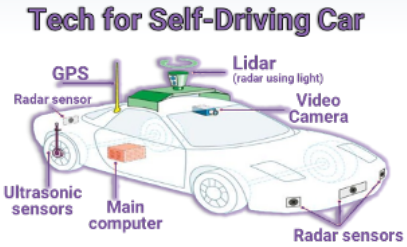
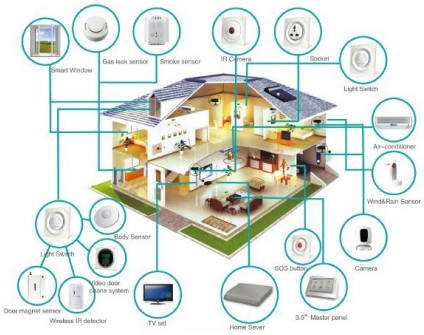
Irini-Eleftheria Mens
VERIMAG, *University of Grenoble-Alpes*

Learning Regular Languages using queries

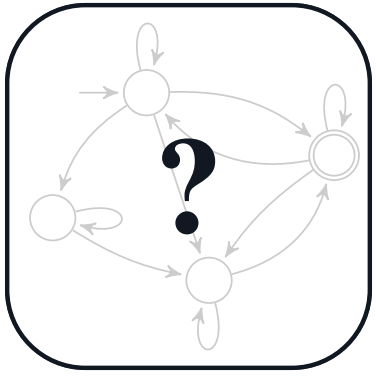
Thessaloniki, Greece
February 22, 2018





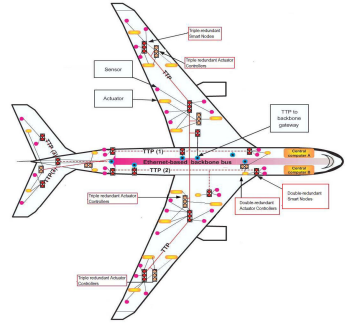
Black Box
Learning

Model



Language
Identification

System
Identification

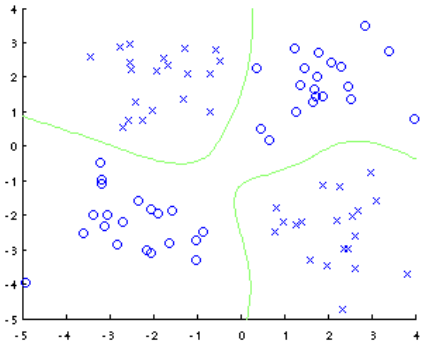


Inductive Inference

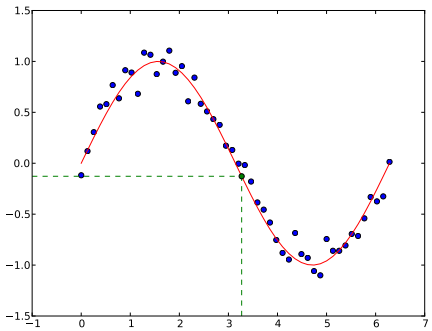
Outline

Outline

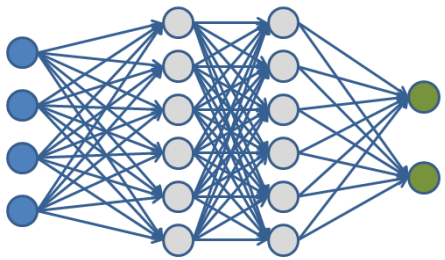
Machine Learning



Classification



Regression



Neural Network

. . .

- What do we know about the underlying model?
- What do we need the model for?
- How do we retrieve information?

Machine Learning

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

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

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

Model

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

The model is an 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.

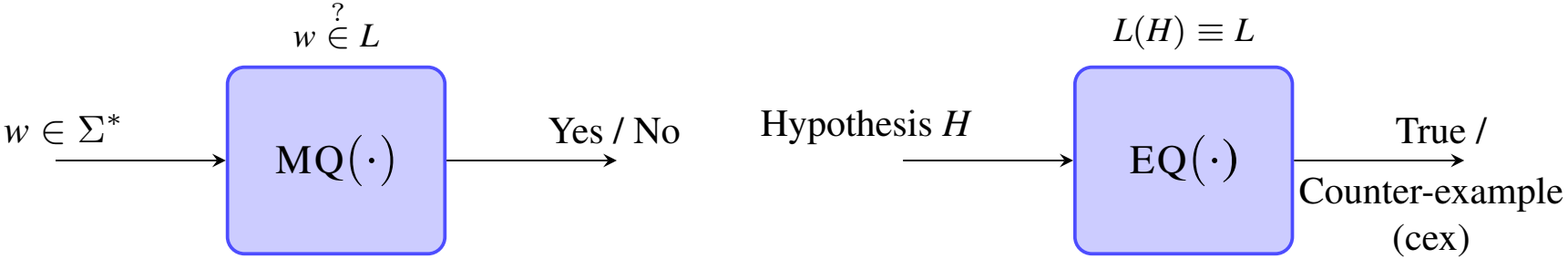
Passive vs Active

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.



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.

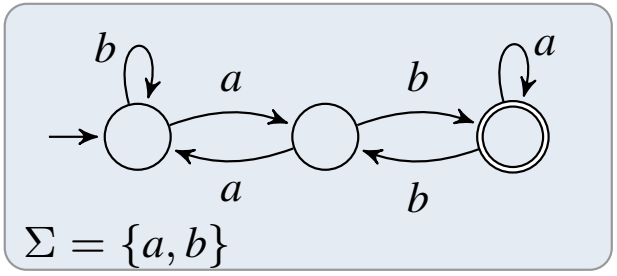
1978 E. Mark Gold. *Complexity of automaton identification from given data.*
Finding the minimal automaton compatible with a given sample is NP-hard.

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.

Outline

Regular Languages and Automata



$L \subseteq \Sigma^*$ 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.

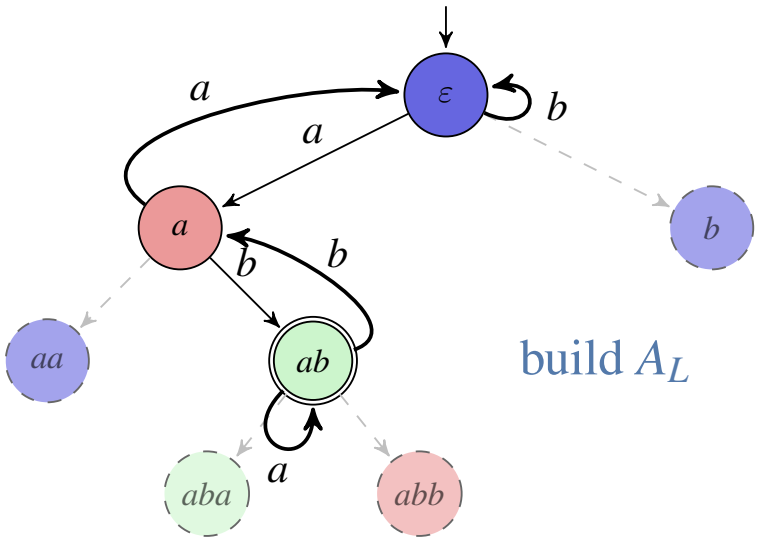
	suffixes								
prefixes	ε	a	b	aa	ab	ba	bb	aaa	$\dots$
ε	−	−	−	−	+	−	−	−	…
a	−	−	+	−	−	+	−	−	…
b	−	−	−	−	+	−	−	−	…
aa	−	−	−	−	+	−	−	−	…
ab	+	+	−	+	−	−	+	+	…
ba	−	−	+	−	−	+	−	−	…
bb	−	−	−	−	+	−	−	−	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱
aba	+	+	−	+	−	−	+	+	…
abb	−	−	+	−	−	+	−	−	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

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

Regular Languages and Automata

A sufficient sample that characterizes the language

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



build A_L

- 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 Observation Table T

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

From a *closed* and *consistent* table T, one can construct a dfa that is compatible with it.

$$A_T = (\Sigma, S, \varepsilon, F, \delta)$$

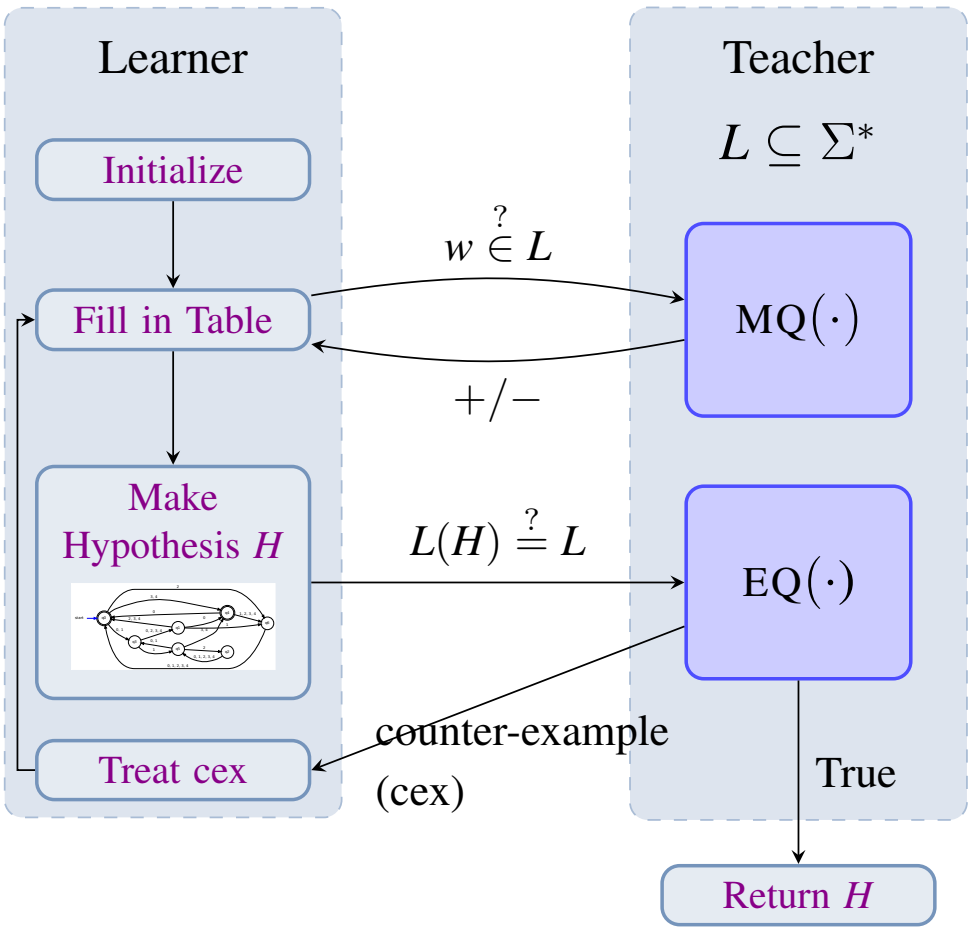
Every reduced table is consistent.

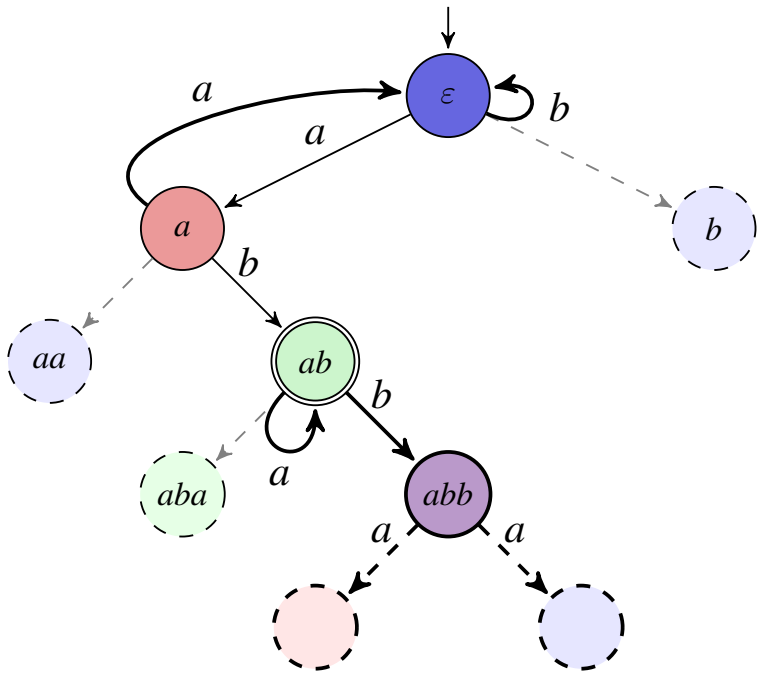
- T *closed* if-f $\forall r \in R, \exists s \in S, f_r = f_s$
- T *consistent* if-f $\forall s, s' \in S, \forall a \in \Sigma, f_s = f_{s'} \Rightarrow f_{s \cdot a} = f_{s' \cdot a}$
- T *reduced* when $\forall s, s' \in S, f_s \neq f_{s'}$

Outline

The L^* Algorithmic Scheme*

Active learning using queries

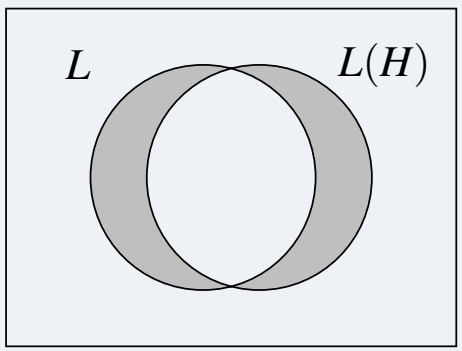




*D. Angluin. *Learning regular sets from queries and counter-examples*, 1987.

Counter-Examples

What is the error?



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

Let $w = a_1 \cdot a_2 \cdots a_n$ be a counter-example

Angluin's Counter-Example Treatment [Angluin'87]

- Add all prefixes $a_1 \cdots a_i$ to the set of prefixes S

Malers's Counter-Example Treatment [Maler'95]

- Add all suffixes $a_i \cdots a_n$ to the set of suffixes E

Breakpoint Method [Rivest Shapire'93]

- Find suitable suffix v_i to add to the set of suffixes E

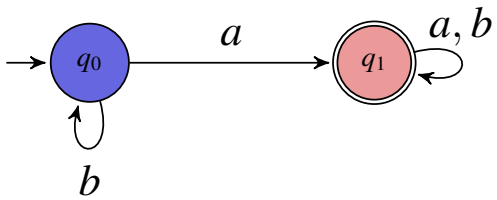
Example of L^*

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

observation table

	ε
ε	$-$
a	$+$
b	$-$
aa	$+$
ab	$+$

hypothesis automaton



counterexample: $-ba$

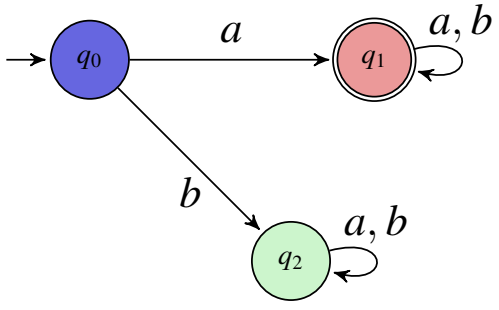
Example of L^*

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

observation table

	ε	a
ε	$-$	$+$
a	$+$	$+$
b	$-$	$-$
ba	$-$	$-$
aa	$+$	$+$
ab	$+$	$+$
bb	$-$	$-$
baa	$-$	$-$
bab	$-$	$-$

hypothesis automaton

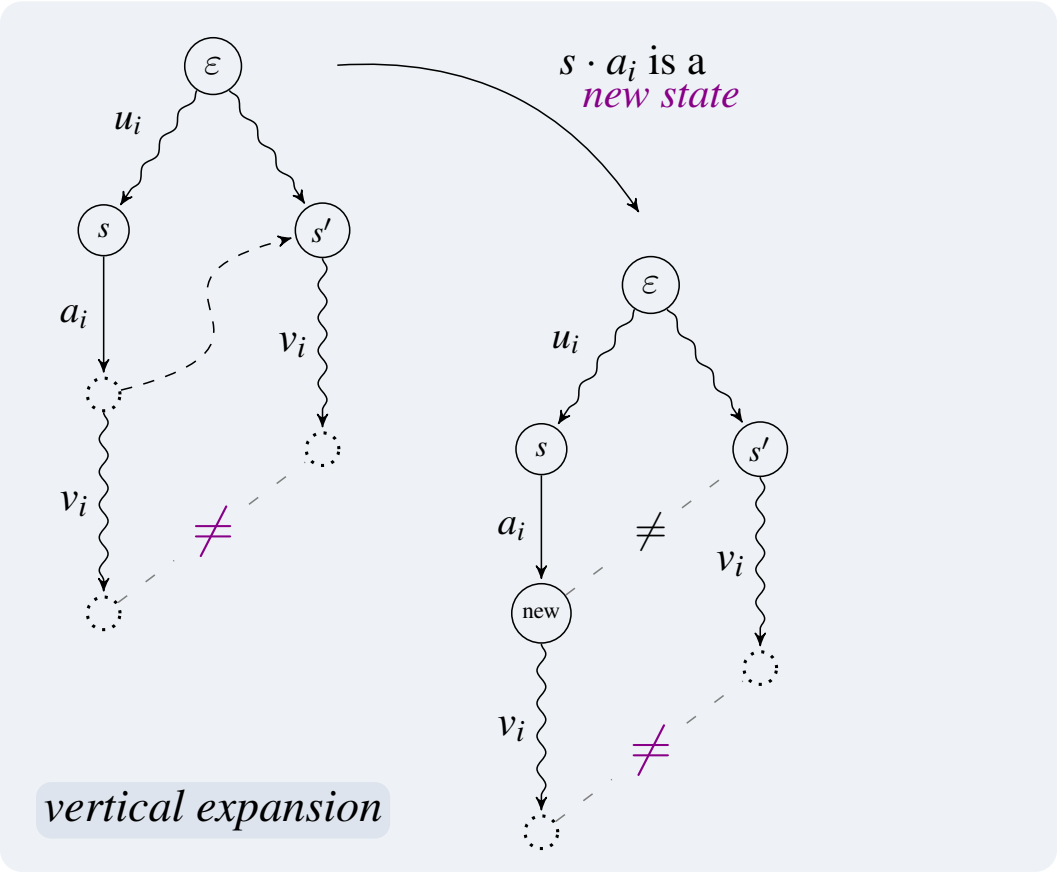


True

Counter-example Treatment (Breakpoint Method)

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

$f(s_{i-1} \cdot a_i \cdot v_i) \neq f(s_i \cdot v_i)$ $\qquad$ $s_i = \delta(\varepsilon, u_i \cdot a_i)$



Counter-example Treatment (Breakpoint Method)

Proposition

If w is a counter-example to $\mathcal{A}_T$ then there exists an i-factorization of w, i.e., $w = u_i \cdot a_i \cdot v_i$, such that either

$$f(s_{i-1} \cdot a_i \cdot v_i) \neq f(s_i \cdot v_i) \tag{1}$$

- If (1), then v_i is a new distinguishing word *vertical expansion*
  - Table not closed $\rightarrow$ new state

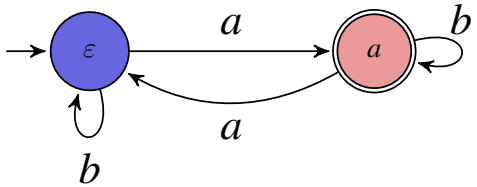
Example of L^* breakpoint method

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

observation table

	ε
ε	$-$
a	$+$
b	$-$
aa	$-$
ab	$+$

hypothesis automaton



Ask Equivalence Query:
counterexample: $-ba$

$a \not\sim ba \longrightarrow a$ is a new distinguishing string

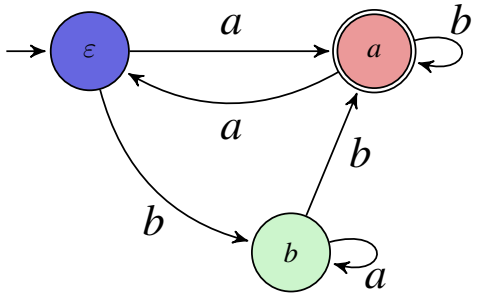
Example of L^* breakpoint method

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

observation table

	ε	a
ε	$-$	$+$
a	$+$	$-$
b	$-$	$-$
aa	$-$	$+$
ab	$+$	$-$
ba	$-$	$-$
bb	$+$	$-$

hypothesis automaton



Ask Equivalence Query:

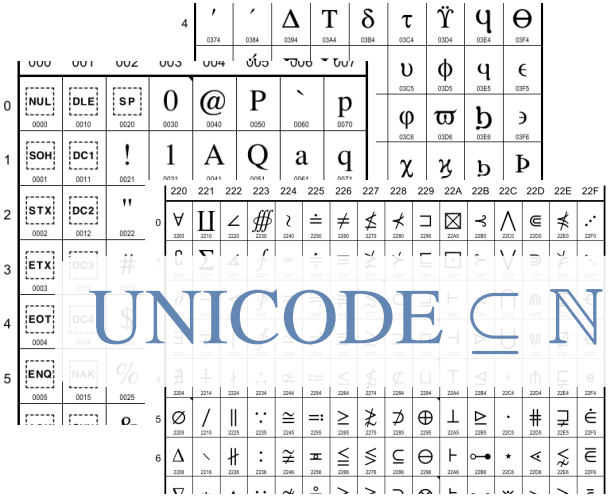
True

Other Automata Learning

- Mealy Machines (R Groz 2009)
- Register Automata (M Isberner, F Howar, B Steffen 2012)
- Timed Automata (Verver 2010, Grinchtein 2008)
- ω-languages (Maler 1995, D Angluin, D Fisman 2016)
- Non-deterministic Automata (P García - 2008)
- Probabilistic Automata (ALERGIA Algorithm for passive learning)
- Grammars (C.de la Higuera. Grammatical inference - Book)
- Large Alphabets (Mens, Maler, Steffen, Isberner)

Outline

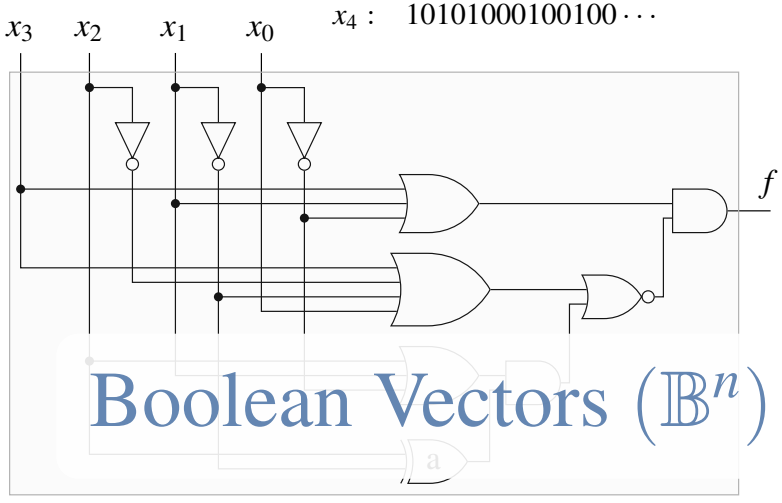
Languages over Large Alphabets



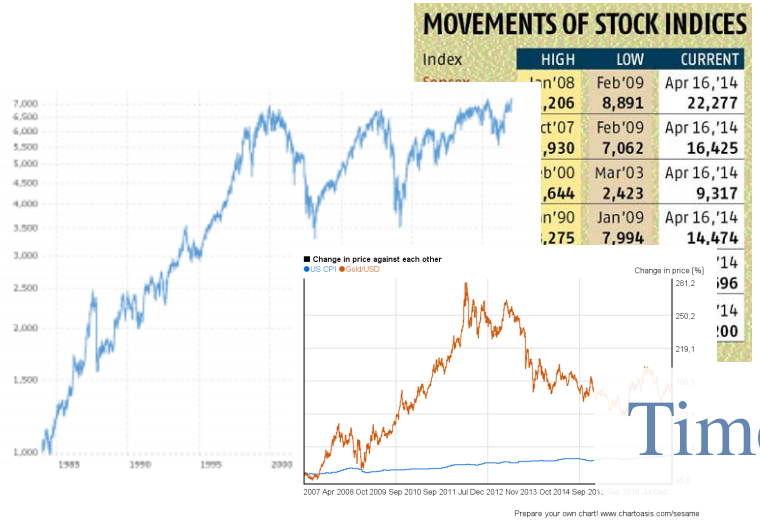
UNICODE $\subseteq \mathbb{N}$

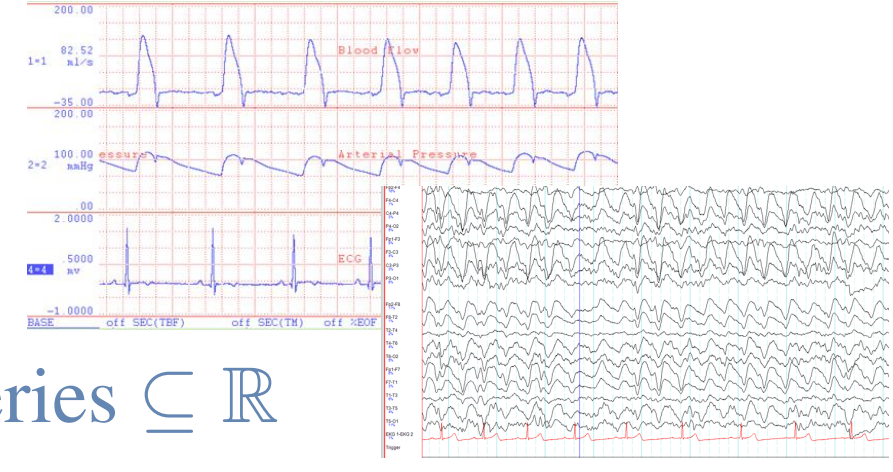
Input:

x_1 : 10101010000100 $\cdots$
x_2 : 10100100100100 $\cdots$
x_3 : 10101000010001 $\cdots$
x_4 : 10101000100100 $\cdots$



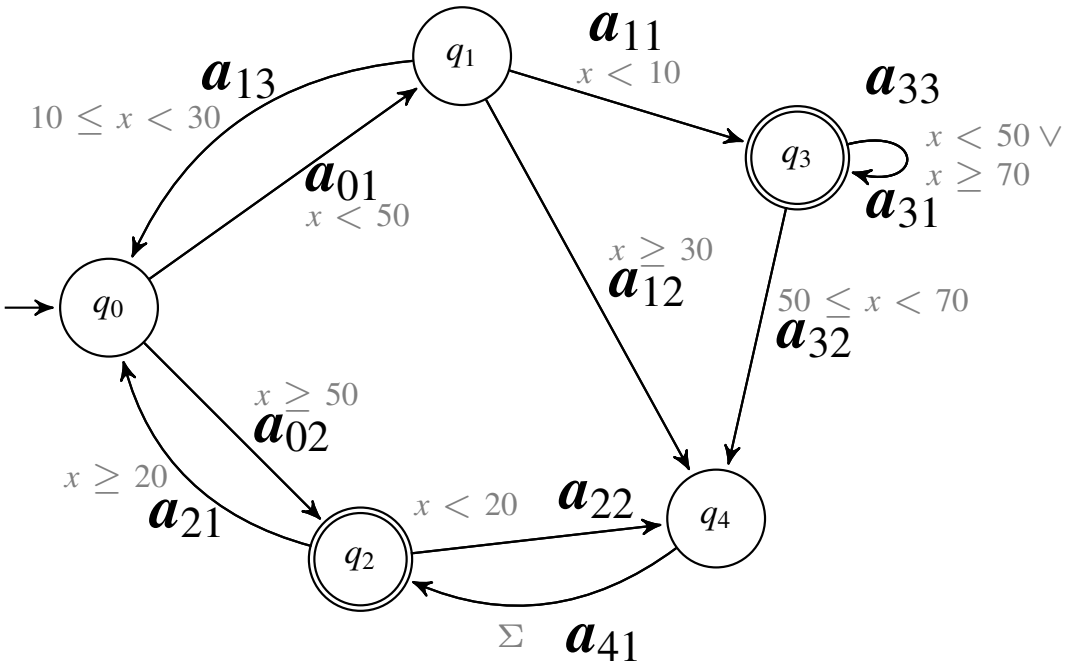
Boolean Vectors $(\mathbb{B}^n)$





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)
- $\psi_q : \Sigma \to \boldsymbol{\Sigma_q}, q \in Q$
- $[\![\boldsymbol{a}]\!] = \{a \in \Sigma \mid \psi(a) = \boldsymbol{a}\}$

$\Sigma \subseteq \mathbb{R}$

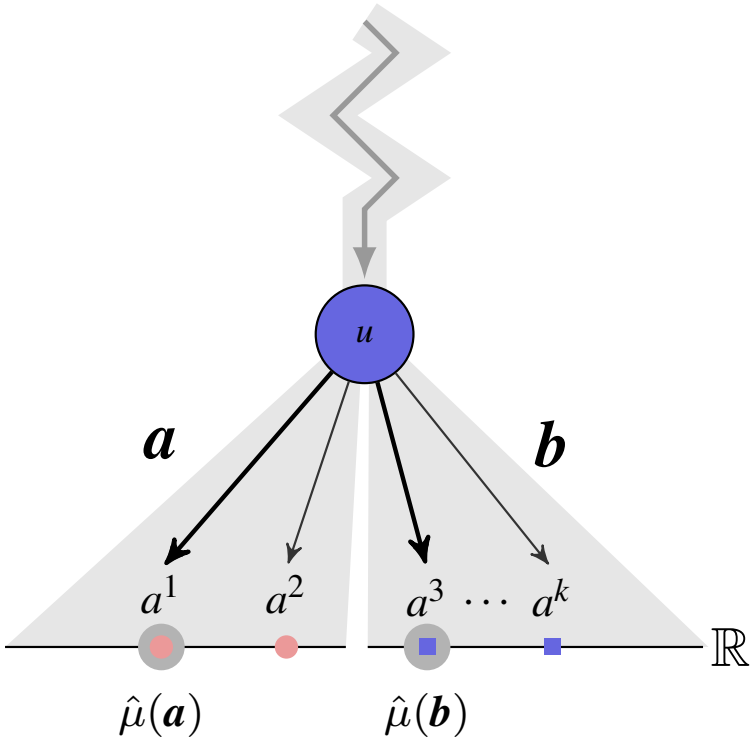
$[\![\boldsymbol{a}_{01}]\!] = \{x \in \Sigma : x < 50\}$

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

$\boldsymbol{w = a_{01} \cdot a_{12} \cdot a_{41}}$

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

Learning over Large Alphabets



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

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

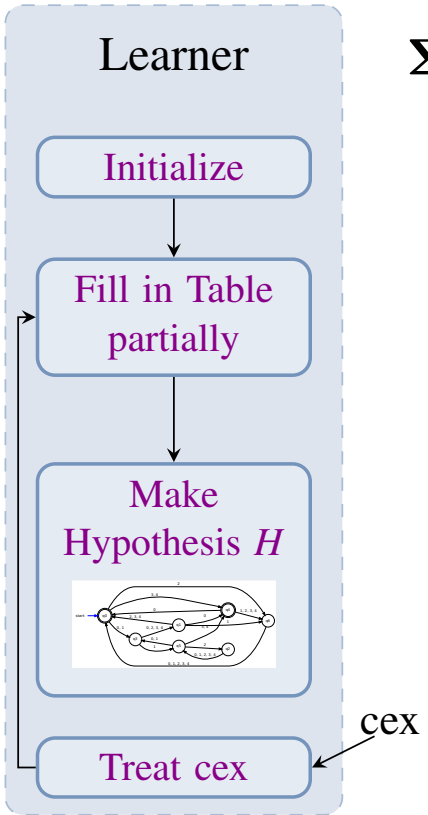
Why L^* cannot be applied?

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

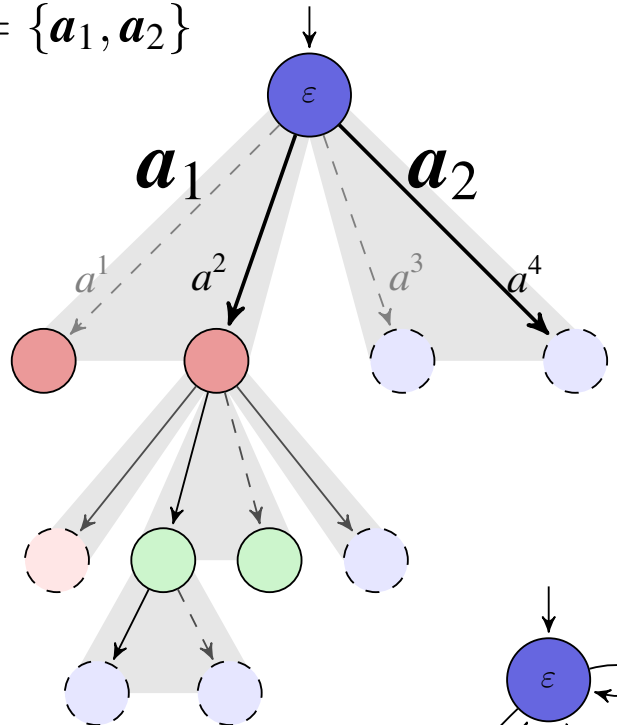
Our solution:

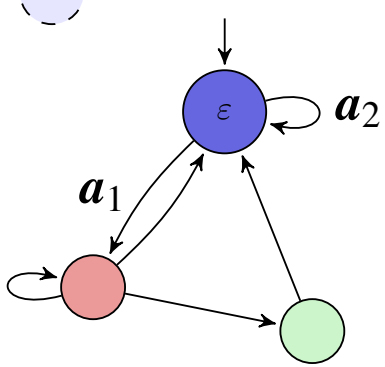
- Use a finite sample of evidences to learn the transitions
- Form evidence compatible partitions
- Associate a symbol to each partition block
- Each symbol has one representative evidence
- The prefixes are symbolic

Symbolic Learning Algorithm



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

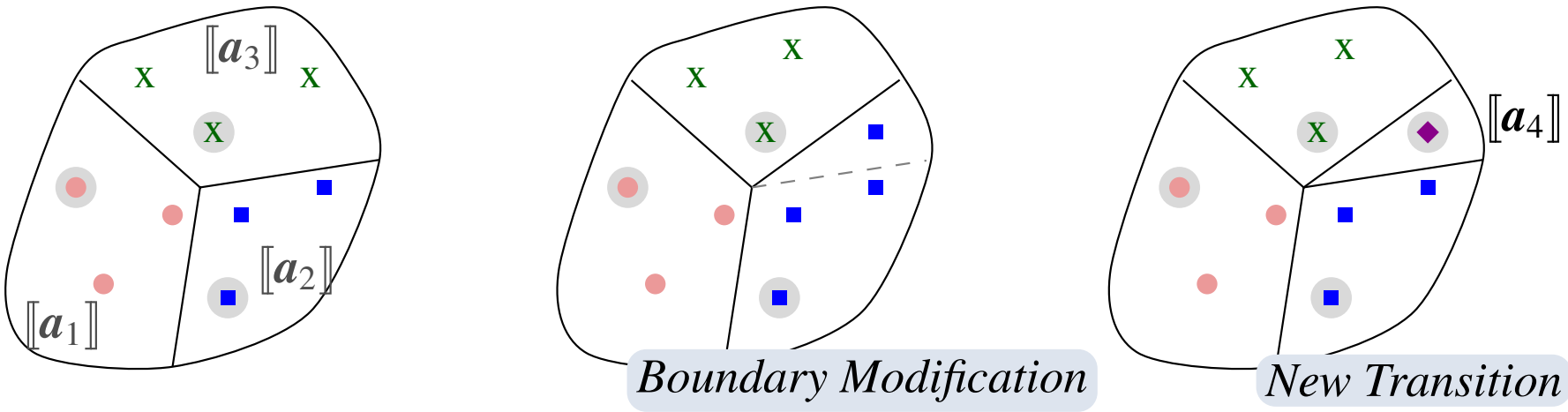




Repeat for each new state q:

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

Evidence Compatibility



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 $\boldsymbol{u}$

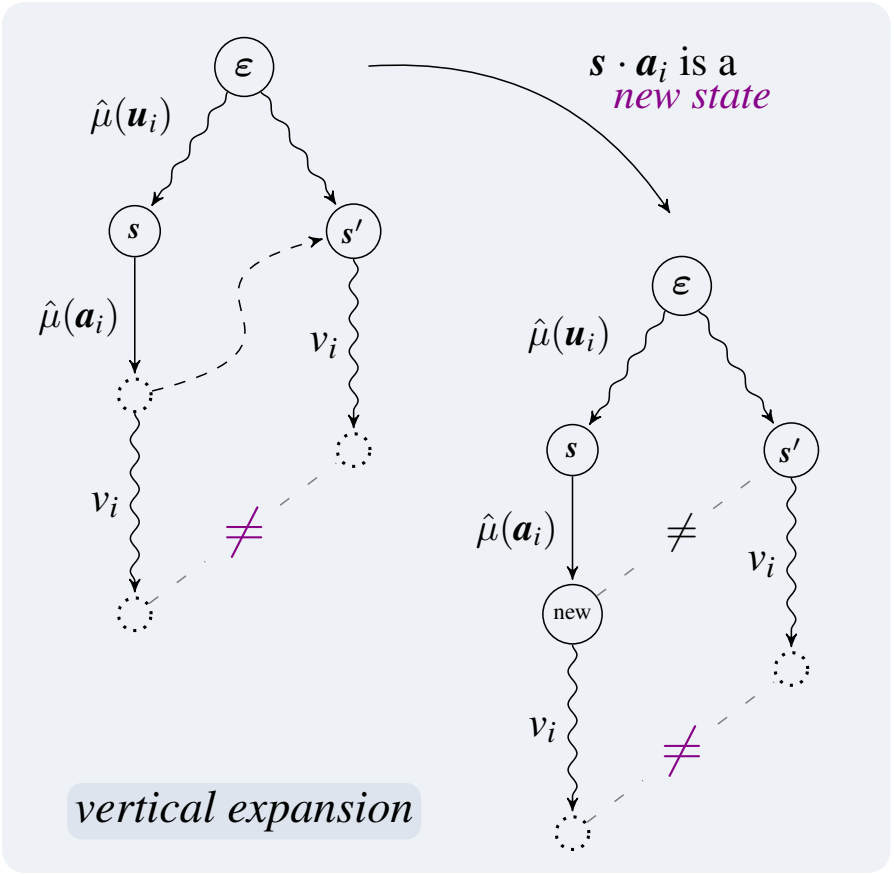
	v		
		$\vdots$	
$\boldsymbol{u}\cdot\hat{\mu}(\boldsymbol{a})$	$\cdots$	$+$	$\cdots$
$\boldsymbol{u}\cdot a$	$\cdots$	$-$	$\cdots$

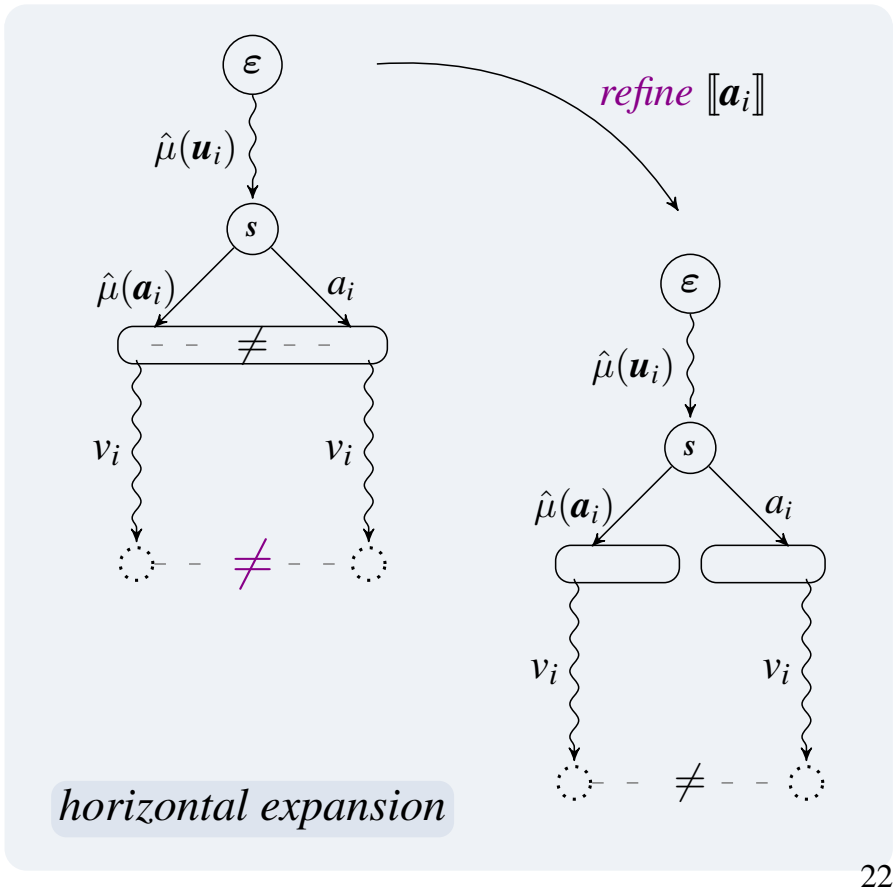
Counter-example Treatment (Symbolic Breakpoint)

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

$$f(\hat{\mu}(s_{i-1} \cdot a_i) \cdot v_i) \neq f(\hat{\mu}(s_i) \cdot v_i) \qquad f(\hat{\mu}(s_{i-1}) \cdot a_i \cdot v_i) \neq f(\hat{\mu}(s_{i-1}) \cdot \hat{\mu}(a_i) \cdot v_i)$$

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



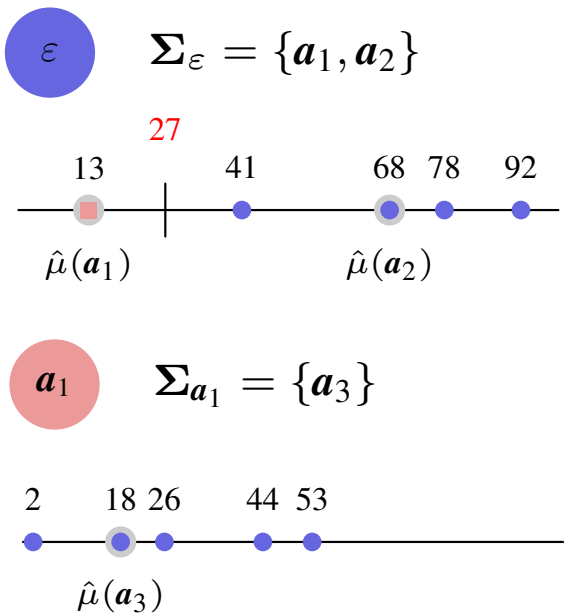


Example over the alphabet $\Sigma = [1, 100)$

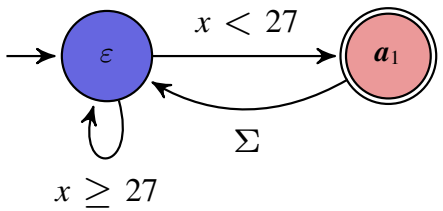
observation table

	ε	11
ε	−	
13 a_1	+	
68 a_2	−	
13 18 a_1a_3	−	

semantics



hypothesis automaton



Ask Equivalence Query:

counter-example:

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

add distinguishing string 11

discover new state

(vertical expansion)

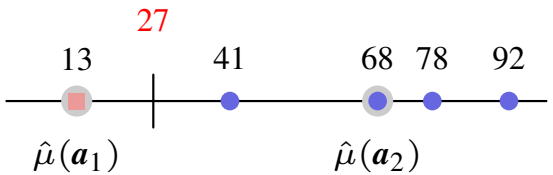
Example over the alphabet $\Sigma = [1, 100)$

observation table

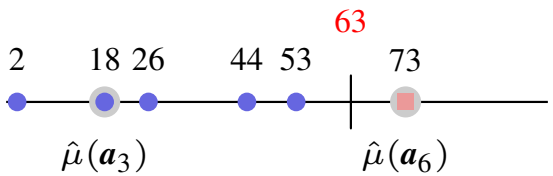
	ε	11
ε	−	+
13 a_1	+	−
68 a_2	−	−
13 18 a_1a_3	−	+
13 73 a_1a_6	+	−
68 17 a_2a_4	−	−
68 75 a_2a_5	+	−

semantics

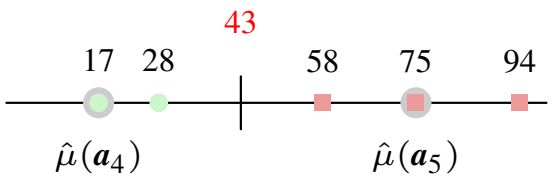
ε $\quad \Sigma_\varepsilon = \{a_1, a_2\}$



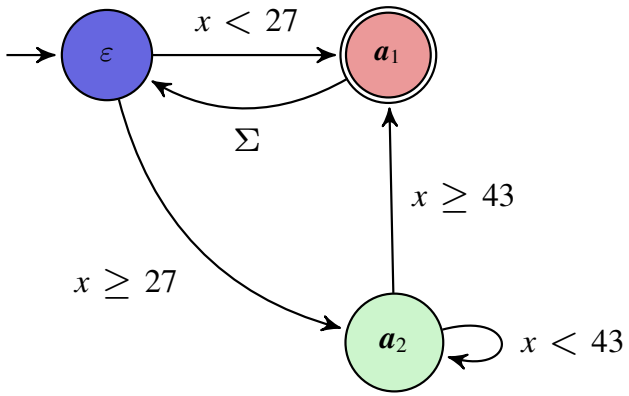
a_1 $\quad \Sigma_{a_1} = \{a_3, a_6\}$



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



hypothesis automaton



Ask Equivalence Query:

counter-example:

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

add 73 as evidence of a_1

add new transition

(horizontal expansion)

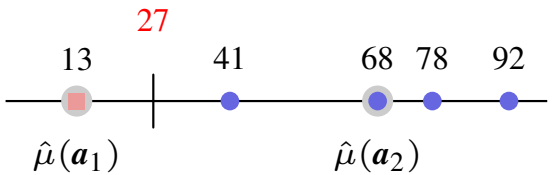
Example over the alphabet $\Sigma = [1, 100)$

observation table

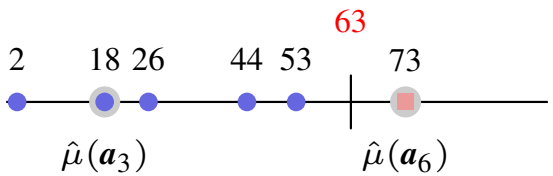
	ε	11
ε	−	+
13 a_1	+	−
68 a_2	−	−
13 18 a_1a_3	−	+
13 73 a_1a_6	+	−
68 17 a_2a_4	−	−
68 75 a_2a_5	+	−

semantics

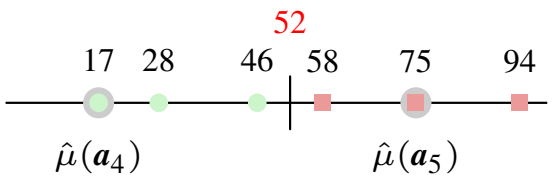
ε $\quad \Sigma_\varepsilon = \{a_1, a_2\}$



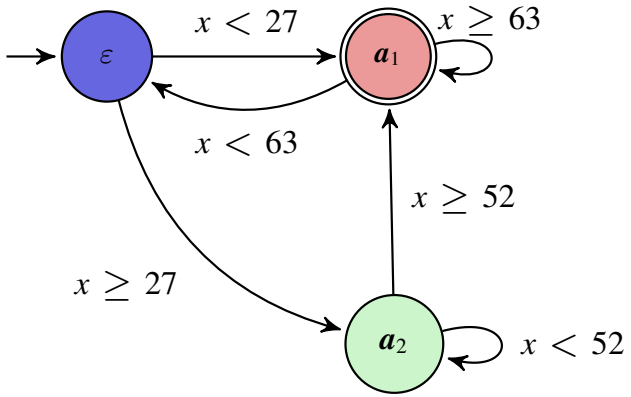
a_1 $\quad \Sigma_{a_1} = \{a_3, a_6\}$



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



hypothesis automaton



Ask Equivalence Query:

True

return current hypothesis

return hypothesis

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

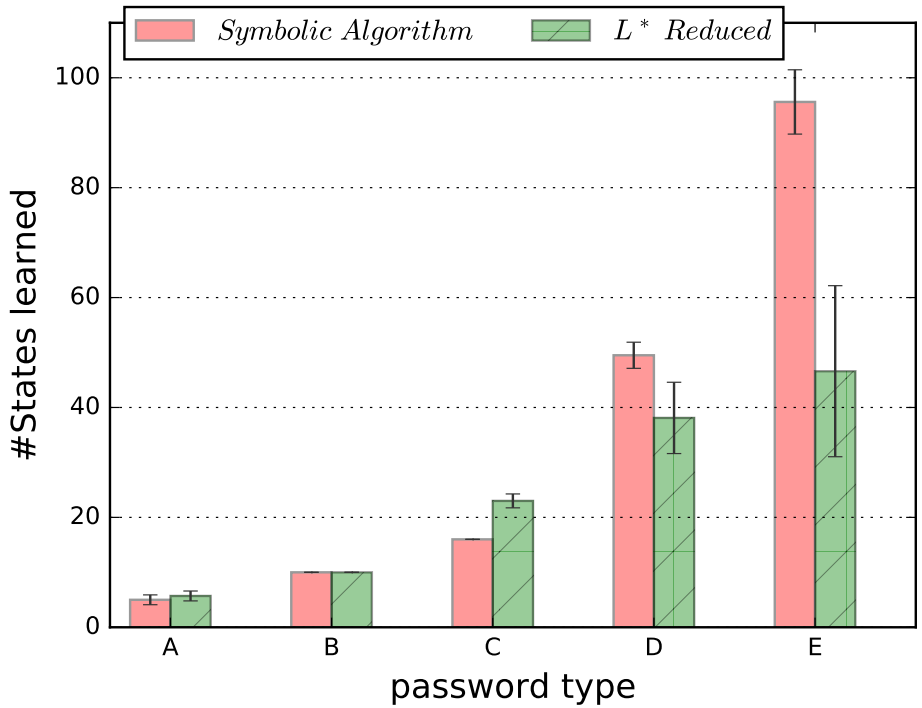
Control Characters Numerals Lower-Case Letters

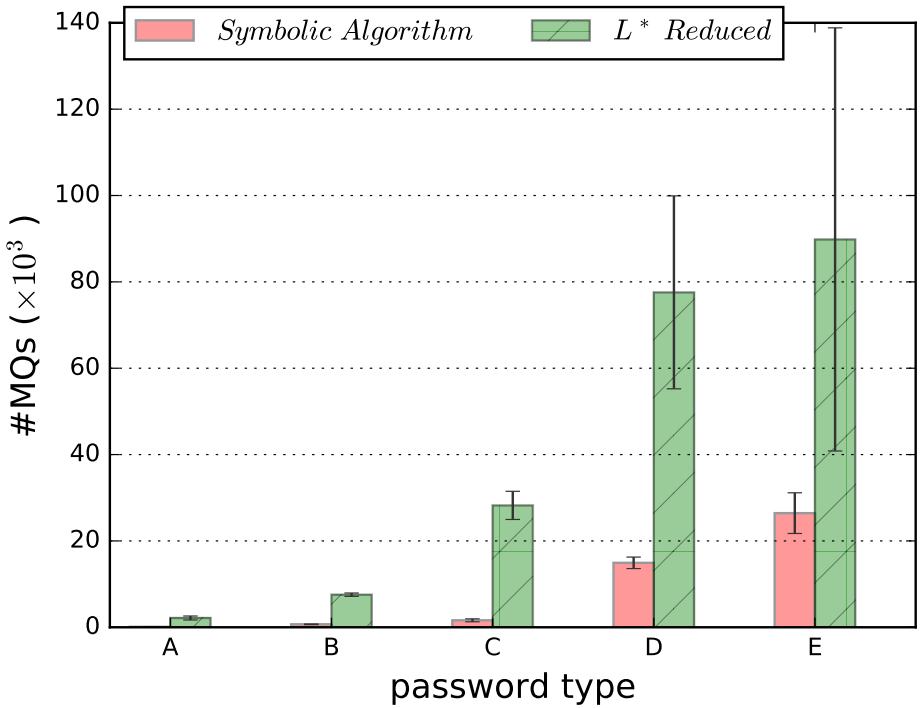
Punctuation Symbols Upper-Case Letters

Empirical Results

Valid passwords over the ASCII characters

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

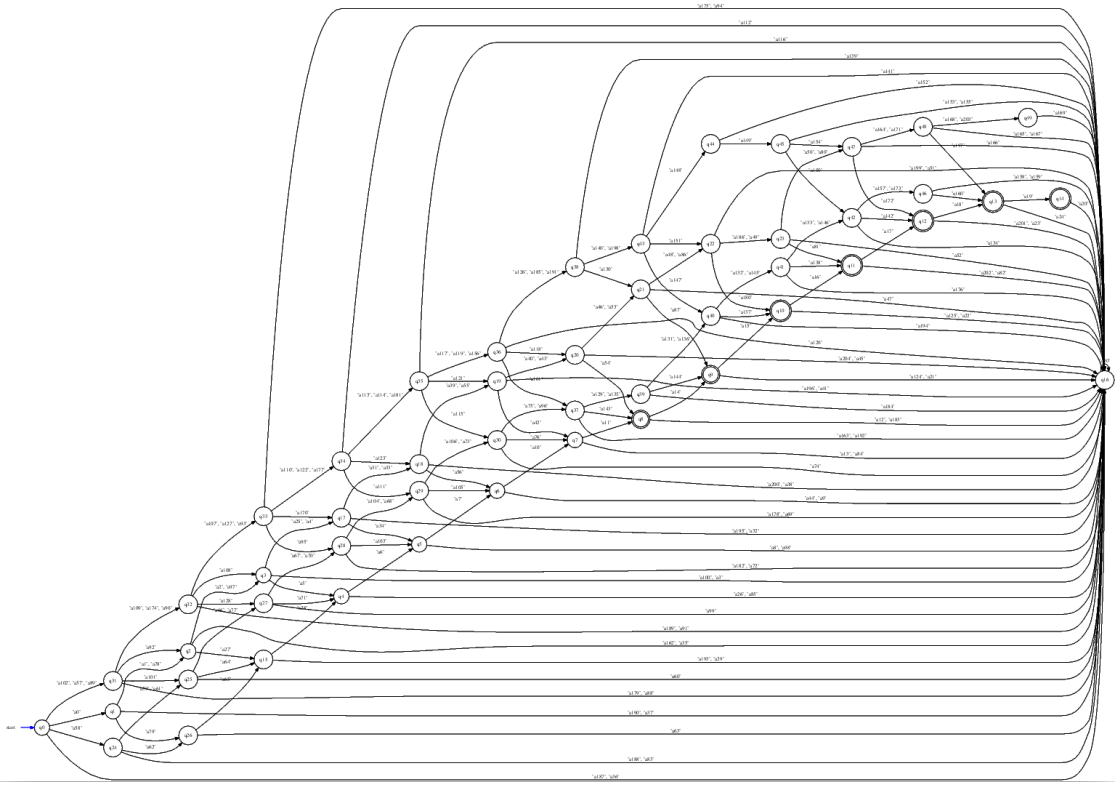


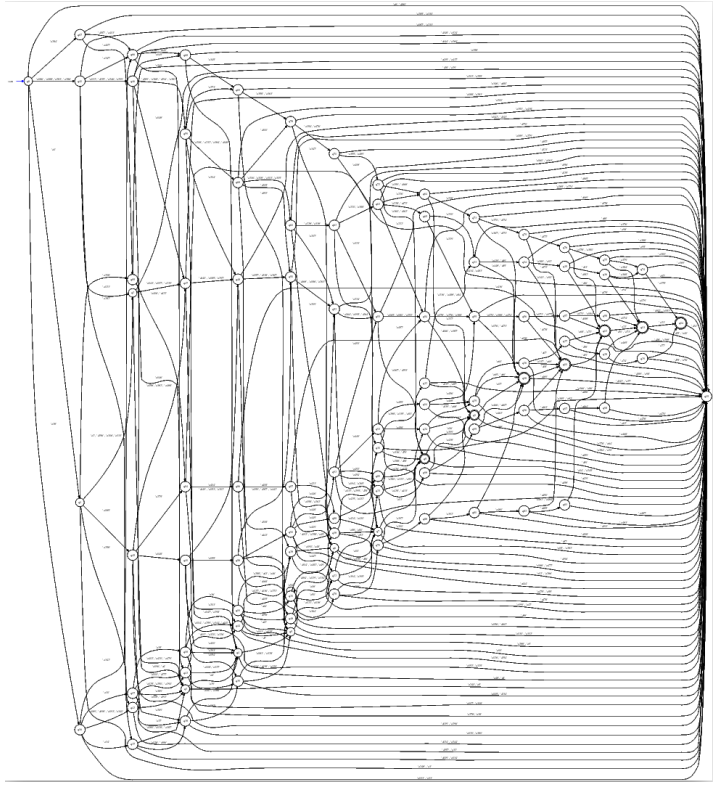


A (pin)	B (easy)	C (medium)	D (medium-strong)	E (strong)
Length: 4 to 8. Contains only numbers.	Length: 4 to 8. It contains any printable character.	Length: 6 to 14. Contains any printable character but punctuation characters.	Length: 6 to 14. Contains at least 1 number and 1 lower-case letter. Punctuation characters are allowed.	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.

Thank you!