Bachelor's Thesis Computing Science

Radboud University Nijmegen

Learning Mealy Intersections

Author:
Julian Put
s1062379

First supervisor/assessor:
dr. Jurriaan Rot

Second assessor:
dr. Sebastian Junges

May 19, 2024

Abstract

A branch of automata theory is active automata learning, where models for systems are created through interactions and observations. Two types of systems learned using active automata learning are Mealy machines [11] and learning products of DFAs [8]. In this thesis, we introduce learning Mealy intersections. First we define what Mealy intersections are. Second, we provide strategies to learn Mealy intersections. Finally, we apply our theory on randomly generated automata and benchmarks implementing common ISO standards.

Contents

Chapter 1

Introduction

Automata provide a mathematical model of everyday responsive and interactive systems, like traffic lights, cryptographic protocols or coffee machines.

The main goal of Automata learning is creating models of black-box systems. Within this field, a distinction is made between active and passive learning, where interactions with the system either is or is not possible.

In passive automata learning, we are given data about the inputs and corresponding outputs of an automata, and are tasked to create our own automata with the same interactions. In active automata learning, we are tasked to create a model for an automaton. Instead of receiving some data about the automaton, we can interact with it ourselves and thus create our own data. The most common form of active automata learning is the Angluin style automata learning with the L^* algorithm [2]. Since its release in 1987, it has become the standard for active automata learning.

These learning methods can be applied in real-world instances model checking, bug finding, minimizing code or machines, and many more [5].

This theory on learning automata can be extended by shifting our focus from single automata to intersections of automata. Such an intersection of automata has its relevance when finding interactions, like obtaining coffee or sounding an alarm, which can be used in multiple environments with different machines. A stub of this theory exists, where we learn the common interactions of multiple Deterministic Finite Automata [8].

In the real world, we have systems modelled by DFAs alongside systems modelled by other types of automata. One such automaton is a Mealy machine, which has interactions between the system and a user. For example, a coffee machine filling cups with different types of coffee, depending on which flashing buttons we pressed.

With the current theory, we can partially look at the intersections of such interactive systems, by modelling each individual system as accepting a single response. However, we cannot learn the common interactions over all the possible responses from a coffee machine, because this would not be

DFAs, instead, it is a Mealy machine.

In this thesis, we seek to bridge this gap. Our first contribution is the definition of intersection applied Mealy machines. Then we describe orthogonal strategies to learn intersections and empirically show their strengths and weaknesses.

Our thesis is structured as follows. In Chapter 2, we introduce the theory of intersection over DFAs and automata learning. In Chapter 3, we define languages of Mealy machines and Mealy language intersections, products of Mealy machines and strategies to learn intersections. In Chapter 4, we use our implementation of the strategies to learn the intersection of both randomly generated mealy machines and benchmarks. In Chapter 5, we briefly discuss where our research is placed in automata theory, and specifically active automata learning. In Chapter 6, we conclude our thesis and look for future research.

Chapter 2

DFA intersections

In this section, we discuss intersections and automata learning for the case of DFAs. First, we show a definition of products over DFAs. Second, we show a learning style to learn automata.

2.1 Product on DFAs

Before defining a product on DFAs, we first show the definition of a DFA and its language.

Definition 2.1 (DFAs)**.** A DFA is a tuple $\mathcal{M} = (Q, q_0, F, \delta, I)$, where

- Q is a finite set of states
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is the set of final states
- $\delta : Q \times I \to Q$ is a transition function
- I is the input alphabet

As we use multiple automata in the same equations, we use superscripts to differentiate the properties of the automata. For example, $Q^{\mathcal{M}}$ are the states of DFA $\mathcal{M}$ and $\delta^{\mathcal{N}}$ is the transition function of DFA $\mathcal{N}$.

For the remainder of this thesis, we assume we are working with a shared input alphabet I.

The language $\mathcal{L}_{\mathcal{M}}$ of a DFA $\mathcal{M}$ is defined as follows.

Definition 2.2. The language $\mathcal{L}_{\mathcal{M}} : I^* \to \{0, 1\}$ of DFA $\mathcal{M} = (Q^{\mathcal{M}}, q_0^{\mathcal{M}}, F^{\mathcal{M}}, \delta^{\mathcal{M}}, I)$ is defined by

$$\mathcal{L}_{\mathcal{M}}(w) = \delta^{*\mathcal{M}}(q_0^{\mathcal{M}}, w)$$

where $\delta^{*\mathcal{M}} : Q^{\mathcal{M}} \times I^* \to \{0, 1\}$ *is defined as*

$$\delta^{*\mathcal{M}}(q, \lambda) = [q \in F]$$
$$\delta^{*\mathcal{M}}(q, aw) = \delta^{*\mathcal{M}}(\delta^{\mathcal{M}}(q, a), w)$$

If $\mathcal{L}_\mathcal{M}$ and $\mathcal{L}_\mathcal{N}$ describe the same language, then we say $\mathcal{L}_\mathcal{M}$ and $\mathcal{L}_N$ are equivalent. If we have multiple DFAs with a shared input alphabet I, then we use them to construct new DFAs.

One method to construct new DFAs is taking their product [7]. For a product $\mathcal{M} \times \mathcal{N}$ of DFAs $\mathcal{M}$ and $\mathcal{N}$, the following should hold for all words $w \in I^*$.

$$\mathcal{L}_{\mathcal{M}\times\mathcal{N}}(w) \iff \mathcal{L}_\mathcal{M}(w) \wedge \mathcal{L}_\mathcal{N}(w)$$

Definition 2.3 (product on DFAs)**.** *For DFAs $\mathcal{M} = (Q^\mathcal{M}, q_0^\mathcal{M}, F^\mathcal{M}, \delta^\mathcal{M}, I)$ and $\mathcal{N} = (Q^\mathcal{N}, q_0^\mathcal{N}, F^\mathcal{N}, \delta^\mathcal{N}, I)$, we construct the product $\mathcal{M} \times \mathcal{N}$ as follows.*

- $Q^{\mathcal{M}\times\mathcal{N}} = Q^\mathcal{M} \times Q^\mathcal{N}$
- $q_0^{\mathcal{M}\times\mathcal{N}} = (q_0^\mathcal{M}, q_0^\mathcal{N})$
- $F^{\mathcal{M}\times\mathcal{N}} = \{(q_m, q_n) \in Q^{\mathcal{M}\times\mathcal{N}} | [q_m \in F^\mathcal{M}] \wedge [q_n \in F^\mathcal{N}]\}$
- $\delta^{\mathcal{M}\times\mathcal{N}}((q_m, q_n), i) = (\delta^\mathcal{M}(q_m, i), \delta^\mathcal{N}(q_n, i))$
- *I is the shared alphabet*

Example 2.4. *Let $\mathcal{M}$ be a DFA accepting words with an odd number of a's, visualised in Figure 2.1, and $\mathcal{N}$ a DFA accepting an even number of b's, visualised in Figure 2.2.*

If we take the product $\mathcal{M} \times \mathcal{N}$ of $\mathcal{M}$ and $\mathcal{N}$ using Definition 2.3, then we get the following visual representation in Figure 2.3.

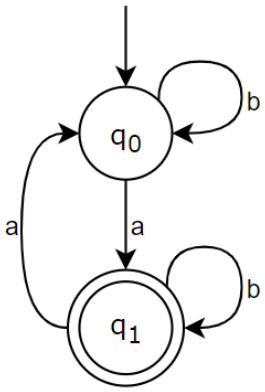


Figure 2.1: Odd number of a's

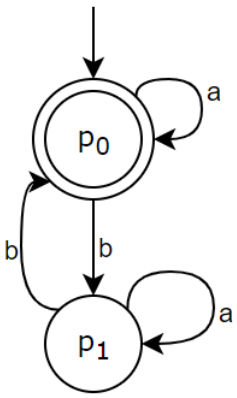


Figure 2.2: Even number of b's

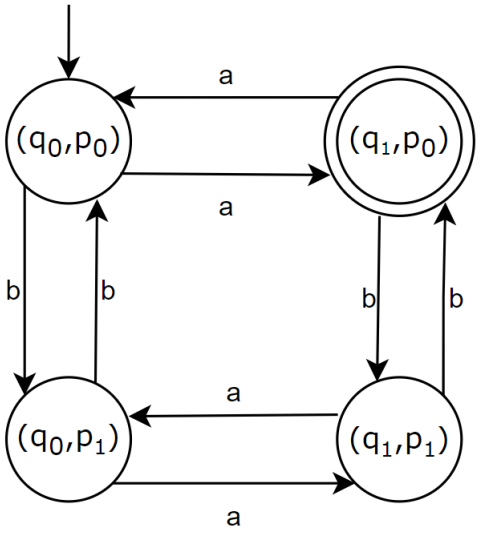


Figure 2.3: Odd number of a's and even number of b's

Using this construction on $\mathcal{M}$ and $\mathcal{N}$, we can distinguish four cases where words are only accepted by $\mathcal{M}$, $\mathcal{N}$ neither or both using the following words.

- *The word babab is neither accepted by $\mathcal{M}$ or $\mathcal{N}$ and thus not accepted by $\mathcal{M} \times \mathcal{N}$*
- *The word ba is accepted by $\mathcal{M}$, but not $\mathcal{N}$, thus not accepted by $\mathcal{M} \times \mathcal{N}$*
- *The word baba is accepted by $\mathcal{N}$, but not $\mathcal{M}$, thus not accepted by $\mathcal{M} \times \mathcal{N}$*
- *The word ababa is accepted by both $\mathcal{M}$ and $\mathcal{N}$ and thus accepted by $\mathcal{M} \times \mathcal{N}$*

Lemma 2.5. For all DFAs $\mathcal{M}$ and $\mathcal{N}$, for all words $w \in I^*$, for all states $q \in Q^{\mathcal{M}}$, $p \in Q^{\mathcal{N}}$

$$\delta^{*\mathcal{M}\times\mathcal{N}}((q,p),w) \iff \delta^{*\mathcal{M}}(q,w) \wedge \delta^{*\mathcal{N}}(p,w)$$

Proof. *Base case $a \in I$.*

$$\begin{aligned}
&\delta^{*\mathcal{M}\times\mathcal{N}}((q,p),a) \\
&\equiv \delta^{*\mathcal{M}\times\mathcal{N}}(\delta^{\mathcal{M}\times\mathcal{N}}((q,p),a),\lambda) \\
&\equiv [\delta^{\mathcal{M}\times\mathcal{N}}((q,p),a) \in F^{\mathcal{M}\times\mathcal{N}}] \\
&\equiv [\delta^{\mathcal{M}}(q,a) \in F^{\mathcal{M}}] \wedge [\delta^{\mathcal{N}}(p,a) \in F^{\mathcal{N}}] \\
&\equiv \delta^{*\mathcal{M}}(\delta^{\mathcal{M}}(q,a),\lambda) \wedge \delta^{*\mathcal{N}}(\delta^{\mathcal{N}}(p,a),\lambda) \\
&\equiv \delta^{*\mathcal{M}}(q,a) \wedge \delta^{*\mathcal{N}}(p,a)
\end{aligned}$$

Inductive case $aw \in I^*$.

$$\begin{aligned}
&\delta^{*\mathcal{M}\times\mathcal{N}}((q,p),aw)\\
&\equiv \delta^{*\mathcal{M}\times\mathcal{N}}(\delta^{\mathcal{M}\times\mathcal{N}}((q,p),a),w)\\
&\equiv \delta^{*\mathcal{M}}(\delta^{\mathcal{M}}(q,a),w) \wedge \delta^{*\mathcal{N}}(\delta^{\mathcal{N}}(p,a),w)\\
&\equiv \delta^{*\mathcal{M}}(q,aw) \wedge \delta^{*\mathcal{N}}(p,aw)
\end{aligned}$$

□

Theorem 1 (Intersection on DFAs)**.** For all DFAs $\mathcal{M}$ and $\mathcal{N}$, for all words $w \in I^*$

$$\mathcal{L}_{\mathcal{M}\times\mathcal{N}}(w) \iff \mathcal{L}_{\mathcal{M}}(w) \wedge \mathcal{L}_{\mathcal{N}}(w)$$

Proof.

$$\begin{aligned}
&\mathcal{L}_{\mathcal{M}\times\mathcal{N}}(w)\\
&\equiv \delta^{*\mathcal{M}\times\mathcal{N}}(q_0^{\mathcal{M}},w)\\
&\equiv \delta^{*\mathcal{M}}(q_0^{\mathcal{M}},w) \wedge \delta^{*\mathcal{N}}(q_0^{\mathcal{N}},w) && \text{(lemma 2.5)}\\
&\equiv \mathcal{L}_{\mathcal{M}}(w) \wedge \mathcal{L}_{\mathcal{N}}(w)
\end{aligned}$$

□

2.2 Learning Intersections

In this section, first, we discuss Angluin style automata learning [2] applied to DFAs. This learning style consists of two main components. The first is a teacher and the second is a learner. Then we briefly discuss strategies built on top of learners.

2.2.1 Minimally adequate teacher

A minimally adequate teacher, also called an oracle, can truthfully answer two types of queries about a hidden DFA $\mathcal{N}$.

The first type of query $\mathcal{M}(w)$, a membership query, asks if w is accepted by $\mathcal{M}$.

The second type of query $\mathcal{E}(\mathcal{H})$, a equivalence query, asks if the given DFA $\mathcal{H}$, called a hypothesis, and $\mathcal{N}$ have equivalent languages.

If $\mathcal{L}_{\mathcal{H}}$ and $\mathcal{L}_{\mathcal{N}}$ are equivalent, then the teacher responds with true, otherwise the teacher produces and responds with a counterexample w, which $\mathcal{H}$ or $\mathcal{N}$ accepts, but not both. Thus for DFAs, these queries are the following.

$$\begin{aligned}
\mathcal{M}(w) &= [w \in \mathcal{L}_{\mathcal{N}}]\\
\mathcal{E}(\mathcal{H}) &= [\mathcal{L}_{\mathcal{H}} = \mathcal{L}_{\mathcal{N}}]
\end{aligned}$$

Example 2.6 (DFA queries)**.** *Let Figure 2.1 be the hidden automaton $\mathcal{N}$. Then*

- *The membership query $\mathcal{M}(a)$ has the response true.*
- *The membership query $\mathcal{M}(aa)$ has the response false.*
- *The equivalence query with Figure 2.3 as hypothesis has a word w as response, because the automata do not describe equivalent languages. A response w could be ba, as this is accepted by $\mathcal{N}$, but not our hypothesis.*
- *The equivalence query with Figure 2.4 as hypothesis has the response true. The languages of the hypothesis and $\mathcal{N}$ are equivalent, even though they do not have the exact same states and transitions.*

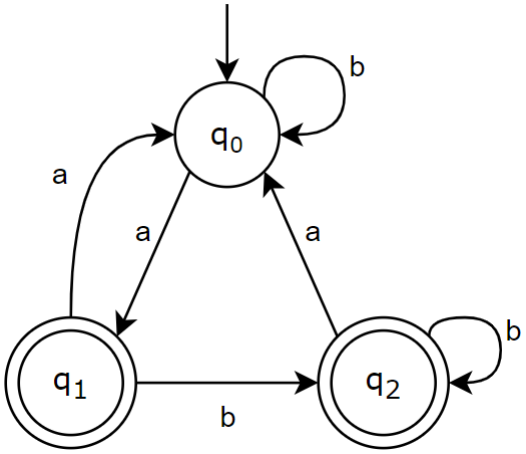


Figure 2.4: Hypothesis for odd number of a

As discussed by Angluin, finding counterexamples using the minimally adequate teacher concept can be problematic in practice [2]. In active automata learning, the teacher does not have a description of the states and transitions of the hidden automaton. It does not have a precise description of the hidden machine, and thus has to make a guess if the hypothesis is equivalent based on testing

We have multiple approaches to find counterexamples.

We could use a breadth-first method given a maximum input length of n, giving both the hypothesis and hidden machine every possible combination of inputs of length n. This would always result in correct answers, if the hidden automaton is smaller than n states. A downside of this method is that it is very expensive, especially because we do not know n and the worst-case for the product is quite large. For DFAs, the worst case product

of DFAs $\mathcal{M}$ and $\mathcal{N}$ is at least $|\mathcal{M}| \cdot |\mathcal{N}|$[1].

Another method is the random walk method. In this method, the teacher starts an input sequence, also called a walk, from the initial state. It chooses a random input and compares the input of the hypothesis and the hidden automaton. For each input, there is a given chance the teacher resets both machines, and starts a new walk from the initial state. This ends either when a given number of inputs have occurred or a counterexample is found.

A more sophisticated approach is the RandomWMethod, which is a randomised version of the Wmethod described by Chow [4]. The RandomWMethod is given a number of walks per state and the length of each walk n. First it computes the prefixes, also known as access sequences, for each state.[2] Then, for each prefix, we perform a given number of walks[3], consisting of the prefix, followed by n random inputs, followed by a random distinguishing sequence from the characterisation set of the hypothesis.

The first advantage of this method, is its dependency on the size of the hypothesis instead of a guess for the hidden automaton. As long as counterexamples are found, this method can learn large automata without the need for an upper bound of states. The second advantage is that repeated validation of a single state is prevented, making it less expensive to test. However, a disadvantage of any randomness such as the RandomWMethod, is that states can still be missed by chance.

2.2.2 Learning algorithm

The second part of Angluin style learning is the learner. This learner can be a learning algorithm, as described by Angluin [2].

The goal of the learner is to learn the hidden automaton. It can receive information about this automaton using the available membership and equivalence queries from the teacher.

The standard learning algorithm to achieve this goal, L^*, described by Angluin, makes use of an observation table, where the outcomes of membership queries are kept.

In the observation table, we have prefixes $S, SI \in I^*$ representing states, and suffixes E, representing distinguishing sequences. For each prefix $s \in S + SI$, $row(s) : E \to \{0, 1\}$ with $row(s)(e) = \mathcal{M}(se)$. $row(q) = row(p)$ if for all $e \in E$, $row(q)(e) = row(p)(e)$.

To create a hypothesis from this observation table, the observation table has to be closed and preferably consistent.

The table is closed if for all $q \in SI$, there exists some $p \in S$ such that $row(q) = row(p)$. If the table is not closed, we add q to S. The table

[1] For example if the languages of $\mathcal{M}$ and $\mathcal{N}$ have the languages $\{x \in \mathcal{Z} \mid x \equiv 1 \mod p\}$ with $p = |\mathcal{M}|$ for $\mathcal{M}$ and $p = |\mathcal{N}|$ for $\mathcal{N}$, and $|\mathcal{M}|$ and $|\mathcal{N}|$ are relatively prime.

[2] Every unique state has an unique prefix.

[3] A total number of walks for each prefix for the entire learning process

is consistent if for all $q, p \in S$, if $row(q) = row(p)$, then for all $i \in I$, $row(qi) = row(pi)$. If the table is not consistent, then for some $e \in E$ with $row(qi)(e) \neq row(pi)(e)$, we add ie to $\mathcal{E}$, such that we can differentiate states q and p. If the table is not consistent or closed, then either a prefix or suffix is added and membership queries are given to the teacher until the table is filled.

If we can construct a hypothesis, then we ask an equivalence query. If the response is true, then we are finished, otherwise we add the counterexample and its prefixes to S and repeat this process.

Example 2.7 (Example run of L^*)**.** *Let $\mathcal{M}$ be the hidden DFA from Figure 2.6. Then our initial observation table would be Table 2.1*[4]*.*

	λ
λ	0
a	1
b	0

Table 2.1: Initial observation table

This observation table is not closed, because row(a) $\neq row(\lambda)$, thus we add a *to our states and use membership queries to fill the empty cells, resulting in Table 2.2.*

	λ
λ	0
a	1
b	0
aa	1
ab	1

Table 2.2: Second observation table

This observation table is both closed and consistent, thus we create our hypothesis as shown in Figure 2.5 and use an equivalence query with this hypothesis.

[4]The horizontal line between prefixes separates S from SI.

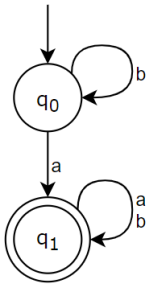


Figure 2.5: First hypothesis

The teacher produces and returns a counterexample, for example aaba. We add a, aa, aab *and* aaba *to our states and update our observation table as shown in Table 2.3.*

	λ
λ	0
a	1
aa	1
aab	1
aaba	0
b	0
ab	1
aaa	0
aabb	1
aabaa	1
aabab	0

Table 2.3: Third observation table

Using this table, we can create the hypothesis as shown in Figure 2.6. The equivalence query of this hypothesis results in true, thus this hypothesis is equivalent to the hidden automaton $\mathcal{M}$.

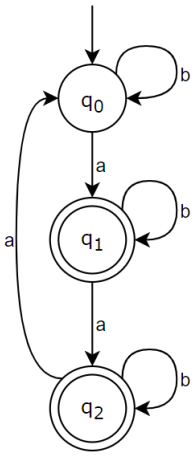


Figure 2.6: DFA accepting the languages $|w|_{\mathrm{a}} \equiv 2 \mod 3$ and $|w|_{\mathrm{a}} \equiv 1 \mod 3$

2.2.3 Learning strategies

As it is possible to learn DFAs, it could also be possible to learn constructions built on DFAs, assuming we can produce a DFA using this construction. For these constructions, we assume the learner has access to membership queries for any combination of hidden automata.

If we want to learn the product of DFAs, then some strategies of querying the DFAs could be more efficient than others.

Three of such strategies are described by Junges and Rot [8] for products of DFAs.

If our goal is to learn the product of DFAs $\mathcal{N}_1, \mathcal{N}_2$ and$\mathcal{N}_3$, then we learn the following language, where $\mathcal{L}_{\mathcal{N}_i}$ is the language associated with DFA $\mathcal{N}_i$.

$$\mathcal{L}_{\mathcal{N}_1} \cap \mathcal{L}_{\mathcal{N}_2} \cap \mathcal{L}_{\mathcal{N}_3} \tag{2.1}$$

The first of the strategies is the independent strategy, where the intersection is avoided by learning the individual automata and taking the product afterwards. The membership and equivalence queries are the following, where i is the ith DFA to be learned.

$$\mathcal{M}_i(w) = [w \in \mathcal{L}_{\mathcal{N}_i}]$$
$$\mathcal{E}_i(\mathcal{H}_i) = [\mathcal{L}_{\mathcal{H}_i} = \mathcal{L}_{\mathcal{N}_i}]$$

We learn the intersection of equation 2.1 with the independent strategy using the following steps.

1. We learn $\mathcal{L}_{\mathcal{N}_1}$.
2. We learn $\mathcal{L}_{\mathcal{N}_2}$.

3. We learn $\mathcal{L}_{\mathcal{N}_3}$.

4. We take $\mathcal{H}_1 \times \mathcal{H}_2 \times \mathcal{H}_3$ as the product of the hypotheses $\mathcal{H}_1, \mathcal{H}_2$ and $\mathcal{H}_3$.

5. $\mathcal{H}_1 \times \mathcal{H}_2 \times \mathcal{H}_3$ is the final product.

The second strategy is the word-by-word strategy, which queries every machine at once, as if it were querying the final product. To accomplish this, it uses the following queries.

$$\mathcal{M}(w) = \bigwedge_i \mathcal{M}_i(w)$$
$$\mathcal{E}(\mathcal{H}) = [\mathcal{L}_{\mathcal{H}} = \bigcap_i \mathcal{L}_{\mathcal{N}_i}]$$

We learn the intersection of equation 2.1 with the word-by-word strategy using the following steps.

1. We learn $\mathcal{L}_{\mathcal{N}_1} \cap \mathcal{L}_{\mathcal{N}_2} \cap \mathcal{L}_{\mathcal{N}_3}$ by querying each input to every machine.

2. The hypothesis $\mathcal{H}$ is the final product.

Lastly there is the machine-by-machine strategy, which learns the intersection iteratively.

Using this strategy, we first learn a single language $\mathcal{L}_1$ from the intersection, then we take the next language $\mathcal{L}_2$ from the intersection and learns $\mathcal{L}_1 \cap \mathcal{L}_2$, and keep adding each individual language to the learned intersection until the entire intersection is learned.

While learning these intermediate intersections, we know that if a word w is not in $\mathcal{L}_1 \cap ... \cap \mathcal{L}_i$, then it is not in $\mathcal{L}_1 \cap ... \cap \mathcal{L}_i \cap \mathcal{L}_{i+1}$. Thus in such cases, we do not have to use the membership query for word w, reducing the number of queries required to learn the intersection $\mathcal{L}_1 \cap \mathcal{L}_2 \cap \mathcal{L}_3$.

Learning the intersection in this manner uses the following queries.

$$\mathcal{M}_{\leq i}(w) = \bigwedge_1^i \mathcal{M}_i(w)$$
$$\mathcal{E}_i(\mathcal{H}_i) = [\mathcal{L}_{\mathcal{H}_i} = \bigcap_1^i \mathcal{L}_{\mathcal{N}_i}]$$

We learn the intersection of equation 2.1 with the machine-by-machine strategy using the following steps.

1. We learn $\mathcal{L}_1$.

2. We learn $\mathcal{L}_1 \cap \mathcal{L}_2$ by testing each word with $\mathcal{H}_1$. If the response is true, then we query the word to the teacher, otherwise we write down false as the response.

3. We learn $\mathcal{L}_1 \cap \mathcal{L}_2 \cap \mathcal{L}_3$ by testing each word with $\mathcal{H}_2$. If the response is true, then we query the word to the teacher, otherwise we write down false as the response.

4. The hypothesis $\mathcal{H}_3$ is the final product.

Chapter 3

Mealy intersections

In this section, we introduce Mealy machines and their language. Then we provide the definitions of intersections on words, languages and Mealy machines. Products on Mealy machines require specific properties, thus we introduce $\cap$-Mealy machines. Finally we adapt learning strategies to learn products of Mealy machines.

3.1 Language of Mealy machines

In this thesis, we use a notion of languages for Mealy machines. We use these languages to implement the concept of giving input words and receiving output words from Mealy machines.

If we have a Mealy machine $\mathcal{M}$, then we have the languages $\mathcal{M}$, $\mathcal{L}_{\mathcal{M}}$, such that $\mathcal{L}_{\mathcal{M}}$ and $\mathcal{M}$ produce an equal output for any input symbols.

Before formally defining the language of Mealy machines, we show the definition of a Mealy machine.

Definition 3.1 (Mealy machines)**.** *A Mealy machine is a tuple* $\mathcal{M} = (Q, q_0, I, O, \delta, \lambda)$*, where*

- Q *is a finite set of states*
- $q_0 \in Q$ *is the initial state*
- I *is the input alphabet*
- O *is the output alphabet*
- $\delta : Q \times I \to Q$ *is a transition function, determining the next state of the machine, based on the current state and input*
- $\lambda : Q \times I \to O$ *is an output function, determining the returned output, based on the current state and input*

The transition and output function for states $q, q' \in Q$ with input $i \in I$ and output $o \in O$ are denoted as $\delta(q, i) = q'$ and $\lambda(q, i) = o$ respectively. A visual representation of these function is $q \xrightarrow{i/o} q'$.

Now that we have shown the definition of a Mealy machine, we use the properties of Mealy machines to formally define a language as follows.

Definition 3.2 (Language of Mealy machines)**.** *The language $\mathcal{L}_{\mathcal{M}} : I^* \to O^*$ of Mealy machine $\mathcal{M} = (Q^{\mathcal{M}}, q_0^{\mathcal{M}}, I^{\mathcal{M}}, O^{\mathcal{M}}, \delta^{\mathcal{M}}, \lambda^{\mathcal{M}})$ is defined by*

$$\mathcal{L}_{\mathcal{M}}(w) = \lambda^{*\mathcal{M}}(q_0^{\mathcal{M}}, w)$$

where $\lambda^{*\mathcal{M}} : Q^{\mathcal{M}} \times I^* \to O^*$ is defined as

$$\begin{aligned} \lambda^{*\mathcal{M}}(q, \lambda) &= \lambda \\ \lambda^{*\mathcal{M}}(q, aw) &= \lambda^{\mathcal{M}}(q, a)\lambda^{*\mathcal{M}}(\delta^{\mathcal{M}}(q, a), w) \end{aligned}$$

Remark 3.3. A consequence of using languages as defined in Definition 3.2, is that multiple Mealy machines can have the same language, as it does not require the Mealy machine to be minimal.

3.2 Product on Mealy machines

Products on automata are dependent on the definition of said automata.

As we have seen in Section 2.1, the definition of a product on DFAs is self-evident. Because DFAs accept words based on the final states, there exists little room for interpretation about what it means for an automaton to be the product of other DFAs.

This is not the case for more complex automata, like the Mealy machine, and thus we first define what it means for an automaton to be the product of Mealy machines.

To help ourselves understand the meaning of intersections, we use the following equation to define products of automata.

$$\mathcal{L}_{\mathcal{M}_1} \cap \mathcal{L}_{\mathcal{M}_2} = \mathcal{L}_{\mathcal{M}_1 \times \mathcal{M}_2} \tag{3.1}$$

Using this equation, we have to define two functions.

1. $\cap$, the intersection of Mealy languages.
2. $\mathcal{M}_1 \times \mathcal{M}_2$, the product such that the equation 3.3 holds.

Remark 3.4. In this thesis, we mainly show the products of two or three automata. However, this concept can be applied to more automata by taking the product of products. The same holds for the intersection of languages.

3.2.1 Intersecting function on Mealy languages

Unlike the intersections described in Section 2.1, an intersection on words can have multiple interpretations.

In this thesis, we interpret an intersection $w \cap v$ on the words $w, v \in O^*$ as the longest common prefix of w and v, followed by special symbols θ, such that $|w \cap v| = max(|w|, |v|)$.

Definition 3.5 (Intersection on Mealy words)**.** *For words $w, v \in O^*$ of equal length, the intersection $\cap : O^* \times O^* \to O^*\{\theta\}^*$ is defined as*

$$w \cap v = lcp(w, v)\theta^{max(|w|,|v|)-|lcp(w,v)|}$$

Where lcp is the longest common prefix of w and v.

Remark 3.6. In this thesis, we generally give a single input to multiple Mealy machines. With the definition of Mealy machines as in Definition 3.1, we have that w and v as in Definition 3.5 are generally of an equal length. In such cases, we use the following equation instead.

$$w \cap v = lcp(w, v)\theta^{|w|-|lcp(w,v)|}$$

Definition 3.7 (Intersection on Mealy languages)**.** *For Mealy languages $\mathcal{L}_\mathcal{M}$, $\mathcal{L}_\mathcal{N}$ and word w, the intersection $\cap : I^* \to O^*\{\theta\}^*$ is defined as*

$$\begin{aligned}\mathcal{L}_\mathcal{M} \cap \mathcal{L}_\mathcal{N}(\lambda) &= \lambda \\ \mathcal{L}_\mathcal{M} \cap \mathcal{L}_\mathcal{N}(w) &= \mathcal{L}_\mathcal{M}(w) \cap \mathcal{L}_\mathcal{N}(w)\end{aligned}$$

Example 3.8. *Let w be a word with $|w| = 5$ such that $\mathcal{L}_\mathcal{M}(w) = abaaa$ and $\mathcal{L}_\mathcal{N}(w) = abbba$. Then*

$$\begin{aligned}\mathcal{L}_\mathcal{M} \cap \mathcal{L}_\mathcal{N}(w) &= \mathcal{L}_\mathcal{M}(w) \cap \mathcal{L}_\mathcal{N}(w) \\ &= lcp(\mathcal{L}_\mathcal{M}(w), \mathcal{L}_\mathcal{N}(w))\theta^{|w|-|lcp(\mathcal{L}_\mathcal{M}(w),\mathcal{L}_\mathcal{N}(w))|} \\ &= lcp(abaaa, abbba)\theta^{5-|lcp(abaaa,abbba)|} \\ &= ab\theta^{5-|ab|} \\ &= ab\theta\theta\theta\end{aligned}$$

Thus the intersection of abaaa and abbba is $ab\theta\theta\theta$.

3.2.2 ∩-Mealy machines

The primary automata model we use, beside the classical Mealy machine, is the ∩-Mealy machine.

A ∩-Mealy machine is an extension of the classical Mealy machine described in Definition 3.1. There are two additions which differentiates a ∩-Mealy machine from a classic Mealy machine.

The first is a special symbol $\theta \in O$, which indicates that every following input character will have θ as output.

The second addition is a sink ρ. For any input given to a $\cap$-Mealy machine, if the output is θ, then the transition function must result in ρ.

Thus we formally define a $\cap$-Mealy machine as follows.

Definition 3.9 ($\cap$-Mealy machines)**.** *A $\cap$-Mealy machine is a Mealy machine $\mathcal{M} = (Q, q_0, \delta, \lambda)$, where*

- *Q is a finite set of states, with $\rho \in Q$*
- *q_0 is the initial state*
- *$\delta : Q \times I \to Q$ is a transition function, with*
  - *for all $q \in Q$ and $i \in I$, if $\lambda(q, i) = \theta$, then $\delta(q, i) = \rho$*
- *$\lambda : Q \times I \to (O + \{\theta\})$ is an output function, with for all $i \in I$, $\lambda(\rho, i) = \theta$*

Remark 3.10. Definition 3.9 does not require every transition to ρ to have an output θ. In figure 3.1, we have a Mealy machine, which adheres to the requirements of a $\cap$-Mealy machine. However, this is not a minimal machine. Every transition from q_1 results in the output θ, thus for every state $q \in Q$ and input $i \in I$, transitions $\delta(q, i) = q_1$ could be replaced by $\delta(q, i) = \rho$ without changing the language of the automaton, resulting in the minimal automaton seen in Figure 3.2.

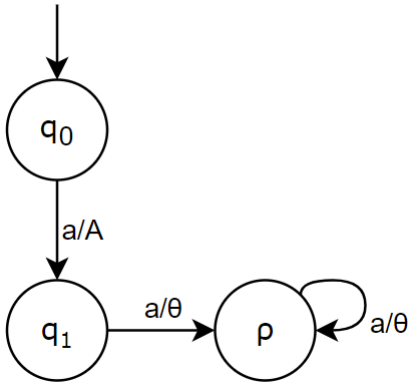


Figure 3.1: Non-minimal $\cap$-Mealy machine

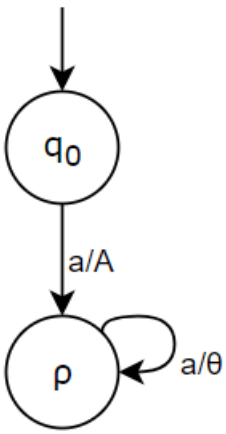


Figure 3.2: Minimal $\cap$-Mealy machine

3.2.3 Constructing products on Mealy machines

Lastly, we have to define the product construction, such that the language of this product is equivalent to the intersection of the languages of the individual Mealy machines.

We construct this using a similar to Definition 2.3.

Definition 3.11. *For $\cap$-Mealy machines $\mathcal{M}$ and $\mathcal{N}$, the product $\mathcal{M} \times \mathcal{N}$ as*

- $Q^{\mathcal{M}\times\mathcal{N}} = Q^{\mathcal{M}} \times Q^{\mathcal{N}} + \rho$
- $q_0^{\mathcal{M}\times\mathcal{N}} = (q_0^{\mathcal{M}}, q_0^{\mathcal{N}})$
- $\lambda^{\mathcal{M}\times\mathcal{N}}(\rho, i) = \theta$
- $\lambda^{\mathcal{M}\times\mathcal{N}}((q_m, q_n), i) = \begin{cases} \lambda^{\mathcal{M}}(q_m, i) & \text{if } \lambda^{\mathcal{M}}(q_m, i) = \lambda^{\mathcal{N}}(q_n, i) \\ \theta & \text{otherwise} \end{cases}$
- $\delta^{\mathcal{M}\times\mathcal{N}}(\rho, i) = \rho$
- $\delta^{\mathcal{M}\times\mathcal{N}}((q_m, q_n), i) = \begin{cases} \rho & \text{if } \lambda^{\mathcal{M}\times\mathcal{N}}((q_m, q_n), i) = \theta \\ (\delta^{\mathcal{M}}(q_m, i), \delta^{\mathcal{N}}(q_n, i)) & \text{otherwise} \end{cases}$

Example 3.12 (Product on Mealy machines)**.** *Let the Mealy machine from Figure 3.3 be $\mathcal{M}$ and Figure 3.4 be $\mathcal{N}$.*

$\mathcal{M}$ and $\mathcal{N}$ differ on two inputs. The first is b in state q_0 and the second is a in state q_1. From Definition 3.11, we have that these transitions should result in an output θ and the state ρ.

From $\mathcal{M}$ and $\mathcal{N}$, we can observe that it would not be possible to reach the combination of states (q_0, q_1) or (q_1, q_0), thus these would not appear in the product of $\mathcal{M}$ and $\mathcal{N}$.

With these observations, if we take the product of $\mathcal{M}$ and $\mathcal{N}$ using Definition 3.11, we create the product shown in Figure 3.5.

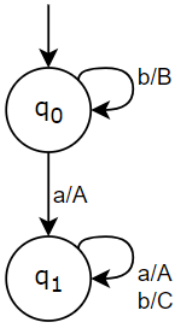


Figure 3.3: Small Mealy machine

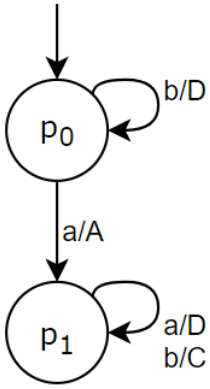


Figure 3.4: Small Mealy machine with changed outputs

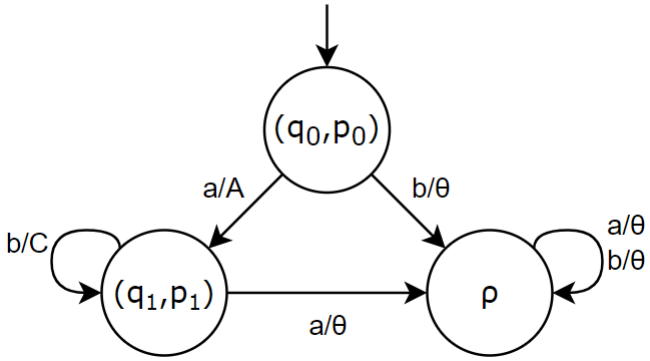


Figure 3.5: Product of small Mealy machines

Lemma 3.13. For all $\cap$-Mealy machines $\mathcal{M}$ and $\mathcal{N}$, For all words $w \in I^*$, states $q \in Q^{\mathcal{M}}$ and $p \in Q^{\mathcal{N}}$

$$\lambda^{*\mathcal{M}\times\mathcal{N}}((q,p),w) = \lambda^{*\mathcal{M}}(q,w) \cap \lambda^{*\mathcal{N}}(p,w)$$

Proof. Base case $a \in I$.

$$\begin{aligned}
&\lambda^{*\mathcal{M}\times\mathcal{N}}((q,p),a) \\
&= \lambda^{\mathcal{M}\times\mathcal{N}}((q,p),a)\lambda^{*\mathcal{M}\times\mathcal{N}}(\delta^{\mathcal{M}\times\mathcal{N}}((q,p),a),\lambda) \\
&= \lambda^{\mathcal{M}\times\mathcal{N}}((q,p),a)
\end{aligned}$$

Case $\lambda^{\mathcal{M}}(q,a) = \lambda^{\mathcal{N}}(p,a)$.

$$\begin{aligned}
&\lambda^{\mathcal{M}\times\mathcal{N}}((q,p),a)\\
&= \lambda^{\mathcal{M}}(q,a)\\
&= lcp(\lambda^{\mathcal{M}}(q,a), \lambda^{\mathcal{M}}(q,a))\theta^{|\lambda^{\mathcal{M}}(q,a)|-|lcp(\lambda^{\mathcal{M}}(q,a),\lambda^{\mathcal{M}}(q,a))|}\\
&= lcp(\lambda^{\mathcal{M}}(q,a), \lambda^{\mathcal{N}}(p,a))\theta^{|\lambda^{\mathcal{M}}(q,a)|-|lcp(\lambda^{\mathcal{M}}(q,a),\lambda^{\mathcal{N}}(p,a))|}\\
&= \lambda^{\mathcal{M}}(q,a) \cap \lambda^{\mathcal{N}}(p,a)
\end{aligned}$$

Case $\lambda^{\mathcal{M}}(q,a) \neq \lambda^{\mathcal{N}}(p,a)$.

$$\begin{aligned}
&\lambda^{\mathcal{M}\times\mathcal{N}}((q,p),a)\\
&= \theta\\
&= lcp(\lambda^{\mathcal{M}}(q,a), \lambda^{\mathcal{N}}(p,a))\theta^{|\lambda^{\mathcal{M}}(q,a)|-|lcp(\lambda^{\mathcal{M}}(q,a),\lambda^{\mathcal{N}}(p,a))|}\\
&= \lambda^{\mathcal{M}}(q,a) \cap \lambda^{\mathcal{N}}(p,a)
\end{aligned}$$

Inductive case $aw \in I^*$.

$$\begin{aligned}
&\lambda^{\mathcal{M}\times\mathcal{N}}((q,p),aw)\\
&= \lambda^{\mathcal{M}\times\mathcal{N}}((q,p),a)\lambda^{*\mathcal{M}\times\mathcal{N}}(\delta^{\mathcal{M}\times\mathcal{N}}((q,p),a),w)
\end{aligned}$$

Case $\lambda^{\mathcal{M}}(q,a) = \lambda^{\mathcal{N}}(p,a)$.

$$\begin{aligned}
&\lambda^{\mathcal{M}\times\mathcal{N}}((q,p),a)\lambda^{*\mathcal{M}\times\mathcal{N}}(\delta^{\mathcal{M}\times\mathcal{N}}((q,p),a),w)\\
&= \lambda^{\mathcal{M}}(q,a)\lambda^{*\mathcal{M}\times\mathcal{N}}(\delta^{\mathcal{M}\times\mathcal{N}}((q,p),a),w)\\
&= lcp(\lambda^{\mathcal{M}}(q,a), \lambda^{\mathcal{M}}(q,a))\lambda^{*\mathcal{M}\times\mathcal{N}}(\delta^{\mathcal{M}\times\mathcal{N}}((q,p),a),w)\\
&= lcp(\lambda^{\mathcal{M}}(q,a), \lambda^{\mathcal{N}}(p,a))(\lambda^{*\mathcal{M}}(\delta(q,a),w) \cap \lambda^{*\mathcal{N}}(\delta(p,a),w))\\
&= lcp(\lambda^{\mathcal{M}}(q,a), \lambda^{\mathcal{N}}(p,a))lcp_1\theta^{|w|-|lcp_1|}\\
&= lcp_2\theta^{|aw|-|lcp_2|}\\
&= lcp(\lambda^{*\mathcal{M}}(q,aw), \lambda^{*\mathcal{N}}(p,aw))\theta^{|aw|-|lcp(\lambda^{*\mathcal{M}}(q,aw),\lambda^{*\mathcal{N}}(p,aw))|}\\
&= \lambda^{*\mathcal{M}}(q,aw) \cap \lambda^{*\mathcal{N}}(p,aw)
\end{aligned}$$

$where$

$$\begin{aligned}
&lcp_1 = lcp(\lambda^{*\mathcal{M}}(\delta^{\mathcal{M}}(q,a),w), \lambda^{*\mathcal{N}}(\delta^{\mathcal{N}}(p,a),w))\\
&lcp_2 = lcp(\lambda^{\mathcal{M}}(q,a)\lambda^{*\mathcal{M}}(\delta^{\mathcal{M}}(q,a),w), \lambda^{\mathcal{N}}(p,a)\lambda^{*\mathcal{N}}(\delta^{\mathcal{N}}(p,a),w))
\end{aligned}$$

Case $\lambda^{\mathcal{M}}(q,a) \neq \lambda^{\mathcal{N}}(p,a)$.

$$\begin{aligned}
&\lambda^{\mathcal{M}\times\mathcal{N}}((q,p),a)\lambda^{*\mathcal{M}\times\mathcal{N}}(\delta^{\mathcal{M}\times\mathcal{N}}((q,p),a),w)\\
&= \theta\lambda^{*\mathcal{M}\times\mathcal{N}}(\rho,w)\\
&= \theta^{aw}\\
&= lcp(\lambda^{*\mathcal{M}}(q,aw), \lambda^{*\mathcal{N}}(p,aw))\theta^{|aw|-|lcp(\lambda^{*\mathcal{M}}(q,aw),\lambda^{*\mathcal{N}}(p,aw))|}\\
&= \lambda^{*\mathcal{M}}(q,aw) \cap \lambda^{*\mathcal{N}}(p,aw)
\end{aligned}$$

□

Theorem 2 (Product on Mealy machines)**.** *For all ∩-Mealy machines, for all words $w \in I^*$*

$$\mathcal{L}_{\mathcal{M}\times\mathcal{N}}(w) = \mathcal{L}_{\mathcal{M}}(w) \cap \mathcal{L}_{\mathcal{N}}(w) \tag{3.2}$$

Proof.

$$\begin{aligned}
&\mathcal{L}_{\mathcal{M}\times\mathcal{N}}(w) \\
&= \lambda^{*\mathcal{M}\times\mathcal{N}}((q_0^{\mathcal{M}}, q_0^{\mathcal{N}}), w) \\
&= \lambda^{*\mathcal{M}}(q_0^{\mathcal{M}}, w) \cap \lambda^{*\mathcal{N}}(q_0^{\mathcal{N}}, w) && (lemma\ 3.13) \\
&= \mathcal{L}_{\mathcal{M}}(w) \cap \mathcal{L}_{\mathcal{N}}(w)
\end{aligned}$$

□

3.3 Learning Mealy Intersections

In this section, we briefly discuss Angluin style learning in the context of Mealy machines. Then we discuss strategies over learning algorithms to learn products of Mealy machines.

3.3.1 Learning Mealy machines

In the context of Mealy machines, the minimally adequate teacher has access to both membership and equivalence queries. However, the membership queries for Mealy machines differ in the exact responses.

For a Mealy machine $\mathcal{N}$, membership queries, also known as output queries [11] [12], ask what response is given for $w \in I^*$ as input to $\mathcal{N}$, thus they are defined as follows.

$$\mathcal{M}(w) = \mathcal{L}_{\mathcal{N}}(w)$$

For equivalence queries, there is no difference between the definition in the context of DFAs or Mealy machines. In both contexts, the query asks if the given hypothesis has the same language as the hidden automaton $\mathcal{N}$. Thus for Mealy machines, we use the same definition as in Section 2.2.1.

$$\mathcal{E}(\mathcal{H}) = [\mathcal{L}_H = \mathcal{L}_{\mathcal{N}}]$$

For learners operating on a structure like an observation table in the L^* algorithm, we keep track of each output symbol for every input symbol. So instead of an observation table filled with 0's and 1's, we have a table with various output symbols, such as shown in Table 3.1.

	a	b
λ	A	B
a	A	C
b	A	B
aa	A	B
ab	A	C

Table 3.1: Example observation table

3.3.2 Learning strategies

In this section we discuss the three learning strategies mentioned in Section 2.2.3 applied to Mealy machines with examples showing their strengths.

The first strategy is the independent, the second is the word-by-word and the third is the machine-by-machine strategy.

If we are learning the product of n Mealy machines, then we are learning the following intersection.

$$\mathcal{L}_{\mathcal{N}_1} \cap ... \cap \mathcal{L}_{\mathcal{N}_n} = \mathcal{L}_{\mathcal{N}_1 \times ... \times \mathcal{N}_n} \tag{3.3}$$

The strategies learn the intersection on the left part of equation 3.3. If we apply Theorem 2, we have that the Mealy machine $\mathcal{N}_1 \times ... \times \mathcal{N}_n$, with $\mathcal{L}_{\mathcal{N}_1 \times ... \times \mathcal{N}_n} = \mathcal{L}_{\mathcal{N}_1} \cap ... \cap \mathcal{L}_{\mathcal{N}_n}$, is the product of $\mathcal{N}_1, ..., N_n$.

The strategies act on the construction found in the left part of the equation 3.3 to learn the automaton on the right part.

Independent strategy

The independent strategy first learns the individual languages $\mathcal{L}_{\mathcal{N}_1}, ..., \mathcal{L}_{\mathcal{N}_n}$ and then takes their intersection. From Definition 3.11, we can produce a product of Mealy machines. Thus to learn the intersection, it suffices to learn each individual machine.

To learn each machine, we use the following membership and equivalence queries.

$$\mathcal{M}_i(w) = \mathcal{L}_{\mathcal{N}_i}(w)$$
$$\mathcal{E}_i(\mathcal{H}_i) = [\mathcal{L}_{\mathcal{H}_i} = \mathcal{L}_{\mathcal{N}_i}]$$

The membership query simply asks the teacher to return the output to a specific numbered hidden machine $\mathcal{N}_i$, and the equivalence query asks if the current hypothesis for machine $\mathcal{N}_i$ produces an equivalent language to $\mathcal{N}_i$.

Example 3.14 (Extreme case for independent strategy)**.** *Let $\mathcal{M}$ and $\mathcal{N}$ be the automata represented by Figure 3.6 and Figure 3.7 respectively.*

For $\mathcal{M}$ and $\mathcal{N}$, usually every a has output A and b output B. However $\mathcal{M}$ gives output A on the third occurring b and $\mathcal{N}$ gives output B on the third a. This results in there always being a difference on the third occurrence of either a or b if we were to give a word to both $\mathcal{M}$ and $\mathcal{N}$.

The minimised product of $\mathcal{M}$ and $\mathcal{N}$ is shown in Figure 3.8. We can see that $\mathcal{M} \times \mathcal{N}$ is larger than the individual automata, thus it is likely to be more efficient to learn $\mathcal{M}$ and $\mathcal{N}$ individually.

This difference in states can be even more extreme as we increase the size of $\mathcal{M}$ an $\mathcal{N}$, with their product growing in an exponential manner.

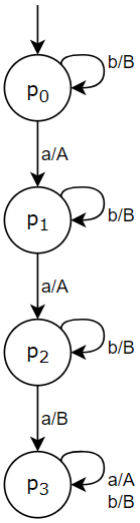


Figure 3.6: Mealy machine with the third A flipped

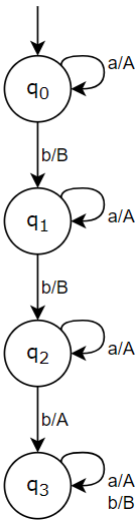


Figure 3.7: Mealy machine with the third B flipped

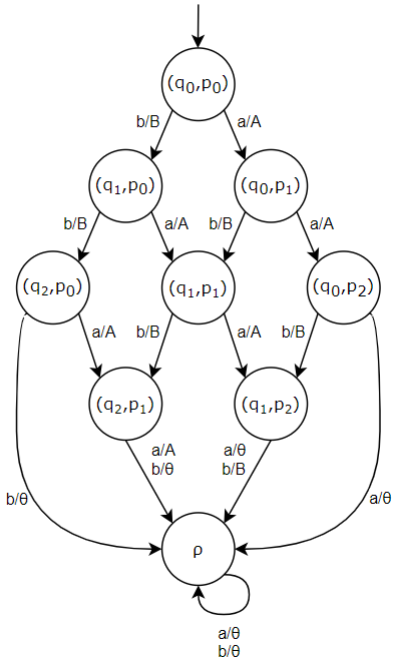


Figure 3.8: Product of Figure 3.6 and 3.7

$$\mathcal{L}_{\mathcal{N}_1} \cap ... \cap \mathcal{L}_{\mathcal{N}_n} = \mathcal{L}_{\mathcal{N}_1 \times ... \times \mathcal{N}_n} \tag{3.3}$$

Word-by-word strategy

The word-by-word strategy learns the intersection, however, instead of learning the individual languages, it learns the entire intersection directly.

Learning this intersection makes use of Definition 3.5 such that the membership queries are the following.

$$\mathcal{M}_{1,...,n}(w) = lcp(\mathcal{L}_{\mathcal{N}_1}(w), ..., \mathcal{L}_{\mathcal{N}_n}(w))\theta^{|w|-|lcp(\mathcal{L}_{\mathcal{N}_1}(w),...,\mathcal{L}_{\mathcal{N}_n}(w))|}$$

These membership queries ask the teacher to return the output of the machines $\mathcal{N}_i$ as if the teacher would first have taken the intersection.

If the observation table is consistent and closed, then it creates an hypothesis $\mathcal{H}_{1,...,n}$ and queries the following equivalence query.

$$\mathcal{E}_{1,...,n}(\mathcal{H}_{1,...,n}) = [\mathcal{L}_{\mathcal{H}_{1,...,n}} = \bigcap_{i=1}^{n} \mathcal{L}_i]$$

Example 3.15 (Extreme case for word-by-word strategy)**.** *Let $\mathcal{M}$ be Figure 3.1 and $\mathcal{N}$ Figure 3.7. In the initial state, $\mathcal{M}$ and $\mathcal{N}$ have different outputs for each input. Thus the language of their product consists only of*

θ's. Taking the product of these machines would result in a trivial $\cap$-Mealy machine with $Q = \{\rho\}$. Learning this single state Mealy machine would obviously be more advantageous than learning $\mathcal{M}$ and $\mathcal{N}$ individually.

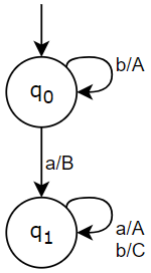


Figure 3.9: Basic Mealy machine

Machine-by-machine strategy

Finally, we have the machine-by-machine strategy. This learns the left part from equation 3.3 in an iterative manner. It first learns $\mathcal{L}_{\mathcal{N}_1}$. Then it learns $\mathcal{L}_{\mathcal{N}_1} \cap \mathcal{L}_{\mathcal{N}_2}$. Then it learns $\mathcal{L}_{\mathcal{N}_1} \cap \mathcal{L}_{\mathcal{N}_2} \cap \mathcal{L}_{\mathcal{N}_3}$, adding a language to the intersection, until the entire intersection is learned.

The membership query is as follows.

$$\mathcal{M}_{1,\dots,i+1}(w) = \mathcal{L}_{\mathcal{N}_{i+1}}(w)$$

Because we have already learned the intersection up to machine $\mathcal{N}_i$, we can first produce $\mathcal{L}_{\mathcal{N}_{1,\dots,i}}(w)$ ourselves. If this results in θ, then we already know what the intersection would produce, and thus do not have to query the teacher. If it results in a normal output, then we only have to query machine $\mathcal{N}_{i+1}$, because we can take the intersection ourselves.

The equivalence queries used for this strategy with $1 \leq i \leq n$ are as follows.

$$\mathcal{E}_{1,\dots,i+1}(\mathcal{H}_{1,\dots,i+1}) = [\mathcal{L}_{\mathcal{H}_{1,\dots,i+1}} = \mathcal{L}_{\mathcal{N}_{1,\dots,i}} \cap \mathcal{L}_{\mathcal{N}_{i+1}}]$$

Because the strategy makes use of an order in which machines are learned, it can have large effects on the amount of states we learn.

Example 3.16 (Order of machines)**.** *Let $\mathcal{M}$ and $\mathcal{N}$ be the machines from Example 3.14, and $\mathcal{P}$ be a Mealy machines of which an intersections with $\mathcal{M}$ or $\mathcal{N}$ would result in a trivial $\cap$-Mealy machine. The following languages are equivalent, because the order of words over which we take an intersection does not matter.*

$$\mathcal{L}_{\mathcal{M}} \cap \mathcal{L}_{\mathcal{N}} \cap \mathcal{L}_{\mathcal{P}} = \mathcal{L}_{\mathcal{M}} \cap \mathcal{L}_{\mathcal{P}} \cap \mathcal{L}_{\mathcal{N}} \tag{3.4}$$

We could learn the left part of this equation, resulting in the following steps.

1. *$\mathcal{L}_{\mathcal{M}}$ is learned.*
2. *$\mathcal{L}_{\mathcal{M}} \cap \mathcal{L}_{\mathcal{N}}$ is learned, from Example 3.14, this intersection is quite large.*
3. *A trivial $\cap$-Mealy machine is learned.*

However, learning the right part is favourable with the following steps.

1. *Again, $\mathcal{L}_{\mathcal{M}}$ is learned.*
2. *A trivial $\cap$-Mealy machine is learned.*
3. *Because we query a trivial machine, we do not have to use membership queries to learn our final intersection.*

Thus the order in which the intersection is learned can have a large effect on the amount of learning we have to do.

Chapter 4

Experiments

In this section, we apply the three strategies defined in Section 3.3.2 to learn the product of various Mealy machines. First we discuss the setup and our metrics. Then, for each of the experiments, we provide details about the experiments, followed by our data and observations of the experiment. Lastly we provide more general observations and a discussion of the experiments.

4.1 Setup

For our experiments, we use our implementation[1] of the three strategies[2] mentioned in Chapter 3. As the learner, we make use of the aalpy implementation of L* [10].

The teacher uses the RandomWMethod[3] strategy for our equivalence queries. This method checks multiple times for each state in the hypothesis, if after a given number of random inputs, the state is in the hypothesis. If this is not the case, then a counterexample has been found.

We chose this equivalence query method, because the cost of the equivalence queries scales with the number of states in our hypotheses instead of the maximum depth or states possible states the product can have. As we have seen from Example 3.14, products can be both very small and very large. Methods based on the maximum number of states would require the teacher to verify that a given hypothesis is equivalent up to the maximum number of states, even though the products we are learning are generally small.

This choice for the equivalence queries has a negative side effect. The teacher can respond that a given hypothesis is correct while it is not. Increasing the number of times we search for a counterexample per state, reduces the likelihood of these incorrect results. In this particular setting,

[1] Using Python 3.11

[2] Can be found at `https://github.com/JulianPut/learning-mealy-intersections`

[3] With $min(10 \cdot \#\mathrm{I}, 100)$ walks per state and 10 inputs per walk

increasing the length of the randomly generated inputs is likely less effective due to the existence of the sink.

After each experiment, we collect the following data from the learning process.

- #MQ, the number of membership queries requested by the learner.
- #EQ, the number of equivalence queries requested by the learner.
- $\#\text{Steps}_{\text{MQ}}$, the number of input symbols performed by the teacher for the membership queries. In general, this is the length of our membership query. This is the main metric we use for determining efficiency, as this is the total cost of receiving information about our hidden product.
- $\#\text{Steps}_{\text{EQ}}$, the number of input symbols performed by the teacher for the equivalence queries.
- $\#S$, the number of states learned by the word-by-word strategy, as learning using this strategy directly produces the product.
- $\%CP$, the percentage of cases where the learner learns the correct product of automata. As we are only working with minimal automata, we can assume that if the learner has learned the correct automaton, then the automaton is equivalent to the product.

Because each strategy learns the intersection slightly differently, we count the data as follows.

For the independent strategy, for each metric, we sum the data to learn each individual automaton. For example, if we learn the product of automata $\mathcal{M}$ and $\mathcal{N}$ requiring 20 and 30 membership queries respectively, then #MQ is 50.

For the word-by-word strategy, the teacher computes the membership queries lazily. Generally, this results in a lower $\#Steps$ for each query. Instead of first computing the entire longest common prefix, it checks for each machine if the first output is the same, then the next input, until outputs are different, or every input is processed. For example, if we have n machines and a membership query of length k, then the number of steps performed could be 2, if the first two machines have different outputs for the first input symbol, k times n, if every machine produces an equal output for the entire word, or somewhere in between if the machines have a partially equal output.

For the machine-by-machine strategy, we sum the data of each intermediate learned intersection. The learner first queries the automaton we have already learned. If one of the symbols results in θ, then we know the following input symbols would result in θ in the product, thus we do not have to query the teacher.

4.2 Detailed results

In this section, we provide the experiments and results from applying the theory to learn the product of Mealy machines.

We have chosen to perform two experiments.

The random experiment is performed on randomly generated mealy machines. As generating random mealy machines is quite fast, we can learn a lot more products than if we have to find real benchmarks.

However, in practice we do not desire to learn the product of random mealy machines, thus in addition to our random experiment, we have a benchmarks experiment. In this experiment, we learn the product of existing benchmarks[4]. These benchmarks are inferred by applying learning algorithms to real world applications.

4.2.1 Random Experiment

For our first experiments, we learn the products of multiple randomly generated mealy machines.

These automata are generated using the aalpy library[5] given the following variables.

- The number of machines to be generated, $\#M$.
- The number of states each generated machine should have, $\#states$.
- The input alphabet. The exact used symbols do not matter, thus in our data we use $\#I$ as the size of the input alphabet.
- The output alphabet. For our generated machines, we use $\{a, b\}$ as our output alphabet.
- The distribution of output symbols in the generated output function, O_r. Given a 4:1 distribution, we can expect that the output function consists of 80% a's and 20% b's.

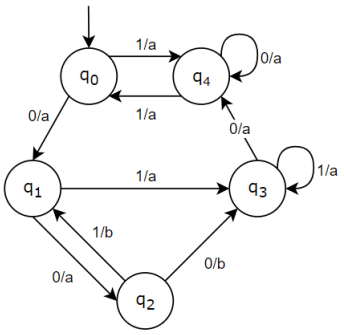
Example 4.1 shows such a generated Mealy machine using these variables.

Example 4.1 (Generated mealy machine)**.** *A generated random mealy machines using aalpy with 5 states,* $\{0, 1\}$ *as input and* {a,b} *as output alphabet with an 4:1 distribution in the output function.*

[4]Selection taken from `https://automata.cs.ru.nl`.

[5]Documentation can be found at `https://des-lab.github.io/AALpy/`.

Figure 4.1: Generated mealy machine



We can observe a bias to the output a *in Figure 4.1, this is due our introduced distribution in the output function*

Our first run of variables uses 4 machines, 5 states per machine, 3 input symbols and a 1:1 distribution. Then we have runs which increase one of these variables. Our final runs have a difference in the number of states for each individual machine. Each run is performed 100 times and has its average data taken.

Table 4.1: learning random intersections

Variables				Independent						Word-By-Word						Machine-By-Machine						
$\#M$	$\#states$	$\#I$	O_r	#MQ	$\#\text{Steps}_{\text{MQ}}$	#EQ	$\#\text{Steps}_{\text{EQ}}$	$\#\text{Steps}_{\text{total}}$	%CP	#MQ	$\#\text{Steps}_{\text{MQ}}$	#EQ	$\#\text{Steps}_{\text{EQ}}$	$\#\text{Steps}_{\text{total}}$	%CP	#MQ	$\#\text{Steps}_{\text{MQ}}$	#EQ	$\#\text{Steps}_{\text{EQ}}$	$\#\text{Steps}_{\text{total}}$	%CP	#S
4	5	3	1:1	279	5631	8	4693	10324	100	16	226	1	301	527	98	308	4030	7	5072	9102	98	2
4	5	3	2:1	290	5737	8	4739	10476	100	47	1682	1	738	2420	87	476	6092	8	7201	13293	88	4
4	5	3	4:1	331	6070	10	4844	10914	99	639	35139	3	5122	40261	64	1572	19605	15	15789	35394	67	18
4	5	3	8:1	368	6356	12	4924	11280	96	3248	186182	9	22460	208642	29	5301	70339	27	42761	113100	27	61
4	5	3	16:1	393	6573	13	4992	11565	95	5224	295932	13	36178	332110	13	7916	104307	34	59895	164202	9	94
4	5	3	32:1	406	6726	13	5048	11774	97	8374	483126	16	54304	537430	19	11465	156048	38	82194	238242	8	129
4	5	4	4:1	482	7771	9	6178	13949	100	3572	183755	8	23391	207146	34	6595	77204	24	50750	127954	31	58
4	5	5	4:1	650	9580	8	7564	17144	100	11736	531245	13	60838	592083	17	18580	195584	32	110534	306118	15	109
4	5	6	4:1	852	11473	7	8954	20427	100	31023	1279265	21	133768	1413033	7	40437	386112	40	201261	587373	8	184
4	6	3	4:1	439	7810	12	6037	13847	100	1433	93704	5	10627	104331	67	3277	46395	21	29432	75827	64	26
4	8	3	4:1	656	11389	15	8469	19858	98	4223	286270	7	25117	311387	50	10102	151830	32	68937	220767	48	68
4	10	3	4:1	914	15278	18	10949	26227	100	19513	1620200	12	84374	1704574	42	30373	536011	45	158924	694935	42	166
6	5	3	16:1	590	9867	19	7492	17359	96	11099	1213224	13	63130	1276354	27	27051	415294	60	168949	584243	22	160
8	5	3	16:1	786	13148	25	9982	23130	93	7349	1344294	7	42062	1386356	52	42808	751888	76	254532	1006420	45	160
10	5	3	16:1	983	16452	32	12479	28931	90	1469	319440	3	9420	328860	69	45902	842344	82	278376	1120720	68	60
4	(3, 5, 7, 11)	3	1:1	419	8160	9	6501	14661	100	17	279	1	310	589	100	191	2214	5	3461	5675	100	1
4	(11, 7, 5, 3)	3	1:1	420	8163	9	6495	14658	100	19	325	1	342	667	98	958	14471	11	12776	27247	97	2
6	(3, 5, 7, 11, 13, 17)	3	1:1	1036	19397	18	14794	34191	100	12	136	1	208	344	100	215	2204	7	3874	6078	100	1
6	(17, 13, 11, 7, 5, 3)	3	1:1	1004	19225	17	14779	34004	99	12	132	1	211	343	100	2918	53223	17	33200	86423	100	1
3	(2, 2, 20)	2	16:1	513	9583	9	5416	14999	94	46	1895	1	777	2672	79	83	1309	3	1573	2882	76	6
3	(3, 3, 20)	2	16:1	539	10044	11	5798	15842	95	269	12110	3	2831	14941	57	307	4621	7	4031	8652	55	18

From the first three rows of Table 4.1, we observe that the increase in their respective variables results in an increase in the number of states in the final product, with as consequence that both the Word-By-Word and Machine-By-Machine strategies have a worse efficiency compared to the Independent strategy.

From the fourth group, we observe that an increase in the amount of machines can cause a decrease in the size of the product, resulting in the word-by-word strategy performing better than the independent strategy.

In the fifth group, we change the order in which we learn automata, from either small to large or large to small automata. We can observe from this group that the order from small to large is favourable in the learning process using the machine-by-machine strategy.

4.2.2 Benchmark Experiment

In the second experiment, we learn the products of benchmarks implementing the same model.

In Table 4.2, the individual benchmarks for each model is listed, together with its minimal size (#S).

The benchmarks listed are produced by using model learning to learn the models and afterwards model checking was used to verify that these protocols follow their respective ISO standard. For example, the MAESTRO benchmarks are inferred by Aarts, de Ruiter and Poll [1] and the TCP benchmarks by Fiterau -Brostean, Janssen and Vaandrager [6][6].

The benchmarks are listed in the order we learn them, which can have an impact on the number of steps and queries performed by both the word-by-word and machine-by-machine strategy.

The input alphabet of each benchmark is not necessarily equivalent to other versions of the same model. For our product, the input alphabet is the union of the input alphabet of each individual benchmark. If we query an input to a machine which does not recognise this input, then θ is returned.

For each model, we learn the product of the benchmarks once.

[6]More information can be found at `automata.cs.ru.nl`

Table 4.2: Model benchmarks part 1

Models	benchmarks	#S
MasterCard	1_learnresult_MasterCard_fix	5
	10_learnresult_MasterCard_fix	6
MAESTRO	4_learnresult_MAESTRO_fix	6
	ASN_learnresult_MAESTRO_fix	6
	Rabo_learnresult_MAESTRO_fix	6
	Volksbank_learnresult_MAESTRO_fix	7
SecureCode	4_learnresult_SecureCode Aut_fix	4
	ASN_learnresult_SecureCode Aut_fix	4
	Rabo_learnresult_SecureCode_Aut_fix	6
QUICprotocol	QUICprotocolwith0RTT	7
	QUICprotocolwithout0RTT	5
SSH	BitVise	66
	DropBear	17
	OpenSSH	31
TCP_client	TCP_FreeBSD_Client	12
	TCP_Linux_Client	15
	TCP_Windows8_Client	13
TCP_server	TCP_FreeBSD_Server	55
	TCP_Linux_Server	57
	TCP_Windows8_Server	38
GnuTLS_client_full	GnuTLS_3.3.8_client_full	15
	GnuTLS_3.3.12_client_full	9
GnuTLS_client_regular	GnuTLS_3.3.8_client_regular	11
	GnuTLS_3.3.12_client_regular	7
GnuTLS_server_full	GnuTLS_3.3.8_server_full	16
	GnuTLS_3.3.12_server_full	9
GnuTLS_server_regular	GnuTLS_3.3.8_server_regular	12
	GnuTLS_3.3.12_server_regular	7
OpenSSL_client	OpenSSL_1.0.1g_client_regular	6
	OpenSSL_1.0.1j_client_regular	10
	OpenSSL_1.0.1l_client_regular	6
	OpenSSL_1.0.2_client_regular	6
OpenSSL_server	OpenSSL_1.0.1g_server_regular	7
	OpenSSL_1.0.1j_server_regular	16
	OpenSSL_1.0.1l_server_regular	11
	OpenSSL_1.0.2_server_regular	10
TLS_RSA_BSAFE_server	RSA_BSAFE_C_4.0.4_server_regular	6
	RSA_BSAFE_Java_6.1.1_server_regular	9
X-ray-system-PCS	learnresult1	7
	learnresult3	6
	learnresult4	6
	learnresult5	8
	learnresult6	8

Table 4.3: learning benchmarks part 1

Models	algorithm	#MQ	$\#Steps_{MQ}$	#EQ	$\#Steps_{EQ}$	$\#Steps_{total}$	%CP	#S
Mastercard	idp	2330	18253	2	9245	27498	100	6
	wbw	1365	13911	1	4896	18807	100	
	mbm	2505	15390	2	9037	24427	100	
MAESTRO	idp	4956	41786	4	21542	63328	100	10
	wbw	1974	51526	1	9105	60631	100	
	mbm	5544	42349	4	24388	66737	100	
SecureCode	idp	2961	22741	3	11585	34326	100	5
	wbw	1140	18381	1	3947	22328	100	
	mbm	2736	17449	3	10338	27787	100	
QUICprotocol	idp	294	5262	3	4167	9429	100	10
	wbw	297	5688	2	3794	9482	35	
	mbm	495	6235	4	6355	12590	18	
SSH	idp	22537	218248	6	81361	299609	1	17
	wbw	6388	115372	1	12733	128105	0	
	mbm	14214	131586	5	60399	191985	0	
TCP_client	idp	4035	52275	4	33882	86157	14	16
	wbw	1789	43437	2	13856	57293	48	
	mbm	5022	50055	7	39178	89233	12	
TCP_server	idp	33948	500452	26	170190	670642	0	52
	wbw	6635	177457	5	31772	209229	0	
	mbm	39882	536644	23	186662	723306	0	
GnuTLS_client_full	idp	1394	11808	2	7257	19065	12	10
	wbw	867	12219	1	4504	16723	16	
	mbm	1492	11241	2	7697	18938	1	
GnuTLS_client_regular	idp	954	13801	2	9889	23690	100	3
	wbw	374	2254	1	2154	4408	100	
	mbm	872	8098	2	7353	15451	100	
GnuTLS_server_full	idp	2876	30414	3	18035	48449	35	10
	wbw	1426	21367	1	7960	29327	44	
	mbm	3079	28670	3	19022	47692	35	
GnuTLS_server_regular	idp	1106	15863	2	11336	27199	100	8
	wbw	520	12194	1	5283	17477	100	
	mbm	1170	15540	2	11936	27476	100	
OpenSSL_client	idp	1400	22409	4	16795	39204	100	7
	wbw	350	18560	1	4101	22661	100	
	mbm	1351	18459	4	15774	34233	100	
OpenSSL_server	idp	2059	33246	4	24569	57815	0	8
	wbw	399	20874	1	4624	25498	100	
	mbm	1547	20869	4	17881	38750	100	
TLS_RSA_BSAFE_server	idp	976	13778	2	10050	23828	100	6
	wbw	392	4816	1	4006	8822	100	
	mbm	784	7894	2	8013	15907	100	
X-ray-system-PCS	idp	2266	29801	5	22041	51842	100	8
	wbw	657	24680	1	5751	30431	100	
	mbm	2462	24650	5	24859	49509	100	

Comparing the sizes of the individual models with their product, we observe that the product is approximately equal in size to the individual models.

From Table 4.3, we observe that the products produced using the Word-By-Word strategy are generally the most likely to be correct and efficient to

produce. Between the Independent and Machine-By-Machine strategy, one of the two is generally a bit more efficient, but which one differs between models.

4.3 Summary and Discussion

First, we summarise our observations of the results, afterwards we briefly discuss some problems we encountered during the experiments.

From our random experiment, we observe that the independent strategy is the most consistent. The efficiency of the word-by-word and machine-by-machine can drastically change based on the specification of machines.

From our benchmarks experiment, we observe that the Word-By-Word strategy is the most efficient for our benchmarks, where the sizes of the product and individual machines only differ slightly.

In general, the larger the product compared to the individual machines, the less efficient the Word-By-Word and Machine-By-Machine strategies are compared to the Independent strategy. Thus there is no general best strategy between the discussed strategies.

For our implementation of the teacher, we used the RandomWMethod with 10 walks per input per state and 10 inputs per walk. After further experimentation on both the benchmarks and randomly generated machines, we observed that these were by no means thorough enough to yield correct intersections. We experimented with 300 walks per state and 12 inputs per walk, and other combinations with a similar total inputs. This slightly increased the correctness of our intersections, but increased $\#Steps_{\text{EQ}}$, and thus the time to run, dramatically.

Some of the benchmarks have been changed to be compatible with the aalpy library. These changes are mainly removing html tags in the input and output, as these conflict with our method of constructing Mealy machines from the benchmarks, and removing redundant states.

Chapter 5

Related Work

There exists a lot of theory surrounding automata learning. In 1987, active automata learning was introduced with the L^* algorithm [2]. This style of learning has branched to various types of machines, but the core principles have stayed the same.

In more recent years, this Angluin style of learning has been applied to more types of machines. For example the N^* algorithm [9], which learns non-deterministic finite Mealy machines, using the adaptations NL^* from Bollig et al. [3] to learn non-deterministic automata and an L^* adaptation from Shahbaz [11] to learn mealy machines.

Not only can this learning style be applied to types of machine, it can also be applied to constructions built on top of automata. One such construction is the intersection of DFAs, which has been explored by Junges and Rot [8], where learning strategies are introduced to potentially learn constructions more efficiently.

Since to the best of our knowledge, there does not exist any theory about learning intersections of Mealy machines, we extended the work of Junges and Rot [8] by applying the ideas presented to Mealy machines. However, not only does there not exist any theory about learning Mealy intersections, to the best of our knowledge, there does not exist any theory about Mealy intersections in general.

Chapter 6

Conclusions

In this thesis, we discussed what a Mealy intersection is and how we can learn this type of intersection.

We first introduced a definition for languages of Mealy machines and the intersection of these languages. Then we introduced products of Mealy machines, using $\cap$-Mealy machines, an extension on the conventional Mealy machine. After adapting learning strategies to be compatible for Mealy machines, we used them to learn the product of both random and benchmark Mealy machines. From these experiments, we found that each strategy has its use cases.

6.1 Future work

One of the potential applications of Mealy intersections is finding common behaviour between multiple Mealy machines. We assumed such common behaviour has to be deterministic. However, one might find it sufficient to have a probabilistic chance of common behaviour between the given Mealy machines. This could change the product of our Mealy intersections to a probabilistic model. For example, given a probability threshold and the number of machines over which we take the intersection, we could learn a partial non-deterministic Mealy machine.

In this thesis, we have applied the theory of learning Mealy intersections to already available benchmarks. However, we have yet to apply our theory directly on real world black box systems. In future work, we can learn intersections directly, instead of first learning each individual system.

As we have mentioned in Section 3.2.1, we can interpret intersections for Mealy machines in multiple ways. One such interpretation makes use of a backwards output, where we first add each output to a stack, and after adding every output, we can apply conditions for popping and returning outputs from the stack. For example, we could apply this idea to intersections of Mealy machines. With the condition that we flush our stack if we pop

a θ, our entire output is a single θ, if the Mealy machines do not have the exact same behaviour for the given input. For future work, we can explore the feasibility of this concept.

Bibliography

[1] Fides Aarts, Joeri de Ruiter, and Erik Poll. Formal models of bank cards for free. In *ICST Workshops*, pages 461–468. IEEE Computer Society, 2013.

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

[3] Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-style learning of NFA. In *IJCAI*, pages 1004–1009, 2009.

[4] Tsun S. Chow. Testing software design modeled by finite-state machines. *IEEE Trans. Software Eng.*, 4(3):178–187, 1978.

[5] Colin de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognit.*, 38(9):1332–1348, 2005.

[6] Paul Fiterau-Brostean, Ramon Janssen, and Frits W. Vaandrager. Combining model learning and model checking to analyze TCP implementations. In *CAV (2)*, volume 9780 of *Lecture Notes in Computer Science*, pages 454–471. Springer, 2016.

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

[8] Sebastian Junges and Jurriaan Rot. Learning language intersections. In *A Journey from Process Algebra via Timed Automata to Model Learning*, volume 13560 of *Lecture Notes in Computer Science*, pages 371–381. Springer, 2022.

[9] Ali Khalili and Armando Tacchella. Learning nondeterministic Mealy machines. In *ICGI*, volume 34 of *JMLR Workshop and Conference Proceedings*, pages 109–123. JMLR.org, 2014.

[10] Edi Muskardin, Bernhard K. Aichernig, Ingo Pill, Andrea Pferscher, and Martin Tappler. Aalpy: an active automata learning library. *Innov. Syst. Softw. Eng.*, 18(3):417–426, 2022.

[11] Muzammil Shahbaz and Roland Groz. Inferring Mealy machines. In *FM*, volume 5850 of *Lecture Notes in Computer Science*, pages 207–222. Springer, 2009.

[12] Frits W. Vaandrager, Bharat Garhewal, Jurriaan Rot, and Thorsten Wißmann. A new approach for active automata learning based on apartness. In *TACAS (1)*, volume 13243 of *Lecture Notes in Computer Science*, pages 223–243. Springer, 2022.