Learning Discrete Timed Automata

Kush Grover

Chennai Mathematical Institute

Supervisor

B. Srivathsan

In partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

May 14, 2018

To my father.

Acknowledgements

I would first like to thank my supervisor B. Srivathsan. He has been very helpful throughout the time we worked on this thesis. He was always there when I was stuck at some point to get me out. He steered me in the right the direction whenever he thought I needed it.

I would also like to thank M. Srivas for nurturing me as a researcher. I did an internship under him which introduced me to the research in theoretical computer science and made me choose an academic career.

I would like to thank my friends Akshay Naik, Arijit Shaw, Pankaj Pundir, Rajarshi Roy, Ritam Raha and Vasudha Sharma for their friendship, talks we had, parties we shared, and games we played. I would like to thank them for all the support that they gave me whenever I needed it and also making my life in CMI very enjoyable.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Abstract

Learning a model has been a popular problem in recent times. Dana Angluin started learning with DFAs. People have tried to learn a lot of different models recently and we also want to learn timed systems, in particular we want to learn some subclass of timed automata. We give an active learning algorithm for discrete deterministic timed automata with resets on every transition.

Contents

Chapter 1

Introduction

In this thesis we describe an active learning algorithm for a subclass of timed automata. Timed automata [1] are finite state machines with some clocks. The transitions can have guards over clocks and the transition can only be taken only if the guard is satisfied. There can be resets of clocks on the transitions which sets the value of those clocks to zero. Timed automata can be used to model and reason about some real-time systems like network protocols, business processes, reactive systems, etc. It can be very difficult to model these as timed automaton by hand therefore we want to identify them automatically.

We want to model these real-time systems using time explicitly. We can model these timed behavior implicitly in a finite state machine but that model can be exponentially large because numbers use a binary representation of time while states use a unary representation of time. Sicco Verwer et al. gave a passive learning algorithm for such systems [2]. But we want to learn timed systems in an active manner like the learning of regular sets as described in [3].

There have been some work related to learning timed systems in addition to Sicco Verwer's passive learning algorithm for simple timed automaton [2]. O. Grinchtein et al. gave an algorithm for learning event-recording automata where there is a clock for every alphabet and it gets reset whenever you see that alphabet [4]. But this algorithm is very convoluted because there are too many degrees of freedom. Bengt Jonsson gave an algorithm to learn mealey machines with timers that capture timing behavior of common network protocols [5]. There is one more paper on learning timed automaton via genetic programming by Martin Tappler et al. but this is also a passive learning algorithm [6].

If we consider that there is only one clock that gets reset on every transition then we can think of it as a symbolic automaton [7]. A symbolic automaton is a finite state machine such that the transitions carry predicates over them and there is already an algorithm for learning symbolic automata[8], we modify that algorithm to work for this subclass of timed automata. This is also the subclass of timed automata that Sicco Verwer refer to as simple timed automata. We call this class r-DDiTA which is short for deterministic discrete timed automata with resets on every transition. We show that the class discrete

timed automata (DiTA) is equivalent to r-DDiTA by converting a discrete timed automata to an automaton in class r-DDiTA. We gave an algorithm to minimize an r-DDiTA and proved that this automaton will be unique. We also give an algorithm to learn a discrete timed automaton as an r-DDiTA and prove that this is the unique minimal automaton in r-DDiTA that accepts the same language.

This thesis is organized as follows. In preliminaries first we describe the L^* algorithm to learn DFAs by D. Angluin [3] and run it on an example. After that we describe symbolic automata [7]. In the next chapter we show how you can learn a deterministic timed automata using L^* i.e. learn a timed language as an untimed language which capture the same behaviour. In the fourth chapter, we first show that the class DiTA is equivalent to the class r-DDiTA then we give an algorithm to learn r-DDiTA and prove that the automaton you get from this algorithm is minimal. We conclude the thesis in the final chapter with some ideas for further work.

Chapter 2

Preliminaries

2.1 Learning Regular Sets

General setting for learning is that there is a *teacher* and a *learner*, *learner* wants to learn some model with the help of the *teacher*. Learning can be of two types, active and passive. In passive learning two sets $\mathcal{P}$ and $\mathcal{N}$ are given to the learner of positive and negative examples respectively. The learner has to predict a model that satisfies the positive examples and does not satisfy the negative examples. And, in active learning the learner ask queries on the fly based on the current situation. As described in {ref angluin}, for learning a regular set L the *learner* can ask the *teacher* two types of questions. First type of query is *membership query*, if some string is in U or not. Second type of query *learner* can ask is *equivalence query* by giving the *teacher* an automaton and asking if language of this automaton is equivalent to L. *Teacher* responds to *membership queries* in **yes** or **no** and for *equivalence queries* teacher responds in **yes** or gives a counter-example w in symmetric difference of the language L and the automaton given by the *learner*. A *teacher* is called *minimally adequate* if it answers both types of queries correctly.

2.1.1 Observation Table

Let Σ be a finite alphabet. A set is *prefix-closed* if and only if every prefix of every member of the set is also a member of the set. *Suffix-closed* is defined analogously. For a language $L \in \Sigma^*$, we define a *characteristic function* $f : \Sigma^* \to \{+, -\}$ such that $f(u) = +$ if $u \in L$ and $f(u) = -$, otherwise.

Definition 2.1.1. (Observation Table). An *observation table* $T = (\Sigma, S, R, E, f)$ is a 5-tuple where,

- S is a non-empty finite prefix-closed set of strings,
- $R = S \cdot \Sigma$,
- E is a non-empty finite suffix-closed set of strings,
- $f : (S \cup R) \cdot E \to \{+, -\}$ is a *characteristic function*.

Rows labelled by elements of S are the candidates for states of the automaton being constructed, and columns labelled by elements of E correspond to distinguishing experiments for these states. Rows labelled by elements of R are used to construct the transition function.

Definition 2.1.2. (Closed, Consistent Observation Table). An *Observation table* is

- *Closed* if for each $t \in R$, $\exists s \in S$ such that $row(t) = row(s)$, where $row(u)$ is the row corresponding to $u \in S \cup R$.
- *Consistent* if $\forall s_1, s_2 \in S$ such that $row(s_1) = row(s_2)$ then $\forall a \in \Sigma$, $row(s_1 \cdot a) = row(s_2 \cdot a)$.

For a *closed*, *consistent* observation table T, we define a corresponding finite automaton $M(T)$ over the alphabet Σ, with state set Q, initial state q_0, accepting states F, and transition function δ as follows: $Q = \{row(s) : s \in S\}$, $q_0 = row(\epsilon)$, $F = \{row(s) : s \in S$ and $T(s) = 1\}$, $\delta(row(s), a) = row(s \cdot a)$.

Theorem 2.1.1. If T is a *closed, consistent observation table*, then the acceptor $M(T)$ is consistent with the table T. Any other acceptor consistent with T but not equivalent to $M(S, E, T)$ must have more states.

For the proof of the above theorem, look at { ref angluin}.

2.1.2 L^* Algorithm

Here we will briefly describe the L^* algorithm. Learner maintains an observation table $T = (\Sigma, S, R, E, f)$ at all times. Initially $S = E = \{\epsilon\}$, $R = \{a \in \Sigma\}$ and f is determined by membership queries for ϵ and for each $a \in \Sigma$.

In the main loop, first check if the current table T is *closed* and *consistent*. If T is not *consistent*, then find s_1 and s_2 in S, e in E, and a in Σ such that $row(s_1) = row(s_2)$ but $f(s_1 \cdot a \cdot e)$ is not equal to $f(s_2 \cdot a \cdot e)$. Add the string $a \cdot e$ to E and extend T to $(S \cup R) \cdot (a \cdot e)$ by asking *membership queries* for missing elements. Note that in this case, s_1, s_2, e, and a must exist, and since e is in E, E remains suffix-closed when $a \cdot e$ is added. If T is not *closed*, then find s_1 in S and a in Σ such that $row(s_1 \cdot a)$ is different from $row(s)$ for all s in S. Now, add the string $s_1 \cdot a$ to S and extend T to $(S \cup S \cdot A) \cdot E$ by asking membership queries for missing elements. Note that in this case, s_1 and a must exist, and since s_1 is in S, S remains prefix-closed when $s_1 \cdot a$ is added.

These two operations are repeated as long as the table T is not *closed* and *consistent*. When T is found to be *closed* and *consistent*, learner asks an *equivalence query* of $M(T)$. The teacher replies either with **yes**, signifying that the automaton accepts the language L, or with a counterexample w. If the *teacher* replies with **yes**, the algorithm terminates with output $M(T)$. If the *teacher* replies with a counterexample w, then w and all its prefixes are added to the set S and the function T is extended to $(S \cup R) \cdot E$ by the means

of membership queries for the missing entries. The main loop of the algorithm is then repeated for this new observation table T.

- *Correctness*: To see that L^* is correct, note that if the *teacher* is *minimally adequate* then if L^* ever terminates its output is clearly an acceptor for the unknown regular set L being presented by the Teacher.
- *Termination*: Following lemma proves the termination of the algorithm.

Lemma 2.1.1. Let T be an observation table. Let n denote the number of different values of $row(s)$ for s in S. Any acceptor consistent with T must have at least n states.

Algorithm 1 L^*

$learned = false$;
Initialize the observation table T with $S = E = \{\epsilon\}, R = \{a \in \Sigma\}$;
Ask membership queries on ϵ and for each $a \in \Sigma$ to fill T
repeat
  if T is not *closed* or *consistent* **then**
    if T is not *consistent* **then**
      find s_1 and s_2 in S, $a \in \Sigma$, and $e \in E$ such that
      $row(s_1) = row(s_2)$ and $f(s_1 \cdot a \cdot e) \neq f(s_2 \cdot a \cdot e)$
      add $a \cdot e$ to E
      extend T to $(S \cup R) \cdot E$ using membership queries.
    end if
    if T is not *closed*. **then**
      find $s_1 \in S$ and $a \in \Sigma$ such that $row(s_1 \cdot a) \neq row(s)$ for all $s \in S$
      add $s \cdot a$ to S
      extend T to $(S \cup R) \cdot E$ using membership queries.
    end if
  end if
  Let $M = M(T)$, Ask equivalence query with M.
  if Teacher replies with counter-example w. **then**
    add w and all its prefixes to S
    extend T to $(S \cup R) \cdot E$ using membership queries.
  else
    $learned = true$
  end if
until $learned$

Theorem 2.1.2 (Canonical Automaton)**.** L^* returns the minimal automata that accepts the unknown language L.

We will not prove this theorem here, you can look at [3] for the proof.

Example 2.1.1. Consider the automaton shown in figure 2.1. Language accepted by this DFA is $(ab)^+$. We will show the run of L^* on this example.

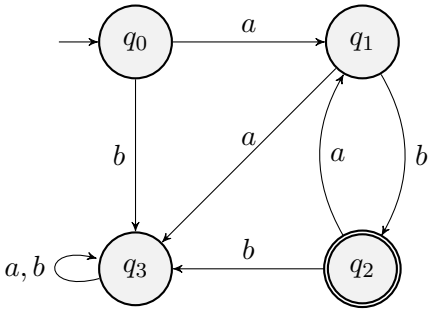


Figure 2.1: Finite Automata

2.2 Timed Automata

2.2.1 Timed Word and Timed Languages

Definition 2.2.1. (Timed Word) A *timed word* $\bar{w}$ over an alphabet Σ is a finite sequence $(a_0, t_0)(a_l, t_l)...(a_n, t_n)$ of symbols $a_i \in \Sigma$ that are paired with nonnegative real numbers $t_i \in \mathbb{R}^{\geq 0}$ such that the sequence $\bar{t} = t_0 t_1 ... t_n$ of time-stamps is nondecreasing (i.e., $t_i \leq t_{i+l}$ for all $0 \leq i < n$).

Without loss of generality it may be assumed that $t_0 = 0$. Sometimes we denote the timed word $\bar{w}$ by the pair $(\bar{a}, \bar{t})$, where $\bar{a} \in \Sigma^*$ is an untimed word over Σ.

Definition 2.2.2. (Timed Language) A *timed language* over the alphabet Σ is a set of timed words over Σ.

Timed automata are finite-state machines whose transitions are constrained with timing requirements so that they accept (or generate) timed words (and thus define timed languages); they were proposed in [1] as an abstract model for real-time systems with finite control. The finite control of a timed automaton consists of a finite set of locations and a finite set of real-valued variables called clocks. Each edge between locations specifies a set of clocks to be reset (i.e., restarted). The value of a clock always records the amount of time that has elapsed since the last time the clock was reset: if the clock z is reset while reading the ith symbol of a timed input word $(\bar{a}, \bar{t})$, then the value of z while reading the jth symbol, for $j > i$, is $t_j - t_i$ (assuming that the clock z is not reset at any position between i and j). The edges of the automaton put arithmetic constraints on the clock values; the automaton control may proceed along an edge only when the values of the clocks satisfy the corresponding constraints. Each clock of a timed automaton, therefore, is a real-valued variable that records the time difference between the current input symbol and a previous input symbol, namely, the input symbol on which the clock was last reset. This association between clocks and input symbols is determined dynamically by the behavior of the automaton. Formally,

Definition 2.2.3. (Timed Automata) A *timed automaton* (TA) is a tuple $A = (Q, \Sigma, X, T, q_0, F)$ where

T_0	ϵ
ϵ	$-$
a	$-$
b	$-$

T_1	ϵ	b
ϵ	$-$	$-$
a	$-$	$+$
ab	$+$	$-$
b	$-$	$-$
aa	$-$	$-$
aba	$-$	$+$
abb	$-$	$-$

T_2	ϵ	b	ab
ϵ	$-$	$-$	$+$
a	$-$	$+$	$-$
b	$-$	$-$	$-$
ab	$+$	$-$	$+$
ba	$-$	$-$	$-$
bab	$-$	$-$	$-$
bb	$-$	$-$	$-$
aa	$-$	$-$	$-$
aba	$-$	$+$	$-$
abb	$-$	$-$	$-$
baa	$-$	$-$	$-$
$baba$	$-$	$-$	$-$
$babb$	$-$	$-$	$-$

Figure 2.2: Observation tables for example 2.1.1

.

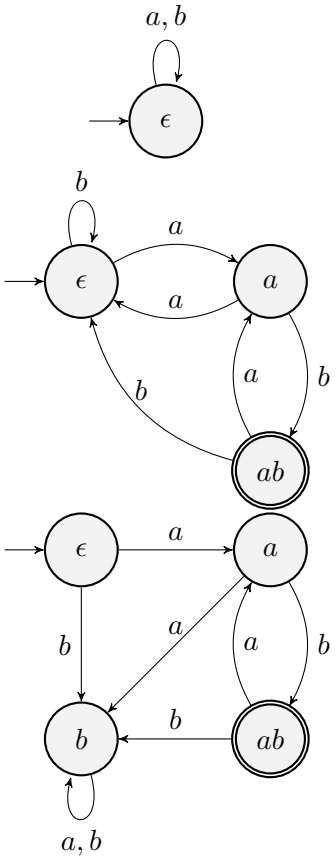


Figure 2.3: Hypothesis automaton for example 2.1.1

.

- Q is a finite set of states,
- Σ is a finite alphabet,
- X is a finite set of clocks,
- $q_0 \in Q$ is the initial state,
- $F \subseteq Q$ is a set of accepting states,
- $T \subseteq Q \times \Sigma \times \phi(X) \times 2^X \times Q$ is a finite set of transitions (q, a, g, R, q') where a is a letter in Σ, g is a clock constraint called the guard, and R is the set of clocks that are reset on the transition.

It accepts timed words of the form $(a_1, t_1) \dots (a_n, t_n)$ where $a_i \in \Sigma$ and $t_i \in \mathbb{R} \, \forall \, i \in \mathbb{N}$.

Define *valuation of clocks* at a given time to be a vector $v = (v_{x_1}, v_{x_2} \dots v_{x_k})$. let R be a set of clocks then we define $[\,R]\,(v)(x) = v(x)$ if $x \notin R$ and 0 otherwise. A *run* of a timed word $(a_1, t_1) \dots (a_n, t_n)$ is a finite sequence of states and transitions $q_0, (a_1, t_1), q_1, (a_2, t_2) \dots q_n, (a_n, t_n)$ such that $(q_i, a_{i+1}, g, R_i, q_{i+1}) \in \Delta$ and g is satisfied by the *valuation of clocks* v_i, where v_i is defined inductively as $v_i(x) = 0$ if $x \in R_i$, $v_i(x) = v_{i-1}(x) + t_i$ otherwise, and $v_0(x) = 0$, for all $x \in X$. Language accepted by this DiTA M is the set $\{(a_1, t_1) \dots (a_n, t_n) \mid$ it has an accepting run on M and $t_i \in \mathbb{R}, \forall i \in \mathbb{N}\}$. The Boolean operations of union, intersection, and complement of timed languages are defined as usual.

2.2.2 Regions in timed automata

Assume $X = \{x_1, x_2 \dots x_n\}$ and for each $x \in X$ let n_x denote the maximum constant x has been compared. For a timed automaton we say that two clock valuations v, w are *region equivalent* if the following conditions hold

- For all clocks $x \in X$, either $\lfloor v(x) \rfloor$ and $\lfloor w(x) \rfloor$ are same or both $v(x)$ and $w(x)$ exceeds n_x.
- For all clocks $x, y \in X$ with $v(x) \leq n_x$ and $v(y) \leq n_y$, $\langle v(x) \rangle \leq \langle v(y) \rangle$ if and only if $\langle w(x) \rangle \leq \langle w(y) \rangle$.
- For all clocks $x \in X$ with $v(x) \leq n_x$, $\langle v(x) \rangle = 0$ if and only if $\langle w(x) \rangle = 0$.

A *clock region* is an equivalence class of region equivalent clock valuations.

2.3 Symbolic Automata

In *symbolic automata*, transitions carry predicates over a Boolean algebra. One symbolic transition denotes many symbolic transitions as shown in 2.4. We need one more definition before defining *symbolic automata* formally.

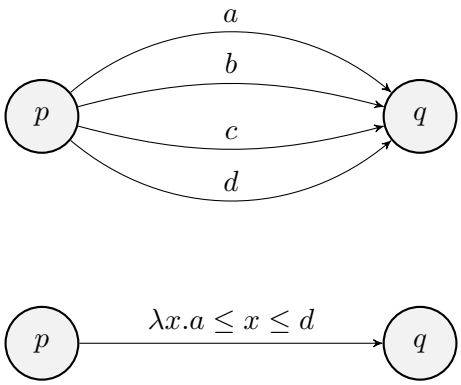


Figure 2.4: One symbolic transition denotes many concrete transitions.

Definition 2.3.1. (Effective Boolean Algebra) An *effective Boolean algebra* A is a tuple $(\Sigma, \Psi, , [\![\_]\!], \bot, \top, \vee, \wedge, \neg)$ where Σ is a set of domain elements, Ψ is a set of predicates closed under the Boolean connectives, with $\bot, \top \in \Psi$, the component $[\![\_]\!] : \Psi \to 2^{\Sigma}$ is a denotation function such that

(i) $[\![\bot]\!] = \emptyset$,

(ii) $[\![\top]\!] = \Sigma$,

(iii) for all $\varphi, \psi \in \Psi$,
$[\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$,
$[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$, and
$[\![\neg\varphi]\!] = \Sigma \setminus [\![\varphi]\!]$.

We also require that checking satisfiability of φ, i.e., whether $[\![\varphi]\!] \neq \emptyset$ is decidable.

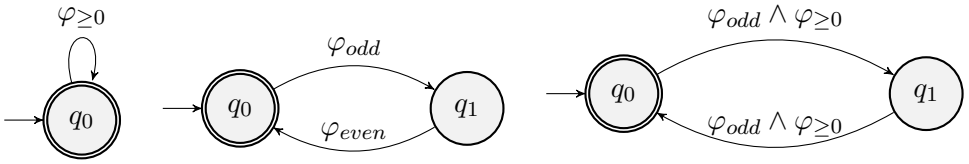
We can now define *symbolic finite automata*, which are finite automata over a symbolic alphabet, where edge labels are replaced by predicates.

Definition 2.3.2. (Symbolic Finite Automaton) A *symbolic finite automaton* (s-FA) is a tuple $M = (A, Q, q_0, F, \Delta)$ where

- $A = (\Sigma, \Psi, [\![\_]\!], \bot, \top, \vee, \wedge, \neg)$ is an effective Boolean algebra,
- Q is a finite set of states,
- $q_0 \in Q$ is the initial state,
- $F \subseteq Q$ is the set of final states, and
- $\Delta \subseteq Q \times \Psi_A \times Q$ is a finite set of transitions.

Elements of Σ are called characters and finite sequences of characters are called strings, i.e., elements of Σ^*. A transition $\rho = (q_1, \varphi, q_2) \in \Delta$, also denoted $q_1 \xrightarrow{\varphi} q_2$, is a transition from the source state q_1 to the target state q_2, where φ is the guard or predicate

of the transition. For a character $a \in \Sigma$, an a-transition of M, denoted $q_1 \xrightarrow{a} q_2$ is a transition $q_1 \xrightarrow{\varphi} q_2$ such that $a \in [\![\varphi]\!]$. An s-FA M is deterministic if, for all transitions $(q, \varphi_1, q_1), (q, \varphi_2, q_2) \in \Delta$, if $q_1 \neq q_2$ then $[\![\varphi_1 \wedge \varphi_2]\!] = \emptyset$, i.e., for each state q and character a there is at most one a-transition from q. A string $w = a_1 a_2 ... a_k$ is accepted at state q iff, for $1 \leq i \leq k$, there exist transitions $q_{i-1} \xrightarrow{a_i} q_i$ such that $q_0 = q$ and $q_k \in F$.



(a) M_{pos} (b) $M_{ev/odd}$ (c) $M_{ev/odd} \times M_{pos}$

Figure 2.5: Symbolic automata

Example 2.3.1. Examples of s-FAs are M_{pos} and $M_{ev/odd}$ in figure 2.5. These two s-FAs have 1 and 2 states respectively, and they both operate over the Boolean algebra with $\mathbb{Z}$ as the domain. The s-FA M_{pos} accepts all strings consisting only of positive numbers, while the s-FA $M_{ev/odd}$ accepts all strings of even length consisting only of odd numbers. For example, $M_{ev/odd}$ accepts the string $[2, 4, 6, 2]$ and rejects strings $[2, 4, 6]$ and $[51, 26]$. The product automaton of M_{pos} and $M_{ev/odd}$ is $M_{ev/odd} \times M_{pos}$ which accepts the language $\mathcal{L}(M_{pos}) \cap \mathcal{L}(M_{ev/odd})$.

2.3.1 Properties of Symbolic Automata

Lemma 2.3.1. Given an s-FA M one can effectively construct a deterministic s-FA M_{det} such that $L(M) = L(M_{det})$.

The determinization algorithm is similar to the subset construction for automata over finite alphabets, but also requires combining predicates appearing in different transitions. If M contains k inequivalent predicates and n states, then the number of distinct predicates in M_{det} is at most $2k$ and the number of states is at most $2n$. In other words, in addition to the classic state space explosion risk there is also a predicate space explosion risk.

Lemma 2.3.2. Given s-FAs M_1 and M_2 one can construct s-FAs M_1^c and $M_1 \times M_2$ such that $L(M_1^c) = \Sigma^* \setminus L(M_1)$ and $L(M_1 \times M_2) = L(M_1) \cap L(M_2)$.

Proof. To complement a deterministic partial s-FA M_1, M_1 is first *completed* by adding a new non-final state s with loop $s \xrightarrow{\top} s$ and for each partial state p a transition $p \xrightarrow{\neg dom(p)} s$ where, $dom(p) := \{\varphi \mid \exists q : (p, \varphi, q) \in \Delta\}$. Then the final states and the non-final states are swapped in M_1^c.

To intersect two s-FAs $M_1 = (A_1, Q_1, q_0^1, F_1, \Delta_1)$ and $M_2 = (A_2, Q_2, q_0^2, F_2, \Delta_2)$, we define $M_1 \times M_2 = (A', Q_1 \times Q_2, \{q_0^1, q_0^2\}, F', \Delta')$ where $A' = (\Sigma, \Psi_1 \cup \Psi_2, [\![\_]\!], \bot, \top, \vee, \wedge, \neg)$,

$F' = \{(q_1, q_2) \mid q_1 \in F_1, q_2 \in F_2\}$ and $\Delta' \subseteq (Q_1 \times Q_2) \times (\Psi_1 \cup \Psi_2) \times (Q_1 \times Q_2)$. If $(q_1, \varphi_1, q_2) \in \Delta_1$ and $(q_3, \varphi_2, q_4) \in \Delta_2$ then $((q_1, q_3), \varphi_1 \wedge \varphi_2, (q_2, q_4)) \in \Delta'$.

□

Lemma 2.3.3. Given s-FAs M_1 and M_2 it is decidable to check if M_1 is empty, i.e., whether $L(M_1) = \emptyset$ and if M_1 and M_2 are language-equivalent, i.e. whether $L(M_1) = L(M_2)$.

Proof. Checking emptiness requires checking what transitions are satisfiable and, once unsatisfiable transitions are removed, any path reaching a final state from an initial state represents at least one accepting string.

Equivalence can be reduce to emptiness using closure under Boolean operations.

□

O. Maler et al. gave an algorithm for learning regular languages over large alphabets using symbolic automata. There is also an active learning algorithm for symbolic automata by S. Drews and L. D'Antoni [8].

Chapter 3

Learning Discrete Timed Automata as a DFA

3.1 Discrete Timed Automata

Definition 3.1.1. (Discrete Timed Automata) A *discrete timed automaton* M is a tuple $(Q, \Sigma, X, \Delta, q_0, F)$ where

- Q is a finite set of states,
- Σ is a finite alphabet,
- $X = \{x_1, x_2 \dots x_k\}$ is a finite set of clocks,
- q_0 is the initial state,
- $F \subseteq Q$ is a set of accepting state,
- $\Delta \subseteq Q \times \Sigma \times \phi(X) \times 2^X \times Q$ is a finite set of transitions (q, a, g, R, q') where $a \in \Sigma$, g is a clock constraint called guard and R is the set of clocks that are reset on the transition.

It accepts timed words of the form $(a_1, t_1) \dots (a_n, t_n)$ where $a_i \in \Sigma$ and $t_i \in \mathbb{N} \, \forall \, i \in \mathbb{N}$.

Define *valuation of clocks* at a given time to be a vector $v = (v_{x_1}, v_{x_2} \dots v_{x_k})$. let R be a set of clocks then we define $[R](v)(x) = v(x)$ if $x \notin R$ and 0 otherwise. A *run* of a timed word $(a_1, t_1) \dots (a_n, t_n)$ is a finite sequence of states and transitions $q_0, (a_1, t_1), q_1, (a_2, t_2) \dots q_n, (a_n, t_n)$ such that $(q_i, a_{i+1}, g, R_i, q_{i+1}) \in \Delta$ and g is satisfied by the *valuation of clocks* v_i, where v_i is defined inductively as $v_i(x) = 0$ if $x \in R_i$, $v_i(x) = v_{i-1}(x) + t_i$ otherwise, and $v_0(x) = 0$, for all $x \in X$. Language accepted by this DiTA M is the set $\{(a_1, t_1) \dots (a_n, t_n) \mid$ it has an accepting run on M and $t_i \in \mathbb{N}, \forall i \in \mathbb{N}\}$.

Intuitively, it can be thought of as a timed automata in which transitions are taken only when time is an integer. We call the class of *discrete timed automata*, DiTA and the class of deterministic DiTA, DDiTA.

Example 3.1.1. write an example.

3.1.1 Regions in DiTA

In the case of discrete timed automata, each equivalence class consists of only one element. Each equivalence class/region looks like $\{x_1 \sim t_1 \wedge x_2 \sim t_2 \cdots \wedge x_k \sim t_k\}$ where $\sim \in \{=, \geq\}$ and $\forall i, \sim \equiv = \implies t_i < n_i$ and $t_i \in \mathbb{N}$ and $\sim \equiv \geq \implies t_i = n_i$. The number of regions for a DiTA are $\prod_{x \in X} (n_x + 1)$. These regions are used to construct *region automaton* which is a DFA.

Definition 3.1.2 (Region Automata)**.** For a TA $A = (Q, \Sigma, X, \Delta, q_0, F)$, the corresponding region automaton $R(A) = (Q', \Sigma, \Delta', q_0', F')$ is a DFA, where

- $Q' = \{(q, \alpha) \mid q \in Q \text{ and } \alpha \text{ is a region}\}$,
- $q_0' = (q_0, v_0)$, where $v_0(x) = 0, \forall x \in X$,
- $F' = \{(q_f, \alpha) \mid q_f \in F \text{ and } \alpha \text{ is a region}\}$,
- $((q_1, \alpha_1), (q_2, \alpha_2), a) \in \Delta'$ iff there is an edge $(q_1, a, g, R, q_2) \in \Delta$ and a region α' such that $\alpha' = \alpha_1 + t$ for some $t \in \mathbb{N}$, α' satisfies g and $\alpha_2 = [R]\alpha'$.

3.2 Conversion of a DiTA to a DFA

Consider a function $\Gamma : \tilde{\Sigma}^* \to (\Sigma \cup \{\checkmark\})^*$ such that $\Gamma((t_1, a_1)(t_2, a_2) \dots (t_n, a_n)) = (\checkmark \dots \checkmark)a_1(\checkmark \dots \checkmark)a_2 \dots (\checkmark$ with t_i ticks in between a_{i-1} and a_i. For a timed language $L \subseteq \tilde{\Sigma}^*$ you can get a language $L' \subseteq (\Sigma \cup \{\checkmark\})^*$ such that $L' = \Gamma(L)$. We call L' the *tick language*.

In the region automata construction instead of adding a transition $((q_1, \alpha_1), (q_2, \alpha_2), a)$ add a path of length $t + 1$ from q_1 to q_2 with a new symbol ✓in first t transitions and a in the last transition. We call this new automaton the *tick automaton* and it accepts the *tick language*.

Now instead of learning a discrete timed automata we can learn the tick automaton. The size of this automaton will be very large but we can learn the language using L^*.

Chapter 4

Learning Discrete Timed Automata

Let $\Sigma = (a_1, a_2, \ldots a_n)$ be a finite alphabet. Define $\tilde{\Sigma} = \mathbb{N} \times \Sigma$, this defines a partial order on the elements of $\tilde{\Sigma}$ i.e. $(t, a) < (t', a')$ if $t < t'$. We will refer to $\tilde{\Sigma}$ as the *concrete alphabet* from now on.

Define r-DDiTA to be the class of DDiTA with resets on every transition.

Theorem 4.0.1. Class DiTA is equivalent to the class r-DDiTA.

Proof. Let M be an automaton in class DiTA, then we look at its region automaton R. When we add transition $((q_1, \alpha_1), (q_2, \alpha_2), a)$ in R we add a guard $g = (x = t)$ to that transition (for the value of t, look at 2.1.2. □

Define a function $h : \tilde{\Sigma}^* \to \tilde{\Sigma}^*$ such that $h((t_1, a_1), (t_2, a_2) \ldots (t_n, a_n)) = (\delta_1, a_1), (\delta_2, a_2) \cdots (\delta_n, a_n)$ where, $\delta_1 = t_1$ and $\delta_i = t_i - t_{i-1}$, for $2 \leq i \leq n$. This is a one-one, onto function. Hence, for a language $L \subseteq \tilde{\Sigma}^*$, we can define $h(L) = \{h(w) \mid w \in L\}$. From now on, whenever we say L, we mean $h(L)$ because we can always get back L from $h(L)$ using h^{-1}. $h(L)$ just represents the words $((\delta_1, a_1) \ldots (\delta_n, a_n))$ where δ_i represents the valuation of the clock at that letter assuming that clocks get reset on every transition.

Definition 4.0.1 ($\equiv_L$). Let L be a language over $\tilde{\Sigma}^*$. Define a relation $\equiv_L$ between timed strings w.r.t. the language L as $x \equiv_L y$ iff $\forall z \in \tilde{\Sigma}^*$, either $xz, yz \in L$ or $xz, yz \notin L$.

Lemma 4.0.1. $\equiv_L$ is an equivalence relation.

Proof. We will show that this relation is reflexive, symmetric and transitive.

- Reflexive: clearly $x \equiv_L x$.
- Symmetric: if $x \equiv_L y$ then trivially $y \equiv_L x$.
- Transitive: if $x \equiv_L y$ and $y \equiv_L z$ then $\forall w$ either $xw, yw \in L$ which implies $zw \in L$ since $y \equiv_L z$, or $xw, yw \notin L$ which implies $zw \notin L$ because $y \equiv_L z$.

Hence, $\equiv_L$ is an equivalence relation. □

Now, we define another relation between strings w.r.t. the automaton,

Definition 4.0.2 ($\equiv_M$)**.** Let M be an 1-DiTA with resets. Define a relation between timed strings w.r.t M as, $x \equiv_M y$ iff the runs of M over x and y leads to the same state, i.e. $\Delta^*(q_0, x) = \Delta^*(q_0, y)$

Lemma 4.0.2. $\equiv_M$ is an equivalence relation.

Proof. Again we will show that this relation is reflexive, symmetric and transitive,

- Reflexive: since it is a deterministic automaton, $x \equiv_M y$.
- Symmetric: if $x \equiv_M y$ then $y \equiv_M x$ trivially.
- Transitive: if $x \equiv_M y$ and $y \equiv_M z$ then again $x \equiv_M z$ trivially.

Hence, $\equiv_M$ is also an equivalence relation. □

Lemma 4.0.3. Let $L = \mathcal{L}(M)$ for a r-DDiTA M, $\forall x, y \in \tilde{\Sigma}^*$, $x \equiv_M y \implies x \equiv_L y$.

Proof. Given $x \equiv_M y$, this says that both x and y goes to the same state on M, i.e. $\Delta^*(q_0, x) = \Delta^*(q_0, y) = q$. Let $z \in \tilde{\Sigma}$. Now, $\Delta^*(q_0, x \cdot z) = \Delta(\Delta^*(q_0, x), z) = \Delta(q, z)$ and $\Delta^*(q_0, y \cdot z) = \Delta(\Delta^*(q_0, y), z) = \Delta(q, z)$ since M is deterministic. This implies that $\forall w \in \tilde{\Sigma}^*, \Delta^*(q_0, x \cdot w) = \Delta^*(q_0, y \cdot w)$, if this state is accepting then, $xw, yw \in L$, otherwise $xw, yw \notin L$. □

Corollary 4.0.1. If there is an r-DDiTA for L then $\equiv_L$ has a finite number of equivalence classes.

Proof. From the lemma 4.0.3, we know that if two strings x and y goes to the same states then $x \equiv_L y$. □

4.1 Minimization algorithm for r-DDiTA

Minimization algorithm for r-DDiTA goes like this

- Step 1: Draw a table of pairs of states (q_i, q_j) are all marked initially.
- Step 2: Mark pairs q_i, q_j if one of them is a final and other is a non-final state.
- Step 3: For an unmarked pair (q_i, q_j), for all pairs of outgoing transitions (g_i, a) and (g_j, a), from q_i and q_j respectively, if $g_i \wedge g_j \neq \perp$ and the pair $(\Delta(q_i, (g_i, a)), \Delta(q_j, (g_j, a)))$ is marked then mark the pair (q_i, q_j).
- Step 4: Repeat step 3 until there are no more pairs to mark.
- Step 5: Merge two states q_i, q_j if they are unmarked and merge the outgoing transitions on same letter from this state if union of them is a continuous set.

Lemma 4.1.1. At any step in algorithm, if a pair of states (q_i, q_j) is marked then $\exists w \in \tilde{\Sigma}^*$ such that, one of $\Delta^*(q_i, w)$ and $\Delta^*(q_j, w)$ is final and the other one is non-final.

Proof. The proof is by induction on the iteration number n. For the base case i.e. $n = 1$, if (q_i, q_j) is marked then take $w = \epsilon$. Now, assume that at nth iteration, K represents the set of marked pair of states and for each element of K, say (q, q'), $\exists w \in \tilde{\Sigma}^*$ such that one of $\Delta^*(q, w)$ and $\Delta^*(q', w)$ is final and the other one is non-final. Now, at the $(n+1)$th iteration if a pair (q_i, q_j) is marked then $\exists x \in \tilde{\Sigma}$ such that $(\Delta(q_i, x), \Delta(q_j, x)) = (q_i', q_j') \in K$. By induction hypothesis $\exists w \in \tilde{\Sigma}^*$ such that q_i', q_j' go to final and non-final states. Now, on the word $w' = x \cdot w$ one of q_i and q_j goes to a final and the other one goes to a non-final state. □

Corollary 4.1.1. For a pair of different states (q_i, q_j), if $\forall z \in \tilde{\Sigma}^*$, $\Delta^*(q_i, z)$ and $\Delta^*(q_j, z)$ both are accepting or both are rejecting then (q_i, q_j) will never be marked.

Proof. This is immediate from the last lemma. □

Theorem 4.1.1 (Correctness)**.** Let M' be the automaton we get from running this algorithm on M which accepts L and is minimal.

Proof. First, we will show that M' indeed accepts L using induction on the length of strings. For the base case, let $x \in \tilde{\Sigma}$ and $\Delta(q_0, x) = q_1$. Assume that the q_0 and q_1 were merged with some other states and formed q_0' and q_1' then clearly $\Delta'(q_0', x) = q_1'$ in the new automaton. This implies that M accepts x iff M' accepts x. Now, assume that $\forall y \in \tilde{\Sigma}^*$ with $|y| \leq n$, M accepts y iff M' accepts y. Let $y' = y_1 \cdot y_2 \in \tilde{\Sigma}^*$ with $|y'| = n + 1$ and $|y_2| = 1$. By induction hypothesis, M' accepts y_1 iff M accepts y_1. Let $\Delta^*(q_0, y_1) = q$ and $(\Delta')^*(q_0', y_1) = q'$. Now, if $\Delta(q, y_2)$ is accepting then $\Delta'(q', y_2)$

Now, we will show that the automaton M' is minimal. If we can show that $x \equiv_L y \implies x \equiv_{M'} y$, then we are done. We will prove the contrapositive of this, which is $x \not\equiv_{M'} y \implies x \not\equiv_L y$. Assume that x and y both go to different states q_1 and q_2 with $q_2 \neq q_2$. Now, by corollary 4.1.1 we know that (q_1, q_2) can never be marked, hence they would be merged by the algorithm, which is a contradiction. Therefore the automaton is minimal.

□

4.2 Timed observation table

Let $\mu : X \to 2^{\tilde{\Sigma}} \setminus \{\emptyset\}$ be a function where $X \subseteq G \times \Sigma$. With a language $L \subseteq \tilde{\Sigma}^*$, we associate a *characteristic function* $f : \tilde{\Sigma}^* \to \{+, -\}$ such that, $f(x) = +$ if $x \in L$ and $f(x) = -$, otherwise. Define $\tilde{f} : G \times \Sigma \to \{+, -\}$ such that $\tilde{f}(x) = +$ if $f(y) = +, \forall y \in \mu(x)$ and $f(x) = -$, otherwise. A set P is *prefix-closed* if $\forall p \in P$ all the prefixes of p are in P. Analogously, we can define a set to be *suffix-closed*.

Definition 4.2.1 (Timed Observation Table)**.** A *timed observation table* is a tuple $T = (\tilde{\Sigma}, G, S, R, E, \tilde{f}, \mu)$ where

- Σ is an alphabet,

- $\tilde{\Sigma}$ is the timed extension of Σ,
- G is a set of guards on the clock c,
- $S \uplus R$ is a prefix-closed subset of $G \times \Sigma$ such that if $w \cdot (g_0, a) \in S \cup R$ then $\exists\ w \cdot (g_1, a), w \cdot (g_2, a) \ldots w \cdot (g_m, a) \in S \cup R$, such that $\bigvee_{i=1}^{m} g_i = \top$ and $g_i \wedge g_j = \bot$ for $i \neq j$. Also, for every $s \in S$, there exists $s \cdot (g, a) \in R$ where $g \neq \bot$ and for every r in R, $r \cdot (g, a) \notin S \cup R$ for any (g, a),
- E is a suffix-closed subset of $\tilde{\Sigma}$,
- $\tilde{f} : (S \cup R) \cdot E \to \{+, -\}$ is the characteristic function,
- $\mu : (S \cup R) \cdot E \to 2^{\tilde{\Sigma}} \setminus \{\emptyset\}$ is the *evidence function* satisfying $\mu(\tilde{w} \cdot (g, a)) = \mu(\tilde{w}) \cdot (g_{min}, a)$ for all $w \in \tilde{\Sigma}^*$, where g_{min} is the minimum value for which the guard g is satisfied.

Define, $\tilde{M}_T = (S \cup R) \cdot E$ and $M_T = \{s \cdot e \mid s \in \mu(\tilde{s}), \tilde{s} \in S \cup R, e \in E\}$ is the *concrete sample*. We refer to the row corresponding to some element u of $S \cup R$ as $\tilde{f}_u$.

Definition 4.2.2. A timed observation table T is

- Closed if $\forall r \in R, \exists s \in S, \tilde{f}_r = row(s)$,
- Reduced if $\forall s, s' \in S, row(s) \neq row(s')$,
- Consistent if $\forall s, s' \in S, row(s) = row(s') \implies row(s \cdot a) = row(s' \cdot a), \forall s \cdot a \in S \cup R$,

Theorem 4.2.1 (r-DDiTA from the table)**.** From a closed, reduced and evidence compatible table T you can construct the minimal r-DDiTA that is compatible with M_T.

Proof. For a table $T = (\tilde{\Sigma}, G, S, R, E, \tilde{f}, \mu)$, define the automaton $M(T) = (Q^M, \Sigma, \Delta^M, q_0^M, F^M)$ where

- $Q^M = S,\ q_0^M = \epsilon$
- $F^M = \{s \in S \mid \text{first entry in } row(s) \text{ is } +\}$
- $\Delta^M : Q^M \times \tilde{\Sigma} \to Q^M$ is defined as $\Delta^M(s, a) = s'$ such that $s' \in S$ and $row(s') = row(s \cdot a)$.

By construction and like the L^* algorithm, $M(T)$ classifies correctly the concrete sample.

Now, we shall show that it is the minimal automaton. Suppose the automaton is not minimal then in the minimization algorithm at least two states will be merged together, s_1 and $s_2 \in S$. Now, $row(s_1) \neq row(s_2)$ since the table is reduced. Find $b \in E$ such that $\mu(s_1) \cdot b \in L$ and $\mu(s_2) \cdot b \notin L$. If s_1 and s_2 are merged in minimization algorithm then on reading b they should go to the same state. This is a contradiction. □

4.3 Learning algorithm for r-DDiTA

There is a partial order on the elements of $\tilde{\Sigma}$, we can extend it to $\tilde{\Sigma}^*$. For u and v, $u \neq v$ in $\tilde{\Sigma}^*$ if $|u| < |v|$ then $u < v$, if $|u| > |v|$ then $u > v$. If $|u| = |v|$ assume $u = u_1 u_2 \dots u_m$ and $v = v_1 v_2 \dots v_m$ then find minimum i such that $u_i \neq v_i$. Now, $u < v$ if $u_i < v_i$ and $u > v$ if $u_i > v_i$. We assume that the teacher provides minimal counter-examples to the learner w.r.t. this partial order. And as the evidence we always select the smallest words.

Now we will describe the learning algorithm. The algorithm is similar to the learning of languages over large alphabets by Oded Maler. Initially the table is $T = (\tilde{\Sigma}, G, S, R, E, \tilde{f}, \mu)$, where $G = \{g_0 := c \geq 0\}$, $S = \{\epsilon\}$, $R = \{(g_0, a) \mid a \in \Sigma\}$, $E = \{\epsilon\}$ and $\mu(g_0, a) = (0, a)$ for all $a \in \Sigma$. The table is filled by membership queries of ϵ and $(0, a)$, $\forall a \in \Sigma$. First, close the table T i.e. if $\exists r \in R$ such that $\tilde{f}_r \neq \tilde{f}_s$ for all $s \in S$ then we move r to S and add $r' = r \cdot (g_0, a)$ in R for all $a \in \Sigma$ and set $\mu(r') = \mu(r) \cdot (0, a)$. Once it is closed, we make a hypothesis automaton as described in theorem 4.2.1.

When a counter-example w is given by the teacher, it means that somewhere a wrong transition is taken in the run of w. Hence w admits a factorization $w = u \cdot b \cdot v$ where $u \in \tilde{\Sigma}^*$ and $b = (t, a) \in \tilde{\Sigma}$. This can happen in the following two cases, (1). b should not satisfy the guard on the transition and the guard need to be refined or (2). b leads to an undiscovered state. We find the factorization $w = u \cdot b \cdot v$ by finding the largest prefix u of w such that $u \in \mu(\tilde{u})$ for some $\tilde{u} \in S \cup R$ and $u \cdot b \notin \mu(\tilde{u}')$ for any $\tilde{u}' \in S \cup R$. As we said earlier, there are two cases.

In the first case, $\tilde{u} \in S$ then $\tilde{u}$ is already a state in the hypothesis but b indicates that t should not satisfy the guard and it needs refinement. Find $\tilde{b} = (g, a) \in G \times \Sigma$ such that t satisfies the guard g. Now, split the guard $g := c \geq t_1 \wedge c < t_2$ into $g_1 := c \geq t_1 \wedge c < t$ and $g_2 := c \geq t \wedge c < t_2$. Remove $\tilde{u} \cdot \tilde{b}$ from R and add $\tilde{u} \cdot (g_1, a)$ and $\tilde{u} \cdot (g_2, a)$ in R.

In the second case, $\tilde{u} \in R$. Now, from the counter-example we can say that $\tilde{u}$ is not equivalent to any of the existing states and it should form a new state in the hypothesis. We find $\tilde{s} \in S$ such that $\tilde{f}_u = \tilde{f}_s$, since T is reduced $\nexists\, \tilde{s}' \neq \tilde{s}$ and $\tilde{s}' \in S$ such that $\tilde{f}_s = \tilde{f}_{s'}$. Now, $\tilde{u}$ should form a new state, hence $row(u)$ should be different from $row(\tilde{s})$. We add $\tilde{u} \in S$ to distinguish $\tilde{u}$ from $\tilde{s}$, we also add $b \cdot v$ and all of its suffixes to E. We also need to add transitions from this new state $\tilde{u}$ i.e. if $t = 0$, we add (g_0, a) in R, where $g_0 = c \geq 0$ and set $\mu(\tilde{u} \cdot (g_0, a)) = \mu(\tilde{u}) \cdot (0, a)$ for all $a \in \Sigma$. If $t \neq 0$, add $\tilde{u} \cdot (g_{new}, a)$ and (g'_{new}, a) to R for all $a \in \Sigma$, where $g_{new} := c \geq 0 \wedge c < t$ and $g'_{new} := c \geq t$. Set $\mu(\tilde{u} \cdot (g_{new}, a)) = \mu(\tilde{u}) \cdot 0$ and $\mu(\tilde{u} \cdot (g'_{new}, a)) = \mu(\tilde{u}) \cdot b$.

Example 4.3.1. Consider the r-DDiTA shown in figure 4.1. We will look at the run of learning algorithm on this language.

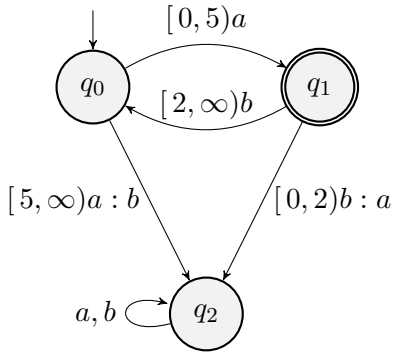


Figure 4.1: Example of a r-DDiTA

T_0	ϵ
ϵ	$-$
$[0,\infty)a$	$+$
$[0,\infty)b$	$-$
$[0,\infty)a[0,\infty)a$	$-$
$[0,\infty)a[0,\infty)b$	$-$

T_1	ϵ
ϵ	$-$
$[0,5)a$	$+$
$[5,\infty)a$	$-$
$[0,\infty)b$	$-$
$[0,5)a([0,\infty)a$	$-$
$[0,5)a([0,\infty)b$	$-$

T_2	ϵ	$(0,a)$
ϵ	$-$	$+$
$[0,5)a$	$+$	$-$
$[5,\infty)a$	$-$	$-$
$[0,\infty)b$	$-$	$-$
$[0,5)a[0,\infty)a$	$-$	$-$
$[0,5)a[0,\infty)b$	$-$	$-$
$[5,\infty)a[0,\infty)a$	$-$	$-$
$[5,\infty)a[0,\infty)b$	$-$	$-$

T_3	ϵ	$(0,a)$
ϵ	$-$	$+$
$[0,5)a$	$+$	$-$
$[5,\infty)a$	$-$	$-$
$[0,\infty)b)$	$-$	$-$
$[0,5)a[0,\infty)a$	$-$	$-$
$[0,5)a[0,2)b$	$-$	$-$
$[0,5)a[2,\infty)b$	$-$	$+$
$[5,\infty)a[0,\infty)a$	$-$	$-$
$[5,\infty)a[0,\infty)b$	$-$	$-$

Figure 4.2: Observation Tables

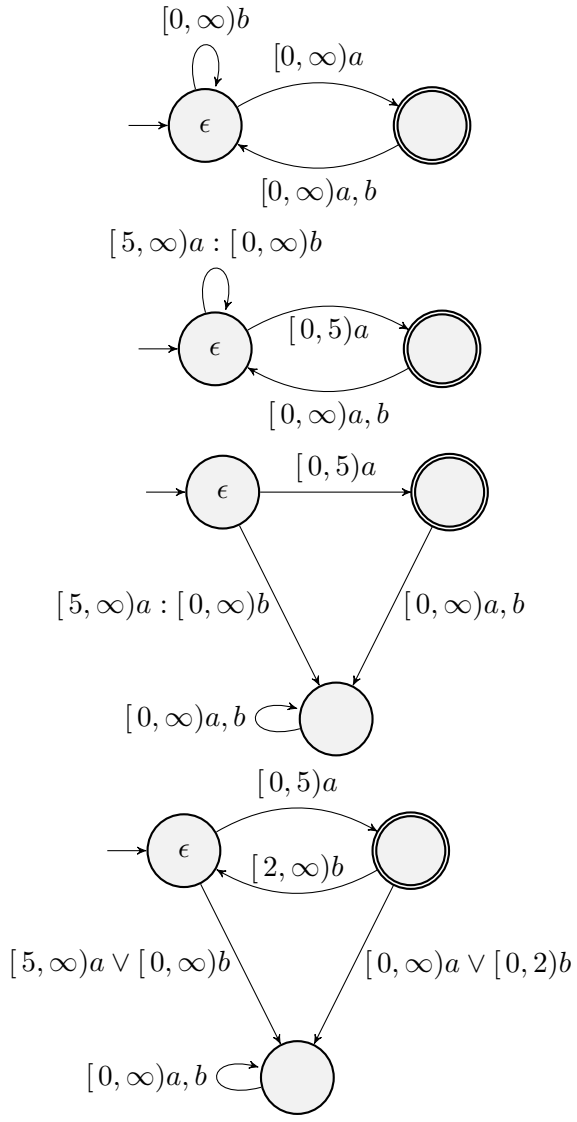


Figure 4.3: Hypothesis automata for example 4.3.1

Chapter 5

Conclusions and Future Work

In this thesis we first described the L^* algorithm to learn regular sets. We also described symbolic automata. After that we described a procedure to convert a timed language to a regular language which capture the same behaviour. In the end we described an active algorithm to learn a discrete timed automaton as an automaton in the class r-DDiTA. We assume that the teacher returns minimal counter-examples, then we get an automaton which is minimal. We also gave a minimizing algorithm for r-DDiTA and proved that this automaton is canonical.

In future we would like to implement this algorithm and build a tool to learn discrete timed automata. We can try to learn some real-time systems using this algorithm. There are some systems that we can model as timed automata described in [9], we can try to learn some of them.

We can also try to find the minimal DiTA for a given r-DDiTA. This would help a lot since right now for a general DiTA there can be exponential blow up in the number of states of the model. The resulting DiTA would be a lot smaller in the size.

References

[1] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183 – 235, 1994. 1, 6

[2] S. Verwer, M. Weerdt, and C. Witteveen, "Efficiently learning simple timed automata," *Techniques in Coloproctology - TECH COLOPROCTOLOGY*, 01 2008. 1

[3] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, no. 2, pp. 87 – 106, 1987. 1, 2, 5

[4] O. Grinchtein, B. Jonsson, and M. Leucker, "Learning of event-recording automata," *Theoretical Computer Science*, vol. 411, no. 47, pp. 4029 – 4054, 2010. 1

[5] B. Jonsson and F. W. Vaandrager, "Learning mealy machines with timers," 2017. 1

[6] M. Tappler, B. K. Aichernig, K. G. Larsen, and F. Lorber, "Learning timed automata via genetic programming," *CoRR*, vol. abs/1808.07744, 2018. 1

[7] L. D'Antoni and M. Veanes, "The power of symbolic automata and transducers," pp. 47–67, 07 2017. 1, 2

[8] S. Drews and L. D'Antoni, "Learning symbolic automata," pp. 173–189, 03 2017. 1, 12

[9] O. Maler, "The unmet challenge of timed systems," in *From Programs to Systems. The Systems perspective in Computing* (S. Bensalem, Y. Lakhneck, and A. Legay, eds.), (Berlin, Heidelberg), pp. 177–192, Springer Berlin Heidelberg, 2014. 22

[10] O. Maler and I. Eleftheria Mens, "Learning regular languages over large ordered alphabets," vol. 11, 04 2014.

[11] D. Peled, M. Y. Vardi, and M. Yannakakis, "Black box checking," *J. Autom. Lang. Comb.*, vol. 7, pp. 225–246, Nov. 2001.