UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

FACULTAD DE
INGENIERÍA
UDELAR

Active Learning of Regular Languages as an Approach to Neural Language Models Verification

Franz Mayr

Tesis de Doctorado presentada a la Facultad de Ingeniería de la Univseridad de la República en cumplimiento parcial de los requisitos para la obtención del título de **Doctor en Informática**

Director de Tesis

Dr. Sergio Yovine - PEDECIBA, Universidad ORT Uruguay

Tribunal

Dr. Alvaro Martín, PEDECIBA, UdelaR **(presidente)**
Dr. Benedikt Bollig - Centre National de la Recherche Scientifique
Dr. Guillaume Rabusseau - Université de Montréal
Dr. Stavros Tripakis - Northeastern University
Dr. Gustavo Esteban Vázquez, PEDECIBA, UCU

Montevideo, Uruguay
Junio de 2024

Resumen

El presente trabajo aborda el problema general de la verificación del comportamiento de redes neuronales que procesan secuencias, en concreto los aceptores neuronales y los modelos neuronales de lenguaje. La tesis desarrolla un marco teórico-práctico para la extracción de abstracciones formales y la verificación de las redes neuronales bajo análisis. Este proceso se basa en dos ideas centrales: 1) tratar la red neuronal como una caja negra, y 2) utilizar un marco probabilístico para analizar en qué medida el modelo extraído se aproxima al original. Para ello, se proponen, desarrollan y analizan una serie de algoritmos y técnicas de aprendizaje activo. Para el caso de los aceptores neuronales se presenta un procedimiento de verificación de propiedades de redes neuronales. Este enfoque es capaz de verificar propiedades sin construir explícitamente representaciones de la red. Se demuestra que este enfoque ofrece mejores garantías y es más eficiente que la verificación posterior al aprendizaje, en la que la propiedad se verifica únicamente sobre el modelo aprendido de la red. Además, no requiere recurrir a un procedimiento de decisión externo para la verificación ni fijar un formalismo específico de especificación de requisitos. Para el caso de los modelos neuronales de lenguaje se presenta un algoritmo de aprendizaje basado en una congruencia sobre secuencias que se parametriza mediante una relación de equivalencia sobre distribuciones de probabilidad. El algoritmo de aprendizaje se implementa utilizando una estructura de datos en árbol y se muestra que es empíricamente más eficiente que las técnicas de referencia.

Abstract

This work tackles the general problem of verifying the behavior of sequence processing neural networks, specifically neural acceptors and neural language models. The contribution is a framework for extracting formal abstractions of the networks under analysis and verifying whether they satisfy given requirements. This process steps on two core ideas: 1) treating the neural network as a black box, and 2) using a probabilistic framework to analyze how much the extracted model approximates the original one. For this matter, a series of active learning algorithms and techniques are proposed, developed and analyzed. For the case of neural acceptors a procedure for checking properties of neural networks is presented. This approach

is able to check properties without explicitly building representations of the network. We show that this approach offers better guarantees and is more efficient than post-learning verification where the property is checked on a learned model of the network. Besides, it does not require resorting to an external decision procedure for verification nor fixing a specific requirement specification formalism. For the case of neural language models, a learning algorithm based on a congruence over strings which is parameterized by an equivalence relation over probability distributions is presented. The learning algorithm is implemented using a tree data structure and shown to be empirically more efficient than reference techniques.

Acknowledgements

Dedicado a mi familia y amigos, quienes fueron mi soporte en esta etapa.

Agradezco a mi tutor, Dr. Sergio Yovine, por las largas discusiones que dieron fruto a este trabajo.

Contents

1 Introduction

Artificial intelligence (AI) is a flourishing field with multiple practical applications and active research topics [1]. In this context learning-enabled components, more specifically artificial neural networks (ANN) stand out as one of the most successful models of AI in various fields of application. This phenomenon is rapidly spreading to high-risk critical systems whose failures can result in serious consequences, such as loss of life or damage to people or the environment. Examples include the use of AI in nuclear fusion process control [2], autonomous vehicle control [3] or medical diagnostics [4]. Therefore, there is a clear need to ensure the proper functioning of critical systems that integrate learning-enabled AI components.

This thesis falls within the general area of *verified AI* [5] whose objective is to design AI systems that have solid guarantees of correctness with respect to mathematically specified requirements. Specifically, it focuses on the design and adaptation of techniques based on automata theory to enable the specification of system behavior and the rigorous demonstration of properties, in particular model extraction [6] and model checking [7]. The theoretical results of this work are the foundation for building tools that contribute to *responsible AI* [8] in two aspects, explainability [9] and safety [10]. In fact, model extraction from ANN is an explainability technique, while formal verification allows early detection of errors or inappropriate uses and their subsequent correction and prevention. The tools developed in this thesis are available as open-source code for public use [11].

Indeed, ANN are "obscure" components in the sense that it is difficult for humans to understand why they make a prediction [12]. A key issue is that ANN have no explicit or constructive characterization of their decision-making strategy. This fact motivated a large amount of research work around explainable AI. In particular, this work follows a so-called *black-box* and *processing* explainability approach. The former is defined in [13] which characterizes the explanation problem as a means for providing a human-understandable model that mimics the ANN. The latter is presented in [14] as a method that seeks answering *why* a given input leads the ANN to produce a particular outcome.

Depending on the modality of the data to be processed different kinds of ANN are identified in the literature [1]. Concretely, the target learning-enabled components studied in this thesis are ANN specifically designed to treat sequences, such as RNN [15, 16] or Transformers [17]. We consider two main types of problems, namely classification and generation. The former consists in processing a sequence and output a discrete value representing the *class* of a sequence. Examples of that are, classifying a system log as normal or abnormal [18], identifying DNA promoter sequences [19] or detecting sentiments in a text [20]. The latter implies emitting a probability distribution of the possible next symbols, for example suggesting query auto-completion [21] or predicting the next event in a system log [22].

This work addresses the problem of black-box and processing explainability on ANN that process sequences via grammatical inference [23]. This approach consists in constructing a surrogate model through the inference of an underlying grammar, such as a deterministic finite automaton (DFA) or a probabilistic one (PDFA). It can be done either by passive learning, that is using a dataset of examples [24], or by active learning, which consists in querying the black-box and comparing the produced hypotheses against the target system. The latter is the central setting of this work. Specifically it relies in the Minimally Adequate Teacher (MAT) learning framework proposed by Angluin in [25]. It consists in a learner and a teacher, where the learner is given the ability to draw examples and to ask queries to the teacher.

Learned surrogate models are used as input to procedures dedicated to assess the correctness and reliability of the ANN. This is done through the specification of properties of interest and the definition of a verification or checking process. In other words, once the model is extracted it can be model-checked against a desired property using an appropriate model-checker [26–29].

This thesis focuses on the learning of surrogate models for which model-checking is feasible, such as regular languages. However, it is worth noting that the class of the ANN that process sequences is Turing complete [30]. Thus, the learned model would be, in general, an approximation of the ANN. Hence, it is important to give a proper notion on how precisely the extracted model characterizes the ANN. To cope with this situation, this work resorts to "Probably Approximately Correct" inference (PAC) [31], which provides a framework for learning surrogate models with required approximation error bounded by a confidence parameter.

The starting point of this PhD research is the work presented in [32,33]. Which consists in extracting DFA from ANN trained to solve *sequence* classification problems [34, 35], restricted to the case of binary classification. In such scenario the problem the ANN is solving is *language membership*, where the language of the

ANN is the set of sequences classified as positive by the network. The trained ANN implicitly defines a model of such sequences which it uses to predict whether a given input sequence belongs to the language. In that work, the Bounded-L*algorithm is presented, which relies on complexity bounds to guarantee termination, and the PAC learning framework is used to compare the surrogate model with the target ANN.

The application of that general approach in several case studies showed that, in practice, the model-then-verify approach has several important drawbacks [26,27]. The first one being state explosion. That is, the model learned from the ANN may be too large to be explicitly constructed. Another important inconvenience is that, since the surrogate model is an approximation of the ANN, errors found on the former could not be real errors of the ANN.

With the aim of overcoming these issues, this thesis develops two main lines of work. The first contribution consists in an on-the-fly property checking technique, where the property is checked during the learning phase, without using an external model-checker. This algorithm handles both the ANN and the property as black-boxes and it does not build, assume, or require them to be expressed in any specific way. This work resulted in the following publications:

- Franz Mayr, Ramiro Visca, Sergio Yovine. On-the-fly Black-Box Probably Approximately Correct Checking of Recurrent Neural Networks. CD-MAKE, Dublin, Ireland, August 25-28, (2020) [26]
- Franz Mayr, Sergio Yovine, Ramiro Visca. Property Checking with Interpretable Error Characterization for Recurrent Neural Networks. Mach. Learn. Knowl. Extr. 3(1): 205-227 (2021) [27].

The second contribution consists in defining equivalence relations over probability distributions that lead to abstractions related to the properties of interest. By integrating them into the learning algorithms this approach manages to reduce the size of the surrogate models by merging equivalent states. This work resulted in the following publications:

- Franz Mayr, Sergio Yovine, Federico Pan, Nicolas Basset, Thao Dang. Towards Efficient Active Learning of PDFA. Learnaut 2022, Paris, France, July 4, (2022) [36]

- Franz Mayr, Sergio Yovine, Matías Carrasco, Federico Pan, Federico Vilensky. A Congruence-based Approach to Active Automata Learning from Neural Language Models. International Conference on Grammatical Inference, Rabat, Morocco, July 10-13,(2023) [37]

- Franz Mayr, Sergio Yovine, Matías Carrasco, Alejo Garat, Martín Iturbide, Juan da Silva, Federico Vilensky. Results of Neural-Checker Toolbox in Taysir 2023 Competition. International Conference on Grammatical Inference, Rabat, Morocco, July 10-13,(2023) [38]

Document outline

The document is organized as follows. Chapter 2 presents a succinct introduction to the field of grammatical inference, where central concepts of formal language theory are revisited. Chapter 3 reviews PAC learning and proposes adaptations of the basic MAT framework to enable the learning of non-regular or possibly unbounded unknown languages. Chapter 4 develops on-the-fly verification through learning and showcases its application to neural binary acceptors. Chapter 6 presents language models, including PDFA, and defines a parameterized notion of congruence based on equivalence relations between probability distributions. Chapter 7 and 8, address the problem of MAT learning PDFA by developing table- and tree-based algorithms, respectively. Chapter 9 presents applications to domains of interest, in particular to neural language models. Lastly, Chapter 10 presents related work and conclusions. Appendix 12 presents an overview of the Neural-Checker [11] toolbox.

2 Grammatical inference

Grammatical inference is defined as the problem of learning formal descriptions of languages such as grammars and automata. It is a field with connections to a series of disciplines such as bio-informatics, computational linguistics and pattern recognition [23]. The goal is to develop algorithms able to infer such descriptions given some information about the languages. This chapter first revisits formal languages and then presents Angluin's MAT framework and L* algorithm for learning DFA which are the cornerstones of this work.

2.1 Formal languages

In this section we will briefly visit the central concepts from formal language theory as they are defined in [39].

An alphabet is defined as a finite, nonempty set of symbols Σ. From now on we will use the binary alphabet $\Sigma = \{0, 1\}$ as an example.

A string or word is a finite sequence of symbols chosen from an alphabet, for example 010101010111. The empty string, that we will denote λ, is the string with zero occurrences of symbols.

The concept of length in strings is defined as the number of positions for symbols in a string, and its notation is as follows, the length of a word w is denoted $|w|$. For instance $|10101| = 5$ and $|\lambda| = 0$.

Given an alphabet Σ, the set of strings of length k is represented by the expression Σ^k. For example, using the binary alphabet:

- $\Sigma^1 = \{0, 1\}$
- $\Sigma^2 = \{00, 01, 10, 11\}$

- $\Sigma^0 = \{\lambda\}$, for any Σ

The set of all strings over an alphabet Σ is denoted Σ^*. The set of all strings of length ≥ 1 over an alphabet Σ is denoted Σ^+.

A language $\mathcal{L}$ is defined as a set of strings chosen from Σ^*, where Σ is some particular alphabet. As examples we can think of:

1. The set of strings of even length: $\lambda, 00, 01, 10, 11, 0000, 0001, ...$
2. Σ^* is a language for any alphabet Σ.
3. $\varnothing$, the empty language is a language over any alphabet too.

We will refer to languages as their characteristic function $\Sigma^* \to \{0, 1\}$, that is $\mathcal{L}(x) = 1 \iff x \in \mathcal{L}$, 0 otherwise.

Mihill-Nerode equivalence As presented in [23] the Mihill-Nerode equivalence $\equiv$ on Σ^* is defined as follows:

$$\forall u, v \in \Sigma^*.\ u \equiv v \overset{\triangle}{\iff} [\forall w \in \Sigma^*.\ \mathcal{L}(uw) = \mathcal{L}(vw)] \tag{2.1}$$

This equivalence relation is also a right congruence:

$$\forall u, v \in \Sigma^*.\ u \equiv v \implies \forall w \in \Sigma^*.\ uw \equiv vw \tag{2.2}$$

Where the congruence class of $u \in \Sigma^*$ is denoted by $[\![u]\!]$.

Grammar In formal language theory, the grammar of a language is a description of the structure of a language [40], in other words a grammar is a set of rules with two goals: first, the rules allow for the construction of words belonging to the language and second, they allow to decide whether a word s in Σ^* belongs to the language of the grammar.

Regular languages Regular languages are those where $\equiv$ has a finite number of equivalence classes[1]. They can also be defined as the ones that can be described by deterministic finite automata (DFA) [39]. A DFA A is formally defined as a tuple $(Q,\ \Sigma,\ \tau,\ q_{\text{in}},\ F)$ where:

1. Q is a finite set of states.

[1]The Chomsky hierarchy defines these languages as the languages that are generated by Type-3 grammars (regular grammars) [40].

2. Σ is a finite set of input symbols.

3. τ is a transition function that takes as arguments a state and an input symbol and returns a state.

4. q_{in} is a start state, belonging to Q.

5. F is a set of final or accepting states, F being a subset of Q.

We define $\tau^*(q, s)$ to be the natural extension of τ to strings, that is, the state reached by A when going through s starting at state q:

$$\tau^*(q, \lambda) \triangleq q \tag{2.3}$$
$$\tau^*(q, \sigma s) \triangleq \tau^*(\tau(q, \sigma), s) \tag{2.4}$$

We define $\mathcal{L}(q, s)$ as checking if the state reached by A when going through s from state q belongs to F:

$$\mathcal{L}(q, s) \triangleq \mathbb{I}(\tau^*(q, s) \in F) \tag{2.5}$$

Finally, we denote $\tau^*(s)$ and $\mathcal{L}(s)$ the state q reached when going through s from the initial state q_{in} and checking if $q \in F$, respectively. Then, A defines the regular grammar such that:

$$A(u) \triangleq \mathcal{L}(u) \tag{2.6}$$

The congruence relation $\equiv$ 2.1 can also be defined over Q as follows:

$$\forall q, q' \in Q.\ q \equiv q' \overset{\triangle}{\iff} [\forall w \in \Sigma^*.\ \mathcal{L}(q, w) = \mathcal{L}(q', w)] \tag{2.7}$$

This equivalence relation is also a right congruence with respect to τ:

$$\forall q, q' \in Q.\ q \equiv q' \implies \forall \sigma \in \Sigma.\ \tau(q, \sigma) \equiv \tau(q', \sigma) \tag{2.8}$$

Where the congruence class of $q \in Q$ is denoted by $[\![q]\!]$.

From the description in [39], the key point is that these models characterize the languages, meaning they provide a constructive way of describing them, and recognizing their elements (that is, checking whether a sequence of symbols does belong to the language), any function that allows for the recognition of sequences in a set is called an acceptor of a language.

A simple example of a regular language is presented in Figure 2.1. This language is described by the regular expresion $(ab)^*$, with:

1. $Q = \{0, 1, 2\}$.
2. $\Sigma = \{a, b\}$ and λ being the empty sequence.
3. τ as presented graphically in the figure or tabularly in Table 2.1.
4. $q_{\text{in}} = 0$ (indicated with an incoming arrow).
5. $F = \{0\}$ (indicated with a double circle).

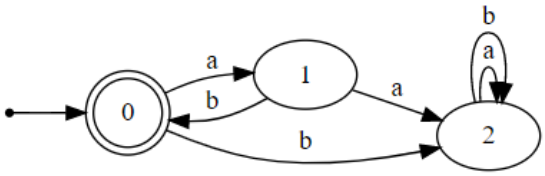


Figure 2.1: Example of automaton

τ	**a**	**b**
0	1	2
1	2	0
2	2	2

Table 2.1: Table of transition function τ of automaton in Fig. 2.1

2.2 Minimally adequate teachers

The L* algorithm, presented in [25] is an instance of a general learning model, refered to as minimally adequate teacher (MAT) also presented in that work. Under this learning model, the learner is supposed to learn by being able to interact with the teacher through two interfaces: A *membership query* (MQ), that is a boolean response if a given sequence is accepted by the language known by the teacher, and an *equivalence test* (EQ), that is a function that compares the target language and the inferred one, if they are equivalent the test returns true, if not it returns a counterexample (a word belonging to one of the languages but not the other).

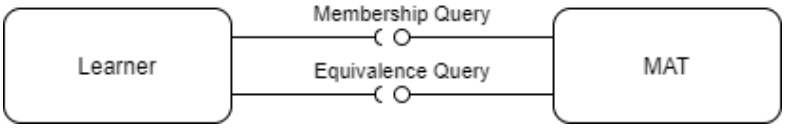


Figure 2.2: Learning model interfaces

In figure 2.2 a representation of the abstractions and interactions present in the learning model is shown.

In 1 the pseudocode of the learning model is presented.

Algorithm 1: Minimally Adequate Teacher Learning Model

Input : A symbolic alphabet Σ
Output: Hypothesis H

```
1  Initialize internal structures;
2  repeat
3      while unable to build Hypotesis do
4          Ask MQ
5      H ← BuildHypothesis();
6      Answer, Counterexample ← EQ(H);
7      if Answer ≠ Yes then
8          ProcessCounterexampe(Counterexample)
9  until Answer = Yes;
10 return H;
```

The interactions between learner and teacher are as follows: Learners mantain structures that register interactions with the MAT. These structures are used to keep track of which words are and are not accepted by the target language. The construction of such structures varies from algorithm to algorithm but is always done in an iterative way by asking the teacher membership queries through the Membership Oracle (MQ) of different words. Once the structure is complete enough to allow building a hypothesis of the target language the learner proceeds to construct the conjectured hypothesis and then asks the Equivalence Oracle (EQ) whether it is equivalent to the target one. If the answer is yes, it terminates and returns the learned concept H. If the answer is no, then it receives a counterexample that proves the hypothesis is wrong, and it proceeds to adjust the internal structure with this new counter example.

Under this learning model several works have proposed variants, to name a few (restricting the target to regular languages) L*Col [41], NL^* [42], Bounded-L* [32], Kearns-Vazirani [43], Rivest-Shapire [44], Observation Pack [45], TTT [46].

2.2.1 L*

As mentioned before, a foundational algorithm in the category of active learning is Angluin's L* [25].

L* constructs a DFA by interacting with a Minimum Adequate Teacher (MAT) that exposes two operations: a *membership query* (MQ), that is a boolean response if a given sequence is accepted by the language known by the teacher, and an *equivalence test* (EQ), that is a function that compares the target language and the inferred one, if they are equivalent the test returns true, if not it returns a counterexample (a word belonging to one of the languages but not the other).

The L* pseudocode is presented in algorithm 2. The way the algorithm achieves the learning is as follows. It builds a table of observations by interacting with the MAT. This table is used to keep track of which words are and are not accepted by the target language. The construction of this table is done in an iterative way by asking the teacher membership queries through the Membership Oracle (MQ) of different words in order to fill the Observation Table (OT).

Algorithm 2: L* learning algorithm

```
   Input  : A symbolic alphabet Σ
   Output: DFA 𝒜
 1 Lstar-Initialise;
 2 repeat
 3     while OT is not closed or not consistent do
 4         if OT is not closed then
 5             OT ← Lstar-Close(OT);
 6         if OT is not consistent then
 7             OT ← Lstar-Consistent(OT);
 8     𝒜← Lstar-BuildAutomaton(OT);
 9     Answer ← EQ(𝒜);
10     if Answer ≠ Yes then
11         OT ← Lstar-UseEQ(OT, Answer);
12 until Answer = Yes;
13 return 𝒜;
```

The information that is in the observation table has three characteristics. A nonempty finite prefix-closed set of strings (every prefix of every member is also

a member of the set), a nonempty finite suffix-closed set of strings (every suffix of every member is also a member of the set), and a finite function that maps a string to either 1 or 0 if it is a member of our target language or not respectively.

The observation table is composed by two sets of rows: the 'upper' rows (or top part, that we will call **RED** following De la Higuera's notation [23]), that represent the elements of the prefix-closed set of strings mentioned earlier, and the 'lower' rows (or bottom part, that we will call **BLUE**), which represent the same elements of this set but concatenated with the set of letters in the language alphabet. On the other hand, columns represent a suffix-closed set of strings, and each cell represents the membership relationship, both also mentioned earlier. An example of the observation table is presented in Table 2.2b.

The observation table is first initialized by building one **RED** row (Ffor the empty word λ) and one **BLUE** row for each symbol in the alphabet Σ (length-one words). Then the iterative process begins.

In order to be able to make sense out of the table, it needs to comply with two properties. First of all, it needs to be closed. The table is considered closed if, for every row in the bottom part of the table, there is an equal row in the top part. The second property is consistency. A table is considered consistent if for every pair of rows in the top part of the table (**RED**) with the same values (same order of 0s and 1s), then all pairs of extensions with the same letter of the alphabet must have the same row in the table. Precisely, a table is consistent if for every different row in **RED**, for every symbol $\sigma \in \Sigma$ if $OT[u] = OT[w]$ then $OT[u\sigma] = OT[w\sigma]$.

If the table is not closed, the algorithm moves to the **RED** part a row in the **BLUE** part that does not have an equal row in the **RED** part and adds to the **BLUE** set all the rows corresponding to the extensions of its associated word with every letter of the alphabet.

To make it consistent, the algorithm expands the original set of suffixes with the letter that makes their corresponding extensions different (a $\sigma \in \Sigma$ such that $OT[u] = OT[w]$ but $OT[u\sigma] \neq OT[w\sigma]$). This is done in order to differentiate between the two words that had the same row values.

Once the table is closed and consistent, the algorithm proceeds to construct the conjectured DFA and then asks the oracle whether it is equivalent to the target one. If the answer is yes, it terminates and returns the learned DFA. If the answer is no, then it receives a counterexample that proves the DFA is wrong, and it proceeds to extend the observation table with this new counter example. This

extension is done by adding every prefix of the counterexample to **RED**, and for each prefix its concatenation with every symbol in Σ to **BLUE** (given that the concatenation is not a prefix).

2.2.1.1 An L* run

Let us take as an example the regular language presented in Figure 2.1, described by the regular expression $(ab)^*$.

First, the algorithm constructs the table as presented in 2.2a. As the table is not closed (not every row in **BLUE** has a representation in **RED**), the algorithm proceeds to close it. To do that, one of the elements in **BLUE** that has not a representative in **RED** is selected, for example a, and moves it to **RED**, adding to **BLUE** its concatenation to every symbol (aa and ab, then the holes are filled. The resulting table can be seen in Table 2.2b.

As the observation table is now closed (the previous step solved this problem) and consistent an automaton can be built.

To build an automaton out of the table, the states are represented by every unique row in **RED**, the final states are those corresponding to the rows w where $OT[w][\lambda] = 1$, and rejecting states are those rows u where $OT[u][\lambda] = 0$. Finally the transition function is defined as: $\tau(q_u, \sigma) = w$ if $OT[u\sigma] = OT[w]$. The resulting automaton is presented in Figure 2.3.

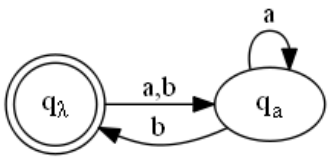


Figure 2.3: First proposed automaton in an L*example run.

This automaton is then presented to the teacher via the **EQ**, which can be implemented by the table-filling algorithm [39]. This query results negative, as the regular language that the conjectured automaton represents is not the same as the target one. Let us suppose that the counterexample returned by the teacher is 'bb'.

OT_0	λ
λ	1
a	0
b	0

(a)

OT_1	λ
λ	1
a	0
b	0
aa	0
ab	1

(b)

OT_2	λ
λ	1
a	0
b	0
bb	0
aa	0
ab	1
ba	0
bba	0
bbb	0

(c)

OT_3	λ	b
λ	1	0
a	0	1
b	0	0
bb	0	0
aa	0	0
ab	1	0
ba	0	0
bba	0	0
bbb	0	0

(d)

Table 2.2: Observation tables during an L*example run.

Now, the learner proceeds to process the counterexample. This is done by adding the counterexample and all its prefixes to **RED**, and at the same time adding for each prefix u and for all symbols σ, $u\sigma$ to **BLUE**, given that $u\sigma$ is not a prefix of the counterexample. Then holes are filled, resulting in the Table 2.2c

The table remains closed, however it is not consistent, as two **RED** rows have different resulting rows if they are added a symbol. To be concrete, $OT[a] = OT[b]$, however $OT[ab] \neq OT[bb]$). This can be informally interpreted as "they seem to be the same state in the table, however they are not", so they have to be separated. This separation is achieved by adding the symbol that makes them differ to the columns of the observation table (in this case symbol b). The symbol is added, holes are filled, the result is Table 2.2d.

The last table is closed and consistent, the conjectured automaton is finally equivalent to the target one, so **EQ** outputs true and L* finishes and the DFA present in Figure 2.4, which is equivalent to the target one, is returned.

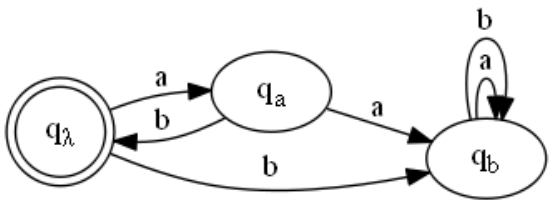


Figure 2.4: Output automaton in an L* example run.

3 Learning approximations

Algorithms presented in 2 assume that the teacher has a means of comparing the target concept and the proposed hypothesis in an exact way, which is not always the case. In her paper [25] Angluin proposes to resort to Valiant's Probably Approximately Correct (PAC) framework [31, 47] in the scenarios where there is no access to a deterministic model comparison. In the case the target language is not regular or its size is too big to compute, works like [32, 48] propose to impose bounds to the execution in order to guarantee termination. This two ideas will be developed in the following sections.

3.1 PAC learning

Let us first give some preliminary definitions. There is a *universe* of examples which is denoted $\mathcal{X}$. A *concept* C is a function $\mathcal{X} \to \mathcal{O}$, as an example C could be a language acceptor, which means $\mathcal{X} = \Sigma^*$ and $\mathcal{O} = \{0, 1\}$. For the ease of notation, we will indifferently treat concepts as functions, relations or sets, so write $x \in C \iff C(x) = 1$ and $\overline{C}$ to denote its complement $\overline{C}(x) = 1 - C(x)$, and use set operations such as $\cap$ and $\cup$ for intersection and union.

A concept class $\mathcal{C}$ is a set of concepts. Given an *unknown* concept $C \in \mathcal{C}$, the purpose of a *learning* algorithm is to output a hypothesis $H \in \mathcal{H}$ that *approximates* C, where $\mathcal{H}$, called *hypothesis space*, is a class of concepts possibly different from $\mathcal{C}$. The *prediction error* $\mathcal{E}$ of a hypothesis H with respect to the unknown concept C measured in terms of the probability distribution $\mathcal{D}$ is the probability of an example $x \in \mathcal{X}$, drawn from $\mathcal{D}$, to be in differently categorized by C and H. Formally:

$$\mathcal{E}_{\mathcal{D},C}(H) = \mathbb{P}_{x\sim\mathcal{D}}\left[x \in C \oplus H\right] = \mathbb{P}_{x\sim\mathcal{D}}\left[C(x) \neq H(x)\right] \tag{3.1}$$

where $C \oplus H$ is the symmetric difference between H and C: $C \cap \overline{H} \cup H \cap \overline{C}$.

An *oracle* $\mathbf{EX}_{\mathcal{D},C}$ draws identically and independently distributed (i.i.d.) examples from $\mathcal{X}$ following $\mathcal{D}$, and associates them labels according to $C(x)$. Repeated calls to $\mathbf{EX}_{\mathcal{D},C}$ are independent of each other.

A Probably Approximately Correct (PAC) learning algorithm [31, 47, 49] takes as input an *approximation* parameter $\varepsilon \in (0,1)$, a *confidence* parameter $\delta \in (0,1)$, a *target* concept $C \in \mathcal{C}$, an oracle $\mathbf{EX}_{\mathcal{D},C}$, and a hypothesis space $\mathcal{H}$, and if it terminates, it outputs an $H \in \mathcal{H}$ which satisfies $\mathbb{P}_{x\sim\mathcal{D}}[C(x) \neq H(x)] \leq \varepsilon$ with confidence at least $1-\delta$. Formally:

$$\mathbb{P}[\mathcal{E}_{\mathcal{D},C}(H) > \varepsilon] < \delta \tag{3.2}$$

The output H of a PAC-learning algorithm is said to be an ε-approximation of C with confidence at least $1-\delta$, or equivalently, an (ε,δ)-approximation of C.

A *distribution-free* algorithm is one that works for every $\mathcal{D}$. Hereinafter, we will focus on distribution-free algorithms, so we will omit $\mathcal{D}$.

Some useful properties of PAC [26, 27] are presented below:

Lemma 3.1. *Let $C \in \mathcal{C}$ and $H \in \mathcal{H}$ such that H is an (ε,δ)-approximation of C. For any subset $X \subseteq C \oplus H$, we have that $\mathbb{P}_{x\sim\mathcal{D}}[x \in X] \leq \varepsilon$ with confidence $1-\delta$.*

Proof. *For any subset $X \subseteq C \oplus H$, it holds that $\mathbb{P}_{x\sim\mathcal{D}}[x \in X] \leq \mathbb{P}_{x\sim\mathcal{D}}[x \in C \oplus H]$. It follows that $\mathbb{P}_{x\sim\mathcal{D}}[x \in C \oplus H] \leq \varepsilon$ implies $\mathbb{P}_{x\sim\mathcal{D}}[x \in X] \leq \varepsilon$. Now, for any $S \subseteq \mathcal{X}$ satisfying $S \cap (C \oplus H) = \varnothing$, we have that $S \cap X = \varnothing$. Hence, any sample $S \sim \mathcal{D}^m$ drawn by $\mathbf{EQ}_{\mathcal{D},C}$ that ensures $\mathbb{P}_{x\sim\mathcal{D}}[x \in C \oplus H] \leq \varepsilon$ with confidence $1-\delta$ also guarantees $\mathbb{P}_{x\sim\mathcal{D}}[x \in X] \leq \varepsilon$ with confidence $1-\delta$.*

Proposition 3.1. *Let $C \in \mathcal{C}$ and $H \in \mathcal{H}$ such that H is an (ε,δ)-approximation of C. For any $X \subseteq \mathcal{X}$:*

$$\mathbb{P}_{x\sim\mathcal{D}}\left[x \in C \cap \overline{H} \cap X\right] \leq \varepsilon \tag{3.3}$$

$$\mathbb{P}_{x\sim\mathcal{D}}\left[x \in \overline{C} \cap H \cap X\right] \leq \varepsilon \tag{3.4}$$

with confidence at least $1-\delta$.

Proof. *From Lemma 3.1 because $C \cap \overline{H} \cap X$ and $\overline{C} \cap H \cap X$ are subsets of $C \oplus H$.*

3.2 PAC-based MAT learning

In this setting, the MAT learner is the same, however, the teacher resorts to a statistical test in order to compare the languages. This test is explained below.

We denote C and H the target and output grammar, respectively. The learner uses **EQ** and **MQ**. Each time **EQ** is called, it must draw a sample of a size large enough to ensure a *total* confidence of the algorithm of at least $1 - \delta$. That is, whenever the statistical test is passed, it is possible to conclude that the candidate output is ε-approximately correct with confidence at least $1 - \delta$.

Say **EQ** is called at iteration i. In order to guarantee the aforementioned property, a sample S_i of size of size μ_i defined as follows [25]:

$$\mu_i = \left\lceil \frac{1}{\varepsilon} \left(i \ln 2 - \ln \delta \right) \right\rceil \tag{3.5}$$

It is of interest to us to guarantee that independently of the moment i that the test is passed, there is a limit δ to the probability of H not being an ε-approximation of C. Following [50], the probability of a hypotheses H_i passing the PAC test and not being an ε-approximation of C is at most $(1 - \varepsilon)^{\mu_i}$.

Theorem 3.1. *The probability that at any stage algorithm 3 terminates with an output H that is not an ε-approximation of C is at most δ.*

Proof.

$$\mathbb{P}\left[\mathcal{E}_{\mathcal{D},C}(H) > \varepsilon\right] \leq \sum_{i>0} \mathbb{P}\left[\mathcal{E}_{\mathcal{D},C}(H_i) > \varepsilon\right] < \sum_{i>0} (1-\varepsilon)^{\mu_i} < \sum_{i>0} 2^{-i}\delta < \delta$$

The PAC-based MAT learner pseudocode is presented in algorithm 3.

Algorithm 3: PAC MAT learner

Input : ε, δ
Output: Hypothesis H

```
1  Initialize internal structures;
2  iterationNumber ← 0 ;
3  repeat
4      while unable to build Hypotesis do
5          Ask MQ
6      H ← BuildHypothesis();
7      Answer, Counterexample ← PAC-EQ(H, ε, δ, iterationNumber);
8      if Answer ≠ Yes then
9          ProcessCounterexample(Counterexample)
10     iterationNumber ← iterationNumber + 1;
11 until Answer = Yes;
12 return H;
```

Corollary 3.1. *If the target concept is regular then PAC* $\mathrm{L}^\star$ *terminates.*

Proof. *The detailed proof is in [25].*

3.3 Bounded PAC-based MAT learning

Most of scenarios of interest are those which C is unknown, meaning even the nature of the concept is out of reach, it could even be the case that the hypothesis space of the learner does not contain the target concept, mathematically $C \notin \mathcal{H}$.

In such scenarios, MAT algorithms described before have no termination guarantee. In other words, reaching an **EQ** returning true is not one of their possibilities, instead, other stopping criteria may be selected.

Some works propose the selection of iteration numbers or time in milliseconds [51], however that selection is arbitrary. Other way of selecting the stopping criteria would be associated to internal data structures sizes of the MAT Learner, or complexity of H [32].

Agnostic to the termination criterion *stop*, a Bounded PAC-based MAT learner is shown in 4.

Algorithm 4: Bounded PAC MAT learner

Input : *stop*, ε, δ, Σ
Output: Hypothesis H

```
1  Initialise internal structures;
2  iterationNumber ← 0 ;
3  repeat
4      while unable to build Hypotesis do
5          Ask MQ
6      H ← BuildHypothesis();
7      Answer, Counterexample ← PAC-EQ(H, ε, δ, iterationNumber);
8      if Answer ≠ Yes then
9          ProcessCounterexample(Counterexample)
10     BoundReached ← stop();
11     iterationNumber ← iterationNumber + 1;
12 until Answer = Yes or BoundReached;
13 return H;
```

3.3.1 Analysis of the approximation error

Upon termination, a Bounded PAC learner may output hypothesis H which fails to pass **EQ**. In such cases, H and the target language eventually disagree in $k > 0$ sequences of the sample S drawn by **EQ**. Therefore, it is important to analyze in detail the approximation error incurred by the learner in such case. In order to do so, let us start by giving the following definition:

$$\phi_i(k) = (\mu_i - k)^{-1}\left(\mu_i\varepsilon + \ln\binom{\mu_i}{k}\right) \tag{3.6}$$

for all $i \in \mathbb{N}, i \geq 1$. Notice that for all $k \in [0, \mu_i)$, $\phi_i(k) \geq \varepsilon$, and $\phi_i(0) = \varepsilon$.

Theorem 1. *For any target concept C, if after n iterations Bounded PAC learner returns a DFA A with $k \in \mathbb{N}$ **EQ**-divergences, such that $\widetilde{\epsilon}(k) \in (0,1)$, then A is an $(\widetilde{\epsilon}(k), \delta)$-approximation of C, where*

$$\widetilde{\epsilon}(k) = \max\{\phi_i(k) \mid 1 \leq i \leq n\} \tag{3.7}$$

Proof. *Let $S_i \sim \mathcal{D}^{\mu_i}$, we define:*

$$K(S_i) = \sum_{u \in S_i} \mathbb{1}_{A(u) \neq C(u)}$$

Using the same arguments as [25], we have that:

$$\mathbb{P}\big[\mathcal{E}_{\mathcal{D},C}(A) > \widetilde{\epsilon}(k)\big] \leq \sum_{i=1}^{n} \mathbb{P}_{S_i \sim \mathcal{D}^{\mu_i}}\big[K(S_i) = k \ ; \ \mathcal{E}_{\mathcal{D},C}(A) > \widetilde{\epsilon}(k)\big] \tag{3.8}$$

Now, for every $1 \leq i \leq n$:

$$\begin{aligned}\mathbb{P}_{S_i \sim \mathcal{D}^{\mu_i}}\big[K(S_i) = k \ ; \ \mathcal{E}_{\mathcal{D},C}(A) > \widetilde{\epsilon}(k)\big] &= \binom{\mu_i}{k}\big(1 - \mathcal{E}_{\mathcal{D},C}(A)\big)^{\mu_i - k}\ \mathcal{E}_{\mathcal{D},C}(A)^k \\ &< \binom{\mu_i}{k}\big(1 - \widetilde{\epsilon}(k)\big)^{\mu_i - k}\end{aligned}$$

Using the inequality $1 - u < e^{-u}$, it follows that:

$$\mathbb{P}_{S_i \sim \mathcal{D}^{\mu_i}}\big[K(S_i) = k \ ; \ \mathcal{E}_{\mathcal{D},C}(A) > \widetilde{\epsilon}(k)\big] < \binom{\mu_i}{k} e^{-\widetilde{\epsilon}(k)(\mu_i - k)} \tag{3.9}$$

Therefore, by Equations (3.6) and (3.7):

$$\phi_i(k) = (\mu_i - k)^{-1}\left(\mu_i\varepsilon + \ln\binom{\mu_i}{k}\right) \leq \widetilde{\epsilon}(k)$$

By definition of μ_i (Equation (3.10)), this entails:

$$-\widetilde{\epsilon}(k)(\mu_i - k) + \ln\binom{\mu_i}{k} \leq -\mu_i\varepsilon \leq -i\ln 2 + \ln\delta$$

Then,

$$\binom{\mu_i}{k} e^{-\widetilde{\epsilon}(k)(\mu_i - k)} \leq 2^{-i}\delta \tag{3.10}$$

Thus, from Equations (3.8)–(3.10), it follows that:

$$\mathbb{P}\big[\mathcal{E}_{\mathcal{D},C}(A) > \widetilde{\epsilon}(k)\big] < \sum_{i=1}^{n} 2^{-i}\delta < \delta \tag{3.11}$$

Hence, A is an $(\widetilde{\epsilon}(k), \delta)$-approximation of C.

Corollary 3.2. *If the algorithm terminates with $k = 0$ errors, then A is an (ε, δ)-approximation of C.*

Proof. *Immediate from 1.*

Remark. *It is important to notice that this result (from [27]) improves the kind of "forensics" analysis developed in [32, 33], which concentrates on studying the approximation error of the actual DFA returned by Bounded-*$\mathrm{L}^\star$ *on a particular run, rather than on* any *outcome of the algorithm, as it is stated by Theorem 1.*

3.4 Bounded L*

Similarly to the description of L* presented in [23] and sketched in subsection 2.2.1, the algorithm Bounded-L* [32](Algorithm 5) is an instance of a Bounded MAT algorithm and can be described as follows:

Algorithm 5: Bounded-L*

Input : MaxQueryLength, MaxStates, ε, δ, Σ
Output: DFA $\mathcal{A}$

```
1  Lstar-Initialize;
2  repeat
3      while OT is not closed or not consistent do
4          if OT is not closed then
5              OT, QueryLengthExceeded ← Lstar-Close(OT);
6          if OT is not consistent then
7              OT, QueryLengthExceeded ← Lstar-Consistent(OT);
8      if not QueryLengthExceeded then
9          A← Lstar-BuildAutomaton(OT);
10         Answer ← EQ(A);
11         MaxStatesExceeded ← STATES(A)>MaxStates;
12         if Answer ≠ Yes and not MaxStatesExceeded then
13             OT ← Lstar-UseEQ(OT, Answer);
14     BoundReached ← QueryLengthExceeded or MaxStatesExceeded;
15 until Answer = Yes or BoundReached;
16 return A;
```

The observation table OT is initialised by Lstar-Initialize in the same manner that it is for L*. This step consists in building the structure of the observation table OT as proposed by Angluin. Then the construction of hypotheses begins.

If OT is not *closed* an extra row is added by the Lstar-Close procedure. If OT is *inconsistent*, an extra column is added by the Lstar-Consistent procedure. Both procedures call **MQ** to fill the holes in the observation table. The length of these queries may exceed the maximum query length, in which case the QueryLength-Exceeded flag is set to true.

When the table is closed and consistent, and in the case that the maximum query length was not exceeded, an equivalence query **EQ** is made and the number

of states of the automaton is compared to the maximum number of states bound. If **EQ** is unsuccessful and the maximum number of states was not reached, new rows are added by Lstar-UseEQ, using the counterexample contained in *Answer*.

Finally, if the hypothesis passes the test or one of the bounds was reached, the algorithm stops and returns the last proposed automaton.

In Fig. 3.1 a diagram of the teacher-learner setting is presented.

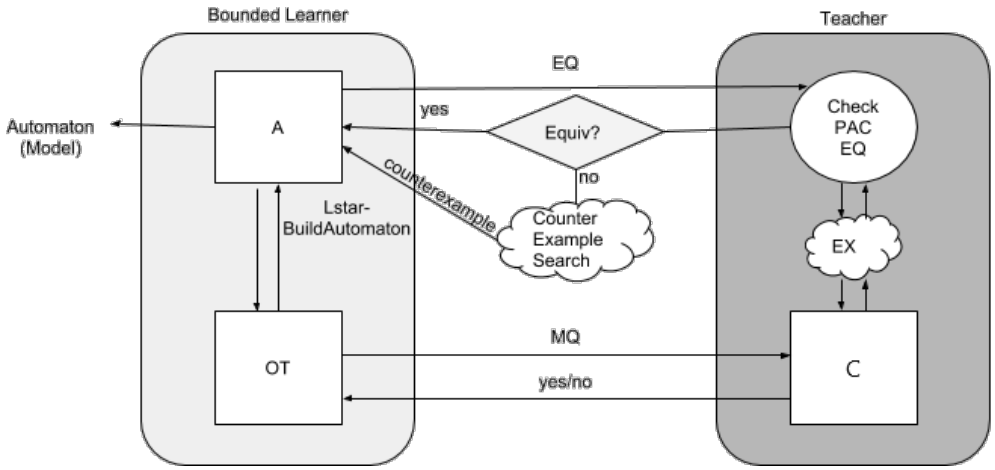


Figure 3.1: Teacher-Bounded Learner setting diagram

It is important to note that lines 3-10 in algorithm 5 correspond to line 4 in algorithm 4, lines 5,8 and 14 in algorithm 5 are responsible of checking if the stopping criteria were met (line 12 in algoritm 4), line 16 is responsible of processing the counterexample (line 10 in algoritm 4) and line 12 from 5 corresponds to line 7 in algoritm 4).

Corollary 3.3. *For any $C \subseteq \Sigma^*$, if Bounded-*L^* *terminates with a DFA A which passes a PAC-***EQ***, then A is an (ε, δ)-approximation of C.*

Proof. *Straightforward from Theorem 3.1 and 1.*

4 Verification through learning

This chapter analyzes the following problem: given an *unknown* concept $C \in \mathcal{C}$, and a *known* property $\psi \in \mathcal{H}$ to be checked on C, we want to answer whether $C \subseteq \psi$ holds, or equivalently $C \cap \overline{\psi} = \varnothing$. To achieve this we resort to the learning techniques presented in previous chapters. These techniques are then applied to *neural acceptors*, that is, ANN that recognize languages.

4.1 Post-learning verification

One way of verifying wether $C \subseteq \psi$ holds in a black-box setting consists in resorting to a *model-checking* approach. That is, first learn a hypothesis $H \in \mathcal{H}$ of C with a PAC-learning algorithm and then check whether H satisfies property ψ. We call this approach *post-learning verification*, also referred to as *model-then-verify*. In order to be feasible, there must be an effective procedure for checking $H \cap \overline{\psi} = \varnothing$.

Assume an algorithm for checking emptiness exists. The relevant question now is what could be said about the outcome of procedure. The following proposition [26] proves that whichever the outcome of the decision procedure for $H \cap \overline{\psi}$, the probability of the same result not being true for C is smaller than ε, with confidence at least $1 - \delta$.

Proposition 4.1. *Let $C \in \mathcal{C}$ and $H \in \mathcal{H}$ such that H is an (ε, δ)-approximation of C. For any $\overline{\psi} \in \mathcal{H}$:*

1. *if $H \cap \overline{\psi} = \varnothing$ then $\mathbb{P}_{x \sim \mathcal{D}}\left[x \in C \cap \overline{\psi}\right] \leq \varepsilon$, and*
2. *if $H \cap \overline{\psi} \neq \varnothing$ then $\mathbb{P}_{x \sim \mathcal{D}}\left[x \in \overline{C} \cap H \cap \overline{\psi}\right] \leq \varepsilon$,*

with confidence at least $1 - \delta$.

Proof. 1. *If* $H \cap \overline{\psi} = \varnothing$ *then* $\overline{\psi} = \overline{H} \cap \overline{\psi}$. *Thus,* $C \cap \overline{\psi} = C \cap \overline{H} \cap \overline{\psi}$ *and from Proposition 3.1(3.3) it follows that* $\mathbb{P}_{x \sim \mathcal{D}}\left[x \in C \cap \overline{H} \cap \overline{\psi}\right] \leq \varepsilon$, *with confidence at least* $1 - \delta$.
2. *If* $H \cap \overline{\psi} \neq \varnothing$, *from Proposition 3.1(3.4) we have that* $\mathbb{P}_{x \sim \mathcal{D}}\left[x \in \overline{C} \cap H \cap \overline{\psi}\right] \leq \varepsilon$, *with confidence at least* $1 - \delta$.

When applied in practice, an important inconvenience of this approach is that whenever ψ is found by the model-checker not to hold on H, even if with small probability, counterexamples found on H may not be counterexamples in C. Therefore, whenever that happens, we would need to resort to **EX** to draw examples from $H \cap \overline{\psi}$ and call **MQ** to figure out whether they belong to C in order to trying finding a concrete counterexample in C.

From a computational perspective, in particular in the application scenario of verifying neural acceptors, we should be aware that the learned hypothesis could be too large and that the running time of the learning algorithm adds up to the running time of the model-checker, thus making the overall procedure impractical.

Last but not least, this approach could only be applied for checking properties for which there exists a model-checking procedure in $\mathcal{H}$. In our context, it will prevent verifying non-regular properties.

4.2 On-the-fly verification via learning

To overcome the aforementioned issues, rather than learning an (ε, δ)-approximation of C, we proposed to use the PAC-learning algorithm to learn an (ε, δ)-approximation of $C \cap \overline{\psi} \in \mathcal{C}$. This approach is called *on-the-fly verification through learning* [26, 27].

Indeed, this idea can be extended to cope with any verification problem which can be expressed as checking the emptiness of some concept $\Psi(C) \in \mathcal{C}$, which in the simplest case is $C \cap \overline{\psi}$. In such context, we have the following, more general, result.

Proposition 4.2. *Let* $C \in \mathcal{C}$, $\Psi(C) \in \mathcal{C}$ *and* $H \in \mathcal{H}$ *such that* H *is an* (ε, δ)*-approximation of* $\Psi(C)$. *Then:*

1. *if* $H = \varnothing$ *then* $\mathbb{P}_{x \sim \mathcal{D}}\left[x \in \Psi(C)\right] \leq \varepsilon$, *and*
2. *if* $H \neq \varnothing$ *then* $\mathbb{P}_{x \sim \mathcal{D}}\left[x \in H \backslash \Psi(C)\right] \leq \varepsilon$,

with confidence at least $1-\delta$.

Proof. *Straightforward since* $\mathbb{P}_{x\sim\mathcal{D}}[x \in \Psi(C) \oplus H] \leq \varepsilon$, *with confidence at least* $1-\delta$, *by the fact that* H *is an* (ε,δ)*-approximation of* $\Psi(C)$.

Proposition 4.2 proves that checking properties during the learning phase yields the same theoretical probabilistic assurance as doing it afterwards on the learned model of the target concept C. Nevertheless, from a practical point of view, on-the-fly property checking through learning has several interesting advantages over post-learning verification. First, no model of C is ever explicitly built which may result in a lower computational effort, both in terms of running time and memory. Therefore, this approach could be used in cases where it is computationally too expensive to construct a hypothesis for C. Second, there is no need to resort to external model-checkers. The approach may even be applied in contexts where such algorithms do not exist. Indeed, in contrast to post-learning verification, an interesting fact in on-the-fly checking is that in the case the PAC-learning algorithm outputs a nonempty hypothesis, it may actually happen that the oracle **EX** draws an example belonging to $\Psi(C)$ at some point during the execution, which constitutes a concrete, real evidence of $\Psi(C)$ not being empty with certainty.

4.2.1 Characterization of the error

Let us recall that the black-box checking problem consists in verifying whether $\Psi(C) = \varnothing$. Solving this task with on-the-fly checking through learning using Bounded-L* as the learning algorithm yields a DFA which is a PAC-approximation of $\Psi(C)$. Indeed, the output DFA serves to characterize the eventual wrong classifications made by the neural acceptor C in an operational and visual formalism. As a matter of fact, Bounded-L* ensures that whenever the returned regular language is nonempty, the language in the black-box is also nonempty. This result is proven below.

Proposition 4.3. *For any* $C \subseteq \Sigma^*$ *and* $i > 1$, *if Bounded-*L$^*(n,\varepsilon,\delta)$ *builds an automaton* $A_i \neq \varnothing$ *at iteration* i, *then* $C \neq \varnothing$.

Proof. *Suppose* $A_i \neq \varnothing$. *Then,* A_i *has at least one accepting state. By construction,* $\exists\, u \in \Sigma^*$ *such that* $OT_i[u][\lambda] = 1$. *For this to be true, it must have occurred a positive membership query for* u *at some iteration* $j \in [1,i]$, *that is,* $\mathbf{MQ}_j(u) = 1$. *Hence,* $u \in C$. *This proves that* $C \neq \varnothing$.

This result is important because it entails that whenever the output for the target language $C \cap \overline{\psi}$ is nonempty, C does not satisfy ψ. Moreover, for every entry of the observation table such that $OT[u][v] = 1$, the sequence $uv \in \Sigma^*$ is a counterexample.

Corollary 4.1. *For any $C, \Psi(C) \subseteq \Sigma^*$, if Bounded-L$^*(n, \varepsilon, \delta)$ returns a DFA $A \neq \varnothing$, then $\Psi(C) \neq \varnothing$. Besides, $\forall\ u, v \in \Sigma^*$ if $OT[u][v] = 1$ then $uv \in \Psi(C)$.*

Proof. *Straightforward from Proposition 4.3.*

Indeed, from Proposition 4.3, it could be argued that Bounded-L* for $\Psi(C)$ could finish as soon as OT has a positive entry, yielding a witness of $\Psi(C)$ being nonempty. However, stopping Bounded-L* at this stage would prevent providing a more detailed, explanatory, even if approximate, characterization of the set of misbehaviors.

Theorem 1 and Corally 4.1 can be combined to show the theoretical guarantees yielded by Bounded-L* when used for black-box property checking through learning.

Theorem 4.1. *For any $C, \Psi(C)$, if Bounded-L$^*(\ell, n, \varepsilon, \delta)$ returns a DFA A with $k \in \mathbb{N}$ **EQ**-divergences and $\widetilde{\epsilon}(k) \in (0, 1)$, then:*

1. *A is an $(\widetilde{\epsilon}(k), \delta)$-approximation of $\Psi(C)$.*
2. *If $A \neq \varnothing$ or $k > 0$, then $\Psi(C) \neq \varnothing$.*

Proof. *1. Straightforward from Theorem 1.*
2. By Corollary 4.1, it follows that $A \neq \varnothing$ implies $\Psi(C) \neq \varnothing$. Let $A = \varnothing$ and $k > 0$. By the fact that $k > 0$, we have that $A \oplus \Psi(C) \neq \varnothing$. Since $A = \varnothing$, it results that $\varnothing \oplus \Psi(C) = \Psi(C)$. Hence, $\Psi(C) \neq \varnothing$.

5 Application to neural acceptors

The motivation of this chapter is to apply on-the-fly verification of neural acceptors on a number of case studies and compare it to the post-learning approach.

The general setting of this chapter is as follows. The teacher is given $\Psi(C)$. For instance, in order to verify language inclusion, that is, to check whether the language of the RNN C is included in some given language ψ (the property), $\Psi(C)$ is $C \cap \overline{\psi}$. The complement of ψ is actually never computed, since the algorithm only requires evaluating membership. That is, to answer $\mathbf{MQ}(u)$ on $C \cap \overline{\psi}$ for a word $u \in \Sigma^*$, the teacher evaluates $\psi(u)$, complements its output, and evaluates the conjunction with the output of $C(u)$. It is straightforward to generalize this idea to any Boolean combination of C with other concepts $\psi_1, \ldots, \psi_r$. Every concept ψ_j may be any kind of property, even a nonregular language, such as a context-free grammar, or an RNN.

We carried out controlled experiments where RNN were trained with sample datasets from diverse sources such as: known automata, context free grammars, and domain specific data as a way of validating the approach. However, it is important to remark that context-free grammars or DFAs are artifacts only used with the purpose of controlling the experiments. In real application scenarios, they are not assumed to exist at all. Unless otherwise stated, RNN consisted of a two-layer network starting with a single-cell three-dimensional LSTM layer [52] followed by a two-dimensional dense classification layer with a *softmax* activation function. The loss function was *categorical cross-entropy*. They were trained with Adam optimizer, with a default learning rate of 0.5, using two-phase early stopping, with an 80%-20% random split for train-validation of the corresponding datasets. The performance of trained RNN was measured on test datasets. Symbols of the alphabet were represented using one-hot encoding. We stress the fact that knowledge of the internal structure, training process, or training data (except for the alphabet) is by no means required by our approach. This information is provided only to describe the performed controlled experiments.

We applied our approach in three kinds of scenarios which are presented in the following sections.

5.1 Scenario 1

First, we studied RNN trained with sequences generated by context-free grammars (CFG) and checked regular and nonregular properties. In addition, we compared two different RNN trained with sequences from the same language specification, in order to check whether they are actually equivalent. Here, Ψ is a Boolean combination of the RNN under analysis.

5.1.1 Context-free language models

Parenthesis prediction is a typical problem used to study the capacity of RNN for context-free language modelling [53].

First, we randomly generated 550,000 sequences upto length 20 labelled as positive or negative according to whether they belong or not to the following 3-symbol Dyck-1 CFG with alphabet $\{(,), c\}$:

$$S \to S\ T \mid T\ S \mid T \qquad T \to (\ T\ ) \mid () \qquad T \to c \tag{5.1}$$

The RNN was trained using a subset of 500,000 samples until achieving 100% accuracy on the remaining validation set of 50,000 sequences. The following properties were checked:

1. The set of sequences recognized by the RNN C are included in the Dyck 1 grammar in equation 5.1. That is, $\Psi_1(C) = C \cap \overline{\psi_1}$, where $\psi_1 = S$. Recall that $\overline{\psi_1}$ is not computed, since only membership queries are posed.
2. The set of sequences recognized by the RNN C are included in the regular property $\psi_2 = (c)^*$. In this case, $\Psi_2(C) = C \cap \overline{\psi}$.
3. The set of sequences recognized by the RNN C are included in the context-free language $\psi_3 = (^m)^n$ with $m < n$. Here, $\Psi_3(C) = C \cap \overline{\psi_3}$.

Experimental results are shown in Tables 5.1 and 5.2. For each (ε, δ), 5 runs were executed. All runs finished with 0-divergence **EQ**. Execution times are in

Parameters		Running time (in secs)			Mean sample size
ε	δ	**min**	**max**	**mean**	
0.005	0.005	1.984	7.205	3.072	1,899
0.0005	0.005	3.713	10.445	5.997	20,093
0.00005	0.005	7.982	30.470	9.997	203,007
0.00005	0.0005	8.128	36.621	9.919	249,059
0.00005	0.00005	9.625	41.884	12.185	295,111

Table 5.1: Dyck 1: PAC DFA extraction from RNN.

Ψ	Parameters		Running time (in secs)			First positive MQ	Mean sample size
	ε	δ	**min**	**max**	**mean**		
Ψ_1	0.005	0.005	0.004	0.012	0.006	-	1,476
	0.0005	0.005	0.051	0.125	0.067	-	14,756
	0.00005	0.005	0.682	0.833	0.747	-	147,556
	0.00005	0.0005	1.164	1.595	1.340	-	193,607
	0.00005	0.00005	1.272	1.809	1.386	-	239,659
Ψ_2	0.005	0.005	0.031	34.525	5.762	0.099	1,948
	0.0005	0.005	0.397	37.846	10.245	0.084	20,370
	0.00005	0.005	4.713	30.714	6.547	0.825	206,473
Ψ_3	0.005	0.005	0.025	0.966	0.302	0.006	1,899
	0.0005	0.005	0.267	1.985	0.787	0.070	20,093
	0.00005	0.005	4.376	6.479	4.775	0.764	203,007

Table 5.2: Dyck 1: On-the-fly verification of RNN.

secs. The mean sample size refers to the average **EQ** test size at the last iteration of each run. Figures show that on average, the running times exhibited by of on-the-fly property checking were typically smaller than those achieved just to extract an automaton from the RNN. It is important to remark that cases 1) and 3) fall in an undecidable playground since checking whether a regular language is contained in a context-free language is undecidable [39]. For case 1), our technique could not find a counterexample, thus giving probabilistic guarantees of emptiness, that is, of the RNN to correctly modelling the 3-symbol parenthesis language. For cases 2) and 3), PAC DFA of the intersection language are found in all runs, showing the properties are indeed not satisfied. Besides, counterexamples are generated orders of magnitude faster (in average) than extracting a DFA from the RNN alone.

Second, we randomly generated 550,000 sequences upto length 20 labelled as positive or negative according to whether they belong or not to the following 5-symbol Dyck-2 CFG with alphabet $\{(,), [,], c\}$:

$$S \to S\ T \mid T\ S \mid T \qquad T \to (\ T\ ) \mid () \qquad T \to [\ T\ ] \mid [] \qquad T \to c \tag{5.2}$$

The RNN was trained on 500,000 samples until achieving 99.646% accuracy on the remaining validation set of 50,000 sequences. This RNN was checked against its specification. For each (ε, δ), 5 runs were executed, with a timeout of 300s. Experimental results are shown in Tables 5.3 and 5.4. For each configuration, at least three runs of on-the-fly checking finished before the timeout and one was able to find, as expected, the property was not verified by the RNN, exhibiting a counterexample showing it did not model the CFG and yielding a PAC DFA of the wrong classifications.

Parameters		**Running time (in secs)**			**Mean sample size**	**Mean** $\widetilde{\epsilon}$
ε	δ	**min**	**max**	**mean**		
0.005	0.005	2.753	149.214	19.958	1,795	0.00559
0.0005	0.005	23.343	300.000	105.367	18,222	0.04432
0.00005	0.005	42.518	139.763	77.652	186,372	0.16248

Table 5.3: Dyck 2: PAC DFA extraction from RNN.

5.1.2 Checking equivalence between neural acceptors

Following Theorem 4.1, we present a case where it is of interest to check two RNNs against each other. An RNN N_1 is trained with data from a given language L, and

Parameters		Running time (in secs)			First positive MQ	Mean sample size	Mean $\widetilde{\epsilon}$
ε	δ	**min**	**max**	**mean**			
0.005	0.005	0.004	122.388	24.483	90.285	1,504	0.00618
0.0005	0.005	55.084	300.000	215.508	42.462	16,604	0.00895
0.00005	0.005	0.695	324.144	158.195	4.545	166,040	0.00005

Table 5.4: Dyck 2: On-the-fly verification of RNN.

a second RNN N_2 is trained with sequences from a language L' contained L. If both networks, when checked against L are found compliant with it, the following question arises: Are the networks equivalent? And, if the answer is negative, can the divergences be modelled? In order to answer those questions, the property to be checked is expressed as a boolean composition $\Psi(N_1, N_2) = N_1 \equiv N_2$.

To illustrate this use case, an RNN N_1 was trained with data from Tomita's 5th grammar [54] (Fig. 5.1) until it reached a 100% accuracy both in all data. Similarly, a second network N_2, with the same characteristics, was trained until complete overfitting with sequences from a sublanguage (Fig. 5.2).

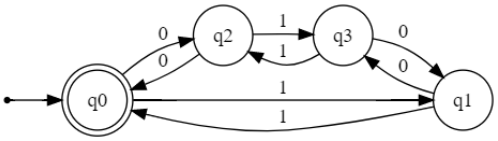


Figure 5.1: DFA recognizing Tomita's 5th grammar

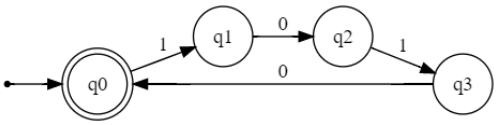


Figure 5.2: DFA recognizing a sublanguage of Tomita's 5th grammar

The architecture of the networks is depicted in Fig. 5.3[1]. For each layer, its type, name (for clarity), and input/output shapes are shown. In all cases, the first component of the shape vector is the batch size and the last component is the number of features. For 3-dimensional shapes, the middle element is the length

[1]Network sketches have been generated using Keras utilities `https://keras.io/api/utils/model_plotting_utils/`.

of the sequence. '?' means that the parameter is not statically fixed but dynamically instantiated at the training phase. The initial layer is a 2-dimensional dense embedding of the input. This layer is followed by a sequence-to-sequence subnetwork composed of a 64-dimensional LSTM chained to a 30-dimensional dense layer with a *ReLU* activation function. The network ends with a classification subnetwork composed of a 62-dimensional LSTM connected to a 2-dimensional dense layer with a *softmax* activation function. This architecture has a total of 42,296 coefficients.

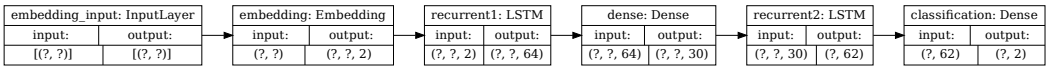


Figure 5.3: Sketch of the architecture used for Tomita's 5th grammar and its variant.

Each network has been trained in a single phase with specific parameters summarized in Table 5.5. This is the reason why batch size and sequence length have not been fixed in Fig. 5.3 and therefore appear as '?'. The training process of both networks used sets of randomly generated sequences labelled as belonging or not to the corresponding target language. These sets have been split in two parts: 80% for the development set and 20% for the test set. The development set has been further partitioned into 67% for train and 33% for validation.

Network	**Dataset size**	**Batch size**	**Sequence length**	**Learning rate**
N_1	5K	30	15	0.01
N_2	1M	100	10	0.001

Table 5.5: Training parameters used for Tomita's 5th grammar and its variant.

When checking both networks for inclusion in Tomita's 5th grammar both of them were found to verify the inclusion, passing PAC tests with $\varepsilon = 0.001$ and $\delta = 0.0001$. However, when the verification goal was to check $N_1 \equiv N_2$, the output was different. In such scenario, on-the-fly verification returned a nonempty DFA, showing that the networks are indeed not equivalent. Fig. 5.4 depicts the DFA approximating the language of their disagreement, that is, the symmetric difference $N_1 \oplus N_2$. After further inspection, we found out that N_2 does not recognize the empty word λ.

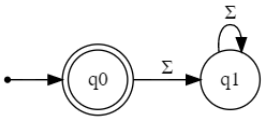


Figure 5.4: DFA approximating $N_1 \oplus N_2$

5.2 Scenario 2

In this scenario, we checked regular properties over RNN trained with sequences of models of two different software systems, namely a cruise controller and an e-commerce application. The former deals with the situation where post-learning model-checking finds the DFA extracted from the RNN to not satisfy the property, but it is not possible to replay the produced counterexample on the RNN. In the latter, we injected canary bad sequences in the training set in order to pinpoint they end up being discovered by on-the-fly black-box checking.

5.2.1 A model of a cruise control software

Here, we analyze an RNN trained with sequences from the model of a cruise controller software [55] depicted in Fig. 5.5. In the figure, only the actions and states modelling the normal operation of the controller are shown. All illegal actions are assumed to go to a non-accepting sink state. The training dataset contained 200,000 randomly generated sequences and labelled as normal and abnormal according to whether they correspond or not to executions of the controller (i.e., they are recognized or not by the DFA in Fig. 5.5). All executions have a length of at most 16 actions. The accuracy of the RNN on a test dataset with 16,000 randomly generated sequences was 99,91%.

The requirement ψ to be checked on the RNN is the following: a *break* action can occur only if action *gas*|*acc* has already happened and no other *break* action has occurred in between. ψ is modelled by the DFA illustrated in Fig. 5.6.

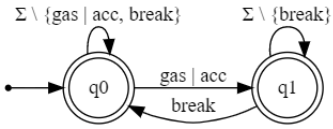


Figure 5.6: Cruise controller: Property ψ

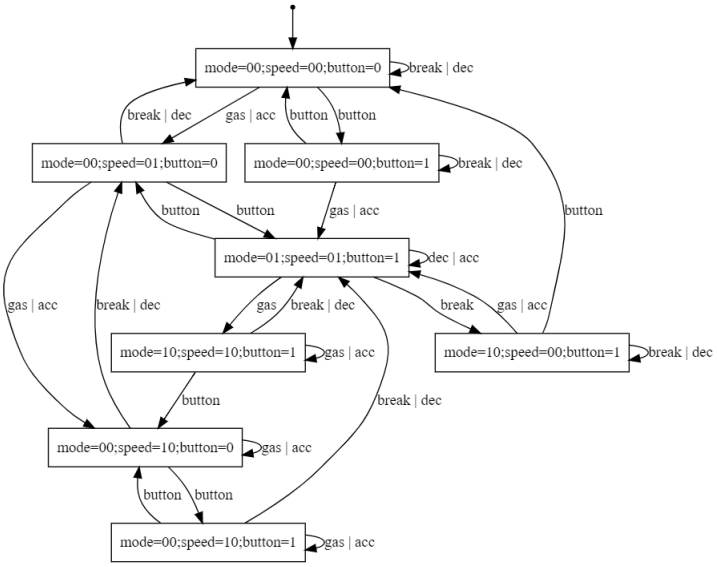


Figure 5.5: Cruise controller: DFA model

In this experiment, we compare both approaches, namely our on-the-fly technique vs post-learning verification.

Every run of on-the-fly verification through learning terminates with perfect **EQ** tests conjecturing that $C \cap \overline{\psi}$ is empty. Table 5.6 shows the metrics obtained in these experiments (running times, **EQ** sample sizes, and $\widetilde{\epsilon}$) for different values of the parameters ε and δ.

Parameters		**Running times (in secs)**			**First positive MQ**	**Mean sample size**
ε	δ	**min**	**max**	**mean**		
0.01	0.01	0.003	0.006	0.004	-	669
0.001	0.01	0.061	0.096	0.075	-	6,685
0.0001	0.01	0.341	0.626	0.497	-	66,847

Table 5.6: Cruise controller: On-the-fly black-box checking.

Table 5.7 shows the metrics for extracting DFA from the RNN. The timeout was set at 200 secs. For the first configuration, four out of five runs terminated before the timeout producing automata that exceeded the maximum number of states. Moreover, three of those were shown to violate the requirement. For the second one, there were three out of five successful extractions with all automata exceeding the maximum number of states, while for two the property did not hold. For the third configuration, all runs hit the timeout. Actually, the RNN

under analysis classified all the counterexamples returned by the model-checker as negative, that is, they do not belong to its language. In other words, there were false positives. In order to look for true violating sequences, we generated 2 million sequences with **EX** for each of the automata H for which the property did not hold. Indeed, none of those sequences was accepted simultaneously by both the RNN under analysis and $H \cap \overline{\psi}$. Therefore, it is not possible to disprove that the RNN is correct with respect to ψ as conjectured bye on-the-fly black-box checking. It goes without saying that post-learning verification required considerable more computational effort as a consequence of its misleading verdicts.

Parameters		**Running times (in secs)**			**Mean sample size**	**Mean** $\widetilde{\epsilon}$
ε	δ	**min**	**max**	**mean**		
0.01	0.01	11.633	200.000	67.662	808	0.07329
0.001	0.01	52.362	200.000	135.446	8,071	0.22684
0.0001	0.01	-	-	-	-	-

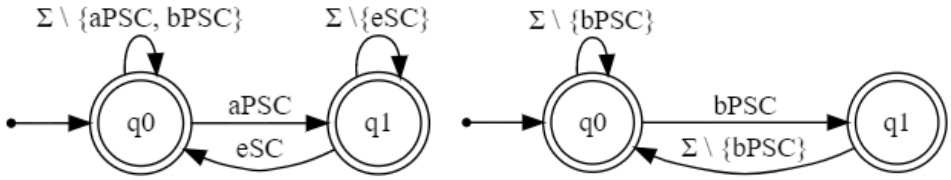
Table 5.7: Cruise controller: Automaton extraction.

The cruise controller case study illustrates an important benefit of our approach vs post-learning verification: every counterexample produced by on-the-fly property checking is a true witness of $\Psi(C)$ being nonempty, while this is certainly false for the latter.

5.2.2 A model of an e-commerce web site

In this case study, the target concept is an RNN trained with the purpose of modelling the behavior of a web application for e-commerce. We used a training dataset of 100,000 randomly generated sequences of length smaller than or equal to 16, using a variant of the model in [32, 56] to tag the sequences as positive or negative. Purposely, we have modified the model so as to add *canary* sequences not satisfying the properties to be checked. The RNN achieved 100% accuracy on a test dataset of 16,000 randomly generated sequences. We overfitted to ensure faulty sequences were classified as positive by the RNN. The goal of this experiment was to verify whether on-the-fly black-box checking could successfully unveil whether the RNN learned these misbehaviors.

We analyzed the regular properties shown in Fig. 5.7, where labels $aPSC$, eSC, and $bPSC$ model the actions (associated with their corresponding buttons) of adding products to the shopping cart, removing all products from the shopping cart, and buying products in the shopping cart, respectively. Requirement ψ_1,



(a) E-commerce system: ψ_1 (b) E-commerce system: ψ_1

Figure 5.7: E-commerce system: Automata of the analyzed requirements

depicted in Fig. 5.7a, states that the e-commerce site must not allow a user to buy products in the shopping cart if the shopping cart does not contain any product. Property ψ_2, depicted in Fig. 5.7b, requires the system to prevent the user to perform consecutive clicks on the buy products button.

Table 5.8 shows the metrics obtained for extracting automata. All runs terminated with an **EQ** with no divergences. Therefore, the extracted automata were (ε, δ)-approximations of the RNN. Although we did not perform post-learning verification, these metrics are helpful to compare the computational performance of both approaches.

Parameters		**Running times (in secs)**			**Mean sample size**
ε	δ	**min**	**max**	**mean**	
0.01	0.01	16.863	62.125	36.071	863
0.001	0.01	6.764	9.307	7.864	8,487
0.0001	0.01	18.586	41.137	30.556	83,482

Table 5.8: E-commerce: PAC DFA extraction from RNN.

For each property ψ_j, $j \in \{1, 2\}$, the concept inside the black-box is $\Psi_j(C)$ is $C \cap \overline{\psi}_j$. As shown in Table 5.9, the on-the-fly method correctly asserted that none of the properties were satisfied. It is worth noticing that all experiments terminated with perfect **EQ**, i.e., $k = 0$. Therefore, the extracted DFA were (ε, δ)-approximations of $\Psi_j(C)$. The average running time to output an automaton of the language of faulty behaviors is bigger than the running time for extracting an automaton of the RNN alone. Nevertheless, the first witness of $\Psi_j(C)$ (i.e., the first witness of non-emptiness) was always found by on-the-fly checking in comparable time.

Fig. 5.8 shows an automaton of $\Psi_1(C)$ built by the on-the-fly algorithm. For instance, it reveals that the RNN classifies as correct a sequence where the user opens a session (label event *os*), consults the list of available products (la-

Ψ	**Parameters**		**Running times (in secs)**			**First positive MQ**	**Mean sample size**
	ε	δ	**min**	**max**	**mean**		
Ψ_1	0.01	0.01	87.196	312.080	174.612	3.878	891
	0.001	0.01	0.774	203.103	102.742	0.744	9,181
	0.0001	0.01	105.705	273.278	190.948	2.627	94,573
Ψ_2	0.01	0.01	0.002	487.709	148.027	80.738	752
	0.001	0.01	62.457	600.000	428.400	36.606	8,765
	0.0001	0.01	71.542	451.934	250.195	41.798	87,641

Table 5.9: E-commerce: On-the-fly verification of RNN.

bel gAP), and then buys products ($bPSC$), but the shopping cart is empty: q_1; os; q_4; gAP; q_3; $bPSC$. Indeed, it provides valuable information about possible causes of the error which are helpful to understand it and correcting it, since it makes apparent that every time gAP occurred in an open session, the property was violated.

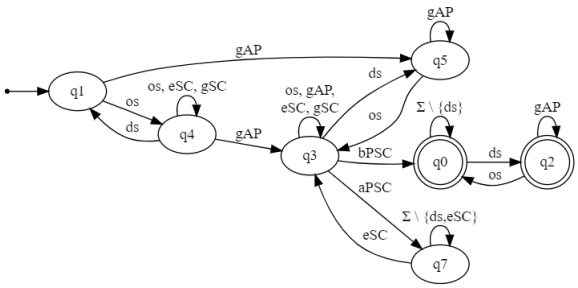


Figure 5.8: E-commerce system: Automaton for $\Psi_1(C)$.

Fig. 5.9 depicts an automaton for $\Psi_2(C)$. A sequence showing that ψ_2 is not satified is: q_1; os; q_5; gAP; q_4; $bPSC$; q_3; $bPSC$. Notice that this automaton shows that ψ_1 is violated as well, since state q_3 is reachable without any occurrence of $aPSC$.

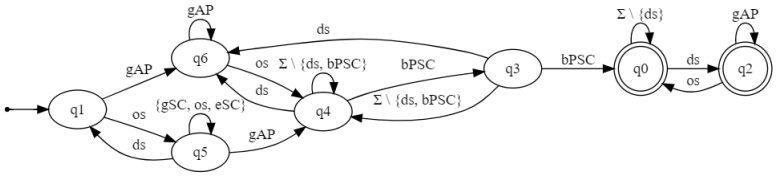


Figure 5.9: E-commerce system: Automaton for $\Psi_2(C)$.

5.3 Scenario 3

Finally, we studied domain-specific datasets, from system security and bioinformatics, where the actual data-generator systems were unknown, and no models of them were available. In one of these case studies the purpose is to analyze the behavior of an RNN trained to identify security anomalies in Hadoop file system (HDFS) logsfrom. The experiment revealed the fact that the RNN could mistakenly classify a log as normal when it is actually abnormal, even if the RNN incurred in no false positives on the test dataset during the training phase. The DFA returned by Bounded-L* served to gain insight on the error. In the last case study, we studied an RNN that classifies promoter DNA sequences as having or not a TATA-box subsequence. Here, post-learning verification was unfeasible because Bounded-L* did not terminate in reasonable time when asked to extract a DFA from the RNN. Nevertheless, it successfully checked the desired requirement using on-the-fly black-box checking through learning.

5.3.1 Hadoop file system logs

This experiment concerns the analysis of an RNN trained to find anomalies in logs of an application software based on Hadoop Distributed File System (HDFS). Data used in this case study come from [18]. Logs are sequences of natural numbers ranging from 0 to 28 which correspond to different kinds of logged messages. That is, the set of symbols is $\Sigma = \{0, \ldots, 29\}$. The training dataset consists of 4856 normal logs of different lengths. We built an auto-regressive network that predicts the probability distribution of symbols at each position in the sequence. Symbols are one-hot encoded. The LSTM layer outputs a 128-dimensional vector which is passed to a 29-dimensional dense layer that outputs the probability distribution of the next symbol. That is, for every position $t \in [0, T-1]$, where T is the length of the sequence, the network outputs a vector $v_t \in [0,1]^{29}$, whose i-th position holds the predicted probability $v_t(i) = \mathbb{P}[\sigma_t = i \mid \sigma_0 \ldots \sigma_{t-1}]$ of number i to be the t-th symbol in the sequence [57]. Fig. 5.10 shows a sketch of the architecture. This network has 84,637 parameters. The activation function of the last layer is a *softmax* and the loss function is the corresponding categorical cross-entropy. For the sake of readability, we fixed the sequence length in Fig. 5.10. However, in the actual architecture this parameter is not statically defined.

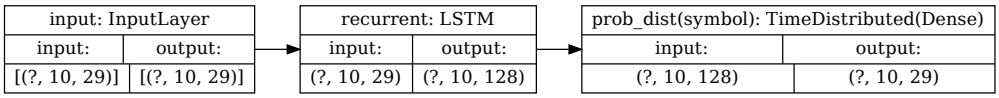


Figure 5.10: Sketch of the architecture of the language model of HDFS logs.

For each log in the training set we obtained all complete subsequences of length $T = 10$ by sliding a window of size 10 from start to end. Overall, there were a total of 56,283 of such subsequences which were split in 80% (36020 samples) for training and 20% (9006 samples) for validation. A single training phase of 5 epochs was performed using a learning rate of 10^{-3} and a batch size of 30.

In order to build a classifier, the RNN is used to predict the probability of a log. Then, a log is considered to be normal if its predicted probability is beyond a threshold of 2×10^{-7}. Otherwise, it is tagged as anomalous. The performance of the classifier was tested on a perfectly balanced subset of 33,600 samples taken from the test dataset of [18]. No false positives were produced by the classifier which incurred in an overall error of 2.65%.

During an exploratory analysis of the training dataset, we made the following observations. First, there were a subset of numbers, concretely $\{6, 7, 9, 11 - 14, 18, 19, 23, 26 - 28\}$, that were not present in the normal logs used for training. Let us call this set A for *anomalous* message types. Second, many logs have a subsequence containing numbers 4 and 21, such that their accumulated count was at most 5, that is, $\#4 + \#21 \leq 5$. We analyzed the classifier with the purpose of investigating whether the RNN actually learned these patterns as characteristic of normal logs.

Based on those observations, we defined the following properties. The first statement, ψ_1, claims that the classifier always labels as anomalous any log containing a number in A. The second one, ψ_2, says that every log satisfying $\#4+\#21 \leq 5$ is classified as normal. As in the case study of the e-commerce, for each property ψ_j, $j \in \{1, 2\}$, the concept inside the black-box is $\Psi_j(C)$ is $C \cap \overline{\psi}_j$, where C is the classifier. It is worth mentioning that C is indeed the composition of an RNN with the decision function that labels logs according to the probability output by the RNN.

Table 5.10 shows the results obtained with on-the-fly checking through learning. As in previous experiments, 5 runs of the algorithm were executed for each configuration. All runs terminated with perfect **EQ** tests. Hence, all output hypothesis were (ε, δ)-approximations of $\Psi_j(C)$.

Prop	Parameters		Running times (in secs)			First positive MQ	Mean sample size
	ε	δ	min	max	mean		
Ψ_1	0.01	0.01	209.409	1,121.360	555.454	5.623	932
	0.001	0.001	221.397	812.764	455.660	1.321	12,037
Ψ_2	0.01	0.01	35.131	39.762	37.226	-	600
	0.001	0.001	252.202	257.312	254.479	-	8,295

Table 5.10: Hadoop file system logs: On-the-fly verification.

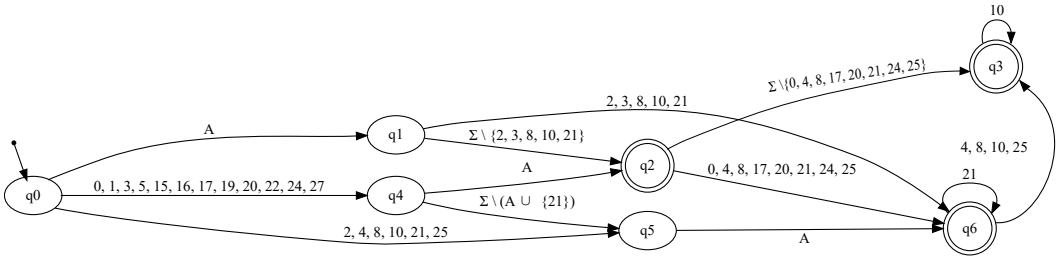


Figure 5.11: Hadoop file system logs: Automaton for $\Psi_1(C)$ obtained with $\varepsilon = 0.01$ and $\delta = 0.01$.

On one hand, property ψ_2 is satisfied by C with PAC guarantees. On the contrary, all runs of the algorithm for $\Psi_2(C)$ returned a non-empty automaton and a set of the logs that violate ψ_2. Therefore, we conclude that C actually classifies as normal some logs containing numbers in A. Fig. 5.11 depicts the automaton obtained for $\Psi_1(C)$. It helps to understand the errors of C. For example, it reveals that C labels as normal a log that contains an occurrence of a number in A in its prefix of length 2. This behavior is captured by paths $q_0\ q_1\ q_2$, $q_0\ q_1\ q_6$, and $q_0\ q_4\ q_2$. Indeed, this outcome highlights the importance of verification, since it revealed a clear mismatch with the results observed on the test dataset where C all logs containing numbers in A were labelled as anomalous because C reported no false positives whatsoever.

5.3.2 TATA-boxes in DNA promoter sequences

DNA promoter sequences are in charge of controlling gene activation or repression. A TATA-box is a promoter subsequence with the special role of indicating other molecules the starting place of the transcription. A TATA-box is a subsequence having a length of 6 base pairs (bp). It is located upstream close to the gene

transcription start site (TSS) from positions –30bp to –25bp[2]. It is characterized by the fact that the accumulated number of occurrences of A's and T's is larger than that of C's and G's.

Recently, RNN-based techniques for recognizing TATA-box promoter regions in DNA sequences have been proposed [19]. Therefore, it is of interest to check whether an RNN classifies as positive sequences having a TATA-box and as negative those not having it. In terms of a formal language, the property can be characterized as the set of sequences $u \in \{A, T, C, G\}^*$ with a subsequence v of length 6 from –30bp to –25bp such that $\#A + \#T > \#C + \#G$, where $\#\sigma$ is the number of occurrences of $\sigma \in \{A, T, C, G\}$ in v.

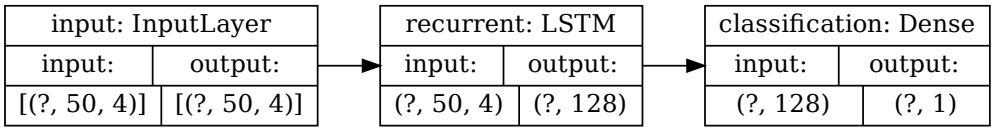


Figure 5.12: Sketch of the architecture of the TATA-Box classification network

For that purpose, we trained an RNN until achieving 100% accuracy on the training data consisting of 16455 aligned TATA and non-TATA promoter sequences of human DNA extracted from the online database EPDnew[3]. All sequences have a total length of 50 and end at the TSS. Overall, there were 2067 sequences with TATA boxes and 14388 sequences without. The LSTM layer had a 128-dimensional output. In this case, training was performed on a single phase with a learning rate of 10^{-3} and a batch size of 64. No validation nor test sets were used. Fig. 5.12 shows a graphical sketch of the model. The input dimension is given by the batch size, the length of the sequence, and the number of symbols.

Parameters		**Running times (in secs)**			**Mean sample size**
ε	δ	**min**	**max**	**mean**	
0.01	0.01	5.098	5.259	5.168	600
0.001	0.001	65.366	66.479	65.812	8,295
0.0001	0.0001	865.014	870.663	867.830	105,967

Table 5.11: TATA-box: On-the-fly verification of RNN.

Table 5.11 shows the results obtained only with the on-the-fly approach. Indeed, every attempt to learn a DFA of the RNN C caused Bounded-L* to terminate

[2]TSS is located at +1bp

[3]`https://epd.epfl.ch//index.php`

with a timeout. Therefore, this case study illustrates the case where post-learning verification is not feasible while on-the-fly checking is. It turns out that all executions concluded that the empty language was an (ε, δ)-approximation of the black-box $\Psi(C)$. Thus, C verifies the requirement with PAC guarantees. It is worth noticing that in the last reported experiment, with ε and δ equal to 0.0001, the sample used for checking equivalence was about an order of magnitude bigger than the dataset used for training.

6 Language models

Until this point, every presented concept is an acceptor, in other words $\mathcal{C}$ is restricted to functions where $\mathcal{X} = \Sigma^*$ and $\mathcal{O} = \{0, 1\}$, in this chapter we will work on language models, that is, $\mathcal{O}$ represents distributions over a given alphabet.

To formalize it, let Σ be a finite *alphabet* and $\Sigma_\$ \triangleq \Sigma \cup \{\$\}$, where \$ is a special *terminal* symbol not in Σ. Also let $\Delta(\Sigma_\$) \triangleq \{\eta : \Sigma_\$ \to \mathbb{R}_+ \mid \sum_{\sigma \in \Sigma_\$} \eta(\sigma) = 1\}$ be the probability simplex over $\Sigma_\$$. A *language model* is a total function $\mathcal{M} : \Sigma^* \to \Delta(\Sigma_\$)$ where $\mathcal{M}(u)$ is interpreted as the *next-symbol* probability distribution, that is,

$$P_{\mathcal{M}}(\sigma|u) \triangleq \mathcal{M}(u)(\sigma) \tag{6.1}$$

$\mathcal{M}$ induces a probability distribution $f_{\mathcal{M}}$ over Σ^*, formally, given a string $u = u_1 \dots u_n \in \Sigma^*$:

$$f_{\mathcal{M}}(u) \triangleq \prod_{i=0}^{n} \mathcal{M}(u[i])(u_{i+1}) \tag{6.2}$$

Where $u[i] \triangleq u_1 \dots u_i$ with $u[0] = \lambda$, and $u_{n+1} \triangleq \$$.

Given a language model $\mathcal{M}$, we can define an equivalence relation $\equiv \subseteq \Sigma^* \times \Sigma^*$ defined as:

$$u \equiv u' \overset{\triangle}{\iff} \forall w \in \Sigma^*.\ \mathcal{M}(uw) = \mathcal{M}(u'w) \tag{6.3}$$

We write $[\![\Sigma^*]\!]$ for the set of equivalence classes defined by $\equiv$ and $[\![u]\!]$ for the class of $u \in \Sigma^*$. Indeed, $\equiv$ is a *right congruence* with respect to concatenation of a symbol:

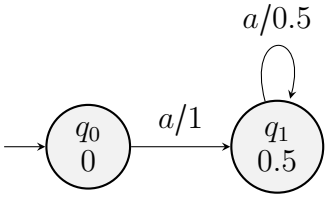
Proposition 6.1. $\forall u, u' \in \Sigma^*.\ u \equiv u' \implies \forall \sigma \in \Sigma.\ u\sigma \equiv u'\sigma$.

Proof. *Let $u \equiv u'$ and $\sigma \in \Sigma$. Then for any $w \in \Sigma^*$ we have $\mathcal{M}((u\sigma)w) = \mathcal{M}(u(\sigma w)) \equiv \mathcal{M}(u'(\sigma w)) = \mathcal{M}((u'\sigma)w)$.*

We define two language models $\mathcal{M}_1$ and $\mathcal{M}_2$ to be equivalent if $\mathcal{M}_1(u) = \mathcal{M}_2(u)$ for all $u \in \Sigma^*$. That is, $\mathcal{M}_1$ and $\mathcal{M}_2$ induce the same equivalence relation over Σ^*, formally $\mathcal{M}_1 \equiv \mathcal{M}_2$. We define $\mathcal{M}$ to be *regular* if $[\![\Sigma^*]\!]$ is finite.

6.1 PDFA

A *probabilistic deterministic finite automaton* (PDFA) [48, 58] is a concrete realization of a language model. Let Σ be a finite *alphabet* and $\Sigma_\$$ to be the set $\Sigma \cup \{\$\}$, where \$ is a special *terminal* symbol not in Σ. A PDFA A over Σ is a tuple $(Q, q_{\text{in}}, \pi, \tau)$, where Q is a finite set of states, $q_{\text{in}} \in Q$ is an initial state, $\pi : Q \to \Delta(\Sigma_\$)$ maps each state to a probability distribution over $\Sigma_\$$, and $\tau : Q \times \Sigma \to Q$ is the transition function. Both π and τ are total functions. Figure 6.1(*a*-*b*) depicts an example.



	π		τ
Q	\$	a	a
q_0	0	1	q_1
q_1	0.5	0.5	q_1

Figure 6.1: PDFA over $\Sigma = \{a\}$ with $q_{\text{in}} = q_0$.

The probability of $u \in \Sigma^*$ from state $q \in Q$, denoted $P(u|q)$, is defined as:

$$P(\lambda|q) \triangleq \pi(q)(\$) \qquad \lambda \text{ is the empty string,} \tag{6.4}$$

$$P(\sigma u|q) \triangleq \pi(q)(\sigma) \cdot P(u|\tau(q,\sigma)) \qquad \sigma \in \Sigma, u \in \Sigma^* \tag{6.5}$$

Analogously to 6.2, a PDFA A computes a function f_A from Σ^* to $[0, 1]$. For any string $u \in \Sigma^*$:

$$f_A(u) \triangleq P(u|q_{\text{in}}) \tag{6.6}$$

For instance, the PDFA in Figure 6.1 maps the empty string λ to 0 and every string a^n, $n \geq 1$, to 0.5^n.

As for DFA, we define $\tau^*(q, u)$ to be the natural extension of τ to strings, that is, the state reached by A when going through u starting at state q:

$$\tau^*(q, \lambda) \triangleq q \tag{6.7}$$

$$\tau^*(q, \sigma u) \triangleq \tau^*(\tau(q, \sigma), u) \tag{6.8}$$

Similarly, we define $\pi^*(q, u)$ to be the probability distribution of the state reached by A when going through u from state q:

$$\pi^*(q, u) \triangleq \pi(\tau^*(q, u)) \tag{6.9}$$

We denote $\tau^*(u)$ and $\pi^*(u)$ the state reached when going through u from the initial state q_{in} and its associated probability distribution, respectively. Then, A defines the language model such that:

$$A(u) \triangleq \pi^*(u) \tag{6.10}$$

6.2 Congruences

Similarly to DFA 2.7, the congruence $\equiv$ can be extended to PDFA as follows:

$$\forall q, q' \in Q.\ q \equiv q' \overset{\triangle}{\iff} \forall w \in \Sigma^*.\ \pi^*(q, w) = \pi^*(q', w) \tag{6.11}$$

Clearly, $\equiv$ is an equivalence relation. Indeed, the following proposition holds:

Proposition 6.2. $\forall u, u' \in \Sigma^*.\ u \equiv u' \iff \tau^*(u) \equiv \tau^*(u')$.

Proof. *Let $u, u' \in \Sigma^*$:*

$$\begin{aligned}
u \equiv u' &\iff \forall w \in \Sigma^*.\ A(uw) = A(u'w) && \textit{Def. 6.3} \\
&\iff \forall w \in \Sigma^*.\ \pi^*(uw) = \pi^*(u'w) && A(\cdot) \triangleq \pi^*(\cdot)\ \textit{Def. 6.10} \\
&\iff \forall w \in \Sigma^*.\ \pi(\tau^*(uw)) = \pi(\tau^*(u'w)) && \textit{Def. of } \pi^* \\
&\iff \forall w \in \Sigma^*.\ \pi(\tau^*(\tau^*(u), w)) = \pi(\tau^*(\tau^*(u'), w)) && \textit{Def. of } \tau^* \\
&\iff \forall w \in \Sigma^*.\ \pi^*(\tau^*(u), w) = \pi^*(\tau^*(u'), w) && \textit{Def. of } \pi^* \\
&\iff \tau^*(u) \equiv \tau^*(u') && \textit{Def. 6.11}
\end{aligned}$$

Equivalent states have the same distribution:

Proposition 6.3. $\forall q, q' \in Q.\ q \equiv q' \implies \pi(q) = \pi(q')$.

Proof.

$$\begin{aligned}
\pi(q) &= \pi^*(q, \lambda) && \textit{by Def. 6.9 and Def. 6.7} \\
&= \pi^*(q', \lambda) && \textit{by hypothesis } q \equiv q' \textit{ and Def. 6.11} \\
&= \pi(q') && \textit{by Def. 6.9 and Def. 6.7}
\end{aligned}$$

Moreover, the following result shows $\equiv$ is a right congruence with respect to the transition function:

Proposition 6.4. $\forall q, q' \in Q, \sigma \in \Sigma.\ q \equiv q' \implies \tau(q, \sigma) \equiv \tau(q', \sigma)$.

Proof. *Let $q \equiv q'$, $\sigma \in \Sigma$, $u \in \Sigma^*$.*

$$\begin{aligned}
\pi^*(\tau(q,\sigma),w) &= \pi(\tau^*(\tau(q,\sigma),w)) && \textit{by Def. 6.9}\\
&= \pi(\tau^*(q,\sigma w)) && \textit{by Def. 6.8}\\
&= \pi^*(q,\sigma w) && \textit{by Def. 6.9}\\
&= \pi^*(q',\sigma w) && \textit{by hypothesis } q \equiv q' \textit{ and Def. 6.11}\\
&= \pi^*(\tau(q',\sigma),w) && \textit{by Def. 6.9 and Def. 6.8}
\end{aligned}$$

Hence, $\tau(q,\sigma) \equiv \tau(q',\sigma)$.

6.3 Quotient PDFA

Given a PDFA A, a state q is *reachable* if $q = \tau^*(u)$ for some string $u \in \Sigma^*$. Any such u is called an *access* string of q. We denote $reach(Q)$ the set of reachable states.

Let $\dot{Q} \triangleq [\![reach(Q)]\!]$ be the set of reachable equivalence classes defined by $\equiv$ and $\dot{q} \triangleq [\![q]\!] \in \dot{Q}$ be the class of $q \in reach(Q)$. It follows from 6.3 and 6.4 that $\equiv$ induces a *quotient* PDFA $\dot{A}$:

$$\dot{A} \triangleq (\dot{Q}, \dot{q}_{\text{in}}, \dot{\pi}, \dot{\tau}) \tag{6.12}$$

Where for every $\dot{q} \in \dot{Q}$:

- the probability distribution is $\dot{\pi}(\dot{q}) \triangleq \pi(q)$, and
- the transition function is $\dot{\tau}(\dot{q},\sigma) \triangleq [\![\tau(q,\sigma)]\!]$ for any $\sigma \in \Sigma$.

Corollary 6.1. $|[\![\Sigma^*]\!]| = |\dot{Q}|$.

Proof. *Let $\phi : [\![\Sigma^*]\!] \to \dot{Q}$ be $\phi([\![u]\!]) \triangleq [\![\tau^*(u)]\!]$. From Prop. 6.2, ϕ is bijective.*

Proposition 6.5. *For every PDFA A, $\dot{A}$ computes the same function as A.*

Proof. *We prove by induction on the length of $u \in \Sigma^*$ that for all $q, q' \in Q$, if $q \equiv q'$ then $P(u|q) = P(u|q')$ for every $u \in \Sigma^*$.*

Base case $u = \lambda$.

$$\begin{aligned}
P(\lambda|q) &= \pi(q)(\$) && \textit{by Def. 6.4}\\
&= \pi(q')(\$) && \textit{by Prop. 6.3}\\
&= P(\lambda|q') && \textit{by Def. 6.4}
\end{aligned}$$

Inductive step $u = \sigma u'$.

$$\begin{aligned} P(\sigma u'|q) &= \pi(q)(\sigma) \cdot P(u|\tau(q,\sigma)) && \textit{by Def. 6.5} \\ &= \pi(q')(\sigma) \cdot P(u|\tau(q,\sigma)) && \textit{by Prop. 6.3} \\ &= \pi(q')(\sigma) \cdot P(u|\tau(q',\sigma)) && \textit{by Prop. 6.4 and ind. hyp.} \\ &= P(\sigma u'|q') && \textit{by Def. 6.5} \end{aligned}$$

Hence, $P(u|\dot{q}_{\text{in}}) = P(u|q_{\text{in}})$.

Therefore, $\equiv$ can be extended to PDFA, where for every PDFA A and B:

$$A \equiv B \iff q_{\text{in}}^A \equiv q_{\text{in}}^B \tag{6.13}$$

Corollary 6.2. *$A \equiv B$ implies $f_A = f_B$.*

Proof. *Let $u \in \Sigma^*$.*

$$\begin{aligned} f_A(u) &= P(u|q_{\text{in}}^A) && \textit{by Def. 6.6} \\ &= P(u|q_{\text{in}}^B) && \textit{by Prop. 6.5 and } q_{\text{in}}^A \equiv q_{\text{in}}^B \\ &= f_B(u) && \textit{by Def. 6.6} \end{aligned}$$

The converse of Cor. 6.2 does not hold in general. Consider for instance PDFA A and B in Figure 6.2. We have that $f_B((ab)^{n+1}) = (0.1 \cdot 0.2)(0.2 \cdot 0.1)^n \cdot 0.3 = (0.1 \cdot 0.2)^{n+1} \cdot 0.3 = f_A((ab)^{n+1})$, for $n \geq 0$. For any other string $u \neq (ab)^{n+1}$, $f_A(u) = f_B(u) = 0$. Hence, A and B compute the same function, that is, $f_A = f_B$. However, $A \not\equiv B$.

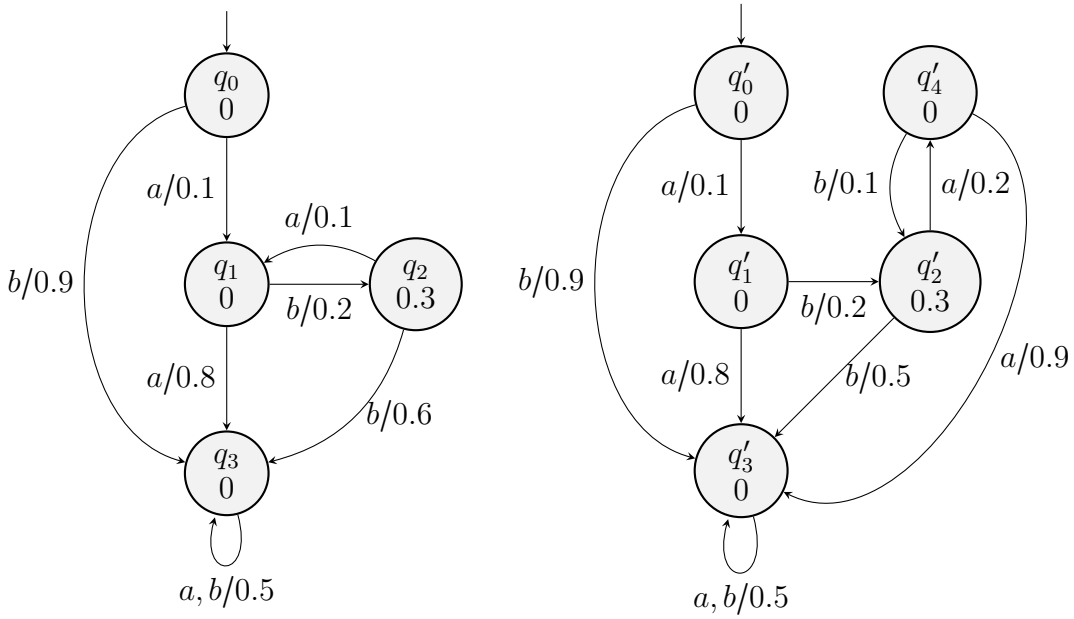


Figure 6.2: PDFA A (left) and B (right).

6.4 Minimality

A PDFA is *minimal* if any other PDFA that computes the same function has no less states. The simple PDFA in Figure 6.1 is minimal since clearly the function cannot be computed by a PDFA with a single state.

Based on $\equiv$ we define a notion of minimality as follows. A PDFA A is said to be $\equiv$-*minimal* if for every $q, q' \in Q$, $q \not\equiv q'$. By definition, for every PDFA A, $\dot{A}$ is $\equiv$-minimal. In other words, a PDFA is $\equiv$-minimal if it is equal to its quotient up to isomorphism.

Proposition 6.6. *Every minimal PDFA A is also $\equiv$-minimal.*

Proof. *Suppose that A is not $\equiv$-minimal. Then, there are states in A which are equivalent. Thus, $\dot{A}$, which computes the same function as A, has strictly less states than A, which contradicts the hypothesis.*

The converse does not hold in general. A and B in Figure 6.2 are both $\equiv$-minimal because all states have different probability distributions. But B is certainly not minimal because it has more states than A.

6.5 Equivalences between distributions

Let $\mathcal{S} \subseteq \Delta(\Sigma_\$) \times \Delta(\Sigma_\$)$ be an equivalence relation. We write $\eta =_\mathcal{S} \eta'$ instead of $\mathcal{S}(\eta, \eta')$, and $[\![\Delta(\Sigma_\$)]\!]_\mathcal{S}$, respectively $[\![\eta]\!]_\mathcal{S}$, to denote the quotient of $\Delta(\Sigma_\$)$ induced by $\mathcal{S}$, and the class of η. Given $\mathcal{M}$, $\mathcal{S}$ induces the equivalence relation $\equiv_\mathcal{S} \subseteq \Sigma^* \times \Sigma^*$ defined as:

$$u \equiv_\mathcal{S} u' \overset{\triangle}{\iff} \forall w \in \Sigma^*.\ \mathcal{M}(uw) =_\mathcal{S} \mathcal{M}(u'w) \tag{6.14}$$

We write $[\![\Sigma^*]\!]_\mathcal{S}$ for the set of equivalence classes defined by $\equiv_\mathcal{S}$ and $[\![u]\!]_\mathcal{S}$ for the class of $u \in \Sigma^*$. Indeed, $\equiv_\mathcal{S}$ is a *right congruence* with respect to concatenation of a symbol:

Proposition 6.7. $\forall u, u' \in \Sigma^*.\ u \equiv_\mathcal{S} u' \implies \forall \sigma \in \Sigma.\ u\sigma \equiv_\mathcal{S} u'\sigma.$

Proof. *Let $u \equiv_\mathcal{S} u'$ and $\sigma \in \Sigma$. Then for any $w \in \Sigma^*$ we have $\mathcal{M}((u\sigma)w) = \mathcal{M}(u(\sigma w)) \equiv_\mathcal{S} \mathcal{M}(u'(\sigma w)) = \mathcal{M}((u'\sigma)w)$.*

Naturally, we can say that two language models $\mathcal{M}_1$ and $\mathcal{M}_2$ are equivalent modulo $\mathcal{S}$:

$$\mathcal{M}_1 \equiv_\mathcal{S} \mathcal{M}_2 \overset{\triangle}{\iff} \forall u \in \Sigma^*.\ \mathcal{M}_1(u) =_\mathcal{S} \mathcal{M}_2(u) \tag{6.15}$$

This implies that $\mathcal{M}_1$ and $\mathcal{M}_2$ induce the same equivalence relation over Σ^*. We say that:

$$\mathcal{M} \text{ is } \mathcal{S}\text{–}\textit{regular} \overset{\triangle}{\iff} [\![\Sigma^*]\!]_\mathcal{S} \text{ is finite} \tag{6.16}$$

Several equivalence relations $\mathcal{S}$ are of interest. One such equivalence is equality modulo *quantization* [36], denoted $=_\kappa$, where $\kappa \in \mathbb{N}$, $\kappa \geq 1$, is the quantization parameter. This equivalence is motivated by the need to cope with small discrepancies between distributions. For $n \in \mathbb{N}$, $0 \leq n < \kappa - 1$, we define the quantization *interval* I_κ^n to be the left-closed right-open interval $[n\kappa^{-1}, (n+1)\kappa^{-1})$, and for $n = \kappa - 1$, to be the closed interval $[n\kappa^{-1}, 1]$. For $x \in \mathbb{R}$, $[\![x]\!]_\kappa$ is the interval I_κ^n such that $x \in I_\kappa^n$. For $x, y \in \mathbb{R}$, $x =_\kappa y$ if $[\![x]\!]_\kappa = [\![y]\!]_\kappa$. For $\eta, \eta' \in \Delta(\Sigma_\$)$, $\eta =_\kappa \eta'$ if $[\![\eta(\sigma)]\!]_\kappa = [\![\eta'(\sigma)]\!]_\kappa$ for all $\sigma \in \Sigma_\$$.

Other equivalences are related to the purpose a language model is used for. For instance, for anomaly detection, the classification approach proposed by [18] relies on comparing each symbol in the sequence with the top-r most likely symbols predicted by the model. Let $\mathit{rank}(\eta) : \Sigma_\$ \to \mathbb{N}$ be the ranking of symbols $\sigma \in$

$\Sigma_\$$ induced by their probability $\eta(\sigma)$: the symbol with the greatest probability has rank 1, the second best has rank 2, and so on, where symbols with equal probability have the same rank. We denote $rank_r$ the restriction to the first r ranked symbols. The top-r ranked symbols are given by the function: $top_r(\eta) \triangleq \{\sigma \in \Sigma_\$ \mid rank(\eta)(\sigma) \leq r\}$. These functions induce the following equivalences between distributions: $\eta =_{rank_r} \eta'$ if $rank_r(\eta) = rank_r(\eta')$, and $\eta =_{top_r} \eta'$ if $top_r(\eta) = top_r(\eta')$. A string $u = u_1 \dots u_n \in \Sigma^*$ is classified as anomalous by $\mathcal{M}$ if there exists $i \in [1 \dots n]$ such that $u_i \notin top_r(\mathcal{M}(u[i-1]))$, where for $i \geq 1$, $u[i] \triangleq u_1 \dots u_i$ and $u[0] \triangleq \lambda$, otherwise it is classified as normal. Therefore, two language models which are $\equiv_{top_r}$-equivalent will classify sequences the same way. Moreover, $\equiv_{rank}$-equivalence implies $\equiv_{top_r}$-equivalence for every cutoff threshold r.

	η_0	η_1	η_2	η_3	η_4
0	7/16	6/16	6/16	4/16	2/16
1	6/16	7/16	3/16	2/16	4/16
\$	3/16	3/16	7/16	10/16	10/16

Table 6.1: Probability distributions

To illustrate the above equivalences, let us consider the distributions in Tab. 6.1. Quantization with $\kappa = 2$ results in two classes corresponding to $\eta_0 =_\kappa \eta_1 =_\kappa \eta_2$ and $\eta_3 =_\kappa \eta_4$. For *rank* there are four classes since η_0, η_1, η_2 and η_4 have all different rankings, and $\eta_2 =_{rank} \eta_3$. In the case of top_1 we obtain three classes: for η_0 and η_1, the top ranked symbol is 0 and 1, resp., while for η_2, η_3 and η_4 it is \$.

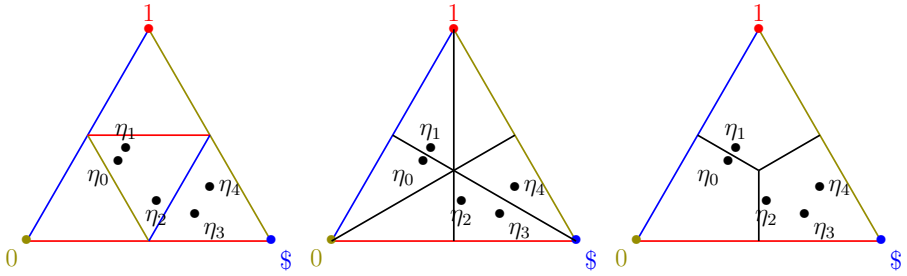


Figure 6.3: From left to right: $=_\kappa$, *rank* and top_1 equivalence classes, respectively.

In Fig. 6.3 the triangle represents the simplex of distributions $\Delta(\Sigma_\$)$ and the partitions the respective equivalence classes in the example of Tab. 6.1. For a distribution (a point in the triangle) the probability of each symbol is given by the distance to the side opposite to the vertex representing that symbol.

6.6 Equivalences in the case of PDFA

Given a PDFA A, it is possible to get a congruence over the set of states for any equivalence relation on distributions. Let $\mathcal{S}$ be an equivalence relation.

Analogous to 6.11, we define the relation $\equiv_{\mathcal{S}}$ as:

$$\forall q, q' \in Q.\ q \equiv_{\mathcal{S}} q' \iff \forall u \in \Sigma^*.\ \pi^*(q, u) =_{\mathcal{S}} \pi^*(q', u) \tag{6.17}$$

This implies the following relationship between states and strings:

Proposition 6.8. $\forall u, u' \in \Sigma^*.\ u \equiv_{\mathcal{S}} u' \iff \tau^*(u) \equiv_{\mathcal{S}} \tau^*(u')$.

Proof. *Analogous to Prop. 6.2*

Equivalent states have equivalent distributions.

Proposition 6.9. $\forall q, q' \in Q.\ q \equiv_{\mathcal{S}} q' \implies \pi(q) =_{\mathcal{S}} \pi(q')$

Proof. *Analogous to Prop. 6.3.*

Moreover, $\equiv_{\mathcal{S}}$ is a right congruence over states with respect to the transition function.

Proposition 6.10. $\forall q, q' \in Q.\ q \equiv_{\mathcal{S}} q' \implies \forall \sigma \in \Sigma.\ \tau(q, \sigma) \equiv_{\mathcal{S}} \tau(q', \sigma)$.

Proof. *Analogous to Prop. 6.4.*

Proposition 6.11. *Let A_1 and A_2 be PDFA, $A_1 \equiv_{\mathcal{S}} A_2 \iff q^1_{\text{in}} \equiv_{\mathcal{S}} q^2_{\text{in}}$*

Proof. *Since A_1 and A_2 are language models, by Def. 6.15, $A_1 \equiv_{\mathcal{S}} A_2 \iff \pi^*_1(u) =_{\mathcal{S}} \pi^*_2(u)$ for all $u \in \Sigma^*$. By Def. 6.17, it means their initial states are equivalent: $q^1_{\text{in}} \equiv_{\mathcal{S}} q^2_{\text{in}}$.*

6.6.1 PDFA quotient modulo $\mathcal{S}$

We define $\dot{Q} \triangleq [\![reach(Q)]\!]_{\mathcal{S}}$ to be the set of equivalence classes of reachable states defined by $\equiv_{\mathcal{S}}$. Differently from 6.12, $\mathcal{S}$ does not define a unique quotient PDFA. It follows that $\equiv_{\mathcal{S}}$ induces a set of *quotient* PDFA $A/_{\equiv_{\mathcal{S}}}$:

$$A/_{\equiv_{\mathcal{S}}} \triangleq \{A' \mid A \equiv_{\mathcal{S}} A' \wedge |\dot{Q}| = |Q'|\} \tag{6.18}$$

$A/_{\equiv_{\mathcal{S}}}$ is a set due to the fact that for some $\dot{q} \in \dot{Q}$, $\dot{\pi}(\dot{q})$ may not be uniquely defined.

From Prop. 6.2, it follows that each $\dot{q} \in \dot{Q}$ can be represented by an access string u of any state in the class, i.e., $\dot{q} = [\![\tau^*(u)]\!]$. We denote $\alpha(\dot{q})$ the designated access string of $\dot{q}$. W.l.o.g., $\alpha(\dot{q}_{\text{in}}) \triangleq \lambda$.

This allows us to define the *quotient* PDFA $\dot{A}_\alpha \in A/_{\equiv_{\mathcal{S}}}$:

$$\dot{A}_\alpha \triangleq (\dot{Q}, \dot{q}_{\text{in}}, \dot{\pi}, \dot{\tau}) \tag{6.19}$$

Where for all $\dot{q} \in \dot{Q}$,

- $\dot{\pi}(\dot{q}) \triangleq \pi^*(\alpha(\dot{q}))$, and
- $\dot{\tau}(\dot{q}, \sigma) \triangleq [\![\tau^*(\alpha(\dot{q})\sigma)]\!]$ for all $\sigma \in \Sigma$.

It is worth noticing that all choices of α lead to isomorphic $\equiv_{\mathcal{S}}$-equivalent PDFA. Therefore, we omit α and simply write $\dot{A}$.

Proposition 6.12. *For all PDFA A, $\dot{A}$ is the smallest PDFA which is $\equiv_{\mathcal{S}}$-equivalent to A.*

Proof. *Let $B = (Q_B, q_{\text{in}}^B, \pi_B, \tau_B)$ be another PDFA $\equiv_{\mathcal{S}}$-equivalent to A. The composition of $\alpha : \dot{Q} \to \Sigma^*$ and $\tau_B^* : \Sigma^* \to Q_B$ defines a map from $\dot{Q}$ to Q_B. We show it is one-to-one: if $\dot{q}, \dot{q}' \in \dot{Q}$ with $\tau_B^*(\alpha(\dot{q})) = \tau_B^*(\alpha(\dot{q}'))$, then for any $w \in \Sigma^*$, we have $\pi^*(\alpha(\dot{q})w) =_{\mathcal{S}} \pi_B^*(\alpha(\dot{q})w) = \pi_B^*(\alpha(\dot{q}')w) =_{\mathcal{S}} \pi^*(\alpha(\dot{q}')w)$ and therefore $\dot{q} \equiv_{\mathcal{S}} \dot{q}'$. Hence $|\dot{Q}| \leq |Q_B|$.*

We say that a PDFA is $\equiv_{\mathcal{S}}$*-minimal* if every pair of distinct states are not $\equiv_{\mathcal{S}}$-equivalent.

We illustrate the construction of the quotient with the following example. Consider the PDFA in Fig. 6.4(a), with $\Sigma = \{0, 1\}$. For each state q_i, $i = 0, \ldots, 4$, $\pi(q_i) = \eta_i$ defined in Tab. 6.1. Let us describe the quotient state set $\dot{Q}$ in each case. For $=_\kappa$, with $\kappa = 2$, the quotient $\dot{Q}$ has five elements. In fact, for $j = 3, 4$, q_j cannot be in the same class as q_i, for $i = 0, 1, 2$, since $\pi^*(q_i, \lambda) = \pi(q_i) \neq_\kappa \pi(q_j) = \pi^*(q_j, \lambda)$ (Fig. 6.3). The following table shows that $q_0 \not\equiv_\kappa q_i$, for $i = 1, 2$, $q_1 \not\equiv_\kappa q_2$, and $q_3 \not\equiv_\kappa q_4$:

$$\begin{aligned} \pi^*(q_0, 11) = \eta_2 \neq_\kappa \eta_4 = \pi^*(q_1, 11) \qquad & \pi^*(q_0, 1) = \eta_1 \neq_\kappa \eta_4 = \pi^*(q_2, 1) \\ \pi^*(q_1, 1) = \eta_2 \neq_\kappa \eta_4 = \pi^*(q_2, 1) \qquad & \pi^*(q_3, 0) = \eta_2 \neq_\kappa \eta_4 = \pi^*(q_4, 0) \end{aligned}$$

In this case the quotient PDFA coincides with the original automaton.

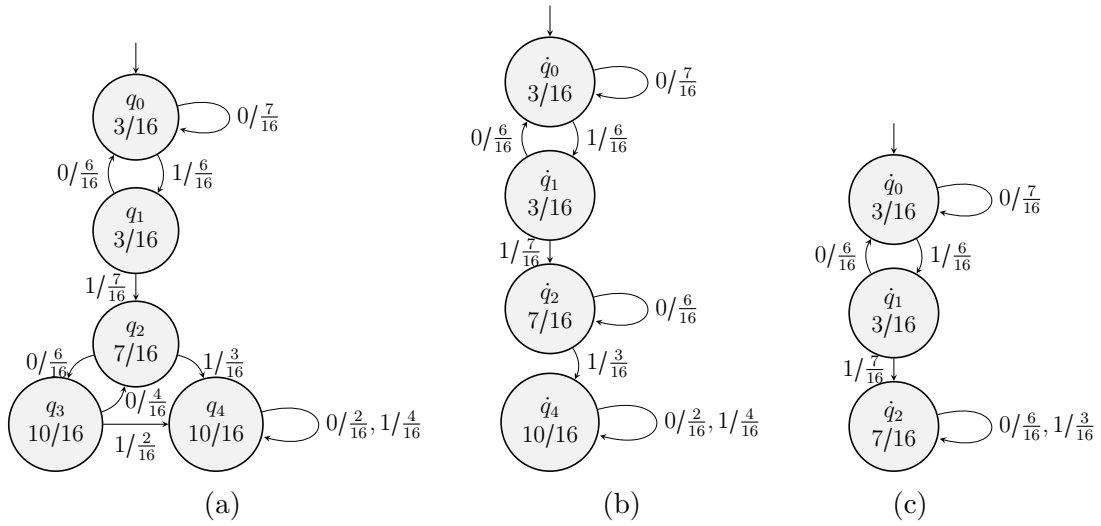


Figure 6.4: (a) Original PDFA (b) *rank*-Quotient PDFA (c) top_1-Quotient PDFA.

For *rank* the quotient $\dot{Q}$ has four elements. Considering the empty word we see that q_0, q_1 and q_4 are the unique elements of their own classes. Let us show that q_2 and q_3 are equivalent. By definition we must show that $\pi^*(q_2, w) =_{rank} \pi^*(q_3, w)$ for any string w. The proof is by induction. It holds for $w = \lambda$ since $\pi(q_2) =_{rank} \pi(q_3)$. Let $w = 0u$:

$$\pi^*(q_2, 0u) \triangleq \pi^*(\tau(q_2, 0), u) = \pi^*(q_3, u) \qquad \pi^*(q_3, 0u) \triangleq \pi^*(\tau(q_3, 0), u) = \pi^*(q_2, u)$$

By induction hypothesis $\pi^*(q_2, u) =_{rank} \pi^*(q_3, u)$, so $\pi^*(q_2, 0u) =_{rank} \pi^*(q_3, 0u)$. If $w = 1u$, transitions by 1 from both q_2 and q_3 go to q_4, so in this case the equality is trivial. The quotient PDFA is shown in Fig. 6.4(b), for $\alpha(\dot{q}_0) = \lambda$, $\alpha(\dot{q}_1) = 1$, $\alpha(\dot{q}_2) = 11$, $\alpha(\dot{q}_4) = 111$. For top_1 a similar argument shows that the quotient $\dot{Q}$ is the one shown in Fig. 6.4(c).

7 Table-based PDFA learning algorithms

In this chapter, we discuss a table-based learning algorithm (and some variants) which consists of a learner that iteratively constructs a PDFA by interacting with a MAT. This chapter introduces the use of an equivalence relation $\mathcal{S}$ for learning quotient PDFA with respect to $\equiv_{\mathcal{S}}$, and it will serve as a reference for comparison in further chapters. Section 7.1 formally presents the algorithm and proves its correctness and termination. Section 7.2 shows that other techniques like WL* [48] may fail to ensure minimality and termination.

7.1 Algorithm $\mathrm{L}^*_{\mathrm{p}}$

We present $\mathrm{L}^*_{\mathrm{p}}$, an abstract learning algorithm based on a data structure called observation table. $\mathrm{L}^*_{\mathrm{p}}$ is an adaptation of L^* for PDFA.

The learner uses an observation table $OT \subseteq \mathit{Pre} \times \mathit{Suf}$ for storing outcomes of **MQ** where $\mathit{Pre} \subset \Sigma^*$ is a prefix-closed set of *prefixes* (stored in row indices) and $\mathit{Suf} \subset \Sigma^*$ is a suffixed-closed set of *suffixes* (stored in column indices). *Pre* is divided in two parts, namely **RED** which are the rows used to construct *states*, and **BLUE** which are the rows representing continuations of **RED** by a symbol [23].

$\mathrm{L}^*_{\mathrm{p}}$ code structure is analogue to L^* (Alg. 2). It expands OT through the use of **MQ** until it becomes *closed* and *consistent*. Then it constructs a hypothesis automaton. Finally it calls **EQ** with the proposed hypothesis. These steps are repeated as long as **EQ** yields a counterexample, otherwise it stops and returns the last hypothesis. The main differences arise in the definitions of closedness and consistency needed for the expansion of OT, and the PDFA construction algorithm. These processes are explained below.

Observation table expansion

In order to define OT, we need to specify **MQ**. In this context the oracle **MQ** is:

$$\mathbf{MQ}(s) \triangleq \pi^*(s) \tag{7.1}$$

The results of **MQ**, called observations, are stored in a table. Besides, the following definitions will be helpful. For any set of strings $W \subseteq \Sigma^*$, we define:

$$\forall q, q' \in Q.\ q =_{\mathcal{S}}^{W} q' \iff \forall w \in W.\ \pi^*(q, w) =_{\mathcal{S}} \pi^*(q', w) \tag{7.2}$$

$$\forall u, u' \in \Sigma^*.\ u =_{\mathcal{S}}^{W} u' \iff \forall w \in W.\ \pi^*(uw) =_{\mathcal{S}} \pi^*(u'w) \tag{7.3}$$

Clearly, Def. 7.3 is equivalent to:

$$\forall u, u' \in \Sigma^*.\ u =_{\mathcal{S}}^{W} u' \iff \tau^*(u) =_{\mathcal{S}}^{W} \tau^*(u') \tag{7.4}$$

It is straightforward to show that $=_{\mathcal{S}}^{W}$ is an equivalence relation. Notice that $=_{\mathcal{S}}^{\Sigma^*}$ is $\equiv_{\mathcal{S}}$. Hereinafter, $[\![\cdot]\!]_{\mathcal{S}}^{W}$ denotes the classes defined by equivalence $=_{\mathcal{S}}^{W}$. For the sake of readability, $\mathcal{S}$ and W are omitted when they are clear from the context.

Now, OT is filled as follows:

$$\forall p \in Pre, s \in Suf.\ OT[p][s] \triangleq \mathbf{MQ}(ps) \tag{7.5}$$

OT is expanded until closedness and consistency are reached:

Closedness $\forall p \in \mathbf{BLUE}$, $\exists p' \in \mathbf{RED}$ such that $p =_{\mathcal{S}}^{Suf} p'$.

Consistency $\forall p, p' \in \mathbf{RED}$, if $p =_{\mathcal{S}}^{Suf} p'$ then $\forall \sigma \in \Sigma$, $p\sigma =_{\mathcal{S}}^{Suf} p'\sigma$.

Quotient PDFA construction

Unlike L* where a unique row represents a state, in L$_{\mathrm{p}}^*$ a state is the class of $=_{\mathcal{S}}^{Suf}$-related rows. Now it is possible to construct a hypothesis PDFA $\widehat{A}$ where:

- the set of states $\widehat{Q}$ is the quotient of Pre induced by $=_{\mathcal{S}}^{Suf}$, and
- the initial state $\widehat{q_{\mathrm{in}}} \triangleq [\![\lambda]\!]$.

For each state $\widehat{q}$ we choose a unique *representative* in **RED**, denoted $\alpha(\widehat{q})$. We define $\alpha(\widehat{q})$ to be any $p \in \mathbf{RED}$ such that $[\![p]\!] = \widehat{q}$. W.l.o.g. we assume $\alpha(\widehat{q}_{\text{in}}) \triangleq \lambda$. Now, for state $\widehat{q}$, the transition function $\widehat{\tau}$ and the probability distribution $\widehat{\pi}$ are defined as follows:

- $\widehat{\tau}(\widehat{q}, \sigma) \triangleq [\![\alpha(\widehat{q})\sigma]\!]$, and
- $\widehat{\pi}(\widehat{q})$ is any $\eta \in \Delta(\Sigma_\$)$ such that $\eta =_{\mathcal{S}} OT[\alpha(\widehat{q})][\lambda]$.

Checking Equivalence

In this chapter, **EQ** is any algorithm that checks Proposition 6.11. This can be done by adapting any algorithm for verifying $\equiv$-equivalence of DFA, such as [59] or the one used in Section 2.2.1.1, to checking $\equiv_{\mathcal{S}}$ on PDFA. Notice that this ensures *exact* $\equiv_{\mathcal{S}}$-equivalence. Nevertheless, this comparison can also be approximate, by adapting PAC error to this context:

$$\mathcal{E}^{\mathcal{S}}_{\mathcal{D},C}(H) = \mathbb{P}_{x\sim\mathcal{D}}\left[C(x) \not\equiv_{\mathcal{S}} H(x)\right] \tag{7.6}$$

7.1.1 Correctness and termination

It is straightforward to prove that $\mathrm{L}^*_{\mathrm{p}}$ is correct.

Proposition 7.1. *For any PDFA A, if $\mathrm{L}^*_{\mathrm{p}}$ with an exact* **EQ** *terminates, it computes a PDFA $\widehat{A} \equiv_{\mathcal{S}} A$.*

Proof. *If $\mathrm{L}^*_{\mathrm{p}}$ terminates with a hypothesis $\widehat{A}$, it means* **EQ** *returns no counterexample. Hence, $\widehat{A} \equiv_{\mathcal{S}} A$.*

In the case of a PAC **EQ**, we have:

Proposition 7.2. *The probability that at any stage algorithm $\mathrm{L}^*_{\mathrm{p}}$ terminates with an output $\widehat{A}$ that is not an ε-approximation of A is at most δ.*

Proof. *Analogous to Theorem 3.1.*

To prove termination we need to show some auxiliary results. For this matter we follow Angluin's approach [25].

Lemma 7.1. *If OT is closed and consistent then $\forall p \in Pre, \widehat{\tau}^*(p) = [\![p]\!]$.*

Proof. *By induction over $p \in Pre$.*

Base case $p = \lambda$.

$$\begin{aligned} \widehat{\tau}^*(\lambda) &= \widehat{q}_{\text{in}} && \textit{by Def. 6.7} \\ &= [\![\lambda]\!] && \textit{by construction} \end{aligned}$$

Inductive step $p = p'\sigma$.

$$\begin{aligned} \widehat{\tau}^*(p'\sigma) &= \widehat{\tau}(\widehat{\tau}^*(p'), \sigma) && \textit{by Def. 6.8} \\ &= \widehat{\tau}([\![p']\!], \sigma) && \textit{by inductive hypothesis} \\ &= [\![\alpha([\![p']\!])\sigma]\!] && \textit{by construction} \\ &= [\![p'\sigma]\!] && \textit{by consistency} \end{aligned}$$

Lemma 7.2. *If OT is closed and consistent then $\forall p \in Pre, s \in Suf.\ \widehat{\pi}^*(ps) =_{\mathcal{S}} \pi^*(ps)$.*

Proof. *By induction in the length of $s \in Suf$.*

Base case *Let $|s| = 0$, i.e., $s = \lambda$.*

$$\begin{aligned} \widehat{\pi}^*(p\lambda) &= \widehat{\pi}^*(p) && \textit{by definition} \\ &= \widehat{\pi}(\widehat{\tau}^*(p)) && \textit{by Def. 6.9} \\ &= \widehat{\pi}([\![p]\!]) && \textit{by proposition 7.1} \\ &=_{\mathcal{S}} OT[p][\lambda] && \textit{by construction} \\ &=_{\mathcal{S}} \mathbf{MQ}(p) && \textit{by Def. 7.5} \\ &=_{\mathcal{S}} \pi^*(p) && \textit{by Def. 7.1} \\ &=_{\mathcal{S}} \pi^*(p\lambda) && \textit{by definition} \end{aligned}$$

Inductive step *Assume it holds for all $s' \in Suf$, with $|s'| = n$. Let $s = \sigma s'$, with*

$|s'| = n$.

$$
\begin{aligned}
\widehat{\pi}^*(p\sigma s') &= \widehat{\pi}(\widehat{\tau}^*(p\sigma s')) && \text{by Def. 6.9}\\
&= \widehat{\pi}(\widehat{\tau}^*(\widehat{\tau}^*(p), \sigma s')) && \text{by Def. 6.8}\\
&= \widehat{\pi}(\widehat{\tau}^*(\llbracket p \rrbracket, \sigma s')) && \text{by Lemma 7.1}\\
&= \widehat{\pi}(\widehat{\tau}^*(\widehat{\tau}(\llbracket p \rrbracket, \sigma), s')) && \text{by Def. 6.8}\\
&= \widehat{\pi}(\widehat{\tau}^*(\widehat{\tau}(\llbracket p' \rrbracket, \sigma), s')) && \alpha(\llbracket p \rrbracket) \triangleq p' \in \mathbf{RED}\\
&= \widehat{\pi}(\widehat{\tau}^*(\llbracket p'\sigma \rrbracket, s')) && \text{by construction}\\
&= \widehat{\pi}(\widehat{\tau}^*(\widehat{\tau}^*(p'\sigma), s')) && \text{by Lemma 7.1}\\
&= \widehat{\pi}(\widehat{\tau}^*(p'\sigma s')) && \text{by Def. 6.8}\\
&= \widehat{\pi}^*(p'\sigma s') && \text{by Def. 6.9}\\
&=_{\mathcal{S}} \pi^*(p'\sigma s') && \text{by i.h: } p'\sigma \in Pre \wedge |s'| = n\\
&=_{\mathcal{S}} \mathbf{MQ}(p'\sigma s') && \text{by Def. 7.1}\\
&=_{\mathcal{S}} OT[p'][\sigma s'] && \text{by Def. 7.5}\\
&=_{\mathcal{S}} OT[p][\sigma s'] && \text{by } \llbracket p \rrbracket = \llbracket p' \rrbracket\\
&=_{\mathcal{S}} \mathbf{MQ}(p\sigma s') && \text{by Def. 7.5}\\
&=_{\mathcal{S}} \pi^*(p\sigma s') && \text{by Def. 7.1}
\end{aligned}
$$

A PDFA $\overline{A}$ is said to be *consistent* with OT iff

$$\forall p \in Pre, s \in Suf.\ \overline{\pi}^*(ps) =_{\mathcal{S}} OT[p][s] \tag{7.7}$$

Corollary 7.1. *Let OT be closed and consistent, $\widehat{A}$ be the PDFA built from OT, and $\overline{A}$ be a PDFA consistent with OT: $\forall p \in Pre, s \in Suf.\ \overline{\pi}^*(ps) =_{\mathcal{S}} \widehat{\pi}^*(ps)$.*

Proof.

$$
\begin{aligned}
\overline{\pi}^*(ps) &=_{\mathcal{S}} OT[p][s] && \text{by Def. 7.7}\\
&=_{\mathcal{S}} \pi^*(ps) && \text{by Def. 7.5 and Def. 7.1}\\
&=_{\mathcal{S}} \widehat{\pi}^*(ps) && \text{by Lemma 7.2}
\end{aligned}
$$

Lemma 7.3. *Let OT be closed and consistent and $\overline{A}$ be a PDFA consistent with OT: $\forall p, p' \in Pre.\ \llbracket p \rrbracket = \llbracket p' \rrbracket \implies \overline{\tau}^*(p) =_{\mathcal{S}}^{Suf} \overline{\tau}^*(p')$.*

Proof.

$$
\begin{aligned}
\llbracket p \rrbracket = \llbracket p' \rrbracket &\implies \forall s \in Suf.\ \pi^*(ps) =_{\mathcal{S}} \pi^*(p's) && \text{by Def. 7.3}\\
&\implies \forall s \in Suf.\ OT[p][s] =_{\mathcal{S}} OT[p'][s] && \text{by Def. 7.5 and Def. 7.1}\\
&\implies \forall s \in Suf.\ \overline{\pi}^*(ps) =_{\mathcal{S}} \overline{\pi}^*(p's) && \text{by Def. 7.7}\\
&\implies \overline{\tau}^*(p) =_{\mathcal{S}}^{Suf} \overline{\tau}^*(p') && \text{by Def. 7.3}
\end{aligned}
$$

Lemma 7.4. *Let OT be closed and consistent and $\widehat{A}$ be the PDFA built from OT. Every PDFA $\overline{A}$ consistent with OT has at least the same number of states.*

Proof. *Let $\phi : \widehat{Q} \to \overline{Q}$ be such that $\phi([\![p]\!]) \triangleq \overline{\tau}^*(p)$. We prove that ϕ is a one-to-one morphism.*

ϕ is one-to-one *Suppose it is not. Then, there exists $p, p' \in$ **RED** such that $[\![p]\!] \neq [\![p']\!]$, and $\phi([\![p]\!]) = \phi([\![p']\!])$, that is, $\overline{\tau}^*(p) = \overline{\tau}^*(p')$. Therefore, for all $s \in Suf$, $\overline{\tau}^*(ps) = \overline{\tau}^*(p's)$. Then, $\overline{\pi}^*(ps) = \overline{\pi}^*(p's)$. Thus, by Coro. 7.1, $\widehat{\pi}^*(ps) =_{\mathcal{S}} \widehat{\pi}^*(p's)$, and so $p =_{\mathcal{S}}^{Suf} p'$, that is, $[\![p]\!] = [\![p']\!]$, which is a contradiction.*

ϕ is a morphism

$$
\begin{aligned}
\phi(\widehat{\tau}(\widehat{q}, \sigma)) &= \phi(\widehat{\tau}([\![p]\!], \sigma)) && \text{where } p = \alpha(\widehat{q}) \\
&= \phi([\![p\sigma]\!]) && \text{by construction of } \widehat{A} \\
&= \overline{\tau}^*(\alpha([\![p\sigma]\!])) && \text{by definition of } \phi \\
&=_{\mathcal{S}}^{Suf} \overline{\tau}^*(p\sigma) && \text{by Lemma 7.3} \\
&=_{\mathcal{S}}^{Suf} \overline{\tau}(\overline{\tau}^*(p), \sigma) && \text{by Def. 6.8} \\
&=_{\mathcal{S}}^{Suf} \overline{\tau}(\phi([\![p]\!]), \sigma) && \text{by definition of } \phi \\
&=_{\mathcal{S}}^{Suf} \overline{\tau}(\phi(\widehat{q}), \sigma) && p = \alpha(\widehat{q})
\end{aligned}
$$

Corollary 7.2. *Let OT be closed and consistent and $\widehat{A}$ be the PDFA built from OT: $|\widehat{Q}| \leq |\dot{Q}|$.*

Proof. *Let $\overline{A} \in A/_{\equiv_{\mathcal{S}}}$.*

$$
\begin{aligned}
\overline{A} \in A/_{\equiv_{\mathcal{S}}} &\implies \overline{A} \equiv_{\mathcal{S}} A && \text{By Def. 6.18} \\
&\implies \overline{q}_{\text{in}} \equiv_{\mathcal{S}} q_{\text{in}} && \text{By Prop. 6.11} \\
&\implies \forall s \in \Sigma^*.\ \overline{\pi}^*(s) =_{\mathcal{S}} \pi^*(s) && \text{By Def. 6.17} \\
&\implies \forall p \in Pre, s \in Suf.\ \overline{\pi}^*(ps) =_{\mathcal{S}} OT[p][s] && \text{By Def. 7.1 and Def. 7.5} \\
&\implies \overline{A} \text{ is consistent with } OT && \text{By Def. 7.7}
\end{aligned}
$$

Hence, by Lemma 7.4, $|\widehat{Q}| \leq |\overline{Q}| = |\dot{Q}|$.

Lemma 7.5. *Let OT_i be closed and consistent, $\gamma \in \Sigma^*$ a counterexample, and OT_{i+1} the new table obtained by the algorithm. Then $|\widehat{Q}_i| < |\widehat{Q}_{i+1}|$.*

Proof. *We have that γ is such that $\widehat{\pi}^*(\gamma) \neq_{\mathcal{S}} \pi^*(\gamma)$. When adding γ and all its prefixes and continuations, either:*

1. *A prefix p is added to Pre_{i+1} such that for all $p' \in \mathbf{RED}_i, [\![p']\!] \neq [\![p]\!]$. In this case, either*
   - *p is in $\mathbf{RED}_{i+1}$ because it is γ or a prefix of γ, or*
   - *p is first added to $\mathbf{BLUE}_{i+1}$ and then it is moved to $\mathbf{RED}_{i+1}$ when closing the table.*
2. *Otherwise, for all Pre_{i+1}, there exists $p' \in \mathbf{RED}_i, [\![p']\!] = [\![p]\!]$. That is, the table remains closed. Suppose it is consistent. Then $\widehat{\pi}^*(\gamma) =_{\mathcal{S}} \pi^*(\gamma)$ by Lemma 7.2, because $\gamma \in Pre_{i+1}$ and $\lambda \in Suf_{i+1}$, which is a contradiction because γ is a counterexample. Hence, it is not consistent. Exists $p, p' \in \mathbf{RED}_{i+1}$ such that $[\![p]\!] = [\![p']\!]$ but $[\![p\sigma]\!] = [\![p'\sigma]\!]$ for some $\sigma \in \Sigma$. Making the table consistent consists in finding an $s \in Suf_{i+1}$ such that $OT_{i+1}[p\sigma][s] \neq_{\mathcal{S}} OT_{i+1}[p'\sigma][s]$, and adding σs to Suf_{i+1}. This will make $p \neq_{\mathcal{S}}^{Suf_{i+1}} p'$, or equivalently $[\![p]\!] \neq [\![p']\!]$.*

This implies $|\widehat{Q}_i| < |\widehat{Q}_{i+1}|$.

Proposition 7.3. *For any PDFA A, $\mathrm{L}^*_{\mathrm{p}}$ terminates.*

Proof. *Whenever a counterexample γ is returned by* **EQ** *at iteration i, Lemma 7.5, implies the number of states of the next hypothesis $\widehat{A}_{i+1}$ strictly increases. Moreover, by Lemma 7.4, $\widehat{A}_{i+1}$ is the smallest PDFA which is consistent with OT_{i+1}. Therefore, by Coro. 7.2, $|\widehat{Q}_{i+1}| \leq |\dot{Q}|$, and if $|\widehat{Q}_{i+1}| = |\dot{Q}|$ then $\mathrm{L}^*_{\mathrm{p}}$ terminates because, by Lemma 7.4 and Lemma 7.5, $\widehat{A}_{i+1}$ is in $\dot{A}$, which implies $\widehat{A}_{i+1} \equiv_{\mathcal{S}} A$. Therefore, $\mathrm{L}^*_{\mathrm{p}}$ terminates in at most $|\dot{Q}|$ iterations.*

Theorem 7.1. *For any PDFA A, $\mathrm{L}^*_{\mathrm{p}}$ with exact* **EQ** *terminates and computes a PDFA $\widehat{A} \equiv_{\mathcal{S}} A$.*

Proof. *By Prop. 7.1 and Prop. 7.3.*

Theorem 7.2. *For any PDFA A, $\mathrm{L}^*_{\mathrm{p}}$ with PAC* **EQ** *terminates and computes a PDFA $\widehat{A}$ such that the probability of $\widehat{A}$ not being an ε-approximation of A is at most δ.*

Proof. *By Prop. 7.2 and Prop. 7.3.*

7.1.2 Columnar version $\mathrm{L}^*_{\mathrm{p}}\mathrm{Col}$

Following the ideas presented in [41], it is possible to avoid checking for consistency. Each time a counterexample is found, rather than adding its prefixes to **RED**, its suffixes are added to *Suf*. By doing so, the table remains consistent. Therefore, elements from **BLUE** are added to **RED** only to ensure closedness. This happens when for some $p \in \mathbf{BLUE}$ there is no $p' \in \mathbf{RED}$ such that $[\![p]\!] = [\![p']\!]$. With this in mind we developed a column expansion version of $\mathrm{L}^*_{\mathrm{p}}$ that we called $\mathrm{L}^*_{\mathrm{p}}\mathrm{Col}$.

7.2 Non-equivalence relations: t-tolerance

Previous work, such as WL* [48], addressed the issue of noise or variations in the probability distribution of a state by introducing a *tolerance* parameter $t \in [0, 1]$. Two distributions $\eta, \eta' \in \Delta(\Sigma_\$)$, are defined to be t-similar, denoted $\eta \approx_t \eta$ whenever $L_\infty(\eta, \eta') \leq t$ where for any $v, v' \in \mathbb{R}^n$, $L_\infty(v, v') = \max_i |v(i) - v'(i)|$. Notice that $\approx_t$ is not an equivalence relation, as it is not transitive. Based on $\approx_t$, the following relation between states, called t-equality, is defined in [48]:

$$\forall q, q' \in Q.\ q \approx_t q' \iff \forall s \in \Sigma^*.\ \pi^*(q, s) \approx_t \pi^*(q', s). \tag{7.8}$$

Two PDFA A and B are t-equal, denoted $A \approx_t B$, if $q^A_{\mathrm{in}} \approx_t q^B_{\mathrm{in}}$. However, though misleadingly called t-equality, $\approx_t$ is not an equivalence relation between states because it is not transitive.

It is worth noticing that κ-equivalence entails t-equality for $t = \kappa^{-1}$.

Proposition 7.4. *For every $q, q' \in Q$, if $q \equiv_\kappa q'$ then $q \approx_{\kappa^{-1}} q'$.*

Proof. *$q \equiv_\kappa q'$ implies $\pi^*(q, s) =_\kappa \pi^*(q', s)$ for all $s \in \Sigma^*$. Thus, $|\pi^*(q, s) - \pi^*(q', s)| \leq \kappa^{-1}$ for all $s \in \Sigma^*$. Hence, $q \approx_{\kappa^{-1}} q'$.*

Corollary 7.3. *For every PDFA A and B, if $A \equiv_\kappa B$ then $A \approx_{\kappa^{-1}} B$.*

Loosing equivalence by relaxing transitivity comes at a price. Notice that several proofs of the auxiliary results to show termination of $\mathrm{L}^*_{\mathrm{p}}$ rely on transitivity. The following two examples pinpoint several places where the previous proofs fail if we use $\approx_t$ instead of $=_{\mathcal{S}}$. Example 7.1 shows that Lemma 7.4 does not hold for $\approx_t$, while Example 7.2 shows that Lemma 7.5 does not hold for $\approx_t$.

Example 7.1. Consider the OT shown in Figure 7.1(Left) and let t be 0.15. OT is closed and consistent. Any PDFA $\widehat{A}$ built from OT will have at least two states because $OT[\lambda][\lambda] \not\approx_t OT[a][\lambda]$. However, the PDFA depicted in Figure 7.1(Right) is consistent with OT because $\overline{\pi}^*(\lambda\lambda) \approx_t OT[\lambda][\lambda]$, $\overline{\pi}^*(a\lambda) \approx_t OT[a][\lambda]$, and $\overline{\pi}^*(aa\lambda) \approx_t OT[a][\lambda]$, but it has only one state.

		λ
RED	λ	[0.4, 0.6]
	a	[0.6, 0.4]
BLUE	aa	[0.4, 0.6]

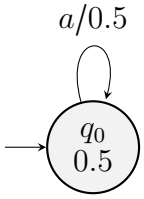


Figure 7.1: (Left) OT. (Right) $\overline{A}$.

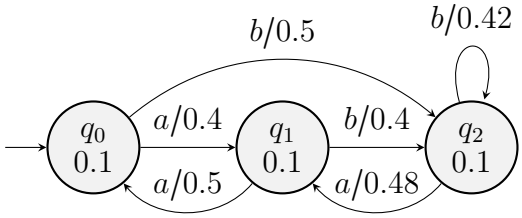


Figure 7.2: Target PDFA A.

Example 7.2. Let us consider the target PDFA A shown in Figure 7.2 and let t be 0.09. The initial observation table, OT_0 would be the one shown in Figure 7.3(left). OT_0 is not closed as there is no $p \in RED$ such that $OT_0[p] \approx_t OT_0[a]$. Once the table is closed, we get OT_0', shown in Figure 7.3(right), which is closed and consistent. At this point depending on the clustering technique, rows may be grouped in two ways: 1) $\{\lambda, aa, b, ab\}$ and $\{a\}$, or 2) $\{\lambda, aa\}$ and $\{a, b, ab\}$, which result in hypotheses PDFA $\widehat{A}_0^1$ and $\widehat{A}_0^2$, respectively (Figure 7.4).

		λ
RED	λ	[0.1, 0.4, 0.5]
BLUE	a	[0.1, 0.5, 0.4]
	b	[0.1, 0.48, 0.42]

		λ
RED	λ	[0.1, 0.4, 0.5]
	a	[0.1, 0.5, 0.4]
BLUE	b	[0.1, 0.48, 0.42]
	aa	[0.1, 0.4, 0.5]
	ab	[0.1, 0.48, 0.42]

Figure 7.3: Observation tables OT_0 (left) and OT_0' (right).

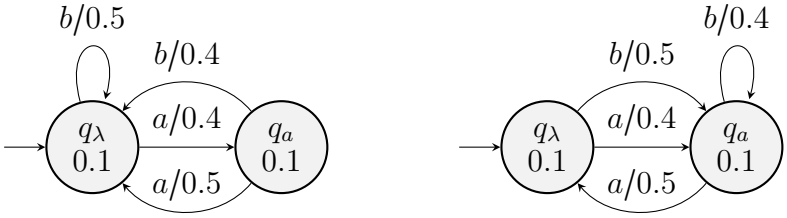


Figure 7.4: Hiptotheses PDFA $\widehat{A}_0^1$ (left) and $\widehat{A}_0^2$ (right).

In first case, the algorithm terminates with a PDFA $\widehat{A}_0^1 \approx_t A$. However, in the second one, $\widehat{A}_0^2 \not\approx_t A$, with the the string ba being a counterexample γ. Processing γ gives the observation table OT_1, shown in Figure 7.5(left), which is closed but not consistent. Making it consistent gives OT_1' (see Figure 7.5(right)).

		λ
RED	λ	[0.1, 0.4, 0.5]
	a	[0.1, 0.5, 0.4]
	b	[0.1, 0.48, 0.42]
	ba	[0.1, 0.5, 0.4]
BLUE	bb	[0.1, 0.48, 0.42]
	aa	[0.1, 0.4, 0.5]
	ab	[0.1, 0.48, 0.42]
	baa	[0.1, 0.4, 0.5]
	bab	[0.1, 0.48, 0.42]

		λ	a
RED	λ	[0.1, 0.4, 0.5]	[0.1, 0.5, 0.4]
	a	[0.1, 0.5, 0.4]	[0.1, 0.4, 0.5]
	b	[0.1, 0.48, 0.42]	[0.1, 0.5, 0.4]
	ba	[0.1, 0.5, 0.4]	[0.1, 0.4, 0.5]
BLUE	bb	[0.1, 0.48, 0.42]	[0.1, 0.5, 0.4]
	aa	[0.1, 0.4, 0.5]	[0.1, 0.5, 0.4]
	ab	[0.1, 0.48, 0.42]	[0.1, 0.5, 0.4]
	baa	[0.1, 0.4, 0.5]	[0.1, 0.5, 0.4]
	bab	[0.1, 0.48, 0.42]	[0.1, 0.5, 0.4]

Figure 7.5: Observation tables OT_1 (left) and OT_1' (right).

Now, rows can only be grouped in the following way: $\{\lambda, b, bb, aa, ab, baa, bab\}$ and $\{a, ba\}$. This results in the hypothesis PDFA $\widehat{A}_0^1$ shown in Figure 7.4(left), which has exactly the same number of states than the one obtained in the previous iteration. Hence, Lemma 7.5 does not hold for $\approx_t$, which hampers the proof of termination.

8 A tree-based PDFA learning algorithm

In this chapter we present a tree-based algorithm for learning PDFA that we will call QNT (Quantization N-ary Tree Learner). In contrast to KV [60] (Kearns and Vazirani) and TTT [46] (named after the three data structures it handles: spanning **T**ree, discrimination **T**ree and discriminator **T**rie) which build a binary tree, our algorithm handles general trees with arbitrary degree, similar to [61] for Mealy machines. This structure is shown to be more compact than observation tables, allowing for more efficient representations and updates, which entail more performant learning algorithms.

8.1 N-ary tree

A tree T maintains a set $Acc \subset \Sigma^*$ of access strings and a set $Dis \subset \Sigma^*$ of distinguishing strings with the following properties:

Property 8.1. *Each $u \in Acc$ is bound to a unique leaf in T labeled with $\mathbf{MQ}(u)$.*

Property 8.2. *Every inner node is labeled with a string $w \in Dis$.*

Property 8.3. *The empty word λ belongs to both Acc and Dis. The root of T is labeled with λ. Also, there is a leaf for λ.*

Property 8.4. *Arcs in T are labeled with classes in $[\![\Delta(\Sigma_\$)]\!]$. Every outgoing arc from an inner node is labeled with a different class.*

Property 8.5. *Strings in Acc and Dis are connected as follows: for every pair of distinct strings $u, u' \in Acc$, the string $w \in Dis$ which is the lowest common ancestor of u and u' in T, denoted $lca(u, u')$, is such that $\mathcal{M}(uw) \neq_{\mathcal{S}} \mathcal{M}(u'w)$. Therefore, $u \not\equiv_{\mathcal{S}} u'$.*

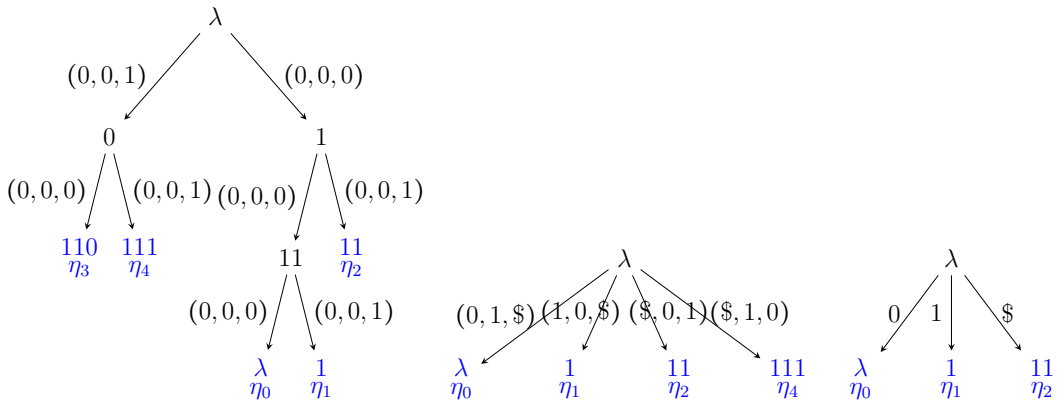


Figure 8.1: From left to right: trees for Fig. 6.4(a-c), respectively.

Let ϕ_T be $\phi_T : Acc \to [\![\Sigma^*]\!]$, with $\phi_T(u) = [\![u]\!]$.

Proposition 8.1. *ϕ_T is one-to-one.*

Proof. *Immediate from Property 8.5.*

If $\mathcal{S}$ induces an infinite partition of $\Delta(\Sigma_\$)$, the degree of a node may be unbounded. Indeed, it may be infinite if $[\![\Sigma^*]\!]$ is infinite. Notice that this does not occur for non-binary trees used for Mealy machines [61]. We will come back to this issue later in Sec. 8.6 after we discuss the learning algorithm.

Fig. 8.1 shows the trees corresponding to the examples in Fig. 6.4. For $=_\kappa$, with $\kappa = 2$, the leafs of the tree in Fig. 8.1(a) correspond to the states of the PDFA of Fig. 6.4(a), identified with their associated access strings: $Acc = \{\lambda, 1, 11, 110, 111\}$. Every leaf is labeled with the probability distribution of the state. Tree arcs are labeled with quantization classes, represented with their partition indexes. For instance, $(1, 0, 0)$ corresponds to the quantization class (I_2^1, I_2^0, I_2^0), where the first coordinate corresponds to the symbol 0, the second to 1, and the third to \$. The root λ of the tree has an arc for each one of the classes in which quantization partitions the set of probability distributions of the states of the PDFA. In the example, there are two, namely $(0, 0, 0)$ and $(0, 0, 1)$. The tree explains that the five states are not $\equiv_\kappa$-equivalent. Indeed, even if there are states with $=_\kappa$-equivalent distributions, like for example q_0 and q_1, the tree shows how the set $Dis = \{\lambda, 0, 1, 11\}$ distinguishes them: $\pi^*(q_0, 11) \in (0, 0, 0)$ and $\pi^*(q_1, 11) \in (0, 0, 1)$. Fig. 8.1(b) corresponds to the PDFA of Fig. 6.4(b) obtained with *rank*. In this case we label the classes by their ranking, for example η_0 ranks first the symbol 0, second the symbol 1 and third the symbol \$, so it belongs to the class labeled $(0, 1, \$)$. Here,

$Acc = \{\lambda, 1, 11, 111\}$ and $Dis = \{\lambda\}$, respectively. Fig. 8.1(c) corresponds to the PDFA of Fig. 6.4(c) obtained with top_1. In this case we label the classes by their top symbol, for example η_2 ranks first the symbol \$, so it belongs to the class labeled \$. Also, $Acc = \{\lambda, 1, 11\}$ and $Dis = \{\lambda\}$.

T defines an equivalence $=_T \subseteq \Sigma^* \times \Sigma^*$. Let $\zeta_u \subseteq Dis$ be the set of distinguishing strings in the path from the root to $u \in Acc$:

$$u_1 =_T u_2 \overset{\triangle}{\iff} \exists u \in Acc.\ \forall w \in \zeta_u.\ \mathcal{M}(u_1 w) =_{\mathcal{S}} \mathcal{M}(uw) =_{\mathcal{S}} \mathcal{M}(u_2 w) \tag{8.1}$$

If $v \in \Sigma^*$ is $=_T$-equivalent to some $u \in Acc$ then $v \not\equiv_{\mathcal{S}} u'$ for any other access string $u' \neq u$. As a corollary, v can be $=_T$-equivalent to at most one access string in T.

Proposition 8.2. *Let* $u \in Acc,\ v \in \Sigma^*,\ u =_T v.\ \forall u' \in Acc.\ u' \neq u \implies [\![v]\!] \neq [\![u']\!]$.

Proof.

$$\begin{aligned} u \neq u' &\implies \exists w \in Dis.\ w = lca(u', u) && \text{By Property 8.5} \\ &\implies \mathcal{M}(u'w) \not\equiv_{\mathcal{S}} \mathcal{M}(uw) && \text{By Property 8.5} \\ &\implies \mathcal{M}(u'w) \not\equiv_{\mathcal{S}} \mathcal{M}(vw) && \text{By } u =_T v \text{ and Def. 8.1} \end{aligned}$$

Hence, $[\![v]\!] \neq [\![u']\!]$.

8.2 *sift* operation

To find the $=_T$-equivalent leaf (if exists) of v, we define the function *sift*(v) as follows. *sift* starts at the root of T and proceeds recursively. If the current node is a leaf $u \in Acc$, it returns u. Otherwise, let $w \in Dis$ be the distinguishing string at the current inner node. If there is an arc labeled $[\![\mathbf{MQ}(vw)]\!]$, *sift* recursively descends through the arc to the subtree. Otherwise, it means $\mathbf{MQ}(vw) \not\equiv_{\mathcal{S}} \mathbf{MQ}(uw)$ for all descendant leafs u of w. In such case, *sift* updates the tree as follows: it adds v to Acc labeled with $\mathbf{MQ}(v)$ and a new arc from w to v labeled with $[\![\mathbf{MQ}(vw)]\!]$, and it returns v*sift* updates are necessary since arcs are discovered on-the-fly because $[\![\Delta(\Sigma_\$)]\!]$ may be unbounded. It is important to remark that, in case *sift* modifies T, only its degree can grow but never its depth.

Proposition 8.3. *For all* $v \in \Sigma^*$, *sift*(v) *returns a leaf* u *such that* $v =_T u$ *and* $[\![v]\!] \neq [\![u']\!]$ *for all leafs* $u' \neq u$. *Also, sift maintains the properties of the (possibly updated) tree.*

Proof. *Suppose sift*(v) *returns an existing leaf* u. *It means it recursively traversed the path* ζ_u *from the root to* u. *Then, by Def. 8.1,* $v =_T u$, *and by Prop. 8.2,*

$\forall u' \in Acc.\ u' \neq u \implies [\![v]\!] \neq [\![u']\!]$. *Suppose* $sift(v)$ *returns* v *after updating the tree. Obviously,* $v =_T v$. *Now, this case occurs because sift arrives at an inner node with label* $w \in Dis$ *for which there is no outgoing arc* $=_{\mathcal{S}}$*-equivalent to* $\mathcal{M}(vw)$. *Then,* $[\![v]\!] \neq [\![u']\!]$ *for any descendant leaf* u' *of the current inner node because* $\mathcal{M}(vw) \neq_{\mathcal{S}} \mathcal{M}(u'w)$, *and sift ensures* $w = lca(u', v)$. *Also,* $[\![v]\!] \neq [\![u']\!]$ *for any other leaf* u' *in the tree because* $\mathcal{M}(vw') \neq_{\mathcal{S}} \mathcal{M}(u'w')$ *for* $w' = lca(u', w)$, *and sift guarantees* $w' = lca(u', v)$. *Therefore,* $[\![v]\!] \neq [\![u']\!]$ *for all already existing leafs* u'. *Moreover, the newly updated tree is such that every pair of distinct leafs is distinguished by its lowest common ancestor and so* ϕ_T *is one-to-one.*

8.3 *build* operation

Given T, *build* constructs a PDFA $A \triangleq (Q, q_{\text{in}}, \pi, \tau)$ where: $Q \triangleq \{q_u \mid u \in Acc\}$ with $\alpha(q_u) \triangleq u$; $q_{\text{in}} \triangleq q_\lambda$; and for all $q_u \in Q$, $\pi(q_u) \triangleq \mathcal{M}(u)$, and for all $\sigma \in \Sigma$, $\tau(q_u, \sigma) \triangleq q_{u'}$ with $u' = sift(u\sigma)$. Whenever *sift* adds a new leaf, it is restarted with the updated tree.

Proposition 8.4. *1) If build terminates, the output* A *is irreducible. 2) If* $[\![\Sigma^*]\!]$ *is finite,* $|Q|$ *is bounded by the size of* $[\![\Sigma^*]\!]$. *3) If one of* $[\![\Delta(\Sigma_\$)]\!]$ *or* $[\![\Sigma^*]\!]$ *is finite, build terminates.*

Proof. *Since* ϕ_T *is one-to-one,* $q \neq q' \implies q \not\equiv_{\mathcal{S}} q'$. *This implies that, if build terminates,* A *is irreducible, and if* $[\![\Sigma^*]\!]$ *is finite,* $|Q|$ *is at most the size of* $[\![\Sigma^*]\!]$ *and also, by Prop. 8.3, build terminates. If* $[\![\Delta(\Sigma_\$)]\!]$ *is finite, the degree of* T *is bounded by the size of* $[\![\Delta(\Sigma_\$)]\!]$. *Since sift can only make* T *to grow in width, it follows build terminates.*

If either $[\![\Delta(\Sigma_\$)]\!]$ or $[\![\Sigma^*]\!]$ is infinite, T could grow without bound. We address this issue in 8.6. Assume A is constructed. If T and A agree for some $v \in \Sigma^*$ but differ in $v\sigma$ for some $\sigma \in \Sigma_\$$ then $[\![v]\!]$ is not a leaf. Moreover, v and $sift(v)$ are not $\equiv_{\mathcal{S}}$-equivalent. Formally:

Proposition 8.5. *Let* $v \in \Sigma^*, \sigma \in \Sigma$. *If* $sift(v) = \alpha(\tau^*(v))$ *and* $sift(v\sigma) \neq \alpha(\tau^*(v\sigma))$ *then (1)* $v \notin Acc$, *and (2)* $sift(v) \not\equiv_{\mathcal{S}} v$.

Proof. *Let* $u_1 = sift(v)$, $u_2 = sift(v\sigma)$, $u_2' = \alpha(\tau^*(v\sigma))$, *and* $w = lca(u_2, u_2')$.
(1) Suppose $v \in Acc$. *By Prop. 8.3,* $u_1 = v$, *and by construction,* $\tau(q_v, \sigma) = q_{u_2}$. *Besides,* $\tau^*(v\sigma) = \tau(\tau^*(v), \sigma) = \tau(q_v, \sigma)$. *Then,* $\tau^*(v\sigma) = q_{u_2}$. *So,* $u_2' = \alpha(\tau^*(v\sigma)) = \alpha(q_{u_2}) = u_2$ *which contradicts the hypothesis* $u_2 \neq u_2'$. *Hence,* $v \notin Acc$.
(2) (i) By Prop. 8.3, $u_2 = sift(v\sigma)$ *implies* $v\sigma =_T u_2$. *So,* $\mathcal{M}(v\sigma w) =_{\mathcal{S}} \mathcal{M}(u_2 w)$

since $w \in \zeta_{u_2}$.
(ii) From $u_1 = \alpha(\tau^*(v))$ *and* $u'_2 = \alpha(\tau^*(v\sigma))$, *it follows by construction of* A *that* $u'_2 = sift(u_1\sigma)$. *Then, by Prop. 8.3,* $u_1\sigma =_T u'_2$. *So,* $\mathcal{M}(u_1\sigma w) =_{\mathcal{S}} \mathcal{M}(u'_2 w)$ *since* $w \in \zeta_{u'_2}$.
Therefore, from $\mathcal{M}(u_2 w) \neq_{\mathcal{S}} \mathcal{M}(u'_2 w)$, *(i) and (ii) it follows that* $\mathcal{M}(v\sigma w) \neq_{\mathcal{S}} \mathcal{M}(u_1\sigma w)$. *So,* $u_1 \not\equiv_{\mathcal{S}} v$. *Hence,* $sift(v) \not\equiv_{\mathcal{S}} v$.

8.4 *update* operation

The procedure *update* modifies T when $\mathbf{EQ}(A)$ returns a counterexample γ as follows.

Let $u_i = sift(\gamma[i])$ and $u'_i = \alpha(\tau^*(\gamma[i]))$. Since $\mathcal{M}(\gamma) \neq_{\mathcal{S}} \pi^*(\gamma)$, there is some i such that $u_i \neq u'_i$. Let j be the first such index. Therefore, Prop. 8.5 implies $\gamma[j-1] \notin Acc$ and $u_{j-1} \not\equiv_{\mathcal{S}} \gamma[j-1]$. Moreover, $\gamma_j w$ distinguishes u_{j-1} and $\gamma[j-1]$ with $w = lca(u_j, u'_j)$.

With this new evidence, *update* adds $\gamma[j-1]$ to Acc, and the leaf u_{j-1} is replaced by an inner node $\gamma_j w$ and two children, namely u_{j-1} and $\gamma[j-1]$.

Proposition 8.6. *update maintains the properties of the tree.*

Proof. *Direct from update definition.*

8.5 QNT algorithm

Algorithm 6: Tree-based learning algorithm.

Parameter: Equivalence relation $\mathcal{S}$
Output : PDFA A

$A \leftarrow \text{CreateInitialHypothesis}(\mathcal{S})$;
$\gamma \leftarrow \mathbf{EQ}(A, \mathcal{S})$;
if $\gamma = \perp$ **then**
  return A;
$T \leftarrow \text{InitializeTree}(\gamma, \mathcal{S})$;
while $\gamma \neq \perp$ **do**
  $A \leftarrow \mathit{build}(T)$; ;
  $\gamma \leftarrow \mathbf{EQ}(A, \mathcal{S})$;
  if $\gamma \neq \perp$ **then**
    $T \leftarrow \mathit{update}(T, \gamma, \mathcal{S})$;
return A;

Algorithm 6 sketches the code. It starts creating an initial A, with a single state q_λ with a loop for each symbol and probability distribution $\mathbf{MQ}(\lambda)$. Then, it calls $\mathbf{EQ}(A)$, which either returns $\perp$ and terminates or a counterexample γ triggering the initialization of T. The first instance of T has a root λ and two children, one labeled λ and the other γ. In the main loop, Algorithm 6 uses T to build a PDFA A, then calls $\mathbf{EQ}(A)$. If a counterexample is returned, T is updated and the loop restarts. Otherwise, it terminates. Algorithm 6 learns an $\mathcal{S}$-equivalent PDFA.

Proposition 8.7 (Correctness)**.** *If Algorithm 6 terminates, it outputs an irreducible PDFA which is $\equiv_{\mathcal{S}}$-equivalent to $\mathcal{M}$.*

Proof. *Let A be the output of Algorithm 6. Then, $A \equiv_{\mathcal{S}} \mathcal{M}$ since $\mathbf{EQ}(A)$ does not return a counterexample. By Prop. 8.4(1), A is irreducible.*

Proposition 8.8 (Strict progress)**.** *Let A_i be the PDFA built by Algorithm 6 at iteration i. If $\mathbf{EQ}(A_i) \neq \perp$ then A_{i+1}, if built, has strictly more states than A_i.*

Proof. *If $\mathbf{EQ}(A_i) \neq \perp$, update adds a new leaf to T. Hence, $|Q_{i+1}| > |Q_i|$.*

Corollary 8.1. *Algorithm 6 always terminates if $[\![\Sigma^*]\!]$ is finite.*

Proof. *From Prop. 8.8, Prop. 8.4, Prop. 6.12, and Prop. 6.2.*

Theorem 8.1. *For any $\mathcal{S}$-regular language model $\mathcal{M}$,* QNT *terminates and returns a PDFA A, such that $A \equiv_{\mathcal{S}} \mathcal{M}$.*

Proof. *By Def. 6.16 and Cor. 8.1* QNT *terminates. By Prop. 8.7, the output* A*, fulfills* $A \equiv_{\mathcal{S}} \mathcal{M}$*.*

8.6 Infinite equivalence classes

If either $[\![\Delta(\Sigma_\$)]\!]$ of $[\![\![\Sigma^*]\!]\!]$ is infinite, Algorithm 6 is not guaranteed to terminate. Indeed, even in the case that termination is theoretically certain, it may take too long to finish. The learning algorithm could be forced to stop by imposing some kind of bound to its execution.

[32] proposes finishing whenever the hypothesis automaton exceeds a maximum number of states or the string passed to a membership query is longer than a certain length. If the length bound is reached, it may occur that some transitions are not properly defined because the corresponding destination state has not yet been discovered. In such case, the last completely constructed automaton is returned. The size condition is checked after the hypothesis automaton is constructed, therefore it can only happen if the length one did not occur before.

[48] resorts to stopping conditions that depend on the algorithm's data structure, together with an execution time bound. As for length cutoff, when time is exhausted, some transitions may have a missing destination. In this case, their algorithm uses the state whose probability distribution is the closest one, with respect to an appropriate distance.

Here, we adopt the above three stopping criteria. When any of them occurs, an exception is launched and the algorithm proceeds to constructing a hypothesis PDFA. Possibly, a missing destination of a transition labeled σ from a state q could be found during the execution of $\mathit{sift}(\alpha(q)\sigma)$ if there is no outgoing arc $[\![\mathbf{MQ}(\alpha(q)\sigma w)]\!]$ for some inner node w. In normal mode, this situation results in T to be updated with a new leaf and the construction to be restarted. In exception mode, however, state q is connected by σ to a dummy *unknown* state which acts as a sink. A hypothesis with a reachable unknown state is called a *partial* PDFA to stress the fact that the learner would have produced a larger (possibly complete) PDFA provided more resources were given. Actually, by Prop. 8.4(3), if $[\![\Delta(\Sigma_\$)]\!]$ is finite, as it is the case for quantization, *rank* and top_r, for instance, it could be possible to let *build* run until termination, and so produce a PDFA with no *unknown* state. However, this may be too costly.

8.7 Language model equivalence

Indeed, the idea of partial PDFA can be used for other purposes. Consider the partial function f defined as follows: $f(w) \triangleq \mathcal{M}_1(w)$ if $\mathcal{M}_1(w) =_{\mathcal{S}} \mathcal{M}_2(w)$, otherwise $f(w) \triangleq \perp$ (undefined). Clearly, f is total and equal to $\mathcal{M}_1$ if and only if $\mathcal{M}_1 \equiv_{\mathcal{S}} \mathcal{M}_2$. As before, Algorithm 6 can be adapted to produce PDFA with a state for $\perp$. Therefore, learning a PDFA from f will either give the quotient of $\mathcal{M}_1$ and prove the equivalence, or return a partial PDFA having a $\perp$ state and so proving they are not equivalent, or return a partial PDFA with no $\perp$-state which results in an inconclusive verdict.

9 Application to language models

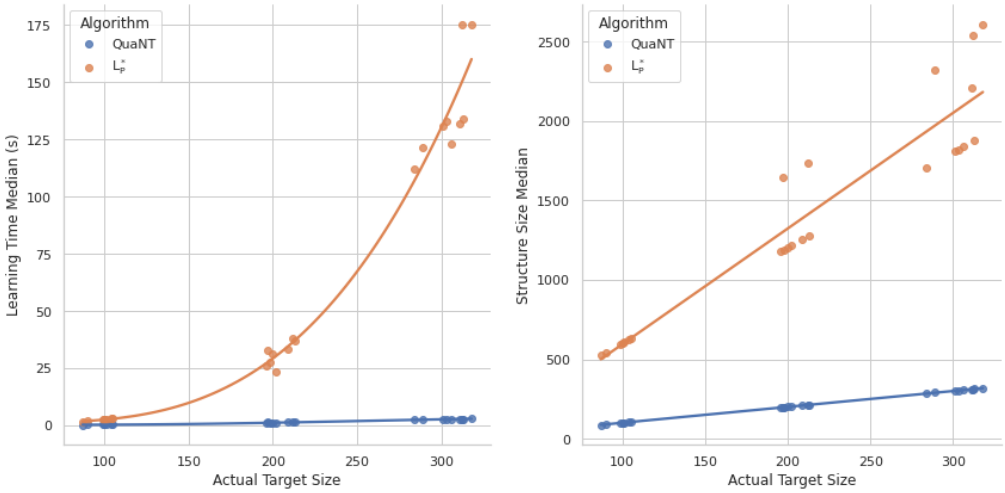
The motivation of this chapter is twofold. First, we asses the performance gain of QNT with respect to observation table based techniques such as $\mathrm{L}^\star_\mathrm{p}$. Second, we study the impact of different equivalence relations on various metrics when learning PDFA from neural language models.

9.1 Synthetically generated PDFA

In order to assess QNT performance, we carry out a series of experiments using $\mathrm{L}^\star_\mathrm{p}$ as a reference. We compared the learning algorithms on randomly generated PDFA. The generation technique works in two steps. First, it constructs random DFA over Σ. Second, DFA are transformed into PDFA by assigning a probability distribution over $\Sigma_\$$ to every state. The first step uses the method described in [62] based on results from [63]. Let n be the desired number of reachable states of a DFA, called its *nominal* size. The method consists in randomly generating DFA of a total of $n \cdot m \cdot \rho_m^{-1}$ possibly unreachable states, for $m = |\Sigma|$, where $\rho_m = m - W_0 \cdot m \cdot e^{-m}$ and W_0 is the Lambert-W function, and then computing its accessible part by a depth-first traversal. It is important to remark that this method does not guarantee the *actual* size of the accessible part to be exactly n, but to be normally distributed around n. To obtain exactly n accessible states, the method could be repeated using a rejection algorithm. However, in practice, this proved to be very inefficient, being almost impossible to generate DFA of accessible size bigger than 100 in reasonable time. All experiments threw perfect scores for all computed metrics (word error rate, normalized discounted cumulative gain, log probability error [48]) for all algorithms on the same test set of strings. Therefore, the analyses of the experimental results are mainly focused on execution time and structure size. For ease of comparison, figures show trend lines. The experiments confirm notable execution time gains achieved by QNT.

9.1.1 Experiment 1

In this experiment we compared QNT and L_p^*. For this, 10 random PDFA over a binary alphabet ($m = 2$) of nominal sizes $n = 100, 200, 300$ were generated, and each algorithm was run 10 times for each PDFA. For QNT $\kappa = 1000$, and for L_p^*, $t = \kappa^{-1}$ (Prop 7.4). Fig. 9.1a shows learning time medians for every actual size. Notably, L_p^* execution time grows much faster than QNT's. Indeed, QNT achieves a speedup of approximately $0.2n$, reaching around 60x for the biggest PDFA (see Fig. 9.3a). This experiment also showed that the size of L_p^*'s observation table grows bigger than QNT's tree which partly explains the gains in execution time (Fig. 9.1b).



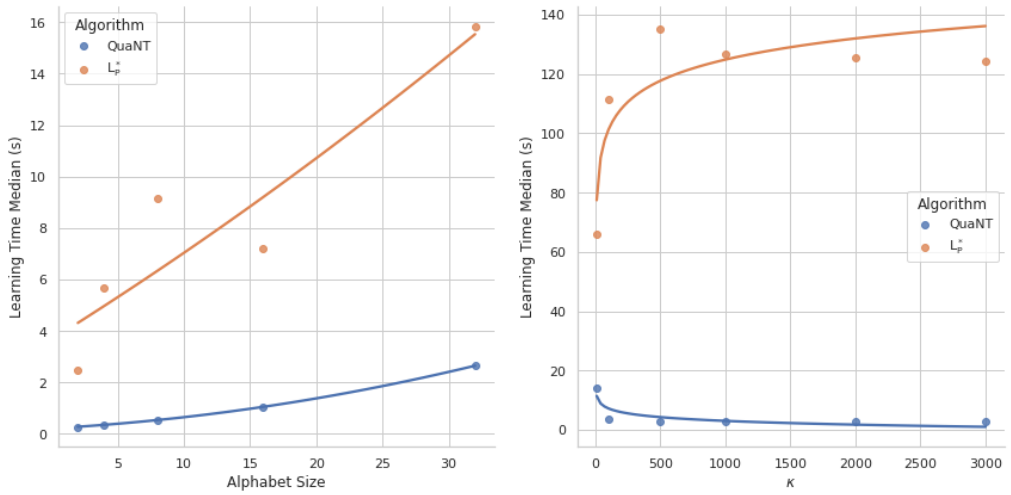
(a) Execution time.

(b) Structure size.

Figure 9.1: Experiment 1

9.1.2 Experiment 2

In this experiment 10 random PDFA of nominal size $n = 100$ were generated for alphabet size $m = 2, 4, 8, 16, 32$. We compared QNT and L_p^* with $\kappa = 1000$ and $t = \kappa^{-1}$. Each algorithm was run 10 times for each PDFA. Fig. 9.2a shows the learning time medians for every alphabet size. As it can be seen L_p^* seems to be more sensitive to the growth in the alphabet size.



(a) Experiment 2.

(b) Experiment 3.

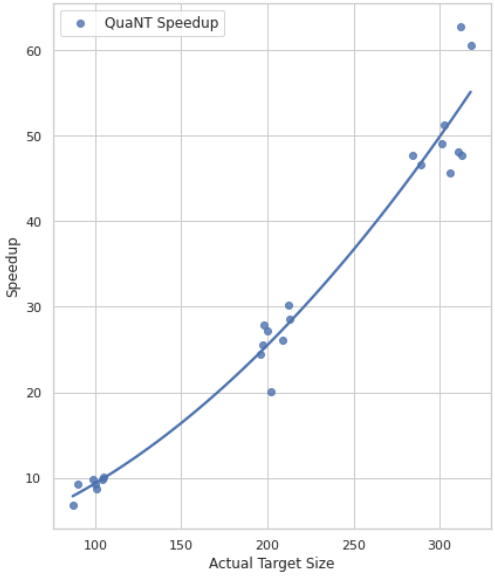
Figure 9.2: Experiments 2 and 3.

9.1.3 Experiment 3

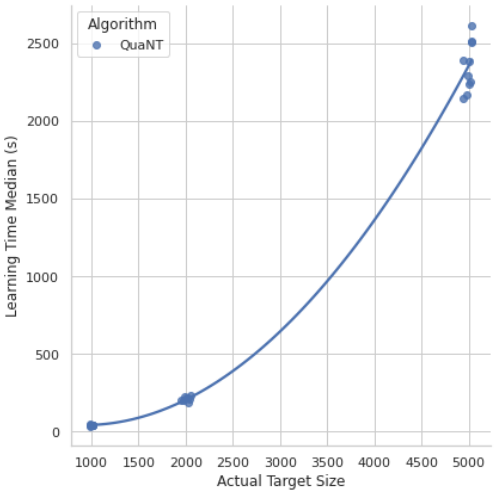
In this experiment, we compare the algorithms for different values of tolerance and quantization parameter: $\kappa = 10, 100, 500, 1000, 2000, 3000$, with $t = \kappa^{-1}$. For every parameter configuration 10 random PDFA of nominal size $n = 300$ and alphabet size $m = 2$ were generated, and each algorithm was run 10 times for each PDFA. Fig. 9.2b shows the median learning times. As it can be seen both algorithms appear to stabilize its execution time after some parameters sizes ($\kappa = 500$, $t = 1/1000$).

9.1.4 Experiment 4

Here, QNT was evaluated on bigger nominal sizes $n = 1000, 2000, 5000$, fixing $\kappa = 1000$ and $m = 2$. For every parameter configuration, 10 random PDFA were generated and each algorithm was run 10 times for each PDFA. Fig. 9.3b shows median learning times. Clearly, QNT still manages to learn PDFA from systems that are intractable for L_p^*. Assuming a linear speedup of $0.2n$ from Experiment 9.1.1, the learning time of a PDFA of size 5000 would be almost a month for L_p^*.



(a) Experiment 1 (Speedup).



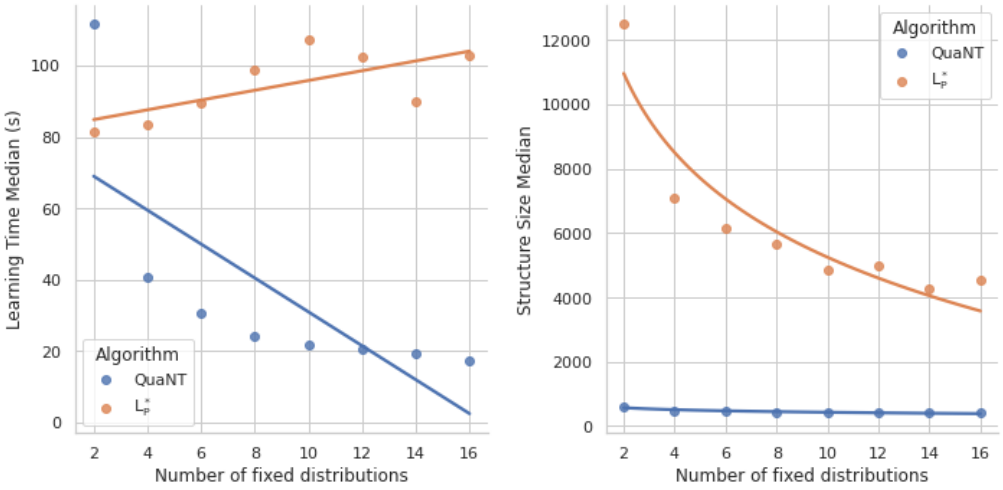
(b) Experiment 4.

Figure 9.3: Experiments 1 (Speedup) and 4.

9.1.5 Experiment 5

In previous experiments it is noted that nearly all states in the randomly generated PDFA have distinct next symbol distributions, that is, most states are distinguished by λ, thus producing shallow tree structures (depth 1 or 2). In order to analyze cases where several states have the same next symbol distribution, the PDFA random generation algorithm is parameterized by the number of different distributions to use, denoted d. Then, the algorithm first randomly generates a set of d different distributions and then labels each state by uniformly picking one distribution in this set.

For this experiment, 10 random PDFA over a binary alphabet ($m = 2$) of nominal size $n = 300$ were generated for different values of d (ranging from 2 to 16). Each algorithm was run 10 times for each PDFA. For QNT, $\kappa = 1000$, and for $\mathrm{L}^*_{\mathrm{p}}$, $t = \kappa^{-1}$ (Prop 7.4). Fig. 9.4a shows learning time medians for every d value. Notably, $\mathrm{L}^*_{\mathrm{p}}$ execution time still grows faster than QNT's. The only case where QNT achieves worse learning time than $\mathrm{L}^*_{\mathrm{p}}$ is for $d = 2$. Otherwise, QNT significantly benefits from the increase in d.



(a) Execution time.

(b) Structure size.

Figure 9.4: Experiment 5.

Regarding structure sizes, L_p^*'s observation table grows bigger than previous experiments, being negatively affected by smaller values of d. However QNT maintains similar sizes to those observed in experiment 1 (Fig. 9.4b).

9.2 Neural language models

We analyze case studies from genetics and cybersecurity application domains, using several equivalences in order to evalute the impact on various metrics, such as execution time, model size, and error, when learning PDFA from neural language models. Those models were trained from publicly available data. Similar to [64], that training data is also used to generate random samples for evaluating **EQ**. Specifically, in order to do this, the oracle uses bootstrap sampling [65] from the given data.

9.2.1 TATA-boxes in DNA promoters

DNA promoter sequences control the activation or repression of genes. It is a sequence of length 6 characterized by totalling more A's and T's than C's and G's. Several works have applied deep learning for classification tasks involving

$\mathcal{S}$	$\lvert Q\rvert$	(d, n)	$\lvert[\![\Delta(\Sigma_\$)]\!]\rvert$	**EQ**	Sec.
$=_{10}$	811	(6, 17)	5×10^4	102	328.0
rank	299	(6, 19)	120	86	130.3
top_1	79	(10, 23)	5	55	65.6
TATA	54	(13, 32)	2	54	71.1

Table 9.1: TATA-Box results

TATA-boxes, e.g., [19]. Here, we use an LSTM-based language model trained on 1400 TATA-boxes of human DNA from EPDnew[1]. Table 9.1 shows the results for top_1, $rank_{,\ =_{10}}$ and a TATA-specific equivalence induced by $P_{\mathcal{M}}[T|w] + P_{\mathcal{M}}[A|w] \geq P_{\mathcal{M}}[G|w] + P_{\mathcal{M}}[C|w]$. PAC parameters are $\varepsilon = \delta = 0.05$. All experiments terminated with a successful **EQ**. Thus, $A \equiv_{\mathcal{S}} \mathcal{M}$ in the PAC sense. Column (d, n) shows the depth d and the number of internal nodes n of T. Column $\lvert[\![\Delta(\Sigma_\$)]\!]\rvert$ gives an approximation of the number of classes defined by $\mathcal{S}$. Note that coarser $\mathcal{S}$ tend to generate smaller Q.

[1] `https://epd.epfl.ch//index.php`

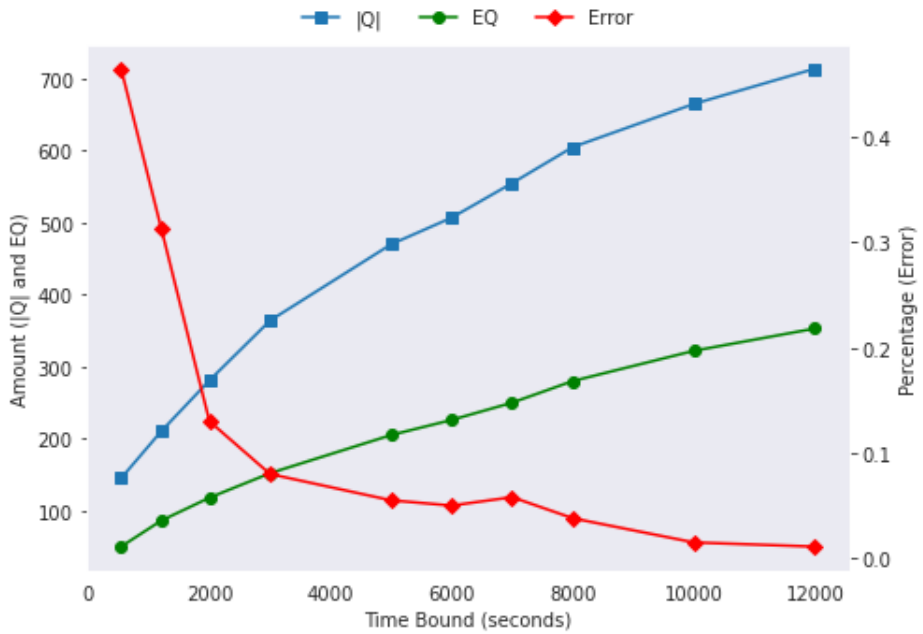


Figure 9.5: HDFS results by time bound

9.2.2 Language model of normal HDFS traces

We trained an LSTM-based language model on a dataset of 4800 normal Hadoop File System (HDFS) logs from [18], which are sequences from a set of 30 symbols including \$.

$\mathcal{S}$	$\lvert Q\rvert$	(d, n)	$\lvert[\![\Delta(\Sigma_\$)]\!]\rvert$	**EQ**	Unk.	Err.
top_3	1616	(9, 56)	4.1×10^3	153	0.00	0.23
$rank_3$	2008	(6, 43)	2.4×10^4	105	0.02	0.39
top_6	2087	(8, 40)	5.9×10^5	87	0.00	0.66
$rank_6$	8837	(2, 1)	4.3×10^8	1	0.91	1.00
$=_3$	660	(14, 148)	1.0×10^{14}	326	0.00	0.03
$=_5$	3014	(6, 20)	4.7×10^{20}	78	0.01	0.80
$=_{10}$	7289	(2, 1)	5.0×10^{29}	1	0.01	0.62

Table 9.2: HDFS results

There, the language model is used to classify logs: if there is a prefix for which its next symbol is not one of the top_r symbols, the log is consider abnormal. Otherwise, it is classified as normal.

Table 9.2 shows the results for *rank*, *top* and $=_\kappa$ for several values of r and κ, with parameters $\varepsilon = \delta = 0.05$. All experiments exhausted a time bound of 10K seconds and generated partial PDFA. To assess the error of the outputs, we sampled a test set of size 1K from available data. "Unk" shows the part of the error corresponding to the fraction of logs in the test set for which A reaches the *unknown* state. This error is included in the total test error shown in column "Err". We observe that coarser equivalences generated automata with fewer states and, generally, smaller total errors. It is of interest to analyze how $|Q|$, **EQ**, "Unk" and "Err" evolve as the execution time augments. The algorithm was run for equivalence $=_3$ with increasing time bounds ranging from 120 to 12K seconds and measure them. In these experiments, "Unk" was always 0. The other obtained results are depicted in Fig. 9.5. The curves show that $|Q|$ (as stated in Prop. 8.8) monotonically increases while "Err" decreases, even if there are some spikes.

9.2.3 Detection of malicious web requests

Here, we study the equivalence between two character-level Transformer language models presented in [66], pre-trained with malicious web requests from [67]. Σ consists of 256 ASCII characters. $\mathcal{M}_1$ has $27K$ parameters and $\mathcal{M}_2$ has $6.5M$ but both have an accuracy greater than 0.97 for predicting the next character given the last 10 observed ones. We ran the algorithm for 60 seconds with the goal of learning whether they were equivalent for top_1.

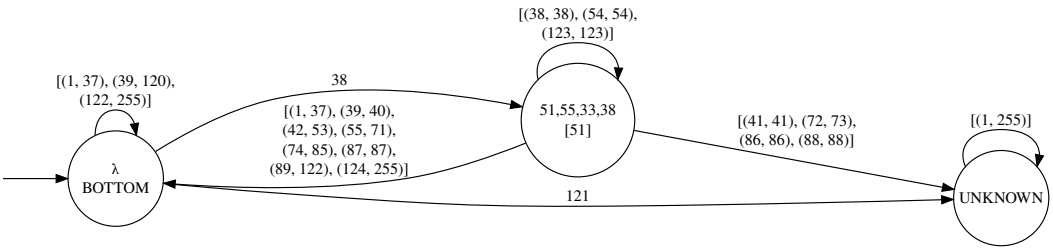


Figure 9.6: Automaton showing both Transformers are not top_1-equivalent

Fig. 9.6 depicts a simplified visualization of the returned PDFA where probability distributions are omitted and multiple transitions with same end points are collapsed into a single one labeled with a set of intervals of symbols. It shows that both Transformers are not top_1-equivalent since $\mathcal{M}_1(\lambda) \neq_{top_1} \mathcal{M}_2(\lambda)$. However, they agree in the class of access string $51, 55, 33, 38$ corresponding to 51 as the most probable symbol. It is worth noting that attempting to learn $\mathcal{M}_1$ and $\mathcal{M}_2$ resulted in high Unk and Err even after a 12K secs.

10 Related Work and Conclusions

10.1 Related work

This thesis presents a general framework for learning regular models from black-box systems with applications to the analysis and verification of neural language acceptors and neural language models.

Verifying properties over sequence processing neural networks has previously been addressed in the literature. For binary predictors, several works have tackled this issue through a *model, then verify* approach by learning some kind of automata that approximates the formal language defined by the network and is amenable for automated verification. Literature for the general problem of learning DFA and other models from RNNs is thoroughly analyzed in the related work section of my Master's Thesis [33], and a recent survey has been pulished by Bollig et al. [68]. A post-learning approach for adversarial accuracy verification is presented in [69] based a white-box rule-extraction technique to extract DFA from RNN. Experimental evaluation is carried out on Tomita grammars [54], which are all regular languages over the $\{0, 1\}$-alphabet. That approach does not offer any guarantee on how well the DFA approximates the RNN. In [70] white-box RNN verification is done by generating a series of abstractions. Specifically, the method strongly relies on the internal structure and weights of the RNN to generate a feed-forward network (FFNN), which is proven to compute the same output. Then, reachability analysis is performed resorting to Linear Programming (LP) and Satisfiability Modulo Theories (SMT) techniques.

If we look into the black-box approaches, the work in [28, 29] presents a post-learning technique that relies on active automata learning. This technique is called property-directed-verification, as the model extraction is intertwined with the checking process. Differently to this thesis, the approach to resolve **EQ** is based on Hoeffding's inequality bound [71] instead of PAC, and the set of properties $\mathcal{H}$ is restricted to DFA. It is important to remark that, at the time of carrying out

the experiments presented in Chapter 5, which appeared in [26, 27], there were no other available tools specifically devoted to verifying neural acceptors in a black-box setting, because property-directed-verification [28] was published afterwards.

A closely related but different area is statistical model checking (SMC), where the system under analysis and/or the property is stochastic [72, 73]. The objective of SMC is to check whether a stochastic system, such as a Markov decision process, satisfies a property with a probability greater or equal to a certain threshold θ. The problem we address in this thesis is different as neither the system nor the property is stochastic. Our approach provides statistical guarantees that the language of an RNN C is included in another language (the property ψ) or provides a PAC model of the language $C \cap \overline{\psi}$, along with actual counterexamples showing it is not.

For general neural language models, the literature on MAT learning automata is more scarce. The algorithms proposed in [74, 75] rely on **MQ** that compute the probability of a string and on an observation table to store the results. In [76], **MQ** return state distributions, that is, the probability that the target probabilistic automaton enters a state after reading an input string. This requires knowing the number of states of the target in advance. These works focused on theoretical results without implementations of them being publicly available. In [77], learning PDFA is a building block of an assume-guarantee framework for verification of probabilistic systems. **MQ** asks for the probability of accepting a string and results are stored in an observation table. However, the overall goal is not to learn a PDFA equivalent to a hidden target but only an appropriate assumption for doing a compositional proof of correctness. To achieve this, the algorithm uses an **EQ** that relies on language inclusion of PDFA and probabilistic model-checking. All these works rely on exact equality of **MQ** outcomes.

To deal with noise in distributions, WL* [48] proposes a non-equivalence similarity relation between probability distributions and develops an algorithm to learn similar hypotheses according to it. Note that WL* uses clustering to group responses to **MQ** stored in the table. WL* extends L* to learn a probabilistic deterministic finite automaton (PDFA) [58] through a non-transitive similarity relation between next-symbol probability distributions. It is important to remark that it was too costly to evaluate WL* outside the experiments presented in [48] with the publicly available code, since it would have required rewriting much of the code base without documentation. As a matter of fact, this motivated the implementation of L$^*_\text{p}$ inside the Neural-Checker [11] tool. Besides, [48] does not provide a proof of minimality nor termination of WL*, and as it is shown in Section 7.2 whether such properties hold is not at all clear.

Other recent related work is the one briefly presented in [78], which adapted

L^* and KV [60] algorithms to directly learn the probability of a string. That is, rather than considering a language model as function from Σ^* to $\Delta(\Sigma_\$)$, they view it as a function from Σ^* to $[0, 1]$. Indeed, this makes a significant difference with the approach developed in this thesis.

It is worth mentioning related work in the context of passive learning, in particular the so-called spectral learning techniques applied to weighted automata, a class of models that includes PDFA. Recently, [79,80] proved the relationship one-to-one between second-order recurrent neural networks and weighted automata, this result has deep implications in the expressivity and training of such models. Besides, [81] apply spectral learning to recurrent neural networks in a black-box setting.

10.2 Conclusions

The framework of this thesis extends initial work from [32] where an active PAC-learning approach for learning regular models that are approximately correct with respect to neural networks is presented. Our algorithms are extensions of Angluin's L^* and other active learning algorithms where a bound on the complexity of the proposed models or computing budget is set to guarantee termination in application domains where the language to be learned may not be a regular one. We also studied the error and confidence of the hypotheses obtained when the algorithm stops by reaching a complexity bound. Chapters 4 and 5 explored the problem of checking properties of neural acceptors devoted to sequence classification over symbolic alphabets in a black-box setting. The approach is not restricted to any particular class of neural network or property. Besides it is on-the-fly because it does not construct a model of the RNN on which the property is verified. The key idea is to express the verification problem on an RNN C as a formula $\Psi(C)$ such that its language is empty if and only if C does not satisfy the requirement and apply a PAC-learning algorithm for learning $\Psi(C)$. On one hand, if the resulting DFA is empty, the algorithm provides PAC-guarantees about the language $\Psi(C)$ being itself empty. On the other, if the output DFA is not empty, it provides an actual sequence of C that belongs to $\Psi(C)$. Besides, the DFA itself serves as an approximate characterization of the set of all sequences in $\Psi(C)$. For instance, our method can be used to verify whether an RNN C satisfies a linear-time temporal property P by checking $C \cap \overline{P}$. Since the approach does not require computing the complement, it can also be applied to verify nonregular properties expressed, for instance, as context-free grammars, and to check equivalence between RNN.

On-the-fly checking through learning has several advantages with respect to

post-learning verification. When the learnt language that approximates $\Psi(C)$ is nonempty, the algorithm provides true evidence of the failure by means of concrete counterexamples. In addition, the algorithm outputs an interpretable characterization of an approximation of the set of incorrect behaviors. Besides, it allows checking properties, with PAC guarantees, for which no decision procedure exists. Moreover, the experimental results on a number of case studies from different application domains provide empirical evidence that the on-the-fly approach typically outperforms post-learning verification if the requirement is probably approximately satisfied. Last but not least, this work also improved theoretical results regarding the probabilistic guarantees of Bounded-L* [32], namely, Theorem 1 provides an upper bound of the error incurred by *any* DFA returned by the algorithm.

On the other hand, Chapters 6, 7, 8 and 9, address the problem of learning quotient language models modulo arbitrary equivalence relations on distributions. Similar to Myhill-Nerode congruence for regular languages, for any equivalence on distributions $\mathcal{S}$ a language model induces a congruence over strings, and leads to correspondences between language models and automata, parameterized by $\mathcal{S}$. This leads to quotient models that abstract away details which are unnecessary for the purpose of the analysis. The size of such models is likely to be smaller than a full fledged one. Besides, we developed a learning algorithm which uses a tree of finite but unbounded width to learn an irreducible PDFA which is PAC-equivalent to the target language model. The algorithm is guaranteed to terminate whenever the congruence induced by the language model for the given equivalence on distributions is finite. We evaluated the techniques on several case studies from cybersecurity and genetics, including pre-trained neural networks, synthetically generated regular models and other non regular models. The experimental results showcased that, despite the complexity of some of the target neural models, learning regular approximations is feasible in the setting of binary acceptors and language models, however when the target system is not regular, the number of states to be learned can be unbounded. At the same time, parameterizing the learning process would help in some cases to overcome the so-called state explosion problem when building state-based models, by choosing appropriate equivalence relations that lead to property-preserving abstractions. Moreover, if the state-space still remains too large, learning can be used to directly verify properties such as the equivalence between language models or the fulfillment of safety properties in binary acceptors.

All of the techniques presented in this work were implemented and evaluated using Neural-Checker [11], a state of the art tool that is part of the results of this thesis. An in detail depiction of the tool is presented in Appendix 12.

Future Research Directions

Given the recent arise of Large Language Models (LLMs) a pertinent research direction is the analysis of such systems under the approaches presented in this thesis. That is, the application of active learning techniques to models containing a large number of parameters and alphabet size. Another interesting direction for future work is to extend our approach to learning more complex hypothesis, such as push-down automata, and their probabilistic extensions. This is pertinent as the complexity and expressivity of certain neural language models has been shown to surpass regular languages [30, 82]. An example of use case for such power is the analysis of code-generation language models.

We expect to continue this research in the context of the recently started project "Tools for the verification of industry-level applications of large language models" (FMV_1_2023_1_175864), in collaboration with two software companies. Its goal is to develop tools that contribute to the responsible implementation and use of industry-level applications of generative artificial intelligence systems that integrate LLM, based on formal languages. The purpose of such tools is to help controlling the LLM in order to make it behave as specified, such as to avoid biases (of any kind) and inappropriate or out-of-context words.

Acknowledgments

This work has been partially funded by Universidad ORT Uruguay and ANII Agencia Nacional de Investigación e Innovación grants FSDA_1_2018_1_154419, FMV_1_2019_1_155913, IA_1_2022_1_173516, FMV_1_2023_1_175864.

11 Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, [Online]. Available: http://www.deeplearningbook.org.

[2] J. Degrave, F. Felici, and et al., "Magnetic control of tokamak plasmas through deep reinforcement learning," *Nature*, vol. 602, pp. 414–419, 02 2022.

[3] J. Kocić, N. Jovičić, and V. Drndarević, "An end-to-end deep neural network for autonomous driving designed for embedded automotive platforms," *Sensors*, vol. 19, no. 9, p. 2064, 2019.

[4] M. Rahman, Y. Cao, and et al., "Deep pre-trained networks as a feature extractor with XGBoost to detect tuberculosis from chest X-ray," *Comp. Elec. Eng.*, vol. 93, p. 107252, 2021.

[5] S. A. Seshia, D. Sadigh, and S. Sastry, "Toward verified artificial intelligence," *Communications of the ACM*, vol. 65, no. 7, pp. 46–55, july 2022.

[6] H. Jacobsson, "Rule extraction from recurrent neural networks: A taxonomy and review," *Neural Computation*, vol. 17, pp. 1223–1263, 06 2005.

[7] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.

[8] V. Dignum, "Responsible artificial intelligence — from principles to practice: A keynote at thewebconf 2022," *ACM SIGIR Forum*, vol. 56, pp. 1–6, 01 2023.

[9] D. Gunning, "Darpa's explainable artificial intelligence (xai) program," in *Proceedings of the 24th International Conference on Intelligent User Interfaces*, ser. IUI '19. New York, NY, USA: ACM, 2019, pp. ii–ii. [Online]. Available: http://doi.acm.org/10.1145/3301275.3308446

[10] J. M. Wing, "Trustworthy ai," *Commun. ACM*, vol. 64, no. 10, p. 64–71, sep 2021. [Online]. Available: https://doi.org/10.1145/3448248

[11] F. Mayr, S. Yovine, F. Pan, and F. Vilensky, "Neural checker," https://github.com/orgs/neuralchecker/, 2021.

[12] T. Lei, R. Barzilay, and T. Jaakkola, "Rationalizing neural predictions," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

[13] R. Guidotti, A. Monreale, F. Turini, D. Pedreschi, and F. Giannotti, "A survey of methods for explaining black box models," *CoRR*, vol. abs/1802.01933, 2018.

[14] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, "Explaining explanations: An approach to evaluating interpretability of machine learning," *CoRR*, vol. abs/1806.00069, 2018.

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," in *Neurocomputing: Foundations of Research*, J. A. Anderson and E. Rosenfeld, Eds. Cambridge, MA, USA: MIT Press, 1988, pp. 696–699. [Online]. Available: http://dl.acm.org/citation.cfm?id=65669.104451

[16] J. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, pp. 179–211, 03 1990.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6000–6010.

[18] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *SIGSAC CCS*. ACM, 2017, p. 1285–1298.

[19] M. Oubounyt, Z. Louadi, H. Tayara, and K.-T. Chong, "Deepromoter: Robust promoter predictor using deep learning," *Frontiers in genetics*, vol. 10, 2019.

[20] L. Arras, G. Montavon, K. Müller, and W. Samek, "Explaining recurrent neural network predictions in sentiment analysis," in *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, WASSA@EMNLP 2017, Copenhagen, Denmark, September 8, 2017*, A. Balahur, S. M. Mohammad, and E. van der Goot, Eds. Association for Computational Linguistics, 2017, pp. 159–168. [Online]. Available: https://doi.org/10.18653/v1/w17-5221

[21] D. H. Park and R. Chiba, "A neural language model for query auto-completion," in *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1189–1192. [Online]. Available: https://doi.org/10.1145/3077136.3080758

[22] T. van Ede, H. Aghakhani, N. Spahn, R. Bortolameotti, M. Cova, A. Continella, M. van Steen, A. Peter, C. Kruegel, and G. Vigna, "DeepCASE: Semi-Supervised Contextual Analysis of Security Events," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2022.

[23] C. de la Higuera, *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.

[24] K. Murphy, "Passively learning finite automata," Santa Fe Institute, Tech. Rep. 96-04-017, 1996.

[25] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, Nov. 1987.

[26] F. Mayr, R. Visca, and S. Yovine, "On-the-fly black-box probably approximately correct checking of recurrent neural networks," in *Machine Learning and Knowledge Extraction - 4th IFIP TC 5, TC 12, WG 8.4, WG 8.9, WG 12.9 International Cross-Domain Conference, CD-MAKE 2020, Dublin, Ireland, August 25-28, 2020, Proceedings*, ser. Lecture Notes in Computer Science, A. Holzinger, P. Kieseberg, A. M. Tjoa, and E. R. Weippl, Eds., vol. 12279. Springer, 2020, pp. 343–363. [Online]. Available: https://doi.org/10.1007/978-3-030-57321-8_19

[27] F. Mayr, S. Yovine, and R. Visca, "Property checking with interpretable error characterization for recurrent neural networks," *Machine Learning and Knowledge Extraction*, vol. 3, no. 1, pp. 205–227, 2021. [Online]. Available: https://www.mdpi.com/2504-4990/3/1/10

[28] I. Khmelnitsky, D. Neider, R. Roy, X. Xie, B. Barbot, B. Bollig, A. Finkel, S. Haddad, M. Leucker, and L. Ye, "Property-directed verification and robustness certification of recurrent neural networks," in *Automated Technology for Verification and Analysis*, Z. Hou and V. Ganesh, Eds. Cham: Springer International Publishing, 2021, pp. 364–380.

[29] ——, "Analysis of recurrent neural networks via property-directed verification of surrogate models," *Int. J. Softw. Tools Technol. Transf.*, vol. 25, no. 3, pp. 341–354, 2023. [Online]. Available: https://doi.org/10.1007/s10009-022-00684-w

[30] J. Pérez, P. Barceló, and J. Marinkovic, "Attention is turing-complete," *Journal of Machine Learning Research*, vol. 22, no. 75, pp. 1–35, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-302.html

[31] L. G. Valiant, "A theory of the learnable," *Commun. ACM*, vol. 27, no. 11, pp. 1134–1142, Nov. 1984.

[32] F. Mayr and S. Yovine, "Regular inference on artificial neuralnetworks," in *Machine Learning and Knowledge Extraction*, A. Holzinger *et al.*, Eds. Cham: Springer International Publishing, 2018, pp. 350–369.

[33] F. Mayr, "Regular inference over recurrent neural networks as a method for black box explainability," Master's thesis, Montevideo Uruguay, 2019.

[34] C. Zhou, B. Cule, and B. Goethals, "Pattern based sequence classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 5, pp. 1285–1298, May 2016.

[35] Z. Xing, J. Pei, and E. Keogh, "A brief survey on sequence classification," *SIGKDD Explor. Newsl.*, vol. 12, no. 1, pp. 40–48, Nov. 2010.

[36] F. Mayr, S. Yovine, F. Pan, N. Basset, and T. Dang, "Towards efficient active learning of PDFA," in *LearnAut 2022*, 2022.

[37] F. Mayr, S. Yovine, M. Carrasco, F. Pan, and F. Vilensky, "A congruence-based approach to active automata learning from neural language models," in *Proceedings of 16th edition of the International Conference on Grammatical Inference*, ser. Proceedings of Machine Learning Research, F. Coste, F. Ouardi, and G. Rabusseau, Eds., vol. 217. PMLR, 10–13 Jul 2023, pp. 250–264. [Online]. Available: https://proceedings.mlr.press/v217/mayr23a.html

[38] F. Mayr, S. Yovine, M. Carrasco, A. Garat, M. Iturbide, J. da Silva, and F. Vilensky, "Results of neural-checker toolbox in taysir 2023 competition," in *Proceedings of 16th edition of the International Conference on Grammatical Inference*, ser. Proceedings of Machine Learning Research, F. Coste, F. Ouardi, and G. Rabusseau, Eds., vol. 217. PMLR, 10–13 Jul 2023, pp. 295–298. [Online]. Available: https://proceedings.mlr.press/v217/mayr23b.html

[39] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006.

[40] N. Chomsky, "Three models for the description of language," *IRE Transactions on Information Theory*, vol. 2, no. 3, pp. 113–124, Sep. 1956.

[41] O. Maler and A. Pnueli, "On the learnability of infinitary regular sets," *Inf. Comput.*, vol. 118, no. 2, p. 316–326, May 1995. [Online]. Available: https://doi.org/10.1006/inco.1995.1070

[42] B. Bollig, P. Habermehl, C. Kern, and M. Leucker, "Angluin-style learning of nfa," in *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, ser. IJCAI'09. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 1004–1009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1661445.1661605

[43] M. Kearns and U. Vazirani, "An introduction to computational learning theory," 1994.

[44] R. Rivest and R. Schapire, "Inference of finite automata using homing sequences," *Information and Computation*, vol. 103, no. 2, pp. 299–347, 1993. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0890540183710217

[45] F. Howar, "Active learning of interface programs," Ph.D. dissertation, 06 2012.

[46] M. Isberner, F. Howar, and B. Steffen, "The TTT algorithm: A redundancy-free approach to active automata learning," in *RV'14*. LNCS 8734, 2014, pp. 307–322.

[47] D. Angluin, "Computational learning theory: Survey and selected bibliography," in *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '92. New York, NY, USA: ACM, 1992, pp. 351–369.

[48] G. Weiss, Y. Goldberg, and E. Yahav, "Learning deterministic weighted automata with queries and counterexamples," in *Adv. in Neural Information Proc. Sys.*, vol. 32, 2019.

[49] S. Ben-David and S. Shalev-Shwartz, *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[50] D. Angluin, "Queries and concept learning," *Mach. Learn.*, vol. 2, no. 4, p. 319–342, apr 1988. [Online]. Available: https://doi.org/10.1023/A:1022821128753

[51] G. Weiss, Y. Goldberg, and E. Yahav, "Learning deterministic weighted automata with queries and counterexamples," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/d3f93e7766e8e1b7ef66dfdd9a8be93b-Paper.pdf

[52] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[53] Y. Hao, W. Merrill, D. Angluin, R. Frank, N. Amsel, A. Benz, and S. Mendelsohn, "Context-free transductions with neural stacks," *preprint arXiv:1809.02836*, 2018.

[54] M. Tomita, "Dynamic construction of finite automata from examples using hill-climbing," in *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, Ann Arbor, Michigan, 1982, pp. 105–108.

[55] K. Meinke and M. A. Sindhu, "LBTest: A learning-based testing tool for reactive systems," in *STVV*. IEEE, March 2013, pp. 447–454.

[56] M. Merten, "Active automata learning for real life applications," Ph.D. dissertation, Technischen Universität Dortmund, 2013.

[57] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *J. Mach. Learn. Res.*, vol. 3, no. null, p. 1137–1155, Mar. 2003.

[58] A. Clark and F. Thollard, "PAC-learnability of probabilistic deterministic finite state automata," *J. Machine Learning Research*, vol. 5, pp. 473–497, 2004.

[59] J. E. Hopcroft and R. M. Karp, "A linear algorithm for testing equivalence of finite automata." 1971.

[60] M. Kearns and U. Vazirani, *An Introduction to Computational Learning Theory*. Cambridge, MA, USA: MIT Press, 1994.

[61] M. Isberner, "Found. active automata learning: An algorithmic persp." Ph.D. dissertation, 2015.

[62] C. Nicaud, "Random deterministic automata," in *MFCS'14*. LNCS 8634, 2014, pp. 5–23.

[63] A. Carayol and C. Nicaud, "Distribution of the number of accessible states in a random deterministic automaton," *Leibniz Int. Proc. in Informatics*, vol. 14, pp. 194–205, 2012.

[64] M. Craven and J. Shavlik, "Extracting tree-structured representations of trained networks," in *NIPS'95*. Cambridge, MA, USA: MIT Press, 1995, p. 24–30.

[65] B. Efron and R. Tibshirani, *An Introduction to the Bootstrap*, ser. Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, 1993, no. 57.

[66] N. Martínez, "Comparison of lstm and transformer neural network on multiple approaches for weblogs attack detection," Master's thesis, Montevideo Uruguay, 2022.

[67] J. Li, H. Zhang, and Z. Wei, "The weighted word2vec paragraph vectors for anomaly detection over http traffic," *IEEE Access*, vol. 8, pp. 141 787–141 798, 2020.

[68] B. Bollig, M. Leucker, and D. Neider, "A survey of model learning techniques for recurrent neural networks," *A Journey from Process Algebra via Timed Automata to Model Learning: Essays Dedicated to Frits Vaandrager on the Occasion of His 60th Birthday*, pp. 81–97, 2022.

[69] Q. Wang, K. Zhang, X. Liu, and C. L. Giles, "Verification of recurrent neural networks through rule extraction," in *AAAI Spring Symposium on Verification of Neural Networks (VNN19)*, 2019.

[70] A. Kevorchian, "Verification of recurrent neural networks," Master's thesis, Imperial College London, 2018.

[71] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 13–30, 1963. [Online]. Available: http://www.jstor.org/stable/2282952

[72] A. Legay, A. Lukina, L. M. Traonouez, J. Yang, S. A. Smolka, and R. Grosu, *Statistical Model Checking*. Cham: Springer, 2019, pp. 478–504.

[73] G. Agha and K. Palmskog, "A survey of statistical model checking," *ACM Trans. Model. Comput. Simul.*, vol. 28, no. 1, Jan. 2018.

[74] A. Kaznatcheev and P. Panangaden, "Weighted automata are compact and actively learnable," *Inf. Process. Lett.*, vol. 171, pp. 106–133, 2021.

[75] A. Beimel, F. Bergadano, N. Bshouty, E. Kushilevitz, and S. Varricchio, "Learning functions represented as multiplicity automata," *Journal of the ACM*, vol. 47, no. 3, p. 506–530, 2000.

[76] W. Tzeng, "Learning probabilistic automata and markov chains via queries," *Machine Learning*, vol. 8, pp. 151–166, 1992.

[77] L. Feng, T. Han, M. Kwiatkowska, and D. Parker, "Learning-based compositional verification for synchronous probabilistic systems," in *ATVA'11*. LNCS 6996, 2011, pp. 511–521.

[78] E. Muškardin, M. Tappler, and B. K. Aichernig, "Testing-based black-box extraction of simple models from rnns and transformers," in *Proceedings of 16th edition of the International Conference on Grammatical Inference*, ser. Proceedings of Machine Learning Research, F. Coste, F. Ouardi, and G. Rabusseau, Eds., vol. 217. PMLR, 10–13 Jul 2023, pp. 291–294. [Online]. Available: https://proceedings.mlr.press/v217/muskardin23a.html

[79] G. Rabusseau, T. Li, and D. Precup, "Connecting weighted automata and recurrent neural networks through spectral learning," in *Proceedings of Machine Learning Research*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and M. Sugiyama, Eds., vol. 89. PMLR, 16–18 Apr 2019, pp. 1630–1639. [Online]. Available: http://proceedings.mlr.press/v89/rabusseau19a.html

[80] T. Li, D. Precup, and G. Rabusseau, "Connecting weighted automata, tensor networks and recurrent neural networks through spectral learning," *CoRR*, vol. abs/2010.10029, 2020. [Online]. Available: https://arxiv.org/abs/2010.10029

[81] S. Ayache, R. Eyraud, and N. Goudian, "Explaining black boxes on sequential data using weighted automata," in *Proceedings of the 14th International Conference on Grammatical Inference, ICGI 2018, Wrocław, Poland, September 5-7, 2018*, ser. Proceedings of Machine Learning Research, O. Unold, W. Dyrka, and W. Wieczorek, Eds., vol. 93. PMLR, 2018, pp. 81–103. [Online]. Available: http://proceedings.mlr.press/v93/ayache19a.html

[82] G. Weiss, Y. Goldberg, and E. Yahav, "On the practical computational power of finite precision RNNs for language recognition," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, I. Gurevych and Y. Miyao, Eds. Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 740–745. [Online]. Available: https://aclanthology.org/P18-2117

[83] R. Eyraud, D. Lambert, B. Tahri Joutei, A. Gaffarov, M. Cabanne, J. Heinz, and C. Shibata, "Taysir competition: Transformer+RNN: Algorithms to yield simple and interpretable representations," in *Proceedings of 16th edition of the International Conference on Grammatical Inference*, ser. Proceedings of Machine Learning Research, F. Coste, F. Ouardi, and G. Rabusseau, Eds., vol. 217. PMLR, 10–13 Jul 2023, pp. 275–290. [Online]. Available: https://proceedings.mlr.press/v217/eyraud23a.html

12 Appendix

Neural Checker Tool

All techniques presented in this work were implemented in the Neural-Checker [11] toolbox, which was used to carry out experiments and case studies.

Neural-Checker is a repository that provides two libraries, namely Pythautomata and PyModelExtractor. Pythautomata's main goal is to provide implementations for the structures needed for working in the Model Extraction Framework (PyModelExtractor). PyModelExtractor's main goal is to enable the explainability and checking of complex systems in a black box context through the use of active learning techniques.

This toolbox was used in the TAYSIR Competition 2023 [83] and its results are presented in [38], achieving the second place in the competition. TAYSIR is an on-line competition about model inference form Neural Networks (NN). The 2023 challenge was divided in two tracks. The goal of track 1 was to learn language acceptors, while track 2 was focusing on language models. The underlying neural architectures were RNN and Transformers.

Tool Structure

The tool is structured in two libraries as it can be seen in Fig. 12.1.

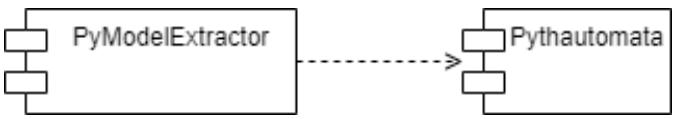


Figure 12.1: Component diagram

Pythautomata

Pythautomata presents a number of packages split by responsibilities as we can see in Fig. 12.2, which only presents the core packages for simplicity.

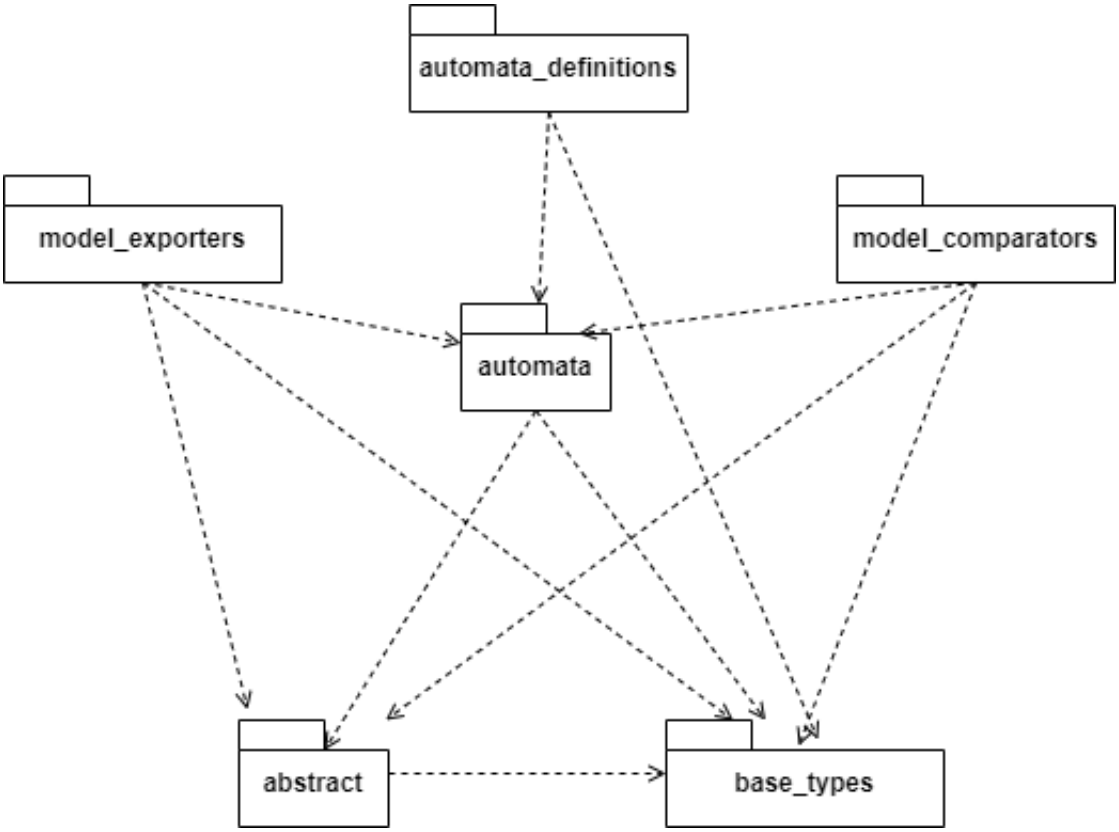


Figure 12.2: Pythautomata's package diagram

base_types implements the basic language theory concepts. These includes the concept of alphabet, sequence, symbol, and variants.

abstract package presents the *FiniteAutomaton* abstract class, wich provides a common interface to be used by every type of finite automaton.

automata package implements concrete automata types (i.e. DFA or PDFA). These automata contain an initial state and a set of states.

automata_definitions package presents some classic examples of automata. For example Tomita's automata [54] are defined inside this package.

model_comparators This package contains comparison strategies, like adaptions of Hopkroft-Karp [59] algorithm and probabilistic approaches like PAC equivalence [31].

pyModelExtractor

This project presents a simpler package structure as we can see in Fig. 12.3.

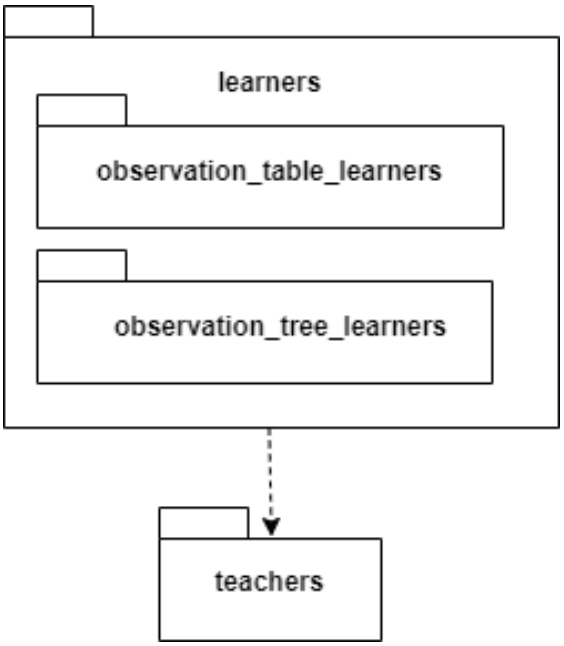


Figure 12.3: pyModelExtractor package diagram

teachers package contains the definition for the MAT interface to be fulfilled by the different teachers. The teachers must implement both **MQ** and **EQ**. Core classes present in this package are PAC teachers, and exact ones that rely on ***model_comparators*** from pythautomata.

learners package contains the abstract class *Learner* which defines the interface all learners must comply with. Besides this, there are two other packages ***observation_table_learners*** and ***observation_tree_learners***, each having a family of learning methods. Bounded-L*and QNT are implemented inside the learners package.