A Tutorial on Automata Learning

Daniel Neider

MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS

MOVEP 2018
ENS Paris-Saclay, Cachan, France
20 July 2018
It is possible to test a deterministic pushdown machine to determine if the language it recognizes is regular.

The object of this paper is to show that, given a deterministic pushdown recognition machine, it is possible to determine if the set of input strings it recognizes is regular. In particular, we will show that if the set is regular, then the number of states in the reduced state machine which recognizes the set may be bounded by an expression of the order $t q^2$ (when $q, t > 1$) where $q$ is the number of control states of the pushdown machine and $t$ is the size of the pushdown tape alphabet.
It is *undecidable* whether a pushdown automaton accepts a regular language.
It is undecidable whether a pushdown automaton accepts a regular language.

It is decidable whether a deterministic pushdown automaton accepts a regular language.
A Regularity Test for Pushdown Machines

R. E. STEARNS

General Electric Research and Development Center, P.O. Box 8, Schenectady, New York 12301

It is possible to test a deterministic pushdown machine to determine if the language it recognizes is regular.

The object of this paper is to show that, given a deterministic pushdown recognition machine, it is possible to determine if the set of input strings it recognizes is regular. In particular, we will show that if the set is regular, then the number of states in the reduced state machine which recognizes the set may be bounded by an expression of the order

\[ q^{t^2} \]

(when \( q, t > 1 \)) where \( q \) is the number of control states of the pushdown machine and \( t \) is the size of the pushdown tape alphabet. There-
Puzzle

Given a deterministic pushdown automaton (PDA) accepting a regular language, construct an equivalent finite automaton.
Puzzle

Given a deterministic pushdown automaton (PDA) accepting a regular language, construct an equivalent finite automaton

**Hint**

- PDAs are closed under complement and intersection with regular languages
- Emptiness of DPAs is decidable
Automata learning is the process of constructing finite-state machines from data (i.e., words)
Automata learning is the process of constructing finite-state machines from data (i.e., words)
Automata learning is the process of constructing finite-state machines from data (i.e., words)
Automata learning is the process of constructing finite-state machines from data (i.e., words)

Why Automata Learning?
Automata are good at

- *pattern recognition*
  spam detection, language translations, ...

- *modeling complex systems*
  model learning, black-box verification, ...

- *symbolically encoding infinite sets*
  learning invariants, learning reachable states in games, ...
Automata learning is the process of constructing finite-state machines from data (i.e., words)

Dimensions of Automata Learning

▶ Automata models
   DFAs, NFA, stochastic automata, register automata, ...

▶ Learning model
   probably approximately correct, in-the-limit, passive, active, ...

▶ Theory and/or applications
Automata learning is the process of constructing finite-state machines from data (i.e., words)

Dimensions of Automata Learning

- **Automata models**
  - DFAs, NFA, stochastic automata, register automata, ...

- **Learning model**
  - probably approximately correct, in-the-limit, passive, active, ...

- **Theory and/or applications**
1. Passive Automata Learning

2. Active Automata Learning

3. Applications
Deterministic Finite Automata (DFAs)

\[ A = (Q, \Sigma, q_0, \delta, F) \]
Deterministic Finite Automata (DFAs)

\[ \mathcal{A} = (Q, \Sigma, q_0, \delta, F) \]
Deterministic Finite Automata (DFAs)

\[ \mathcal{A} = (Q, \Sigma, q_0, \delta, F) \]
Deterministic Finite Automata (DFAs)

\[ A = (Q, \Sigma, q_0, \delta, F) \]
A deterministic finite automaton (DFA) is defined as:

\[ A = (Q, \Sigma, q_0, \delta, F) \]

where \( \delta : Q \times \Sigma \rightarrow Q \) is the transition function.
Deterministic Finite Automata (DFAs)

\[ A = (Q, \Sigma, q_0, \delta, F) \]
Deterministic Finite Automata (DFAs)

Run on word $u = a_1 \ldots a_n \in \Sigma^*$: $A$: $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} q_n$
Deterministic Finite Automata (DFAs)

Run on word $u = a_1 \ldots a_n \in \Sigma^*$: $A$: $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} q_n$

A run is accepting if $q_n \in F$
Deterministic Finite Automata (DFAs)

Run on word \( u = a_1 \ldots a_n \in \Sigma^* \): \( A \): \( q_0 \overset{a_1}{\rightarrow} q_1 \overset{a_2}{\rightarrow} \ldots \overset{a_n}{\rightarrow} q_n \)

A run is accepting if \( q_n \in F \)

\[
\begin{align*}
ab & \checkmark \\
abaab & \checkmark \\
babba & \times
\end{align*}
\]
Deterministic Finite Automata (DFAs)

- **Run** on word $u = a_1 \ldots a_n \in \Sigma^*$: $\mathcal{A}$: $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} q_n$
- A run is **accepting** if $q_n \in F$
- The **language** of a DFA $\mathcal{A}$ is $L(\mathcal{A}) = \{ u \in \Sigma^* \mid \mathcal{A} \text{ accepts } u \}$
Deterministic Finite Automata (DFAs)

Run on word \( u = a_1 \ldots a_n \in \Sigma^* \): \( A: q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} q_n \)

- A run is accepting if \( q_n \in F \)
- The language of a DFA \( A \) is \( L(A) = \{ u \in \Sigma^* \mid A \text{ accepts } u \} \)
- A language \( L \subseteq \Sigma^* \) is regular if there exists a DFA \( A \) with \( L = L(A) \)
Passive Automata Learning
A sample is a tuple $S = (S_+, S_-)$ where $S_+, S_- \subseteq \Sigma^*$ are finite sets of words (and $S_+ \cap S_- = \emptyset$).
A sample is a tuple $S = (S_+, S_-)$ where $S_+, S_- \subseteq \Sigma^*$ are finite sets of words (and $S_+ \cap S_- = \emptyset$).

A DFA $A$ is called consistent with $S$ if it satisfies

$$S_+ \subseteq L(A) \text{ and } S_- \cap L(A) = \emptyset$$
A **sample** is a tuple $S = (S_+, S_-)$ where $S_+, S_- \subseteq \Sigma^*$ are finite sets of words (and $S_+ \cap S_- = \emptyset$).

A DFA $A$ is called **consistent** with $S$ if it satisfies

$$S_+ \subseteq L(A) \quad \text{and} \quad S_- \cap L(A) = \emptyset$$

**Passive Learning**

Given a sample $S$, the passive learning task is to compute a (smallest) DFA that is consistent with $S$. 
Passive Learning – A Separation Problem

$A \in S^+ + S^-$
Consider $S = (S_+, S_-)$ with $S_+ = \{a\}$ and $S_- = \{b\}$
\( S_+ = \{aa, ba, aba\} \)
\( S_- = \{a, ab\} \)
The prefix tree acceptor is the DFA $A_{Pref}(S_+) = (Q, \Sigma, q_0, \delta, F)$ with

- $Q = \text{Pref}(S_+) \cup \{\bot\}$
- $q_0 = \varepsilon$
- $\delta(q, a) = \begin{cases} ua & \text{if } q = u \text{ and } ua \in \text{Pref}(S_+) \\ \bot & \text{otherwise} \end{cases}$
- $F = S_+$

where $S_+ = \{aa, ba, aba\}$ and $S_- = \{a, ab\}$.
Observation

Let $S_+ \subset \Sigma^*$ be a finite set of words. Then, $L(A_{Pref}(S_+)) = S_+$

- Thus, the prefix tree acceptor solves the passive learning task
- The prefix tree acceptor has $|S_+| + 1$ states
Observation

Let \( S_+ \subset \Sigma^* \) be a finite set of words. Then, \( L(A_{Pref}(S_+)) = S_+ \)

- Thus, the prefix tree acceptor solves the passive learning task
- The prefix tree acceptor has \( |S_+| + 1 \) states

Problem

The prefix tree acceptor is overfitting the training data

- It does not generalize the data
- It does not predict well
William of Ockham (c. 1287 – 1347)

“Entities must not be multiplied beyond necessity”

Ptolemy (c. AD 100 – c. 170)

“We consider it a good principle to explain the phenomena by the simplest hypothesis possible”
Theorem (Gold [1], 1978)

The decision problem "Given a sample $S$ and $k \in \mathbb{N}$. Does a DFA with at most $k$ states that is consistent with $S$ exist?" is NP-complete.
Theorem (Gold [1], 1978)

The decision problem “Given a sample $S$ and $k \in \mathbb{N}$. Does a DFA with at most $k$ states that is consistent with $S$ exist?” is NP-complete.
1.1. Regular Positive Negative Inference
Let $\Sigma$ be an alphabet and $<_{\Sigma}$ a total order on $\Sigma$

A word $u = a_1 \ldots a_m$ is *smaller* than a word $v = a'_1 \ldots a'_n$, denoted by $u < v$, if

- $m < n$; or

- $m = n$ and there exists an $i \in \{1, \ldots, n\}$ such that $a_i <_{\Sigma} a'_i$ and $a_j = a'_j$ for each $j \in \{1, \ldots, i - 1\}$

**Example**

Let $\Sigma = \{a, b\}$ and $a <_{\Sigma} b$

$$\varepsilon < a < b < aa < ab < ba < bb < aaa < aab < \ldots$$
Lemma

Let $A$ be a DFA and $A'$ the DFA resulting from merging states in $A$. Then, $L(A) \subseteq L(A')$. 
Lemma

Let $A$ be a DFA and $A'$ the DFA resulting from merging states in $A$. Then, $L(A) \subseteq L(A')$. 

Daniel Neider: A Tutorial on Automata Learning
Lemma

Let \( A \) be a DFA and \( A' \) the DFA resulting from merging states in \( A \). Then, \( L(A) \subseteq L(A') \).
Lemma

Let $A$ be a DFA and $A'$ the DFA resulting from merging states in $A$. Then, $L(A) \subseteq L(A')$. 

Daniel Neider: A Tutorial on Automata Learning
Lemma

Let $A$ be a DFA and $A'$ the DFA resulting from merging states in $A$. Then, $L(A) \subseteq L(A')$. 
Algorithm 1: RPNI (simplified)

Input: A sample $S = (S_-, S_+)$

1. Construct $A_{Pref}(S_+) = (Q, \Sigma, q_0, \delta, F)$ without sink state
2. Order the set $Q = \{u_0, \ldots, u_n\}$ in canonical order (s.t. $u_i < u_{i+1}$)
3. for $i = 1, \ldots, n$ do
   4. if state $u_i$ is not yet merged with a smaller state then
      5. Try to merge $u_i$ with a smaller state $u_0, \ldots, u_{i-1}$ until the first merged DFA does not accept a negative example
   6. end
4. end
8. return the final DFA
Consider the sample $S = (S_+, S_-)$ with

- $S_+ = \{b, aa, aba\}$
- $S_- = \{a, ab\}$
Consider the sample $S = (S_+, S_-)$ with

- $S_+ = \{b, aa, aba\}$
- $S_- = \{a, ab\}$

$\varepsilon < a < b < aa < ab < aba$
Consider the sample $S = (S_+, S_-)$ with

- $S_+ = \{ b, aa, aba \}$
- $S_- = \{ a, ab \}$

$\varepsilon < a < b < aa < ab < aba$
Consider the sample $S = (S_+, S_-)$ with

$\triangleright S_+ = \{b, aa, aba\}$

$\triangleright S_- = \{a, ab\}$

$\epsilon < a < b < aa < ab < aba$
Consider the sample $S = (S_+, S_-)$ with

- $S_+ = \{b, aa, aba\}$
- $S_- = \{a, ab\}$

$\epsilon < a < b < aa < ab < aba$
Consider the sample $S = (S_+, S_-)$ with

- $S_+ = \{ b, aa, aba \}$
- $S_- = \{ a, ab \}$

$\varepsilon < a < b < aa < ab < aba$
Consider the sample $S = (S_+, S_-)$ with

- $S_+ = \{b, aa, aba\}$
- $S_- = \{a, ab\}$

$$\varepsilon < a < b < aa < ab < aba$$
Consider the sample $S = (S_+, S_-)$ with

- $S_+ = \{b, aa, aba\}$
- $S_- = \{a, ab\}$

$$\varepsilon < a < b < aa < ab < aba$$
Consider the sample $S = (S_+, S_-)$ with

- $S_+ = \{b, aa, aba\}$
- $S_- = \{a, ab\}$

$$\varepsilon < a < b < aa < ab < aba$$
Consider the sample $S = (S_+, S_-)$ with

- $S_+ = \{b, aa, aba\}$
- $S_- = \{a, ab\}$

$\epsilon < a < b < aa < ab < aba$
Consider the sample $S = (S_+, S_-)$ with

- $S_+ = \{b, aa, aba\}$
- $S_- = \{a, ab\}$

$\varepsilon < a < b < aa < ab < aba$
Consider the sample $S = (S_+, S_-)$ with

- $S_+ = \{ b, aa, aba \}$
- $S_- = \{ a, ab \}$

$\varepsilon < a < b < aa < ab < aba$
Example

Consider the sample $S = (S_+, S_-)$ with

- $S_+ = \{ b, aa, aba \}$
- $S_- = \{ a, ab \}$

$\varepsilon < a < b < aa < ab < aba$
Consider the sample $S = (S_+, S_-)$ with

- $S_+ = \{b, aa, aba\}$
- $S_- = \{a, ab\}$

$\varepsilon < a < b < aa < ab < aba$
Theorem (Oncina and Garcia [2], 1992)

Let $S = (S_+, S_-)$ be a sample, $n = |\text{Pref}(S_+)|$, and $m = \sum_{u \in S_-} |u|$. Then,

- RPNI produces a (partial) DFA that is consistent with $S$
- the runtime is bounded by $\mathcal{O}(n^2(n + m))$
Theorem (Oncina and Garcia [2], 1992)

Let $S = (S_+, S_-)$ be a sample, $n = |\text{Pref}(S_+)|$, and $m = \sum_{u \in S_-} |u|$. Then,

- RPNI produces a (partial) DFA that is consistent with $S$
- the runtime is bounded by $\mathcal{O}(n^2(n + m))$

Proof

- $\mathcal{A}_{\text{Pref}}(S_+)$ is consistent with $S$
- Merging states only increases the accepted language
- A merged DFA is only kept if it is still consistent with $S$
Theorem (Oncina and Garcia [2], 1992)

Let $S = (S_+, S_-)$ be a sample, $n = |\text{Pref}(S_+)|$, and $m = \sum_{u \in S_-} |u|$. Then,

- RPNI produces a (partial) DFA that is consistent with $S$
- the runtime is bounded by $\mathcal{O}(n^2(n + m))$

Proof

- At most $\sum_{i=1}^{n-1} i \in \mathcal{O}(n^2)$ merges
- Each merge can be performed in time $\mathcal{O}(n)$
- Checking whether a DFA is consistent can be done in time $\mathcal{O}(m)$
Why does RPNI merge states in its particular order?

**Theorem (Oncina and Garcia [2], 1992)**

*If a sample contains “sufficient” information about a regular language \( L \subseteq \Sigma^* \) (called a “characteristic sample”), then RPNI computes the unique minimal DFA accepting \( L \).*
Abbadingo One: DFA Learning Competition

The Abbadingo One competition is over. Plans are being made for Abbadingo 2, with new sorts of problems.

A test version of a follow-on to Abbadingo One (but not Abbadingo Two, which will be different) called Gowachin is now available for your grammar learning pleasure.

Meanwhile, here is a forthcoming paper describing the outcome of the first competition, which will appear in the volume of Springer Verlag's Lecture Notes in Computer Science containing the proceedings of ICGI-98. And here is an expanded version of the ICGI-98 paper, currently in review by a journal. And here is code for 4 DFA learning algorithms, including 3 versions of Rodney Price's EDSM idea. These 3 programs can solve Abbadingo problems 1, 2, 3, R, 4, 6, and 8.

High-Level Executive Summary

This research competition is based on grammar induction, a difficult problem in machine learning. The winners will receive scientific admiration, one thousand twenty four dollars, guest of honor status at a formal award ceremony, and more.

Read the description, read the details and clarifications, learn from the data sets, satisfy the Abbadingo Oracle, then email the administrators!

Mid-Level Executive Summary

Announcement for Technical Forums

Press Release for General Distribution
Results of the Abbadingo One DFA Learning Competition

and

a New Evidence Driven State Merging Algorithm

Kevin J. Lang
NEC Research Institute
4 Independence Way
Princeton, NJ 08540
kevin@research.nj.nec.com

Barak A. Pearlmutter
Comp Sci Dept, FEC 313
Univ of New Mexico
Albuquerque, NM 87131
bap@cs.unm.edu

Rodney A. Price*
Emtex
Milton Keynes, England
rod@emtex.com

May 8, 1998

Abstract

This paper first describes the structure and results of the Abbadingo One DFA Learning Competition. The competition was designed to encourage work on algorithms that scale well—both to larger DFAs and to sparser training data. We then describe and discuss the winning algorithm of Rodney Price, which orders...
1.2. Passive Learning of Minimal DFAs
Passive Learning of Minimal DFAs

Idea

- Translate the passive learning problem into a sequence of satisfiability problems in a suitable logic
- Use a highly optimized solver to solve these logical problems
Passive Learning of Minimal DFAs

Idea

- Translate the passive learning problem into a sequence of satisfiability problems in a suitable logic
- Use a highly optimized solver to solve these logical problems

For a sample $S$ and natural numbers $n \geq 1$, we construct a sequence of formula $\varphi_n^S$ that satisfy the following:

1. $\varphi_n^S$ is satisfiable if and only if there exists a DFA with $n$ states that is consistent with $S$
2. A model $M$ of $\varphi_n^S$ contains sufficient information to construct a consistent DFA $A_M$
Algorithm 2: Passive Learning of Minimal DFAs

Input: A sample $S$

1. $n ← 0$
2. repeat
3. $n ← n + 1$
4. Construct and solve $\varphi_n^S$
5. until $\varphi_n^S$ is satisfiable, say with model $M$
6. return $A_M$
Algorithm 2: Passive Learning of Minimal DFAs

Input: A sample $S$

1. $n \leftarrow 0$
2. repeat
3. $n \leftarrow n + 1$
4. Construct and solve $\varphi_n^S$
5. until $\varphi_n^S$ is satisfiable, say with model $\mathcal{M}$
6. return $A_{\mathcal{M}}$

- This algorithm is guaranteed to terminate (given the properties of $\varphi_n^S$), and $|A_{\text{Pref}}(S_+)|$ is an upper bound for $n$
- But: what is the best way to find the minimal value of $n$?
Various Passive Learning Algorithms

We will consider four different encodings into logic:

1. Biermann and Feldmann’s method
2. Grinchtein, Leucker, and Piterman’s unary method
3. Grinchtein, Leucker, and Piterman’s binary method
4. Heule and Verwer’s method
We will consider four different encodings into logic:

1. Biermann and Feldmann’s method
2. Grinchtein, Leucker, and Piterman’s unary method
3. Grinchtein, Leucker, and Piterman’s binary method
4. Heule and Verwer’s method

Fix the size $n \geq 1$ of the prospective DFA
Biermann and Feldman’s Method
Biermann and Feldman’s Method

Idea

Encode the run of a DFA on (prefixes of) words from the sample $S$
Biermann and Feldman’s Method

Idea
Encode the run of a DFA on (prefixes of) words from the sample $S$

Variables
For $u \in \operatorname{Pref}(S_+ \cup S_-)$, let $x_u \in \{1, \ldots, n\}$ be the state reached after reading $u$
Biermann and Feldman’s Method

**Idea**

Encode the run of a DFA on (prefixes of) words from the sample $S$

**Variables**

For $u \in \text{Pref}(S_+ \cup S_-)$, let $x_u \in \{1, \ldots, n\}$ be the state reached after reading $u$

**Constraints**

\[
\bigwedge_{ua, u' a \in \text{Pref}(S_+ \cup S_-)} x_u = x_{u'} \rightarrow x_{ua} = x_{u'a} \\
\bigwedge_{u \in S_+, u' \in S_-} x_u \neq x_{u'}
\]
Biermann and Feldman’s Method

Idea

Encode the run of a DFA on (prefixes of) words from the sample \( S \)

Variables

For \( u \in \text{Pref}(S_+ \cup S_-) \), let \( x_u \in \{1, \ldots, n\} \) be the state reached after reading \( u \)

Constraints

\[
\bigwedge_{ua,u'a \in \text{Pref}(S_+ \cup S_-)} x_u = x_{u'} \rightarrow x_{ua} = x_{u'a}
\]

\[
\bigwedge_{u \in S_+, u' \in S_-} x_u \neq x_{u'}
\]

Let \( \varphi^S_n(\vec{x}) \) denote the conjunction of these constraints
Let $\mathcal{M}: \mathcal{X} \to \{1, \ldots, n\}$ be a model of $\varphi_n^S(\vec{x})$

We define the DFA $A_{\mathcal{M}} = (Q, \Sigma, q_0, \delta, F)$ by

$\triangleright Q = \{1, \ldots, n\}$
Let $M: X \rightarrow \{1, \ldots, n\}$ be a model of $\varphi^S_n(\vec{x})$

We define the DFA $A_M = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = \{1, \ldots, n\}$
- $q_0 = i$ if and only if $M(x^i) = i$
Let $\mathcal{M}: \mathcal{X} \rightarrow \{1, \ldots, n\}$ be a model of $\varphi_n^S(\bar{x})$

We define the DFA $A_{\mathcal{M}} = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = \{1, \ldots, n\}$
- $q_0 = i$ if and only if $\mathcal{M}(x_\varepsilon) = i$
- $\delta(i, a) = \begin{cases} j & \text{if } ua \in \text{Pref}(S_+ \cup S_-)\text{ exists such that } \mathcal{M}(x_u) = i \text{ and } \mathcal{M}(x_{ua}) = j \\ i & \text{otherwise} \end{cases}$

Note that $A_{\mathcal{M}}$ is well-defined
Let $\mathcal{M}: \mathcal{X} \rightarrow \{1, \ldots, n\}$ be a model of $\varphi_n^S(\vec{x})$

We define the DFA $A_{\mathcal{M}} = (Q, \Sigma, q_0, \delta, F)$ by

$\begin{align*}
\text{\textbullet } Q &= \{1, \ldots, n\} \\
\text{\textbullet } q_0 &= i \text{ if and only if } \mathcal{M}(x_\varepsilon) = i \\
\text{\textbullet } \delta(i, a) &= \begin{cases} 
  j & \text{if } ua \in \text{Pref} (S_+ \cup S_-) \text{ exists such that} \\
  \mathcal{M}(x_u) = i \text{ and } \mathcal{M}(x_{ua}) = j \\
  i & \text{otherwise} 
\end{cases} \\
\text{\textbullet } F &= \{i \in Q \mid \exists u \in S^+: \mathcal{M}(x_u) = i\}
\end{align*}$
Let $\mathcal{M}: \mathcal{X} \rightarrow \{1, \ldots, n\}$ be a model of $\varphi_n^{S}(\vec{x})$

We define the DFA $A_{\mathcal{M}} = (Q, \Sigma, q_0, \delta, F)$ by

$\hspace{1cm} Q = \{1, \ldots, n\}$

$\hspace{1cm} q_0 = i$ if and only if $\mathcal{M}(x_{\epsilon}) = i$

$\hspace{1cm} \delta(i, a) = \begin{cases} j & \text{if } ua \in \text{Pref}(S_{+} \cup S_{-}) \text{ exists such that} \\ M(x_u) = i \text{ and } M(x_{ua}) = j \\ i & \text{otherwise} \end{cases}$

$\hspace{1cm} F = \{i \in Q \mid \exists u \in S^{+}: M(x_u) = i\}$

Note that $A_{\mathcal{M}}$ is well-defined
Theorem (Biermann and Feldman [4], 1972)

1. Let $S$ be a sample, $\mathcal{M} : \mathcal{X} \rightarrow \{1, \ldots, n\}$ a model of $\varphi^n_S(\vec{x})$, and $A_{\mathcal{M}}$ the DFA constructed from $\mathcal{M}$. Then, $A_{\mathcal{M}}$ is a DFA with $n$ states that is consistent with $S$.

2. If a consistent DFA with $n$ states exists, then $\varphi^n_S(\vec{x})$ is satisfiable.
Theorem (Biermann and Feldman [4], 1972)

1. Let $S$ be a sample, $M \colon X \rightarrow \{1, \ldots, n\}$ a model of $\varphi^S_n(\vec{x})$, and $A_M$ the DFA constructed from $M$. Then, $A_M$ is a DFA with $n$ states that is consistent with $S$

2. If a consistent DFA with $n$ states exists, then $\varphi^S_n(\vec{x})$ is satisfiable

Proof Sketch

1. Show (by induction) that $M(x_u) = i$ indeed implies $A_M : q_0 \xrightarrow{u} i$ for all $u \in \text{Pref}(S_+ \cup S_-)$
   - Using this result, conclude that $A_M$ is consistent with $S$

2. Use the runs of a consistent DFA on words from $S$ to define a model of $\varphi^S_n$
Theorem (Biermann and Feldman [4], 1972)

1. Let $S$ be a sample, $\mathcal{M} : \mathcal{X} \rightarrow \{1, \ldots, n\}$ a model of $\varphi_n^S(\vec{x})$, and $A_M$ the DFA constructed from $\mathcal{M}$. Then, $A_M$ is a DFA with $n$ states that is consistent with $S$

2. If a consistent DFA with $n$ states exists, then $\varphi_n^S(\vec{x})$ is satisfiable

Observation

- When adding the additional constraints $1 \leq x_u \leq n$, one can use off-the-shelf solvers for linear integer arithmetic
- There exist solver that can find models with minimal range
Grinchtein, Leucker, and Piterman’s Methods
Idea

Translate Biermann and Feldman’s method into *propositional Boolean logic* and use a highly-optimized SAT solver to find a solution.
Grinchtein, Leucker, and Piterman’s Method

Idea

Translate Biermann and Feldman’s method into propositional Boolean logic and use a highly-optimized SAT solver to find a solution.

Variables

For $u \in Pref(S_+ \cup S_-)$ and $q \in \{1, \ldots, n\}$, let $x_{u,q} \in \{0, 1\}$ indicate that state $q$ is reached after reading the word $u$.

- That is, each variable $x_u \in \{1, \ldots, n\}$ in Biermann and Feldman’s method is encoded by the Boolean variables $x_{u,1}, \ldots, x_{u,n}$.
Let $\psi_{S_n}(\vec{x})$ denote the conjunction of these constraints.
Constraints

\[ \bigwedge_{u \in \text{Pref}(S_+ \cup S_-)} \bigvee_{q \in Q} x_{u,q} \]

\[ \bigwedge_{u \in \text{Pref}(S_+ \cup S_-)} \bigwedge_{q \neq q' \in Q} \neg x_{u,q} \vee \neg x_{u,q'} \]

\[ \bigwedge_{ua, u'a \in \text{Pref}(S_+ \cup S_-)} \bigwedge_{p, q \in Q} (x_{u,p} \land x_{u',p}) \rightarrow (x_{ua,q} \leftrightarrow x_{u'a,q}) \]

\[ \bigwedge_{u \in S_+, u' \in S_-} \bigwedge_{q \in Q} \neg x_{u,q} \vee \neg x_{u',q} \]

Let \( \psi_n^S(\vec{x}) \) denote the conjunction of these constraints
Let $\mathcal{M}: \mathcal{X} \rightarrow \{0, 1\}$ be a model of $\psi_n^S(\vec{x})$

We define the DFA $A_{\mathcal{M}} = (Q, \Sigma, q_0, \delta, F)$ by

$\triangleright Q = \{1, \ldots, n\}$
Let $\mathcal{M} : \mathcal{X} \rightarrow \{0, 1\}$ be a model of $\psi_n^S(\vec{x})$

We define the DFA $A_{\mathcal{M}} = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = \{1, \ldots, n\}$
- $q_0 = i$ for the unique $i \in Q$ with $\mathcal{M}(x_\varepsilon, i) = 1$
Let $\mathcal{M} : \mathcal{X} \rightarrow \{0, 1\}$ be a model of $\psi^S_n(\vec{x})$

We define the DFA $A_{\mathcal{M}} = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = \{1, \ldots, n\}$
- $q_0 = i$ for the unique $i \in Q$ with $\mathcal{M}(x_\varepsilon, i) = 1$
- $\delta(i, a) = \begin{cases} j & \text{if } ua \in Pref(S_+ \cup S_-) \text{ exists such that} \\ M(x_u, i) = 1 \text{ and } M(x_{ua}, j) = 1 \\ i & \text{otherwise} \end{cases}$

$F = \{i \in Q \mid \exists u \in S_+^*: M(x_u, i) = 1 \}$
Let $\mathcal{M}: \mathcal{X} \rightarrow \{0, 1\}$ be a model of $\psi_n^{\mathcal{S}}(\mathcal{X})$

We define the DFA $A_{\mathcal{M}} = (Q, \Sigma, q_0, \delta, F)$ by

1. $Q = \{1, \ldots, n\}$
2. $q_0 = i$ for the unique $i \in Q$ with $\mathcal{M}(x_{\varepsilon,i}) = 1$
3. $\delta(i, a) = \begin{cases} j & \text{if } ua \in \text{Pref}(S_+ \cup S_-) \text{ exists such that } \mathcal{M}(x_u,i) = 1 \text{ and } \mathcal{M}(x_{ua,j}) = 1 \\ i & \text{otherwise} \end{cases}$
4. $F = \{i \in Q \mid \exists u \in S^+: \mathcal{M}(x_u,i) = 1\}$
Theorem (Grinchtein, Leucker, and Piterman [5], 2006)

- Let $S$ be a sample, $\mathcal{M}: \mathcal{X} \rightarrow \{0, 1\}$ a model of $\psi^S_n(\vec{x})$, and $A_{\mathcal{M}}$ the DFA constructed from $\mathcal{M}$. Then, $A_{\mathcal{M}}$ is a DFA with $n$ states that is consistent with $S$.

- If a consistent DFA with $n$ states exists, then $\psi^S_n(\vec{x})$ is satisfiable.
Idea

Encode the state reached after reading some input \( u \in \text{Pref}(S_+ \cup S_-) \) in binary rather than in unary

Variables

For \( u \in \text{Pref}(S_+ \cup S_-) \) use Boolean variables \( x_{u,1}, \ldots, x_{u,[\log_2 n]} \) to encode the state reached after reading \( u \)
Heule and Verwer’s Method
Heule and Verwer’s Method

Idea

Encode the transitions and final states of an automaton explicitly

\[ \delta(p, a) = q \]

\[ f(q) = \begin{cases} 0, & \text{for } q \in Q \text{ indicates that } q \text{ is a final state} \\ 1, & \text{otherwise} \end{cases} \]

\[ x(u, q) = \begin{cases} 0, & \text{for } u \in \text{Pref}(S^+ \cup S^-) \text{ and } q \in Q \text{ indicates that } q \text{ is reached after reading } u \end{cases} \]
Heule and Verwer’s Method

Idea

Encode the transitions and final states of an automaton explicitly

Variables

- \( d_{p,a,q} \in \{0, 1\} \) for \( p, q \in Q \) and \( a \in \Sigma \) encodes the transition \( \delta(p, a) = q \)
- \( f_q \in \{0, 1\} \) for \( q \in Q \) indicates that \( q \) is a final state
- \( x_{u,q} \in \{0, 1\} \) for \( u \in \text{Pref}(S_+ \cup S_-) \) and \( q \in Q \) indicates that state \( q \) is reached after reading \( u \)
The following constraints enforce that the variables $d_{p,a,q}$ encode a deterministic transition function:

$$\bigwedge_{p \in Q} \bigwedge_{a \in \Sigma} \bigvee_{q \in Q} d_{p,a,q}$$

$$\bigwedge_{p \in Q} \bigwedge_{a \in \Sigma} \bigwedge_{q \neq q' \in Q} \neg d_{p,a,q} \lor \neg d_{p,a,q'}$$
\[ \bigwedge_{u \in \text{Pref}(S_+ \cup S_-)} \bigvee_{q \in Q} x_{u,q} \]

\[ \bigwedge_{u \in \text{Pref}(S_+ \cup S_-)} \bigwedge_{q \neq q' \in Q} \neg x_{u,q} \lor \neg x_{u,q'} \]
\[
\begin{align*}
&\bigwedge_{u \in \text{Pref}(S_+ \cup S_-)} \bigvee_{q \in Q} x_{u,q} \\
&\bigwedge_{u \in \text{Pref}(S_+ \cup S_-)} \bigwedge_{q \neq q' \in Q} \neg x_{u,q} \lor \neg x_{u,q'} \\
&\bigwedge_{ua \in \text{Pref}(S_+ \cup S_-)} \bigwedge_{p, q \in Q} (x_{u,p} \land d_{p,a,q}) \rightarrow x_{ua,q}
\end{align*}
\]
Constraints II

\[
\bigwedge_{u \in \text{Pref}(S_+ \cup S_-)} \bigvee_{q \in Q} x_{u,q} \\
\bigwedge_{u \in \text{Pref}(S_+ \cup S_-)} \bigwedge_{q \neq q' \in Q} \neg x_{u,q} \lor \neg x_{u,q'} \\
\bigwedge_{ua \in \text{Pref}(S_+ \cup S_-)} \bigwedge_{p,q \in Q} (x_{u,p} \land d_{p,a,q}) \rightarrow x_{ua,q} \\
\left[ \bigwedge_{u \in S_+} \bigwedge_{q \in Q} x_{u,q} \rightarrow f_q \right] \land \left[ \bigwedge_{u \in S_-} \bigwedge_{q \in Q} x_{u,q} \rightarrow \neg f_q \right]
\]

Let \( \chi_n^{S}(\vec{d}, \vec{f}, \vec{x}) \) denote the conjunction of these constraints.
Let $\mathcal{M} : \mathcal{X} \rightarrow \{0, 1\}$ be a model of $\chi^n_S(\vec{d}, \vec{f}, \vec{x})$

We define the DFA $A_M = (Q, \Sigma, q_0, \delta, F)$ by

1. $Q = \{1, \ldots, n\}$
Let $\mathcal{M}: \chi \to \{0, 1\}$ be a model of $\chi^S_n(\vec{d}, \vec{f}, \vec{x})$

We define the DFA $A_\mathcal{M} = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = \{1, \ldots, n\}$
- $q_0 = i$ for the unique $i \in Q$ with $\mathcal{M}(x_\varepsilon, i) = 1$
Let $\mathcal{M}: \mathcal{X} \rightarrow \{0, 1\}$ be a model of $\chi_n^S(\vec{d}, \vec{f}, \vec{x})$

We define the DFA $A_{\mathcal{M}} = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = \{1, \ldots, n\}$
- $q_0 = i$ for the unique $i \in Q$ with $\mathcal{M}(x, i) = 1$
- $\delta(i, a) = j$ for the unique $j \in Q$ with $\mathcal{M}(d_i, a, j) = 1$
Let $\mathcal{M}: \mathcal{X} \to \{0, 1\}$ be a model of $\chi_n^S(\vec{d}, \vec{f}, \vec{x})$

We define the DFA $A_{\mathcal{M}} = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = \{1, \ldots, n\}$
- $q_0 = i$ for the unique $i \in Q$ with $\mathcal{M}(x_\varepsilon, i) = 1$
- $\delta(i, a) = j$ for the unique $j \in Q$ with $\mathcal{M}(d_i, a, j) = 1$
- $F = \{i \in Q \mid \mathcal{M}(f_i) = 1\}$
Theorem (Heule and Verwer [6], 2010)

Let $S$ be a sample, $M: \mathcal{X} \rightarrow \{0, 1\}$ a model of $\chi^S_n(\vec{d}, \vec{f}, \vec{x})$, and $A_M$ the DFA constructed from $M$. Then, $A_M$ is a DFA with $n$ states that is consistent with $S$.

If a consistent DFA with $n$ states exists, then $\chi^S_n(\vec{d}, \vec{f}, \vec{x})$ is satisfiable.
Let $k = |\text{Pref}(S_+ \cup S_-)|$

<table>
<thead>
<tr>
<th>Method</th>
<th>Logic</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biermann and Feldmann</td>
<td>Equality logic</td>
<td>$O(k)$ variables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O(k +</td>
</tr>
<tr>
<td>Grinchtein, Leucker, and Pitermann (unary)</td>
<td>Propositional Boolean logic</td>
<td>$O(kn)$ variables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O(kn^2 + n</td>
</tr>
<tr>
<td>Grinchtein, Leucker, and Pitermann (binary)</td>
<td>Propositional Boolean logic</td>
<td>$O(k \log_2 n)$ variables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O(kn \log_2 n + n</td>
</tr>
<tr>
<td>Heule and Verwer</td>
<td>Propositional Boolean logic</td>
<td>$O(n^2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O(n^3</td>
</tr>
</tbody>
</table>
Heule and Verwer’s benchmark suite: 810 samples, containing approx. 550 words over $\Sigma = \{0, 1\}$ each
Empirical Comparison

250 random samples, containing 150 words each
http://libalf.informatik.rwth-aachen.de/
LibALF

libalf: The Automata Learning Framework
A comprehensive, open-source library for learning finite-state automata

- About
- Demo
- Download
- Documentation
- Literature
- Team
- Contact

Welcome to the home of libalf

The libalf library is a comprehensive, open-source library for learning finite-state automata covering various well-known learning techniques (such as Angluin’s L*, Biermann’s learning approach, and RPN1), as well as novel learning algorithms (e.g., for NFA and visibly one-counter automata).

Please refer to the About page for more information about libalf.

libalf in action

We are happy that various researchers are already using libalf for their projects. You may want to have a look at the following papers, which give an impression about what libalf can be used for:


In-development notice

Please note that libalf (and the libalf website) are still in development. In particular, we are currently hard working on a comprehensive documentation.

Please come back later or subscribe to our feed to stay informed.

libalf demo

Launch the libalf demo
(Requires the Java Web Start technology)
http://libalf.informatik.rwth-aachen.de/

**Challenge 1**

Find a sample such that RPNI does not produce a minimal consistent DFA
2. Active Automata Learning
Angluin’s Active Learning Setting

Learning algorithm

Teacher

Is \( ab \in L \)?

Is \( aab \in L \)?

Is \( abu \in L \) \( u \not\in L \) \( ab \in L \) \( u \not\in L \) \( ab \in L \) \( u \not\in L \) \( ab \in L \) \( u \not\in L \) 

▶ Membership query

▶ Equivalence query

Daniel Neider: A Tutorial on Automata Learning
Angluin’s Active Learning Setting

Learning algorithm

Is $ab \in L$?

Teacher

▶ Membership query $u \in L$?
Angluin’s Active Learning Setting

Learning algorithm

Teacher

▶ Membership query $u \in L$?
Angluin’s Active Learning Setting

Learning algorithm

Teacher

Membership query $u \in L$?
Angluin’s Active Learning Setting

Learning algorithm

Teacher

- Membership query $u \in L$?
- Equivalence query $L(\mathcal{A}) = L$?
Angluin’s Active Learning Setting

Teacher: No, counter-example $u = aa$!

Learning algorithm

- Membership query $u \in L$?
- Equivalence query $L(A) = L$?
Angluin’s Active Learning Setting

Learning algorithm

- Membership query $u \in L$?
- Equivalence query $L(\mathcal{A}) = L$?

Teacher

$u \in L$ $u \not\in L$

$ab$ $aa$
Angluin’s Active Learning Setting

Learning algorithm

- Membership query $u \in L$?
- Equivalence query $L(A) = L$?

Teacher

$\text{Is } aaa \in L$?
Angluin’s Active Learning Setting

Learning algorithm

Teacher

- Membership query $u \in L$?
- Equivalence query $L(A) = L$?

No!
Angluin’s Active Learning Setting

<table>
<thead>
<tr>
<th>$u \in L$</th>
<th>$u \notin L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ab$</td>
<td>$aa$</td>
</tr>
<tr>
<td>$aaa$</td>
<td>$____$</td>
</tr>
</tbody>
</table>

Learning algorithm

Teacher

- Membership query $u \in L$?
- Equivalence query $L(\mathcal{A}) = L$?
Angluin’s Active Learning Setting

Learning algorithm

Teacher

- Membership query $u \in L$?
- Equivalence query $L(\mathcal{A}) = L$?

Daniel Neider: A Tutorial on Automata Learning
Angluin’s Active Learning Setting

Learning algorithm

- Membership query \( u \in L \) ?
- Equivalence query \( L(A) = L \) ?

Teacher

Yes!
Algorithm 3: Naïve active learning algorithm

Input: A teacher for a regular target language $L \subseteq \Sigma$

1. Fix a total order $\prec$ on DFAs (e.g., canonical order of their string-representation)

2. $i \leftarrow 0$

3. repeat

4. Let $A_i$ be the $i$-th DFA in the order $\prec$

5. Pose an equivalence query on $A_i$

6. until the teacher replies “yes” on an equivalence query

7. return $A_i$
Algorithm 3: Naïve active learning algorithm

Input: A teacher for a regular target language \( L \subseteq \Sigma \)

1. Fix a total order \( \prec \) on DFAs (e.g., canonical order of their string-representation)
2. \( i \leftarrow 0 \)
3. repeat
   4. Let \( A_i \) be the \( i \)-th DFA in the order \( \prec \)
   5. Pose an equivalence query on \( A_i \)
4. until the teacher replies “yes” on an equivalence query
5. return \( A_i \)

▶ Equivalence queries without counterexamples already suffice (but exponentially many queries are necessary in general)
Algorithm 4: Iterative Passive Learning

Input: A teacher for a regular target language $L \subseteq \Sigma$

1. Initialize an empty sample $S = (S_+, S_-)$ with $S_+ = S_- = \emptyset$

2. repeat
   3. Compute a (minimal) DFA $A_S$ that is consistent with $S$
   4. Pose an equivalence query on $A$
   5. if the teacher replies with a counterexample $u$ then
      6. Add $u$ to $S$
   end

8. until the teacher replies “yes” on an equivalence query

9. return $A_S$
Theorem

Iterative Passive Learning of minimal consistent DFAs is guaranteed to produce the minimal DFA accepting the target language $L$. 

Proof (sketch)

Let $n$ be the size of the minimal DFA accepting $L$. Let $A_0, A_1, \ldots$ be the sequence of DFAs produced during the learning process. Let $|A_i| \leq |A_{i+1}|$ for all $i \in \mathbb{N}$. Let $L(A_i) \neq L(A_j)$ for all $i \in \mathbb{N}$ and $j < i$. Thus, the minimal DFA accepting $L$ is eventually produced.
Theorem

Iterative Passive Learning of minimal consistent DFAs is guaranteed to produce the minimal DFA accepting the target language $L$

Proof (sketch)

- Let $n$ be the size of the minimal DFA accepting $L$
Theorem

Iterative Passive Learning of minimal consistent DFAs is guaranteed to produce the minimal DFA accepting the target language $L$.

Proof (sketch)

- Let $n$ be the size of the minimal DFA accepting $L$.
- Let $A_0, A_1, \ldots$ be the sequence of DFA produced during the learning process.
  - $|A_i| \leq |A_{i+1}|$ for all $i \in \mathbb{N}$
  - $L(A_i) \neq L(A_j)$ for all $i \in \mathbb{N}$ and $j < i$
Theorem

Iterative Passive Learning of minimal consistent DFAs is guaranteed to produce the minimal DFA accepting the target language $L$.

Proof (sketch)

- Let $n$ be the size of the minimal DFA accepting $L$.
- Let $A_0, A_1, \ldots$ be the sequence of DFA produced during the learning process.
  - $|A_i| \leq |A_{i+1}|$ for all $i \in \mathbb{N}$.
  - $L(A_i) \neq L(A_j)$ for all $i \in \mathbb{N}$ and $j < i$.
- Thus, the minimal DFA accepting $L$ is eventually produced.
Theorem (Angluin [7], 1990)

“There is no polynomial time algorithm using only equivalence queries that exactly identifies deterministic or nondeterministic finite state acceptors [...]”
Theorem (Angluin [7], 1990)

“There is no polynomial time algorithm using only equivalence queries that exactly identifies deterministic or nondeterministic finite state acceptors [...]”

Theorem (Angluin [8], 1981)

There is no polynomial time learning algorithm for DFAs that only uses membership queries
Theorem (Angluin [7], 1990)

“There is no polynomial time algorithm using only equivalence queries that exactly identifies deterministic or nondeterministic finite state acceptors [...].”

Theorem (Angluin [8], 1981)

There is no polynomial time learning algorithm for DFAs that only uses membership queries.

Two Polynomial Time Active Learning Algorithms

1. Angluin’s Algorithm
2. Kearns and Vazirani’s algorithm
2.1. Angluin’s Algorithm (L*)
Let $L \subseteq \Sigma^*$ be a language.

- Two words $u, v \in \Sigma^*$ are \textit{L-equivalent}, denoted by $u \sim_L v$, if

$$uw \in L \iff vw \in L$$

holds for all $w \in \Sigma^*$
Let \( L \subseteq \Sigma^* \) be a language.

- Two words \( u, v \in \Sigma^* \) are \( L \)-equivalent, denoted by \( u \sim_L v \), if

\[
uw \in L \iff vw \in L
\]

holds for all \( w \in \Sigma^* \)

- \( \sim_L \) is a congruence with respect to concatenation, meaning

\[
u \sim_L v \text{ implies } ua \sim_L va \text{ for all } a \in \Sigma
\]
Let $L \subseteq \Sigma^*$ be a language.

- Two words $u, v \in \Sigma^*$ are $L$-equivalent, denoted by $u \sim_L v$, if
  \[
  uw \in L \iff vw \in L
  \]
  holds for all $w \in \Sigma^*$

- $\sim_L$ is a congruence with respect to concatenation, meaning
  \[
  u \sim_L v \text{ implies } ua \sim_L va \text{ for all } a \in \Sigma
  \]

- The $L$-equivalence class of $u \in \Sigma^*$ is defined by
  \[
  \llbracket u \rrbracket_L = \{ v \in \Sigma^* \mid u \sim_L v \}
  \]
Let $L \subseteq \Sigma^*$ be a language.

▶ Two words $u, v \in \Sigma^*$ are \textit{L-equivalent}, denoted by $u \sim_L v$, if

$$uw \in L \iff vw \in L$$

holds for all $w \in \Sigma^*$

▶ $\sim_L$ is a \textit{congruence} with respect to concatenation, meaning

$$u \sim_L v \text{ implies } ua \sim_L va \text{ for all } a \in \Sigma$$

▶ The \textit{L-equivalence class} of $u \in \Sigma^*$ is defined by

$$[u]_L = \{v \in \Sigma^* \mid u \sim_L v\}$$

▶ The number of equivalence classes of $L$ is defined by

$$\text{index}(\sim_L) = |\{[u]_L \mid u \in \Sigma^*\}|$$
### Hankel Matrix

![Automaton Diagram](image)

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$b$</th>
<th>$aa$</th>
<th>$ab$</th>
<th>$ba$</th>
<th>$bb$</th>
<th>$aaa$</th>
<th>$aab$</th>
<th>$\cdots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$a$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$aa$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$ab$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$ba$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$bb$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
</tbody>
</table>
### Hankel Matrix

![Automaton Diagram](image)

<table>
<thead>
<tr>
<th></th>
<th>ε</th>
<th>a</th>
<th>b</th>
<th>aa</th>
<th>ab</th>
<th>ba</th>
<th>bb</th>
<th>aaa</th>
<th>aab</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>aa</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>ab</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>ba</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>bb</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
</tbody>
</table>

...
Theorem (Myhill and Nerode)

- A language $L$ is regular if and only if $\text{index}(\sim_L)$ is finite.
- For every regular language, there exists a (up to isomorphism) unique minimal DFA $A_L$ that recognizes $L$. 
Theorem (Myhill and Nerode)

- A language $L$ is regular if and only if $\text{index}(\sim_L)$ is finite.
- For every regular language, there exists a (up to isomorphism) unique minimal DFA $A_L$ that recognizes $L$.

Proof (sketch)

The DFA $A_L = (Q, \Sigma, q_0, \delta, F)$ where

- $Q = \{[u]_L \mid u \in \Sigma^*\}$;
- $q_0 = [\varepsilon]_L$;
- $\delta([u]_L, a) = [ua]_L$; and
- $F = \{[u]_L \mid u \in L\}$

recognizes $L$ and is minimal.
An observation table is a triple $O = (R, S, T)$ where

### Example

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$ba$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$a$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$aa$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$ab$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
An observation table is a triple $O = (R, S, T)$ where

- $R$ is a nonempty, finite, prefix-closed set of representatives
- $S$ is a nonempty, finite set of separating words containing $\varepsilon$
- $T: (R \cup R \cdot \Sigma) \cdot S \rightarrow \{0, 1\}$ stores the table entries

**Example**

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$ba$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$a$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$aa$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$ab$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
An observation table is a triple $O = (R, S, T)$ where

- $R$ is a nonempty, finite, prefix-closed set of representatives
- $S$ is a nonempty, finite set of separating words containing $\varepsilon$

**Example**

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$ba$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$a$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$aa$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$ab$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
An observation table is a triple \( O = (R, S, T) \) where

- \( R \) is a nonempty, finite, prefix-closed set of representatives
- \( S \) is a nonempty, finite set of separating words containing \( \varepsilon \)
- \( T : (R \cup R \cdot \Sigma) \cdot S \rightarrow \{0, 1\} \) stores the table entries

**Example**

<table>
<thead>
<tr>
<th></th>
<th>( \varepsilon )</th>
<th>a</th>
<th>ba</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varepsilon )</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( a )</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( b )</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( aa )</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( ab )</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
An observation table is a triple $O = (R, S, T)$ where

- $R$ is a nonempty, finite, prefix-closed set of representatives
- $S$ is a nonempty, finite set of separating words containing $\varepsilon$
- $T : (R \cup R \cdot \Sigma) \cdot S \rightarrow \{0, 1\}$ stores the table entries

- Two words $u, v \in R \cup R \cdot \Sigma$ are $O$-equivalent, denoted by $u \sim_O v$, if
  \[ T(uw) = T(vw) \]
  holds for all $w \in S$
An observation table is a triple $O = (R, S, T)$ where

- $R$ is a nonempty, finite, prefix-closed set of representatives
- $S$ is a nonempty, finite set of separating words containing $\varepsilon$
- $T : (R \cup R \cdot \Sigma) \cdot S \to \{0, 1\}$ stores the table entries

- Two words $u, v \in R \cup R \cdot \Sigma$ are $O$-equivalent, denoted by $u \sim_O v$, if
  $$T(uw) = T(vw)$$
  holds for all $w \in S$

- The $O$-equivalence class of $u \in R \cup R \cdot \Sigma$ is defined by
  $$[u]_O = \{ v \in R \cup R \cdot \Sigma \mid u \sim_O v \}$$
An observation table \( O = (R, S, T) \) is \textit{closed} if for all \( u \in R \) and \( a \in \Sigma \), there exists a \( v \in R \) such that \( ua \sim_O v \).
An observation table $O = (R, S, T)$ is \textit{closed} if for all $u \in R$ and $a \in \Sigma$, there exists a $v \in R$ such that $ua \sim_O v$.

### Example

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$ba$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$a$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$aa$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$ab$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
An observation table $O = (R, S, T)$ is *closed* if for all $u \in R$ and $a \in \Sigma$, there exists a $v \in R$ such that $ua \sim_O v$

- If $O$ is not closed, add $ua$ to $R$

**Example**

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$ba$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$a$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$aa$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$ab$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$ba$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$a$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$aa$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$ab$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$ba$</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$bb$</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An observation table \( O = (R, S, T) \) is consistent if for all \( u, v \in R \) and \( a \in \Sigma \), \( u \sim_O v \) implies \( ua \sim_O va \).
An observation table $O = (R, S, T)$ is consistent if for all $u, v \in R$ and $a \in \Sigma$, $u \sim_O v$ implies $ua \sim_O va$.

**Example**

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$ba$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$a$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$aa$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$ab$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
An observation table \( O = (R, S, T) \) is \textit{consistent} if for all \( u, v \in R \) and \( a \in \Sigma \), \( u \sim_O v \) implies \( ua \sim_O va \)

▶ If \( O \) is \textit{not consistent}, there exists a \( w \in S \) such that \( T(uaw) \neq T(vaw) \); add \( aw \) to \( S \)

Example

\[
\begin{array}{ccc}
\varepsilon & a & ba \\
\hline
\varepsilon & 1 & 0 & 1 \\
b & 1 & 0 & 1 \\
\hline
a & 0 & 1 & 0 \\
ba & 0 & 0 & 0 \\
bb & 0 & 1 & 1 \\
\end{array}
\]
An observation table $O = (R, S, T)$ is **consistent** if for all $u, v \in R$ and $a \in \Sigma$, $u \sim_O v$ implies $ua \sim_O va$.

If $O$ is **not consistent**, there exists a $w \in S$ such that $T(uaw) \neq T(vaw)$; add $aw$ to $S$.

### Example

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$ba$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$a$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$ba$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$bb$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$ba$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$a$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$ba$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$bb$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
If an observation table $O$ is closed and consistent, then $\sim_O$ is a congruence and one can define a DFA $A_O = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = \{\llbracket u \rrbracket_O \mid u \in R\}$
- $q_0 = \llbracket \varepsilon \rrbracket_O$
- $\delta(\llbracket u \rrbracket_O, a) = \llbracket ua \rrbracket_O$
- $F = \{\llbracket u \rrbracket_O \mid u \in R, T(u) = 1\}$
If an observation table $O$ is closed and consistent, then $\sim_O$ is a congruence and one can define a DFA $A_O = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = \{ [u]_O \mid u \in R \}$
- $q_0 = [\varepsilon]_O$
- $\delta([u]_O, a) = [ua]_O$
- $F = \{ [u]_O \mid u \in R, T(u) = 1 \}$

**Example (Construction)**

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$a$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$aa$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$ab$</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
If an observation table $O$ is closed and consistent, then $\sim_O$ is a congruence and one can define a DFA $A_O = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = \{[[u]]_O \mid u \in R\}$
- $q_0 = [[\varepsilon]]_O$
- $\delta([[u]]_O, a) = [[ua]]_O$
- $F = \{[[u]]_O \mid u \in R, T(u) = 1\}$

**Example (Construction)**

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$a$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$aa$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$ab$</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
If an observation table $O$ is closed and consistent, then $\sim_O$ is a congruence and one can define a DFA $A_O = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = \{ [[u]]_O \mid u \in R \}$
- $q_0 = [[\varepsilon]]_O$
- $\delta([[u]]_O, a) = [[ua]]_O$
- $F = \{ [[u]]_O \mid u \in R, T(u) = 1 \}$

Example (Construction)

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$a$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$aa$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$ab$</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

$\rightarrow [[\varepsilon]]$ $\rightarrow [[a]]$
If an observation table $O$ is closed and consistent, then $\sim_O$ is a congruence and one can define a DFA $A_O = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = \{[u]_O \mid u \in R\}$
- $q_0 = [\varepsilon]_O$
- $\delta([u]_O, a) = [ua]_O$
- $F = \{[u]_O \mid u \in R, T(u) = 1\}$

Example (Construction)

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$a$</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$aa$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$ab$</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
If an observation table $O$ is closed and consistent, then $\sim_O$ is a congruence and one can define a DFA $A_O = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = \{[u]_O \mid u \in R\}$
- $q_0 = [\varepsilon]_O$
- $\delta([u]_O, a) = [ua]_O$
- $F = \{[u]_O \mid u \in R, T(u) = 1\}$

**Example (Construction)**

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$a$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$aa$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$ab$</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ \rightarrow [\varepsilon] \quad \xrightarrow{a, b} \quad [a] \]

Daniel Neider: A Tutorial on Automata Learning
If an observation table \( O \) is closed and consistent, then \( \sim_O \) is a congruence and one can define a DFA \( A_O = (Q, \Sigma, q_0, \delta, F) \) by

- \( Q = \{ [u]_O \mid u \in R \} \)
- \( q_0 = [\varepsilon]_O \)
- \( \delta([u]_O, a) = [ua]_O \)
- \( F = \{ [u]_O \mid u \in R, T(u) = 1 \} \)

**Lemma (Angluin [9], 1987)**

Let \( O = (R, S, T) \) be a closed and consistent observation table for a regular language \( L \subseteq \Sigma^* \) and \( A_O \) as defined above. Then, \( A_O \) is correct on all representatives \( u \in R \) (i.e., \( A_O \) satisfies \( u \in L(A_O) \) if and only if \( u \in L \) for each \( u \in R \))

Algorithm 5: Angluin’s Algorithm (L*)

1. Create an observation table $O = (R, S, T)$ with $R = S = \{\varepsilon\}$
2. repeat
3. Make $O$ closed and consistent (using membership queries)
4. Construct $A_O$ and perform an equivalence query
5. if the teacher returns a counterexample $u$ then
6. \[ R \leftarrow R \cup \text{Pref}(u) \]
7. update $O$ (using membership queries)
8. end
9. until the teacher replies “yes” on an equivalence query
10. return $A_O$
\[
\begin{array}{c|c}
\varepsilon & 0 \\
\varepsilon & 0 \\
 b & 1 \\
a & 0 \\
\end{array}
\]
Example

\[
\begin{array}{c|c|c|c|c|c|}
& \varepsilon & \varepsilon & b & a & a, b \\
\hline
\varepsilon & 0 & 0 & 1 & 0 & 0 \\
\hline
\end{array}
\]
Example

\[
\begin{array}{c|c}
\varepsilon \\
\hline
\varepsilon & 0 \\
b & 1 \\
\hline
a & 0 \\
ba & 0 \\
bb & 0 \\
\end{array}
\]
Example

\[
\begin{array}{c|c}
 & \varepsilon \\
\hline 
\varepsilon & 0 \\
 b & 1 \\
\hline 
a & 0 \\
 ba & 0 \\
 bb & 0 \\
\end{array}
\]
Example

\[ a \quad b \quad a, b \]

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$ba$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$bb$</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Counterexample $bbb$
Example

\[
\begin{array}{c|c}
\varepsilon & 0 \\
\hline
\varepsilon & 0 \\
b & 1 \\
bb & 0 \\
bbb & 0 \\
a & 0 \\
ba & 0 \\
bb & 0 \\
bbba & 0 \\
bbba & 0 \\
\end{array}
\]
Daniel Neider: A Tutorial on Automata Learning
Example

\[\begin{array}{c|cc}
\varepsilon & b \\
\hline
\varepsilon & 0 & 1 \\
b & 1 & 0 \\
bb & 0 & 0 \\
bbb & 0 & 0 \\
\hline
a & 0 & 1 \\
ba & 0 & 1 \\
bba & 0 & 0 \\
bbba & 0 & 0 \\
bbbb & 0 & 0 \\
\end{array}\]
Lemma (Angluin [9], 1987)

Let $O = (R, S, T)$ be a closed and consistent observation table for a regular language $L \subseteq \Sigma^*$. If $\text{index}(\sim_O \cap R \times R) = \text{index}(\sim_L)$, then $A_O$ is isomorphic to $A_L$. 
**Lemma (Angluin [9], 1987)**

Let $O = (R, S, T)$ be a closed and consistent observation table for a regular language $L \subseteq \Sigma^*$. If $\text{index}(\sim_O \cap R \times R) = \text{index}(\sim_L)$, then $A_O$ is isomorphic to $A_L$.

**Theorem (Angluin [9], 1987)**

Given a teacher for a regular target language $L \subseteq \Sigma^*$, Angluin’s algorithm learns a DFA isomorphic to $A_L$ in time polynomial in the size $n$ of $A_L$ and the length $m$ of the longest counterexample returned by the teacher. It asks $O(n)$ equivalence queries and $O(mn^2)$ membership queries.
Kearns and Vazirani’s Algorithm
## Improvements of Angluin’s Algorithm

<table>
<thead>
<tr>
<th></th>
<th>ε</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>bb</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bbb</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ba</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>bba</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bbba</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bbbb</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
### Improvements of Angluin’s Algorithm

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$bb$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$bbb$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$a$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$ba$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$bba$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$bbba$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$bbbb$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
## Improvements of Angluin’s Algorithm

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$bb$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$bbb$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$a$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$ba$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$bba$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$bbba$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$bbbb$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

1. Only store one representative per $\sim_L$-equivalence class
   - Rivest and Schapire’s algorithm
### Improvements of Angluin’s Algorithm

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$bb$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$bba$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$bbaa$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$bbaa$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$bbaaa$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

This table shows the states and transitions of an automaton.

1. Only store one representative per $\sim_L$-equivalence class
   - Rivest and Schapire’s algorithm

---

**Daniel Neider: A Tutorial on Automata Learning**

53
### Improvements of Angluin’s Algorithm

<table>
<thead>
<tr>
<th></th>
<th>(\varepsilon)</th>
<th>(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\varepsilon)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(b)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(bb)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(bbb)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(a)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(ba)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(bba)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(bbba)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(bbbb)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

1. Only store one representative per \(\sim_L\)-equivalence class
   - Rivest and Schapire’s algorithm

2. Use a compressed data structure
   - Kearns and Vazirani’s algorithm
A *classification tree* \( t \) is a binary tree

- whose inner nodes are labeled with *separating words* from a set \( S \)
- whose leaf nodes are labeled with *representatives* from a set \( R \)
- whose root and one leaf is labeled with \( \varepsilon \) (i.e., \( \varepsilon \in S \) and \( \varepsilon \in R \))
A *classification tree* \( t \) is a binary tree

- whose inner nodes are labeled with *separating words* from a set \( S \)
- whose leaf nodes are labeled with *representatives* from a set \( R \)
- whose root and one leaf is labeled with \( \varepsilon \) (i.e., \( \varepsilon \in S \) and \( \varepsilon \in R \))

We will maintain the following invariant for \( u \in R \) and \( v \in S \):

- the *left* subtree only contains representatives with \( uv \notin L \)
- the *right* subtree only contains representatives with \( uv \in L \)
Let $t$ be a classification tree and $u \in \Sigma^*$ a word.

**Sifting**

Starting at the root, recursively descent at an inner node labeled with $v \in S$

- *left* if $uv \notin L$ or
- *right* if $uv \in L$
Let $t$ be a classification tree and $u \in \Sigma^*$ a word.

**Sifting**

Starting at the root, recursively descent at an inner node labeled with $v \in S$

- *left* if $uv \notin L$ or
- *right* if $uv \in L$

**Example**

Let $u = ba$
Let $t$ be a classification tree and $u \in \Sigma^*$ a word.

**Sifting**

Starting at the root, recursively descent at an inner node labeled with $v \in S$

- **left** if $uv \notin L$ or
- **right** if $uv \in L$

**Example**

Let $u = ba$

---

In the diagram, the left and right branches of the tree represent the recursion paths based on the classification rules. The transition from the initial state to the final state is illustrated through the sequence $ba$. The automaton transitions through states labeled $a$, $b$, and $a, b$ according to the input $u$.
Let $t$ be a classification tree and $u \in \Sigma^*$ a word.

**Sifting**

Starting at the root, recursively descend at an inner node labeled with $v \in S$

- *left* if $uv \notin L$ or
- *right* if $uv \in L$

**Example**

Let $u = ba$

![Diagram of a finite automaton and a classification tree]
Let $t$ be a classification tree and $u \in \Sigma^*$ a word.

**Sifting**

Starting at the root, recursively descent at an inner node labeled with $v \in S$

- *left* if $uv \notin L$ or
- *right* if $uv \in L$

**Example**

Let $u = ba$

![Diagram showing the sifting process with a classification tree and a word path example]
Let $t$ be a classification tree and $u \in \Sigma^*$ a word.

**Sifting**

Starting at the root, recursively descent at an inner node labeled with $v \in S$

- *left* if $uv \notin L$ or
- *right* if $uv \in L$

**Example**

Let $u = ba$
Let $t$ be a classification tree and $u \in \Sigma^*$ a word.

**Sifting**

Starting at the root, recursively descent at an inner node labeled with $v \in S$

- *left* if $uv \notin L$
- *right* if $uv \in L$

**Example**

Let $u = ba$

- $\epsilon$
- $b$
- $ba$
- $bb$

\[ a \rightarrow b \rightarrow a, b \rightarrow b \rightarrow a \]
Let \( t \) be a classification tree and \( u \in \Sigma^* \) a word.

**Sifting**

Starting at the root, recursively descent at an inner node labeled with \( v \in S \)

- *left* if \( uv \notin L \) or
- *right* if \( uv \in L \)

**Sifting Operator**

Let \( \text{sift}_t(u) \) denote the representative at the leaf node reached by sifting \( u \) down \( t \)

**Note:** Sifting invokes membership queries
Given a classification tree $t$, we define the DFA $A_t = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = R$
- $q_0 = \varepsilon$
- $\delta(u, a) = \text{sift}_t(ua)$
- $F = \{u \in R \mid u \in L\}$ (i.e., all representatives in the right subtree)
Given a classification tree $t$, we define the DFA $A_t = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = R$
- $q_0 = \varepsilon$
- $\delta(u, a) = sift_t(ua)$
- $F = \{ u \in R \mid u \in L \}$ (i.e., all representatives in the right subtree)

**Example (Construction)**

```
\varepsilon
  \ /
 b /  \ b
 /    /    
bb \   \   \  
  \   \   \  
   \   \   \  
    \   \   \  
     \   \   \  
      \   \   \  
       \   \   \  
        \   \   \  
         \   \   \  
          \   \   \  
           \   \   \  
            \   \   \  
             \   \   \  
              \   \   \  
               \   \   \  
                \   \   \  
                 \   \   \  
                  \   \   \  
                   \   \   \  
                    \   \   \  
                     \   \   \  
                      \   \   \  
                       \   \   \  
                        \   \   \  
                         \   \   \  
                          \   \   \  
                           \   \   \  
                            \   \   \  
                             \   \   \  
                              \   \   \  
                               \   \   \  
                                \   \   \  
                                 \   \   \  
                                  \   \   \  
                                   \   \   \  
                                    \   \   \  
                                     \   \   \  
                                      \   \   \  
                                       \   \   \  
                                        \   \   \  
                                         \   \   \  
                                          \   \   \  
                                           \   \   \  
                                            \   \   \  
                                             \   \   \  
                                              \   \   \  
                                               \   \   \  
                                                \   \   \  
                                                 \   \   \  
                                                  \   \   \  
                                                   \   \   \  
                                                    \   \   \  
                                                     \   \   \  
                                                      \   \   \  
                                                       \   \   \  
                                                        \   \   \  
                                                         \   \   \  
                                                          \   \   \  
                                                           \   \   \  
                                                            \   \   \  
                                                             \   \   \  
                                                              \   \   \  
                                                               \   \   \  
                                                                \   \   \  
                                                                 \   \   \  
                                                                  \   \   \  
                                                                   \   \   \  
                                                                    \   \   \  
                                                                     \   \   \  
                                                                                       
```
Given a classification tree $t$, we define the DFA $A_t = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = R$
- $q_0 = \varepsilon$
- $\delta(u, a) = \text{sift}_t(ua)$
- $F = \{u \in R \mid u \in L\}$ (i.e., all representatives in the right subtree)

**Example (Construction)**

```
  \[ \begin{array}{ccc}
      & \varepsilon & \\
  b & & b \\
  \varepsilon & & \varepsilon \\
  bb & b & b
  \end{array} \]```

\[ \varepsilon \quad b \quad bb \]
Given a classification tree $t$, we define the DFA $A_t = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = R$
- $q_0 = \varepsilon$
- $\delta(u, a) = sift_t(ua)$
- $F = \{u \in R \mid u \in L\}$ (i.e., all representatives in the right subtree)

**Example (Construction)**
Given a classification tree $t$, we define the DFA $A_t = (Q, \Sigma, q_0, \delta, F)$ by

$\begin{align*}
Q &= R \\
q_0 &= \varepsilon \\
\delta(u, a) &= \text{sift}_t(ua) \\
F &= \{u \in R \mid u \in L\} \text{ (i.e., all representatives in the right subtree)}
\end{align*}$

Example (Construction)
Given a classification tree $t$, we define the DFA $A_t = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = R$
- $q_0 = \varepsilon$
- $\delta(u, a) = \text{sift}_t(ua)$
- $F = \{u \in R \mid u \in L\}$ (i.e., all representatives in the right subtree)

Example (Construction)
Given a classification tree \( t \), we define the DFA \( A_t = (Q, \Sigma, q_0, \delta, F) \) by

\begin{itemize}
  \item \( Q = R \)
  \item \( q_0 = \varepsilon \)
  \item \( \delta(u, a) = \text{sift}_t(ua) \)
  \item \( F = \{ u \in R \mid u \in L \} \) (i.e., all representatives in the right subtree)
\end{itemize}

**Example (Construction)**

[Diagram of a classification tree and its corresponding DFA]
Given a classification tree $t$, we define the DFA $A_t = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = R$
- $q_0 = \varepsilon$
- $\delta(u, a) = sift_t(ua)$
- $F = \{u \in R \mid u \in L\}$ (i.e., all representatives in the right subtree)

**Example (Construction)**
Given a classification tree $t$, we define the DFA $A_t = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = R$
- $q_0 = \varepsilon$
- $\delta(u, a) = \text{sift}_t(ua)$
- $F = \{u \in R \mid u \in L\}$ (i.e., all representatives in the right subtree)

**Example (Construction)**
Given a classification tree $t$, we define the DFA $\mathcal{A}_t = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = R$
- $q_0 = \varepsilon$
- $\delta(u, a) = \text{sift}_t(ua)$
- $F = \{u \in R \mid u \in L\}$ (i.e., all representatives in the right subtree)

**Example (Construction)**

```
  b
 / \  \
 \  \  \\
 bb ε ba
```

```
  ?
 / \
 \ \\
 b
```

```
  ε
 / \\
 b
```

```
 bb
```

```
 ba
```
Given a classification tree $t$, we define the DFA $A_t = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = R$
- $q_0 = \varepsilon$
- $\delta(u, a) = \text{sift}_t(ua)$
- $F = \{u \in R \mid u \in L\}$ (i.e., all representatives in the right subtree)

Example (Construction)
Given a classification tree $t$, we define the DFA $A_t = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = R$
- $q_0 = \varepsilon$
- $\delta(u, a) = \text{sift}_t(ua)$
- $F = \{u \in R \mid u \in L\}$ (i.e., all representatives in the right subtree)

### Example (Construction)

Daniel Neider: A Tutorial on Automata Learning
Given a classification tree $t$, we define the DFA $A_t = (Q, \Sigma, q_0, \delta, F)$ by

- $Q = R$
- $q_0 = \varepsilon$
- $\delta(u, a) = \text{sift}_t(ua)$
- $F = \{u \in R \mid u \in L\}$ (i.e., all representatives in the right subtree)

Example (Construction)
Let $v = a_1 \ldots a_m$ be a counterexample and

\[
A_t: u_0 \xrightarrow{a_1} u_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} u_m
\]

the run of $A_t$ on $v$
Let $v = a_1 \ldots a_m$ be a counterexample and
\[ A_t : u_0 \xrightarrow{a_1} u_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} u_m \]
the run of $A_t$ on $v$

**Breakpoint Position**

A *breakpoint position* is an index $i \in \{1, \ldots, m\}$ with
\[ u_{i-1}a_i \ldots a_m \in L \iff u_ia_{i+1} \ldots a_m \notin L \]
Analyzing Counterexamples

Let $v = a_1 \ldots a_m$ be a counterexample and

$$A_t: u_0 \xrightarrow{a_1} u_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} u_m$$

the run of $A_t$ on $v$

Breakpoint Position

A \textit{breakpoint position} is an index $i \in \{1, \ldots, m\}$ with

$$u_{i-1} a_i \ldots a_m \in L \iff u_i a_{i+1} \ldots a_m \notin L$$

![Diagram of Automaton $A_t$]
Let $v = a_1 \ldots a_m$ be a counterexample and

$$\mathcal{A}_t: u_0 \xrightarrow{a_1} u_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} u_m$$

the run of $\mathcal{A}_t$ on $v$

**Breakpoint Position**

A *breakpoint position* is an index $i \in \{1, \ldots, m\}$ with

$$u_{i-1}a_i \ldots a_m \in L \iff u_ia_{i+1} \ldots a_m \notin L$$
Analyzing Counterexamples

Let $v = a_1 \ldots a_m$ be a counterexample and

$$A_t: u_0 \xrightarrow{a_1} u_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} u_m$$

the run of $A_t$ on $v$

Breakpoint Position

A \textit{breakpoint position} is an index $i \in \{1, \ldots, m\}$ with

$$u_{i-1}a_i \ldots a_m \in L \Leftrightarrow u_ia_{i+1} \ldots a_m \notin L$$

A Breakpoint Position Always Exists

Since $v$ is a counterexample, we know

$$v \in L \Leftrightarrow v \notin L(A_t)$$
Let $v = a_1 \ldots a_m$ be a counterexample and

$$A_t: u_0 \xrightarrow{a_1} u_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} u_m$$

the run of $A_t$ on $v$

### Breakpoint Position

A *breakpoint position* is an index $i \in \{1, \ldots, m\}$ with

$$u_{i-1}a_i \ldots a_m \in L \iff u_ia_{i+1}\ldots a_m \notin L$$

### A Breakpoint Position Always Exists

Since $v$ is a counterexample, we know

$$v \in L \iff v \notin L(A_t) \iff u_m \notin F$$
Analyzing Counterexamples

Let \( v = a_1 \ldots a_m \) be a counterexample and

\[
A_t: u_0 \xrightarrow{a_1} u_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} u_m
\]

the run of \( A_t \) on \( v \)

Breakpoint Position

A **breakpoint position** is an index \( i \in \{1, \ldots, m\} \) with

\[
u_{i-1}a_i \ldots a_m \in L \iff u_i a_{i+1} \ldots a_m \notin L
\]

A Breakpoint Position Always Exists

Since \( v \) is a counterexample, we know

\[
v \in L \iff v \notin L(A_t) \iff u_m \notin F \iff u_m \notin L
\]
Analyzing Counterexamples

Let $v = a_1 \ldots a_m$ be a counterexample and

$$A_t: \ u_0 \xrightarrow{a_1} u_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} u_m$$

the run of $A_t$ on $v$

Breakpoint Position

A breakpoint position is an index $i \in \{1, \ldots, m\}$ with

$$u_{i-1}a_i \ldots a_m \in L \iff u_ia_{i+1} \ldots a_m \notin L$$

A Breakpoint Position Always Exists

Since $v$ is a counterexample, we know

$$v \in L \iff v \notin L(A_t) \iff u_m \notin F \iff u_m \notin L$$

Thus, $v = u_0a_1 \ldots a_m \in L \iff u_ma_{m+1} \ldots a_m = u_m \notin L$
Ask a membership query with $\varepsilon$
Initial Classification Tree

- Ask a membership query with $\varepsilon$
  - $\varepsilon \in L$
  - Ask an equivalence query with $\varepsilon$
  - $\varepsilon \not\in L$
  - $\varepsilon \in L$

Daniel Neider: A Tutorial on Automata Learning
Initial Classification Tree

- Ask a membership query with $\varepsilon$
- $\varepsilon \in L$
- Ask an equivalence query with $\Sigma$
- "yes"
- Return

Daniel Neider: A Tutorial on Automata Learning
Initial Classification Tree

Ask a membership query with $\varepsilon$

$\varepsilon \in L$

Ask an equivalence query with $\Sigma$

“yes”

Return

$\varepsilon \notin L$

$u

$\Sigma$

Daniel Neider: A Tutorial on Automata Learning
Initial Classification Tree

- Ask a membership query with $\varepsilon$
  - $\varepsilon \in L$
    - Ask an equivalence query with $\varepsilon$
      - “yes”
        - Return
    - $u \not\in L$
      - $\varepsilon \not\in L$
  - $\varepsilon \not\in L$
    - Ask an equivalence query with $\varepsilon$
      - $u \not\in L$
      - $\varepsilon$
        - $u$
          - $\varepsilon$

Daniel Neider: A Tutorial on Automata Learning
Initial Classification Tree

Ask a membership query with $\varepsilon$

- $\varepsilon \in L$
  - Ask an equivalence query with $\Sigma$
    - "yes"
      - Return $\Sigma$
    - $u \notin L$
      - $\varepsilon$
      - $u$
      - $\varepsilon$

- $\varepsilon \notin L$
  - Ask an equivalence query with $\Sigma$
    - "yes"
      - Return $\Sigma$

Initial Classification Tree

- **Ask a membership query with \( \varepsilon \)**
  - \( \varepsilon \in L \)
    - **Ask an equivalence query with \( \Sigma \)**
      - "yes"
        - **Return**
      - \( \varepsilon \in \Sigma \)
      - \( u \notin L \)
        - **Return**
    - \( \varepsilon \notin L \)
      - **Ask an equivalence query with \( \Sigma \)**
      - \( u \in L \)
        - "yes"
        - **Return**
      - \( \varepsilon \notin \Sigma \)
        - **Ask an equivalence query with \( \Sigma \)**
        - \( u \notin \Sigma \)
          - **Return**
        - \( \varepsilon \in \Sigma \)
          - \( u \in \Sigma \)
            - **Return**

Daniel Neider: A Tutorial on Automata Learning
Algorithm 6: Kearns and Vazirani’s Algorithm

1. Handle cases $L = \emptyset$ and $L = \Sigma^*$
2. Initialize the classification tree $t$
3. repeat
   4. Construct $A_t$ and ask an equivalence query
   5. if the teacher returns a counterexample $u$ then
      6. Identify a breakpoint position $i \in \{1, \ldots, m\}$
      7. $R \leftarrow R \cup \{u_{i-1}a_i\}$
      8. $S \leftarrow S \cup \{a_{i+1}\ldots a_m\}$
      9. Update $t$ by splitting the leaf node labeled with $u_i$
   end
4. until the teacher replies “yes” on an equivalence query
5. return $A_t$
1. Ask a membership query with $\varepsilon$
   - Answer: $\varepsilon \notin L$
1. Ask a membership query with $\varepsilon$
   - Answer: $\varepsilon \notin L$

2. Ask an equivalence query with $\rightarrow \quad a, b$
   - Counterexample $b \in L$
Example

Counterexample $bbb$
Example

![Automaton Diagram]

\[ A_t : \varepsilon \xrightarrow{b} b \xrightarrow{b} \varepsilon \xrightarrow{b} b \]

Thus, \( u_1 \not\sim L u_2 \), witnessed by \( b \).

\[ \text{Split } u_2 = \varepsilon, \text{ new leaf is } u_1 \text{ and new inner node is} \]

\[ \varepsilon \]
Example

\[ A_t: \varepsilon \xrightarrow{b} b \xrightarrow{b} \varepsilon \xrightarrow{b} b \]

\[ i = 1: \]

\[ u_0 bbb = bbb \notin L \text{ and } u_1 bb = bbb \notin L \]
Example

\[ A_t : \varepsilon \xrightarrow{b} b \xrightarrow{b} \varepsilon \xrightarrow{b} b \]

- \( i = 1 \):
  - \( u_0 bbb = bbb \notin L \) and \( u_1 bb = bbb \notin L \)
- \( i = 2 \):
  - \( u_1 bb = bbb \notin L \) and \( u_2 b = b \in L \)
\[
A_t : \varepsilon \xrightarrow{b} b \xrightarrow{b} \varepsilon \xrightarrow{b} b
\]

\begin{itemize}
  \item \( i = 1 \):
    \[ u_0 bbb = bbb \notin L \text{ and } u_1 bb = bbb \notin L \]
  \item \( i = 2 \):
    \[ u_1 bb = bbb \notin L \text{ and } u_2 b = b \in L \]
\end{itemize}

Thus, \( u_1 b \not\in_L u_2 \), witnessed by \( b \)

\begin{itemize}
  \item Split \( u_2 = \varepsilon \), new leaf is \( u_1 b = bb \) and new inner node is \( b \)
\end{itemize}
Example

\[ A_t : \varepsilon \xrightarrow{b} b \xrightarrow{b} \varepsilon \xrightarrow{b} b \]

- **\( i = 1 \):**
  
  \[ u_0 bbb = bbb \notin L \text{ and } u_1 bb = bbb \notin L \]

- **\( i = 2 \):**
  
  \[ u_1 bb = bbb \notin L \text{ and } u_2 b = b \in L \]

Thus, \( u_1 b \not\sim_L u_2 \), witnessed by \( b \)

- Split \( u_2 = \varepsilon \), new leaf is \( u_1 b = bb \) and new inner node is \( b \)
Example

Daniel Neider: A Tutorial on Automata Learning
Theorem (Kearns and Vazirani [10], 1994)

Given a teacher for a regular target language $L \subseteq \Sigma^*$, Kearns and Vazirani’s algorithm learns a DFA isomorphic to $A_L$ in time polynomial in the size $n$ of $A_L$ and the length $m$ of the longest counterexample returned by the teacher. It asks exactly $n$ equivalence queries and $O(n^2 + n \log_2 m)$ membership queries.
## Comparison of Active Learning Algorithms

<table>
<thead>
<tr>
<th>Learning algorithm</th>
<th>$MQ$</th>
<th>$EQ$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angluin’s algorithm</td>
<td>$\mathcal{O}(n^2 m)$</td>
<td>$\leq n$</td>
</tr>
<tr>
<td>Kearns and Vazirani’s algorithm</td>
<td>$\mathcal{O}(n^2 + n \log_2 m)$</td>
<td>$n$</td>
</tr>
<tr>
<td>Rivest and Schapire’s algorithm</td>
<td>$\mathcal{O}(n^2 + n \log_2 m)$</td>
<td>$\leq n$</td>
</tr>
</tbody>
</table>
Challenge 2

Find a family \((L_n)_{n \in \mathbb{N}}\) of regular languages such that

- \(A_{L_n}\) has at least \(n\) states and
- Angluin’s algorithm learns \(L_n\) with a constant number of equivalence queries

http://libalf.informatik.rwth-aachen.de/
Applications
Learning Meets Verification

Martin Leucker

Institut für Informatik
TU München, Germany

Abstract. In this paper, we give an overview on some algorithms for learning automata. Starting with Biermann’s and Angluin’s algorithms, we describe some of the extensions catering for specialized or richer classes of automata. Furthermore, we survey their recent application to verification problems.

1 Introduction

Recently, several verification problems have been addressed by using learning techniques. Given a system to verify, typically its essential part is learned and
Applications of Active Automata Learning


M. Leucker

Model and system do not conform

Incremental Learning (Angluin)

No counterexample

Check equivalence (VC algorithm)

Conformance established

report no error found

Conformance established

Model Checking wrt. current model

Counterexample found

Compare counterexample with system

Counterexample confirmed

Counterexample found

Countercexample confirmed

Counterexample refuted

Model and system do not conform

Fig. 3. Black Box Checking
An Automaton Learning Approach to Solving Safety Games over Infinite Graphs

Daniel Neider$^1$ and Ufuk Topcu$^2$ (✉)

$^1$ University of California, Los Angeles, USA
neider@ucla.edu

$^2$ The University of Texas at Austin, Austin, USA
utopcu@utexas.edu

Abstract. We propose a method to construct finite-state reactive controllers for systems whose interactions with their adversarial environment are modeled by infinite-duration two-player games over (possibly) infinite graphs.
Synthesis of Reactive Controllers

Plant + Specification

Infinite duration, two-player game over a graph

Strategy / Controller
Safety Games

Vertices $V_0$ of Player 0, vertices $V_1$ of Player 1

Edges $E$

Initial vertices $I$

Safe vertices $F$
Safety Games

Vertices $V_0$ of Player 0

Daniel Neider: A Tutorial on Automata Learning
Vertices $V_0$ of Player 0, vertices $V_1$ of Player 1
Safety Games

Vertices $V_0$ of Player 0, vertices $V_1$ of Player 1

Edges $E$
Safety Games

- Vertices $V_0$ of Player 0, vertices $V_1$ of Player 1
- Edges $E$
- Initial vertices $I$
- Vertices $V_0$ of Player 0, vertices $V_1$ of Player 1
- Edges $E$
- Initial vertices $I$
- Safe vertices $F$
- Vertices $V_0$ of Player 0, vertices $V_1$ of Player 1
- Edges $E$
- Initial vertices $I$
- Safe vertices $F$
Safety Games

- Vertices $V_0$ of Player 0, vertices $V_1$ of Player 1
- Edges $E$
- Initial vertices $I$
- Safe vertices $F$
Safety Games

- Vertices $V_0$ of Player 0, vertices $V_1$ of Player 1
- Edges $E$
- Initial vertices $I$
- Safe vertices $F$
Safety Games

- Vertices $V_0$ of Player 0, vertices $V_1$ of Player 1
- Edges $E$
- Initial vertices $I$
- Safe vertices $F$
Successively remove vertices from which a stay inside the safe vertices cannot be enforced
Successively remove vertices from which a stay inside the safe vertices cannot be enforced.
Successively remove vertices from which a stay inside the safe vertices cannot be enforced.
Successively remove vertices from which a stay inside the safe vertices cannot be enforced.
Successively remove vertices from which a stay inside the safe vertices cannot be enforced

Winning strategy for Player 0, winning strategy for Player 1
Safety Games Over Infinite Game Graphs
Safety Games Over Infinite Game Graphs
Safety Games Over Infinite Game Graphs

\[
\varepsilon \quad I \quad II \quad \boxed{III} \quad IIII \quad \ldots
\]
Safety Games Over Infinite Game Graphs

\[ V_0: \quad \begin{array}{c}
\varepsilon \\
\square \\
\II \\
\boxed{\III} \\
\III \\
\cdots
\end{array}
\]

\[ V_1: 
\begin{array}{c}
\varepsilon \\
\square \\
\II \\
\boxed{\III} \\
\III \\
\cdots
\end{array}
\]
Safety Games Over Infinite Game Graphs

\[ \varepsilon \rightarrow I \rightarrow \text{ll} \rightarrow \text{lll} \rightarrow \cdots \]

\[ I: \quad \rightarrow \quad \rightarrow \quad \rightarrow \quad \rightarrow \quad \rightarrow \]

\[ F: \quad \rightarrow \quad \rightarrow \quad \rightarrow \quad \rightarrow \quad \rightarrow \]

Daniel Neider: A Tutorial on Automata Learning
Safety Games Over Infinite Game Graphs

Model the edge II → III as word

\[
\begin{array}{c}
\_ \\
\_ \\
\_ \\
\end{array}
\]
Definition

A *rational safety games* is a five-tuple

\[ G = (A_{V_0}, A_{V_1}, A_I, A_F, T_E) \]

consisting of

- an NFA \( A_{V_0} \) encoding the vertices of Player 0
- an NFA \( A_{V_1} \) encoding the vertices of Player 1
- an NFA \( A_I \) encoding the initial vertices
- an NFA \( A_F \) encoding the safe vertices
- a rational transducer \( T_E \) encoding the edges

Assumption

Each vertex has only a finite number of successors

Daniel Neider: A Tutorial on Automata Learning
Rational Safety Games

Definition

A rational safety games is a five-tuple

\[ G = (A_{V_0}, A_{V_1}, A_I, A_F, T_E) \]

consisting of

- an NFA $A_{V_0}$ encoding the vertices of Player 0
- an NFA $A_{V_1}$ encoding the vertices of Player 1
- an NFA $A_I$ encoding the initial vertices
- an NFA $A_F$ encoding the safe vertices
- a rational transducer $T_E$ encoding the edges

Assumption

Each vertex has only a finite number of successors
Winning Sets

A (regular) winning set of vertices is a winning set if it satisfies:

1. $I \subseteq W$
2. $W \subseteq F$
3. If $v \in W \setminus V_0$, then there exists $(v, v') \in E$ such that $v' \in W$
4. If $v \in W \setminus V_1$, then $(v, v') \in E$ implies $v' \in W$.

Daniel Neider: A Tutorial on Automata Learning
Winning Set

A (regular) $W$ of vertices is a winning set if it satisfies:

1. $I \subseteq W$
2. $W \subseteq F$
3. If $v \in W \cap V_0$, then there exists a $(v, v') \in E$ such that $v' \in W$
4. If $v \in W \cap V_1$, then $(v, v') \in E$ implies $v' \in W$
Iterative Passive Learning

Learner

Teacher

Conjecture DFA

Counterexample
Today: counterexample-guided inductive synthesis
Counterexample

Let $C$ be the conjectured DFA.

Positive counterexample: $v \in I \setminus L(C)$

Negative counterexample: $v \in L(C) \setminus F$

Existential implication: $(v, U)$ with $v \in V_0 \setminus L(C)$, $U = E(v)$, and $U \not\subseteq L(C)$

Universal implication: $(v, U)$ with $v \in V_1 \setminus L(C)$, $U = E(v)$, and $U \not\subseteq L(C)$
Counterexample

Let $C$ be the conjectured DFA

- Positive counterexample: $v \in I \setminus L(C)$
Counterexample

Let $C$ be the conjectured DFA

- Positive counterexample: $v \in I \setminus L(C)$
- Negative counterexample: $v \in L(C) \setminus F$
Counterexample

Let $C$ be the conjectured DFA

- **Positive counterexample:** $v \in I \setminus L(C)$
- **Negative counterexample:** $v \in L(C) \setminus F$
- **Existential implication:** $(v, U)$ with $v \in V_0 \cap L(C)$, $U = E(v)$, and $U \cap L(C) = \emptyset$

Daniel Neider: A Tutorial on Automata Learning
Counterexample

Let $C$ be the conjectured DFA

- Positive counterexample: $v \in I \setminus L(C)$
- Negative counterexample: $v \in L(C) \setminus F$
- Existential implication: $(v, U)$ with $v \in V_0 \cap L(C)$, $U = E(v)$, and $U \cap L(C) = \emptyset$
- Universal implication: $(v, U)$ with $v \in V_1 \cap L(C)$, $U = E(v)$, and $U \not\subseteq L(C)$
The learners maintain a sample $S = (S_+, S_-, I\exists, I\forall)$ to store counterexamples.

**Learning Task**

Given a sample $S$, construct a DFA $C$ that is consistent with $S$:

1. $S_+ \subseteq L(C)$
2. $S_- \cap L(C) = \emptyset$
3. $v \in L(C)$ implies $U \cap L(C) \neq \emptyset$ for each $(v, U) \in I\exists$
4. $v \in L(C)$ implies $U \subseteq L(C)$ for each $(v, U) \in I\forall$
The learners maintain a sample $S = (S_+, S_-, I_\exists, I_\forall)$ to store counterexamples.

**Learning Task**

Given a sample $S$, construct a DFA $C$ that is consistent with $S$:

1. $S_+ \subseteq L(C)$
2. $S_- \cap L(C) = \emptyset$
3. $v \in L(C)$ implies $U \cap L(C) \neq \emptyset$ for each $(v, U) \in I_\exists$
4. $v \in L(C)$ implies $U \subseteq L(C)$ for each $(v, U) \in I_\forall$

**Two Consistent Learners**

1. SAT Learner
2. RPNI Learner (requires number of successors to be finite)
Idea

Extend Heule and Verwer’s method

- Construct formula $\psi_n^S$ that is satisfiable if and only if a consistent DFA with $n$ states exists; increase $n$ until $\psi_n^S$ is satisfiable
Idea

Extend Heule and Verwer’s method

- Construct formula $\psi_n^S$ that is satisfiable if and only if a consistent DFA with $n$ states exists; increase $n$ until $\psi_n^S$ is satisfiable

Encoding Deterministic Automata

Fix a set of state $Q = \{1, \ldots, n\}$ and an alphabet $\Sigma$

- $d_{p,a,q}$ where $p, q \in Q$ and $a \in \Sigma$ encoding transitions
- $f_q$ where $q \in Q$ encoding final states
Idea

Extend Heule and Verwer’s method

- Construct formula $\psi_n^S$ that is satisfiable if and only if a consistent DFA with $n$ states exists; increase $n$ until $\psi_n^S$ is satisfiable

Encoding Deterministic Automata

Fix a set of state $Q = \{1, \ldots, n\}$ and an alphabet $\Sigma$

- $d_{p,a,q}$ where $p, q \in Q$ and $a \in \Sigma$ encoding transitions
- $f_q$ where $q \in Q$ encoding final states
- Formula $\psi_n^{DFA}$ enforcing deterministic automata:

$$\bigwedge_{p \in Q} \bigwedge_{a \in \Sigma} \bigwedge_{q \neq q' \in Q} \neg d_{p,a,q} \lor \neg d_{p,a,q'}$$

$$\bigwedge_{p \in Q} \bigwedge_{a \in \Sigma} \left( \bigvee_{q \in Q} d_{p,a,q} \right)$$
Use Boolean variables $x_{u,q}$ where $u \in \text{Pref}(S)$ and $q \in Q$

- Consistency with $S_+$ and $S_-$ as before
Use Boolean variables $x_{u,q}$ where $u \in Pref(S)$ and $q \in Q$

- Consistency with $S_+$ and $S_-$ as before

- Consistency with existential implications

$$\bigwedge_{(v, U) \in I} \bigwedge_{p \in Q} (x_{u,p} \land f_p) \rightarrow \bigvee_{u \in U} \bigwedge_{q \in Q} (x_{u,q} \rightarrow f_q)$$
Use Boolean variables $x_{u,q}$ where $u \in \text{Pref}(S)$ and $q \in Q$

- Consistency with $S_+$ and $S_-$ as before

- Consistency with existential implications

$$\bigwedge_{(v,U) \in I_\exists} \bigwedge_{p \in Q} \left[ (x_{u,p} \land f_p) \rightarrow \bigvee_{u \in U} \bigwedge_{q \in Q} (x_{u,q} \rightarrow f_q) \right]$$

- Consistency with universal implications

$$\bigwedge_{(v,U) \in I_\forall} \bigwedge_{p \in Q} \left[ (x_{u,p} \land f_p) \rightarrow \bigwedge_{u \in U} \bigwedge_{q \in Q} (x_{u,q} \rightarrow f_q) \right]$$
The Max Planck Institute for Software Systems is offering opportunities:
- Internships
- PhD positions
- PostDoc positions

https://www.mpi-sws.org


Puzzle

Given a deterministic pushdown automaton $\mathcal{P}$ accepting a regular language, construct an equivalent DFA

Possible Solution

Use active learning:

- A teacher can be constructed as follows:
  - to answer a membership query with a word $u \in \Sigma^*$, simulate the PDA $\mathcal{P}$ on $u$
  - to answer an equivalence query with a DFA $\mathcal{A}$, check $L(\mathcal{A}) \cap (\Sigma^* \setminus L(\mathcal{P})) = \emptyset$ and $L(\mathcal{P}) \cap (\Sigma^* \setminus L(\mathcal{A})) = \emptyset$; return “yes“ if both check pass or a counterexample

- Use any active learning algorithm (e.g., Angluin’s algorithm)
Challenge 1

Find a sample such that RPNI does not produce a minimal consistent DFA

Possible Solution

Consider \( S = (S_+, S_-) \) with \( S_+ = \{abbab\} \) and \( S_- = \{\varepsilon, ab, ba\} \)

\[\begin{align*}
\text{RPNI:} & \quad \begin{array}{c}
\text{a} \\
\text{b} \\
\text{a}
\end{array} \\
\text{Minimal consistent DFA:} & \quad \begin{array}{c}
\text{a, b} \\
\text{a, b}
\end{array}
\end{align*}\]
Challenge 2

Find a family \((L_n)_{n \in \mathbb{N}}\) of regular languages such that

- \(A_{L_n}\) has at least \(n\) states and
- Angluin’s algorithm learns \(L_n\) with a constant number of equivalence queries

Possible Solution

For every \(n \in \mathbb{N}\), let \(L_n = \{a^n\}\)

- \(A_{L_n}\) has \(n + 2\) states and
- Angluin’s algorithm requires 3 equivalence queries