Irini-Eleftheria Mens

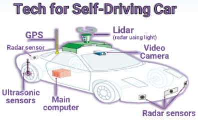
VERIMAG, *University of Grenoble-Alpes*

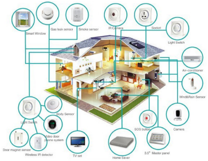
Learning Regular Languages over Large Alphabets

10 October 2017

Jury Members

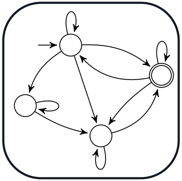
Oded Maler	Directeur de thèse	Laurent Fribourg	Examinateur
Dana Angluin	Rapporteur	Eric Gaussier	Examinateur
Peter Habermehl	Rapporteur	Frits Vaandrager	Examinateur





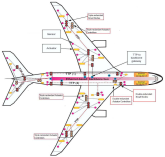
Black Box Learning

Model



Language Identification

System Identification



Inductive Inference

A Short Prehistory and History of Automaton Learning

1956 Edward F Moore. *Gedanken-experiments on sequential machines.*
Defines the problem as a black box model inference.

1967 E. Mark Gold. *Language identification in the limit.*

1972 E. Mark Gold. *System identification via state characterization.*
Learning finite automata is possible in finite time. He first uses the basic idea that underlies table-based methods.

1978 E. Mark Gold. *Complexity of automaton identification from given data.*
Finding the minimal automaton compatible with a given sample is NP-hard.

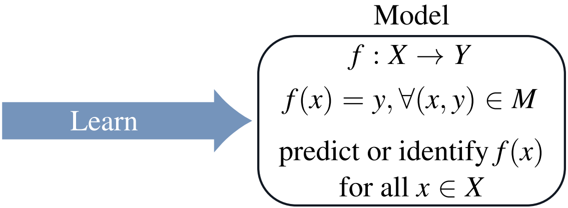
1987 Dana Angluin. *Learning regular sets from queries and counter-examples.*
The L^* active learning algorithm with membership and equivalence queries. Polynomial in the automaton size.

1993 Ronald L. Rivest and Robert E. Schapire. *Inference of finite automata using homing sequences.*
An improved version of the L^* algorithm using the breakpoint method to treat counter-examples.

Machine Learning

a small sample

$M = \{(x, y) : x \in X,\ y \in Y\}$



Learn

Model

$f : X \to Y$

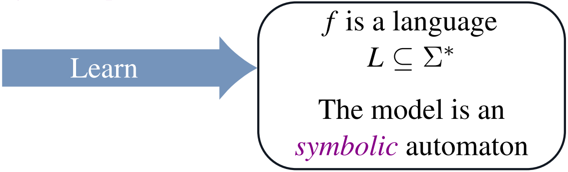
$f(x) = y, \forall (x, y) \in M$

predict or identify $f(x)$ for all $x \in X$

Learning Regular Languages

over large or infinite alphabets

- Σ an alphabet
- $X = \Sigma^*$ set of words
- $Y = \{+, -\}$



Learn

Model

f is a language $L \subseteq \Sigma^*$

The model is an *symbolic* automaton

Types of Learning

Off-line vs Online

The sample M is known before the learning procedure starts.

The sample M is updated during learning.

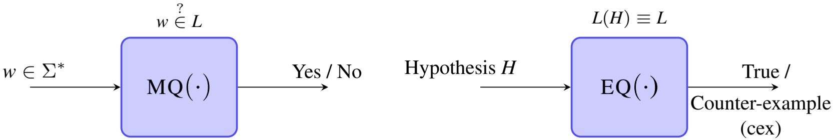
Passive vs Active

The sample M is given.

The sample M is chosen by the learning algorithm.

Learning using Queries

The learning algorithm can access queries e.g., membership queries, equivalence queries, etc.



Outline

Preliminaries

- Regular Languages and Automata
- The L^* Algorithmic Scheme

Large Alphabets

- Motivation
- Symbolic Representation of Transitions - Symbolic Automata

Learning Symbolic Automata

- Why L^* cannot be applied?
- Our Solution
- The Algorithm

Equivalence Queries and Counter-Examples

Adaptation to the Boolean Alphabet

Experimental Results

Conclusion

Outline

Preliminaries

Regular Languages and Automata

The L^* Algorithmic Scheme

Large Alphabets

Motivation

Symbolic Representation of Transitions - Symbolic Automata

Learning Symbolic Automata

Why L^* cannot be applied?

Our Solution

The Algorithm

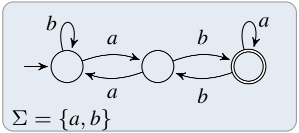
Equivalence Queries and Counter-Examples

Adaptation to the Boolean Alphabet

Experimental Results

Conclusion

Regular Languages and Automata



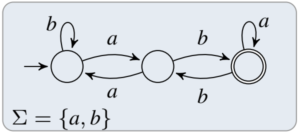
$\Sigma = \{a, b\}$

$L \subseteq \Sigma^*$ is a *language*

- Σ is an *alphabet*
- $w = a_1 \cdots a_n$ is a *word*
- Σ^* is the set of all words

						suffixes				
		ε	a	b	aa	ab	ba	bb	aaa	$\dots$
	ε	$-$	$-$	$-$	$-$	$+$	$-$	$-$	$-$	$\dots$
	a	$-$	$-$	$+$	$-$	$-$	$+$	$-$	$-$	$\dots$
	b	$-$	$-$	$-$	$-$	$+$	$-$	$-$	$-$	$\dots$
	aa	$-$	$-$	$-$	$-$	$+$	$-$	$-$	$-$	$\dots$
prefixes	ab	$+$	$+$	$-$	$+$	$-$	$-$	$+$	$+$	$\dots$
	ba	$-$	$-$	$+$	$-$	$-$	$+$	$-$	$-$	$\dots$
	bb	$-$	$-$	$-$	$-$	$+$	$-$	$-$	$-$	$\dots$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$
	aba	$+$	$+$	$-$	$+$	$-$	$-$	$+$	$+$	$\dots$
	abb	$-$	$-$	$+$	$-$	$-$	$+$	$-$	$-$	$\dots$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Regular Languages and Automata



$\Sigma = \{a, b\}$

$L \subseteq \Sigma^*$ is a *language*

Equivalence relation

$u \sim_L v$ iff $u \cdot w \in L \Leftrightarrow v \cdot w \in L$

Nerode's Theorem

L is a regular language iff $\sim_L$ has finitely many equivalence classes.

$Q = \Sigma^* / _\sim$ (states in the minimal representation of L.

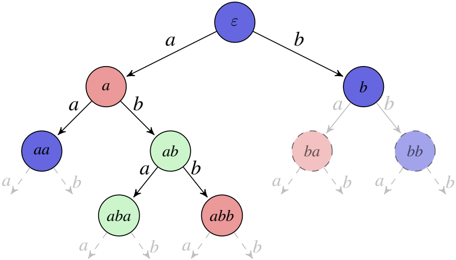
	suffixes								
prefixes	ε	a	b	aa	ab	ba	bb	aaa	…
ε	−	−	−	−	+	−	−	−	…
a	−	−	+	−	−	+	−	−	…
b	−	−	−	−	+	−	−	−	…
aa	−	−	−	−	+	−	−	−	…
ab	+	+	−	+	−	−	+	+	…
ba	−	−	+	−	−	+	−	−	…
bb	−	−	−	−	+	−	−	−	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱
aba	+	+	−	+	−	−	+	+	…
abb	−	−	+	−	−	+	−	−	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

$\varepsilon \sim b \sim aa \quad a \sim ba \sim abb \quad ab \sim aba$

Regular Languages and Automata

A sufficient sample that characterizes the language

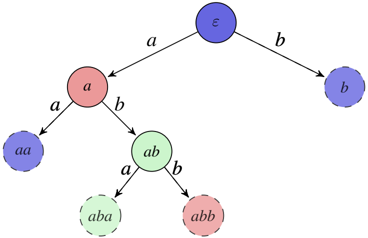
	ε	a	b
ε	−	−	−
a	−	−	+
ab	+	+	−
b	−	−	−
aa	−	−	−
aba	+	+	−
abb	−	−	+



Regular Languages and Automata

A sufficient sample that characterizes the language

		E		
		ε	a	b
S	ε	$-$	$-$	$-$
	a	$-$	$-$	$+$
	ab	$+$	$+$	$-$
R	b	$-$	$-$	$-$
	aa	$-$	$-$	$-$
	aba	$+$	$+$	$-$
	abb	$-$	$-$	$+$



S *prefixes* (states)

R *boundary* ($R = S \cdot \Sigma \setminus S$)

E *suffixes* (distinguishing strings)

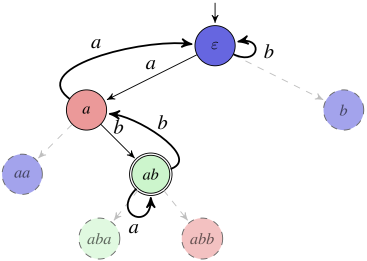
$f : S \cup R \times E \to \{+, -\}$ *classif. function*

$f_s : E \to \{+, -\}$ *residual functions*

Regular Languages and Automata

A sufficient sample that characterizes the language

		E: ε	a	b
S	ε	$-$	$-$	$-$
	a	$-$	$-$	$+$
	ab	$+$	$+$	$-$
R	b	$-$	$-$	$-$
	aa	$-$	$-$	$-$
	aba	$+$	$+$	$-$
	abb	$-$	$-$	$+$



S *prefixes* (states)

R *boundary* ($R = S \cdot \Sigma \setminus S$)

E *suffixes* (distinguishing strings)

$f : S \cup R \times E \to \{+, -\}$ *classif. function*

$f_s : E \to \{+, -\}$ *residual functions*

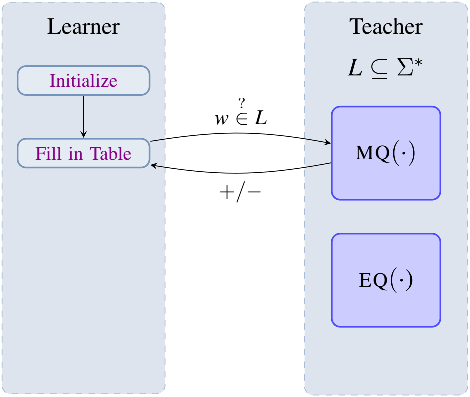
$\mathcal{A}_L = (\Sigma, Q, q_0, \delta, F)$

- $Q = S$
- $q_0 = [\varepsilon]$
- $\delta([u], a) = [u \cdot a]$
- $F = \{[u] : (u \cdot \varepsilon) \in L\}$

The minimal automaton for L

The L^* Algorithmic Scheme*

Active learning using queries



Learner

Initialize

Fill in Table

$w \overset{?}{\in} L$

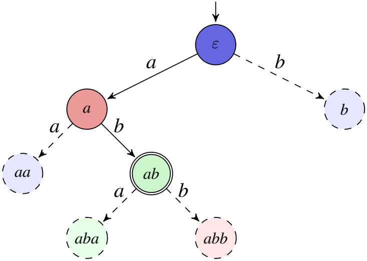
$+/-$

Teacher

$L \subseteq \Sigma^*$

$\mathrm{MQ}(\cdot)$

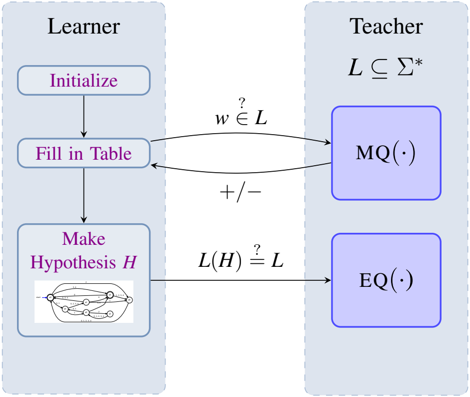
$\mathrm{EQ}(\cdot)$



[*] D. Angluin. *Learning regular sets from queries and counter-examples*, 1987.

The L^* Algorithmic Scheme[*]

Active learning using queries



Learner

Initialize

Fill in Table

Make Hypothesis H

Teacher

$L \subseteq \Sigma^*$

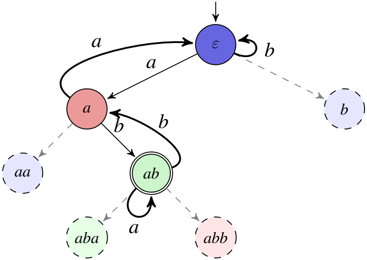
$w \overset{?}{\in} L$

$+/-$

$MQ(\cdot)$

$L(H) \overset{?}{=} L$

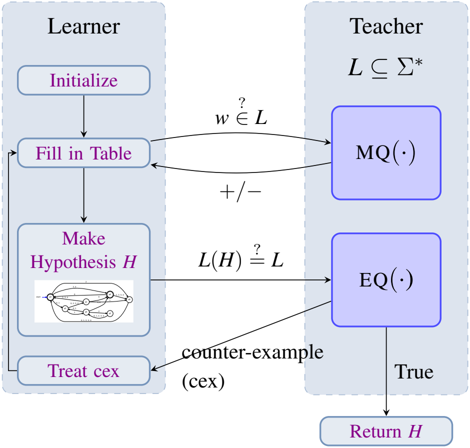
$EQ(\cdot)$



[*] D. Angluin. *Learning regular sets from queries and counter-examples*, 1987.

The L^* Algorithmic Scheme[*]

Active learning using queries



Learner

Initialize

Fill in Table

Make Hypothesis H

Treat cex

$w \overset{?}{\in} L$

$+/-$

$L(H) \overset{?}{=} L$

counter-example (cex)

Teacher

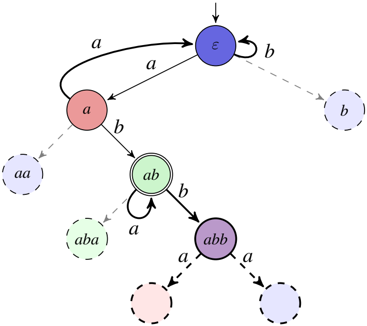
$L \subseteq \Sigma^*$

$\mathrm{MQ}(\cdot)$

$\mathrm{EQ}(\cdot)$

True

Return H



[*] D. Angluin. *Learning regular sets from queries and counter-examples*, 1987.

Outline

Preliminaries

- Regular Languages and Automata
- The L^* Algorithmic Scheme

Large Alphabets

- Motivation
- Symbolic Representation of Transitions - Symbolic Automata

Learning Symbolic Automata

- Why L^* cannot be applied?
- Our Solution
- The Algorithm

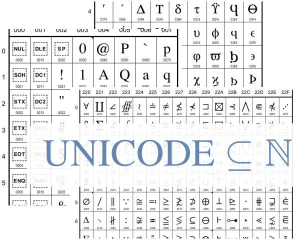
Equivalence Queries and Counter-Examples

Adaptation to the Boolean Alphabet

Experimental Results

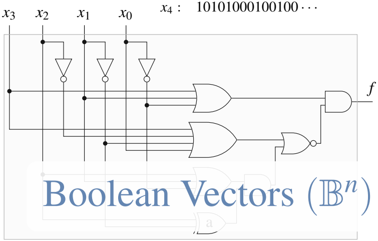
Conclusion

Languages over Large Alphabets

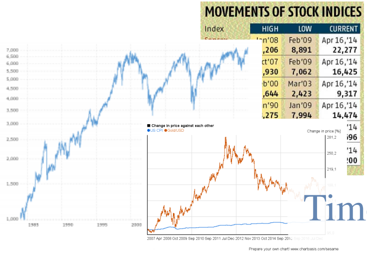


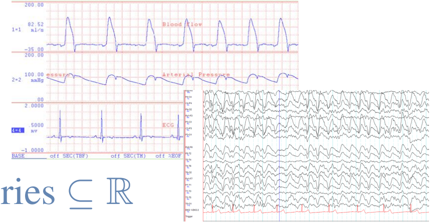
UNICODE $\subseteq \mathbb{N}$

Input:
x_1 : 10101010000100 · · ·
x_2 : 10100100100100 · · ·
x_3 : 10101000010001 · · ·
x_4 : 10101000100100 · · ·



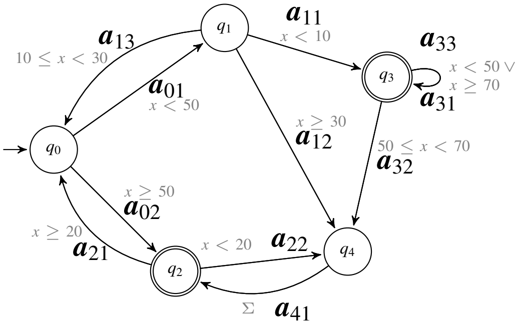
Boolean Vectors $(\mathbb{B}^n)$





Time Series $\subseteq \mathbb{R}$

Symbolic Automata



$$\mathcal{A} = (\Sigma, \boldsymbol{\Sigma}, \psi, Q, \boldsymbol{\delta}, q_0, F)$$

- Q finite set of states,
- q_0 initial state,
- F accepting states,
- Σ large concrete alphabet,
- $\boldsymbol{\delta} \subseteq Q \times \boldsymbol{\Sigma} \times Q$
- $\boldsymbol{\Sigma}$ finite alphabet (symbols)
- $\psi_q : \Sigma \rightarrow \boldsymbol{\Sigma}_q,\, q \in Q$
- $[\![\boldsymbol{a}]\!] = \{a \in \Sigma \mid \psi(a) = \boldsymbol{a}\}$

$\Sigma \subseteq \mathbb{R}$

$[\![\boldsymbol{a}_{01}]\!] = \{x \in \Sigma : x < 50\}$

$(w = 20 \cdot 40 \cdot 60\,,\, +)$

$\boldsymbol{w} = \boldsymbol{a}_{01} \cdot \boldsymbol{a}_{12} \cdot \boldsymbol{a}_{41}$

$\mathcal{A}$ is complete and deterministic if $\forall q \in Q$ $\{[\![\boldsymbol{a}]\!] \mid \boldsymbol{a} \in \boldsymbol{\Sigma}_q\}$ forms a *partition of* Σ.

Outline

Preliminaries

- Regular Languages and Automata
- The L^* Algorithmic Scheme

Large Alphabets

- Motivation
- Symbolic Representation of Transitions - Symbolic Automata

Learning Symbolic Automata

- Why L^* cannot be applied?
- Our Solution
- The Algorithm

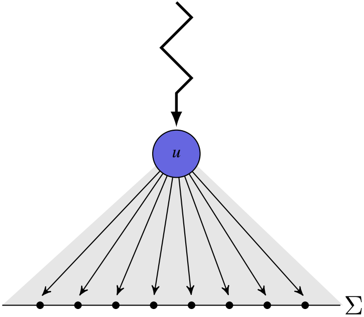
Equivalence Queries and Counter-Examples

Adaptation to the Boolean Alphabet

Experimental Results

Conclusion

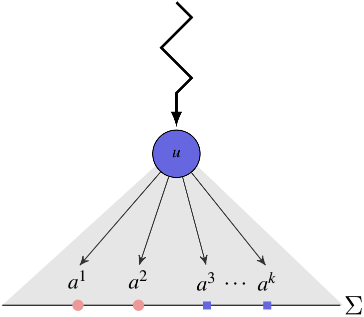
Learning over Large Alphabets



Why L^* cannot be applied?

- The learner asks MQ's for all continuations of a state ($\forall a \in \Sigma$, ask $\text{MQ}(u \cdot a)$)
- Inefficient for large finite alphabets
- Not applicable to infinite alphabets

Learning over Large Alphabets



Why L^* cannot be applied?

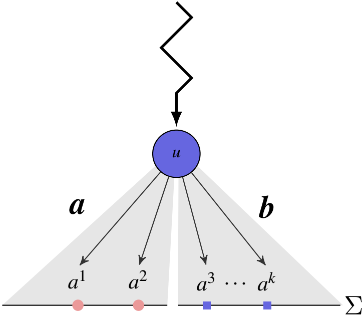
- The learner asks MQ's for all continuations of a state ($\forall a \in \Sigma$, ask $\text{MQ}(u \cdot a)$)
- Inefficient for large finite alphabets
- Not applicable to infinite alphabets

Our solution:

- Use a finite sample of evidences to learn the transitions

Evidences: $\mu(\boldsymbol{a}) = \{a^1, a^2\}$

Learning over Large Alphabets



Evidences: $\mu(\boldsymbol{a}) = \{a^1, a^2\}$

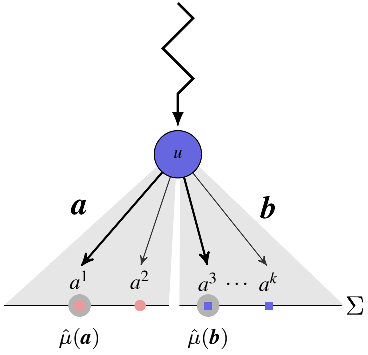
Why L^* cannot be applied?

- The learner asks MQ's for all continuations of a state ($\forall a \in \Sigma$, ask $\text{MQ}(u \cdot a)$)
- Inefficient for large finite alphabets
- Not applicable to infinite alphabets

Our solution:

- Use a finite sample of evidences to learn the transitions
- Form evidence compatible partitions
- Associate a symbol to each partition block

Learning over Large Alphabets



Evidences: $\mu(\boldsymbol{a}) = \{a^1, a^2\}$

Representative: $\hat{\mu}(\boldsymbol{a}) = a^1$

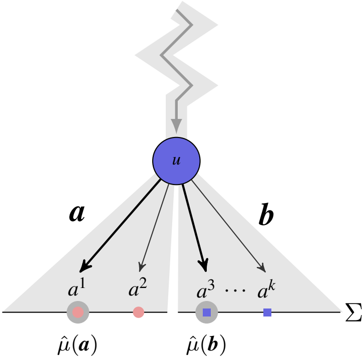
Why L^* cannot be applied?

- The learner asks MQ's for all continuations of a state ($\forall a \in \Sigma$, ask $\text{MQ}(u \cdot a)$)
- Inefficient for large finite alphabets
- Not applicable to infinite alphabets

Our solution:

- Use a finite sample of evidences to learn the transitions
- Form evidence compatible partitions
- Associate a symbol to each partition block
- Each symbol has one representative evidence

Learning over Large Alphabets



Evidences: $\mu(\boldsymbol{a}) = \{a^1, a^2\}$

Representative: $\hat{\mu}(\boldsymbol{a}) = a^1$

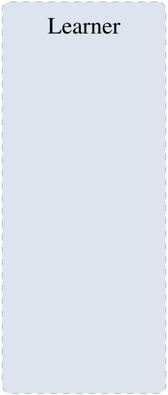
Why L^* cannot be applied?

- The learner asks MQ's for all continuations of a state ($\forall a \in \Sigma$, ask $\text{MQ}(u \cdot a)$)
- Inefficient for large finite alphabets
- Not applicable to infinite alphabets

Our solution:

- Use a finite sample of evidences to learn the transitions
- Form evidence compatible partitions
- Associate a symbol to each partition block
- Each symbol has one representative evidence
- The prefixes are symbolic

Symbolic Learning Algorithm

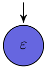


Learner

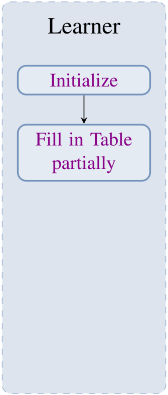
Symbolic Learning Algorithm

Learner

Initialize



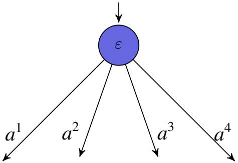
Symbolic Learning Algorithm



Learner

Initialize

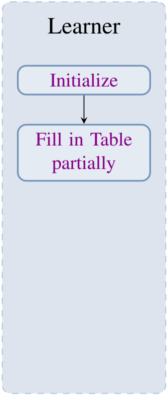
Fill in Table partially

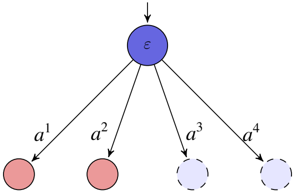


Repeat for each new state q:

- Sample *evidences*

Symbolic Learning Algorithm

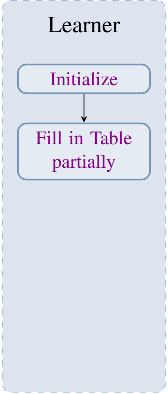




Repeat for each new state q:

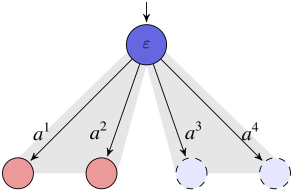
- Sample *evidences*
- Ask MQ*'s*

Symbolic Learning Algorithm


Learner

Initialize

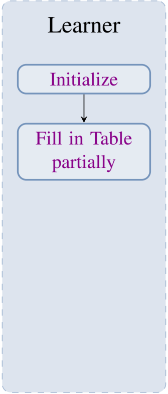
Fill in Table partially



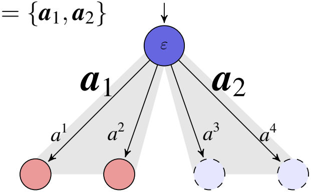
Repeat for each new state q:

- Sample *evidences*
- Ask MQ*'s*
- Learn *partitions*

Symbolic Learning Algorithm



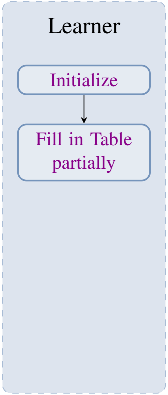
$\Sigma_\varepsilon = \{\boldsymbol{a}_1, \boldsymbol{a}_2\}$



Repeat for each new state q:

- Sample *evidences*
- Ask MQ*'s*
- Learn *partitions*
- Define the *symbolic alphabet* $\boldsymbol{\Sigma}_q$

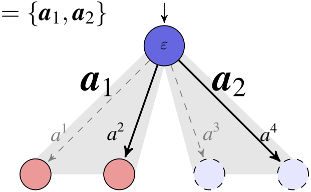
Symbolic Learning Algorithm


Learner

Initialize

Fill in Table partially

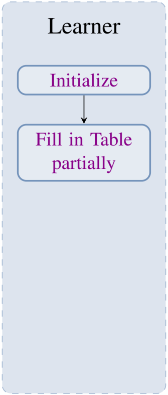
$\boldsymbol{\Sigma}_\varepsilon = \{\boldsymbol{a}_1, \boldsymbol{a}_2\}$



Repeat for each new state q:

- Sample *evidences*
- Ask MQ's
- Learn *partitions*
- Define the *symbolic alphabet* $\boldsymbol{\Sigma}_q$
- Select *representative* $\hat{\mu}(\boldsymbol{a})$, $\forall \boldsymbol{a} \in \boldsymbol{\Sigma}_q$

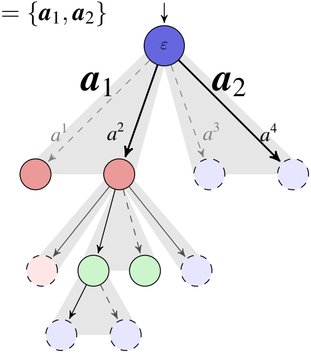
Symbolic Learning Algorithm


Learner

Initialize

Fill in Table partially

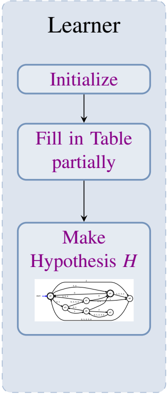
$\Sigma_\varepsilon = \{\boldsymbol{a}_1, \boldsymbol{a}_2\}$



Repeat for each new state q:

- Sample *evidences*
- Ask MQ*'s*
- Learn *partitions*
- Define the *symbolic alphabet* $\boldsymbol{\Sigma}_q$
- Select *representative* $\hat{\mu}(\boldsymbol{a})$, $\forall \boldsymbol{a} \in \boldsymbol{\Sigma}_q$

Symbolic Learning Algorithm



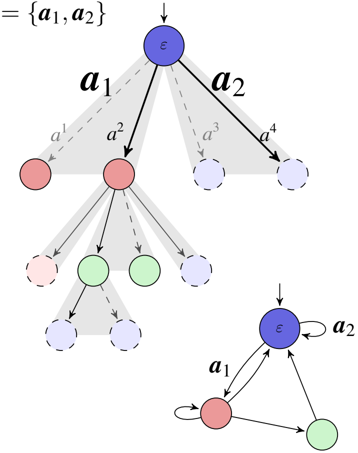
Learner

Initialize

Fill in Table partially

Make Hypothesis *H*

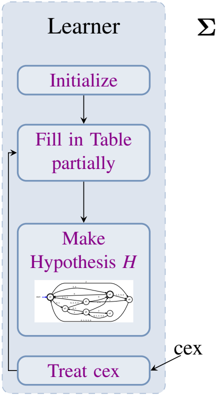
$\boldsymbol{\Sigma}_\varepsilon = \{\boldsymbol{a}_1, \boldsymbol{a}_2\}$



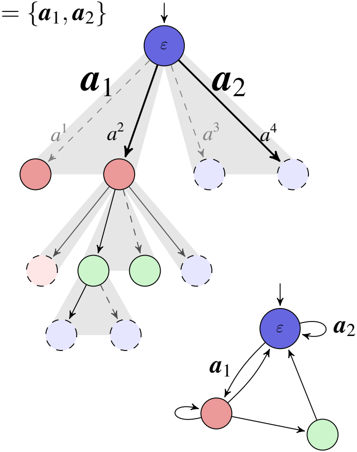
Repeat for each new state q:

- Sample *evidences*
- Ask MQ's
- Learn *partitions*
- Define the *symbolic alphabet* $\boldsymbol{\Sigma}_q$
- Select *representative* $\hat{\mu}(\boldsymbol{a})$, $\forall \boldsymbol{a} \in \boldsymbol{\Sigma}_q$

Symbolic Learning Algorithm



$\boldsymbol{\Sigma}_\varepsilon = \{\boldsymbol{a}_1, \boldsymbol{a}_2\}$

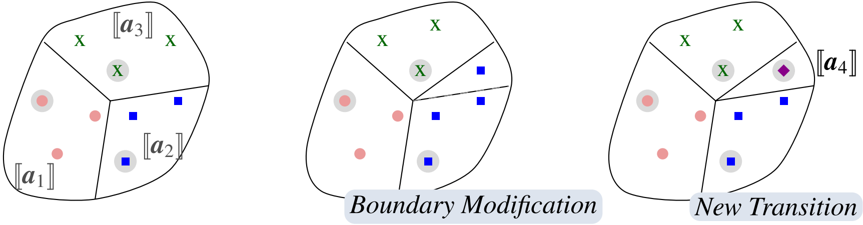


Repeat for each new state q:

- Sample *evidences*
- Ask MQ’*s*
- Learn *partitions*
- Define the *symbolic alphabet* $\boldsymbol{\Sigma}_q$
- Select *representative* $\hat{\mu}(\boldsymbol{a})$, $\forall \boldsymbol{a} \in \boldsymbol{\Sigma}_q$

Evidence Compatibility

Solve Incompatibility


$[\![\boldsymbol{a}_3]\!]$
x
x
x
$[\![\boldsymbol{a}_2]\!]$
$[\![\boldsymbol{a}_1]\!]$
$[\![\boldsymbol{a}_4]\!]$

Boundary Modification

New Transition

Evidence Compatibility

A state $\boldsymbol{u}$ is *evidence compatible* when

$$f_{\boldsymbol{u}\cdot a} = f_{\boldsymbol{u}\cdot\hat{\mu}(\boldsymbol{a})}$$

for every evidence $a \in [\![\boldsymbol{a}]\!]$

Evidence incompatibility at state $\boldsymbol{u}$

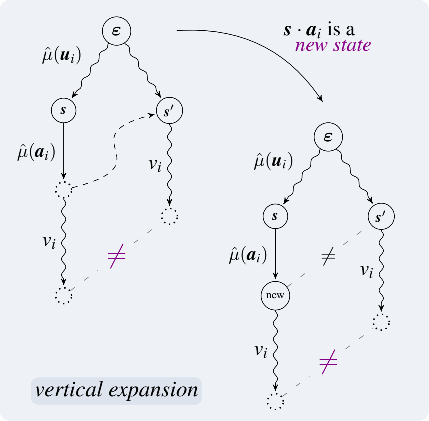
		v	
		$\vdots$	
$\boldsymbol{u}\cdot\hat{\mu}(\boldsymbol{a})$	$\cdots$	$+$	$\cdots$
$\boldsymbol{u}\cdot a$	$\cdots$	$-$	$\cdots$

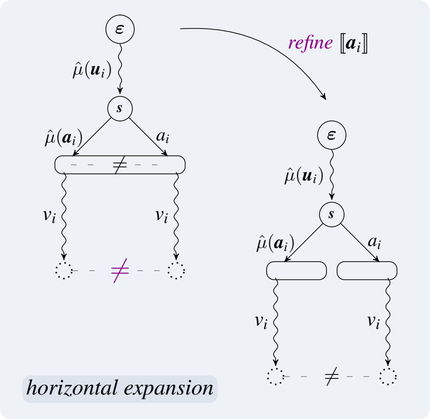
Counter-example Treatment (Symbolic Breakpoint)

Let $w = a_1 \cdot \cdots \cdot a_i \cdots a_{|w|} = u_i \cdot a_i \cdot v_i$ be a counter-example.

$$f(\hat{\mu}(\boldsymbol{s}_{i-1} \cdot \boldsymbol{a}_i) \cdot v_i) \neq f(\hat{\mu}(\boldsymbol{s}_i) \cdot v_i) \qquad f(\hat{\mu}(\boldsymbol{s}_{i-1}) \cdot a_i \cdot v_i) \neq f(\hat{\mu}(\boldsymbol{s}_{i-1}) \cdot \hat{\mu}(\boldsymbol{a}_i) \cdot v_i)$$

$$\boldsymbol{s_i} = \boldsymbol{\delta}(\boldsymbol{\varepsilon}, u_i \cdot a_i)$$


ε
$\hat{\mu}(\boldsymbol{u}_i)$
$\boldsymbol{s}$
$\boldsymbol{s}'$
$\hat{\mu}(\boldsymbol{a}_i)$
v_i
v_i
$\neq$
$\boldsymbol{s} \cdot \boldsymbol{a}_i$ is a *new state*
ε
$\hat{\mu}(\boldsymbol{u}_i)$
$\boldsymbol{s}$
$\boldsymbol{s}'$
$\hat{\mu}(\boldsymbol{a}_i)$
$\neq$
v_i
new
v_i
$\neq$
vertical expansion

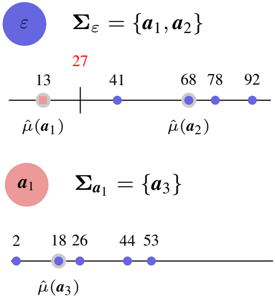

ε
$\hat{\mu}(\boldsymbol{u}_i)$
$\boldsymbol{s}$
$\hat{\mu}(\boldsymbol{a}_i)$
a_i
$\neq$
v_i
v_i
$\neq$
refine $[\![\boldsymbol{a}_i]\!]$
ε
$\hat{\mu}(\boldsymbol{u}_i)$
$\boldsymbol{s}$
$\hat{\mu}(\boldsymbol{a}_i)$
a_i
v_i
v_i
$\neq$
horizontal expansion

Example over the alphabet $\Sigma = [1, 100)$

observation table

	ε
ε	$-$
13 a_1	$+$
68 a_2	$-$
13 18 $a_1 a_3$	$-$

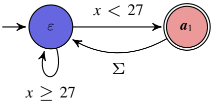
semantics



$\Sigma_\varepsilon = \{a_1, a_2\}$

$\Sigma_{a_1} = \{a_3\}$

hypothesis automaton



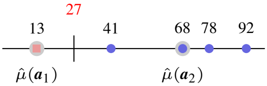
Example over the alphabet $\Sigma = [1, 100)$

observation table

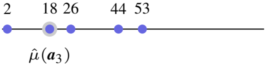
	ε	11
ε	$-$	$+$
13 a_1	$+$	$-$
68 a_2	$-$	$-$
13 18 a_1a_3	$-$	$+$

semantics

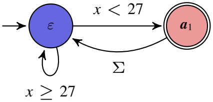
ε $\Sigma_\varepsilon = \{a_1, a_2\}$



a_1 $\Sigma_{a_1} = \{a_3\}$



hypothesis automaton



Ask Equivalence Query:

counter-example:

$w = 35 \cdot 52 \cdot 11, -$

add distinguishing string 11

discover new state

(vertical expansion)

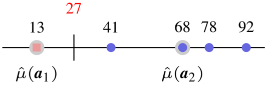
Example over the alphabet $\Sigma = [1, 100)$

observation table

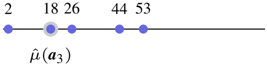
	ε	11
ε	$-$	$+$
13 a_1	$+$	$-$
68 a_2	$-$	$-$
13 18 $a_1 a_3$	$-$	$+$
68 17 $a_2 a_4$	$-$	$-$
68 75 $a_2 a_5$	$+$	$-$

semantics

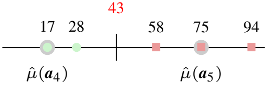
$$\boldsymbol{\Sigma}_{\varepsilon} = \{\boldsymbol{a}_1, \boldsymbol{a}_2\}$$



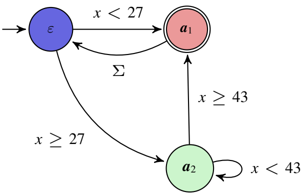
$$\boldsymbol{\Sigma}_{\boldsymbol{a}_1} = \{\boldsymbol{a}_3\}$$



$$\boldsymbol{\Sigma}_{\boldsymbol{a}_2} = \{\boldsymbol{a}_4, \boldsymbol{a}_5\}$$



hypothesis automaton



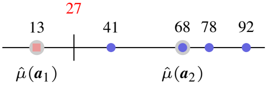
Example over the alphabet $\Sigma = [1, 100)$

observation table

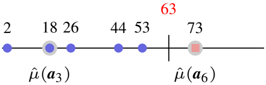
	ε	11
ε	$-$	$+$
13 a_1	$+$	$-$
68 a_2	$-$	$-$
13 18 $a_1 a_3$	$-$	$+$
13 73 $a_1 a_6$	$+$	$-$
68 17 $a_2 a_4$	$-$	$-$
68 75 $a_2 a_5$	$+$	$-$

semantics

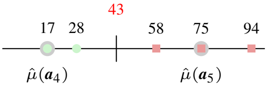
ε $\quad \Sigma_\varepsilon = \{a_1, a_2\}$



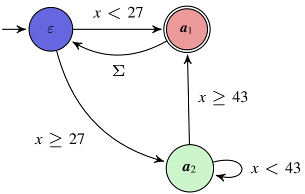
a_1 $\quad \Sigma_{a_1} = \{a_3, a_6\}$



a_2 $\quad \Sigma_{a_2} = \{a_4, a_5\}$



hypothesis automaton



Ask Equivalence Query:

counter-example:

$w = 12 \cdot 73 \cdot 4, -$

add 73 as evidence of a_1

add new transition
(horizontal expansion)

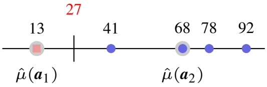
Example over the alphabet $\Sigma = [1, 100)$

observation table

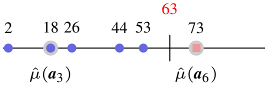
	ε	11
ε	$-$	$+$
13 a_1	$+$	$-$
68 a_2	$-$	$-$
13 18 $a_1 a_3$	$-$	$+$
13 73 $a_1 a_6$	$+$	$-$
68 17 $a_2 a_4$	$-$	$-$
68 75 $a_2 a_5$	$+$	$-$

semantics

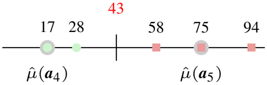
ε $\quad \Sigma_{\varepsilon} = \{a_1, a_2\}$



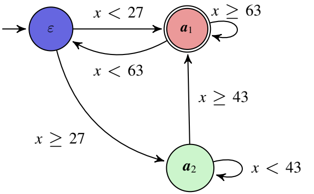
a_1 $\quad \Sigma_{a_1} = \{a_3, a_6\}$



a_2 $\quad \Sigma_{a_2} = \{a_4, a_5\}$



hypothesis automaton



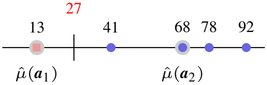
Example over the alphabet $\Sigma = [1, 100)$

observation table

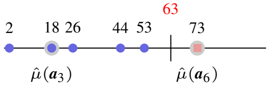
	ε	11
ε	−	+
13 a_1	+	−
68 a_2	−	−
13 18 a_1a_3	−	+
13 73 a_1a_6	+	−
68 17 a_2a_4	−	−
68 75 a_2a_5	+	−

semantics

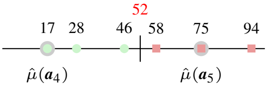
ε $\quad \Sigma_\varepsilon = \{a_1, a_2\}$



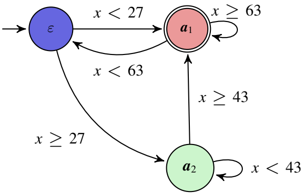
a_1 $\quad \Sigma_{a_1} = \{a_3, a_6\}$



a_2 $\quad \Sigma_{a_2} = \{a_4, a_5\}$



hypothesis automaton



Ask Equivalence Query:

counter-example: $w = 52 \cdot 46, -$

add 46 as evidence of a_2

refine existing transition

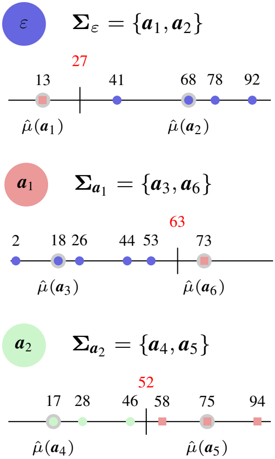
(horizontal expansion)

Example over the alphabet $\Sigma = [1, 100)$

observation table

	ε	11
ε	$-$	$+$
13 a_1	$+$	$-$
68 a_2	$-$	$-$
13 18 a_1a_3	$-$	$+$
13 73 a_1a_6	$+$	$-$
68 17 a_2a_4	$-$	$-$
68 75 a_2a_5	$+$	$-$

semantics

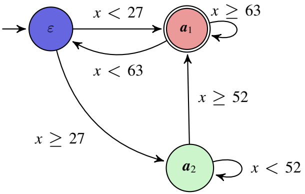


$\Sigma_\varepsilon = \{a_1, a_2\}$

$\Sigma_{a_1} = \{a_3, a_6\}$

$\Sigma_{a_2} = \{a_4, a_5\}$

hypothesis automaton



Ask Equivalence Query:
True

return current hypothesis

return hypothesis

Outline

Preliminaries
- Regular Languages and Automata
- The L^* Algorithmic Scheme

Large Alphabets
- Motivation
- Symbolic Representation of Transitions - Symbolic Automata

Learning Symbolic Automata
- Why L^* cannot be applied?
- Our Solution
- The Algorithm

Equivalence Queries and Counter-Examples

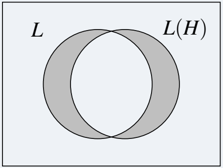
Adaptation to the Boolean Alphabet

Experimental Results

Conclusion

Equivalence Queries and Counter-Examples

What is the error?



All $w \in L \oplus L(H)$ are counter-examples

A helpful teacher can compute $L \oplus L(H)$ to find counter-examples.

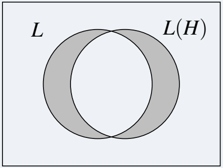
When the teacher provides *minimal* counter-examples (i.e., minimal in length-lexicographic order), then

- one evidence per partition is used
- the boundaries are exactly determined
- final hypothesis contains no error

The algorithm terminates with a correct conjecture after asking at most $\mathcal{O}(mn^2)$ MQ's and at most $\mathcal{O}(mn)$ EQ's, when Σ is totally-ordered.

Equivalence Queries and Counter-Examples

What is the error?



All $w \in L \oplus L(H)$ are counter-examples

In the absence of a helpful teacher and the learner can use only MQ's

EQ's are approximated by testing:

- select a set of words randomly
- ask MQ's for them
- check if the result matches with H
- return counter-example

A hypothesis automaton H is *Probably Approximately Correct* (PAC) iff

$$Pr(\mathcal{P}(L \oplus L(H)) < \epsilon) > 1 - \delta.$$

Sufficient tests for a hypothesis H_i to be PAC: $r_i = \frac{1}{\epsilon}(\ln \frac{1}{\delta} + (i + 1) \ln 2)$.
$[Ang87]$

Outline

Preliminaries

- Regular Languages and Automata
- The L^* Algorithmic Scheme

Large Alphabets

- Motivation
- Symbolic Representation of Transitions - Symbolic Automata

Learning Symbolic Automata

- Why L^* cannot be applied?
- Our Solution
- The Algorithm

Equivalence Queries and Counter-Examples

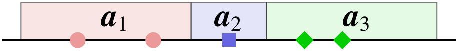
Adaptation to the Boolean Alphabet

Experimental Results

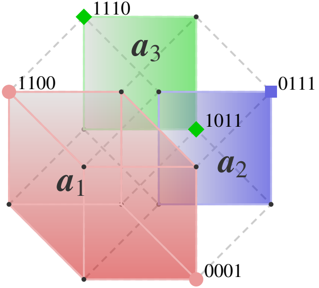
Conclusion

Adaptation to the Boolean Alphabet

Partition of $\mathbb{R}$ (or $\mathbb{N}$) into finite number of intervals



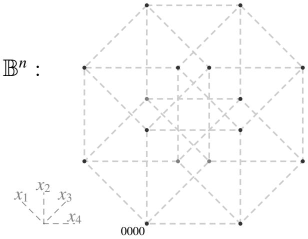
Partition of $\mathbb{B}^n$ into finite number of cubes

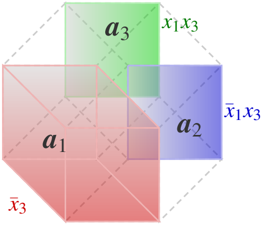


Adaptation to the Boolean Alphabet

Representations of the Boolean Cube

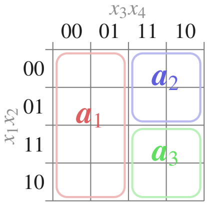
$\psi : \mathbb{B}^4 \to \{\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3\}$



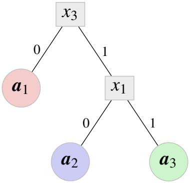


$$\psi(a) = \begin{cases} \boldsymbol{a}_1, & \text{if } \bar{x}_3 \\ \boldsymbol{a}_2, & \text{if } \bar{x}_1 \cdot x_3 \\ \boldsymbol{a}_3, & \text{if } x_1 \cdot x_3 \end{cases}$$

Boolean Function



Karnaugh map



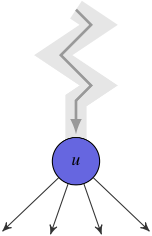
Binary Decision Tree

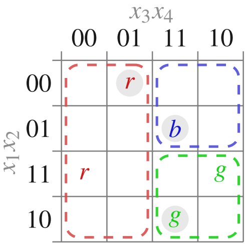
Adaptation to the Boolean Alphabet

Learning Partitions

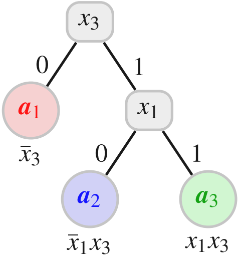
$\Sigma = \mathbb{B}^4$

Learning Binary Decision Trees using the Greedy Splitting Algorithm CART[†]





x_1x_2 \ x_3x_4	00	01	11	10
00		r		
01			b	
11	r			g
10			g	



Best split: x_1

$$\psi(a) = \begin{cases} \boldsymbol{a}_1, & \text{if } \bar{x}_3 \\ \boldsymbol{a}_2, & \text{if } \bar{x}_1 \cdot x_3 \\ \boldsymbol{a}_3, & \text{if } x_1 \cdot x_3 \end{cases}$$

Use Information Gain (Entropy) Measure to find Best Split

[*] Breiman et al. *Classification and regression trees*, 1984.

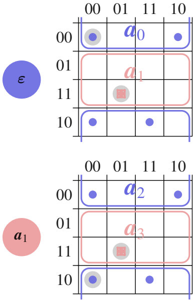
Adaptation to the Boolean Alphabet

Example over $\Sigma = \mathbb{B}^4$

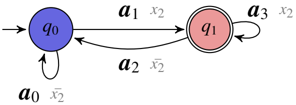
observation table

	ε
ε	$-$
a_1	$+$
a_0	$-$
$a_1 a_2$	$-$
$a_1 a_3$	$+$

semantics



hypothesis automaton



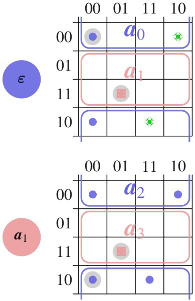
Adaptation to the Boolean Alphabet

Example over $\Sigma = \mathbb{B}^4$

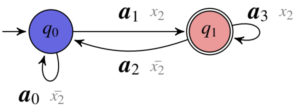
observation table

	ε	0000
ε	$-$	$-$
a_1	$+$	$-$
a_0	$-$	$-$
$a_1 a_2$	$-$	$+$
$a_1 a_3$	$+$	$-$

semantics



hypothesis automaton



Ask Equivalence Query:

counter-example:

$w = (1010) \cdot (0000)\ ,\ +$

$\boldsymbol{w} = \boldsymbol{a}_0 \cdot \boldsymbol{a}_0\ ,\ -$

add distinguishing string (0000)

discover new state

evidence incompatibility

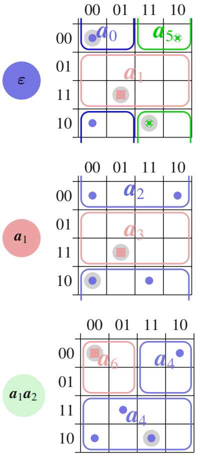
Adaptation to the Boolean Alphabet

Example over $\Sigma = \mathbb{B}^4$

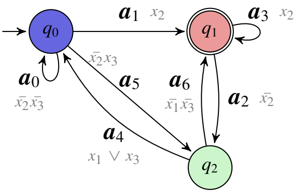
observation table

	ε	0000
ε	$-$	$-$
a_1	$+$	$-$
a_1a_2	$-$	$+$
a_0	$-$	$-$
a_5	$-$	$+$
a_1a_3	$+$	$-$
$a_1a_2a_4$	$-$	$-$
$a_1a_2a_6$	$+$	$-$

semantics



hypothesis automaton



Ask Equivalence Query:

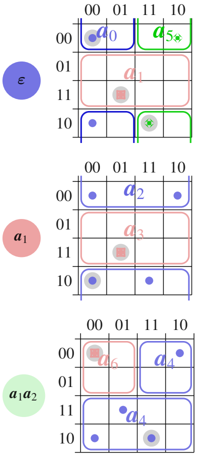
Adaptation to the Boolean Alphabet

Example over $\Sigma = \mathbb{B}^4$

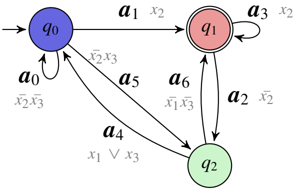
observation table

	ε	0000
ε	$-$	$-$
a_1	$+$	$-$
a_1a_2	$-$	$+$
a_0	$-$	$-$
a_5	$-$	$+$
a_1a_3	$+$	$-$
$a_1a_2a_4$	$-$	$-$
$a_1a_2a_6$	$+$	$-$

semantics



hypothesis automaton



Ask Equivalence Query:

True

terminate: Return H

Outline

Preliminaries

- Regular Languages and Automata
- The L^* Algorithmic Scheme

Large Alphabets

- Motivation
- Symbolic Representation of Transitions - Symbolic Automata

Learning Symbolic Automata

- Why L^* cannot be applied?
- Our Solution
- The Algorithm

Equivalence Queries and Counter-Examples

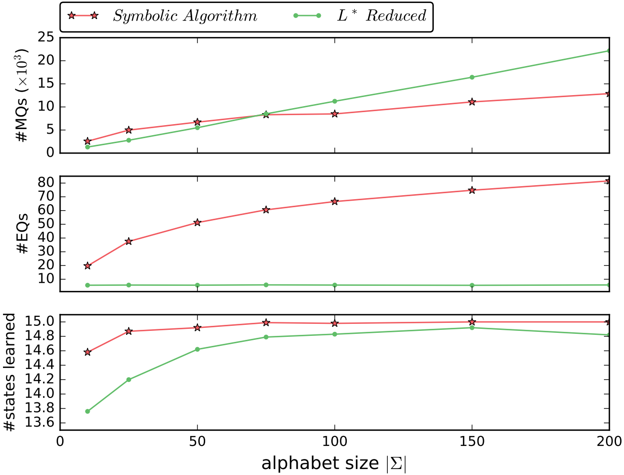
Adaptation to the Boolean Alphabet

Experimental Results

Conclusion

Empirical Results

Comparison to the best L^* algorithm[‡]



Experiment:

Target automaton:

- $\Sigma \subseteq \mathbb{N}$
- $10 \leq |\Sigma| \leq 200$
- $|Q| = 15$,
- $|\mathbf{\Sigma}_q| \leq 5, \forall q \in Q$

Structure is fixed

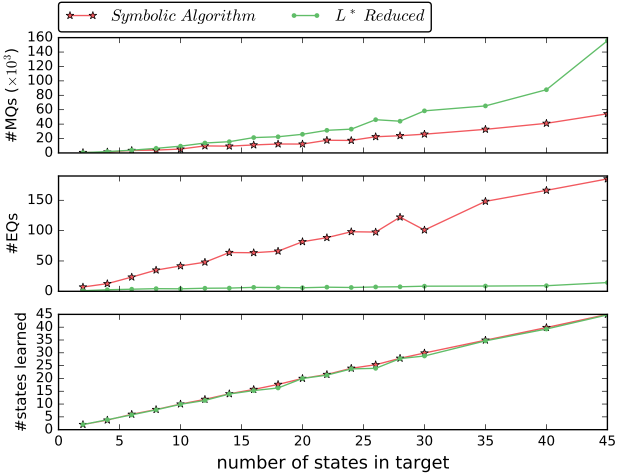
PAC criterion for $\epsilon = \delta = 0.05$

MQ's = MQ's for learning + MQ's for testing

[‡] Rivest and Schapire. *Inference of finite automata using homing sequences*, 1993.

Empirical Results

Comparison to the best L^* algorithm[§]



Experiment:

Target automaton:

- $\Sigma \subseteq \mathbb{N}$
- $|\Sigma| = 150$
- $3 \leq |Q| \leq 45$
- $|\mathbf{\Sigma}_q| \leq 5, \forall q \in Q$

Random structure

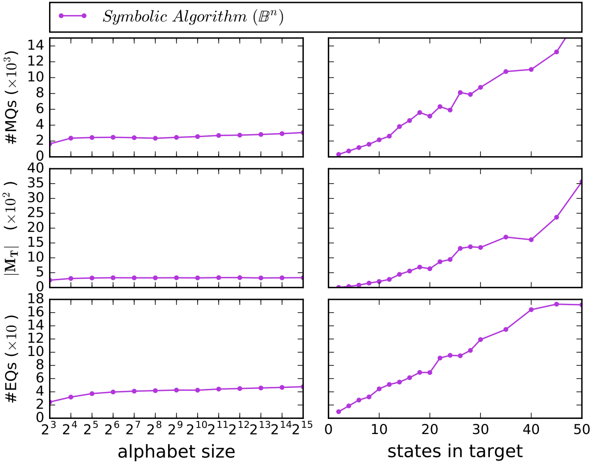
PAC criterion for $\epsilon = \delta = 0.05$

MQ's = MQ's for learning + MQ's for testing

[§] Rivest and Schapire. *Inference of finite automata using homing sequences*, 1993.

Empirical Results

Applying the symbolic algorithm over the Booleans



Experiment:

Target automaton:

Left: $|Q| = 15$
$2^3 \leq |\Sigma| \leq 2^{15}$

Right: $|\Sigma| = \mathbb{B}^8$
$3 \leq |Q| \leq 50$

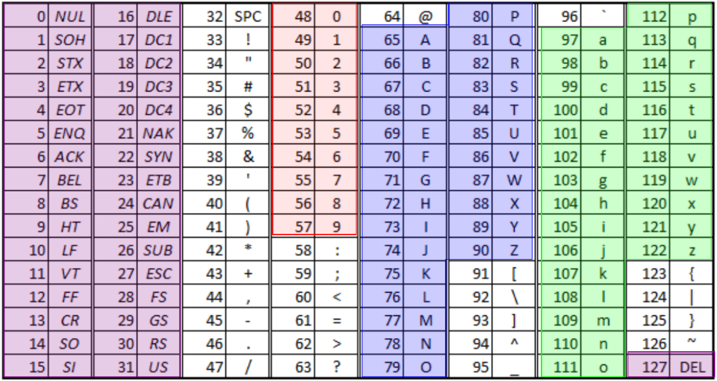
BDTs depth ≤ 4, $\forall q \in Q$

PAC criterion for $\epsilon = \delta = 0.05$

MQ's = MQ's for learning + MQ's for testing

Empirical Results

Valid passwords over the ASCII characters



0	*NUL*	16	*DLE*	32	SPC	48	0	64	@	80	P	96	`	112	p
1	*SOH*	17	*DC1*	33	!	49	1	65	A	81	Q	97	a	113	q
2	*STX*	18	*DC2*	34	"	50	2	66	B	82	R	98	b	114	r
3	*ETX*	19	*DC3*	35	#	51	3	67	C	83	S	99	c	115	s
4	*EOT*	20	*DC4*	36	$	52	4	68	D	84	T	100	d	116	t
5	*ENQ*	21	*NAK*	37	%	53	5	69	E	85	U	101	e	117	u
6	*ACK*	22	*SYN*	38	&	54	6	70	F	86	V	102	f	118	v
7	*BEL*	23	*ETB*	39	'	55	7	71	G	87	W	103	g	119	w
8	*BS*	24	*CAN*	40	(	56	8	72	H	88	X	104	h	120	x
9	*HT*	25	*EM*	41	)	57	9	73	I	89	Y	105	i	121	y
10	*LF*	26	*SUB*	42	*	58	:	74	J	90	Z	106	j	122	z
11	*VT*	27	*ESC*	43	+	59	;	75	K	91	[	107	k	123	{
12	*FF*	28	*FS*	44	,	60	<	76	L	92	\	108	l	124	|
13	*CR*	29	*GS*	45	-	61	=	77	M	93	]	109	m	125	}
14	*SO*	30	*RS*	46	.	62	>	78	N	94	^	110	n	126	~
15	*SI*	31	*US*	47	/	63	?	79	O	95	_	111	o	127	DEL

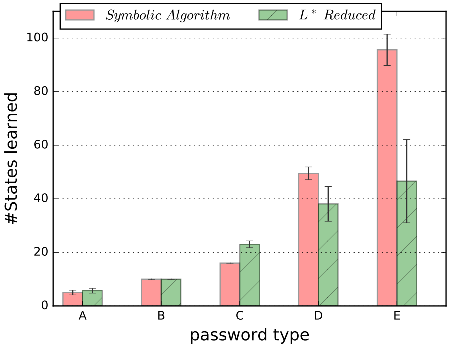
Control Characters Numerals Lower-Case Letters

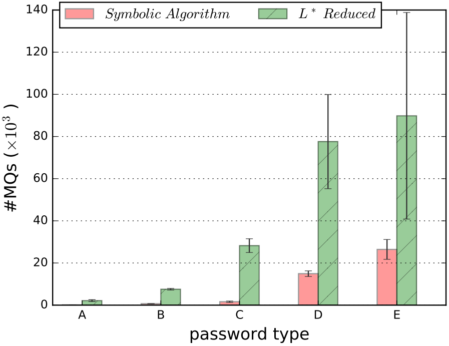
Punctuation Symbols Upper-Case Letters

Empirical Results

Valid passwords over the ASCII characters

The Symbolic Algorithm, $L^* - Reduced$: [RS93]





A (pin)

Length: 4 to 8.
Contains only numbers.

B (easy)

Length: 4 to 8.
It contains any printable character.

C (medium)

Length: 6 to 14.
Contains any printable character but punctuation characters.

D (medium-strong)

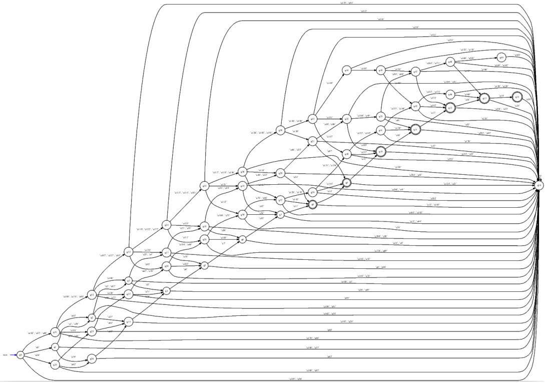
Length: 6 to 14.
Contains at least 1 number and 1 lower-case letter. Punctuation characters are allowed.

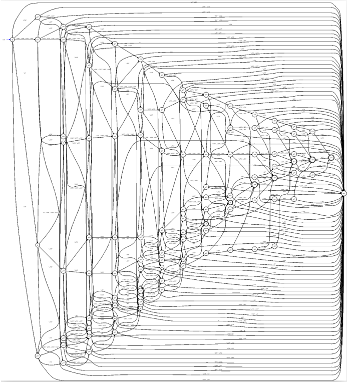
E (strong)

Length: 6 to 14.
Contains at least 1 character from each group.

Empirical Results

Valid passwords over the ASCII characters





A (pin)

Length: 4 to 8.
Contains only numbers.

B (easy)

Length: 4 to 8.
It contains any printable character.

C (medium)

Length: 6 to 14.
Contains any printable character but punctuation characters.

D (medium-strong)

Length: 6 to 14.
Contains at least 1 number and 1 lower-case letter.
Punctuation characters are allowed.

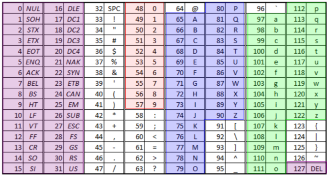
E (strong)

Length: 6 to 14.
Contains at least 1 character from each group.

Empirical Results

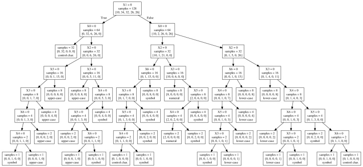
Valid passwords over the ASCII characters

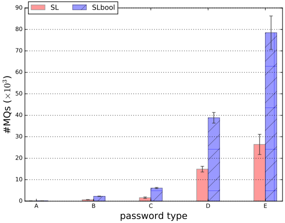
$$\Sigma = \{0, 1, \ldots, 127\}$$

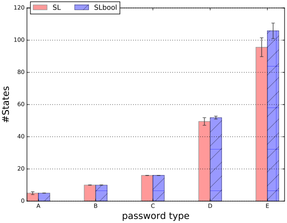


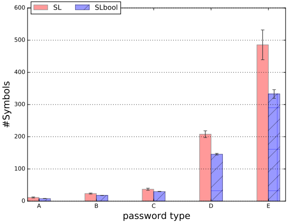
0	NUL	16	DLE	32	SPC	48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

$$\Sigma = \mathbb{B}^7$$


X1 = 0 samples = 128 [10, 34, 32, 26, 26]
True
False
X0 = 0 samples = 64 [0, 32, 6, 26, 0]
X0 = 0 samples = 64 [10, 2, 26, 0, 26]
samples = 32 [0, 32, 0, 0, 0] control char.
X2 = 0 samples = 32 [0, 0, 6, 26, 0]
X2 = 0 samples = 32 [10, 1, 21, 0, 0]
X2 = 0 samples = 32 [0, 1, 5, 0, 26]
X5 = 0 samples = 16 [0, 0, 1, 15, 0]
X3 = 0 samples = 16 [0, 0, 5, 11, 0]
X6 = 0 samples = 16 [0, 1, 15, 0, 0]
X3 = 0 samples = 16 [10, 0, 6, 0, 0]
X6 = 0 samples = 16 [0, 0, 1, 0, 15]
X3 = 0 samples = 16 [0, 1, 4, 0, 11]
X3 = 0 samples = 8 [0, 0, 1, 7, 0]
samples = 8 [0, 0, 0, 8, 0] upper-case
samples = 8 [0, 0, 0, 8, 0] upper-case
X4 = 0 samples = 8 [0, 0, 5, 3, 0]
X3 = 0 samples = 8 [0, 1, 7, 0, 0]
samples = 8 [0, 0, 8, 0, 0] symbol
samples = 8 [8, 0, 0, 0, 0] numeral
X5 = 0 samples = 8 [2, 0, 6, 0, 0]
X4 = 0 samples = 8 [0, 0, 1, 0, 7]
samples = 8 [0, 0, 0, 0, 8] lower-case
samples = 8 [0, 0, 0, 0, 8] lower-case
X4 = 0 samples = 8 [0, 1, 4, 0, 3]
X6 = 0 samples = 4 [0, 0, 1, 3, 0]
samples = 4 [0, 0, 0, 4, 0] upper-case
X5 = 0 samples = 4 [0, 0, 1, 3, 0]
samples = 4 [0, 0, 4, 0, 0] symbol
X5 = 0 samples = 4 [0, 1, 3, 0, 0]
samples = 4 [0, 0, 4, 0, 0] symbol
X4 = 0 samples = 4 [2, 0, 2, 0, 0]
samples = 4 [0, 0, 4, 0, 0] symbol
X5 = 0 samples = 4 [0, 0, 1, 0, 3]
samples = 4 [0, 0, 0, 0, 4] lower-case
X6 = 0 samples = 4 [0, 0, 1, 0, 3]
X5 = 0 samples = 4 [0, 1, 3, 0, 0]
X4 = 0 samples = 2 [0, 0, 1, 1, 0]
samples = 2 [0, 0, 0, 2, 0] upper-case
samples = 2 [0, 0, 0, 2, 0] upper-case
X6 = 0 samples = 2 [0, 0, 1, 1, 0]
X4 = 0 samples = 2 [0, 1, 1, 0, 0]
samples = 2 [0, 0, 2, 0, 0] symbol
samples = 2 [2, 0, 0, 0, 0] numeral
samples = 2 [0, 0, 2, 0, 0] symbol
X3 = 0 samples = 2 [0, 0, 1, 0, 1]
samples = 2 [0, 0, 0, 0, 2] lower-case
samples = 2 [0, 0, 0, 0, 2] lower-case
X5 = 0 samples = 2 [0, 0, 1, 0, 1]
samples = 2 [0, 0, 2, 0, 0] symbol
X6 = 0 samples = 2 [0, 1, 1, 0, 0]
samples = 1 [0, 0, 1, 0, 0] symbol
samples = 1 [0, 0, 0, 1, 0] upper-case
samples = 1 [0, 0, 0, 1, 0] upper-case
samples = 1 [0, 0, 1, 0, 0] symbol
samples = 1 [0, 1, 0, 0, 0] control char.
samples = 1 [0, 0, 1, 0, 0] symbol
samples = 1 [0, 0, 1, 0, 0] symbol
samples = 1 [0, 0, 0, 0, 1] lower-case
samples = 1 [0, 0, 0, 0, 1] lower-case
samples = 1 [0, 0, 1, 0, 0] symbol
samples = 1 [0, 0, 1, 0, 0] symbol
samples = 1 [0, 1, 0, 0, 0] control char.







Outline

Preliminaries

- Regular Languages and Automata
- The L^* Algorithmic Scheme

Large Alphabets

- Motivation
- Symbolic Representation of Transitions - Symbolic Automata

Learning Symbolic Automata

- Why L^* cannot be applied?
- Our Solution
- The Algorithm

Equivalence Queries and Counter-Examples

Adaptation to the Boolean Alphabet

Experimental Results

Conclusion

Related Work

Ideas similar to ours have been suggested and explored in a series of papers, which also adapt automaton learning and the L^* algorithm to large alphabets.

F Howar, B Steffen, and M Merten (2011).
Automata learning with automated alphabet abstraction refinement.

M Isberner, F Howar, and B Steffen (2013).
Inferring automata with state-local alphabet abstractions.

- The hypothesis is a partially defined hypothesis where the transition function is not defined outside the observed evidence.

T Berg, B Jonsson, and H Raffelt (2006).
Regular inference for state machines with parameters.

- Based on alphabet refinement that generates new symbols indefinitely.

Related Work

Ideas similar to ours have been suggested and explored in a series of papers, which also adapt automaton learning and the L^* algorithm to large alphabets.

S Drews and L D'Antoni (2017). *Learning symbolic automata.*

- Gives a more general justification for a learning scheme like ours by providing that learnability is closed under product and disjoint union.

M Botinčan and D Babić (2013). *Sigma*: Symbolic learning of input-output specifications.*

- Weaker termination results that is related to the counter-example guided abstraction refinement procedure. Handles transducers instead of automata.

Contribution

O Maler and IE Mens. Learning regular languages over large alphabets. *In TACAS*, vol 8413 of LNCS, pages 485–499. Springer, 2014.

O Maler and IE Mens. Learning regular languages over large ordered alphabets. *Logical Methods in Computer Science (LMCS)*, 11(3), 2015.

O Maler and IE Mens. A Generic Algorithm for Learning Symbolic Automata from Membership Queries. *In Models, Algorithms, Logics and Tools*, vol 10460 of LNCS, pages 146-169. Springer, 2017.

Conclusions

- We presented an algorithm for learning regular languages over large alphabets using symbolic automata.
- We decomposed the problem into learning new states (as in standard automaton learning) and learning the alphabet partitions in each state.
- Modification of alphabet partitions are treated in a rigorous way that does not introduce superfluous symbols.
- It can be done as static learning of concepts/partitions in the alphabet domain.
- We defined the notion of evidence compatibility which is an invariance of the algorithm and extended the breakpoint method to detect its violation.
- We explored in detail and implemented the cases where alphabets are numbers or Boolean vectors.
- We handle both helpful and non-helpful teachers.

Future Work

- Extend the algorithm to alphabets such as $\mathbb{R}^n$ and $\mathbb{R}^n \times \mathbb{B}^n$ using regression trees.
- Explore the use of other “deep learning” methods to learn the alphabet partitions.
- Study more realistic situations where the learner does not have full control over the sample and when some noise is present.
- Make more experiments and algorithmic improvement for the Boolean case.
- Find and explore a convincing class of applications.

Thank you !