bernoulli variables with probability p. The observed frequency is $f = \sum_{i=1}^{n} \mathcal{X}_i$. With probability at least $1 - \delta$,

$$\left| p - \frac{f}{n} \right| < \sqrt{\frac{1}{2n} \log \frac{2}{\delta}}. \tag{5.1}$$

We will use inequality 5.1 to derive the formula for testing how close are two relative frequencies. Let f_1, n_1 and f_2, n_2 be two pairs of observed frequencies

and number of trials respectively.

$$\left|p - \frac{f_1}{n_1}\right| + \left|p - \frac{f_2}{n_2}\right| < \sqrt{\frac{1}{2n_1}\log\frac{2}{\delta}} + \sqrt{\frac{1}{2n_2}\log\frac{2}{\delta}}$$

The version of the Hoeffding bound given in (Carrasco and Oncina, 1999), which will be used in the proposed algorithm, is the following:

$$\left|\frac{f_1}{n_1} - \frac{f_2}{n_2}\right| < \sqrt{\frac{1}{2}\left(\frac{1}{n_1} + \frac{1}{n_2}\right)\log\frac{2}{\delta}}. \tag{5.2}$$

It is noteworthy that the Hoeffding bound relatively weak or bad approximation and there are better alternatives. However, to demonstrate the proof of concept in our algorithm we will only use the Hoeffding bound.

5.4. Frequency Transducers

The proposed algorithm works with relative frequencies of the observed data. We use the empirical distribution of the observed data as one of the evidences for state merging decisions. We are primarily interested in the distribution of the domain language. The idea of using the empirical distribution of the domain language as a merge decision criterion is inspired by an algorithm for learning STs in non-probabilistic setting where knowledge of the domain language is used (Oncina and Varó, 1996).

At this point we will define a new type of transducer, called *frequency subsequential transducer*.

Definition 25 (Frequency Finite Subsequential Transducer) *A frequency finite subsequential transducer (*FFST*) is a 6-tuple* $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E, F_r\rangle$ *where* $\psi(T) = \langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E\rangle$ *is a* ST *and* F_r *is the frequency function defined as* $F_r : e \to \mathbb{N}$*, where* $e \in E$*.*

A FFST is *well defined* or *consistent* if the following property holds:

$$\forall q \in Q\backslash\{q_0\}, \sum_{e\in E:next[e]=q} F_r(e) = \sum_{e\in E[q]} F_r(e) \tag{5.3}$$

Since q_0 is the only one initial, the sum of the frequencies of the outgoing edges are assumed to be number of frequency entering the initial state, and therefore q_0 is treated specially. For example, Figure 5.2(a) depicts an example of a consistent FFST.

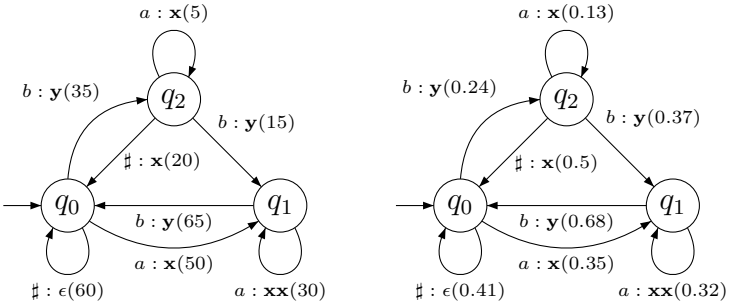
Intuitively, an FFST is an object where the weights are the frequencies of the transitions instead of probabilities. $F_r(e) = n$ should be interpreted as the edge e is used n times.

FFSTs can be converted to equivalent PSTs. Algorithm 13 shows a conversion algorithm that converts an FFST to a PST.

Algorithm 13: CONVERTFFSTTOPST

Input: a FFST $T = \left\langle Q^T, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}^T, E^T, F_r^T \right\rangle$
Output: a PST $T' = \left\langle Q^{T'}, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}^{T'}, E^{T'} \right\rangle$

```
1 Q^{T'} ← Q^T;
2 for q ∈ Q^T do
3     for e ∈ E^T[q] do
4         α ← F_r(e) / Σ_{e∈E[e]} F_r(e);
5         E^{T'} ← E^{T'} ∪ (q, i[e], o[e], α, next[e]);
6 return T';
```



(a) (b)

Figure 5.2.: Figure 5.2(a) shows an example of an FFST. Figure 5.2(b) shows an equivalent PST built from the FFST of Figure 5.2(a) by means of Algorithm 13. Notice that the FFST in Figure 5.2(a) is consistent as per condition (5.3).

Next we define a prefix tree transducer that is an exact representation of the observed sample S_n and holds the frequencies of the strings.

Definition 26 (Frequency Prefix Tree Subsequential Transducer) *A frequency prefix tree subsequential transducer (*FPTST*) is a 6-tuple* $T =$

$\langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E, F_r \rangle$ *where* $\psi(T) = \langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E, F_r \rangle$ *is a* FST *and* T *is built from a training sample* S_n *such that:*

- $Q = \bigcup_{(u,\mathbf{v}) \in S_n} \{q_x : x \in \mathit{Pref}(u)\}$,
- $E = \{e \mid \mathit{prev}[e] = q_u, \mathit{next}[e] = q_v \Rightarrow v = ua, a \in \Sigma, i[e] = a, o[e] = \epsilon\}$,
- $\forall q_u \in Q, \forall e \in E[q],\ i[e] = \sharp, o[e] = \mathbf{v}$ *if* $(u, \mathbf{v}) \in S_n$, $\perp$ *otherwise,*
- $\forall e \in E[q_0], F_r(e) = |\{a : i[e] = a, a \in \mathit{Pref}(\{u : (u, \mathbf{v}) \in S_n\})\}|$,
- $\forall q_u \in Q \setminus \{q_0\}, \forall e \in E[q_u],\ F_r(e) = |\{a : ua \in \mathit{Pref}(\{x : (x, \mathbf{y}) \in S_n\})\}|$.

A FPTST is said to be in an *onward* form if the following condition holds:

$$\forall q \in Q \backslash \{q_0\}, \mathit{lcp}\left(\bigcup_{e \in E[q]} \{o[e]\}\right) = \epsilon.$$

An FPTST is analogous to a PTST (see page 81). An FPTST is essentially an exact representation of the training data augmented with the observed frequencies of the training data. The frequencies can be utilized to make statistical tests such as Hoeffding bound during the state merging phase of the algorithm.

5.5. The Algorithm

Similar to other state merging algorithms such as RPNI (Oncina and García, 1992), ALERGIA (Carrasco and Oncina, 1999), OSTIA (Oncina and García, 1991; Oncina et al., 1993) and others, the proposed algorithm (Algorithm 18) starts by constructing an initial tree transducer which is the exact representation of the given data. The tree is built in the form of a FPTST (Algorithm 14). Notice that the initial FPTST acts like a rote learner, *i.e.*, the initial tree transducer *overfits* the training data by simply memorizing. Generalization takes place in the state merging phase of the algorithm.

For state merging we use the RED, BLUE notations similar to the ones introduced in (Lang et al., 1998), although the terminology does not exactly coincide with the previous definitions. The recursive framework of merge (Algorithm 15) and fold (Algorithm 17) using the RED, BLUE notations has been adapted from (de la Higuera, 2010). Initially, the root node of the tree is marked as RED state and all its direct children nodes are marked as BLUE. Whenever, a merge is rejected, the BLUE state is promoted to a RED state and consequently the

Algorithm 14: OnwardFptst

Input: a stochastic sample S_n
Output: an Fptst T

$Q \leftarrow \{q_\epsilon\}$;
$q_x \leftarrow q_\epsilon$;
for $(u, \mathbf{v}) \in S_n$ **do**
  Let $u_i \in \{u_i | u_1 \cdot \ldots \cdot u_{|u|} = u, u_i \in \Sigma \cup \{\sharp\}, 1 \geq i \geq |u|\}$;
  for $i = 1$ *to* $i = |u|$ **do**
    if $\exists e \in E[q_x]$ *such that* $i[e] = u_i$ **then**
      for $d \in E[next[e]]$ **do**
        $o[d] \leftarrow lcp(o[e], \mathbf{v})^{-1} o[e] \cdot o[d]$;
      $o[e] \leftarrow lcp(o[e], \mathbf{v})$;
      $\mathbf{v} \leftarrow lcp(o[e], \mathbf{v})^{-1}\mathbf{v}$;
      $F_r(e) \leftarrow F_r(e) + 1$;
    else
      $Q \leftarrow Q \cup \{q_{x \cdot u_i}\}$;
      Let e' be an edge such that:
      $prev[e'] \leftarrow q_x$;
      $next[e'] \leftarrow q_{x \cdot u_i}$;
      $i[e'] \leftarrow u_i$;
      $o[e'] \leftarrow \mathbf{v}$;
      $F_r(e) \leftarrow 1$;
      $E \leftarrow E \cup \{e'\}$;
return T;

direct children nodes of the new RED states are marked as BLUE. The algorithm iteratively tries to merge a RED and BLUE pair of states following the lex-length order. When the first merge is accepted from the initial FFST, some generalization takes place, *i.e.* the hypothesis machine represents a stochastic bi-language which is a superset of the training sample. Similarly, for every merge accepted at i^{th} iteration of the algorithm, the stochastic bi-language represented by the hypothesis PST H_i is a superset of the stochastic bi-language represented by hypothesis PST H_j where $i \leq j$. The algorithm terminates whenever there are no more BLUE states to be merged.

Algorithm 15: StochasticMerge

Input: an FFST T, two states $q \in$ RED and $q' \in$ BLUE
Output: an updated FFST T

1 Let $e \in E$ such that $next\,[e] = q'$;
2 $next\,[e] \leftarrow q$;
3 **if** $q = q_0 \wedge |E\,[q']| = 0$ **then return** $(T, true)$;
4 **return** StochasticFold(T);

Algorithm 16: StatisticalTest

Input: an FFST T, two states $q \in$ RED and $q' \in$ BLUE, a symbol $a \in \Sigma \cup \{\sharp\}$
Output: a boolean

1 $f_1 \leftarrow F_r(e)$ such that $e \in E\,[q] \wedge i\,[e] = a$;
2 $f_2 \leftarrow F_r(e)$ such that $e \in E\,[q'] \wedge i\,[e] = a$;
3 $n_1 \leftarrow \sum_{e \in E[q]} F_r(e)$;
4 $n_2 \leftarrow \sum_{e \in E[q']} F_r(e)$;
5 **return** $\left|\frac{f_1}{n_1} - \frac{f_2}{n_2}\right| < \sqrt{\frac{1}{2}\left(\frac{1}{n_1} + \frac{1}{n_2}\right) \log \frac{2}{\delta}}$;

Notice that the merge decision is made during the recursive fold operation (Algorithm 17). The merge acceptance criteria is checked by the routine StatisticalTest (Algorithm 16) and by compatibility test *w.r.t.* the training data (Algorithm 17, line 7). Suppose we want to merge and fold two states $q_b \in$ BLUE and $q_r \in$ RED. The algorithm APTI2 employs the following condition for a merge to be accepted:

- $\forall e_b \in E\,[q_b]\,, \forall e_r \in E\,[q_r]\,, i\,[e_b] = i\,[e_r] \Rightarrow$

Algorithm 17: StochasticFold

Input: an Ffst T, two states $q \in$ Red and $q' \in$ Blue
Output: an updated Ffst T

```
for a ∈ Σ ∪ {♯} do
    if ¬StatisticalTest(T, q, q', a) then
        return (T, false);
    if ∃e' ∈ E[q'] such that i[e] = a then
        if ∃e ∈ E[q] such that i[e'] = a then
            if  if (next[e] ∈ Red ∧ o[e'] ∉ Pref(o[e])) or (o[e] ≠ o[e'] then
            then
                return (T, false);
            else
                Pushback(T, e, e');
                F_r(e) ← F_r(e) + F_r(e');
                StochasticFold(T);
return (T, true);
```

Algorithm 18: Apti2

Input: a sample S_n
Output: a Pst T

```
T ← OnwardFptst(S_n);
Red ← {q_ε};
Blue ← {q_a : a ∈ Σ ∩ Pref({u : (u, v) ∈ S_n})};
while Blue ≠ ∅ do
    q = Choose_{<lex}(Blue);
    for p ∈ Red in lex-length order do
        (T', IsAccept) ← StochasticMerge(T, p, q);
        if IsAccept then
            T ← T';
        else
            Red ← Red ∪ {q};
            Blue ← {q : ∀p ∈ Red, ∀e ∈ E[p],
                        q = next[e] ∧ q ∉ Red};
ConvertFfstToPst(T);
return T;
```

1. $o\,[e_b] = o\,[e_r]$,
2. inequality (5.2) holds for $f_1 = F_r(e_b)$, $n_1 = \sum_{e \in E[q_b]} F_r(e)$ and $f_2 = F_r(e_r)$, $n_2 = \sum_{e \in E[q_r]} F_r(e)$.

Condition 1, is essentially similar to OSTIA. In order to make merges to happen it is necessary to push the outputs toward the leaf of the tree in the same manner as it is done in OSTIA (see (Oncina et al., 1993; de la Higuera, 2010)). In STOCHASTICFOLD (Algorithm 17) this is achieved using the routine PUSHBACK (line 10) (see page 85 Algorithm 7). The second condition is to check whether the two states are close enough in terms of the distribution of the out going edges. In this case Hoeffding bound is used as a statistical test.

5.6. A Run of the Algorithm

As an example of a stochastic regular bi-language, let us consider the following:

Example 2 *The stochastic regular bi-language* $\mathcal{R} \in \boldsymbol{\mathcal{SREGBL}(\Sigma, \Omega)}$ *where* $Pr_{\mathcal{R}}(a^n b a^m \sharp, \mathbf{x}^n \mathbf{y} \mathbf{x}^m) = \frac{1}{2(3^n 4^m)}, \forall n, m \geq 0$. *and* $Pr_{\mathcal{R}}(u, \mathbf{v}) = 0$ *for every other pairs.*

Figure 5.3 shows the PST in canonical normal form (see page 78) that generates $\mathcal{R}$. We will consider the PST shown in Figure 5.3 as the target PST. The training data *w.r.t.* the target PST (Figure 5.3) is given in Table 5.1.

Next, an onward FPTST is built from the data given in Table 5.1. The FPTST is shown in Figure 5.4. The states q_0 is initiated as a RED state and states q_1 and q_2 are initiated as BLUE states. Here for the Hoeffding bound test, the value of δ is set arbitrarily to 0.5.

The first merge candidate pair of states are q_0 and q_1. The merge between them is accepted and the resulting transducer is shown in Figure 5.5.

Next, the algorithm tries to merge q_0 and q_0 (Figure 5.6). This merge is rejected because the Hoeffding bound test by Algorithm 16 returns false. The state q_2 is promoted to RED and consequently the states q_5 and q_6 are added as RED states (Figure 5.7).

Then the Algorithm tries to merge the states q_0 and q_5 which is also rejected because the Algorithm 16 returns false. Now the next candidate merge pair is q_2 and q_5. This merge is accepted (Figure 5.8).

The next candidate pair of merge is q_0 and q_6 which is accepted and the resulting transducer is depicted in Figure 5.9.

Finally, the FFST is converted into a PST (Figure 5.10) using Algorithm 13 and the algorithm terminates. The inferred PST is shown in Figure 5.10.

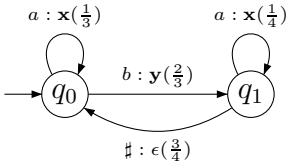


Figure 5.3.: The PST in canonical normal form that generates $\mathcal{R}$ defined in Example 2.

Table 5.1.: Training data for the target PST in Figure 5.3.

input	output	frequency
$b\sharp$	**y**	500
$ab\sharp$	**xy**	160
$ba\sharp$	**x**	120
$aab\sharp$	**xxy**	50
$aba\sharp$	**xyx**	40
total		870

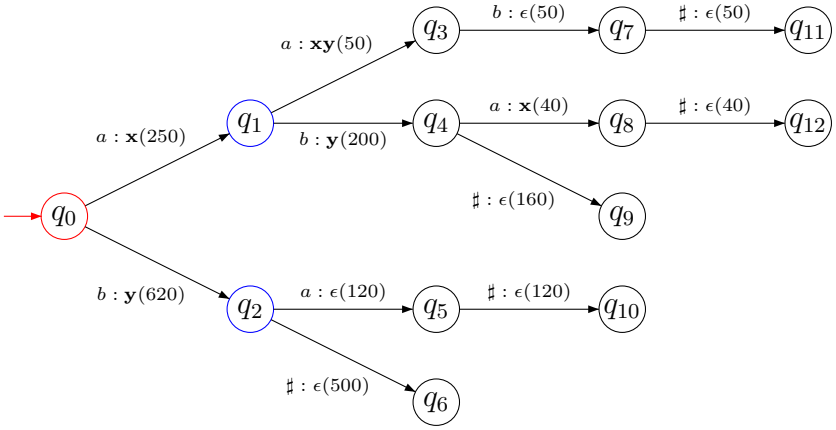


Figure 5.4.: An onward FPTST built from stochastic sample given in Table 5.1.

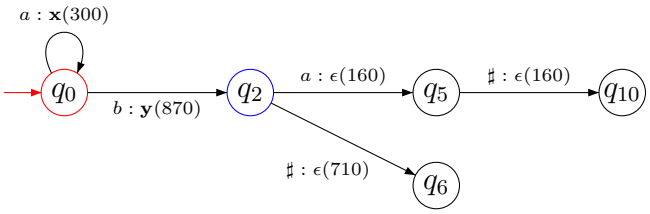


Figure 5.5.: After merging and folding q_0 and q_1.

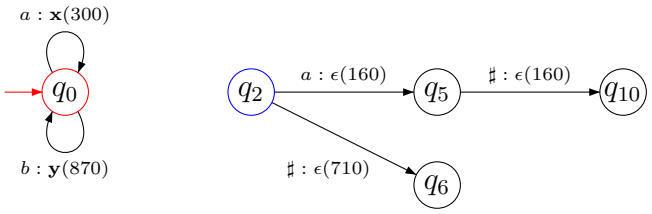


Figure 5.6.: Merge between q_0 and q_2 is rejected.

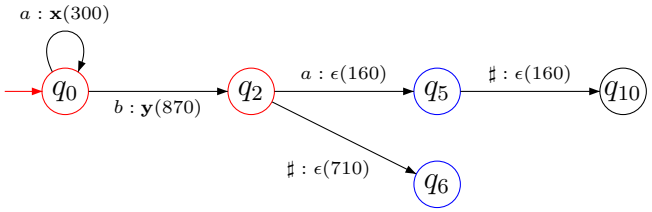


Figure 5.7.: q_2 is promoted to RED and as a consequence of that q_5 and q_6 are added to BLUE.

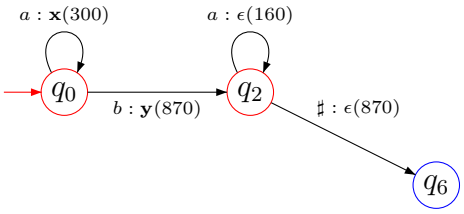


Figure 5.8.: After merging and folding q_2 and q_5.

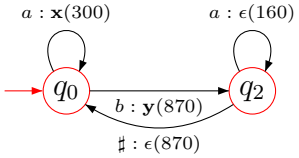


Figure 5.9.: After merging and folding q_0 and q_6.

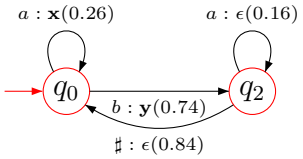


Figure 5.10.: After converting the FFST shown in Figure 5.7 to a PST using Algorithm 13.

5.7. Analysis of the Algorithm

The proposed algorithm APTI2 is inherently an optimistic algorithm. It is optimistic from the view point of the merge strategy. If there is no evidence in the observed data for a pair of candidate states to disallow their merging, the algorithm will simply do the merge. Therefore, while defining the merge strategy of the algorithm, the question to be asked is *"when should two candidate states not be merged?"* rather than the question *"when should two candidate states be merged?"*. The lack of evidence in the data to make the correct merge decisions may lead to two situations: 1) too many merges take place and the algorithm over generalizes; the size of the hypothesis machine is usually smaller than the target machine, 2) too few merges take place, *i.e.*, insufficient evidences or misleading evidences in the data that forbid even the correct merges to take place; the size of the hypothesis machine is usually bigger than the target machine. In both the cases, the data lacks the characteristics of having the evidences that allows the algorithm to infer the correct machine.

As an example, let us consider the run of APTI2 given in the previous section. In this example, if the merge between q_0 and q_2 (see Figure 5.6) were not rejected due to statistical test, the algorithm would perform a bad early merge and as a result infer a wrong hypothesis. On the other hand, if one of the valid merges, *e.g.*, the merge between q_0 and q_1 (see Figure 5.5), were rejected due to statistical test, the algorithm would also lead to a wrong hypothesis. Here we define the properties a stochastic sample must have, for the algorithm to infer the correct machine.

We recall the definition of a stochastic sample $S_n \langle X, f \rangle$ (see page 80) where the frequency of a translation pair $(u, \mathbf{v})$ is given by $f(u, \mathbf{v})$.

Similarly, we define the *prefix frequency* F_{S_n} *w.r.t.* a stochastic sample S_n as the following:

$$F_{S_n}(u\Sigma^*, \mathbf{v}) = |\{(w, \mathbf{x}) : (w, \mathbf{x}) \in S_n \wedge u \in \mathit{Pref}(w)\}| .$$

Less formally, $F_{S_n}(u\Sigma^*, \mathbf{v})$ is the number of translation pairs where the input string starts with the substring u.

We use the definitions above to define the conditions a stochastic sample must obey in order to obtain a correct PST.

A stochastic sample $S_n \langle X, f \rangle$ must satisfy the following conditions in order to learn the syntactic machine *w.r.t.* an SDRT $\mathcal{R}$:

1. $\forall u \in \mathbb{K}, \exists (uw, \mathbf{vx}) \in X$ such that $w \in \Sigma^*, \mathbf{vx} \in \Omega^*, (w, \mathbf{x}) \in (u, \mathbf{v})^{-1}\mathcal{R}$,
2. $\forall u \in \mathbb{SP}, u' \in \mathbb{K}$, if $(u, \mathbf{v})^{-1}\mathcal{R} \neq (u', \mathbf{v}')^{-1}\mathcal{R}$ where $\mathbf{v}, \mathbf{v}' \in \Omega^*$, then any one of the following holds:

a) $\exists (uw, \mathbf{vx}), (u'w, \mathbf{v'x'}), (ur, \mathbf{vy}), (u'r, \mathbf{v'y'}) \in X$ such that for a given value of δ:
$$\left| \frac{\left| f(uw,\mathbf{vx}) - f(ur,\mathbf{vy}) \right|}{F_{S_n}(u\Sigma^*,\mathbf{z})} - \frac{\left| f(u'w,\mathbf{v'x'}) - f(u'r,\mathbf{v'y'}) \right|}{F_{S_n}(u'\Sigma^*,\mathbf{z'})} \right| < \sqrt{\frac{1}{2} \log \frac{2}{\delta} \left(\frac{1}{F_{S_n}(u\Sigma^*,\mathbf{z})} + \frac{1}{F_{S_n}(u'\Sigma^*,\mathbf{z'})} \right)}$$

b) $\exists (uw, \mathbf{vx}), (u'w, \mathbf{v'x'}), \in X$ such that: $(w, \mathbf{x}) \in (u, \mathbf{v})^{-1}\mathcal{R}, (w, \mathbf{x'}) \in (u', \mathbf{v'})^{-1}\mathcal{R}, x \neq x'$

3. $\forall u \in \mathbb{K}, \exists (uw, \mathbf{vx})(uw', \mathbf{vx'}) \in X$ such that $lcp(\mathbf{x}, \mathbf{x'}) = \epsilon \wedge w \neq w'$.

With the conditions above, the properties a stochastic sample S_n have been formalized in order to guarantee the algorithm to learn correctly. The condition 1, 2(b), and 3 are essentially similar to its non-probabilistic counterpart OSTIA (Oncina et al., 1993). Condition 1 is to ensure that there are at least as many states in the learning phase as the target transducer. Condition 2(b) is for making merge decisions *w.r.t.* the output strings. Condition 3 ascertains the alignments of the output strings by factorizing them during the state merging phase. Condition 2(b) is not guaranteed if the transduction scheme is not a total function (Oncina et al., 1993), *i.e.*, it is not sufficient to make the merge decision if the transduction scheme is a partial function. In order to overcome this issue, in our proposed algorithm we have used the frequencies of the given data. Condition 2(a) ensures that the relative frequencies of the observed data is sufficient to distinguish non-mergable states by means of Hoeffding bound (see Section 5.3). The inequality shown in condition 2(a) is from inequality 5.2.

Notice that condition 2(a) depends on the value of δ. Carrasco and Oncina have discussed how a large or a small value of δ effects the merge decision in (1994). Basically, if the size of the stochastic sample S_n is significantly large, one could keep δ negligibly small. On the contrary, for a relatively small size of stochastic sample S_n, δ requires to be sufficiently high. We will show some experimental results in Section 5.8 with different values of δ to show how the choice of δ influences the learning outcome.

The runtime complexity of algorithm APTI2 is given by $\mathcal{O}((\|S_n\|)^3(m+n) + \|S_n\| mn)$, where:

- $\|S_n\| = \sum_{(u,\mathbf{v}) \in S_n} |u|$
- $m = max\{|u| : (u, \mathbf{v}) \in S_n\}$.

The FPTST can be built in linear time *w.r.t.* $\|S_n\|$ (Algorithm 14). We will now analyze the outermost while loop in the APTI2 algorithm. Being pessimistic,

there will be at most $\|S_n\|$ number of states in the FPTST. In the worse case, if no merges are accepted, there will be $\mathcal{O}(\|S_n\|)$ executions of the outer most while loop and $\mathcal{O}(\|S_n\|^2)$ executions of the inner for loop, resulting $\mathcal{O}(\|S_n\|^3)$ executions of the core algorithm. In each of these executions, *lcp* operation can be implemented in $\mathcal{O}(m)$ times and the pushback operation in $\mathcal{O}(n)$ times. Assuming that all arithmetic operations are computed in unit time, the total core operation of APTI2 can be bounded by $\mathcal{O}((\|S_n\|)^3(m+n)+\|S_n\|\,mn)$. This runtime complexity is pessimistic and the runtime of APTI2 is much lower in practice. Experimental evidence of runtime of APTI2 is presented in the next section.

5.8. Experimental results

5.8.1. Data

The experiments have been conducted using synthetic data generated from random PSTs. The random transducer construction procedure is fundamentally similar to the random PFA construction procedure in the PAutomaC[1] competition (Verwer et al., 2012) with some modifications. The construction procedure of a random PST takes the following parameters:

- An integer N as the number of states;
- Σ and Ω, the sets of input and output symbols respectively;
- An integer k as the maximum output length, *i.e.*, the output strings are in $\Omega^{\leq k}$;
- Two integers max and min as a range such that $max > min$ in order to generate random probabilities for the transitions.

First, a random DFA is generated in the minimal form of size N by connecting the states randomly with input symbols. Each state is randomly selected as a final state with probability 0.5. An extra edge with symbol $\sharp$ is added for each of the final states going to the initial state. Second, for each of the edges, a random input string is generated with maximum length of k. To generate a random string, we randomly draw a symbol $\mathbf{a} \in \Omega$ uniformly with replacement n times, where n is a random number such that $0 \leq n \leq k$. After drawing n symbols randomly, the symbols are concatenated to form the output string. If

[1] `http://ai.cs.umbc.edu/icgi2012/challenge/Pautomac/index.php`

the random number $n = 0$, then the output string is ϵ. Finally, to assign the probabilities of the edges of each state q, we generate $|E[q]|$ random numbers ranging from min to max using a uniform distribution. After that the transducer is normalized using Algorithm 13. Notice that the lower is the difference between max and min, the more uniform will be the distribution of the states, in case of $max = min$ the distribution will be uniform.

Using the target PST, the training sample is generated following the paths of the PST. The test data is also generated in the similar manner. In order to test the algorithm with unseen examples, we make sure that the test set and the training set are disjoint.

5.8.2. Evaluation

As a measure of correctness we compute two metrics: word error rate (WER) and sentence error rate (SER). Intuitively, WER is the percentage of symbol error in the hypothesis translation *w.r.t.* the reference translation. For each test pair, the Levenshtein distance (Levenshtein, 1966) between the reference translation and hypothesis translation is computed and then the Levenshtein distance is divided by the length of the reference string. The mean of the scores computed for each test pair is reported as the WER. On the other hand SER is more strict; it is the percentage of wrong hypothesis translations *w.r.t.* the reference translations. The number of times the hypothesis translation does not match the reference translation is counted, then it is divided by the number of test pairs and the score is reported as SER.

Figure 5.11(a) and Figure 5.11(b) show the results of our first experiment. The objectives of the first experiment are to demonstrate the correctness of our algorithm and to show that the runtime in practice. Figure 5.11(a) shows experiments conducted on randomly generated PSTs with 5 states and $|\Sigma| = |\Omega| = 2$. We start with a training sample size 200, we keep incrementing the training size by 200 up to a size of 10000. For every training size the experiment is repeated 10 times by generating new datasets. The mean of these 10 experimental results is reported. We have conducted the experiment for 10 random PSTs. Thus, in total we have conducted 5000 trials. Figure 5.11(a) shows the mean of the results obtained from 10 random PSTs. As the Figure 5.11(a) shows, the error rate is approaching zero. As expected, the WER in most cases remains below SER. The execution time for this experiment is reported in Figure 5.11(b).

The results of our second experiment are shown in 5.11(c) and Figure 5.11(d). In this experiment, the objective was to demonstrate the correctness and runtime of APTI2 for PSTs bigger number of states. In the second experiment we have taken a randomly generated PSTs with 10 states and $|\Sigma| = |\Omega| = 2$. We start

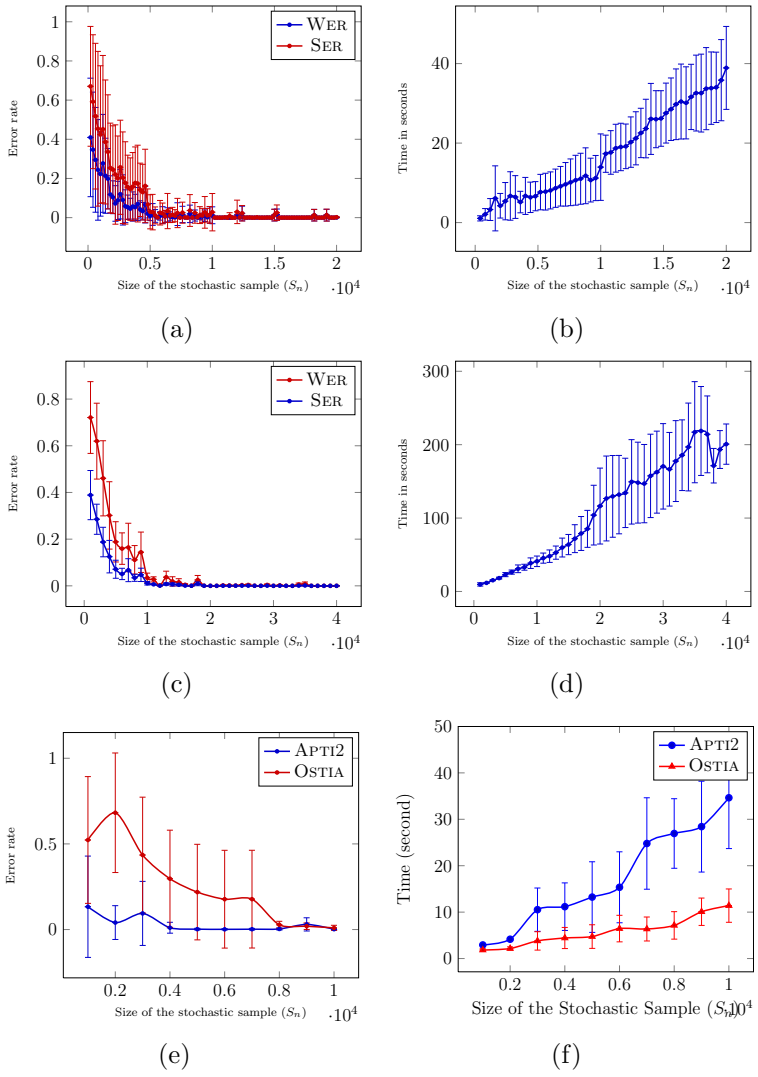


Figure 5.11.: Performance of APTI2 reported for the artificially generated data set (Figures 5.11(a), 5.11(b), 5.11(c), and 5.11(d)) and the data set for the MLA task (Figures 5.11(e), 5.11(f)).

with a training sample size 1000, we keep incrementing the training size by 1000 up to a size of 40000. Similar to the previous experiment, experiment with each training sample size is repeated 10 times. The experiment is conducted for 10 random PSTs. The results are shown in Figures 5.11(c) and 5.11(d).

Finally, we have conducted another experiment with the Feldman dataset (see page 4.15.1). The objectives of this experiment are 1) to do a comparison of

prediction accuracy of OSTIA and APTI2 and 2) a comparison of runtime of OSTIA and APTI2. Here we start with 1000 training pairs and incremented by 1000 till 10000 training pairs. Each data point is repeated 10 times for statistical significance. The results are depicted in Figure 5.11(e) and 5.11(f). The results of Figure 5.11(e) demonstrate significant improvement in prediction accuracy of APTI2 in comparison with OSTIA.

We have implemented APTI2 using an open source C++ library for weighted transducers (Allauzen et al., 2007). Figures 5.11(b), 5.11(d), and 5.11(f) shows the execution times for APTI2 using our implementation. Although, the theoretical worst case runtime complexity of APTI2 is cubic, in practice APTI2 exhibits much lower execution time.

5.9. Summary

In this chapter we have endeavored to solve the problem of learning PSTs from positive examples, which is one of the research questions of the thesis formulated as **RQ**5 (see page 10). We have presented a learning algorithm APTI2 that learns any PST provided a characteristic training sample is given. We have also presented experimental results based on synthetic data to proof the correctness of our algorithm. Moreover, based on our implementation, we have reported that the runtime complexity of APTI2 in practice is much lower than the theoretical worst case runtime complexity.

The limitation of our work is twofold: first, our model is restricted to regular stochastic bi-grammar, and hence not capable of capturing many practical scenarios, *e.g.*, natural languages. Second, as a statistical test we have used a basic Hoeffding bound. We believe that more sophisticated statistical tests will lead to better accuracy of the algorithm.

Chapter 6

Conclusion

6.1. Overview

We started the thesis with several motivating examples where transducers are used in real life. The theoretical results and algorithms we have developed in this thesis are all having these potential applications in mind. Within the scope of the thesis, a possible application that has been experimentally demonstrated is: the potential application of transducer learning in an automated machine translation task. The experiments have been reported based on synthetic data. We believe that the algorithms presented here can also be adapted to other application scenarios presented in Chapter 1 with little efforts.

The objectives of the thesis were defined as five research questions (**RQ**s) in Chapter 1. We attempted to solve the **RQ**s throughout the thesis and have presented our findings and results. Here we will summarize the contributions of the thesis:

6.1.1. Chapter 2

The contributions in Chapter 2 are primarily theoretical. In this chapter our aim was to provide solutions to **RQ**1, **RQ**2, and **RQ**3 defined in Chapter 1. The results and findings of this chapter are summarized as the following:

Research Question 1 ($RQ1$)

The objective of **RQ**1 is to examine the relative expressive power of different types of probabilistic transducers. In other words, the objective is to provide

a hierarchical view of stochastic bi-languages generated by different types of syntactic machines. We have presented three hierarchies:

- The hierarchy of bi-languages modeled by finite state machines (Chapter 2 Figure 2.11, page 30).
- The hierarchy of stochastic bi-languages modeled by finite state machines (Chapter 2 Figure 2.20, page 40).
- The hierarchy of different types of probabilistic transducers and other probabilistic models in terms of their relative expressive power (Chapter 2 Figure 2.24, page 62).

Research Question 2 (*RQ*2)

RQ2 concerns the runtime complexities of computation problems *w.r.t.* a probabilistic transducer. Depending on the area of application, there can be various types of computation requirements using the same syntactic machine. We have provided general definitions of several possible types of computations using probabilistic transducers independent of the application areas. We have presented runtime complexities of these computation tasks using each type of probabilistic transducers we have defined. As by-products of **RQ**2 we have developed novel algorithms for parsing using probabilistic transducers.

Research Question 3 (*RQ*3)

RQ3 is about comparison of our learning model *w.r.t.* the other existing models. We have shown that PFSTs are equivalent to another widely used model, namely PHMMs under certain conditions. Moreover, we have also showed that PFSTs are a specific use of WFSTs which is another well known model and widely applied.

All in all, the results presented in Chapter 2 provide supports in favor of our learning model: probabilistic subsequential transducers (PSTs). It shows, although PSTs are not the most powerful machines *w.r.t.* expressive power, they are relatively inexpensive *w.r.t.* parsing. Moreover, the comparison results to other existing models support the generality of our model.

6.1.2. Chapter 3

In Chapter 3 we have presented an extensive literature review regarding transducer learning. Our survey shows that there have been three waves of transducer learning. The early work involves sequential models, which are deterministic

models and subsets of subsequential models (see Chapter 2). The second wave starts with the commencement of the well known OSTIA algorithm which is capable of learning subsequential transducers that represent total functions. A number of follow ups of OSTIA algorithm emerged in the last decade of the 20th century and continued till the middle of the first decade of the 21st century. As the third wave of transducer learning, some recent attempts have been made to learn transducers which have got more expressive power than a subsequential transducer. The third wave continues till today. The problem of learning PSTs is also an open problem till date.

6.1.3. Chapter 4

In this chapter we have worked on **RQ**4, which involves identification of PSTs in the limit. The results presented in this chapter are summarized as the following:

- We have shown a novel approach of asking queries *w.r.t.* the observed data, which is traditionally not the case in the field of grammatical inference.
- We have presented a novel algorithm for learning PSTs, which is a hybridization of active learning and state merging paradigm.
- We have proved that PSTs are identifiable in the limit with probability 1 from positive presentation and probabilistic queries *w.r.t.* the domain language.
- We have overcome the limitation of total function of OSTIA by introducing the notion of phantoms.
- We have presented experimental results with synthetic data to backup the correctness of our algorithm.

6.1.4. Chapter 5

In this chapter we have endeavored **RQ**5, which is about learning PSTs from positive data only. The idea is to substituting the oracle by an empirical distribution of the positive data to learn PSTs. We have presented a novel algorithm that uses the frequencies of the given data to learn the target PST. This algorithm works as a support for the theoretical results in Chapter 4 to illustrate the practicability of our approach. The contributions of this chapter are summarized as the following:

- We have presented a novel algorithm for learning PSTs from empirical distribution of positive data.
- We have formalized the conditions under which learning is guaranteed.
- We have presented experimental results using synthetic data to support the correctness of our algorithm.

We have implemented the proposed learning algorithms APTI and APTI1 using an open source C++ library called OpenFST (Allauzen and Mohri, 2002) and conducted our experiments using the implementations.

6.1.5. Other Contributions

Besides the main contributions of the thesis, we have presented a parallel algorithm for learning DFAs from positive and negative data and have shown significant speedup by experimentations in Appendix A. The algorithm presented is essentially the parallel counterpart of the basic RPNI (Oncina and García, 1992). The parallel algorithm is designed in the master-slave paradigm. We believe that our approach of parallelization can easily be adapted for the other algorithms presented in this thesis.

Moreover, we have implemented an open source library for grammatical inference algorithm in MATLAB and reported the feasibility and performance in Appendix B. This toolbox for grammatical inference allows easy and intuitive experimentations, comparisons and parallelizations of grammatical inference algorithms for learning automata and transducers.

6.2. Open Problems

We started with five **RQ**s in the beginning of the thesis and while pursuing the **RQ**s, new questions, problems and intuition for alternative solutions came into our mind. In order to stay focused toward the thesis objectives, we decided to put those new questions, problems and intuition aside as *open problems* for future work. In this section, we will highlight some of those open problems.

Open Problem 1 *Can we improve the prediction accuracy by asking probabilistic queries w.r.t. the output language?*

This problem involves asking queries about the output languages and using that information to improve prediction accuracy. This problem can be looked at

from two settings: theoretical setting (the queries are asked to an oracle) and practical setting (the queries are simulated by statistical tests). The feasibility of the later setting depends on the first one. The issues in the theoretical setting are: first, since PSTs are non-deterministic *w.r.t.* the output and have ϵ outputs, how to compute probabilistic queries *w.r.t.* the output strings; second, how to utilize such queries in a learning algorithm to gain prediction accuracy. Having such issues resolved in the theoretical setting should allow us to develop another algorithm by using statistical tests in order to make the approach practicable.

Open Problem 2 *Is it possible to find an algorithm for learning* PSTs *for which the characteristic sample is highly probable to appear in a stochastic sample* S_n*?*

The proposed algorithm APTI requires the presence of a characteristic sample in the training data to learn the PST. However, the definition of the characteristic sample does not tell us what is the probability that a given stochastic sample will contain a characteristic sample. This open problem aims to find a learning algorithm for which the characteristic sample is less demanding and more probable to appear in a training data of relatively smaller size.

6.3. A Conjecture

Based on the lessons learnt from the thesis work and the deterministic properties of a probabilistic p-subsequential transducer, we believe that identification of probabilistic p-subsequential transducers in the limit is plausible. We postulate the following conjecture:

Conjecture 1 *There exists an algorithm* Φ *such that* $\forall \mathcal{R} \in \boldsymbol{\mathcal{SPREGBL}(\Sigma, \Omega)}$, Φ *identifies the probabilistic* p*-subsequential transducers* T *with probability* 1 *from positive presentation and* EXPQ *where* T *generates* $\mathcal{R}$*.*

We believe that similar state merging strategies used in learning PSTs with some additional measures for probabilistic p-subsequential transducers will allow us to achieve this. However, it still requires formal proof and experimental evidences for practicability, and therefore, it remains a conjecture.

6.4. Epilogue

We will conclude the thesis by making the following remarks based on the experience gained through our work: first, the problem of learning probabilistic

transducers is a relatively young branch of grammatical inference with a lot of unsolved problems and potential applications. We believe the contributions of this thesis will surely open doors for further investigations. We also believe that the open problems illustrated in this chapter will help leading the problem domain toward the next steps.

Bibliography

Alfred V. Aho and Jeffrey D. Ullman. Syntax directed translations and the pushdown assembler. *Journal of Computer and System Sciences*, 3(1):37–56, 1969.

Hasan Ibne Akram, Alban Batard, Colin de la Higuera, and Claudia Eckert. Psma: A parallel algorithm for learning regular languages. In *Proceedings of NIPS Workshop on Learning on Cores, Clusters and Clouds*, 2010a.

Hasan Ibne Akram, Colin de la Higuera, Huang Xiao, and Claudia Eckert. Grammatical inference algorithms in matlab. In *Proceedings of ICGI*, volume 6339 of *Lecture Notes in Computer Science*, pages 262–266. Springer-Verlag, 2010b.

Cyril Allauzen and Mehryar Mohri. p-subsequentiable transducers. In *Proceedings of CIAA*, volume 2608 of *Lecture Notes in Computer Science*, pages 24–34. Springer-Verlag, 2002.

Cyril Allauzen and Mehryar Mohri. Efficient algorithms for testing the twins property. *Journal of Automata, Languages and Combinatorics*, 8(2):117–144, 2003a.

Cyril Allauzen and Mehryar Mohri. Finitely subsequential transducers. *International Journal of Foundations of Computer Science*, 14(6):983–994, 2003b.

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. Openfst: A general and efficient weighted finite-state transducer library. In *Proceedings of CIAA*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer-Verlag, 2007.

Ross Anderson. *Security Engineering*. John Wiley & Sons, 2008.

Dana Angluin. A note on the number of queries needed to identify regular languages. *Information and Control*, 51(1):76–87, 1981.

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

Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987b.

Dana Angluin and Leonor Becerra-Bonache. Experiments using ostia for a language production task. In *Proceedings of the EACL*, CLAGI '09, pages 16–23. Association for Computational Linguistics, 2009.

Dana Angluin and Carl H. Smith. Inductive inference: Theory and methods. *ACM Computer Surveys*, 15(3):237–269, 1983.

Domagoj Babić, Daniel Reynaud, and Dawn Song. Malware analysis with tree automata inference. In *Proceedings of the Computer aided verification*, Lecture Notes in Computer Science, pages 116–131. Springer-Verlag, 2011.

Domagoj Babić, Daniel Reynaud, and Dawn Song. Recognizing malicious software behaviors with tree automata inference. *Formal Methods in System Design*, pages 1–22, 2012.

José L. Balcázar, Josep Díaz, Ricard Gavaldà, and Osamu Watanabe. An optimal parallel algorithm for learning dfa. In *Proceedings of COLT '94: the seventh annual conference on Computational learning theory*, pages 208–217. ACM Press, 1994.

Borja Balle, Ariadna Quattoni, and Xavier Carreras. A spectral learning algorithm for finite state transducers. In *Proceedings of ECML/PKDD Part I*, volume 6911 of *Lecture Notes in Computer Science*, pages 156–171. Springer-Verlag, 2011.

Srinivas Bangalore and Giuseppe Riccardi. A finite-state approach to machine translation. In *Proceedings of the North American Chapter of the Association for Computational Linguistics*, pages 1–8. Association for Computational Linguistics, 2001.

Srinivas Bangalore and Giuseppe Riccardi. Stochastic finite-state models for spoken language machine translation. *Machine Translation*, 17(3):165–184, 2002.

Leonard E Baum, Ted Petrie, Goerge Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

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

Joseph Berger and Claude Pair. Inference for regular bilanguages. *Journal of Computer and System Sciences*, 16(1):100–122, 1978.

Jean Berstel. *Transductions and Context-Free Languages*. Teubner Studienbücher, 1979.

Jean Berstel and Christophe Reutenauer. *Rational series and their languages*. Springer-Verlag, 1988.

Robert K. Bradley and Ian Holmes. Transducers: an emerging probabilistic framework for modeling indels on trees. *Bioinformatics*, 23(23):3258–3262, 2007.

Laurent Bréhélin, Olivier Gascuel, and Gilles Caraux. Hidden markov models with patterns to learn boolean vector sequences and application to the built-in self-test for integrated circuits. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(9):997–1008, 2001.

Julien Carme, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Interactive learning of node selecting tree transducer. *Machine Learning*, 66(1):33–67, 2007.

Rafael C. Carrasco and José Oncina. Learning stochastic regular grammars by means of a state merging method. In *Proceedings of ICGI*, volume 862 of *Lecture Notes in Computer Science*, pages 139–152. Springer-Verlag, 1994.

Rafael C. Carrasco and José Oncina. Learning deterministic regular grammars from stochastic samples in polynomial time. *Rairo - Theoratical Informatics and Applications*, 33(1):1–20, 1999.

Francisco Casacuberta. Some relations among stochastic finite state networks used in automatic speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):691–695, 1990.

Francisco Casacuberta. Probabilistic estimation of stochastic regular syntax-directed translation schemes. In A. Calvo and R. Medina, editors, *Proceedings of VI Spanish Symposium on Pattern Recognition and Image Analysis*, pages 201–207. AERFAI, 1995.

Francisco Casacuberta. Inference of finite-state transducers by using regular grammars and morphisms. In *Proceedings of ICGI*, volume 1891 of *Lecture Notes in Computer Science*, pages 1–14. Springer-Verlag, 2000.

Francisco Casacuberta. Finite-state transducers for speech-input translation. In *IEEE Automatic Speech Recognition and Understanding Workhsop, ASRU01*. IEEE Press, 2001.

Francisco Casacuberta and Colin de la Higuera. Computational complexity of problems on probabilistic grammars and transducers. In *Proceedings of ICGI*, volume 1891 of *Lecture Notes in Computer Science*, pages 15–24. Springer-Verlag, 2000.

Francisco Casacuberta and Enrique Vidal. Machine translation with inferred stochastic finite-state transducers. *Computational Linguistics*, 30(2):205–225, 2004.

Francisco Casacuberta and Enrique Vidal. Learning finite-state models for machine translation. *Machine Learning*, 66(1):69–91, 2007.

George Casella and Roger Berger. *Statistical Inference*. Duxbury Resource Center, 2001.

Antonio Castellanos, Enrique Vidal, Miguel Angel Varó, and José Oncina. Language understanding and subsequential transducer learning. *Computer Speech & Language*, 12(3):193–228, 1998.

Chia Yuan Cho, Domagoj Babić, Eui Chul Richard Shin, and Dawn Song. Inference and analysis of formal models of botnet command and control protocols. In *Proceedings of ACM Conference on Computer and Communications Security*, pages 426–439. ACM, 2010a.

Chia Yuan Cho, Juan Caballero, Chris Grier, Vern Paxson, and Dawn Song. Insights from the Inside: A View of Botnet Management from Infiltration. In *Proceedings of the 3rd USENIX Workshop on Large-Scale Exploits and Emergent Threats*. USENIX Association, 2010b.

Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124, 1956.

Noam Chomsky. *Syntactic Structures*. Mouton, 1957.

Alexander Clark. Partially supervised learning of morphology with stochastic transducers. In *Proceedings of NLPRS*, pages 341–348, 2001a.

Alexander Clark. Learning morphology with pair hidden Markov models. In *Proceedings of the Student Workshop at the Annual Meeting of the ACL*, pages 55–60. The Association for Computer Linguistics, 2001b.

Alexander Clark. Memory-based learning of morphology with stochastic transducers. In *Proceedings of ACL*, pages 513–520. The Association for Computer Linguistics, 2002.

Alexander Clark. Large scale inference of deterministic transductions: Tenjinno problem 1. In *Proceedings of ICGI*, volume 4201 of *Lecture Notes in Computer Science*, pages 227–239. Springer-Verlag, 2006.

Alexander Clark. Towards general algorithms for grammatical inference. In *Proceedings of ALT*, volume 6331 of *Lecture Notes in Computer Science*, pages 11–30. Springer-Verlag, 2010.

Alexander Clark. Inference of inversion transduction grammars. In *Proceedings of ICML*, pages 210–208. Omnipress, 2011.

Alexander Clark and Shalom Lappin. *Linguistic Nativism and the Poverty of the Stimulus*. Wiley-Blackwell, 2011.

Corinna Cortes and Mehryar Mohri. Learning with weighted transducers. In *Proceedings of FSMNLP*, volume 19 of *Frontiers in Artificial Intelligence and Applications*, pages 14–22. IOS Press, 2008.

Corinna Cortes, Patrick Haffner, and Mehryar Mohri. Rational kernels. In *Proceedings of NIPS*, pages 601–608. MIT Press, 2002.

François Coste. Tutorial on modelling biological sequences by grammatical inference: Bibliography, 2010. URL `http://www.irisa.fr/symbiose/people/fcoste/pub/biblio_tutoICGI2010_coste.pdf`.

Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.

Colin de la Higuera and José Oncina. Learning stochastic finite automata. In *Proceedings of ICGI*, volume 3264 of *Lecture Notes in Computer Science*, pages 175–186. Springer-Verlag, 2004.

Colin de la Higuera and José Oncina. Finding the most probable string and the consensus string: an algorithmic study. In *Proceedings of IWPT*, pages 26–36. The Association for Computational Linguistics, 2011.

Colin de la Higuera, José Oncina, and Enrique Vidal. Identification of dfa: data-dependent vs data-independent algorithms. In *Proceedings of ICGI*, volume 1147 of *Lecture Notes in Computer Science*, pages 313–325. Springer-Verlag, 1996.

Thomas Dean, Dana Angluin, Kenneth Basye, Sean Engelson, Leslie Kaelbling, Evangelos Kokkevis, and Oded Maron. Inferring finite automata with stochastic output functions and an application to map learning. *Machine Learning*, 18(1):81–108, 1995.

George Doddington. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the ARPA Workshop on Human Language Technology*. Morgan Kaufmann, 2002.

Manfred Droste and Werner Kuich. *Handbook of Weighted Automata*, chapter Semiring and formal power series, pages 213–254. Monographs in Theoretical Computer Science, an EATCS Series. Springer-Verlag, 2009.

Werner; Vogler Heiko Droste, Manfred; Kuich. *Handbook of Weighted Automata*. Monographs in Theoretical Computer Science. an EATCS Series. Springer-Verlag, 2009.

Devdatt Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 1st edition, 2009.

P. Dupont, F. Denis, and Y. Esposito. Links between probabilistic automata and hidden markov models: probability distributions, learning models and induction algorithms. *Pattern Recognition*, 38:1349–1371, 2005.

Pierre Dupont. Regular grammatical inference from positive and negative samples by genetic search: the gig method. In *Proceedings of ICGI*, volume 862 of *Lecture Notes in Computer Science*, pages 236–245. Springer-Verlag, 1994.

Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

Claudia Eckert. *IT-Sicherheit - Konzepte, Verfahren, Protokolle (6. Aufl.)*. Oldenbourg, 2009.

Jason Eisner. Expectation semirings: Flexible em for finite-state transducers. In *Proceedings of the ESSLLI Workshop on Finite-state Methods in Natural Language Processing*, 2001.

Jason Eisner. Parameter estimation for probabilistic finite-state transducers. In *Proceedings of ACL*, pages 1–8. The Association for Computational Linguistics, 2002.

Jerome A. Feldman, George Lakoff, Andreas Stolcke, and Susan H. Weber. Miniature language acquisition: A touchstone for cognitive science. Technical Report TR-90-009, International Computer Science Institute, Berkeley CA, 1990.

William Feller. *An introduction to probability theory and its applications. Vol. I & II.* Second edition. John Wiley & Sons Inc., 1971.

Pedro García and Enrique Vidal. Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):920–925, 1990.

Pedro García, Manuel Vazquez de Parga, Damián López, and José Ruiz. Learning automata teams. In *Proceedings of ICGI*, volume 6339 of *Lecture Notes in Computer Science*, pages 52–65. Springer-Verlag, 2010.

E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

Jorge González, Germán Sanchis-Trilles, and Francisco Casacuberta. Learning finite state transducers using bilingual phrases. In *Proceedings of CICLing*, volume 4919 of *Lecture Notes in Computer Science*, pages 411–422. Springer-Verlag, 2008.

Jonathan Graehl, Kevin Knight, and Jonathan May. Training tree transducers. *Computational Linguistics*, 34(3):391–427, 2008.

Anshul Gupta and Vipin Kumar. Performance properties of large scale parallel systems. *Journal of Parallel and Distributed Computing*, 19(3):234–244, 1993.

Thomas Hanneforth. A memory-efficient epsilon-removal algorithm for weighted acyclic finite-state automata. In *Proceedings of FSMNLP*, volume 19 of *Frontiers in Artificial Intelligence and Applications*, pages 72–81. IOS Press, 2008.

Thomas Hanneforth and Colin de la Higuera. Epsilon-removal by loop reduction for finite-state automata. *Language and Logos*, 72:297–312, 2010.

Marijn J. H. Heule and Sicco Verwer. Exact dfa identification using sat solvers. In *Proceedings of ICGI*, volume 6339 of *Lecture Notes in Computer Science*, pages 66–79. Springer-Verlag, 2010.

Wassilij Hoeffding. Probability inequalities for sums of bounded random variables. *American Statistical Association Journal*, 58:13–30, 1963.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman, 2006.

S. Jain, D. Osherson, J. Royer, A. Sharma, and S. Weinstein. *Systems That Learn*. MIT Press, 1999.

C. Douglas Johnson. *Formal Aspects of Phonological Description*. Mouton, 1972.

Lauri Karttunen. Applications of finite-state transducers in natural language processing. In Sheng Yu and Andrei Paun, editors, *Proceedings of CIAA*, volume 2088 of *Lecture Notes in Computer Science*, pages 34–46. Springer-Verlag, 2000.

Kevin Knight and Jonathan Graehl. An overview of probabilistic tree transducers for natural language processing. In *Proceedings of CICLing*, volume 3406 of *Lecture Notes in Computer Science*, pages 1–24. Springer-Verlag, 2005.

Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, 1st edition, 2010.

András Kornai. *Outstanding Dissertations in Linguistics*, chapter Formal Phonology. Garland Publishing, 1995.

András Kornai. *Mathematical Linguistics (Advanced Information and Knowledge Processing)*. Springer-Verlag, 2007.

Werner Kuich. Handbook of formal languages. volume 1, chapter Semirings and formal power series: their relevance to formal languages and automata, pages 609–677. Springer-Verlag, 1997.

Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages*, volume 5 of *Monographs in Theoretical Computer Science. An EATCS Series*. Springer-Verlag, 1986.

Kevin J. Lang and Eric B. Baum. Query learning can work poorly when a human oracle is used. In *Proceedings of IJCNN*, pages 335–340. IEEE Press, 1992.

Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. In *Proceedings of ICGI*, volume 1433 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, 1998.

Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707, 1966.

Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall PTR, 2nd edition, 1997.

Zhifei Li and Jason Eisner. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proceedings of EMNLP*, pages 40–51. The Association for Computational Linguistics, 2009.

Simon M. Lucas. Evolving finite state transducers: Some initial explorations. In *EuroGP*, volume 2610 of *Lecture Notes in Computer Science*, pages 130–141. Springer-Verlag, 2003.

Simon M. Lucas and T. Jeff Reynolds. Learning dfa: evolution versus evidence driven state merging. In *Proceedings the IEEE Congress on Evolutionary Computation*, pages 351–358. IEEE, 2003.

Simon M. Lucas and T. Jeff Reynolds. Learning deterministic finite automata with a smart state labeling evolutionary algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1063–1074, 2005.

Simon M. Lucas and T. Jeff Reynolds. Learning finite-state transducers: Evolution versus heuristic state merging. *IEEE Transactions on Evolutionary Computation*, 11(3):308–325, 2007.

P. Luneau, M. Richetin, and C. Cayla. Sequential learning of automata from input-output behaviour. *Robotica*, 1(03):151–159, 1983.

Wesley Mackay and Grzegorz Kondrak. Computing word similarity and identifying cognates with pair hidden markov models. In *Proceedings of CONLL*, pages 40–47. The Association for Computational Linguistics, 2005.

Andreas Maletti. A tree transducer model for synchronous tree-adjoining grammars. In *Proceedings of ACL*, pages 1067–1076. The Association for Computer Linguistics, 2010.

MathWorks. Matlab - the language of technical computing. 2010. http://www.mathworks.com/products/matlab.

George H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955.

Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

Mehryar Mohri. Generic epsilon-removal algorithm for weighted automata. In *Proceedings of CIAA*, volume 2088 of *Lecture Notes in Computer Science*, pages 230–242. Springer-Verlag, 2000a.

Mehryar Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234(1-2):177–201, 2000b.

Mehryar Mohri. Generic e-removal and input e-normalization algorithms for weighted transducers. *International Journal of Foundations of Computer Science*, 13(1):129–143, 2002a.

Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata Language and Combnatorics*, 7:321–350, 2002b.

Mehryar Mohri. *Applied Combinatorics on Words*, chapter Statistical Natural Language Processing, pages 199–225. Cambridge University Press, 2005.

Mehryar Mohri. *Handbook of Weighted Automata*, chapter Weighted automata algorithms, pages 213–254. Monographs in Theoretical Computer Science, an EATCS Series. Springer-Verlag, 2009.

Mehryar Mohri, Fernando Pereira, O Pereira, and Michael Riley. Weighted automata in text and speech processing. In *Proceedings of ECAI-96 Workshop*, pages 46–50. John Wiley and Sons, 1996.

Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

Mehryar Mohri, Cyril Allauzen, and Michael Riley. Statistical modeling for unit selection in speech synthesis. In *Proceedings of ACL*, pages 55–62. Association for Computational Linguistics, 2004.

Edward F. Moore. Gedanken-experiments on sequential machines. In Claude Shannon and John McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, 1956.

Peter Nabende. Transliteration system using pair hmm with weighted fsts. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration*, NEWS '09, pages 100–103. Association for Computational Linguistics, 2009.

Peter Nabende, Jörg Tiedemann, and John Nerbonne. Pair hidden markov model for named entity matching. In *Volume I of the proceedings of the 2008 International Conference on Systems, Computing Sciences and Software Engineering (SCSS)*, pages 497–502, 2008.

José Oncina. The data driven approach applied to the ostia algorithm. In *Proceedings of ICGI*, volume 1433 of *Lecture Notes in Computer Science*, pages 50–56. Springer-Verlag, 1998.

José Oncina. Using multiplicity automata to identify transducer relations from membership and equivalence queries. In *Proceedings of ICGI*, volume 5278 of *Lecture Notes in Computer Science*, pages 154–162. Springer-Verlag, 2008.

José Oncina and Pedro García. Inductive learning of subsequential functions. Technical report, Univ. Polit'ecnica de Valencia, 1991.

José Oncina and Pedro García. Identifying regular languages in polynomial time. In H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition*, volume 5 of *Series in Machine Perception and Artificial Intelligence*, pages 99–108. World Scientific, 1992.

José Oncina and Miguel Angel Varó. Using domain information during the learning of a subsequential transducer. In *Proceedings of ICGI*, volume 1147 of *Lecture Notes in Computer Science*, pages 301–312. Springer-Verlag, 1996.

José Oncina, Pedro García, and Enrique Vidal. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):448–458, 1993.

Lior Pachter, Marina Alexandersson, and Simon Cawley. Applications of generalized pair hidden markov models to alignment and gene finding problems. *Journal of Computational Biology*, 9(2):389–399, 2002.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Procceedings of ACL*, pages 311–318. The Association for Computer Linguistics, 2002.

Azaria Paz. *Introduction to probabilistic automata*. Academic Press, 1971.

Fernando Pereira, Michael Riley, and Richard Sproat. Weighted rational transductions and their application to human language processing. In *Proceedings of HLT*. Morgan Kaufmann, 1994.

Piedachu Peris and Damián López. Transducer inference by assembling specific languages. In *Proceedings of ICGI*, volume 6339 of *Lecture Notes in Computer Science*, pages 178–188. Springer-Verlag, 2010.

David Picó and Francisco Casacuberta. Some statistical-estimation methods for stochastic finite-state transducers. *Machine Learning*, 44(1/2):121–141, 2001.

Leonard Pitt and Manfred K. Warmuth. The minimum consistent dfa problem cannot be approximated within any polynomial. *Journal of the Association of Computing Machinery*, 40(1):95–142, 1993.

Lawrence R. Rabiner. Readings in speech recognition. chapter A tutorial on hidden Markov models and selected applications in speech recognition, pages 267–296. Morgan Kaufmann Publishers Inc., 1990.

Harald Raffelt and Bernhard Steffen. Learnlib: A library for automata learning and experimentation. In *proceedings of FASE*, volume 3922 of *Lecture Notes in Computer Science*, pages 377–380. Springer-Verlag, 2006.

Christophe Reutenauer and Marcel Paul Schützenberger. Minimization of rational word functions. *SIAM Journal on Computing*, 20(4):669–685, 1991.

Christophe Reutenauer and Marcel Paul Schützenberger. Variétés et fonctions rationnelles. *Theoretical Computer Science*, 145(1&2):229–240, 1995.

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

Brian Roark and Richard Sproat. *Computational Approaches to Morphology and Syntax*. Oxford University Press, 2007.

Azriel Rosenfeld. *An Introduction to Algebraic Structures*. Holden-Day, 1968.

Yasubumi Sakakibara. Pair hidden Markov models on tree structures. In *Proceedings of the Eleventh International Conference on Intelligent Systems for Molecular Biology*, pages 232–240, 2003.

Yasubumi Sakakibara. Grammatical inference in bioinformatics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1051–1062, 2005.

Arto Salomaa and Matti Soittola. *Automata: Theoretic Aspects of Formal Power Series*. Springer-Verlag, 1978.

David B. Searls. Molecules, languages and automata. In *Proceedings of ICGI*, volume 6339 of *Lecture Notes in Computer Science*, pages 5–10. Springer-Verlag, 2010.

David B. Searls and Kevin P. Murphy. Automata-theoretic models of mutation and alignment. In *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, pages 341–349. AAAI, 1995.

Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

Burr Settles. From theories to queries: Active learning in practice. In *Proceeingds of Active Learning and Experimental Design workshop*, volume 16 of *JMLR: Workshop and Conference Proceedings*, pages 1–18. MIT Press, 2011.

Michael Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition, 1996.

Richard Sproat. A finite-state architecture for tokenization and grapheme-to-phoneme conversion in multilingual text analysis. In *Proceedings of the ACL SIGDAT Workshop, Dublin, Ireland. ACL*. Association for Computational Linguistics, 1995.

Bradford Starkie, Menno van Zaanen, and Dominique Estival. The tenjinno machine translation competition. In *Proceedings of ICGI*, volume 4201 of *Lecture Notes in Computer Science*, pages 214–226. Springer-Verlag, 2006.

Yuji Takada. Grammatical interface for even linear languages based on control sets. *Information Processing Letters*, 28(4):193–199, 1988.

Franck Thollard, Pierre Dupont, and Colin de la Higuera. Probabilistic dfa inference using Kullback-Leibler divergence and minimality. In *Proceedings of ICML*, pages 975–982. Morgan Kaufmann, 2000.

Boris Avraamovich Trakhtenbrot and Ya Martynovich Barzdin. *Finite Automata: Behavior and Synthesis*. North Holland Pub. Comp, 1973.

Leo P. J. Veelenturf. Inference of sequential machines from sample computations. *IEEE Transactions on Computers*, 27(2):167–170, 1978.

Sicco Verwer, Rémi Eyraud, and Colin de la Higuera. Pautomac: a pfa/hmm learning competition. *JMLR: Workshop and Conference Proceedings*, 2012.

Enrique Vidal and Francisco Casacuberta. Learning finite-state models for machine translation. In *Proceedings of ICGI*, volume 3264 of *Lecture Notes in Computer Science*, pages 3–15. Springer-Verlag, 2004.

Enrique Vidal, Pedro García, and Encarna Segarra. Inductive learning of finite-state transducers for the interpretation of unidimensional objects. *Structural Pattern Analysis*, pages 17–35, 1990.

Enrique Vidal, Franck Thollard, Colin de la Higuera, Francisco Casacuberta, and Rafael C. Carrasco. Probabilistic finite-state machines-part ii. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1026–1039, 2005a.

Enrique Vidal, Franck Thollard, Colin de la Higuera, Francisco Casacuberta, and Rafael C. Carrasco. Probabilistic finite-state machines-part i. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1013–1025, 2005b.

Juan Miguel Vilar. Query learning of subsequential transducers. In *Proceedings of ICGI*, volume 1147 of *Lecture Notes in Computer Science*, pages 72–83. Springer-Verlag, 1996.

Juan Miguel Vilar. Improve the learning of subsequential transducers by using alignments and dictionaries. In *Proceedings of ICGI*, volume 1891 of *Lecture Notes in Computer Science*, pages 298–311. Springer-Verlag, 2000.

Juan Miguel Vilar, Víctor M. Jiménez, Juan-Carlos Amengual, Antonio Castellanos, David Llorens, and Enrique Vidal. Text and speech translation by means of subsequential transducers. *Natural Language Engineering*, 2:351–354, 1996.

Mitsuo Wakatsuki and Etsuji Tomita. Polynomial time identification of strict prefix deterministic finite state transducers. In *Proceedings of ICGI*, volume 6339 of *Lecture Notes in Computer Science*, pages 313–316. Springer-Verlag, 2010.

Appendix A

PSMA: A Parallel Algorithm for Learning Regular Languages

A.1. Preamble

The results presented here have been jointly conducted in collaboration with Alban Batard, Colin de la Higuera and Claudia Eckert and a preliminary version of this work has been published in (Akram et al., 2010a). This appendix is self contained in terms of the notations and definitions used and not to be confused with the notations used in the rest of the thesis.

A.2. Introduction

Inferring a regular language from examples and counter-examples is a classical problem in grammatical inference (de la Higuera, 2010). It is also known as automata synthesis or grammar induction and corresponds to finding the smallest Dfa consistent with a labelled sample of strings. The classical algorithm (Rpni (Oncina and García, 1992)) to solve this problem runs in polynomial (but cubic) time, and in practical situations where the size of the learning sample is large the algorithm cannot be used. A number of alternative algorithms have been proposed in the past 40 years (Trakhtenbrot and Barzdin, 1973; Lang et al., 1998; Heule and Verwer, 2010). In grammatical inference, the only other attempt (to our knowledge) of parallelizing learning algorithms was made in the alternative framework of active learning (Balcázar et al., 1994; Angluin, 1987a).

Our Parallel State Merging Algorithm (Psma) is an EREW PRAM learning algorithm for learning Dfa. This algorithm is strongly based on the sequential

state merging algorithm RPNI (Oncina and García, 1992) and adopts a multi-core processor computation paradigm that allows to test possible state merges in parallel.

A.3. Preliminaries

Let Σ be a non-empty set of symbols called ***letters***. Σ^* is the set of all strings over the alphabet Σ where a ***string*** $x \in \Sigma^*$ is a finite sequence of letters $x = x_1 x_2 \cdots x_n$. A ***language*** $\mathcal{L}$ is any subset of Σ^*. If $x = uv$ is a string, then u is a ***prefix*** of the string x. The ***prefix set*** $Pref(\mathcal{L})$ of the language $\mathcal{L}$ is defined as $Pref(\mathcal{L}) = \{u \in \Sigma^* : uv \in \mathcal{L}\}$.

A ***Deterministic Finite Automaton (*Dfa*)*** is a quintuple $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ where Σ is an alphabet, Q is a set of finite states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $\delta : Q \times \Sigma \to Q$ is the transition function.

A.4. The problem

Let $\langle S_+, S_- \rangle$ be a finite sample of some language $\mathcal{L}$ consisting of a subset $S_+ \subseteq \mathcal{L}$, set of positive strings of the language $\mathcal{L}$ and $S_- \subseteq \Sigma^* \setminus \mathcal{L}$, set of negative strings of the language $\mathcal{L}$. Throughout the paper we assume the samples to be ***non-conflicting***, *i.e.*, $S_+ \cap S_- = \emptyset$.

In a DFA learning (or synthesis) problem, we are given a sample $\langle S_+, S_- \rangle$ as above, and the goal is to find *the* language $\mathcal{L}$. Obviously, there is a number of languages such that $S_+ \subseteq \mathcal{L}$ and $S_- \subseteq \Sigma^* \setminus \mathcal{L}$: such a language is said to be ***consistent*** with $\langle S_+, S_- \rangle$. As a combinatorial problem, the corresponding goal is to find the *smallest* consistent DFA. As an inference problem, we want to have an algorithm which returns a DFA and furthermore we would like the algorithm to converge with the data, *i.e.*, there is a guarantee that with more and more elements in S_+ and S_-, we can be sure to find $\mathcal{L}$.

The general strategy used by the most common family of DFA learning algorithms is that of ***state merging***. The starting point is the ***Prefix Tree Acceptor*** (PTA), built from S_+: this is a tree-like DFA $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ PTA(S_+) defined as follows: $\{Q = q_u : u \in Pref(S_+)\}$, $\forall ua \in Pref(S_+) : \delta(q_u, a) = q_{ua}$, $F = \{q_u : u \in S_+\}$.

State-merging algorithms maintain a set of RED states corresponding to the *confirmed* states, *i.e.*, those present in the final DFA, and a set of BLUE states, successors of the RED states and candidates for merging. The goal is to generalise

the language recognised by the running DFA by iteratively choosing one RED state and one BLUE state and attempting to merge these together: if the result of this merge is not over-general (*i.e.*, no string from S_- is recognised: the two states are then said to be ***compatible*** and ***incompatible*** otherwise) the merge is kept; if not, it is rejected. Whenever a particular BLUE state can be merged with no RED state it gets promoted: it becomes RED and all its non RED successors become BLUE.

RPNI (Oncina and García, 1992) is a deterministic algorithm where the merge compatibilities between two states are checked sequentially in a predefined (length lexicographic) order. Whenever a merge is rejected, RPNI tries to merge another pair of states and continues until no further merges are possible. It is known that RPNI ensures identification of DFA in the limit and works in polynomial time (de la Higuera, 2010).

A.5. The PSMA Algorithm

We introduce a new learning algorithm called PSMA (*Parallel State Merging Algorithm*) which obtains the same result as RPNI but makes use of a multi processor architecture.

Let us denote by n the number of states of the PTA and suppose these are numbered (in a breadth-first way) from 0 to $n-1$. Pairs of (indexes of) states will be ordered: $\langle i,j\rangle < \langle i',j'\rangle$ if $j < j'$ or $j = j'$ and $i < i'$. The first element of each pair corresponds to a RED state, the second to a BLUE state. For example, $\langle 3,5\rangle < \langle 2,6\rangle$ and $\langle 3,5\rangle < \langle 4,5\rangle$.

The master processor node M takes the responsibility of initialising and updating the shared data:

- a set $\mathcal{R}$ of RED states; initially $\mathcal{R} = \{0\}$;
- a set $\mathcal{B}$ of BLUE states; initially $\mathcal{B} = \{i : \exists a \in \Sigma \text{ such that } \delta(q_0, a) = q_i\}$;
- a table containing for each pair $\langle i,j\rangle$ with $i \in \mathcal{R}$, $j \in \mathcal{B}$, the information $T[i][j]$;
- a counter called SIT which corresponds to the position in the table up to which the merges are validated by the master so far.

T can be implemented as a queue in order to have a small data struicture: it will only contain the values corresponding to *active* pairs of states. $T[i][j]$ will take the following possible values:

- if the merge between state $i \in \mathcal{R}$ and $j \in \mathcal{B}$ has an unknown status the value is **U**;
- if the merge is being considered in the actual context by some processor the value is **C**;
- if the merge is being considered by some processor, but the master has accepted a merge the value is **R**;
- if the merge has been discarded the value is **D**;
- if the merge is proposed in the current context by one processor the value is **P**;
- if the merge is accepted and has been validated by the master processor, the value is **A**;

The **master processor** is in charge of updating the shared information. It does so by looking at the entry in the table T corresponding to SIT= $\langle i, j\rangle$:

- if $T[i][j]$ =**D** and is the last corresponding to that particular BLUE (j) the table is updated by promoting state j. This consists in (1) updating the local $\mathcal{B}$ and $\mathcal{R}$ sets (*i.e.*, state j is moved from $\mathcal{B}$ to $\mathcal{R}$, and successors of state j are moved to $\mathcal{B}$; (2) pairs $\langle j, k\rangle$ are inserted in their correct position in the table, where k is any BLUE state; (3) counter SIT is incremented.
- if $T[i][j]$ =**P** the following update takes place: (1) $T[i][j] \leftarrow$ **A**; (2) $\forall k > j$, $T[i][k] \leftarrow$ **D**; (3) $\forall \langle h, k\rangle >$SIT, if $T[h][k]$ =**C**, $T[h][k] \leftarrow$**R** and if $T[h][k]$ =**P**, $T[h][k] \leftarrow$**U**; (4) $\mathcal{B}$ and $\mathcal{R}$ are updated in the usual way RPNI does it; (5) SIT is incremented to $< \langle i', j'\rangle$ where i' is the first RED state and j' is the next BLUE state.

Each other processor P_z plays the part of a slave processor. It finds the smallest $\langle i, j\rangle$, from SIT onwards such that $T[i][j]$ =**U**. P_z sets $T[i][j]$ to **C**.

Then P_z runs RPNI with the help of the table T: every time a merge between two states r and b is to be tested, P_z checks the table T. If $T[r][b]$ is **A** the merge is done; in all other cases the compatibility result is *fail*. When P_z has to test the merge between i and j, it really does it and returns the compatibility result.

Processor P_z does the following when it has finished its task, consisting in testing the merge between i and j:

1. If $T[i][j]$=**C** then if the merge P_z has tested is OK, it updates $T[i][j]$ to **P** (proposed). If the merge is not OK, it updates $T[i][j]$ to **D**.

2. If $T[i][j]$=**R** (which means that at least one new merge has been taken into account), then if the merge is not OK, it updates $T[i][j]$ to **D**. If the merge is OK, nothing can be decided: it updates $T[i][j]$ to **U**.

3. It searches for another merge to be checked.

A small analysis of the algorithm

The key idea is that each slave processor is kept busy at all times: it finds the next job in the queue and will try to check a merge in which the context is that all the previous unfinished jobs are going to fail. If the result of this job is **incompatible**, then the result will hold even if an unfinished job returns **compatible**. The *bad* case occurs when the job returns **compatible**: the result in only kept if all previous compatibility tests fail.

A.6. Experimental Results

We have conducted our experiments on a multi-core machine having Intel® Xeon® processors, 4x6 cores, 2.66 GHz, 16 MB cache memory, RAM: 128 GB. A series of runs of the algorithm was performed using datasets generated by Gowachin[1], with different sizes of targets DFA, number of examples, and number of slaves used in the algorithm.

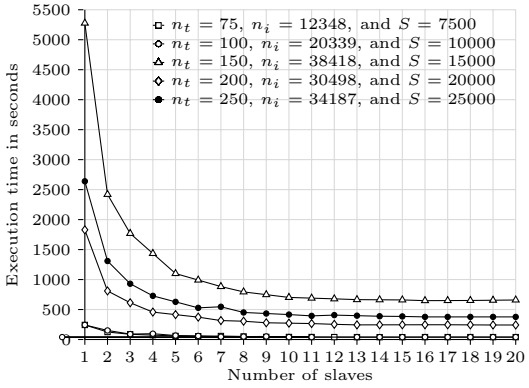


Figure A.1.: Plot of time taken to execute different sizes of datasets over number of slaves. n_t is the number of nodes in the target DFA, n_i is the number of nodes in the initial PTA and S is the sample size.

[1]Gowachin allows to generate artificial datasets for testing DFA learning methods: `http://www.irisa.fr/Gowachin/`

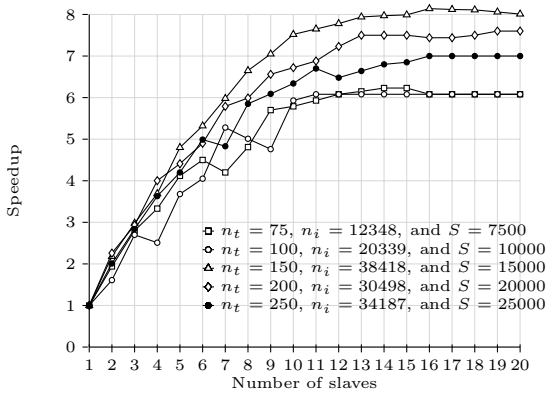


Figure A.2.: Plot of speedup gained over the number of processors. n_t is the number of nodes in the target DFA, n_i is the number of nodes in the initial PTA and S is the sample size.

$p \rightarrow$ $n \downarrow$	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20
12348	1	0.97	0.93	0.83	0.82	0.75	0.6	0.6	0.63	0.58	0.51	0.45	0.38	0.34	0.3
20339	1	0.8	0.9	0.63	0.74	0.68	0.75	0.63	0.53	0.59	0.51	0.43	0.38	0.34	0.3
38418	1	1.09	0.99	0.92	0.96	0.89	0.85	0.83	0.78	0.75	0.65	0.57	0.51	0.45	0.4
30498	1	1.13	0.99	1	0.88	0.82	0.83	0.75	0.73	0.67	0.6	0.53	0.46	0.42	0.38
34187	1	1.01	0.95	0.91	0.84	0.83	0.69	0.73	0.68	0.63	0.54	0.49	0.44	0.39	0.35

Table A.1.: Efficiency E as a function of n (PTA size) and p (number of processors).

From Figure 1 it appears that significant speedup can be gained when using large datasets (leading to large PTA). As we increase the number of slaves, the speedup decreases until reaching a saturation point (here with 13 slaves). This behaviour can be explained as follows: the hypothesis under which a processor checks the merge it has to test has a probablity of being invalidated that grow with the number of processors.Therefore, at some point, increasing the number of processors brings no speedup.

Let the *sequential execution time* (Gupta and Kumar, 1993), *i.e.*, the time taken to execute the algorithm with a single processor be denoted by T_1. The *parallel execution time*, *i.e.*, the execution time for the algorithm with p processors be denoted by T_p. The *speedup* is defined as $S = \frac{T_1}{T_p}$. The *efficiency* is defined as $E = \frac{S}{p}$. Figure 2 depicts the speedup of the algorithm over the number of processors. Table 1 shows the efficiency table as a function of number of states in the PTA and number of slaves. Efficiency appears to be very high (Table 1) with lower number of slaves and decreases as the number of slaves increases due to the similar reason as limitation in speedup gain explianed earlier. Adding more resources to gain relatively small fraction of speedup (or no speedup) results in

low efficeincy.

A.7. Conclusion & Future Outlook

In this paper we have described a parallel version of a state merging algorithm for inferring regular languages (RPNI). Experimental results have been presented based on a Java implementation[2] of the algorithm, where a significant performance gain has been obtained. However, the results also indicate that there is a limit for the speedup gain.

In this version, the algorithm remains deterministic and depends on a predefined ordering of the states. A parallelisation of the Evidence Driven State Merging algorithm (EDSM, (Lang et al., 1998)) can also be done: in this case, between the different **P** values returned by the slave processors, the master will choose the one with the highest score.

[2] `http://pagesperso.lina.univ-nantes.fr/~cdlh/Downloads/RPNIP.tar.gz`

Appendix B

GIToolBox: A Grammatical Inference Library in MATLAB

B.1. Preamble

The results presented here have been jointly conducted in collaboration with Huang Xiao, Colin de la Higuera and Claudia Eckert and a preliminary version of this work has been published in (Akram et al., 2010b). This appendix is self contained in terms of the notations and definitions used and not to be confused with the notations used in the rest of the thesis.

B.2. Introduction

In this paper we focus on two important tasks for GI - learning regular languages from an informant and learning *k-testable* languages (García and Vidal, 1990) from text. We have implemented the RPNI (Oncina and García, 1992) and EDSM (Lang et al., 1998) algorithms to investigate the feasibility of running such classes of algorithm in MATLAB[1]. We have also implemented algorithms to learn *k-testable* languages which corresponds to a subset of regular languages.

We have followed the notations and algorithms given in Colin de la Higuera's (de la Higuera, 2010) book. The details of RPNI algorithm can be found in chapter 12, EDSM in chapter 14 and *k-testable* language in chapter 11 of the book. In this section we briefly introduce notations and definitions.

[1]MATLAB is a registered trademark of The MathWorks, Inc.

B.2.1. Preliminaries

Definition 27 *A **Deterministic Finite Automaton** is defined as a 6-tuple* $\mathcal{A} = \langle \Sigma, Q, q_0, \mathbb{F}_{\mathbb{A}}, \mathbb{F}_{\mathbb{R}}, \delta \rangle$*, where* Σ *is the set of alphabets,* Q *is the set of finite states,* $q_0 \in Q$ *is the initial state,* $\mathbb{F}_{\mathbb{A}} \subseteq Q$ *is the set of final accepting states,* $\mathbb{F}_{\mathbb{R}} \subseteq Q$ *is set of final rejecting states,* δ *is the transition function.*

Definition 28 *A **Prefix Tree Acceptor (PTA)** is a tree-like DFA generated by extracting all the prefixes of the samples as states that only accepts the samples it is built from. A **prefix tree** is also known as **trie** which is an ordered data structure and it is expressed as a DFA to form a PTA. Let* S *be the sample from which we build a PTA.* $\mathcal{A}_{\mathcal{PTA}} = PTA(S)$ *is a DFA that contains a path from the initial state to a final accepting state for each strings in* S*.*

To recognize *k-testable* languages we would require a special machine called *k-testable machine* from which we can build an equivalent DFA.

Definition 29 *Given* $k > 0$*, a **k-testable machine*** $k - \mathcal{TSS}$ *is a 5-tuple* $Z_k = \langle \Sigma, I, F, T, C \rangle$ *where* Σ *is the set of alphabets,* $I \subseteq \Sigma^{k-1}$ *set of prefixes of length* $k-1$*,* $F \subseteq \Sigma^{k-1}$ *suffixes of length* $k-1$*,* $C \subseteq \Sigma^k$ *set of short strings, and* $T \subseteq \Sigma^k$ *set of allowed segments.*

A *k-testable machine* $k - \mathcal{TSS}$ will recognize strings only either exactly in C, or those whose prefix of length $k-1$ is in I, suffix of length $k-1$ is in F, and where all substrings of length k is in T.

B.3. State Merging Algorithms

The basic idea of state merging algorithm to infer a DFA is to build a PTA from the positive sample (S_+), conduct the state-merging iteratively, each intermediate DFA is verified by examining the negative samples. Only the merge resulting in a DFA that rejects all the negative samples is kept as current status, otherwise the merge is discarded. This process is repeated until the target automaton is found. Examples of such state merging algorithms are RPNI, EDSM, Blue-Fringe etc.

B.3.1. RPNI

The RPNI version given in (de la Higuera, 2010) uses two labels for the states in the automaton: *red* states and *blue* states. After a series of merges between the *red* states and *blue* states, promoting the states (e.g., *blue* to *red*), the target

DFA is produced. For the details of RPNI please consult chapter 12 of (de la Higuera, 2010).

By consideration of overhead of computing, we initiated another version of RPNI, Parallel RPNI, which opens multiple sessions for states in PTA and runs those sessions concurrently so that if the merge at state i is failed, then the merges at states $\{i+1, i+2, i+3, \cdots\}$ might be already prepared to be taken into execution.

B.3.2. EDSM

RPNI basically performs a greedy search to find out the target DFA, meaning whenever two states are mergable, they are merged. Obviously there could be other option, e.g., choosing a *better* or even *best* merge by means of some heuristics. EDSM algorithm introduced by Lang and et. al., (Lang et al., 1998) which takes such heuristics into consideration.

B.4. MATLAB GI Toolbox

In this section we present the MATLAB GI Toolbox, an open source implementation of a set of GI algorithms in MATLAB. The Toolbox offers out-of-the-box implementations of a range of GI algorithms that can be used like every other machine learning tool provided in MATLAB.

The fundamental data structures in the MATLAB (MathWorks, 2010) platform are matrices. Therefore, in our implementation we have used matrices as primary data structure to represent a DFA. The DFA object contains eight different sets represented as matrices: ***FiniteSetOfStates:*** the set of finite states (Q) of the DFA, it is stored as an integer vector in MATLAB. ***Alphabets***: the set of symbols. It is stored as a cell array where each cell contains a character. ***TransitionMatrix***: each column of the matrix corresponds to a symbol $a \in \Sigma$ *(Alphabets)* and each row corresponds to a state $q \in Q$ *(FiniteSetOfStates)*. Each cell of the matrix is a transition δ, e.g., if there is a transition from a state q_i to q_j via a symbol a, then the corresponding cell for q_i and a is marked as q_j. No transition cells are marked with -1. ***InitialState:*** the set of initial states, an integer vector which contains only state. ***FinalAcceptedStates:*** the set of final accepted states, an integer vector which is a subset of *FiniteSetOfStates*. ***FinalRejectStates:*** the set of final rejecting states, an integer vector which is a subset of *FiniteSetOfStates*. ***RED:*** the set of *red* states, an integer vector which is a subset of *FiniteSetOfStates*. ***BLUE:*** the set of *blue* states, an integer vector which is a subset of *FiniteSetOfStates*.

The input file is given in the similar format as the Abbadingo[2] format. However, internally in MATLAB the training dataset is represented as cell arrays (MathWorks, 2010), each cell containing a character, which is independent of the input file format.

B.4.1. Features

The MATLAB GI Toolbox provides the GI algorithms in a modular fashion so that they can be reused to enhance or improve the existing GI algorithms. The DFA data structure and the built-in methods can be used for RPNI, Blue-Fringe, EDSM and can be extended to incorporate other methods such as Genetic Algorithm techniques to optimize the search strategy. Moreover, this toolbox has been made absolutely compatible to other MATLAB features and toolboxes which enables ways of trying out new experiments using other MATLAB Toolboxes in an extremely easy manner. It is simple to use the toolbox to do a *k-fold cross-validation*, ROC analysis etc. using internal MATLAB classes to obtain fast results.

B.5. Experimental Results

In this section we present our experiments which are conducted on the Gowachin[3] dataset varying the size of the target DFA and the sample size Table B.1 for RPNI and EDSM. The results obtained from the MATLAB GI Toolbox are tested also with Gowachin. Table 2 shows experiments on learning k-testable machines. The experiments were run in a machine having two CPUs, each with 4 core Quad-Core AMD Opteron^{TM}Processor 2384, cache size: 512 KB, memory 66175292 KB. In these experiments, the accuracy results are as expected, bad when an insufficient amount of data is provided.

B.6. Conclusion & Future Work

The experimental results shown above clearly indicate that MATLAB is perfectly suitable for GI algorithms and experiments, at least for reasonable sizes of datasets. Besides the two state merging algorithms, we have also implemented

[2]Abbadingo is a DFA learning competition held in 1998. The details about the competition and data format can be found at: `http://www-bcl.cs.may.ie/`

[3]Gowachin: DFA Learning Competition is a test version of a follow-on to Abbadingo One. Artificial datasets for training and testing can be generated using this website: `http://www.irisa.fr/Gowachin/`

sample size →	*200*		*500*		*1000*		*5000*		
DFA size ↓	RPNI	EDSM	RPNI	EDSM	RPNI	EDSM	RPNI	EDSM	↓
3	99.17	99	100	99.28	99.33	100	99.17	100	***a***
	0.49	0.44	0.48	2.25	1.03	1221.6	43.12	23.09	***t***
5	100	77.94	92.28	100	99.78	99.67	100	99.72	***a***
	0.50	639.59	511.67	31.84	2.28	26.06	11.10	245.95	***t***
10	67.39	66.78	100	99.89	99.61	100	100	100	***a***
	16.09	736.31	2.9	257.79	4.57	464.95	31.47	1421.8	***t***
20	63.94	52.5	61.67	60.77	99.06	99.33	100	100	***a***
	13.10	2043.6	180.71	1873.8	21.46	3677.99	99.20	3989.71	***t***

Table B.1.: MATLAB GI ToolBox executions of RPNI and EDSM with different target DFA sizes and sample sizes. Row ***a*** shows the accuracy (%) and row ***t*** shows the time cost in seconds.

sample size →	*200*			*500*			*1000*			*5000*		
k ↓	***t***	***p***	***r***	***t***	***p***	***r***	***t***	***p***	***r***	***t***	***p***	***r***
2	1.16	0.12	0.14	2.98	1	0.58	6.03	1	0.408	31.77	0.5	1
3	1.16	1	0.14	2.84	1	0.579	5.83	1	0.407	31.30	1	1
5	1.17	1	0.04	2.86	1	0.142	5.89	1	0.023	30.78	1	0.543
10	4.58	1	0.002	9.19	NaN	0	16.08	1	0.007	65.06	1	0.002

Table B.2.: MATLAB GI ToolBox executions of learning ***k-testable machine***. Column ***t*** shows the time cost in seconds, ***p*** the precision and ***r*** the recall.

learning algorithm for *k-testable machine*. Moreover, we have implemented a parallel version of RPNI using the MATLAB Parallel Toobox (MathWorks, 2010), where we have been able to gain 10-15% speedup for each additional CPU. Our future plan is to incorporate other GI algorithms such as L*, OSTIA (for learning *transducers*) etc. in the toolbox.

To the best of our knowledge this is the first open source implementation of GI algorithms in MATLAB. We plan to publish the MATLAB GI ToolBox as an open source library under the MIT License for open source software. The beta version of MATLAB GI ToolBox can be downloaded from the following link: `http://www.sec.in.tum.de/~hasan/matlab/gi_toolbox/`.

Appendix C

Auxiliary Properties of Transducers

C.1. Properties of Sequential & Subsequential Transducers

Property 6 *Let $t_T : \Sigma^* \to \Omega^*$ be a transduction realized by a sequential transducer T where every state of T is reachable from the initial state. t_T is a total function iff $\forall q \in Q, \forall a \in \Sigma, \exists e \in E[q] : i[e] = a$.*

Proof First, we set our premise as the following: the transduction $t_T(x)$ is realized by a subsequential transducer T such that $\forall q \in Q, \forall a \in \Sigma, \exists e \in E[q] : i[e] = a$. We prove by induction on string length that $t_T(x)$ is a total function, *i.e.*, $\forall x \in \Sigma^*$, $|t_T(x)| = 1$, where $|t_T(x)|$ stands for the cardinality of $t_T(x)$.

Basis: Let $t_T(q, a)$ be the transduction produced by T, when $q \in Q$ is treated as the initial state of T and $a \in \Sigma$. It is easy to see that: $\forall a \in \Sigma, |t_T(q, a)| = 1$. Similarly, it is easy to see that the following is also true:

$$\forall x \in \Sigma^*, |t_T(q, x)| = 1 \text{ if } |x| = 1 \tag{C.1}$$

If $q = q_0$, we have,

$$\forall x \in \Sigma^*, |t_T(x)| = 1 \text{ if } |x| = 1$$

Inductive step: Let $x = yw$ such that $|w| = 1$. We set our induction hypothesis as:

$$\forall y \in \Sigma^*, |t_T(y)| = 1, \text{ if } |y| = n$$

We prove,

$$\forall x \in \Sigma^*, |t_T(x)| = 1, \text{ if } |x| = n + 1$$

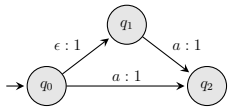


Figure C.1.: A counterexample showing that ϵ-transitions cause ambiguity in the output strings.

Let $\pi_y \in \Pi_T : i[\pi_y] = y$.

$$\begin{aligned} |t_T(x)| &= |t_T(yw)| \\ &= |t_T(y) \cdot t_T(next[\pi_y], w)| \\ &= |t_T(y)|\,|t_T(next[\pi_y], w)| \end{aligned}$$

From induction hypothesis and (C.1):

$$\forall x \in \Sigma^*, |t_T(x)| = 1, \text{ if } |x| = n + 1$$

Second, it is easy to see that the converse is true, *i.e.*, if t_T is a total function and it is realized by T then $\forall q \in Q, \forall a \in \Sigma, \exists e \in E[q] : i[e] = a$. □

Corollary 5 *For any sequential transducer T that realizes a total function and if every state of T is reachable from the initial state the following property holds:* $\forall x \in \Sigma^*, \exists! \pi \in \Pi_T(I) : i[\pi] = x$.

From Property 6, it logically follows that if t_T is not a total function, t_T may only be defined for a subset of Σ^*. We denote $\mathfrak{Dom}(t_T) \subseteq \Sigma^*$ for which t_T is defined. In less formal terms, t_T is a total function *iff* $\mathfrak{Dom}(t_T) = \Sigma^*$.

Another important restriction that has been added in Definition 8 is that no ϵ-transitions for the input is allowed. This restriction is necessary to guarantee the unambiguity of the output string for a given input string.

Property 7 *Sequential transducers that allow ϵ-transitions do not always realize functions.*

Proof We prove the proposition by showing the counterexample depicted in Figure C.1. Clearly there are two different outputs for the input string a: $(a, 1)$ and $(a, 11)$. This violates the condition of being a function. □

Property 8 *Let $t_T : \Sigma^* \rightarrow \Omega^*$ be a transduction realized by a subsequential transducer T where every state of T is reachable from the initial state. t_T is a total function iff $\forall q \in Q, \forall a \in \Sigma, \exists e \in E[q] : i[e] = a$ and $\forall q \in Q, \sigma(q)$ is defined.*

Proof The proof is analogous to the proof of Property 6. It can be shown by applying induction on string length and by contradiction that the property holds. □

A subsequential transducer that realizes a total function is said to be a *total subsequential transducer*. Figure 2.6 depicts an example of a total subsequential transducer.

C.2. The Twins Property

One of the reasons why p-subsequential transducers (p-STs) are useful is because they are equivalent to rational transducers with the twins property, *i.e.*, rational transducers having the twins property, *a.k.a.* $p-$subsequentiable transducers, can be determinized to p-STs. Here we will briefly discuss the twins property. The twins property was first introduced by Berstel in (1979). Afterwards, a generic definition of the twins property over an arbitrary semiring was presented in (Allauzen and Mohri, 2003a). Here to discuss the idea, we define the twins property *w.r.t.* a string to string rational transducer.

Definition 30 (Siblings) *Let T be a rational transducer. The two states q_1 and q_2 of T are said to be siblings if there exist two strings x and y in Σ^* such that:*

- $\exists \pi_{x_1}, \pi_{x_2} \in \Pi_T(I)$:
  - $i[\pi_{x_1}] = x \wedge next[\pi_{x_1}] = q_1$
  - $i[\pi_{x_2}] = x \wedge next[\pi_{x_2}] = q_2$
- $\exists \pi_{y_1}, \pi_{y_2} \in \Pi_T$:
  - $i[\pi_{y_1}] = y \wedge prev[\pi_{y_1}] = next[\pi_{y_1}] = q_1$
  - $i[\pi_{y_2}] = y \wedge prev[\pi_{y_2}] = next[\pi_{y_2}] = q_2$

Less formally, the first condition for the two states q_1 and q_2 to be siblings is that they both have to be reachable from the initial state with input string $x \in \Sigma^*$. The second condition is that q_1 and q_2 both must have a cycle with

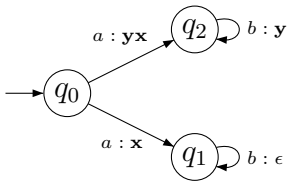


Figure C.2.: Illustration of the twins property.

the input string $y \in \Sigma^*$. In Figure C.2 q_1 and q_2 are siblings because both are reachable from the initial state q_0 with the input string a and both the states have got cycles with the input string b.

Definition 31 (Twins) *Let T be a rational transducer where q_1 and q_2 are two sibling states with $\pi_{x_1}, \pi_{x_2} \in \Pi_T(I)$ and $\pi_{y_1}, \pi_{y_2} \in \Pi_T$ have the same meaning for q_1 and q_2 as in Definition 30. q_1 and q_2 are said to be twins if the following equation and its symmetric counterpart obtained by transposing q_1 and q_2 hold:*

$$lcp(o\left[\pi_{x_1}\right], o\left[\pi_{x_2}\right])^{-1} o\left[\pi_{x_2}\right] = lcp(o\left[\pi_{x_1}\right] o\left[\pi_{y_1}\right], o\left[\pi_{x_2}\right] o\left[\pi_{y_2}\right])^{-1} o\left[\pi_{x_2}\right] o\left[\pi_{y_2}\right] \tag{C.2}$$

In Figure C.2 the two states q_1 and q_2 satisfy equation C.2 and therefore, are twins.

Definition 32 (The Twins Property) *A rational transducer T has the twins property if any two sibling states of T are twins.*

The rational transducer depicted in Figure C.2 has two sibling states q_1 and q_2 which are twins and hence why, the transducer has the twins property.

Appendix D

The Length-lex Order

Formally, the length-lex order is defined as:

Definition 33 (Length-lex Order) *Let $\leq_\Sigma$ be the total order relation over the symbols of Σ. The length-lex order $\leq_{lex}$: $\Sigma^* \times \Sigma^* \to \mathbb{B}$ of two strings $w, w' \in \Sigma^*$ is a total order relation defined as: suppose $w = w_1 \dots w_{|w|}$, $w' = w'_1 \dots w'_{|w'|}$ where $w_i, w'_i \in \Sigma$, then*

$$w \leq_{lex} w' = \begin{cases} |w| < |w'| & \text{if } |w| \neq |w'| \\ w_i \leq_\Sigma w'_i & \text{if } w_i \neq w'_i \wedge w_j = w'_j \text{ where } 1 \leq j < i \leq |w|\,, \\ \mathit{true} & \text{if } w = w' = \epsilon \\ \mathit{false} & \text{otherwise.} \end{cases}$$

Let us consider the example $S_n = \{(\sharp, \mathbf{xx}), (a\sharp, \mathbf{xx}), (b\sharp, \mathbf{yx}), (aa\sharp, \mathbf{xxx}), (baa\sharp, \mathbf{yxxx})\}$.

For example, the elements of the set $\mathit{Pref}(\{x : (x, \Omega^*) \in S_n\})$ is sorted in length-lex order in Table D.1. The second row of the table shows the corresponding states in the PTST (Figure D.1) for each prefixes numbered according to their length-lex order.

Table D.1.: Prefixes of the input strings of the training data in length-lex order and their corresponding states in the PTST.

prefixes	ϵ	a	b	$\sharp$	aa	$a\sharp$	ba	$b\sharp$	$aa\sharp$	baa	$baa\sharp$
corresponding states in the PTST	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_{10}

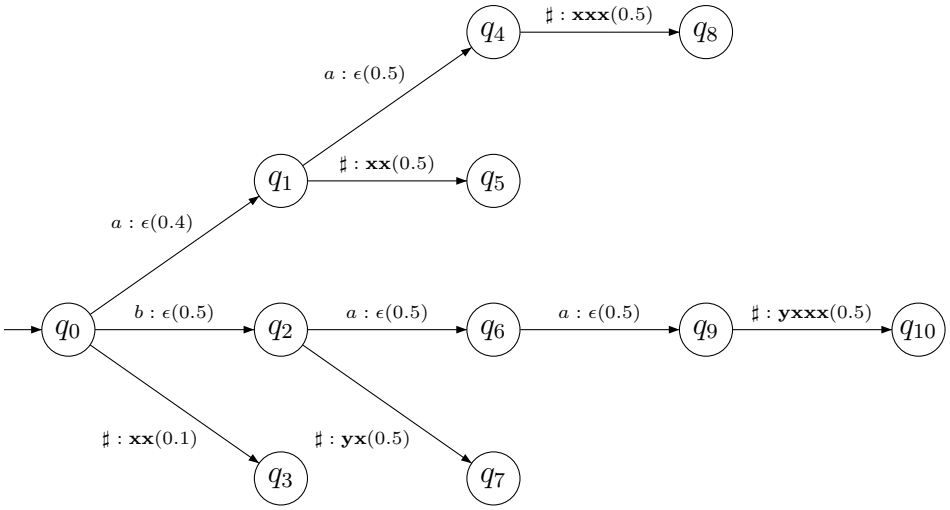


Figure D.1.: A PTST built from $S_n = \{(\sharp, \mathbf{xx}), (a\sharp, \mathbf{xx}), (b\sharp, \mathbf{yx}), (aa\sharp, \mathbf{xxx}), (baa\sharp, \mathbf{yxxx})\}$ where states are numbered in length-lex order (see Table D.1).

List of Figures

List of Tables

List of Algorithms