Learning Regular Languages over Large Alphabets

10 October 2017

Jury Members

Oded Maler Directeur de thèse
Dana Angluin Rapporteur
Peter Habermehl Rapporteur
Laurent Fribourg Examinateur
Eric Gaussier Examinateur
Frits Vaandrager Examinateur
A Short Prehistory and History of Automaton Learning

      Defines the problem as a black box model inference.


      Learning finite automata is possible in finite time. He first uses
      the basic idea that underlies table-based methods.

      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 equiva-
      lence queries. Polynomial in the automaton size.

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

A small sample

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

### Learning Regular Languages

*over large or infinite alphabets*

- \( \Sigma \) an alphabet
- \( X = \Sigma^* \) set of words
- \( Y = \{ +, - \} \)
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.

\[ \text{MQ}(\cdot) \quad w \notin L \]
\[ w \in \Sigma^* \quad \rightarrow \quad \text{Yes / No} \quad \rightarrow \quad \text{Hypothesis } H \]
\[ \text{EQ}(\cdot) \quad L(H) \equiv L \]
\[ \rightarrow \quad \text{True / Counter-example (cex)} \]
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

- $\Sigma$ is an alphabet
- $w = a_1 \cdots a_n$ is a word
- $\Sigma^*$ is the set of all words
Regular Languages and Automata

\[ \Sigma = \{a, b\} \]

\[ L \subseteq \Sigma^* \text{ is a language} \]

**Equivalence relation**

\[ u \sim_L v \iff u \cdot w \in L \iff v \cdot w \in L \]

**Nerode’s Theorem**

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

\[ Q = \Sigma^*/\sim \text{ (states in the minimal representation of } L \text{).} \]

\[ \varepsilon \sim b \sim aa \quad a \sim ba \sim abb \quad ab \sim aba \]
Regular Languages and Automata

A sufficient sample that characterizes the language

<table>
<thead>
<tr>
<th></th>
<th>$\epsilon$</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>$-$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
<td>$+$</td>
<td>$-$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$-$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$aba$</td>
<td>$+$</td>
<td>$+$</td>
<td>$-$</td>
</tr>
<tr>
<td>$abb$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
</tbody>
</table>
Regular Languages and Automata

A sufficient sample that characterizes the language

\[
\begin{array}{c|ccc}
S & \varepsilon & a & b \\
\hline
\varepsilon & - & - & - \\
a & - & - & + \\
ab & + & + & - \\
\hline
b & - & - & - \\
aa & - & - & - \\
aba & + & + & - \\
abb & - & - & + \\
\end{array}
\]

- \(S\) prefixes (states)
- \(R\) boundary \((R = S \cdot \Sigma \setminus S)\)
- \(E\) suffixes (distinguishing strings)

\(f : S \cup R \times E \rightarrow \{+, -\}\) classification function
\(f_s : E \rightarrow \{+, -\}\) residual functions
A sufficient sample that characterizes the language

\[ \begin{array}{|c|ccc|} \hline \ & \varepsilon & a & b \\ \hline \varepsilon & - & - & - \\ a & - & - & + \\ ab & + & + & - \\ b & - & - & - \\ aa & - & - & - \\ aba & + & + & - \\ abb & - & - & + \\ \hline \end{array} \]

\[ S \text{ prefixes (states)} \]
\[ R \text{ boundary (} R = S \cdot \Sigma \setminus S \) \]
\[ E \text{ suffixes (distinguishing strings)} \]

\[ f : S \cup R \times E \to \{+, -\} \ 	ext{classif. function} \]
\[ f_s : E \to \{+, -\} \ 	ext{residual functions} \]

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

The minimal automaton for \( L \)
The \textit{L*} Algorithmic Scheme*

Active learning using queries

The $L^*$ Algorithmic Scheme*

Active learning using queries

---

The $L^*$ Algorithmic Scheme*

Active learning using queries

---

*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

Input:

\[
\begin{align*}
x_1 &: 10101010000100 \cdots \\
x_2 &: 10100100100100 \cdots \\
x_3 &: 1010100010001 \cdots \\
x_4 &: 10101000100100 \cdots
\end{align*}
\]

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

Time Series \(\subseteq \mathbb{R}\)
Symbolic Automata

\[ \mathcal{A} = (\Sigma, \Sigma, \psi, Q, \delta, q_0, F) \]

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

\( \Sigma \subseteq \mathbb{R} \)

\([a_{01}] = \{ x \in \Sigma : x < 50 \} \)

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

\( w = a_{01} \cdot a_{12} \cdot a_{41} \)

\( \mathcal{A} \) is complete and deterministic if \( \forall q \in Q \)

\( \{[a] \mid a \in \Sigma_q \} \) forms a partition of \( \Sigma. \)
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 (\( \forall a \in \Sigma, \text{ask MQ}(u \cdot a) \))
- 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 $MQ(u \cdot a)$)
- Inefficient for large finite alphabets
- Not applicable to infinite alphabets

Our solution:

- Use a finite sample of evidences to learn the transitions

Evidences: $\mu(a) = \{a^1, a^2\}$
Learning over Large Alphabets

Why \( L^* \) cannot be applied?

- The learner asks MQ’s for all continuations of a state (\( \forall a \in \Sigma, \text{ask MQ}(u \cdot a) \))
- 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

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

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

Evidences: $\mu(a) = \{a^1, a^2\}$
Representative: $\hat{\mu}(a) = a^1$
Symbolic Learning Algorithm

Learner
Symbolic Learning Algorithm

Learner

Initialize

\[ \varepsilon \]

\[ \varepsilon = \{a_1, a_2\} \]
Symbolic Learning Algorithm

Learner

Initialize

Fill in Table partially

Repeat for each new state $q$:

- Sample *evidences*
Symbolic Learning Algorithm

Learner

- Initialize
- Fill in Table partially

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

\[ \Sigma_\varepsilon = \{a_1, a_2\} \]

Repeat for each new state \( q \):
- Sample evidences
- Ask MQ’s
- Learn partitions
- Define the symbolic alphabet \( \Sigma_q \)

**Learner**
- Initialize
- Fill in Table partially
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 \( \Sigma_q \)
- Select representative \( \hat{\mu}(a), \forall a \in \Sigma_q \)
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* \( \Sigma_q \)
- Select representative \( \hat{\mu}(a), \forall a \in \Sigma_q \)
Symbolic Learning Algorithm

Learner

- Initialize
- Fill in Table partially
- Make Hypothesis $H$

Repeat for each new state $q$:
- Sample *evidences*
- Ask MQ’s
- Learn *partitions*
- Define the *symbolic alphabet* $\Sigma_q$
- Select representative $\hat{\mu}(a)$, $\forall a \in \Sigma_q$

$$\Sigma_\varepsilon = \{a_1, a_2\}$$
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 \( \Sigma_q \)
- Select representative \( \hat{\mu}(a), \forall a \in \Sigma_q \)

Learner

- Initialize
- Fill in Table partially
- Make Hypothesis \( H \)
- Treat cex
Evidence Compatibility

A state \( u \) is evidence compatible when

\[
\hat{f}_u \cdot a = \hat{f}_u \cdot \hat{\mu}(a)
\]

for every evidence \( a \in [a] \)

Evidence incompatibility at state \( u \)

<table>
<thead>
<tr>
<th>( u \cdot \hat{\mu}(a) )</th>
<th>( u \cdot a )</th>
<th>( v )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \cdots )</td>
<td>( + )</td>
<td>( : )</td>
</tr>
<tr>
<td>( \cdots )</td>
<td>( - )</td>
<td>( \cdots )</td>
</tr>
</tbody>
</table>
Counter-example Treatment (Symbolic Breakpoint)

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

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

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

- **Vertical expansion**
  - $s \cdot a_i$ is a new state

- **Horizontal expansion**
  - refine $[a_i]$
Example over the alphabet $\Sigma = [1, 100)$

**Observation Table**

<table>
<thead>
<tr>
<th></th>
<th>$\epsilon$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_1a_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>$-$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_1$</td>
<td>$+$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_2$</td>
<td>$-$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_1a_3$</td>
<td>$-$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Semantics**

- $\Sigma_{\epsilon} = \{a_1, a_2\}$
- $\Sigma_{a_1} = \{a_3\}$

**Hypothesis Automaton**

- $a_1$ on $x < 27$
- $a_3$ on $x ≥ 27$
Example over the alphabet $\Sigma = [1, 100)$

**observation table**

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>13</td>
<td>a₁</td>
</tr>
<tr>
<td>13</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>a₁</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td>68</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>a₂</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>13 18</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td>a₁a₃</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**semantics**

- $\Sigma_{\varepsilon} = \{a₁, a₂\}$
- $\Sigma_{a₁} = \{a₃\}$

**hypothesis automaton**

- $\varepsilon$
- $x < 27$
- $x \geq 27$
- $\Sigma$

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**

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td>$a_1$</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>$a_2$</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>$a_1a_3$</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td>$a_2a_4$</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>$a_2a_5$</td>
<td>+</td>
<td>−</td>
</tr>
</tbody>
</table>

**Semantics**

- $\Sigma_{\varepsilon} = \{a_1, a_2\}$
- $\Sigma_{a_1} = \{a_3\}$
- $\Sigma_{a_2} = \{a_4, a_5\}$

**Hypothesis Automaton**

- $a_1$ transitions
  - $\mu(a_1)$
  - $\mu(a_2)$
- $a_2$ transitions
  - $\mu(a_3)$
  - $\mu(a_4)$
  - $\mu(a_5)$

Counter-example: $15 / 31$
Example over the alphabet $\Sigma = [1, 100)$

Observation table:

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>$a_1$</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>$a_2$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$a_1a_3$</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>$a_1a_6$</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>$a_2a_4$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$a_2a_5$</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

Semantics:

- $\Sigma_{\varepsilon} = \{a_1, a_2\}$
- $\Sigma_{a_1} = \{a_3, a_6\}$
- $\Sigma_{a_2} = \{a_4, a_5\}$

Hypothesis automaton:

- $\varepsilon \rightarrow x < 27 \rightarrow a_1 \rightarrow x \geq 43 \rightarrow a_2 \rightarrow x < 43$
- $\Sigma \rightarrow 27 \rightarrow 41 \rightarrow 68 \rightarrow 78 \rightarrow 92$
- $\hat{\mu}(a_1) \rightarrow 13 \rightarrow \hat{\mu}(a_2)$
- $\hat{\mu}(a_3) \rightarrow 2 \rightarrow 18 \rightarrow 26 \rightarrow 44 \rightarrow 53 \rightarrow 63 \rightarrow \hat{\mu}(a_6)$
- $\hat{\mu}(a_4) \rightarrow 17 \rightarrow 28 \rightarrow 58 \rightarrow 75 \rightarrow 94 \rightarrow \hat{\mu}(a_5)$

Ask Equivalence Query:

- Counter-example: $w = 12 \cdot 73 \cdot 4, -$ add 73 as evidence of $a_1$
Example over the alphabet $\Sigma = [1, 100)$

### Observation Table

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$+$</td>
<td>$-$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$a_1a_3$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a_1a_6$</td>
<td>$+$</td>
<td>$-$</td>
</tr>
<tr>
<td>$a_2a_4$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$a_2a_5$</td>
<td>$+$</td>
<td>$-$</td>
</tr>
</tbody>
</table>

### Semantics

- $\Sigma_{\varepsilon} = \{a_1, a_2\}$
- $\Sigma_{a_1} = \{a_3, a_6\}$
- $\Sigma_{a_2} = \{a_4, a_5\}$

### Hypothesis Automaton

- $\varepsilon$ to $a_1$ with $x < 27$
- $a_1$ to $a_1$ with $x \geq 63$
- $a_2$ to $a_2$ with $x < 43$
- $a_1$ to $a_2$ with $x \geq 43$
Example over the alphabet $\Sigma = [1, 100)$

**Observation Table**

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>$a_1$</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>$a_2$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$a_1a_3$</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>$a_1a_6$</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>$a_2a_4$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$a_2a_5$</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

**Semantics**

- $\Sigma_{\varepsilon} = \{a_1, a_2\}$
- $\Sigma_{a_1} = \{a_3, a_6\}$
- $\Sigma_{a_2} = \{a_4, a_5\}$

**Hypothesis Automaton**

- $\varepsilon$ transition
- $a_1$ transition
- $a_2$ transition

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 $\Sigma = [1, 100)$

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td>$a_1$</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>$a_2$</td>
<td>−</td>
<td>−</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_6$</th>
<th>$a_4$</th>
<th>$a_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$13$</td>
<td>+</td>
<td>−</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$73$</td>
<td>+</td>
<td>−</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$68$</td>
<td>−</td>
<td>−</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$18$</td>
<td>−</td>
<td>−</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$26$</td>
<td>−</td>
<td>−</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$44$</td>
<td>−</td>
<td>−</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$53$</td>
<td>−</td>
<td>−</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$73$</td>
<td>−</td>
<td>−</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Observation table:

Semantics:

- $\varepsilon \triangleright \Sigma \varepsilon = \{a_1, a_2\}$
- $a_1 \triangleright \Sigma a_1 = \{a_3, a_6\}$
- $a_2 \triangleright \Sigma a_2 = \{a_4, a_5\}$

Hypothesis automaton:

Ask Equivalence Query: True

Return current 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

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 $O(mn^2)$ MQ’s and at most $O(mn)$ EQ’s, when $\Sigma$ 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

$$\psi : \mathbb{B}^4 \rightarrow \{a_1, a_2, a_3\}$$

Boolean Function

Karnaugh map

Binary Decision Tree
Adaptation to the Boolean Alphabet

Learning Partitions

\[ \Sigma = \mathbb{B}^4 \]

\[ \psi(a) = \begin{cases} 
    a_1, & \text{if } \overline{x}_3 \\
    a_2, & \text{if } \overline{x}_1 \cdot x_3 \\
    a_3, & \text{if } x_1 \cdot x_3 
\end{cases} \]

Use Information Gain (Entropy) Measure to find Best Split

Adaptation to the Boolean Alphabet

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

observation table

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$a_1$</th>
<th>$a_0$</th>
<th>$a_1a_2$</th>
<th>$a_1a_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$-$</td>
<td>$+$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_1$</td>
<td></td>
<td></td>
<td>$-$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_0$</td>
<td></td>
<td>$-$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_1a_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_1a_3$</td>
<td></td>
<td>$+$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

semantics

hypothesis automaton
Adaptation to the Boolean Alphabet

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

**Observation Table**

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$a_1$</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>$a_0$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$a_1a_2$</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>$a_1a_3$</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

**Semantics**

**Hypothesis Automaton**

Ask Equivalence Query:

- **Counter-example:**
  
  $w = (1010) \cdot (0000)$, +

  $w = a_0 \cdot a_0$, −

- Add distinguishing string (0000)
  - discover new state
  - evidence incompatibility
Adaptation to the Boolean Alphabet

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

### Observation Table

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$a_1$</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>$a_1a_2$</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>$a_0$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$a_5$</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>$a_1a_3$</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>$a_1a_2a_4$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$a_1a_2a_6$</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

### Semantics

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>q_0</td>
<td>a_0</td>
<td>a_1</td>
<td>a_5</td>
</tr>
<tr>
<td>$a_1$</td>
<td>a_2</td>
<td>a_3</td>
<td>a_4</td>
<td>a_6</td>
</tr>
<tr>
<td>$a_0$</td>
<td>x_2</td>
<td>x_2 \lor x_3</td>
<td>x_1 \lor x_3</td>
<td>x_2</td>
</tr>
<tr>
<td>$a_5$</td>
<td>a_1</td>
<td>a_3</td>
<td>a_4</td>
<td>a_6</td>
</tr>
<tr>
<td>$a_1a_3$</td>
<td>q_0</td>
<td>a_1</td>
<td>a_2</td>
<td>a_5</td>
</tr>
<tr>
<td>$a_1a_2a_4$</td>
<td>a_0</td>
<td>a_1</td>
<td>a_2</td>
<td>a_5</td>
</tr>
<tr>
<td>$a_1a_2a_6$</td>
<td>q_0</td>
<td>a_1</td>
<td>a_2</td>
<td>a_5</td>
</tr>
</tbody>
</table>

### Hypothesis Automaton

Ask Equivalence Query:
Adaptation to the Boolean Alphabet

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

observation table

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$a_1$</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>$a_1a_2$</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>$a_0$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$a_5$</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>$a_1a_3$</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>$a_1a_2a_4$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$a_1a_2a_6$</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

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

Structure is fixed

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

MQ’s = MQ’s for learning + MQ’s for testing

Empirical Results

Comparison to the best $L^*$ algorithm

Experiment:

Target automaton:
- $\Sigma \subseteq \mathbb{N}$
- $|\Sigma| = 150$
- $3 \leq |Q| \leq 45$
- $|\Sigma_q| \leq 5$, $\forall q \in Q$

Random structure

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

MQ’s = MQ’s for learning + MQ’s for testing

---

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 $\leq 4$, $\forall q \in Q$

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

<table>
<thead>
<tr>
<th>0</th>
<th>16</th>
<th>32</th>
<th>48</th>
<th>64</th>
<th>80</th>
<th>96</th>
<th>112</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUL</td>
<td>DLE</td>
<td>SPC</td>
<td>0</td>
<td>@</td>
<td>P</td>
<td>`</td>
<td>p</td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>DC1</td>
<td>!</td>
<td>1</td>
<td>A</td>
<td>97</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>DC2</td>
<td>&quot;</td>
<td>2</td>
<td>B</td>
<td>98</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>DC3</td>
<td>#</td>
<td>3</td>
<td>C</td>
<td>99</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>DC4</td>
<td>$</td>
<td>4</td>
<td>D</td>
<td>100</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>NAK</td>
<td>%</td>
<td>5</td>
<td>E</td>
<td>101</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>SYN</td>
<td>&amp;</td>
<td>6</td>
<td>F</td>
<td>102</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td>ETB</td>
<td>'</td>
<td>7</td>
<td>G</td>
<td>103</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>CAN</td>
<td>(</td>
<td>8</td>
<td>H</td>
<td>104</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
<td>EM</td>
<td>)</td>
<td>9</td>
<td>I</td>
<td>105</td>
<td>i</td>
</tr>
<tr>
<td>10</td>
<td>LF</td>
<td>SUB</td>
<td>*</td>
<td>10</td>
<td>J</td>
<td>106</td>
<td>j</td>
</tr>
<tr>
<td>11</td>
<td>VT</td>
<td>ESC</td>
<td>+</td>
<td>11</td>
<td>K</td>
<td>107</td>
<td>k</td>
</tr>
<tr>
<td>12</td>
<td>FF</td>
<td>FS</td>
<td>,</td>
<td>12</td>
<td>L</td>
<td>108</td>
<td>l</td>
</tr>
<tr>
<td>13</td>
<td>CR</td>
<td>GS</td>
<td>-</td>
<td>13</td>
<td>M</td>
<td>109</td>
<td>m</td>
</tr>
<tr>
<td>14</td>
<td>SO</td>
<td>RS</td>
<td>.</td>
<td>14</td>
<td>N</td>
<td>110</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td>SI</td>
<td>US</td>
<td>/</td>
<td>15</td>
<td>O</td>
<td>111</td>
<td>o</td>
</tr>
</tbody>
</table>

Control Characters
Numerals
Punctuation Symbols
Lower-Case Letters
Upper-Case Letters
Empirical Results

Valid passwords over the ASCII characters

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

<table>
<thead>
<tr>
<th>Password Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (pin)</td>
<td>Length: 4 to 8. Contains only numbers.</td>
</tr>
<tr>
<td>B (easy)</td>
<td>Length: 4 to 8. It contains any printable character.</td>
</tr>
<tr>
<td>D (medium-strong)</td>
<td>Length: 6 to 14. Contains at least 1 number and 1 lower-case letter. Punctuation characters are allowed.</td>
</tr>
<tr>
<td>E (strong)</td>
<td>Length: 6 to 14. Contains at least 1 character from each group.</td>
</tr>
</tbody>
</table>
Empirical Results

Valid passwords over the ASCII characters

<table>
<thead>
<tr>
<th>Alphabet</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (pin)</td>
<td>Length: 4 to 8. Contains only numbers.</td>
</tr>
<tr>
<td>B (easy)</td>
<td>Length: 4 to 8. It contains any printable character.</td>
</tr>
<tr>
<td>D (medium-strong)</td>
<td>Length: 6 to 14. Contains at least 1 number and 1 lower-case letter. Punctuation characters are allowed.</td>
</tr>
<tr>
<td>E (strong)</td>
<td>Length: 6 to 14. Contains at least 1 character from each group.</td>
</tr>
</tbody>
</table>
Empirical Results

Valid passwords over the ASCII characters

\[ \Sigma = \{0, 1, \ldots, 127\} \]

\[ \Sigma = \mathcal{B}^7 \]
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.


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


- 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.


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


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


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!