Implementation of Mealy machines and L# algorithm

THOMAS XU
(supervisor: ADRIEN POMMELLET)

Technical Report n^ooutput, June 2024
revision

We call active learning a set of methods aiming to create a model as accurate as possible of a black box to which we can submit queries. More formally, a learner tries to submit a model of an automaton recognizing a language that is only known by a teacher. To do so, the learner submits queries to the teacher: it can be output queries (What is the output from a given input) or equivalence queries (Does the submitted automata, called hypothesis, recognize exactly the target language, or is there a counter-example?). L# is an active learning algorithm bases on the notion of apartness[1] using Mealy Machines. Mealy machine is a good formalism to modelize systems that has inputs and outputs. Practically, it can be viewed as a graph where the nodes represent different states of a system, and each transition between states is associated with an output dependent on both the current state and the input. However, some reactive systems can generate a set of outputs from a given input, and these are modeled by incompletely specified machines. Incompletely specified Mealy machines (IGMMs) are used to represent systems designed from imprecise specifications, where in some cases there are multiple possible outputs for the same input. A specialization is therefore just an implementation (i.e., a choice of outputs in non-deterministic cases) derived from these specifications. The project involves implementing IGMMs (Incompletely Specified Mealy Machines) and the active learning of a specialization from an IGMM in C++.

On appelle apprentissage actif un ensemble de méthodes visant à créer un modèle aussi exact que possible d'une boıte noire à laquelle on peut soumettre des requêtes. Plus formellement, un élève cherche à proposer un modèle d'automate reconnaissant un langage qui n'est connu que d'un professeur. Pour ce faire, l'élève soumet des requêtes au professeur : il peut ainsi s'agir de requêtes d'appartenance (Est-ce qu'un mot soumis appartient au langage ou non ?) ou de requêtes d'équivalence (Est-ce que l'automate soumis, dit hypothèse, reconnaıt exactement le langage cible, ou existe t-il un contre-exemple ?). L# est un algorithme d'apprentissage actif qui se base sur la notion de distinguabilité[1] en utilisant les machines de Mealy. Une machine de Mealy est un bon formalisme pour modéliser des systèmes avec des entrées et des sorties. En pratique, on peut le voir comme un graphe où les nœuds représentent les différents états d'un système et où chaque transition entre les états est associée à une sortie dépendante à la fois de l'état actuel et de l'entrée. Cependant, certains systèmes réactifs peuvent à partir d'une entrée, générer un ensemble de sorties, on les modélise alors par des machines incomplètement spécifiées. Les IGMMs (incompletely specified mealy machine) servent à traduire des systèmes conçus à partir de spécifications imprécises, où dans certains cas il y a plusieurs sorties possibles pour une même entrée. Une spécialisation est donc juste une implémentation (i.e. un choix sur les sorties dans les cas non-déterministes) obtenue à partir de ces spécifications. Le projet prévoit l'implémentation des IGMM et l'apprentissage actif d'une spécialisation à partir d'une IGMM en C++.

[1] Frits Vaandrager, Bharat Garhewal, Jurriaan Rot, and Thorsten Wißmann. A new approach for active automata learning based on apartness. 2022.

Keywords
Automata, Mealy machine, Active learning, L#, DFA, IDFA, IGMM

Laboratoire de Recherche de l'EPITA
14-16, rue Voltaire – FR-94276 Le Kremlin-Bicêtre CEDEX – France
Tél. +33 1 53 14 59 22 – Fax. +33 1 53 14 59 13
thomas.xu@lre.epita.fr – http://www.lre.epita.fr/

Chapter 1

Introduction

Many real-world systems we use every day can be seen as reactive systems that interact with their environment. However, some of them hide their internal work which is a problem while trying to comprehend them. This emergence of black box systems has raised the interest to find effective methods to learn and understand them. One powerful formalism that can be used to represent black box systems is the Mealy machine[6].

Mealy machines are a type of finite state machine that from a certain state, takes an input and determines the corresponding output. Mealy machines can be introduced as an adequate formalism for modeling systems with inputs and outputs. We will see that Mealy machines can be given semantics in terms of runs just like finite automatas are interpreted as representations of formal languages. Most of the systems we are using in our life can be seen as systems that interact with their environment. They expose a set of actions considered as input and a Mealy machine processes sequences of inputs to produce an output. This versatility makes Mealy machines particularly well-suited for capturing the dynamics of various real-world systems.

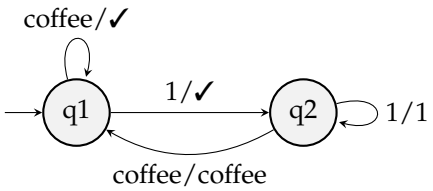

Figure 1.1: Mealy machine for a coffee machine

For instance, the machine 1.1 models a coffee machine, to which the user has to give a coin before pressing the button to get his coffee and when he gives more coins than needed then the machine will give them back.

Dana Angluin, a professor at Yale University, released a paper in 1987[1] that introduced the first algorithm for learning regular sets from queries and counterexamples. The paper made significant contributions to the field of machine learning and formal language theory. In Angluin's approach, referred to as the minimally adequate teacher (MAT), the process of learning is framed as a game. The learner's objective is to deduce a deterministic finite automaton (DFA) that represents an unknown regular language L by posing queries to a teacher. The learner has the option to ask two types of queries: membership queries, where they ask whether a specific word w belongs to L, and equivalence queries, where they inquire if the language recognized

by DFA H is equivalent to L.

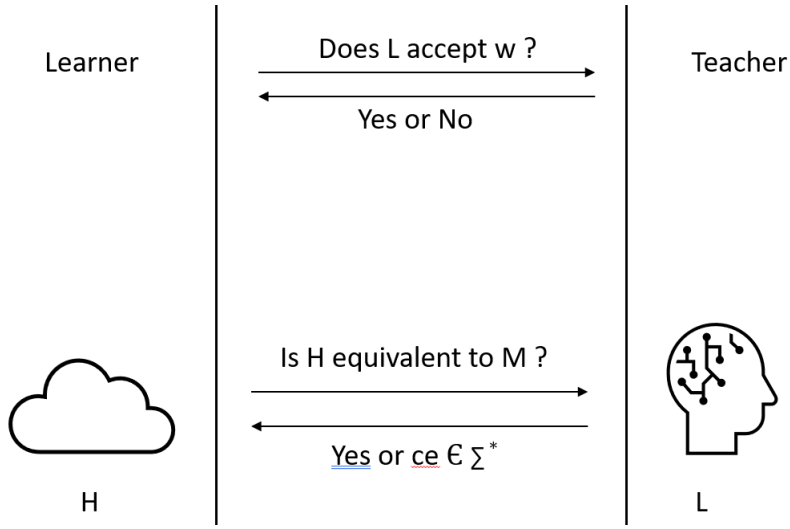

Figure 1.2: Minimal adequate teacher

His work triggered a lot of research on active automata learning, and among those, the L#[7] is a new algorithm that emerged as a leading approach for building a Mealy machine from a black box system. However, to describe more complex systems, it might not be enough which could be resolved if we manage to make it work on IGMMs which we will present later on.

In this work, we will study the fundamental concepts of Mealy machine and L# to understand how they can bring an answer to the challenge posed by black box systems. After that, we will show the results of its naive implementation[7] in C++, then we will try to explain our approach to extend it to IGMMs and the difficulties we faced doing that.

Chapter 2

Mealy Machine

A mealy machine is a strong formalism for modeling systems with inputs and outputs.

2.1 Properties

Definition 1 (Mealy machine) *: A Mealy machine is defined as a tuple* $M = \langle S, s0, \Sigma, \Omega, \delta, \lambda \rangle$ *where*

- *S is a finite nonempty set of states (be n =* $|S|$ *the size of the Mealy machine)*
- *s0 is the initial state*
- Σ *is a finite input alphabet*
- Ω *is a finite output alphabet*
- $\delta : S \times \Sigma \to S$ *is the transition function*
- $\lambda : S \times \Sigma \to \Omega$ *is the output function*

A Mealy machine processes sequences of inputs and produces outputs. In practice, we can see it as a graph where nodes represent the different states of a system for each input $I \in \Sigma$ the machine goes into a new state according to δ (s, I) and produces an output according to λ(s, I).

We write $s \xrightarrow{I/o} s'$ to denote that on input symbol I the Mealy machine moves from state s to state s′ producing output symbol o.
We will extend δ and λ to be able to process a word as their second component which gives us:
$\delta^* : S \times \Sigma^* \to S$ defined by $\delta^*(s, \epsilon) = s$ and $\delta^*(s, \alpha w) = \delta^*(\delta(s, \alpha), w)$
$\lambda^* : S \times \Sigma^* \to \Omega^*$ defined by $\lambda^*(s, \epsilon) = \emptyset$ and $\lambda^*(s, \alpha w) = \lambda(s, \alpha)\lambda(\delta^*(s, \alpha), w)$

Example 1 (Modeling coffee machine) *:*

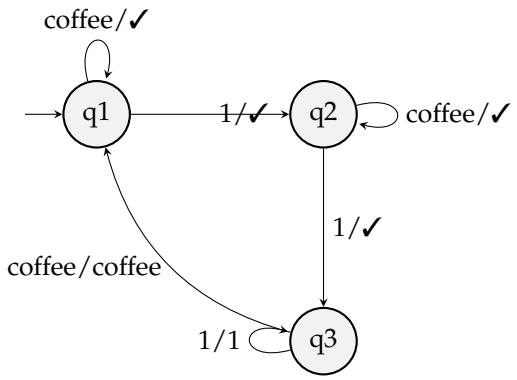

Figure 2.1: Model of a coffee machine M.

- S = $\{q1, q2, q3\}$
- s0 = q1
- Σ = $\{coffee, 1\}$
- Ω = $\{coffee, 1, ✓\}$

The mealy machine above models a coffee machine that sells coffee to customers for two coins. When the user asks for the coffee without putting in two coins then nothing happens, if he puts more than 2 coins then the machine gives them back and finally when he presses the button to get coffee with two coins then he gets his beverage and the machines returns to the initial state.

When a Mealy machine process a sequence of inputs $\alpha_1\alpha_2...\alpha_n$ its concrete behavior can be seen as a consistent sequence of alternating input and output symbols, where each input symbol is followed by its corresponding output symbol. We can define the semantic functional $[\![M]\!] : \Sigma^* \to \Omega^*$ by

$$[\![M]\!](w) = \lambda^*(s_0, w)$$

Definition 2 (Semantic of a state) *: The semantic of a state s is a map* $[\![s]\!] : \Sigma^* \to \Omega^*$ *defined by* $\lambda^*(s, w)$.

Example 2 (Runs of Mealy machines) *:*

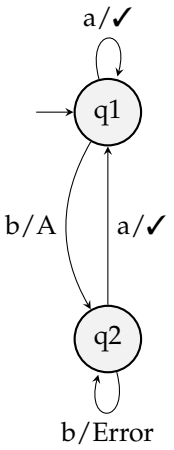

Figure 2.2: A random Mealy machine M2

The run $\langle a - a - b, \checkmark - \checkmark - A\rangle$ is in $[\![M2]\!]$.
But $\langle a - b, \checkmark\rangle$ is not because 1-coffee does not produce this output.

Definition 3 (Equivalence of words) *: For a mapping T: $\Sigma^* \to \Omega$, two words u and u' $\in \Sigma*$ are equivalent written $\equiv_T$, if and only if $\forall v \in \Sigma*$, uv and u'v are mapped to the same output by T:*

$$u \equiv_T u' \iff (\forall v \in \Sigma*, T(uv) = T(u'v))$$

Example 3 (Equivalence of words) *:*
With the figure 2.2 we can see that aab $\not\equiv_{M2}$ aabb but aab $\equiv_{M2}$ ab.

Definition 4 (Equivalence of Mealy machines) *: Two Mealy machines A and B are equivalent if their initial states s_0^A and s_0^B are equivalent: $s_0^A \approx s_0^B$. We say that two states s_0^A and s_0^B are equivalent if for each word u $\in \Sigma*$, the output produces from s_0^A and s_0^B is the same.*

Definition 5 (k-distinguishability) *: Two states s and s' $\in$ of a Mealy machine are k-distinguishable, if and only if there is a word $w \in \Sigma*$ of length k or shorter for which $\lambda(s, w) \neq \lambda(s', w)$.*

Example 4 (k-distinguishability) *: With figure 2.2 we can observe that q1 and q2 are k-distinguishable with I = b.*

Definition 6 (functional simulation) *: For Mealy machines M and N, a functional simulation $f : M \to N$ is a map f: $S^M \xrightarrow{i/o} S^N$ and $s \xrightarrow{i/o} s'$ implies $f(s) \xrightarrow{i/o} f(s')$.*

Functional simulation ensures that the behavior of M is replicated by N. Every sequence of state transitions and outputs in M must be mirrored by corresponding transitions and outputs in N under the mapping f.

Lemma 1 *: For a functional simulation* $f : M \to N$ *and* $s \in S^M$*, we have* $[\![s]\!] \subset [\![f(s)]\!]$

2.2 Motivation

When trying to build a Mealy machine from a black box, we have to adjust to a new constraint which is the fact that we have limited information on the machine we are trying to create. That is why we will use active learning.

Chapter 3

Active learning

L# is a new algorithm that emerged as a leading approach for building a Mealy machine from a black box system.

3.1 Teacher

In our case, we will use Mealy machines, which involves that determining whether a word should be recognized by the machine becomes irrelevant so does the membership query. Thus, we can change the membership query into an output query so that instead of asking whether a specific word belongs to the language, we can ask the teacher to provide the corresponding output for a given input sequence.

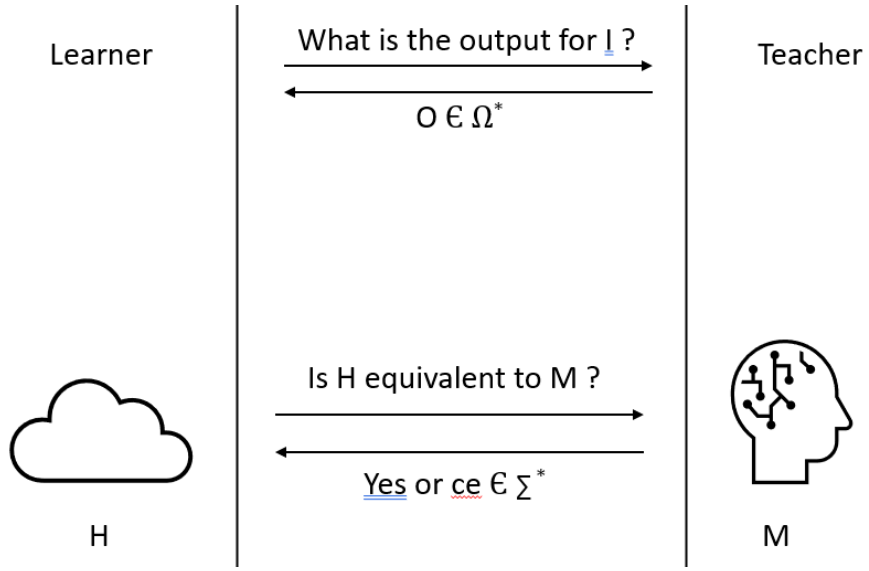

Figure 3.1: New teacher for L#

3.2 Apartness concept

Unlike Angluin's algorithm L^*[1] and its successors [2] which focuses on equivalence of observations, this paper presents L# a new and simple approach to active automata learning. L# adopts a different approach by aiming to establish "apartness," which is a constructive form of inequality. L# does not need auxiliary data structures such as observation tables like L^* and operates directly on an observation tree.
The notion of apartness is presented here [7] as the fundamental concept of this algorithm.

Definition 7 (Apartness) *: For a Mealy machine M, we state that two states q,p $\in S^*$ are apart (written a#b) if there is some $i \in \Sigma^*$ such as $[\![q]\!](i)$ and $[\![p]\!](i)$ are defined, and $[\![q]\!](i) \neq [\![p]\!](i)$. We say that i is the witness of q # p and write $i \vdash q \# p$. The concept of apartness is crucial for proving that nodes in the computation tree of a Mealy machine correspond to distinct states in the machine.*

Example 5 (Apartness between two states) *: Let us take the Mealy machine 2.1, we have:*

- $i \vdash q1 \# q2$ *with* $i = \langle 1, coffee \rangle$
- $i \vdash q1 \# q3$ *with* $i = \langle coffee \rangle$
- $i \vdash q2 \# q3$ *with* $i = \langle coffee \rangle$

3.3 Observation tree

Definition 8 (Observation tree) *: A mealy machine T is a tree if for each $s \in S^T$ there is a unique sequence $\sigma \in \Sigma^*$ for which $\delta^T(s_0^T, \sigma) = s$. We write access(s) for the sequence of inputs that lead to s. A tree T is a function simulation $f : T \to M$.*

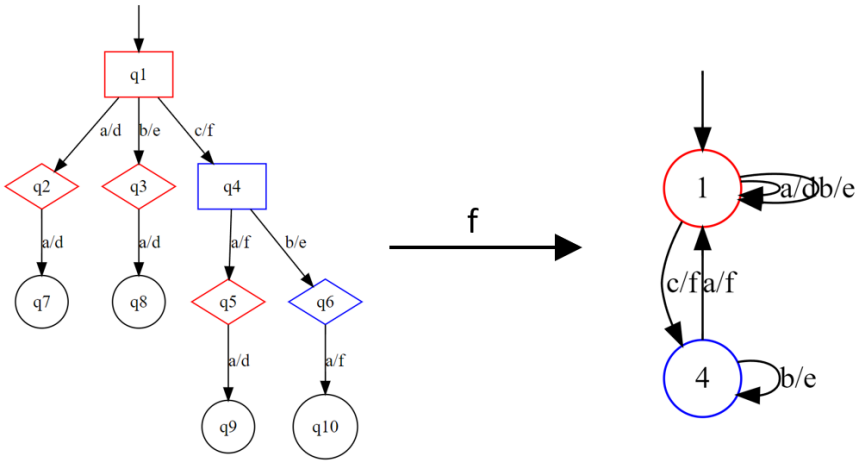

Figure 3.2: Observation tree for a Mealy machine

3.2 shows an observation tree on the left for the Mealy machine on the right where the functional simulation f is represented by the color of the states. Throughout the algorithm, the learner can build the observation tree with queries for the hidden Mealy machine but does not know the simulation function f. However, the learner can distinguish that some states of the tree should not be mapped to the same color by f. That is the reason apartness was introduced earlier. Indeed, when two states are apart in the observation tree T, the learner can deduce that they represent distinct states in the hidden Mealy machine.

3.4 Learning algorithm

3.4.1 Introduction

The objective of this algorithm is to determine a strategy for the learner in the given game.
In this learning game, there is a teacher who possesses a complete Mealy machine M which is able to answer queries 2.2 where the goal is to submit a hypothesis equivalent to the hidden machine for the learner.
L# operates directly on the observation tree T = $\langle S, s_0, \Sigma, \delta \rangle$. The observation tree contains the results of every query sent to the teacher. Every node can be seen as a state of the system under learning and each transition represents the output from a state applying an input. That is the reason why, in practice, the tree is just a Mealy machine itself and close to an actual hypothesis that can be submitted to the teacher, making the whole process simpler.

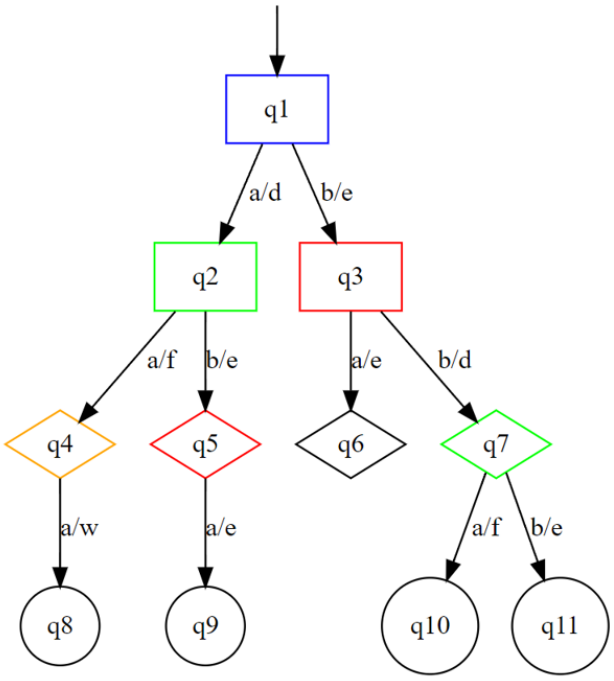

Figure 3.3: Observation tree T

The nodes of the observation tree are subdivided into three sets:

- The basis $B \in S^T$ (represented by a rectangle shape in 3.3) contains states that have been fully identified i.e. the learner realized that they represent distinct states in the hidden machine. Initially, B:= $\{s_0\}$ and evolves during the execution forming a subtree of T. The main characteristic of these states is that each of them is apart from the others.
- The frontier $F \in S^T$ (represented by a diamond shape in 3.3) that contains the set of immediate non-basis successors of basis states: $F := \{s' \in S \backslash B \mid \exists s \in B, i \in \Sigma\text{: } s' = \delta(s, i)\}$.
  The next nodes that will be added to the basis states are included in the frontier.
- The remaining states (represented by a circle shape in 3.3): S $\backslash (B \cup F)$.

Within the frontier, we can categorize the states into three groups again:

- Isolated states: states of the frontier that are apart from every state in the basis (a state that has a color not present in the basis state i.e. q4 in 3.3).
- Identified states: states of the frontier that are apart from every state in the basis except one (a state that shares the same color with a basis state i.e. q5 and q7 in 3.3).
- The remaining ones (states in black in 3.3).

In the beginning, T consists of a single initial state q_0^T with no transition. After every $OutputQuery(\sigma)$ that generates $o \in \Omega^*$ during the execution, $\langle \sigma, o \rangle$ will be added to T and the same applies to every negative response to EquivalenceQuery which leads T to be extended. With T's extension, the learner can deduce more apartness relations which is the key point of the algorithm.

3.4.2 Hypothesis

Frequently, the learner can submit a hypothesis H built from the knowledge accumulated in T. The states of H, $S^H = F^T$. Its transitions are created with transitions in T that start on a basis state, if its destination is a basis state then the hypothesis will keep it as it is, and otherwise, it means that it goes onto a frontier state f, in that case, the learner resolves it by finding a basis state identified to f for which he conjectures that they are equivalent in the hidden Mealy machine.

Definition 9 (Hypothesis) *: Let T be an observation tree with basis B and frontier F.*

1. *A mealy machine H contains the basis if $S^H = B$ and $\delta^H(q_0^H, access(q)) = q$ for all $q \in B$.*
2. *A hypothesis is a complete Mealy machine H containing the basis such that $q \xrightarrow{i/o'} p$ in H ($q \in B$) and $q \xrightarrow{i/o} p$ in T implies $o = o'$ and $\neg(p \# p')$ in T, so we need the output for each state in the basis for each i in Σ.*
3. *A hypothesis H is consistent if there is a functional simulation $f : T \to H$ implying that each state in the frontier is identified to exactly one state in the basis.*
4. *For a Mealy machine H containing the basis, an input sequence $\sigma \in \Sigma^*$ is said to lead to a conflict if $\delta^T(q_0^T, \sigma) \# \delta^H(q_0^H, \sigma)$ (in T).*

These rules ensure that the hypothesis H is consistent with the tree T so the learner won't submit an equivalence query when it is not needed to extend the tree.

Lemma 2 *For an observation tree T, if F has no isolated states then there exists a hypothesis H for T. If B is complete and all states in F are identified then the hypothesis is unique.*

Theorem 1 *Suppose T is an observation tree for the hidden Mealy machine M such that B is complete, all states in F are identified and $|B|$ is the number of equivalence classes of M then $H \approx M$ for the unique hypothesis H.*

Definition 10 (Complete basis) *: The basis B is complete if each state in B has a transition for each input in Σ.*

Definition 11 (Weak transitivity) *: In a Mealy machine M,*
$\sigma \vdash r\#r' \wedge \delta(q, \sigma)$ is defined $\implies$ $r\#q \vee r'\#q$ for all r, r', $q \in S^M$ and $\sigma \in \Sigma^*$

3.4.3 Main loop of L#

Algorithm 1 L# algorithm

while true **do**
  if for some $q \in F$, q is isolated **then**
    S $\leftarrow S \cup \{q\}$ (R1)
  end if
  if for some $q \in B, i \in \Sigma$, $\delta^T(q, i)$ is not defined **then**
    OutputQuery(access(q), i) (R2)
  end if
  if for some $q \in F$, $r, r' \in B$, $\neg(q\#r)$, $\neg(q\#r')$, $r \neq r'$ **then**
    $\sigma \leftarrow$ witness of $r\#r'$
    OutputQuery(access(q), σ) (R3)
  end if
  if F has no isolated states and basis B is complete **then**
    $H \leftarrow$ BuildHypothesis (R4)
    $(b, \sigma) \leftarrow$ CheckConsistency(H)
    if b = true **then**
      $(b, \sigma) \leftarrow$ EquivalenceQuery(H)
      if b = true **then**
        return H;
      end if
      $\sigma \leftarrow$ shortest prefix p such that $\delta^H(q_0^H, \sigma) \neq \delta^T(q_0^T, \sigma)$ (in T)
    end if
    ProcessCounterExample(H, σ)
  end if
end while

(R1) If F contains an isolated state, then we discovered a new state not contained in B, thus we move it from F to B.

(R2) When a state $q \in B$ has no transition for an input $i \in \Sigma$, the learner will send an OutputQuery(access(q)i) to generate the successor and extend the frontier F.

(R3) When $q \in B$ is identified to more than one state in the basis $r, r' \in B$, then we can use the weak transitivity to grow our apartness relations. For this to happen, we take the witness σ of the basis states and send an OutputQuery(access(q)σ) to extend the observation. Consequently, q # r or q # r′.

(R4) When F has no isolated state and B is complete, BuildHypothesis will create the hypothesis. If H is not consistent then we get a free conflict σ. Otherwise, we pose an EquivalenceQuery for H, if the hypothesis is equivalent to the hidden machine, we stop L# else, we get a counter-example p.

3.4.4 Consistency checking

Sometimes, the hypothesis submitted is not consistent with the tree T in the sense of a functional simulation $T \rightarrow H$. If that happens, it means that a state the state in the frontier that we go through with our witness is not identified to the correct basis state which leads to an error. We can check in linear time O(size(T)) if a functional simulation exists. To do that, we can use a BFS of the cartesian product of T and H.

Algorithm 2 Consistency checking

$Q \leftarrow$ new queue $\subseteq B \times B$
$enqueue(Q, (s_0^T, s_0^H))$
while $(q, r) \leftarrow dequeue(Q)$ **do**
  if q # r **then**
    return (false, access(q))
  end if
  for all $q \xrightarrow{i/o} p$ in T **do**
    $enqueue(Q, (p, \delta^h))$
  end for
end while
return true

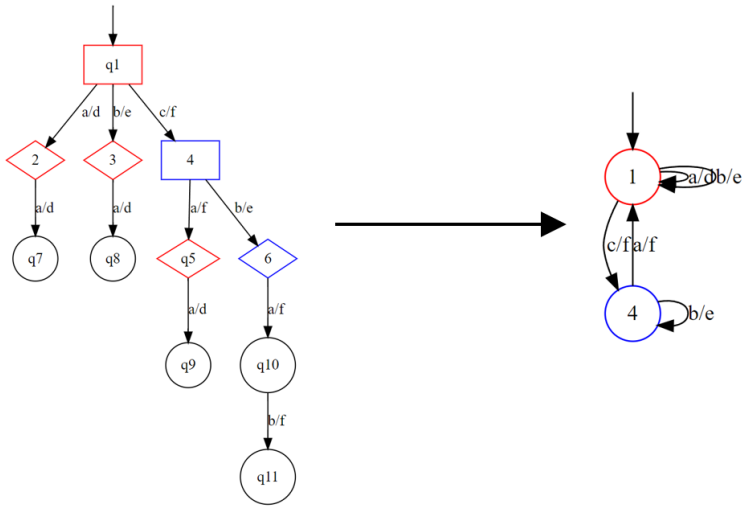

Figure 3.4: Non-consistent hypothesis

Example 6 *In the figure 3.4, the hypothesis is not consistent for the sequence* $\sigma = \langle c, b, a, b \rangle$ *which does not produce the same output in T and H.*

3.4.5 Process counter example

The L* algorithm [1][6] analyze a counter-example of length m in (m) queries. That being said, if the teacher returns a very long counter-example, this will dominate the learning process. However, this paper [5] improves counter-example processing using binary search which needs only $O(log(m))$ queries and the same trick will be used here.

Algorithm 3 Process-counter-example(H, $\sigma \in \Sigma^*$)

$q \leftarrow \delta^H(s_0^H, \sigma)$
$r \leftarrow \delta^T(s_0^T, \sigma)$
if $r \in B \cup F$ **then**
  return
end if
$p \leftarrow$ unique prefix of σ with $\delta^T(s_0^T, p) \in F$
h = $\lfloor \frac{|p|+|\sigma|}{2} \rfloor$
$\sigma_1 \leftarrow \sigma[1..h]$
$\sigma_2 \leftarrow \sigma[h+1..|\sigma|]$
$q' \leftarrow \delta^H(s_0^H, \sigma_1)$
$r' \leftarrow \delta^T(s_0^T, \sigma_1)$
$w \leftarrow$ witness for q # r
OutputQuery(access(q') σ_2 w)
if q′ # r′ **then**
  Process-counter-example(H, σ_1)
else
  Process-counter-example(H, access(q')σ_2)
end if

The purpose of this algorithm is to extend T such as H will never be a hypothesis for T again. Until r is within $B \cup F$ meaning that the conflict r # q is obvious and H is not a hypothesis, we will use binary search to get σ_1. σ_1 is p to which we added half of the sequence outside of the frontier and the learning will try to catch the shortest sequence σ_1 that leads to a conflict and use an OutputQuery on it. This leads us to reduce the number of transitions outside the frontier by half.

Chapter 4

Implementation of L# for Mealy Machines

4.1 Work done

We implemented everything mentioned earlier in Pallas using C++17, an active and passive learning library of Adrien Pommelet, a researcher at LRDE. This library already implemented a deterministic automata class which made the work a bit easier. We wrote a class for deterministic Mealy machine which is able to read and parse a file to create our Mealy machine, execute operations on them, and export it in a file using dot format. We also completed the implementation of the active learning part from the teacher to the actual algorithm L#.

4.2 Result

In the paper introducing L#[7], they submitted their benchmark on models from [3] so we can compare ours to theirs. L#'s complexity is measured by the number of queries sent to the teacher.

Models	n	k	OutputQuery (Ref | Ours)	EquivalenceQuery (Ref | Ours)
10_learnresult_Mastercard	6	14	229 | 197	4| 4
Volksbank_learnresult_MAESTRO	7	14	342 | 346	5 | 5

Table 4.1: Benchmark (n is the number of states and k is the number of input symbols)

The results are pretty similar, we can suppose that the minor differences probably come from the teacher's responses to the queries. Those results are satisfactory but around what we expected since we implemented their algorithm.

Chapter 5

IGMM

Now that we are able to modelize systems with inputs and deterministic outputs we would like to extend our algorithm of non-deterministic systems.

5.1 Definitions[4]

Given a set of propositions (i.e., Boolean variables) X, let B^X be the set of all possible valuations on X, and let 2^{B^X} be its set of subsets. Any element of 2^{B^X} can be expressed as a Boolean formula over X. The negation of proposition p is denoted $\overline{p}$. We use T to denote the Boolean formula that is always true, or equivalently the set B^X, and assume that X is clear from the context

5.1.1 Incompletely Specified Generalized Mealy Machine (IGMM)

An IGMM is a tuple $M = (I, O, Q, q_{\text{init}}, \delta, \lambda)$, where:

- I is a set of input propositions.
- O is a set of output propositions.
- Q is a finite set of states.
- q_{init} is an initial state.
- $\delta : (Q \times B^I) \to Q$ is a partial transition function.
- $\lambda : (Q \times B^I) \to 2^{B^O} \setminus \{\emptyset\}$ is an output function such that $\lambda(q, i) = \top$ when $\delta(q, i)$ is undefined.

In this context, B^I represents the set of all possible valuations over input propositions I, and B^O represents the set of all possible valuations over output propositions O. The output function λ can return multiple possible output valuations for a given state and input valuation, representing the non-deterministic nature of the machine.

If the transition function δ is defined for all possible state-input pairs, the IGMM is said to be input-complete.

In the figure 5.1.1 we can see that it is basically a Mealy machine where for each entry for each state a set of outputs is produced.

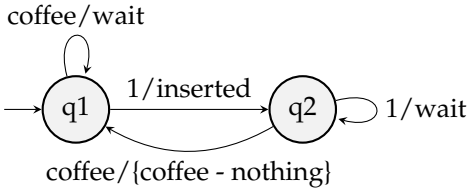

Figure 5.1: IGMM for a broken coffee machine

5.1.2 IGMM Specialization

A specialization of an IGMM is a fully specified implementation derived from the original IGMM by resolving non-deterministic outputs to single deterministic outputs. Formally, a specialization of an IGMM $M = (I, O, Q, q_{\text{init}}, \delta, \lambda)$ is another IGMM $M' = (I, O, Q', q'_{\text{init}}, \delta', \lambda')$ where:

- $Q' \subseteq Q$
- $q'_{\text{init}} \in Q'$
- δ' is a total function.
- λ' maps each state-input pair to a single output valuation B^O.

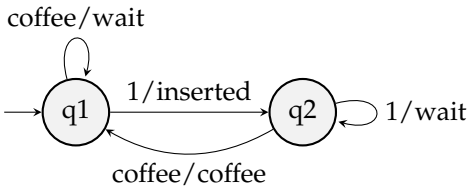

Figure 5.2: Specialized IGMM for a broken coffee machine

5.2 Problems

In order to extend L# to IGMMs, we faced two major theorical issues in order for it to converge with as less queries to the oracle as possible.

5.3 New Apartness

In order to put the proper states in the basis and to identify the correct ones in the frontier to them, we had to change the definition of apartness and we concluded that we had two solutions.

Definition 12 (Weak Apartness) *We state that two states q1 and q2 are apart if there is an input i in I^* such as the outputs $\lambda(q1, i)$ and $\lambda(q2, i)$ could be different depending on the specialization we choose.*

Definition 13 (Strong Apartness) *We state that two states q1 and q2 are apart if there is an input i in I^* such as the outputs $\lambda(q1, i) \neq \lambda(q2, i)$ for every specialization, we can also see it as their symmetric difference $L1 \triangle L2 = L1 \setminus L2 \cup L2 \setminus L1 \neq \emptyset$.*

This makes hard for us to determine which state will be in the basis since every state might be incomplete.

5.4 Specialization

Here again, we had the same issue where there were multiples ways to specialize our IGMM when we built the hypothesis with the tree.

1. We try to minimize the number of states: everytime a state can be fused with another one then we will do so (we think this one the is best so far).
2. We try to maximize the number of states: whenever a specialization allows us to put a new state in the basis then we do so.
3. Randomly or more precise methods to define.

These two problems come from the fact we have to deal with a non-deterministic observation tree instead of a deterministic one. One way to begin our study was to start to learn how to implement L# on non-deterministic trees.

Chapter 6

L# on DFAs

Definition 14 (IDFA) *A IDFA (Incomplete deterministic finite complete automaton) is a tuple* $M = \langle Q, q0, \Sigma, F, \delta \rangle$ *where:*

1. Σ *is the alphabet.*
2. Q *is a finite set of states.*
3. $q0 \in Q$ *a subset of initial states.*
4. $F \subseteq Q$ *a set of accepting states.*
5. $R \subseteq Q$ *a set of non-accepting states.*
6. $\delta : Q \times \Sigma \to Q$ *the transition relation.*

Definition 15 (DFA) *A DFA is simply an IDFA where* $F \cup R = Q$*. It is an IDFA where the acceptance of each state is known.*

While it does not seem like it has anything to do with our original purpose, the observation tree for a DFA is an incomplete DFA (IDFA) because when we insert a sequence in the tree with its acceptance, the acceptances of the states before the last one are unknown making the tree incomplete.

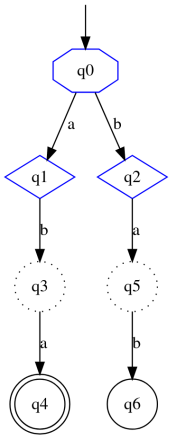

Figure 6.1: Incomplete Observation tree for DFAs

Initially, we wanted to experiments multiples things on those trees to get as much hints as we could for IGMMs trees ultimately but changing the apartness definition alone was pretty much enough.

Definition 16 (Apartness for IDFA) *In an IDFA, two states q0 and q1 are appart if the status of their acceptance differ (known / unknown) or if both are known and the acceptance differ.*

The hypothesis we make is a DFA so there is no state with unknown acceptance within it.

Definition 17 (Apartness for IDFA and hypothesis) *For an IDFA T and a DFA H, two states $q_t \in T$ and $q_h \in H$ are appart if the acceptance of q_t is unknown or if their acceptance differ.*

Those redefinitions of the apartness rules will have an impact on L# main loop's rule 1, 3 and 4.

The thing that I thought would have been more of an issue was the fact that since the oracle had the form of a DFA, it was forced to answer false for the acceptance of sequences that were not originally in the DFA and so L# would not work as intended (minimize the result) but it did after some work. It finishes in a polynomial number of queries depending on the number of states and the size of the alphabet which we could expect.

Chapter 7

Conclusion

In conclusion, Mealy machines and the L# algorithm look promising in order to understand black box systems. The Mealy machines by their ability to represent real-world systems and L# as an active learning approach, has proven effective in inferring the hidden Mealy machine by interacting with the black box system. While this implementation enables the understanding of black box systems, it can still be improved. That is why we thought that its extension to IGMMs would be a great feat but due to time constraints it has not been done yet.

Copying this document

Chapter 8

Bibliography

[1] Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106. (pages 3, 10, and 16)

[2] Howar, F., Bauer, O., Merten, M., Steffen, B., and Margaria, T. (2011). The teachers' crowd: The impact of distributed oracles on active automata learning. In Hähnle, R., Knoop, J., Margaria, T., Schreiner, D., and Steffen, B., editors, *Leveraging Applications of Formal Methods, Verification, and Validation - International Workshops, SARS 2011 and MLSC 2011, Held Under the Auspices of ISoLA 2011 in Vienna, Austria, October 17-18, 2011. Revised Selected Papers*, volume 336 of *Communications in Computer and Information Science*, pages 232–247. Springer. (page 10)

[3] Neider, D., Smetsers, R., Vaandrager, F. W., and Kuppens, H. (2018). Benchmarks for automata learning and conformance testing. In Margaria, T., Graf, S., and Larsen, K. G., editors, *Models, Mindsets, Meta: The What, the How, and the Why Not? - Essays Dedicated to Bernhard Steffen on the Occasion of His 60th Birthday*, volume 11200 of *Lecture Notes in Computer Science*, pages 390–416. Springer. (page 17)

[4] Renkin, F., Schlehuber-Caissier, P., Duret-Lutz, A., and Pommellet, A. (2022). Effective reductions of Mealy machines. In *Proceedings of the 42nd International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE'22)*, volume 13273 of *Lecture Notes in Computer Science*, pages 114–130. Springer. (pages 18 and 25)

[5] Rivest, R. L. and Schapire, R. E. (1989). Inference of finite automata using homing sequences (extended abstract). In Johnson, D. S., editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 411–420. ACM. (page 16)

[6] Steffen, B., Howar, F., and Merten, M. (2011). Introduction to active automata learning from a practical perspective. In Bernardo, M. and Issarny, V., editors, *Formal Methods for Eternal Networked Software Systems - 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures*, volume 6659 of *Lecture Notes in Computer Science*, pages 256–296. Springer. (pages 3 and 16)

[7] Vaandrager, F. W., Garhewal, B., Rot, J., and Wißmann, T. (2021). A new approach for active automata learning based on apartness. *CoRR*, abs/2107.05419. (pages 4, 10, and 17)

Contents