A Tutorial on Automata Learning

Daniel Neider

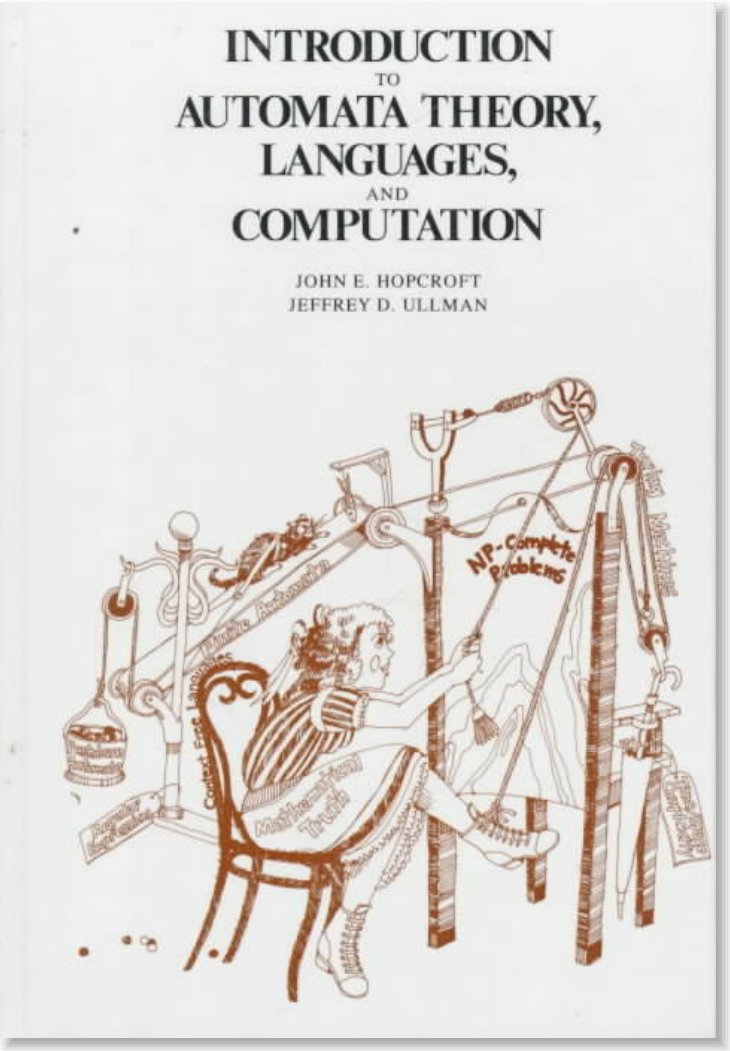
MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

MOVEP 2018

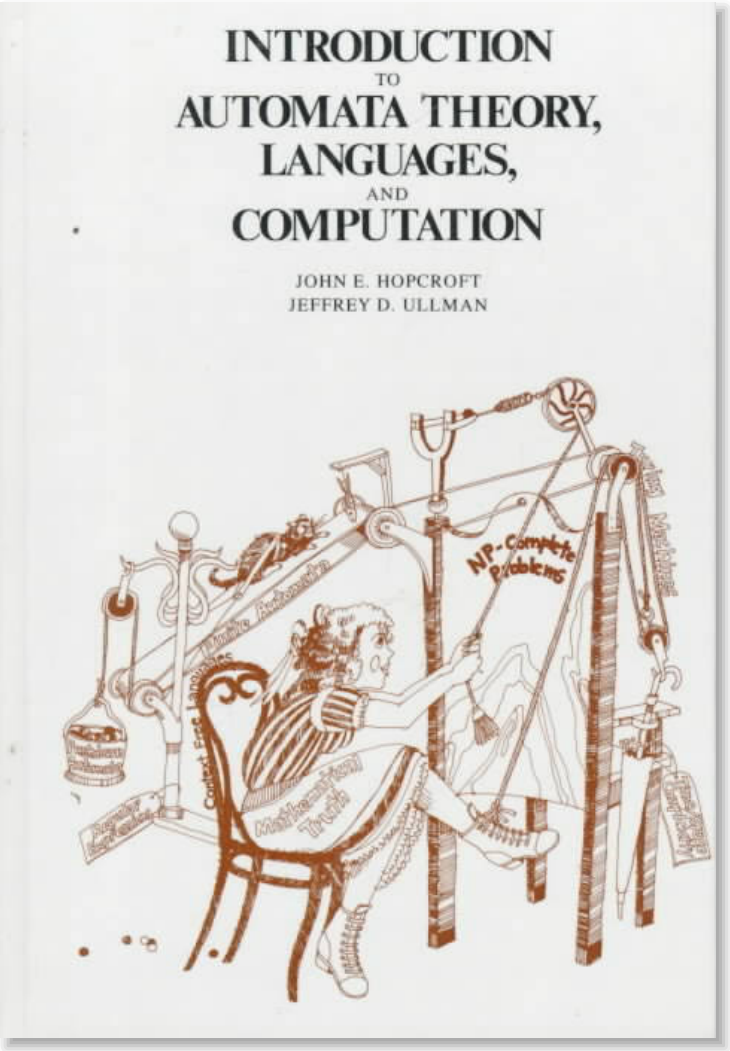
ENS Paris-Saclay, Cachan, France

20 July 2018

A Puzzle

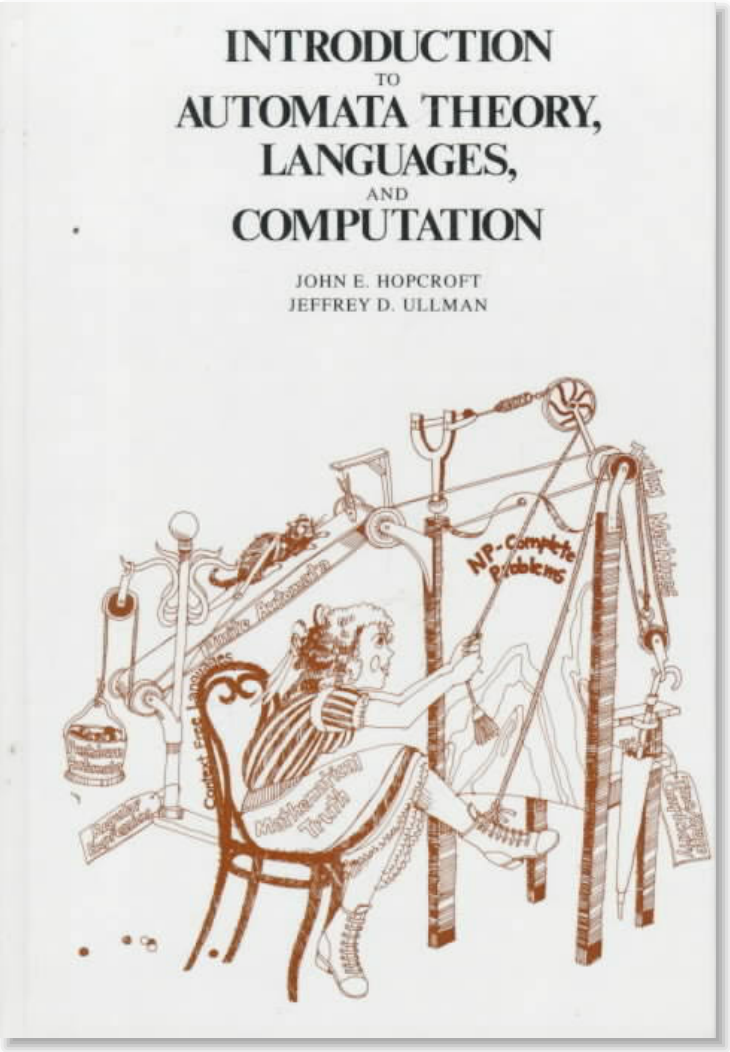

INTRODUCTION
TO
AUTOMATA THEORY,
LANGUAGES,
AND
COMPUTATION
JOHN E. HOPCROFT
JEFFREY D. ULLMAN
NP-Complete Problems

A Puzzle



- It is *undecidable* whether a pushdown automaton accepts a regular language

A Puzzle



- It is *undecidable* whether a pushdown automaton accepts a regular language
- It is *decidable* whether a deterministic pushdown automaton accepts a regular language

INFORMATION AND CONTROL 11, 323–340 (1967)

A Regularity Test for Pushdown Machines

R. E. STEARNS

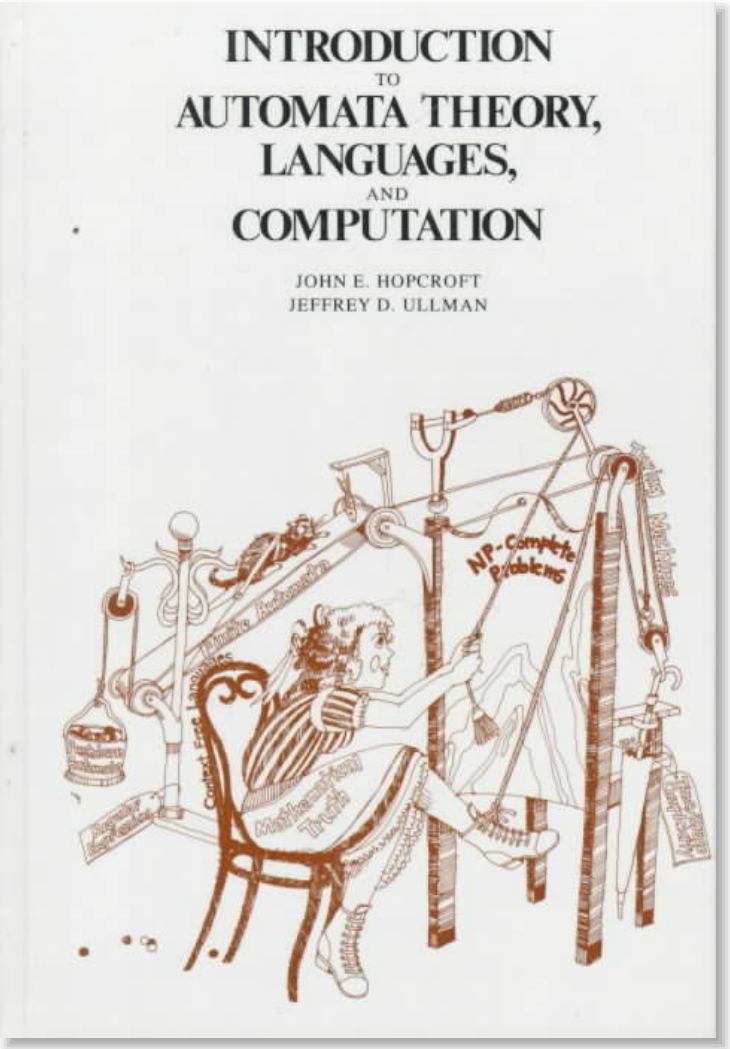
General Electric Research and Development Center, P.O. Box 8, Schenectady, New York 12301

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

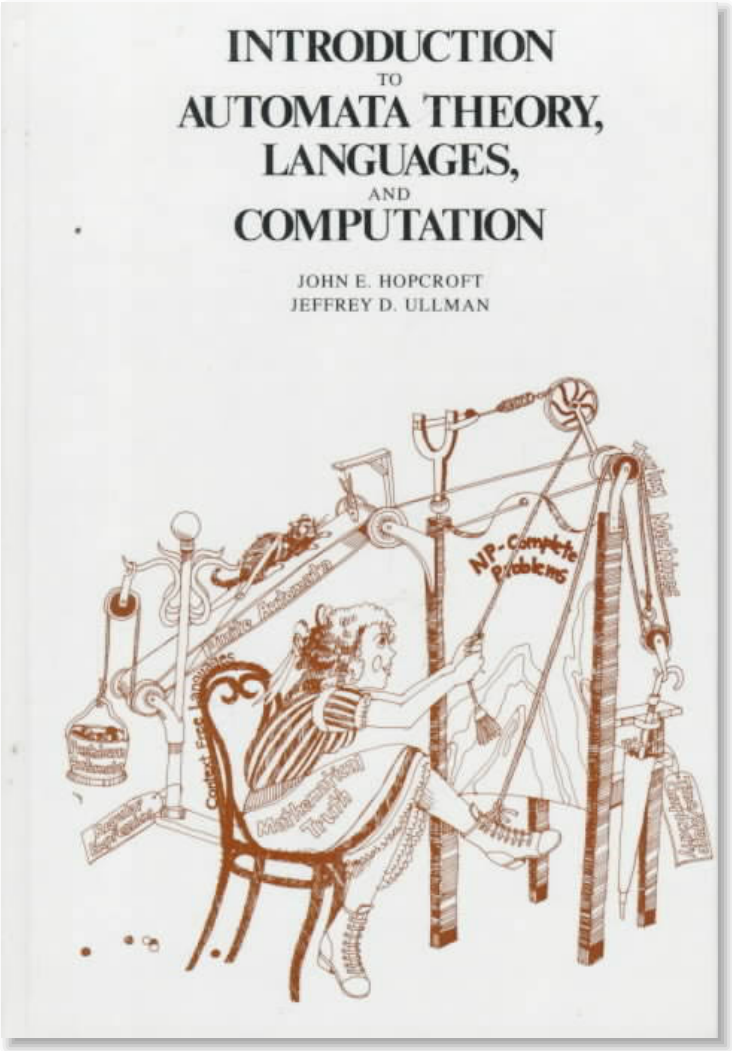
$$t^{q^{q^q}}$$

(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

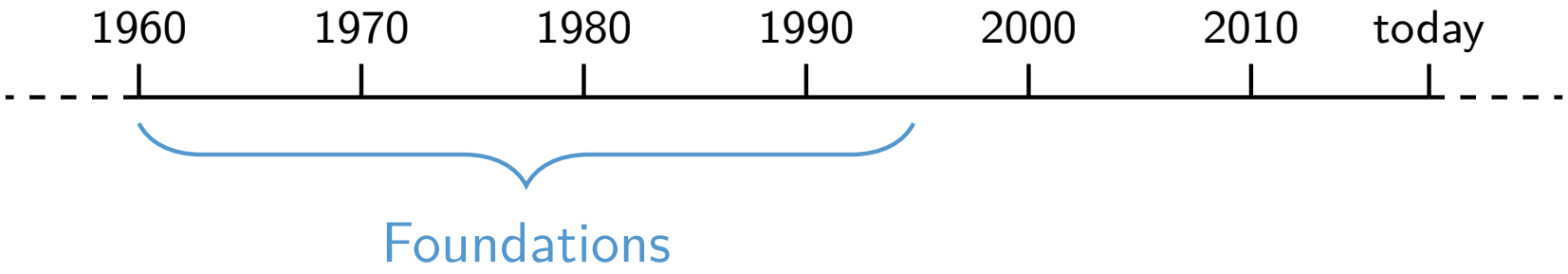
Given a deterministic pushdown automaton (PDA) accepting a regular language, construct an equivalent finite automaton

Hint

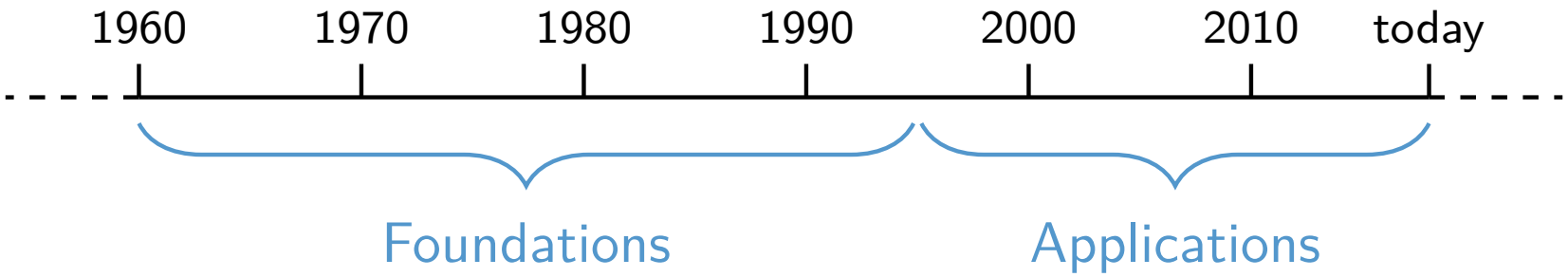
- PDAs are closed under complement and intersection with regular languages
- Emptiness of DPAs is decidable

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)

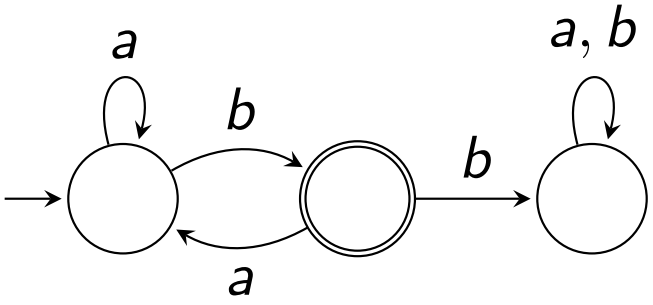
Dimensions of Automata Learning

- *Automata models*
  DFAs, NFA, stochastic automata, register automata, ...
- *Learning model*
  probably approximately correct, in-the-limit, passive, active, ...
- Theory and/or applications

Outline

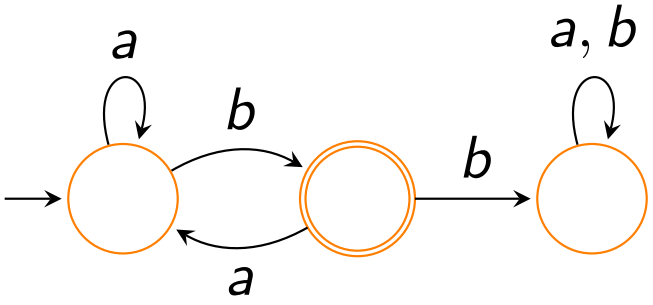
1. Passive Automata Learning
2. Active Automata Learning
3. Applications

Deterministic Finite Automata (DFAs)



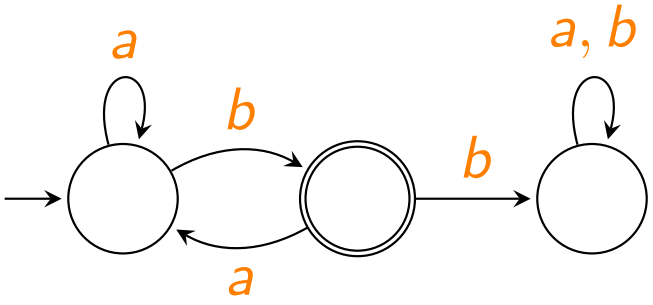
$$\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$$

Deterministic Finite Automata (DFAs)



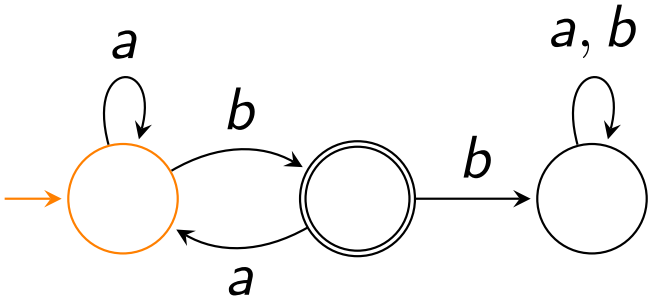
$$\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$$

Deterministic Finite Automata (DFAs)



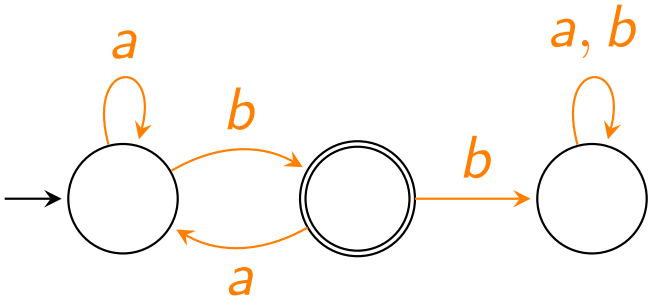
$$\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$$

Deterministic Finite Automata (DFAs)



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

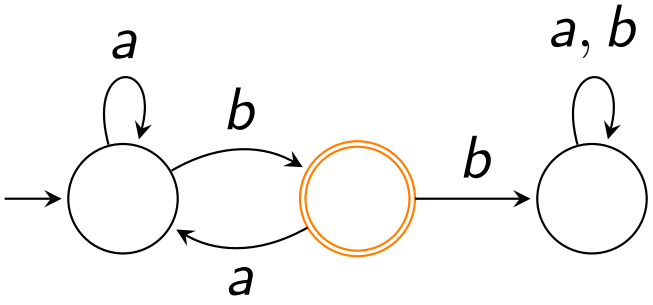
Deterministic Finite Automata (DFAs)



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

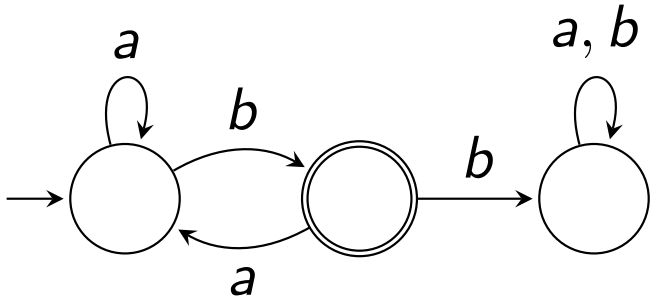
$$\delta \colon Q \times \Sigma \to Q$$

Deterministic Finite Automata (DFAs)



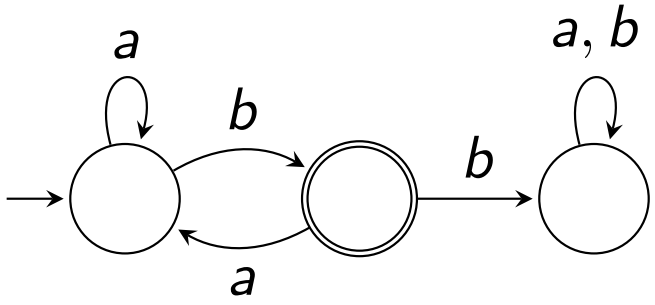
$$\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$$

Deterministic Finite Automata (DFAs)



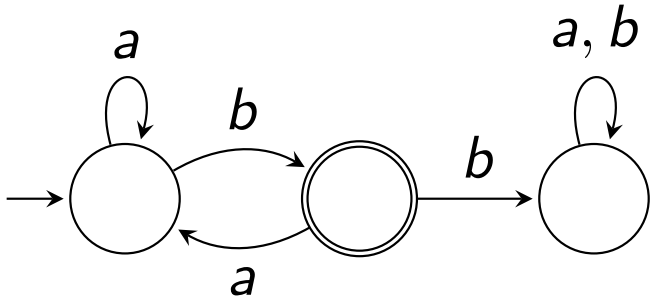
- *Run* on word $u = a_1 \dots a_n \in \Sigma^*$: $\mathcal{A}\colon q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n$

Deterministic Finite Automata (DFAs)



- *Run* on word $u = a_1 \dots a_n \in \Sigma^*$: $\mathcal{A}\colon q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n$
- A run is accepting if $q_n \in F$

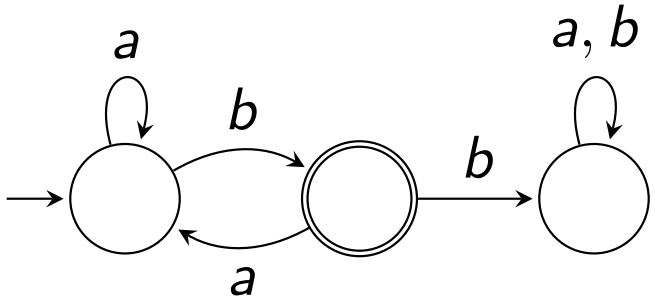
Deterministic Finite Automata (DFAs)



- *Run* on word $u = a_1 \dots a_n \in \Sigma^*$: $\mathcal{A}\colon q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n$
- A run is accepting if $q_n \in F$

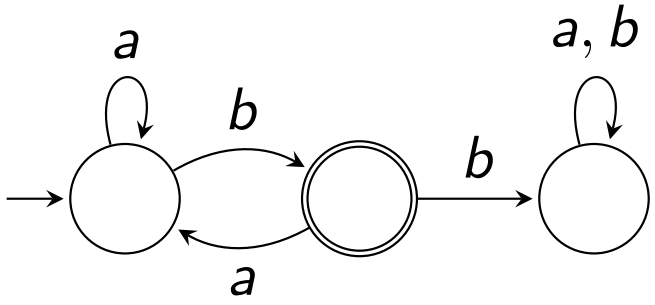
ab	✓
$abaab$	✓
$babba$	✗

Deterministic Finite Automata (DFAs)



- *Run* on word $u = a_1 \dots a_n \in \Sigma^*$: $\mathcal{A}\colon q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n$
- 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 $u = a_1 \dots a_n \in \Sigma^*$: $\mathcal{A}\colon q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n$
- A run is accepting if $q_n \in F$
- The language of a DFA $\mathcal{A}$ is $L(\mathcal{A}) = \{u \in \Sigma^* \mid \mathcal{A} \text{ accepts } u\}$
- A language $L \subseteq \Sigma^*$ is regular if there exists a DFA $\mathcal{A}$ with $L = L(\mathcal{A})$

1.

Passive Automata Learning

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

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

A DFA $\mathcal{A}$ is called *consistent* with $\mathcal{S}$ if it satisfies

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

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

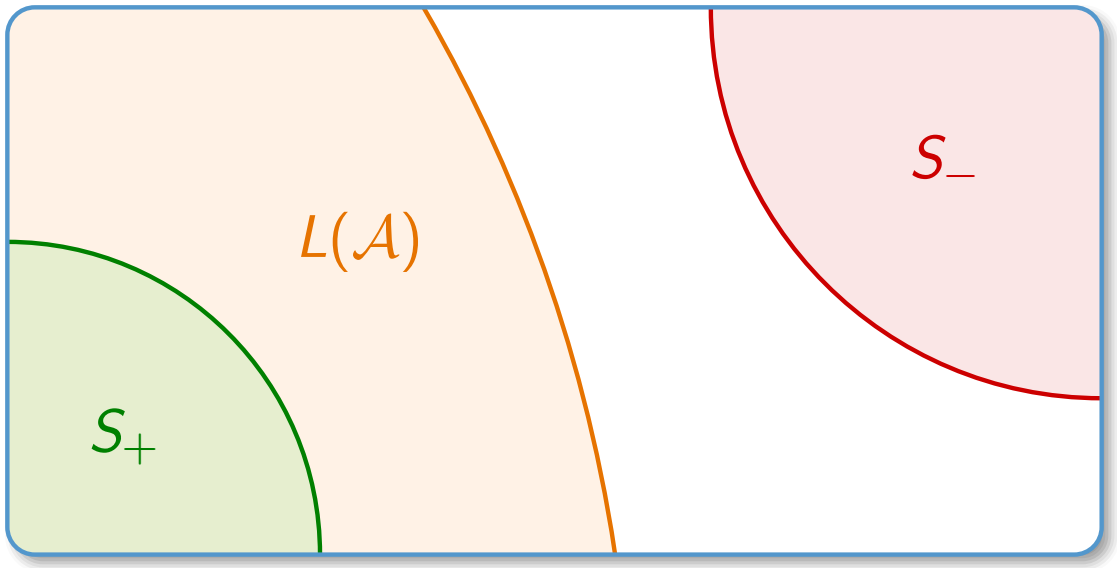
A DFA $\mathcal{A}$ is called *consistent* with $\mathcal{S}$ if it satisfies

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

Passive Learning

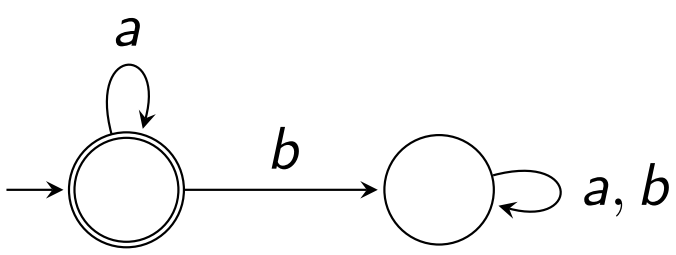
Given a sample $\mathcal{S}$, the passive learning task is to compute a (smallest) DFA that is consistent with $\mathcal{S}$

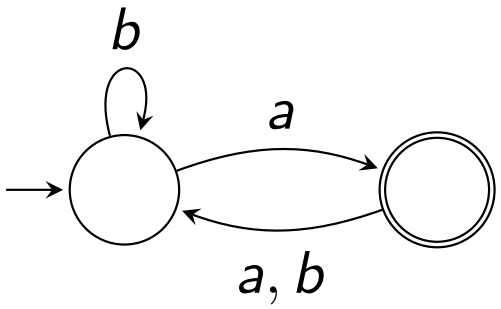
Passive Learning – A Separation Problem



Example: Passive Learning

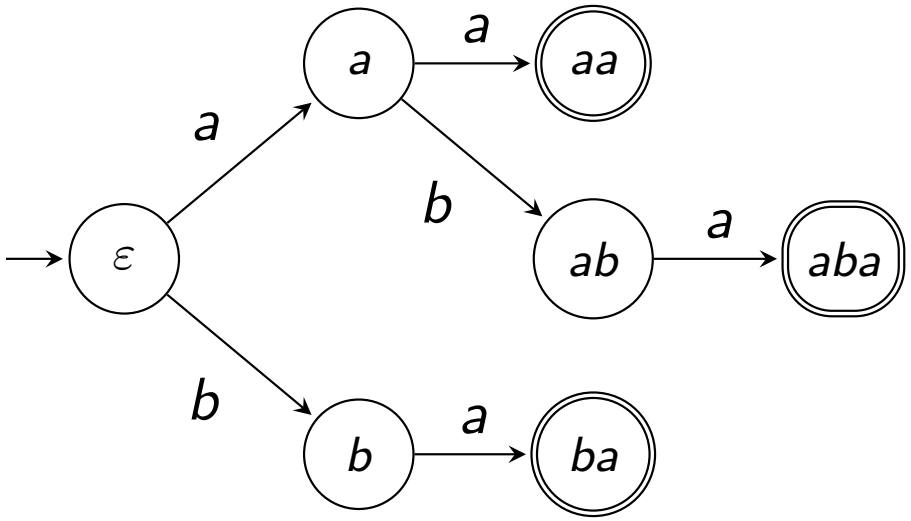
Consider $\mathcal{S} = (S_+, S_-)$ with $S_+ = \{a\}$ and $S_- = \{b\}$



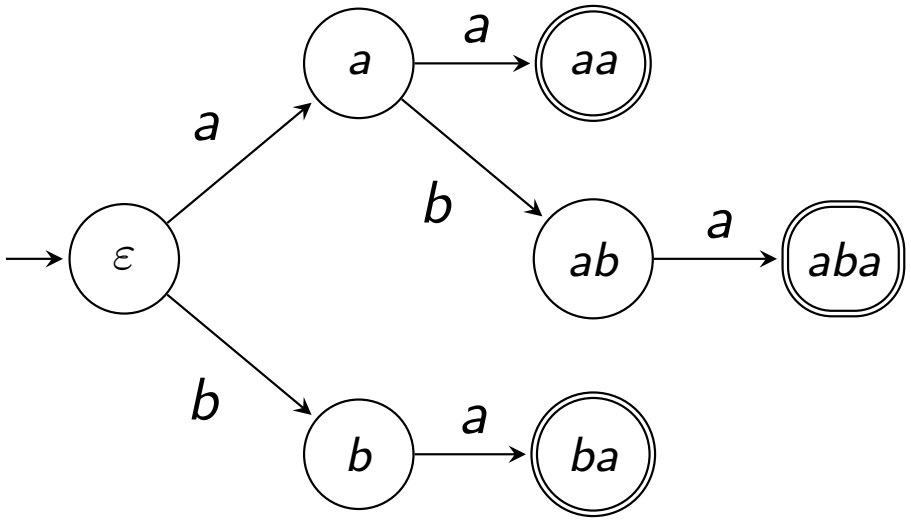


Prefix Tree Acceptor

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



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



The *prefix tree acceptor* is the DFA $\mathcal{A}_{Pref}(S_+) = (Q, \Sigma, q_0, \delta, F)$ with

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

Prefix Tree Acceptor – Continued

Observation

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

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

Prefix Tree Acceptor – Continued

Observation

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

- *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

Wikipedia

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"

Wikipedia

Passive Learning – Revisited

[1] E. M. Gold: *Complexity of automaton identification from given data*, 1978

Theorem (Gold [1], 1978)

The decision problem "Given a sample $\mathcal{S}$ and $k \in \mathbb{N}$. Does a DFA with at most k states that is consistent with $\mathcal{S}$ exist?" is NP-complete.

[1] E. M. Gold: *Complexity of automaton identification from given data*, 1978

Theorem (Gold [1], 1978)

The decision problem "Given a sample $\mathcal{S}$ and $k \in \mathbb{N}$. Does a DFA with at most k states that is consistent with $\mathcal{S}$ exist?" is NP-complete.

1.1.

Regular Positive Negative Inference

Canonical Order of Words

Let Σ be an alphabet and $<_\Sigma$ a total order on Σ

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

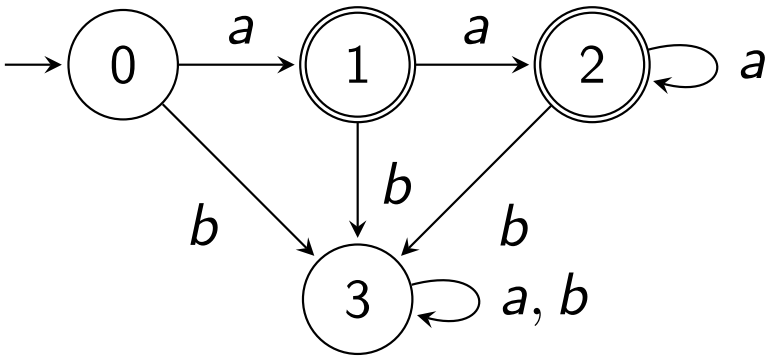
- $m < n$; or
- $m = n$ and there exists an $i \in \{1, \dots, n\}$ such that $a_i <_\Sigma a'_i$ and $a_j = a'_j$ for each $j \in \{1, \dots, i-1\}$

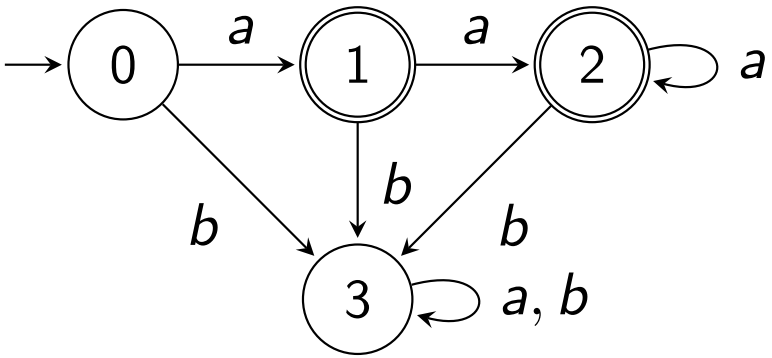
Example

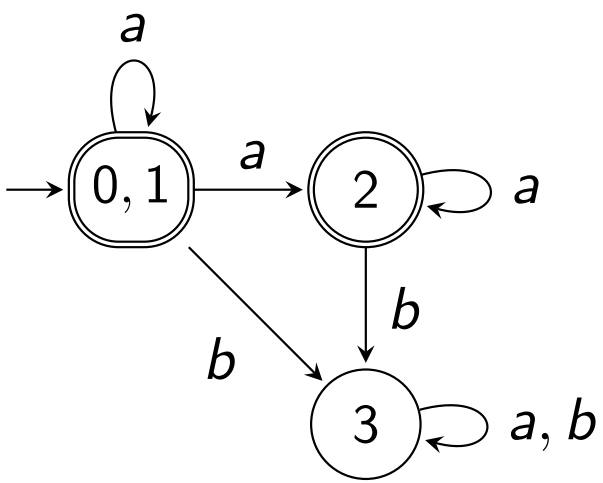
Let $\Sigma = \{a, b\}$ and $a <_\Sigma b$

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

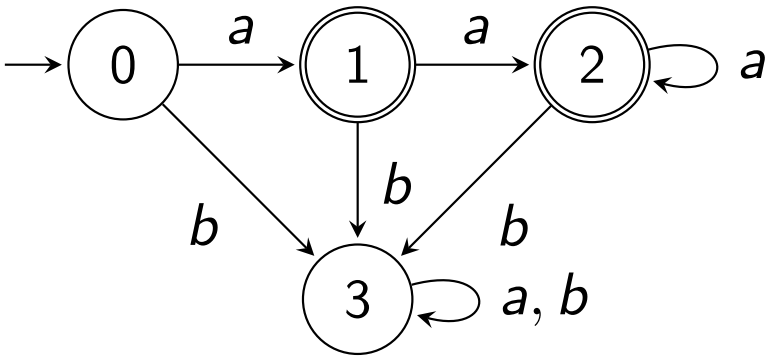
State Merging

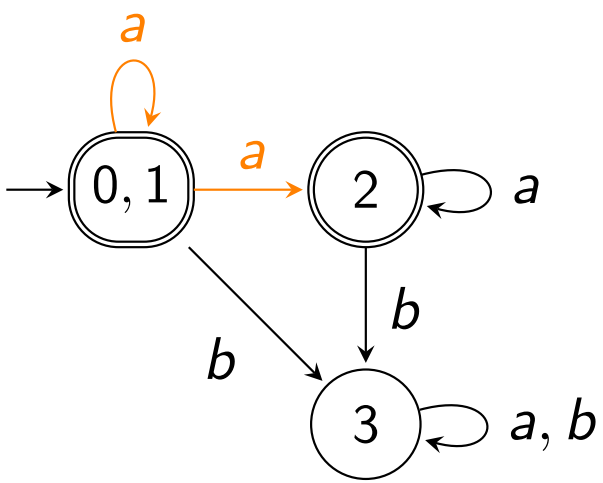


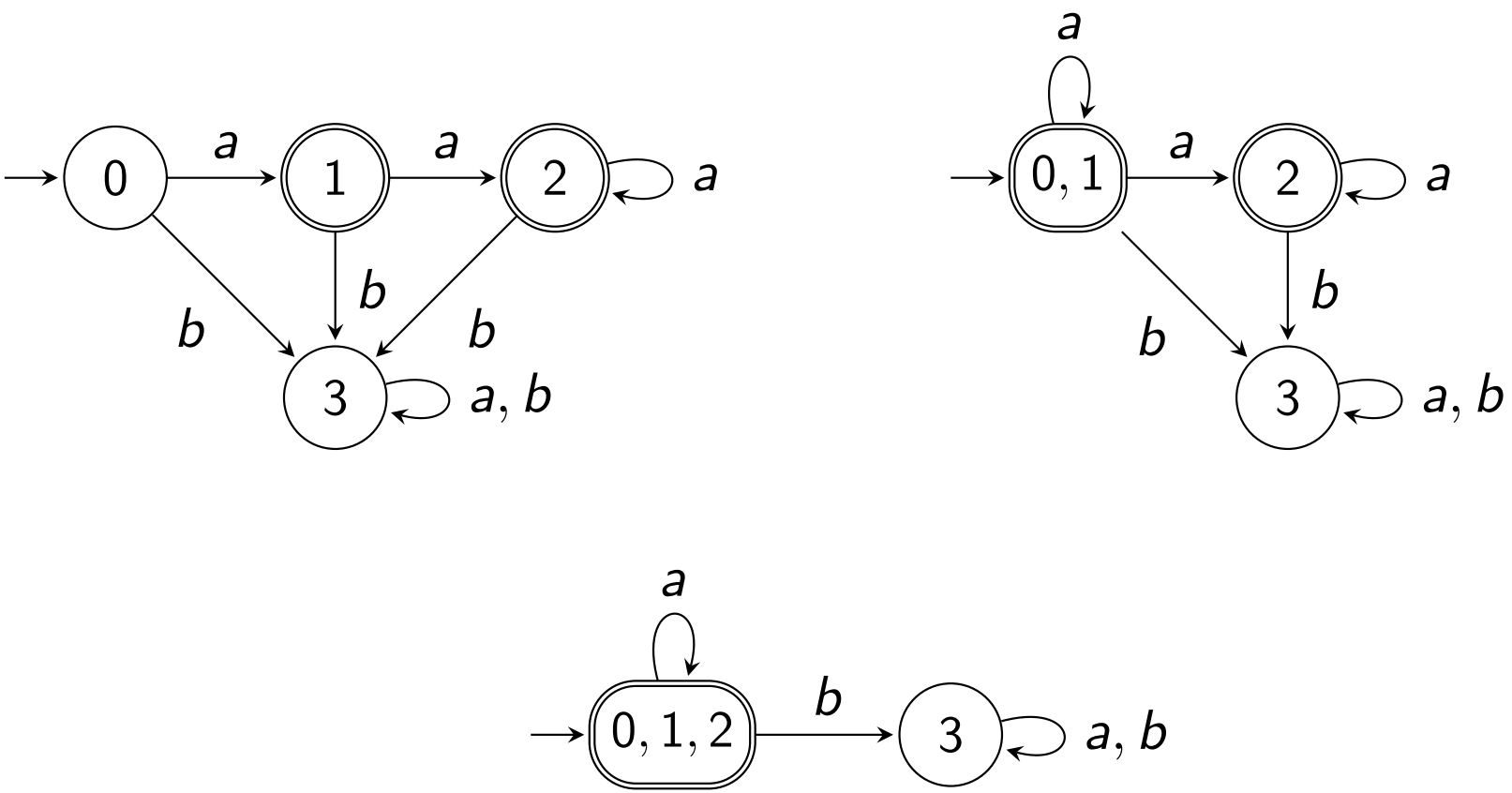

0
1
2
3
a
a
a
b
b
b
a, b

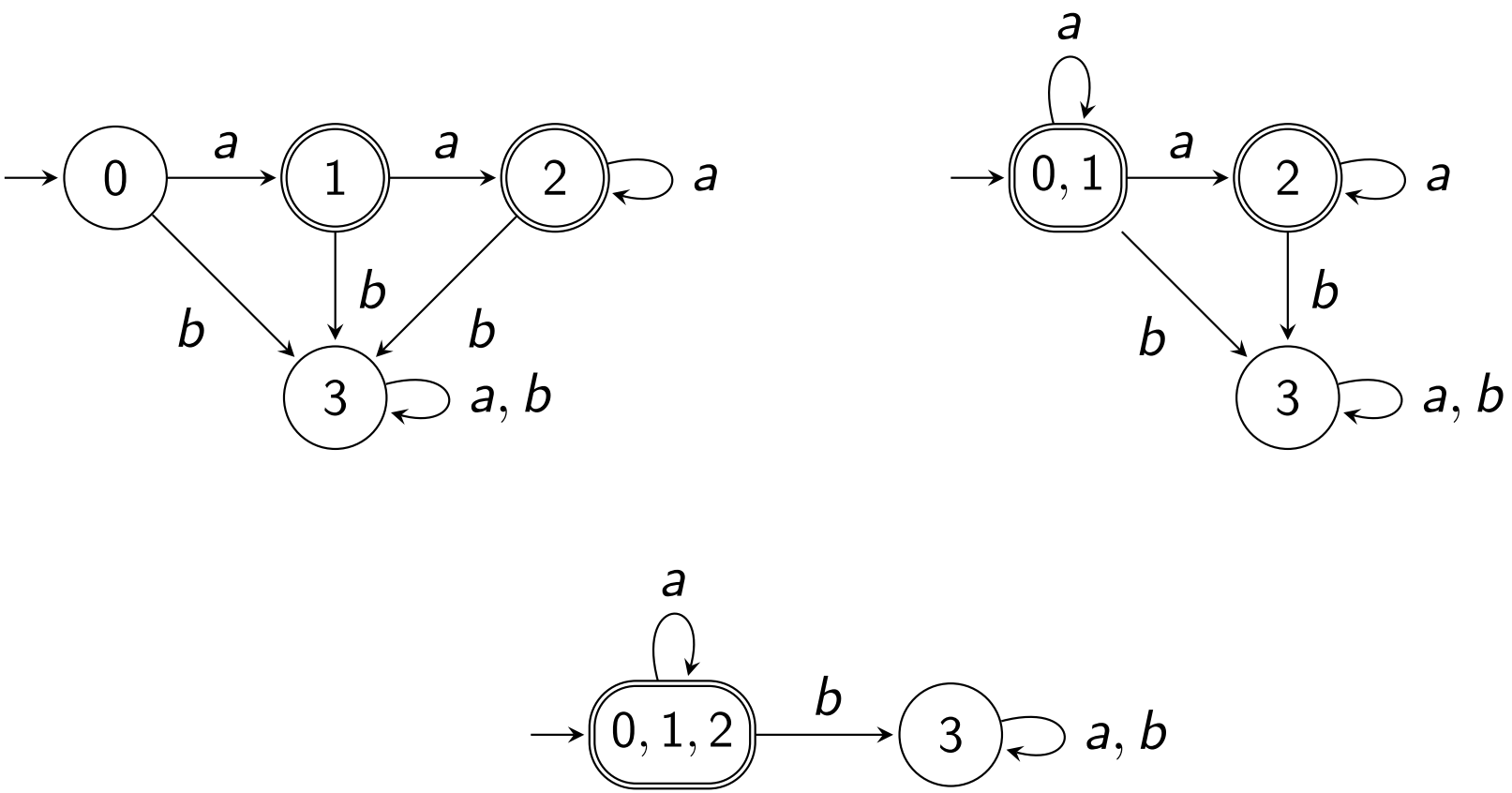

0, 1
2
3
a
a
a
b
b
a, b

State Merging






0
1
2
3
a
a
a
b
b
b
a, b
0, 1
2
3
a
a
a
b
b
a, b
0, 1, 2
3
a
b
a, b



Lemma

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

Regular Positive Negative Inference (RPNI)

Algorithm 1: RPNI (simplified)

Input: A sample $\mathcal{S} = (S_-, S_+)$

Construct $\mathcal{A}_{Pref}(S_+) = (Q, \Sigma, q_0, \delta, F)$ without sink state
Order the set $Q = \{u_0, \ldots, u_n\}$ in canonical order (s.t. $u_i < u_{i+1}$)

for $i = 1, \ldots, n$ **do**
  if *state u_i is not yet merged with a smaller state* **then**
    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
  end
end

return the final DFA

Example

Consider the sample $\mathcal{S} = (S_+, S_-)$ with

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

Example

Consider the sample $\mathcal{S} = (S_+, S_-)$ with

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

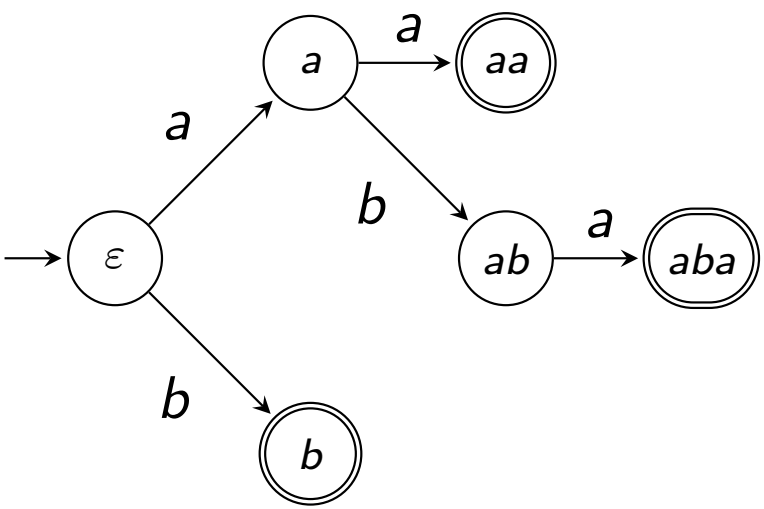
$$\varepsilon < a < b < aa < ab < aba$$

Example

Consider the sample $\mathcal{S} = (S_+, S_-)$ with

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

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

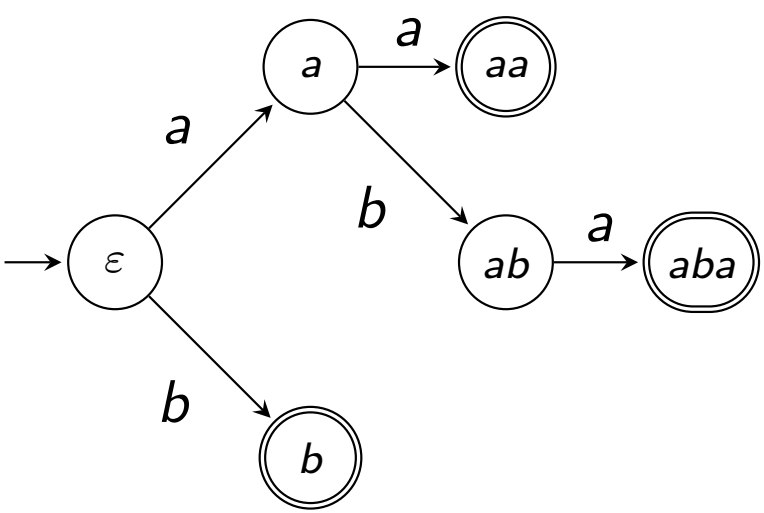


Example

Consider the sample $\mathcal{S} = (S_+, S_-)$ with

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

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

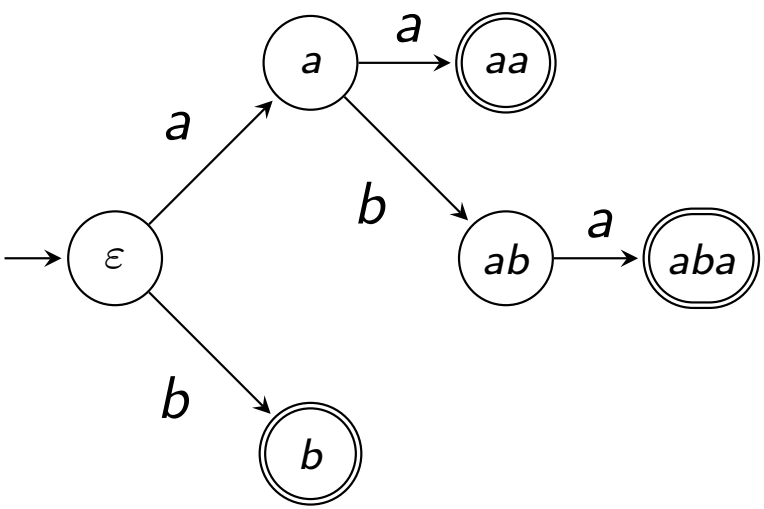


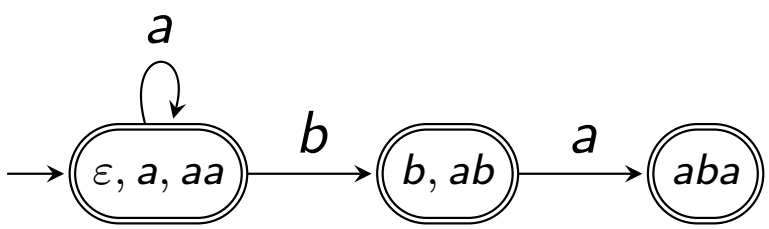
Example

Consider the sample $\mathcal{S} = (S_+, S_-)$ with

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

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



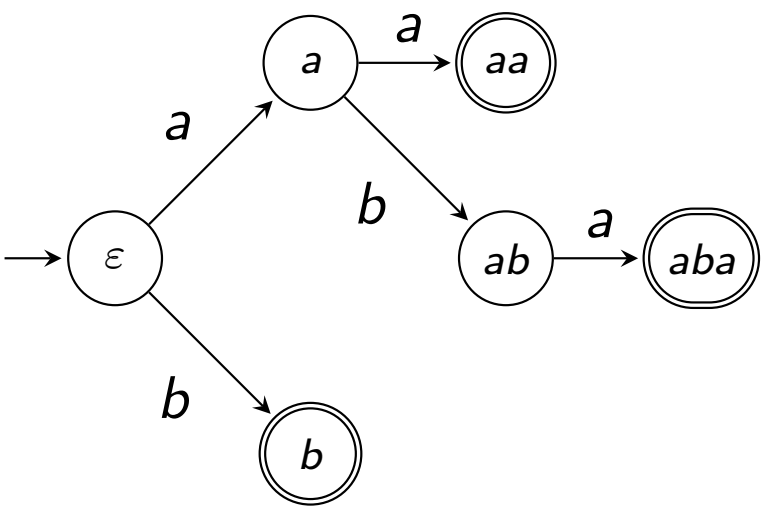


Example

Consider the sample $\mathcal{S} = (S_+, S_-)$ with

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

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

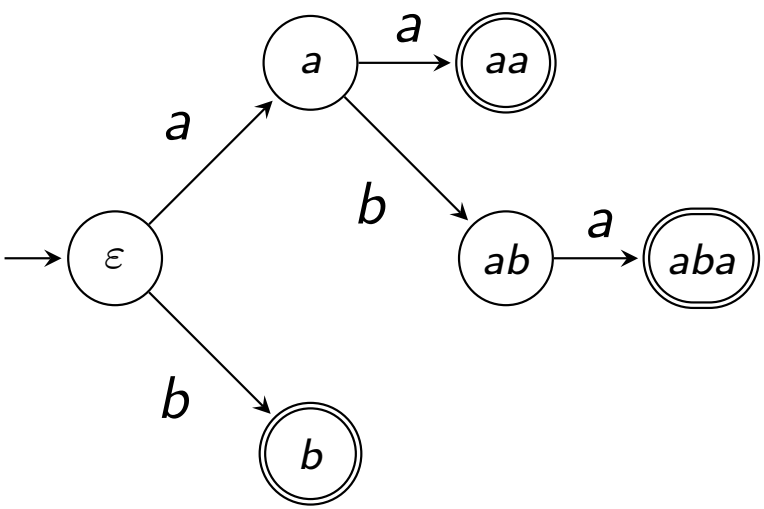


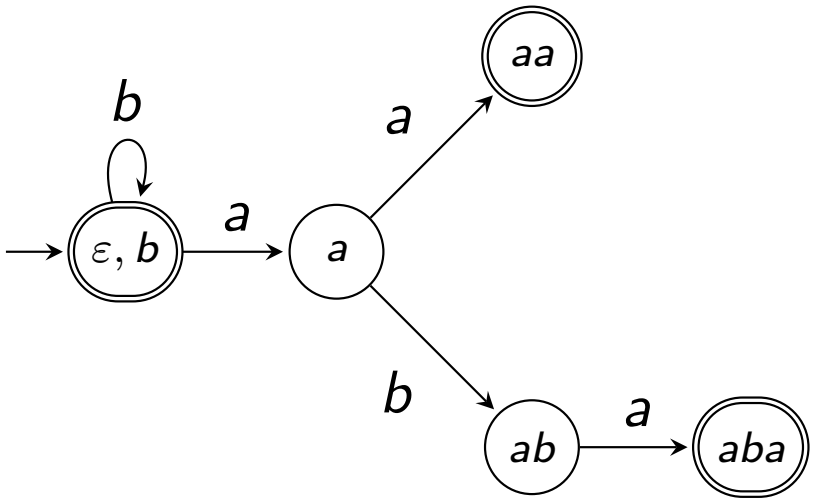
Example

Consider the sample $\mathcal{S} = (S_+, S_-)$ with

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

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



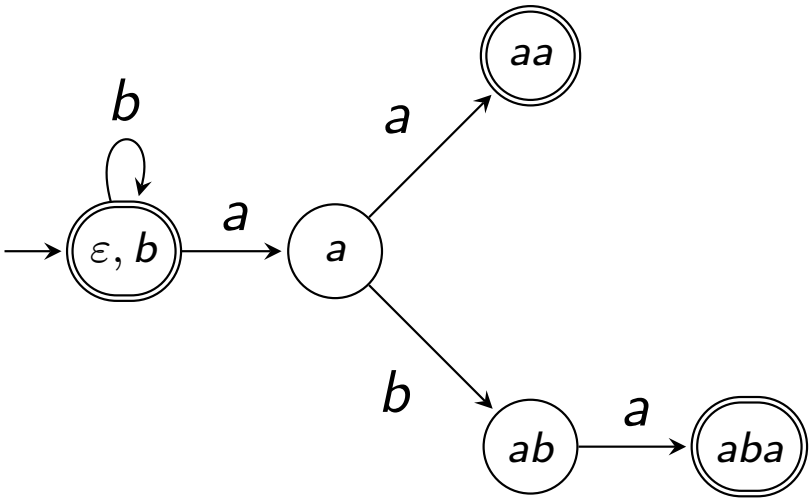


Example

Consider the sample $\mathcal{S} = (S_+, S_-)$ with

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

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

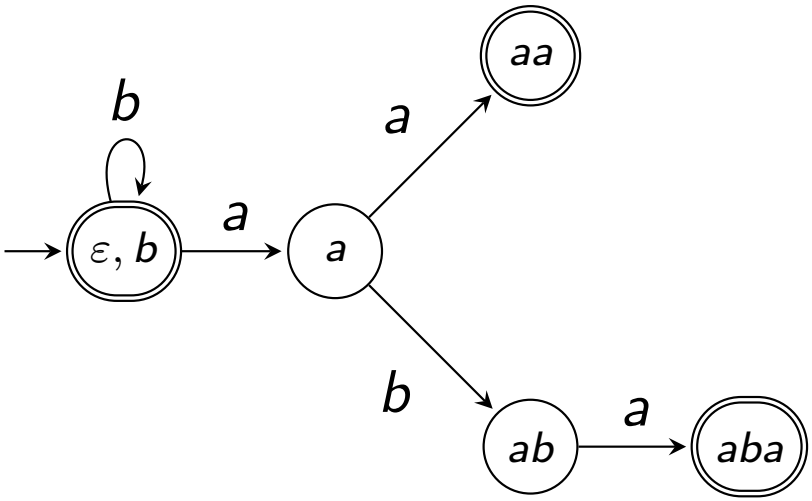


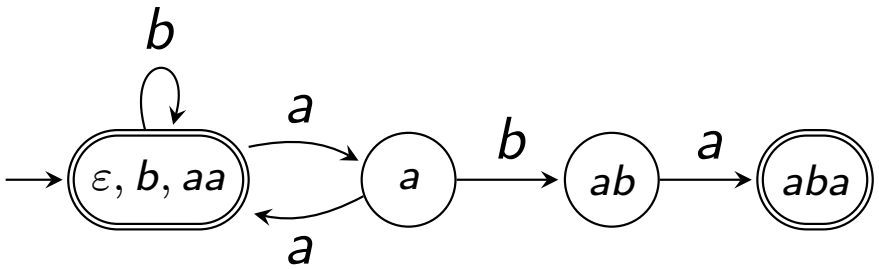
Example

Consider the sample $\mathcal{S} = (S_+, S_-)$ with

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

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



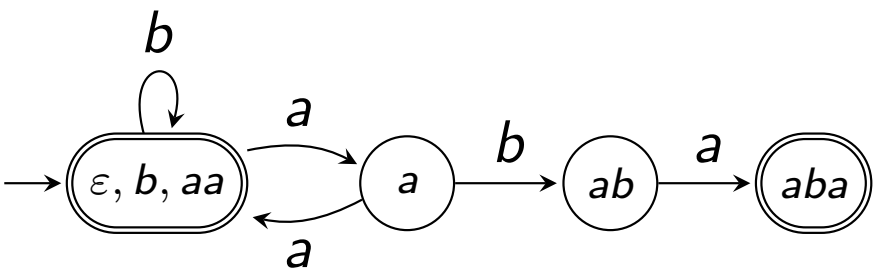


Example

Consider the sample $\mathcal{S} = (S_+, S_-)$ with

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

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

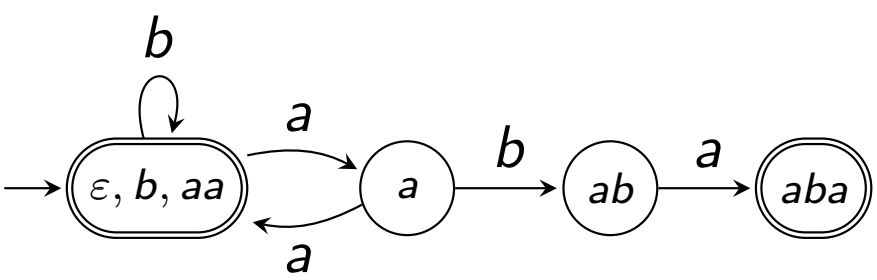


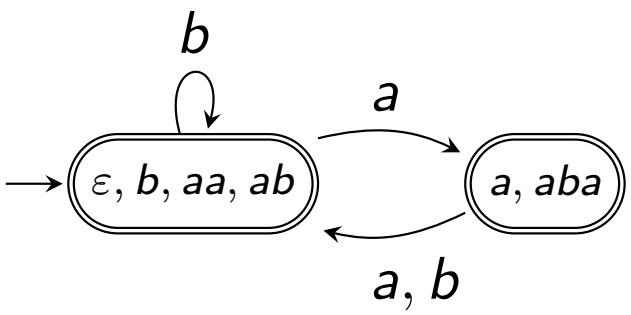
Example

Consider the sample $\mathcal{S} = (S_+, S_-)$ with

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

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



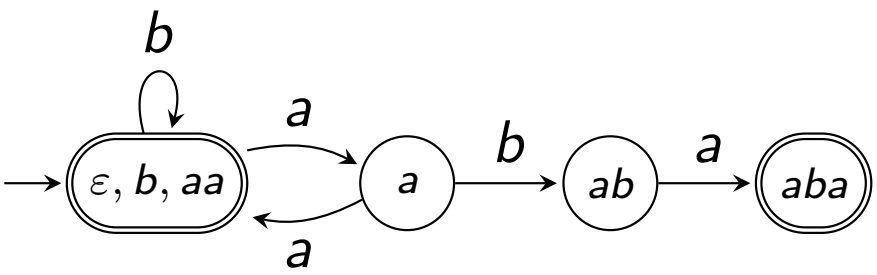


Example

Consider the sample $\mathcal{S} = (S_+, S_-)$ with

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

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

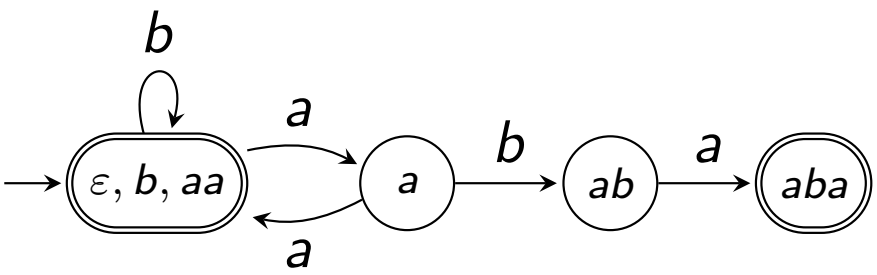


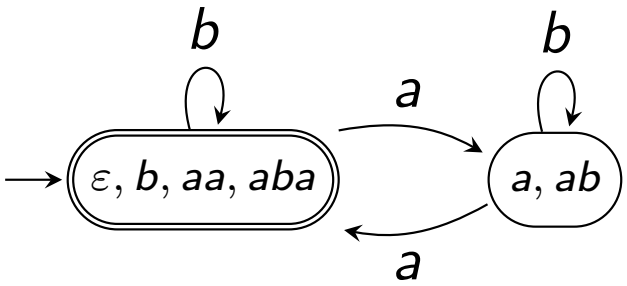
Example

Consider the sample $\mathcal{S} = (S_+, S_-)$ with

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

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



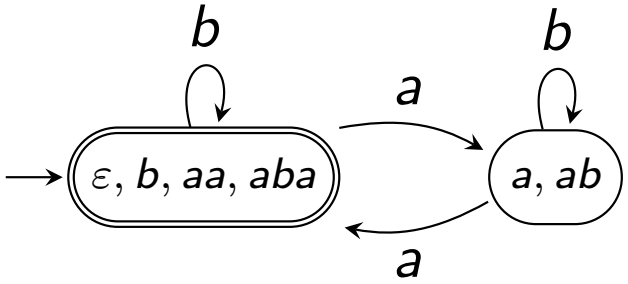


Example

Consider the sample $\mathcal{S} = (S_+, S_-)$ with

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

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



[2] J. Oncina and P. Garcia: *Inferring regular languages in polynomial update time*, 1992

Theorem (Oncina and Garcia [2], 1992)

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

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

[2] J. Oncina and P. Garcia: *Inferring regular languages in polynomial update time*, 1992

Theorem (Oncina and Garcia [2], 1992)

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

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

Proof

- $\mathcal{A}_{Pref}(S_+)$ is consistent with $\mathcal{S}$
- Merging states only increases the accepted language
- A merged DFA is only kept if it is still consistent with $\mathcal{S}$

[2] J. Oncina and P. Garcia: *Inferring regular languages in polynomial update time*, 1992

Theorem (Oncina and Garcia [2], 1992)

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

- *RPNI produces a (partial) DFA that is consistent with $\mathcal{S}$*
- *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 $\mathcal{O}(m)$

[2] J. Oncina and P. Garcia: *Inferring regular languages in polynomial update time*, 1992

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 Lea

abbadingo.cs.nuim.ie

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 S.

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

[3] K. J. Lang, B. A. Pearlmutter, and R. A. Price: *Results of the Abbadingo One DFA Learning Competition*, 1998

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

Idea

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

Idea

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

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

1. $\varphi_n^{\mathcal{S}}$ is satisfiable if and only if there exists a DFA with n states that is consistent with $\mathcal{S}$
2. A model $\mathfrak{M}$ of $\varphi_n^{\mathcal{S}}$ contains sufficient information to construct a consistent DFA $\mathcal{A}_{\mathfrak{M}}$

Algorithm 2: Passive Learning of Minimal DFAs

Input: A sample $\mathcal{S}$

```
n ← 0
repeat
    n ← n + 1
    Construct and solve φ_n^S
until φ_n^S is satisfiable, say with model 𝔐
return A_𝔐
```

Algorithm 2: Passive Learning of Minimal DFAs

Input: A sample $\mathcal{S}$

$n \leftarrow 0$
repeat
  $n \leftarrow n + 1$
  Construct and solve $\varphi_n^{\mathcal{S}}$
until $\varphi_n^{\mathcal{S}}$ *is satisfiable, say with model* $\mathfrak{M}$
return $\mathcal{A}_{\mathfrak{M}}$

- This algorithm is guaranteed to terminate (given the properties of $\varphi_n^{\mathcal{S}}$), and $|\mathcal{A}_{Pref}(S_+)|$ 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

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

Fix the size $n \geq 1$ of the prospective DFA

Biermann and Feldman's Method

Idea

Encode the run of a DFA on (prefixes of) words from the sample $\mathcal{S}$

Biermann and Feldman's Method

Idea

Encode the run of a DFA on (prefixes of) words from the sample $\mathcal{S}$

Variables

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

Idea

Encode the run of a DFA on (prefixes of) words from the sample $\mathcal{S}$

Variables

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

Constraints

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

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

Biermann and Feldman's Method

Idea

Encode the run of a DFA on (prefixes of) words from the sample $\mathcal{S}$

Variables

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

Constraints

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

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

Let $\varphi_n^{\mathcal{S}}(\vec{x})$ denote the conjunction of these constraints

From Model to DFA

Let $\mathfrak{M}\colon \mathcal{X} \to \{1, \ldots, n\}$ be a model of $\varphi_n^{\mathcal{S}}(\vec{x})$

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

- $Q = \{1, \ldots, n\}$

From Model to DFA

Let $\mathfrak{M}\colon \mathcal{X} \to \{1, \ldots, n\}$ be a model of $\varphi_n^{\mathcal{S}}(\vec{x})$

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

- $Q = \{1, \ldots, n\}$
- $q_0 = i$ if and only if $\mathfrak{M}(x_\varepsilon) = i$

From Model to DFA

Let $\mathfrak{M}\colon \mathcal{X} \to \{1, \ldots, n\}$ be a model of $\varphi_n^{\mathcal{S}}(\vec{x})$

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

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

From Model to DFA

Let $\mathfrak{M}\colon \mathcal{X} \to \{1, \ldots, n\}$ be a model of $\varphi_n^{\mathcal{S}}(\vec{x})$

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

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

From Model to DFA

Let $\mathfrak{M}\colon \mathcal{X} \to \{1, \ldots, n\}$ be a model of $\varphi_n^{\mathcal{S}}(\vec{x})$

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

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

Note that $\mathcal{A}_{\mathfrak{M}}$ is well-defined

[4] A. W. Biermann and J. A. Feldman: *On the synthesis of finite-state machines from samples of their behavior*, 1972

Theorem (Biermann and Feldman [4], 1972)

1. *Let $\mathcal{S}$ be a sample, $\mathfrak{M}\colon \mathcal{X} \to \{1, \ldots, n\}$ a model of $\varphi_n^{\mathcal{S}}(\vec{x})$, and $\mathcal{A}_{\mathfrak{M}}$ the DFA constructed from $\mathfrak{M}$. Then, $\mathcal{A}_{\mathfrak{M}}$ is a DFA with n states that is consistent with $\mathcal{S}$*
2. *If a consistent DFA with n states exists, then $\varphi_n^{\mathcal{S}}(\vec{x})$ is satisfiable*

[4] A. W. Biermann and J. A. Feldman: *On the synthesis of finite-state machines from samples of their behavior*, 1972

Theorem (Biermann and Feldman [4], 1972)

1. *Let $\mathcal{S}$ be a sample, $\mathfrak{M}\colon \mathcal{X} \to \{1, \ldots, n\}$ a model of $\varphi_n^{\mathcal{S}}(\vec{x})$, and $\mathcal{A}_{\mathfrak{M}}$ the DFA constructed from $\mathfrak{M}$. Then, $\mathcal{A}_{\mathfrak{M}}$ is a DFA with n states that is consistent with $\mathcal{S}$*
2. *If a consistent DFA with n states exists, then $\varphi_n^{\mathcal{S}}(\vec{x})$ is satisfiable*

Proof Sketch

1. Show (by induction) that $\mathfrak{M}(x_u) = i$ indeed implies $\mathcal{A}_{\mathfrak{M}}\colon q_0 \xrightarrow{u} i$ for all $u \in \mathit{Pref}(S_+ \cup S_-)$
   - Using this result, conclude that $\mathcal{A}_{\mathfrak{M}}$ is consistent with $\mathcal{S}$
2. Use the runs of a consistent DFA on words from $\mathcal{S}$ to define a model of $\varphi_n^{\mathcal{S}}$

[4] A. W. Biermann and J. A. Feldman: *On the synthesis of finite-state machines from samples of their behavior*, 1972

Theorem (Biermann and Feldman [4], 1972)

1. *Let $\mathcal{S}$ be a sample, $\mathfrak{M}\colon \mathcal{X} \to \{1, \ldots, n\}$ a model of $\varphi_n^{\mathcal{S}}(\vec{x})$, and $\mathcal{A}_{\mathfrak{M}}$ the DFA constructed from $\mathfrak{M}$. Then, $\mathcal{A}_{\mathfrak{M}}$ is a DFA with n states that is consistent with $\mathcal{S}$*
2. *If a consistent DFA with n states exists, then $\varphi_n^{\mathcal{S}}(\vec{x})$ 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

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

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 $x_{u,1}, \ldots, x_{u,n}$

Constraints

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

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

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

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

$$\bigwedge_{ua, u'a \in Pref(S_+ \cup S_-)} \bigwedge_{p,q \in Q} \left(x_{u,p} \wedge x_{u',p}\right) \rightarrow \left(x_{ua,q} \leftrightarrow x_{u'a,q}\right)$$

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

Let $\psi_n^{\mathcal{S}}(\vec{x})$ denote the conjunction of these constraints

From Model to DFA

Let $\mathfrak{M}\colon \mathcal{X} \to \{0,1\}$ be a model of $\psi_n^{\mathcal{S}}(\vec{x})$

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

- $Q = \{1, \ldots, n\}$

From Model to DFA

Let $\mathfrak{M}\colon \mathcal{X} \to \{0,1\}$ be a model of $\psi_n^{\mathcal{S}}(\vec{x})$

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

- $Q = \{1, \ldots, n\}$
- $q_0 = i$ for the unique $i \in Q$ with $\mathfrak{M}(x_{\varepsilon,i}) = 1$

From Model to DFA

Let $\mathfrak{M}\colon \mathcal{X} \to \{0,1\}$ be a model of $\psi_n^{\mathcal{S}}(\vec{x})$

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

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

Let $\mathfrak{M}\colon \mathcal{X} \to \{0,1\}$ be a model of $\psi_n^{\mathcal{S}}(\vec{x})$

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

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

[5] O. Grinchtein, M. Leucker, and N. Piterman: *Inferring network invariants automatically*, 2006

Theorem (Grinchtein, Leucker, and Piterman [5], 2006)

- *Let $\mathcal{S}$ be a sample, $\mathfrak{M}\colon \mathcal{X} \to \{0,1\}$ a model of $\psi_n^{\mathcal{S}}(\vec{x})$, and $\mathcal{A}_{\mathfrak{M}}$ the DFA constructed from $\mathfrak{M}$. Then, $\mathcal{A}_{\mathfrak{M}}$ is a DFA with n states that is consistent with $\mathcal{S}$*
- *If a consistent DFA with n states exists, then $\psi_n^{\mathcal{S}}(\vec{x})$ is satisfiable*

Grinchtein, Leucker, and Piterman's Binary Method

Idea

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

Variables

For $u \in Pref(S_+ \cup S_-)$ use Boolean variables $x_{u,1}, \ldots, x_{u,\lceil \log_2 n \rceil}$ to encode the state reached after reading u

Heule and Verwer's Method

Idea

Encode the transitions and final states of an automaton explicitly

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 $u \in Pref(S_+ \cup S_-)$ 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:

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

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

Constraints II

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

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

Constraints II

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

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

$$\bigwedge_{ua \in Pref(S_+ \cup S_-)} \bigwedge_{p,q \in Q} (x_{u,p} \wedge d_{p,a,q}) \rightarrow x_{ua,q}$$

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

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

$$\bigwedge_{ua \in Pref(S_+ \cup S_-)} \bigwedge_{p,q \in Q} (x_{u,p} \wedge d_{p,a,q}) \rightarrow x_{ua,q}$$

$$\left[\bigwedge_{u \in S_+} \bigwedge_{q \in Q} x_{u,q} \rightarrow f_q \right] \wedge \left[\bigwedge_{u \in S_-} \bigwedge_{q \in Q} x_{u,q} \rightarrow \neg f_q \right]$$

Let $\chi_n^{\mathcal{S}}(\vec{d}, \vec{f}, \vec{x})$ denote the conjunction of these constraints

From Model to DFA

Let $\mathfrak{M}\colon \mathcal{X} \to \{0,1\}$ be a model of $\chi_n^{\mathcal{S}}(\vec{d}, \vec{f}, \vec{x})$

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

- $Q = \{1, \ldots, n\}$

From Model to DFA

Let $\mathfrak{M}\colon \mathcal{X} \to \{0,1\}$ be a model of $\chi_n^{\mathcal{S}}(\vec{d}, \vec{f}, \vec{x})$

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

- $Q = \{1, \ldots, n\}$
- $q_0 = i$ for the unique $i \in Q$ with $\mathfrak{M}(x_{\varepsilon,i}) = 1$

From Model to DFA

Let $\mathfrak{M}\colon \mathcal{X} \to \{0,1\}$ be a model of $\chi_n^{\mathcal{S}}(\vec{d}, \vec{f}, \vec{x})$

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

- $Q = \{1, \ldots, n\}$
- $q_0 = i$ for the unique $i \in Q$ with $\mathfrak{M}(x_{\varepsilon,i}) = 1$
- $\delta(i, a) = j$ for the unique $j \in Q$ with $\mathfrak{M}(d_{i,a,j}) = 1$

From Model to DFA

Let $\mathfrak{M}\colon \mathcal{X} \to \{0,1\}$ be a model of $\chi_n^{\mathcal{S}}(\vec{d}, \vec{f}, \vec{x})$

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

- $Q = \{1, \ldots, n\}$
- $q_0 = i$ for the unique $i \in Q$ with $\mathfrak{M}(x_{\varepsilon,i}) = 1$
- $\delta(i, a) = j$ for the unique $j \in Q$ with $\mathfrak{M}(d_{i,a,j}) = 1$
- $F = \{i \in Q \mid \mathfrak{M}(f_i) = 1\}$

[6] M. Heule and S. Verwer: *Exact DFA Identification Using SAT Solvers*, 2010

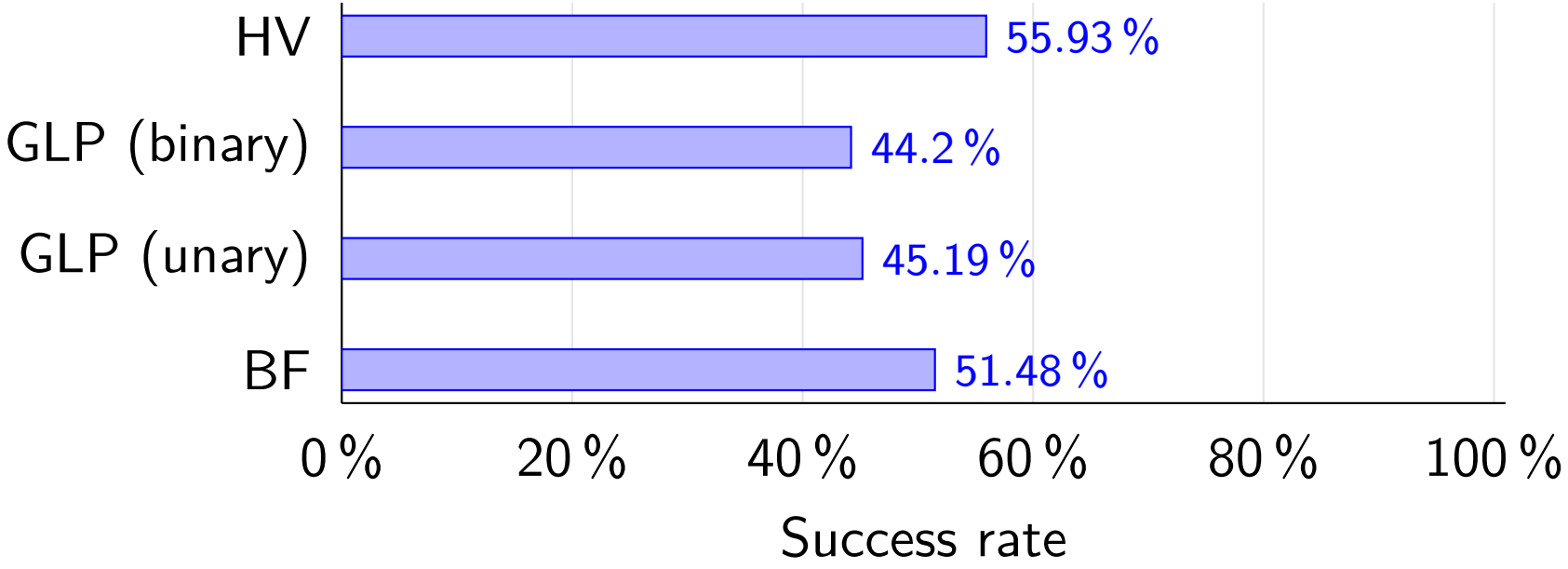
Theorem (Heule and Verwer [6], 2010)

- *Let $\mathcal{S}$ be a sample, $\mathfrak{M}\colon \mathcal{X} \to \{0,1\}$ a model of $\chi_n^{\mathcal{S}}(\vec{d},\vec{f},\vec{x})$, and $\mathcal{A}_{\mathfrak{M}}$ the DFA constructed from $\mathfrak{M}$. Then, $\mathcal{A}_{\mathfrak{M}}$ is a DFA with n states that is consistent with $\mathcal{S}$*
- *If a consistent DFA with n states exists, then $\chi_n^{\mathcal{S}}(\vec{d},\vec{f},\vec{x})$ is satisfiable*

Let $k = |Pref(S_+ \cup S_-)|$

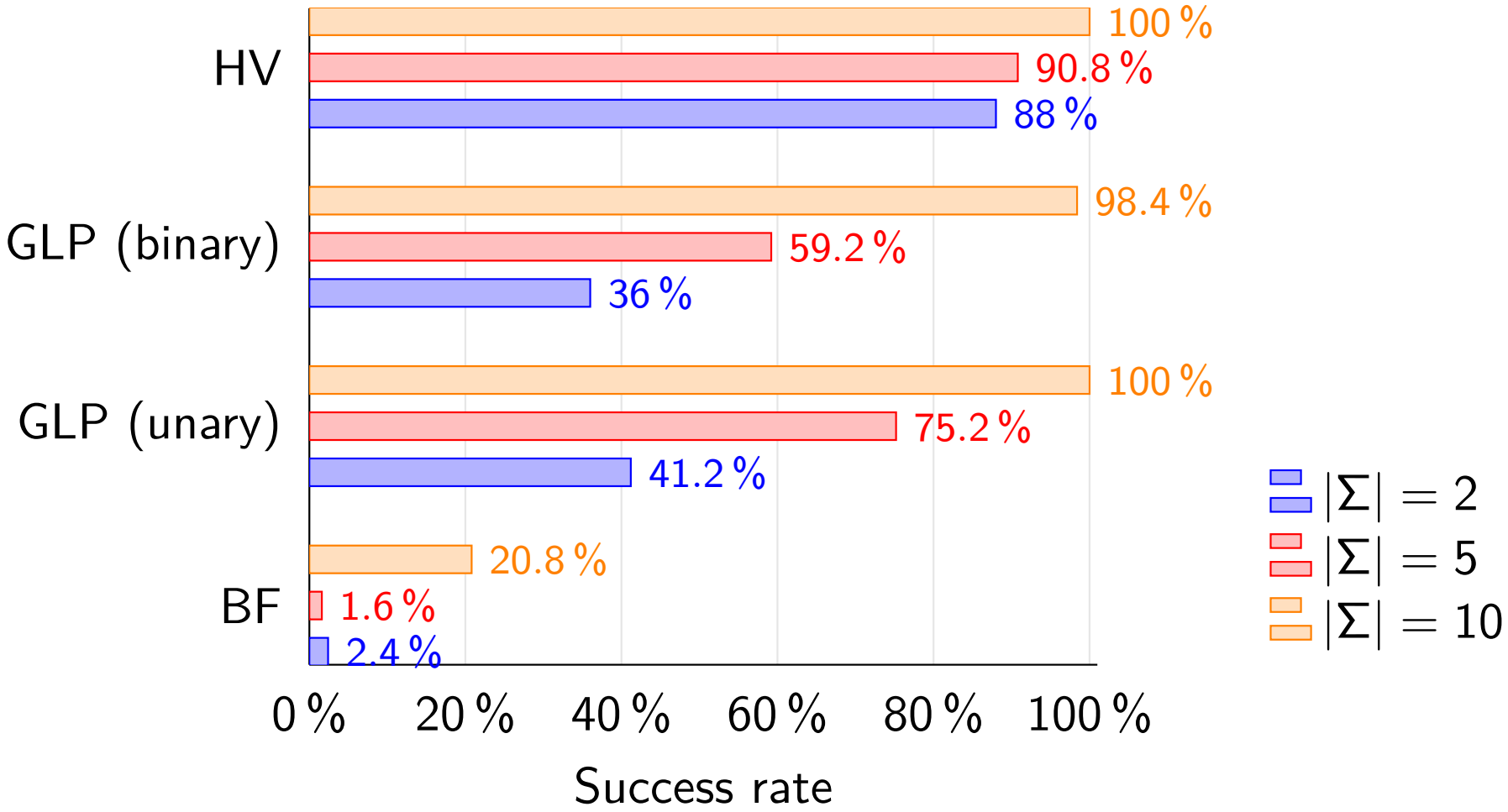
Method	*Logic*	*Size*
Biermann and Feldmann	Equality logic	$\mathcal{O}(k)$ variables $\mathcal{O}(k + \lvert S_+\rvert \cdot \lvert S_-\rvert)$ clauses
Grinchtein, Leucker, and Pitermann (unary)	Propositional Boolean logic	$\mathcal{O}(kn)$ variables $\mathcal{O}(kn^2 + n\lvert S_+\rvert \cdot \lvert S_-\rvert)$ clauses
Grinchtein, Leucker, and Pitermann (binary)	Propositional Boolean logic	$\mathcal{O}(k \log_2 n)$ variables $\mathcal{O}(kn \log_2 n + n\lvert S_+\rvert \cdot \lvert S_-\rvert)$ clauses
Heule and Verwer	Propositional Boolean logic	$\mathcal{O}(n^2\lvert\Sigma\rvert + kn)$ variables $\mathcal{O}(n^3\lvert\Sigma\rvert + kn^2)$ clauses

Empirical Comparison



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: The Automata Lear

libalf.informatik.rwth-aachen.de

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 RPNI), 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.

- L. Feng, M. Kwiatkowska, and D. Parker. Compositional verification of probabilistic systems using learning. In the Seventh International Conference on the Quantitative Evaluation of Systems, pages 133 to 142, IEEE, 2010.
- J. Esparza, M. Leucker, and M. Schlund. Learning Workflow Petri Nets. In Applications and Theory of Petri Nets, pages 206 to 225, Springer, 2010.
- M. Merten, B. Steffen, F. Howar, and T. Margaria. Next Generation LearnLib. In Tools and Algorithms for the Construction and Analysis of Systems, pages 220 to 223, Springer, 2011.
- D. N. Reachability Games on Automatic Graphs. In 15th International Conference on Implementation and Application of Automata, pages 222 to 230, Springer, 2011.

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.

Latest news

April 9th, 2011

Release of libalf 0.3. Version 0.3 features some new features and the Rivest / Schapire learning algorithm. Please have a look at the Changelog.

March 10th, 2010

Release of libalf 0.2. Version 0.2 now features the Kearns / Vazirani and Biermann's original learning algorithm. Additionally, various bugs are fixed.

October 19th, 2009

libalf website launched. On October 19th, 2009 the libalf team launched the new website.

October 12th, 2009

libalf released. The first beta of the libalf library has been released.

libalf demo

Launch the libalf demo

(Requires the Java Web Start technology)

read more

http://libalf.informatik.rwth-aachen.de/

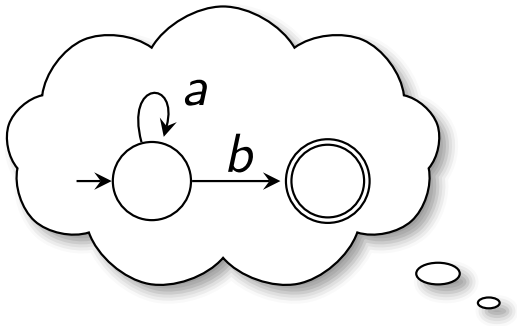
Challenge 1

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

2.

Active Automata Learning

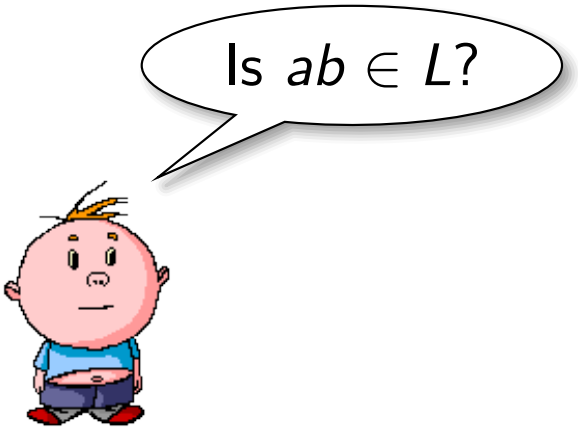
Angluin's Active Learning Setting



Learning algorithm

Teacher

Angluin's Active Learning Setting



Learning algorithm

Teacher

- Membership query $u \in L$?

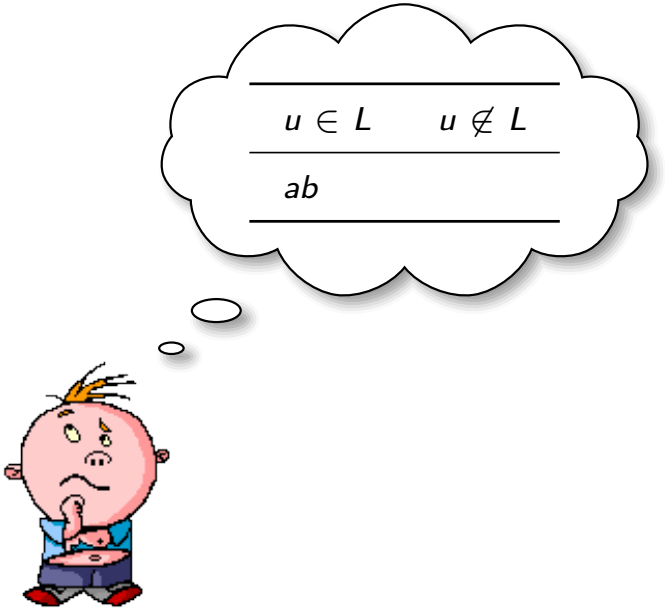
Angluin's Active Learning Setting

Learning algorithm

Teacher

- Membership query $u \in L$?

Angluin's Active Learning Setting

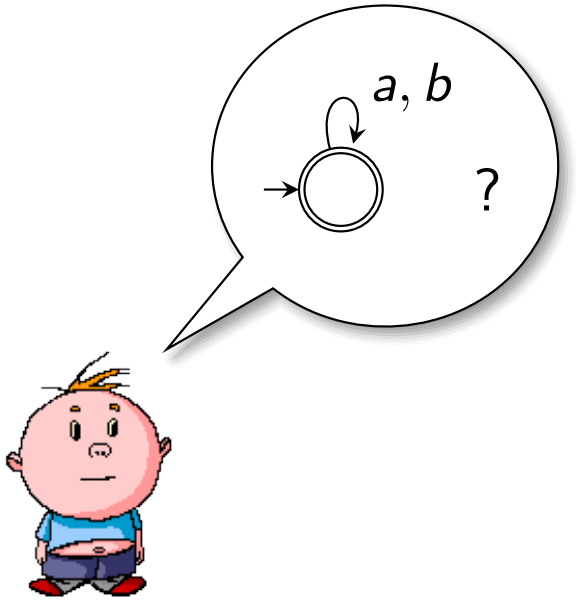


Learning algorithm

Teacher

- Membership query $u \in L$?

Angluin's Active Learning Setting



Learning algorithm

Teacher

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

Angluin's Active Learning Setting

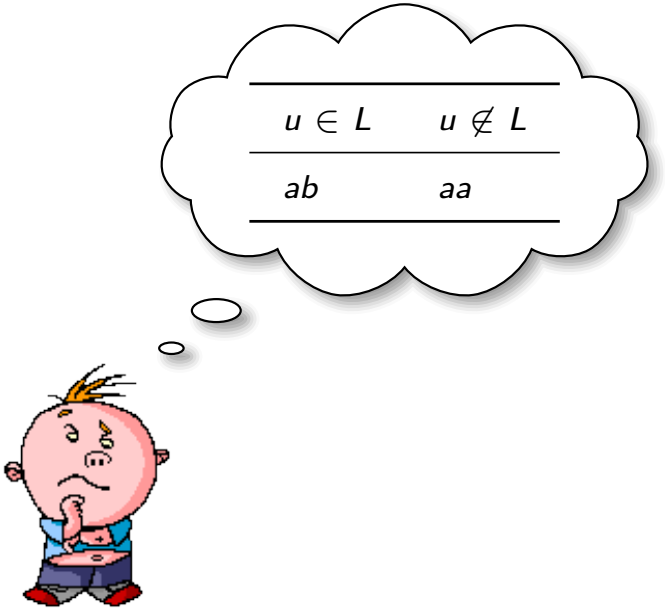
No, counter-example $u = aa$!

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 $L(\mathcal{A}) = L$?

Angluin's Active Learning Setting

Learning algorithm

Teacher

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

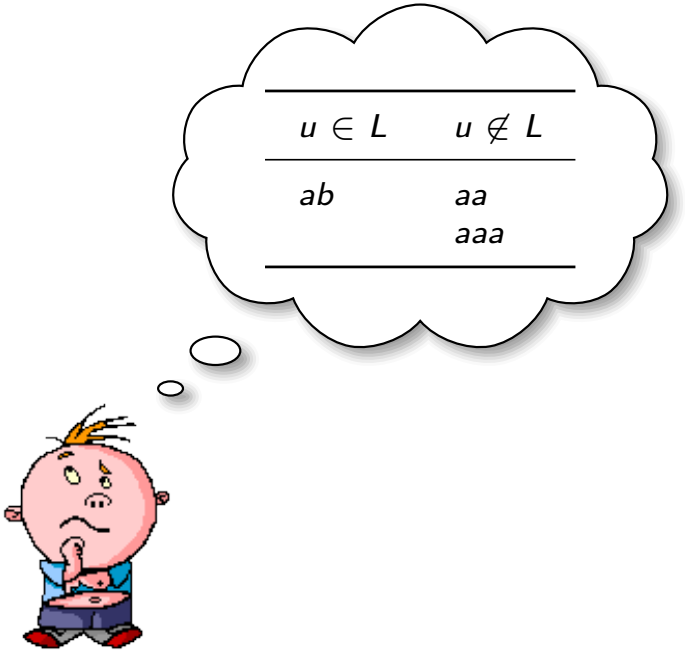
Angluin's Active Learning Setting

Learning algorithm

Teacher

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

Angluin's Active Learning Setting

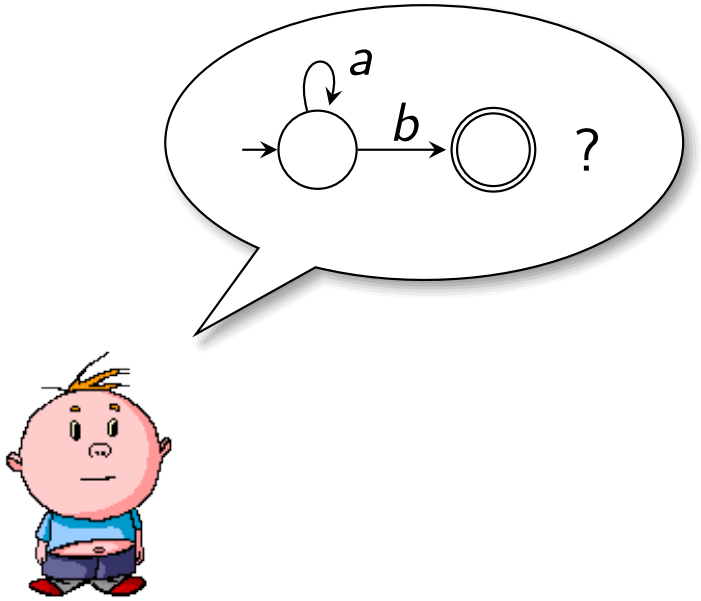


Learning algorithm

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 $L(\mathcal{A}) = L$?

Angluin's Active Learning Setting

Learning algorithm

Teacher

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

Algorithm 3: Naïve active learning algorithm

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

```
Fix a total order ≺ on DFAs (e.g., canonical order of their
  string-representation)
i ← 0
repeat
    Let A_i be the i-th DFA in the order ≺
    Pose an equivalence query on A_i
until the teacher replies "yes" on an equivalence query
return A_i
```

Algorithm 3: Naïve active learning algorithm

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

```
Fix a total order ≺ on DFAs (e.g., canonical order of their
  string-representation)
i ← 0
repeat
    Let A_i be the i-th DFA in the order ≺
    Pose an equivalence query on A_i
until the teacher replies "yes" on an equivalence query
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 $L \subseteq \Sigma$

```
Initialize an empty sample 𝒮 = (S_+, S_−) with S_+ = S_− = ∅
repeat
    Compute a (minimal) DFA 𝒜_𝒮 that is consistent with 𝒮
    Pose an equivalence query on 𝒜
    if the teacher replies with a counterexample u then
        Add u to 𝒮
    end
until the teacher replies "yes" on an equivalence query
return 𝒜_𝒮
```

Theorem

Iterative Passive Learning of minimal consistent DFAs is guaranteed to produce the minimal DFA accepting the target language 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 the 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 the the size of the minimal DFA accepting L
- Let $\mathcal{A}_0, \mathcal{A}_1, \ldots$ be the sequence of DFA produced during the learning process
  - $|\mathcal{A}_i| \leq |\mathcal{A}_{i+1}|$ for all $i \in \mathbb{N}$
  - $L(\mathcal{A}_i) \neq L(\mathcal{A}_j)$ 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 the the size of the minimal DFA accepting L
- Let $\mathcal{A}_0, \mathcal{A}_1, \ldots$ be the sequence of DFA produced during the learning process
  - $|\mathcal{A}_i| \leq |\mathcal{A}_{i+1}|$ for all $i \in \mathbb{N}$
  - $L(\mathcal{A}_i) \neq L(\mathcal{A}_j)$ for all $i \in \mathbb{N}$ and $j < i$
- Thus, the minimal DFA accepting L is eventually produced

[7] D. Angluin: *Negative Results for Equivalence Queries*, 1990

Theorem (Angluin [7], 1990)

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

[7] D. Angluin: *Negative Results for Equivalence Queries*, 1990

[8] D. Angluin: *A Note on the Number of Queries Needed to Identify Regular Language*, 1981

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

Learning in Polynomial Time

[7] D. Angluin: *Negative Results for Equivalence Queries*, 1990

[8] D. Angluin: *A Note on the Number of Queries Needed to Identify Regular Language*, 1981

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^*)

Myhill-Nerode Congruence

Let $L \subseteq \Sigma^*$ be a language.

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

$$uw \in L \Leftrightarrow vw \in L$$

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

Myhill-Nerode Congruence

Let $L \subseteq \Sigma^*$ be a language.

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

$$uw \in L \;\Leftrightarrow\; vw \in L$$

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

- $\sim_L$ is a *congruence* with respect to concatenation, meaning

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

Myhill-Nerode Congruence

Let $L \subseteq \Sigma^*$ be a language.

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

$$uw \in L \;\Leftrightarrow\; vw \in L$$

  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

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

Myhill-Nerode Congruence

Let $L \subseteq \Sigma^*$ be a language.

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

$$uw \in L \;\Leftrightarrow\; vw \in L$$

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

- $\sim_L$ is a *congruence* with respect to concatenation, meaning

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

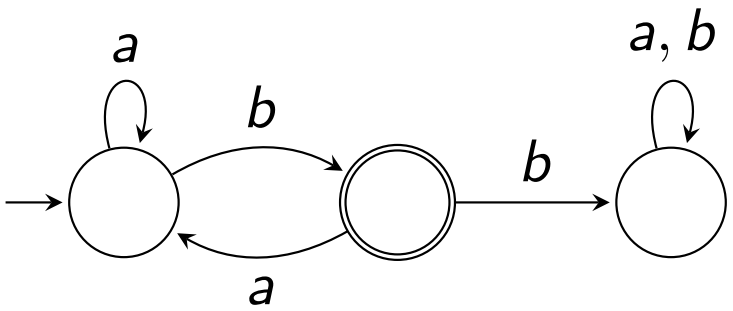
- The *L-equivalence class* of $u \in \Sigma^*$ is defined by

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

- The number of equivalence classes of L is defined by

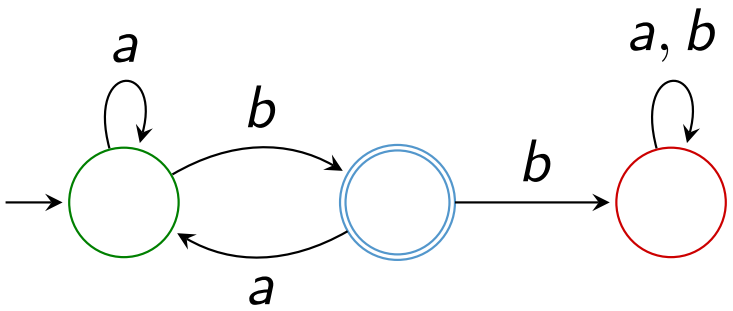
$$index(\sim_L) = |\{[\![u]\!]_L \mid u \in \Sigma^*\}|$$

Hankel Matrix



	ε	a	b	aa	ab	ba	bb	aaa	aab	$\cdots$
ε	0	0	1	0	1	0	0	0	1	$\cdots$
a	0	0	1	0	1	0	0	0	1	$\cdots$
b	1	0	0	0	1	0	0	0	1	$\cdots$
aa	0	0	1	0	1	0	0	0	1	$\cdots$
ab	1	0	0	0	1	0	0	0	1	$\cdots$
ba	0	0	1	0	1	0	0	0	1	$\cdots$
bb	0	0	0	0	0	0	0	0	0	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Hankel Matrix



	ε	a	b	aa	ab	ba	bb	aaa	aab	$\cdots$
ε	0	0	1	0	1	0	0	0	1	$\cdots$
a	0	0	1	0	1	0	0	0	1	$\cdots$
b	1	0	0	0	1	0	0	0	1	$\cdots$
aa	0	0	1	0	1	0	0	0	1	$\cdots$
ab	1	0	0	0	1	0	0	0	1	$\cdots$
ba	0	0	1	0	1	0	0	0	1	$\cdots$
bb	0	0	0	0	0	0	0	0	0	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Theorem (Myhill and Nerode)

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

Theorem (Myhill and Nerode)

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

Proof (sketch)

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

Observation Table

An *observation table* is a triple $O = (R, S, T)$ where

Example

	ε	a	ba
ε	1	0	1
a	0	1	0
b	1	1	0
aa	1	0	0
ab	1	0	1

Observation Table

An *observation table* is a triple $O = (R, S, T)$ where

- R is a nonempty, finite, prefix-closed set of *representatives*

Example

	ε	a	ba
ε	1	0	1
a	0	1	0
b	1	1	0
aa	1	0	0
ab	1	0	1

Observation 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 ε

Example

	ε	a	ba
ε	1	0	1
a	0	1	0
b	1	1	0
aa	1	0	0
ab	1	0	1

Observation 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 ε
- $T : (R \cup R \cdot \Sigma) \cdot S \to \{0, 1\}$ stores the table entries

Example

	ε	a	ba
ε	1	0	1
a	0	1	0
b	1	1	0
aa	1	0	0
ab	1	0	1

Observation 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 ε
- $T : (R \cup R \cdot \Sigma) \cdot S \to \{0, 1\}$ 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 $w \in S$

Observation 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 ε
- $T \colon (R \cup R \cdot \Sigma) \cdot S \to \{0, 1\}$ 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

$$[\![u]\!]_O = \{v \in R \cup R \cdot \Sigma \mid u \sim_O v\}$$

Observation Table – Closedness

An observation table 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 $ua \sim_O v$

Observation Table – Closedness

An observation table 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 $ua \sim_O v$

Example

	ε	a	ba
ε	1	0	1
a	0	1	0
b	1	1	0
aa	1	0	0
ab	1	0	1

Observation Table – Closedness

An observation table 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

	ε	a	ba
ε	1	0	1
a	0	1	0
b	1	1	0
aa	1	0	0
ab	1	0	1

	ε	a	ba
ε	1	0	1
a	0	1	0
b	1	1	0
aa	1	0	0
ab	1	0	1
ba	1		
bb		0	

Observation Table – Consistency

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 $ua \sim_O va$

Observation Table – Consistency

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

	ε	a	ba
ε	1	0	1
a	0	1	0
b	1	1	0
aa	1	0	0
ab	1	0	1

Observation Table – Consistency

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

	ε	a	ba
ε	1	0	1
b	1	0	1
a	0	1	0
ba	0	0	0
bb	0	1	1

Observation Table – Consistency

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

	ε	a	ba
ε	1	0	1
b	1	0	1
a	0	1	0
ba	0	0	0
bb	0	1	1

	ε	a	ba	aa
ε	1	0	1	1
b	1	0	1	0
a	0	1	0	
ba	0	0	0	
bb	0	1	1	

Hypothesis DFA

If an observation table O is closed and consistent, then $\sim_O$ is a congruence and one can define a DFA $\mathcal{A}_O = (Q, \Sigma, q_0, \delta, F)$ by

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

If an observation table O is closed and consistent, then $\sim_O$ is a congruence and one can define a DFA $\mathcal{A}_O = (Q, \Sigma, q_0, \delta, F)$ 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)

	ε	b
ε	0	1
a	1	0
b	1	0
aa	0	1
ab	0	1

If an observation table O is closed and consistent, then $\sim_O$ is a congruence and one can define a DFA $\mathcal{A}_O = (Q, \Sigma, q_0, \delta, F)$ 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)

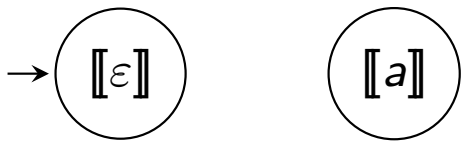
	ε	b
ε	0	1
a	1	0
b	1	0
aa	0	1
ab	0	1

If an observation table O is closed and consistent, then $\sim_O$ is a congruence and one can define a DFA $\mathcal{A}_O = (Q, \Sigma, q_0, \delta, F)$ 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)

	ε	b
ε	0	1
a	1	0
b	1	0
aa	0	1
ab	0	1

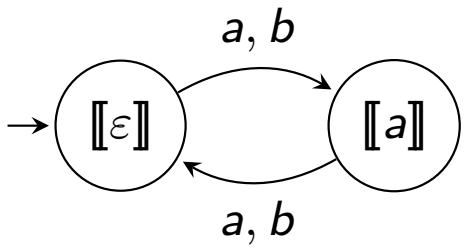


If an observation table O is closed and consistent, then $\sim_O$ is a congruence and one can define a DFA $\mathcal{A}_O = (Q, \Sigma, q_0, \delta, F)$ by

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

Example (Construction)

	ε	b
ε	0	1
a	1	0
b	1	0
aa	0	1
ab	0	1

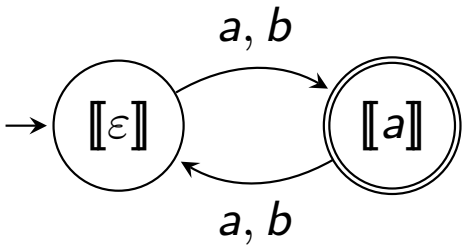


If an observation table O is closed and consistent, then $\sim_O$ is a congruence and one can define a DFA $\mathcal{A}_O = (Q, \Sigma, q_0, \delta, F)$ by

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

Example (Construction)

	ε	b
ε	0	1
a	1	0
b	1	0
aa	0	1
ab	0	1



Hypothesis DFA

If an observation table O is closed and consistent, then $\sim_O$ is a congruence and one can define a DFA $\mathcal{A}_O = (Q, \Sigma, q_0, \delta, F)$ 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)

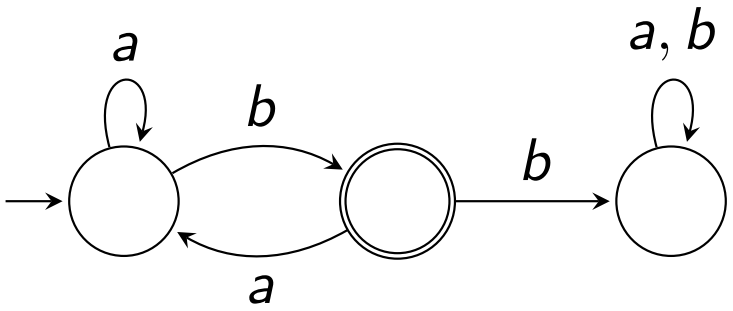
Let $O = (R, S, T)$ be a closed and consistent observation table for a regular language $L \subseteq \Sigma^*$ and $\mathcal{A}_O$ as defined above. Then, $\mathcal{A}_O$ is correct on all representatives $u \in R$ (i.e., $\mathcal{A}_O$ satisfies $u \in L(\mathcal{A}_O)$ if and only if $u \in L$ for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

[9] Dana Angluin: *Learning Regular Sets From Queries and Counterexamples*, 1987

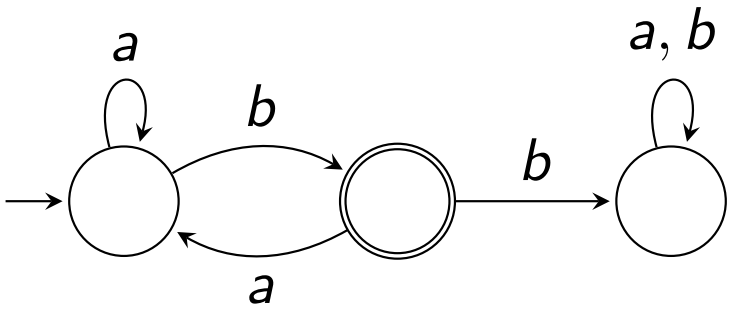
Algorithm 5: Angluin's Algorithm (L^*)

```
Create an observation table O = (R, S, T) with R = S = {ε}
repeat
    Make O closed and consistent (using membership queries)
    Construct A_O and perform an equivalence query
    if the teacher returns a counterexample u then
        R ← R ∪ Pref(u)
        update O (using membership queries)
    end
until the teacher replies "yes" on an equivalence query
return A_O
```

Example

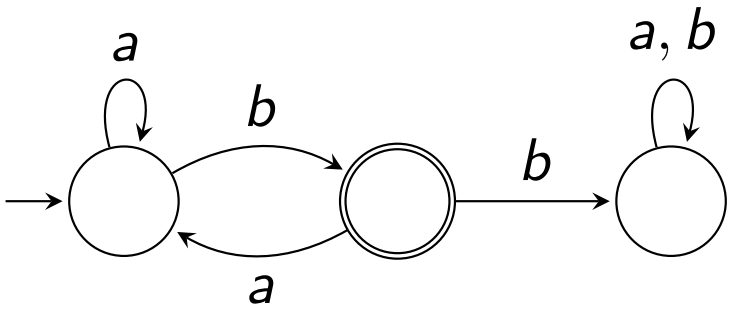


Example



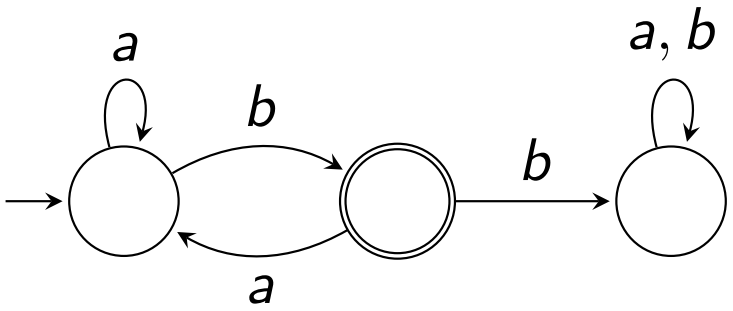
	ε
ε	0
b	1
a	0

Example



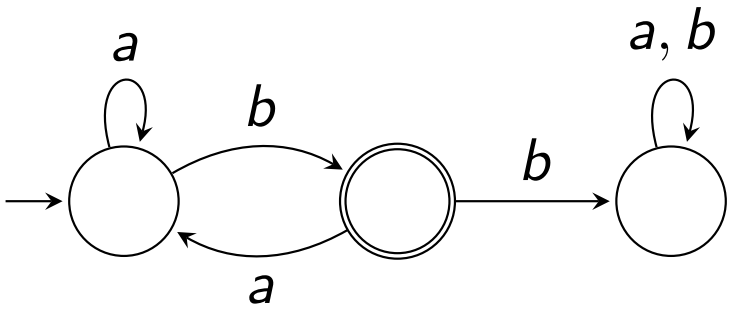
	ε
ε	0
b	1
a	0

Example



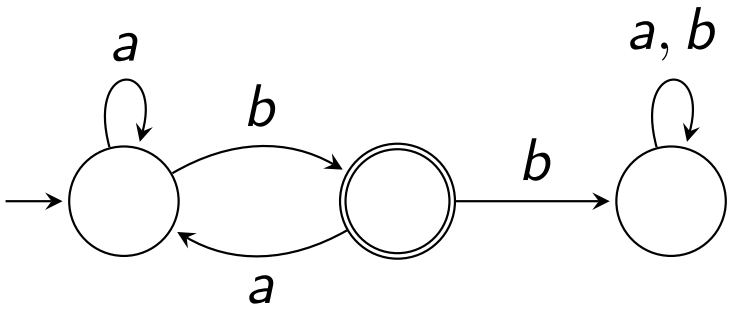
	ε
ε	0
b	1
a	0
ba	
bb	

Example

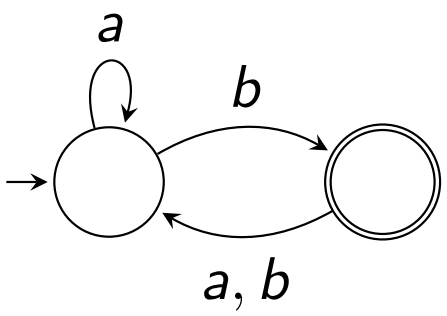


	ε
ε	0
b	1
a	0
ba	0
bb	0

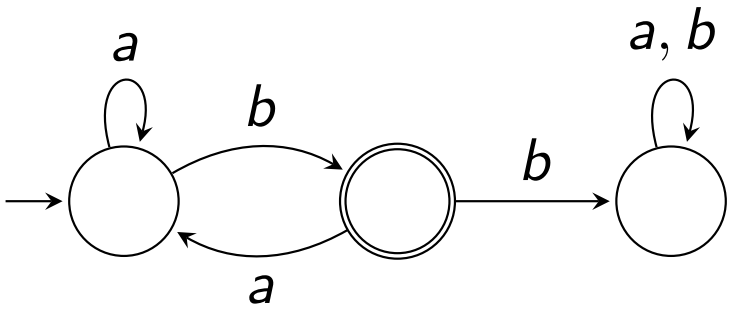
Example



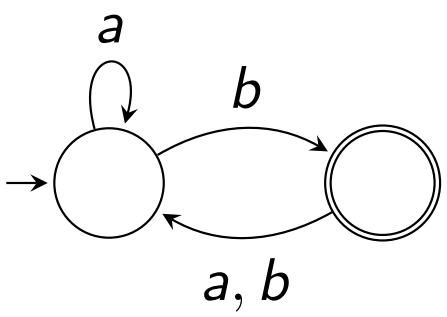
	ε
ε	0
b	1
a	0
ba	0
bb	0



Example

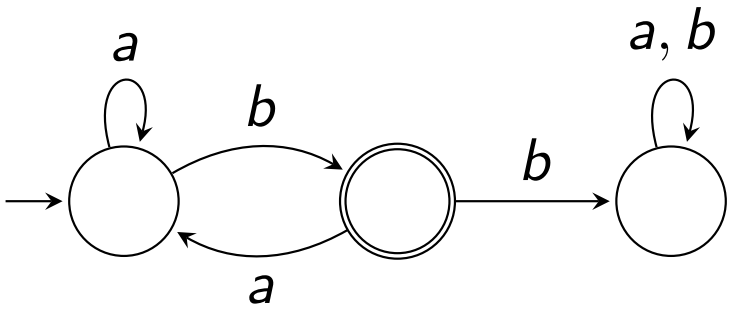


	ε
ε	0
b	1
a	0
ba	0
bb	0



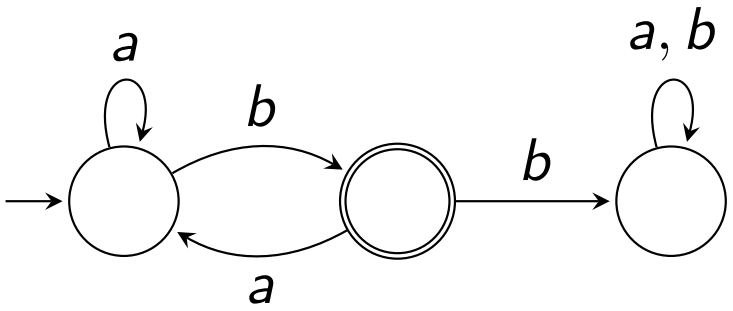
Counterexample bbb

Example



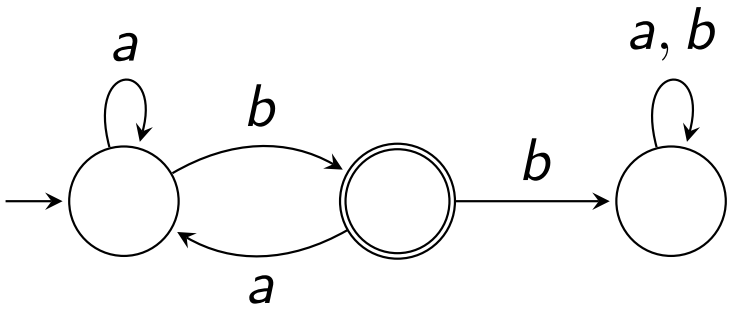
	ε
ε	0
b	1
bb	0
bbb	
a	0
ba	0
bba	
$bbba$	
$bbbb$	

Example



	ε
ε	0
b	1
bb	0
bbb	0
a	0
ba	0
bba	0
$bbba$	0
$bbbb$	0

Example

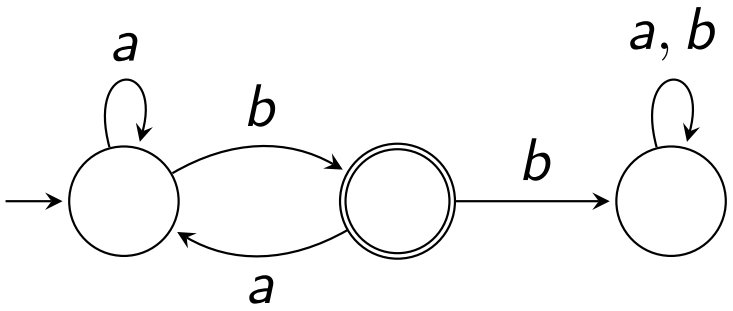


	ε
ε	0
b	1
bb	0
bbb	0
a	0
ba	0
bba	0
$bbba$	0
$bbbb$	0

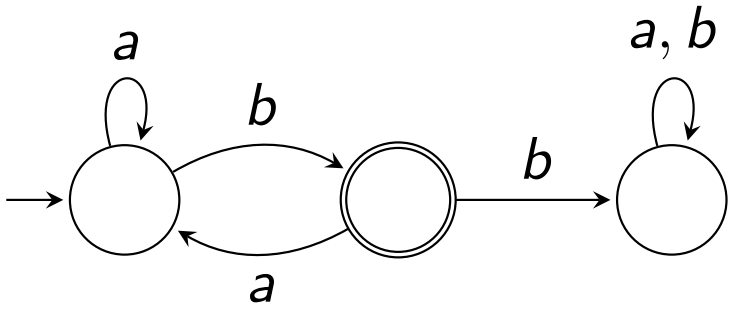
b

b

Example



	ε	b
ε	0	1
b	1	0
bb	0	0
bbb	0	0
a	0	1
ba	0	1
bba	0	0
$bbba$	0	0
$bbbb$	0	0



[9] D. Angluin: *Learning Regular Sets From Queries and Counterexamples*, 1987

Lemma (Angluin [9], 1987)

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

[9] D. Angluin: *Learning Regular Sets From Queries and Counterexamples*, 1987

Lemma (Angluin [9], 1987)

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

Theorem (Angluin [9], 1987)

Given a teacher for a regular target language $L \subseteq \Sigma^*$, Angluin's algorithm learns a DFA isomorphic to $\mathcal{A}_L$ in time polynomial in the size n of $\mathcal{A}_L$ and the length m of the longest counterexample returned by the teacher. It asks $\mathcal{O}(n)$ equivalence queries and $\mathcal{O}(mn^2)$ membership queries.

2.2.

Kearns and Vazirani's Algorithm

Improvements of Angluin's Algorithm

	ε	b
ε	0	1
b	1	0
bb	0	0
bbb	0	0
a	0	1
ba	0	1
bba	0	0
$bbba$	0	0
$bbbb$	0	0

Improvements of Angluin's Algorithm

	ε	b
ε	0	1
b	1	0
bb	0	0
~~bbb~~	~~0~~	~~0~~
a	0	1
ba	0	1
bba	0	0
$bbba$	0	0
$bbbb$	0	0

Improvements of Angluin's Algorithm

	ε	b
ε	0	1
b	1	0
bb	0	0
~~bbb~~	~~0~~	~~0~~
a	0	1
ba	0	1
bba	0	0
$bbba$	0	0
$bbbb$	0	0

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

Improvements of Angluin's Algorithm

	ε	b
ε	0	1
b	1	~~0~~
bb	0	0
~~bbb~~	~~0~~	~~0~~
a	0	1
ba	0	1
bba	0	0
$bbba$	0	0
$bbbb$	0	0

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

Improvements of Angluin's Algorithm

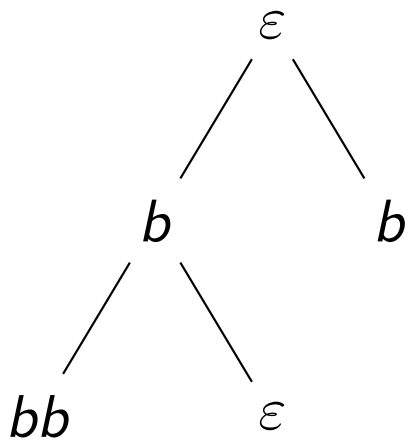
	ε	b
ε	0	1
b	1	~~0~~
bb	0	0
~~bbb~~	~~0~~	~~0~~
a	0	1
ba	0	1
bba	0	0
$bbba$	0	0
$bbbb$	0	0

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

Classification Tree

A *classification tree t* is a binary tree

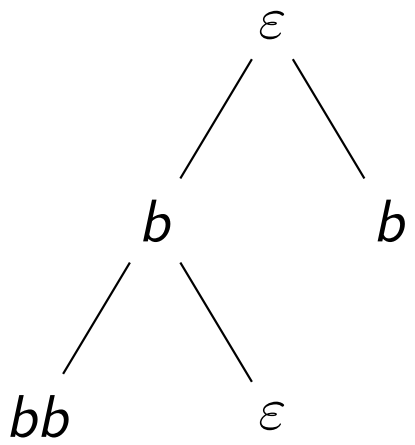
- whose inner nodes are labeled with *separating words* from a set S
- whose leaf nodes are labeled with *representatives* from a set R
- whose root and one leaf is labeled with ε (i.e., $\varepsilon \in S$ and $\varepsilon \in R$)



Classification Tree

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 ε (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.

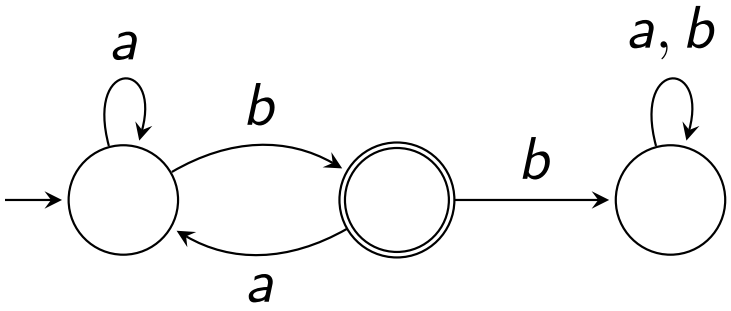
Sifting

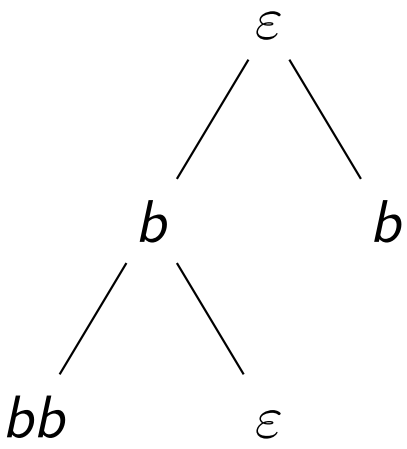
Starting at the root, recursively descent at an inner node labeled with $v \in S$

- *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.

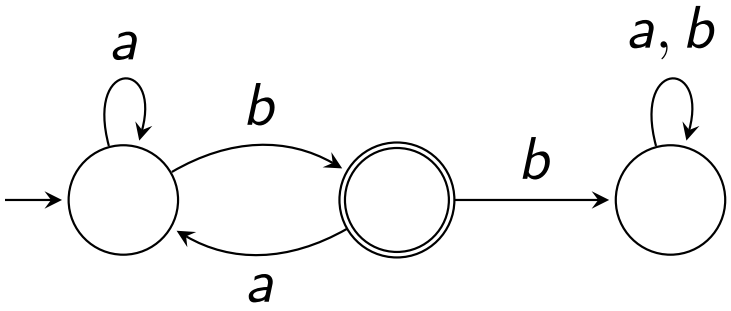
Sifting

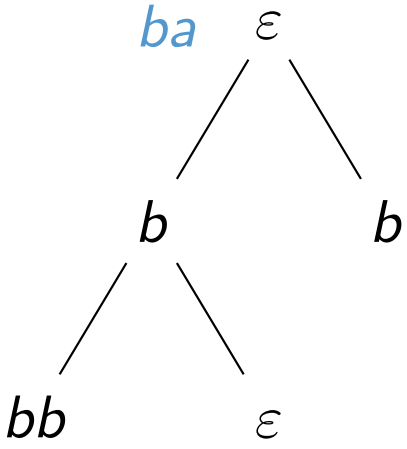
Starting at the root, recursively descent at an inner node labeled with $v \in S$

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

Example

Let $u = ba$





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

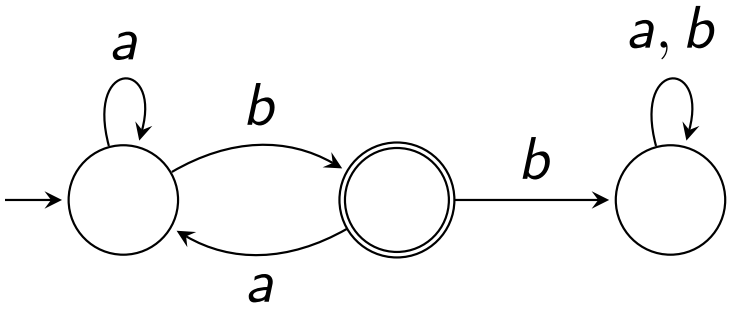
Sifting

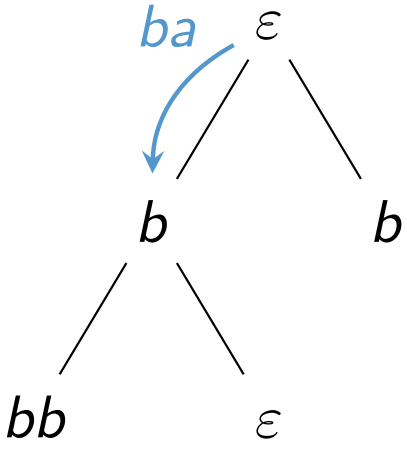
Starting at the root, recursively descent at an inner node labeled with $v \in S$

- *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.

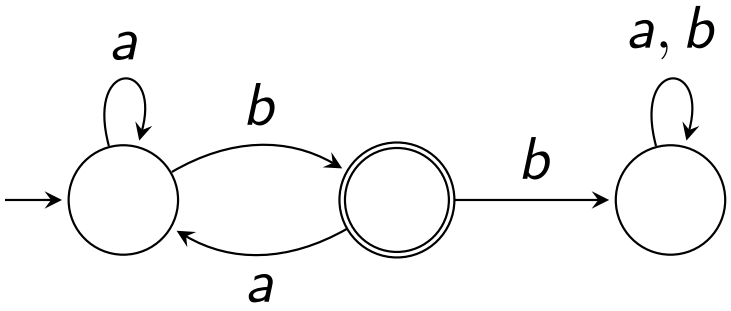
Sifting

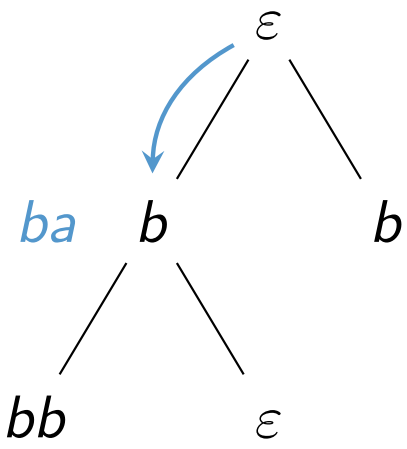
Starting at the root, recursively descent at an inner node labeled with $v \in S$

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

Example

Let $u = ba$





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

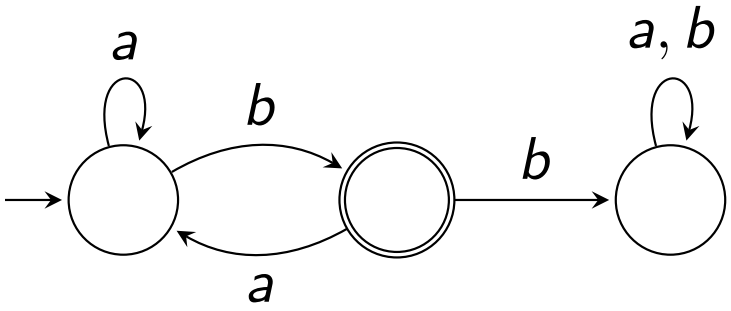
Sifting

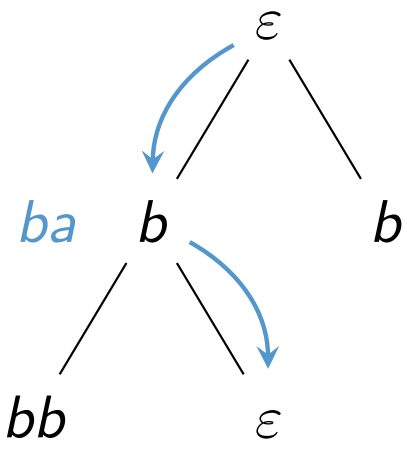
Starting at the root, recursively descent at an inner node labeled with $v \in S$

- *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.

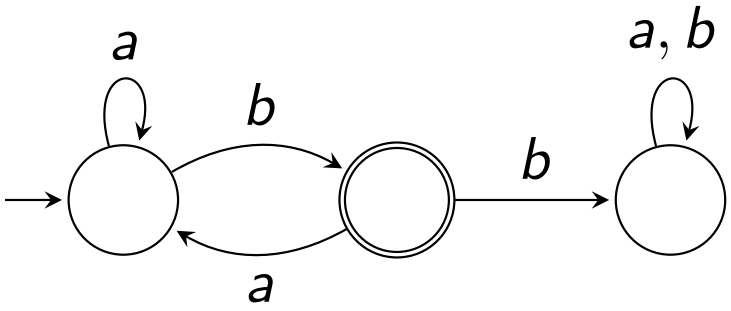
Sifting

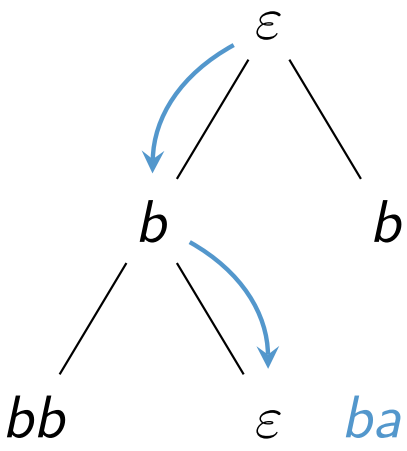
Starting at the root, recursively descent at an inner node labeled with $v \in S$

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

Example

Let $u = ba$





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 $sift_t(u)$ denote the representative at the leaf node reached by sifting u down t

Note: Sifting invokes membership queries

Hypothesis DFA

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

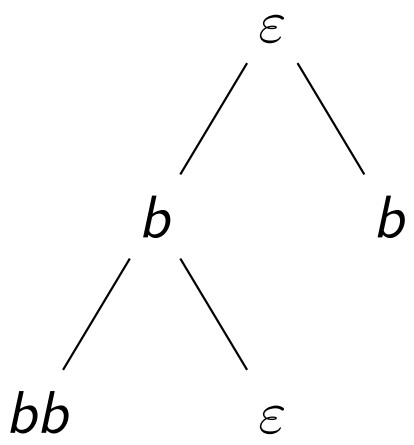
- $Q = R$
- $q_0 = \varepsilon$
- $\delta(u, a) = \mathit{sift}_t(ua)$
- $F = \{u \in R \mid u \in L\}$ (i.e., all representatives in the right subtree)

Hypothesis DFA

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

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

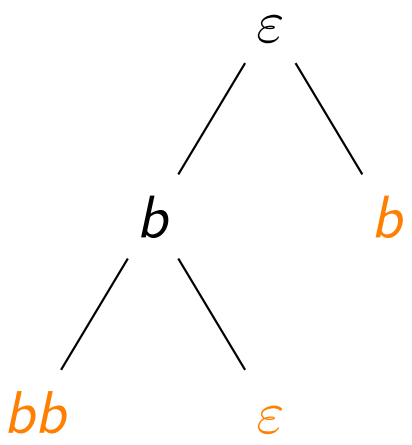
Example (Construction)

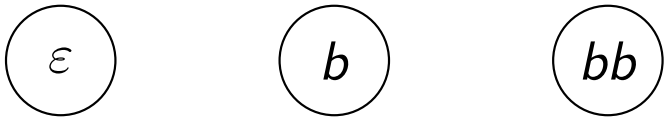


Given a classification tree t, we define the DFA $\mathcal{A}_t = (Q, \Sigma, q_0, \delta, F)$ by

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

Example (Construction)

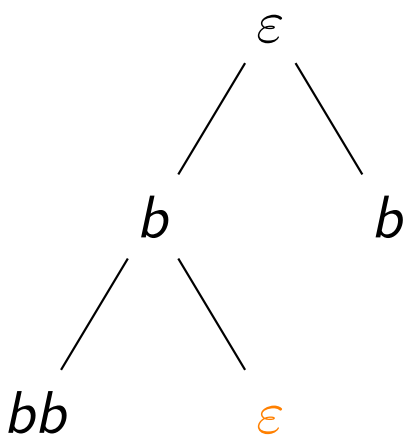


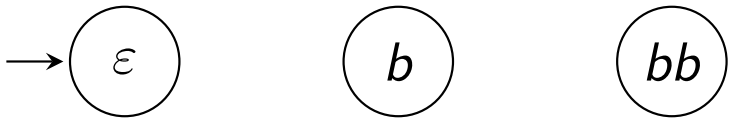


Given a classification tree t, we define the DFA $\mathcal{A}_t = (Q, \Sigma, q_0, \delta, F)$ by

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

Example (Construction)

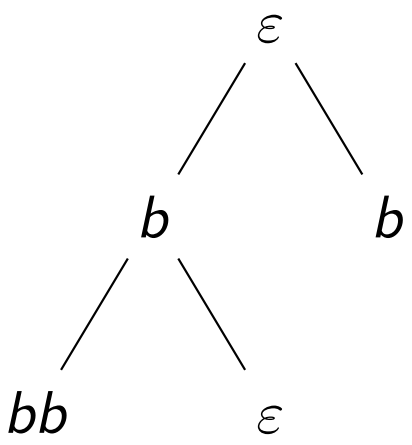


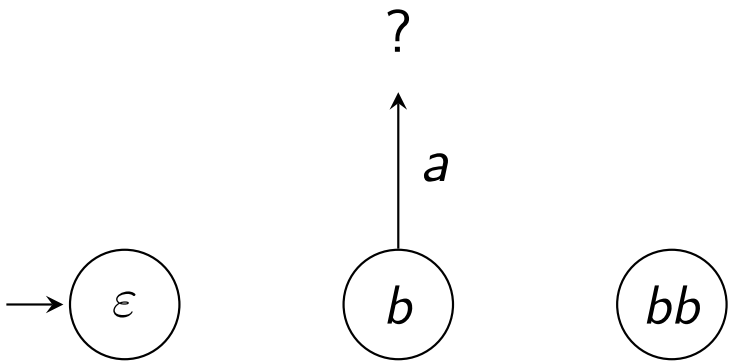


Given a classification tree t, we define the DFA $\mathcal{A}_t = (Q, \Sigma, q_0, \delta, F)$ by

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

Example (Construction)

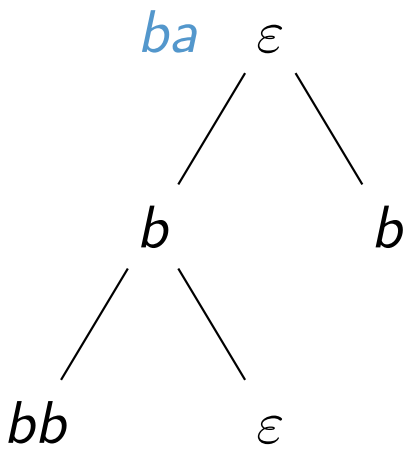


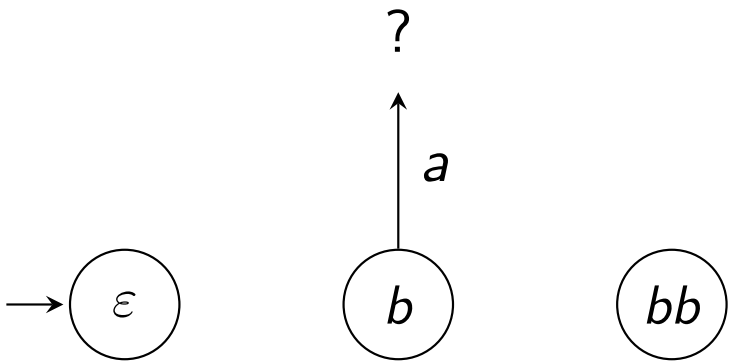


Given a classification tree t, we define the DFA $\mathcal{A}_t = (Q, \Sigma, q_0, \delta, F)$ by

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

Example (Construction)

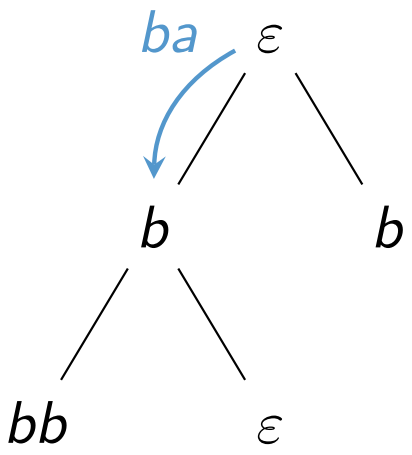


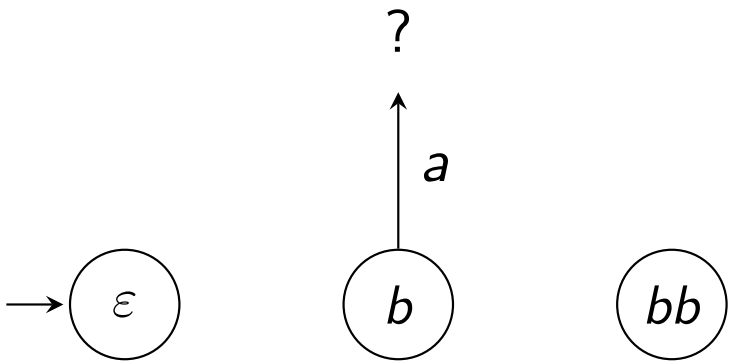


Given a classification tree t, we define the DFA $\mathcal{A}_t = (Q, \Sigma, q_0, \delta, F)$ by

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

Example (Construction)

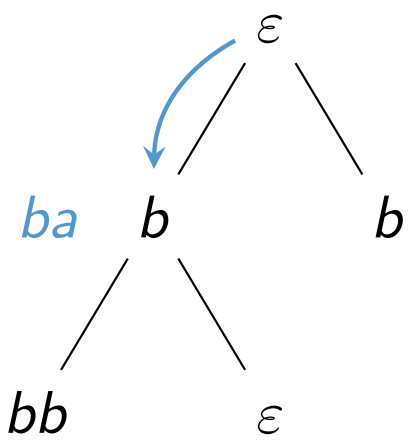


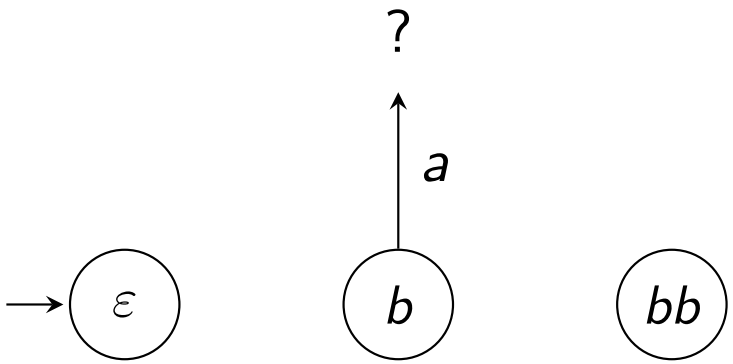


Given a classification tree t, we define the DFA $\mathcal{A}_t = (Q, \Sigma, q_0, \delta, F)$ by

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

Example (Construction)

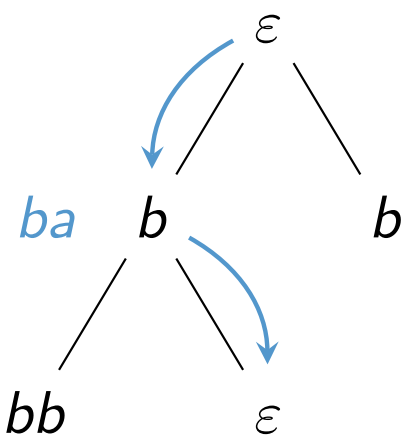


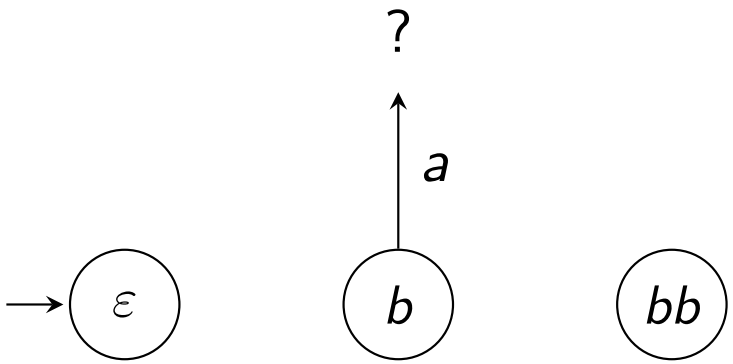


Given a classification tree t, we define the DFA $\mathcal{A}_t = (Q, \Sigma, q_0, \delta, F)$ by

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

Example (Construction)

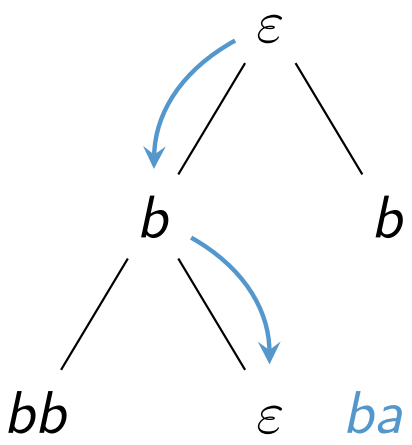


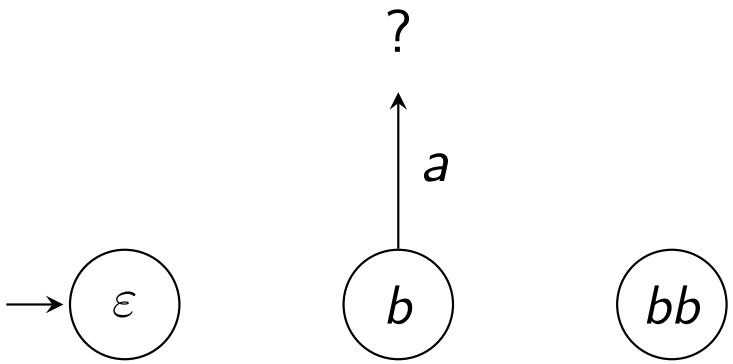


Given a classification tree t, we define the DFA $\mathcal{A}_t = (Q, \Sigma, q_0, \delta, F)$ by

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

Example (Construction)



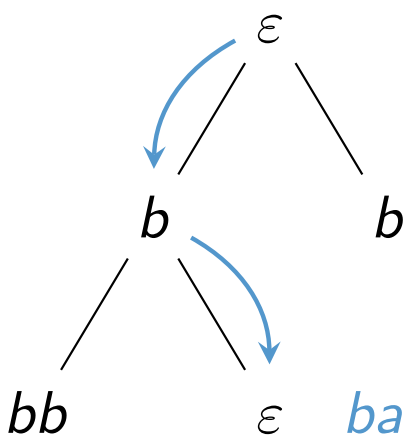


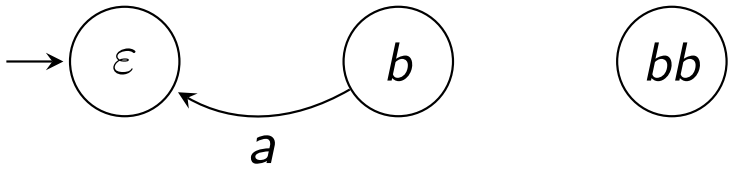
Hypothesis DFA

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

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

Example (Construction)

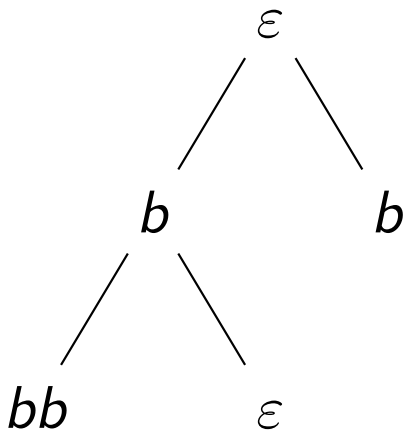


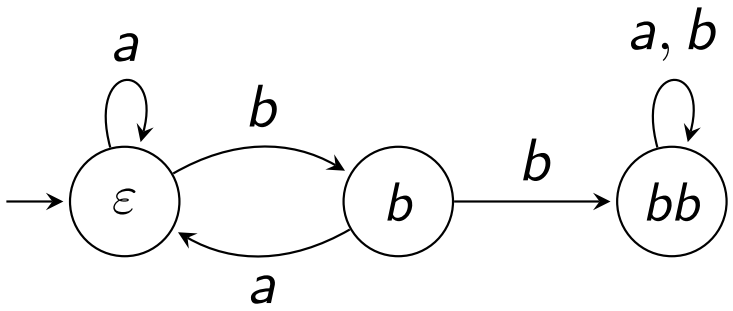


Given a classification tree t, we define the DFA $\mathcal{A}_t = (Q, \Sigma, q_0, \delta, F)$ by

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

Example (Construction)

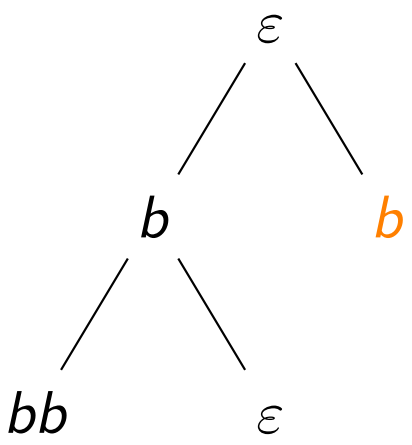


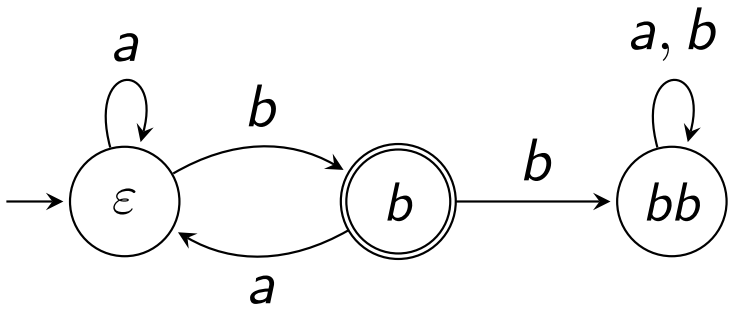


Given a classification tree t, we define the DFA $\mathcal{A}_t = (Q, \Sigma, q_0, \delta, F)$ by

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

Example (Construction)





Analyzing Counterexamples

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

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

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

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

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

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

Breakpoint Position

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

$$u_{i-1} a_i \dots a_m \in L \;\Leftrightarrow\; u_i a_{i+1} \dots a_m \notin L$$

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

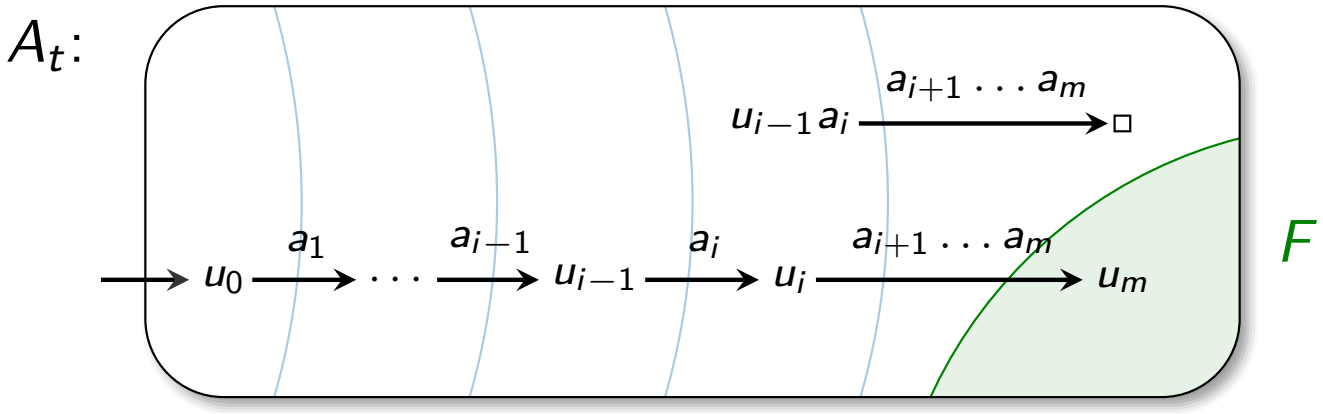
$$\mathcal{A}_t \colon u_0 \xrightarrow{a_1} u_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} u_m$$

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

Breakpoint Position

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

$$u_{i-1} a_i \dots a_m \in L \;\Leftrightarrow\; u_i a_{i+1} \dots a_m \notin L$$



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

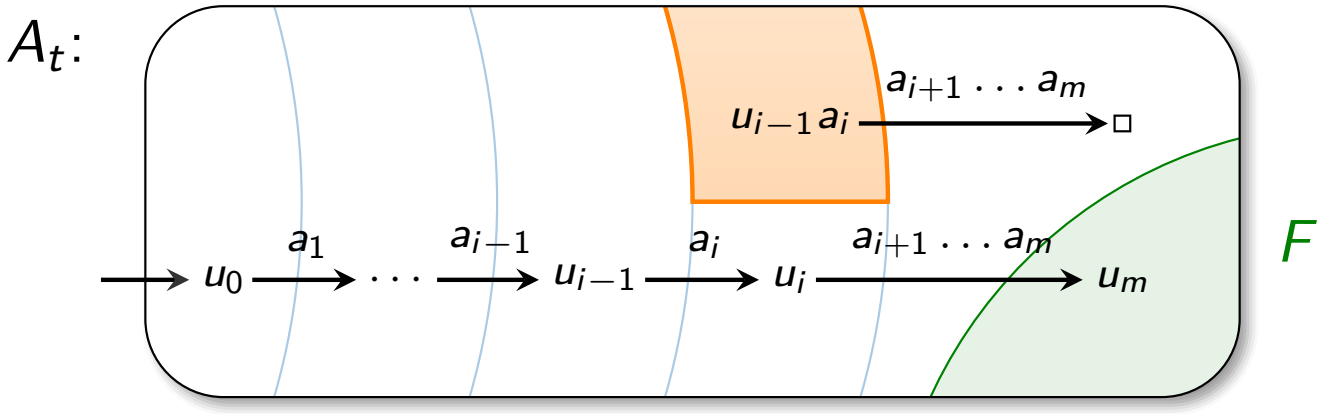
$$\mathcal{A}_t \colon u_0 \xrightarrow{a_1} u_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} u_m$$

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

Breakpoint Position

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

$$u_{i-1} a_i \dots a_m \in L \;\Leftrightarrow\; u_i a_{i+1} \dots a_m \notin L$$



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

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

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

Breakpoint Position

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

$$u_{i-1} a_i \dots a_m \in L \;\Leftrightarrow\; u_i a_{i+1} \dots a_m \notin L$$

A Breakpoint Position Always Exists

Since v is a counterexample, we know

$$v \in L \;\Leftrightarrow\; v \notin L(\mathcal{A}_t)$$

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

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

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

Breakpoint Position

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

$$u_{i-1} a_i \dots a_m \in L \;\Leftrightarrow\; u_i a_{i+1} \dots a_m \notin L$$

A Breakpoint Position Always Exists

Since v is a counterexample, we know

$$v \in L \;\Leftrightarrow\; v \notin L(\mathcal{A}_t) \;\Leftrightarrow\; u_m \notin F$$

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

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

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

Breakpoint Position

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

$$u_{i-1} a_i \dots a_m \in L \;\Leftrightarrow\; u_i a_{i+1} \dots a_m \notin L$$

A Breakpoint Position Always Exists

Since v is a counterexample, we know

$$v \in L \;\Leftrightarrow\; v \notin L(\mathcal{A}_t) \;\Leftrightarrow\; u_m \notin F \;\Leftrightarrow\; u_m \notin L$$

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

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

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

Breakpoint Position

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

$$u_{i-1} a_i \dots a_m \in L \;\Leftrightarrow\; u_i a_{i+1} \dots a_m \notin L$$

A Breakpoint Position Always Exists

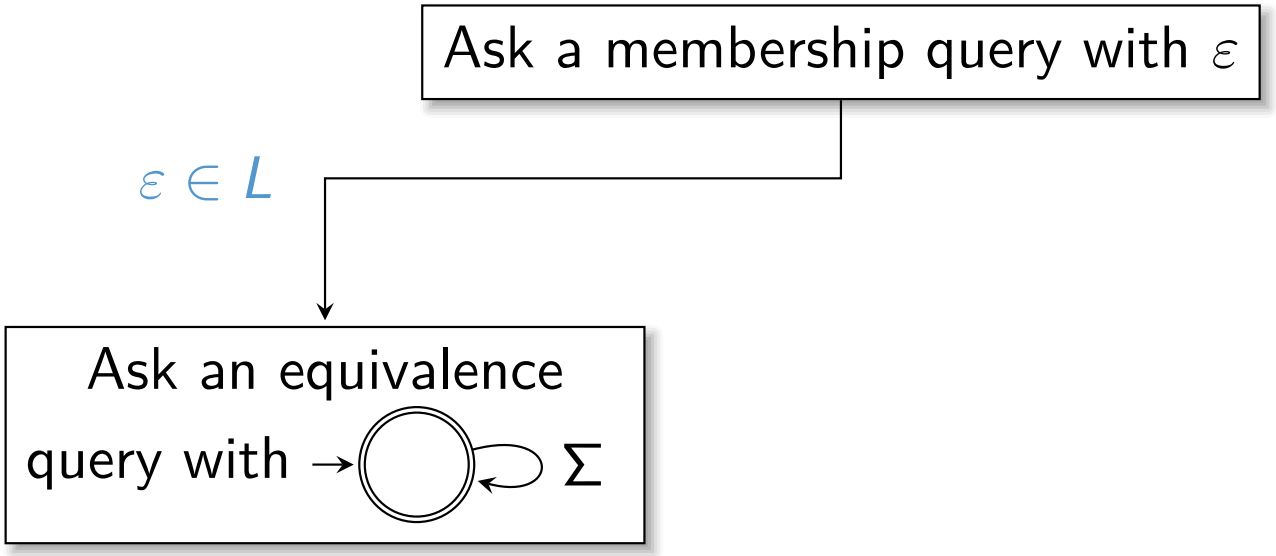
Since v is a counterexample, we know

$$v \in L \;\Leftrightarrow\; v \notin L(\mathcal{A}_t) \;\Leftrightarrow\; u_m \notin F \;\Leftrightarrow\; u_m \notin L$$

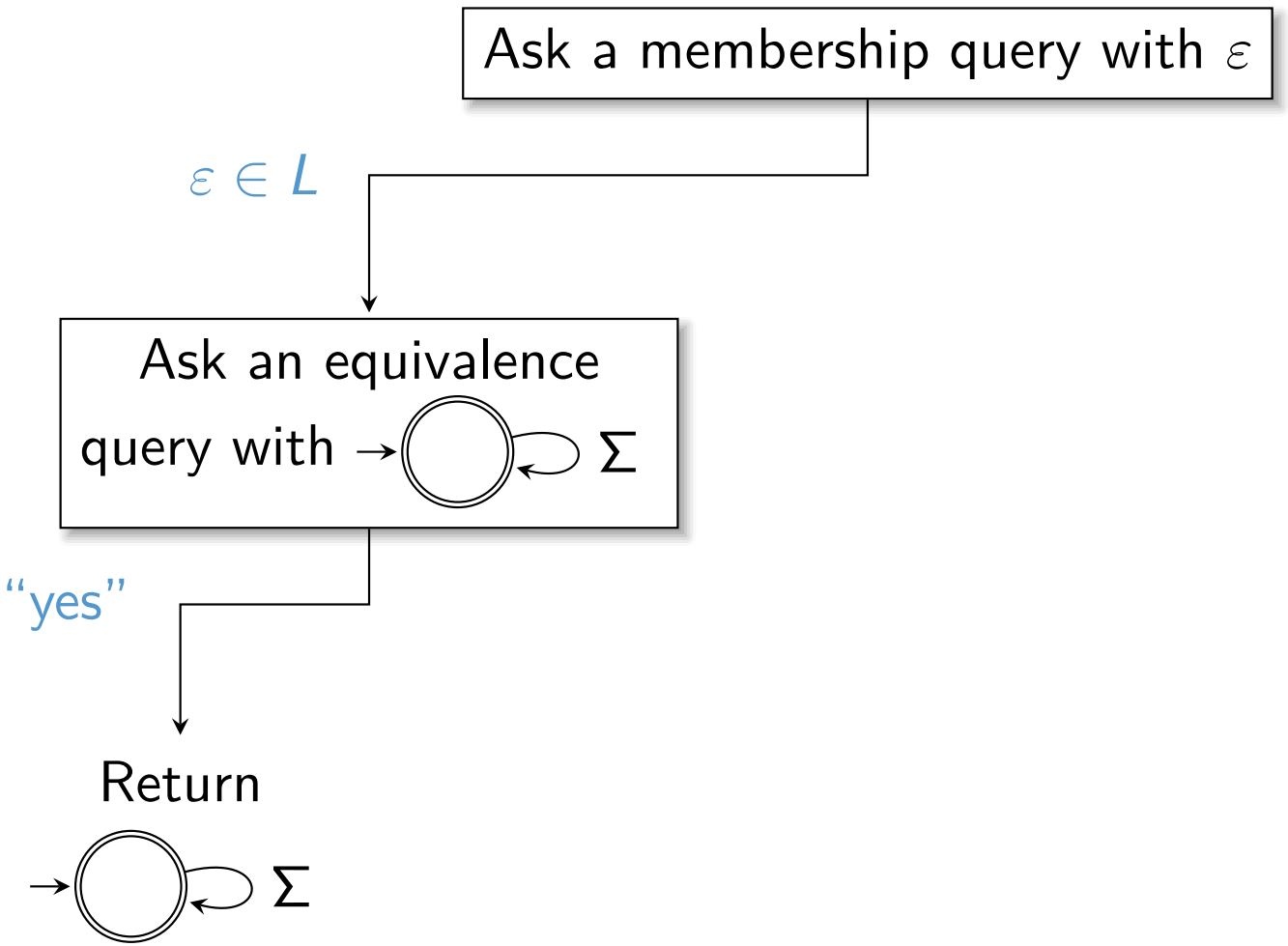
Thus, $v = u_0 a_1 \dots a_m \in L \;\Leftrightarrow\; u_m a_{m+1} \dots a_m = u_m \notin L$

Ask a membership query with ε

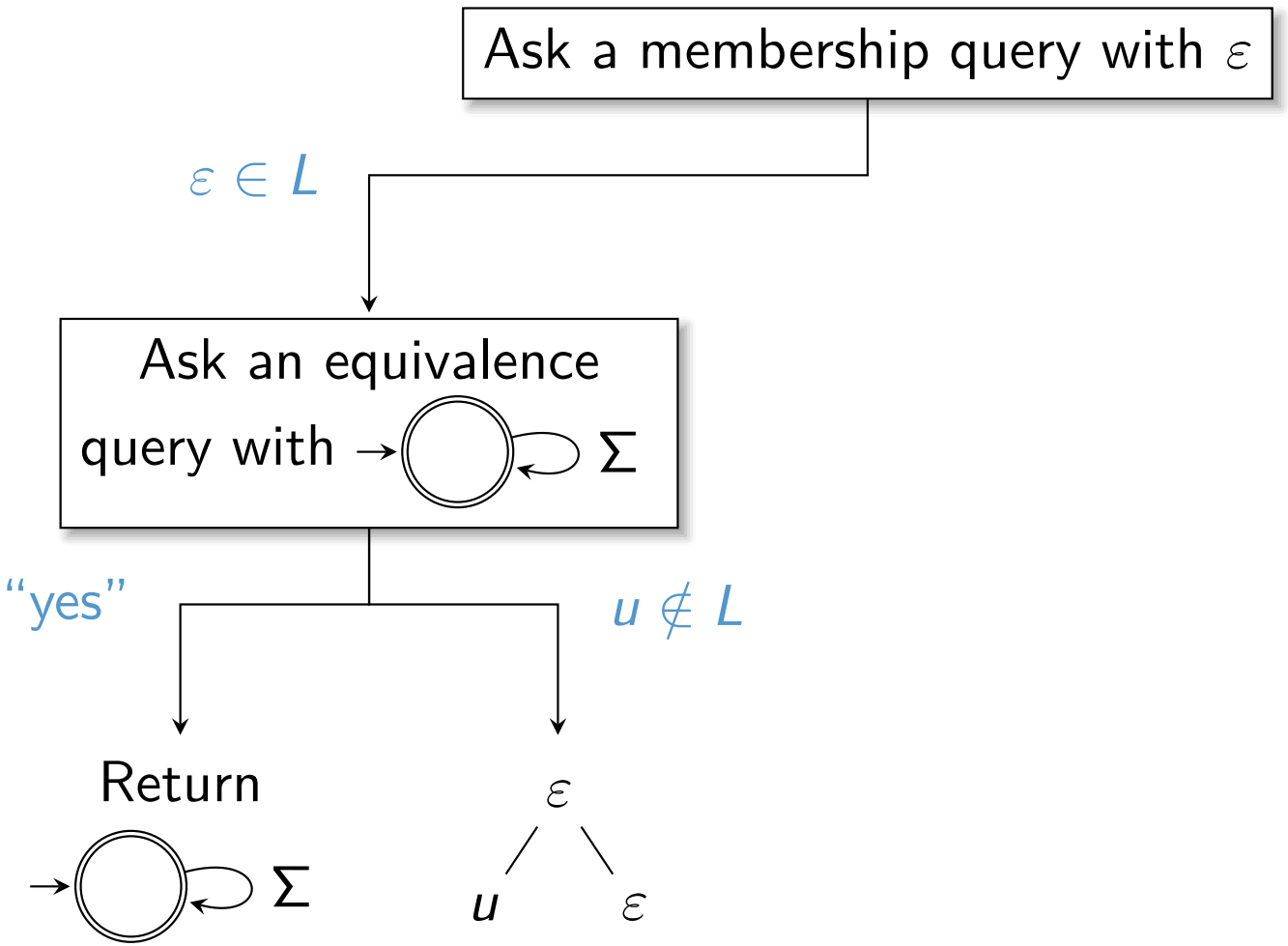
Initial Classification Tree



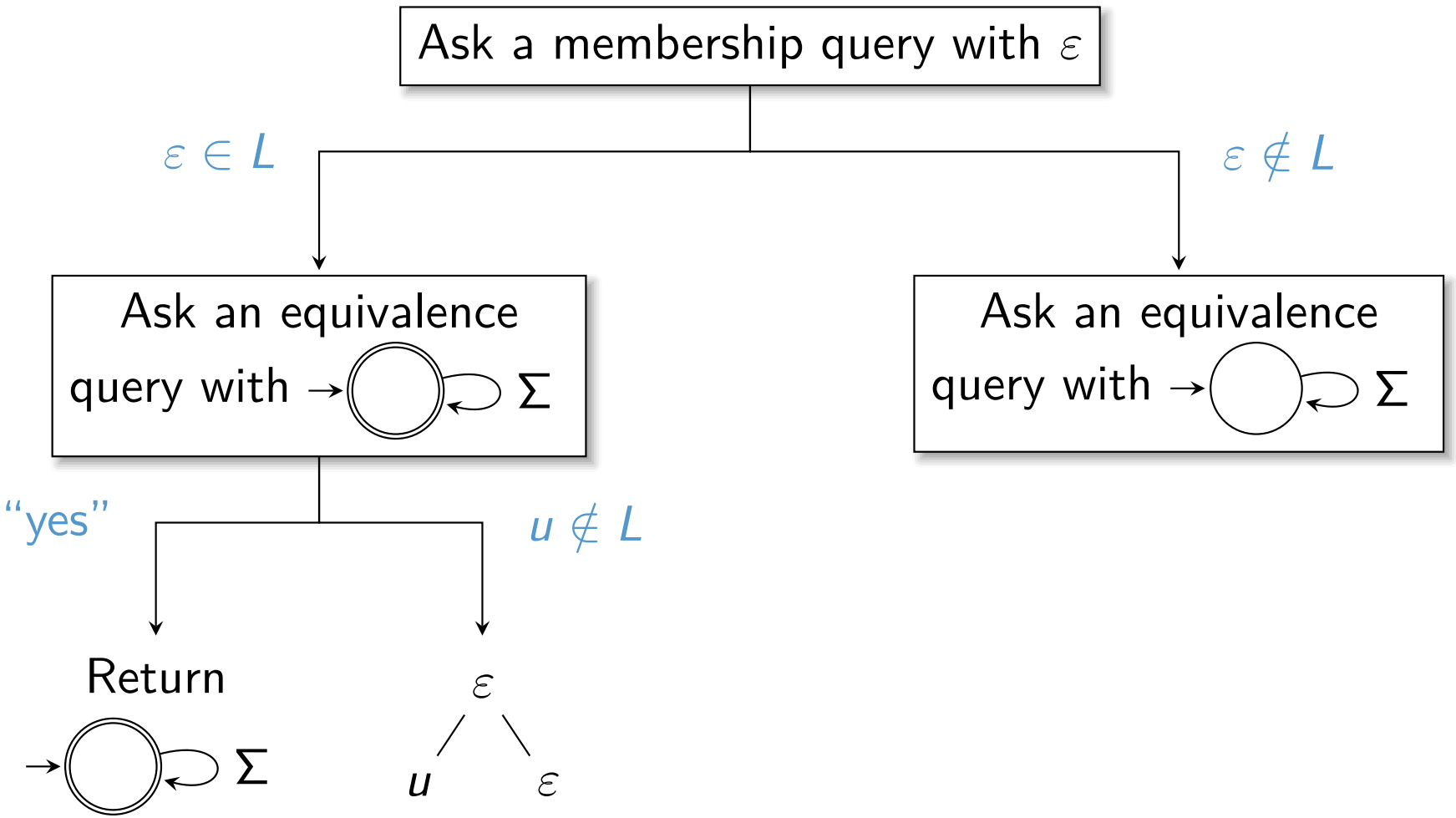
Initial Classification Tree



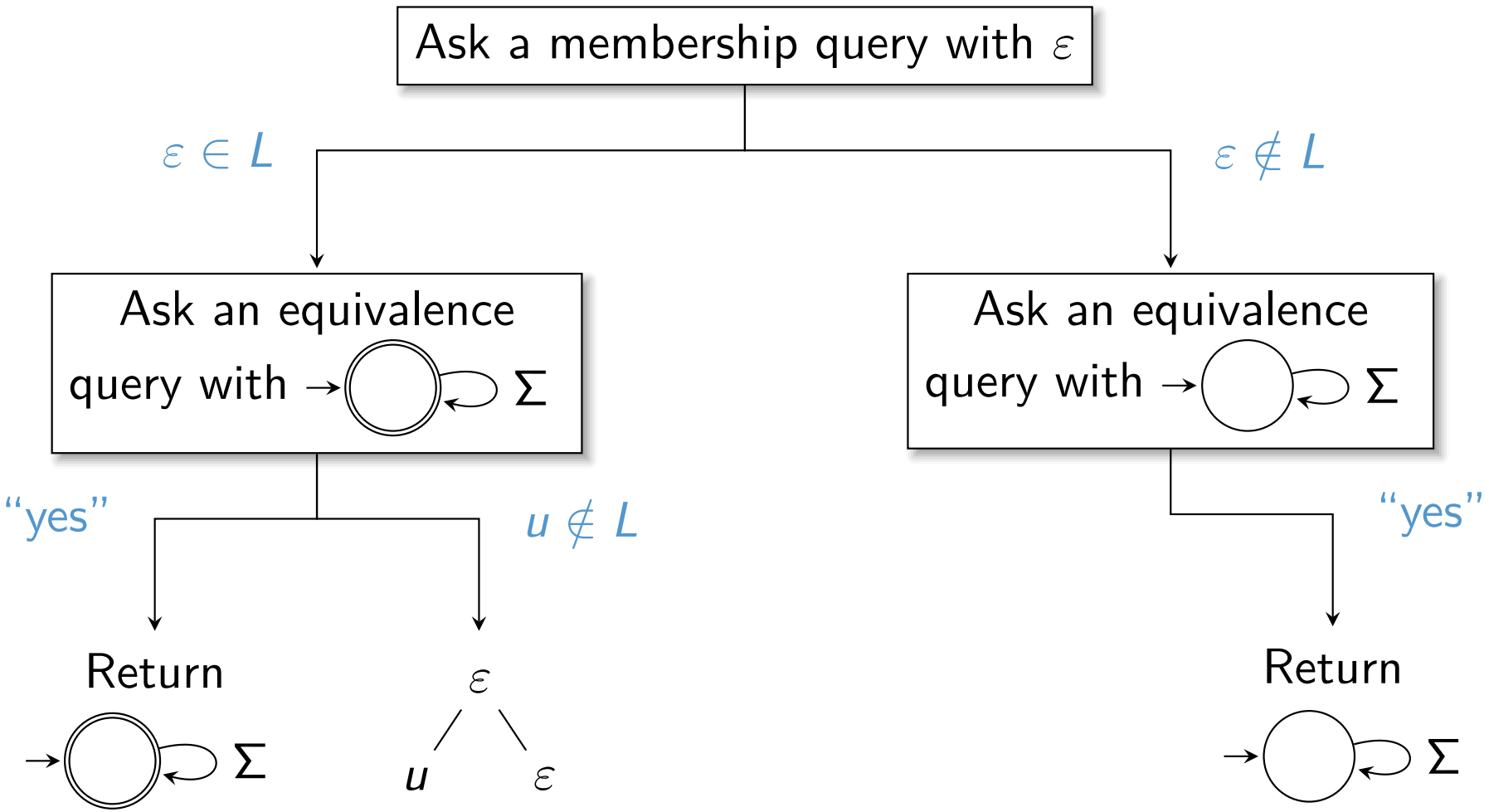
Initial Classification Tree



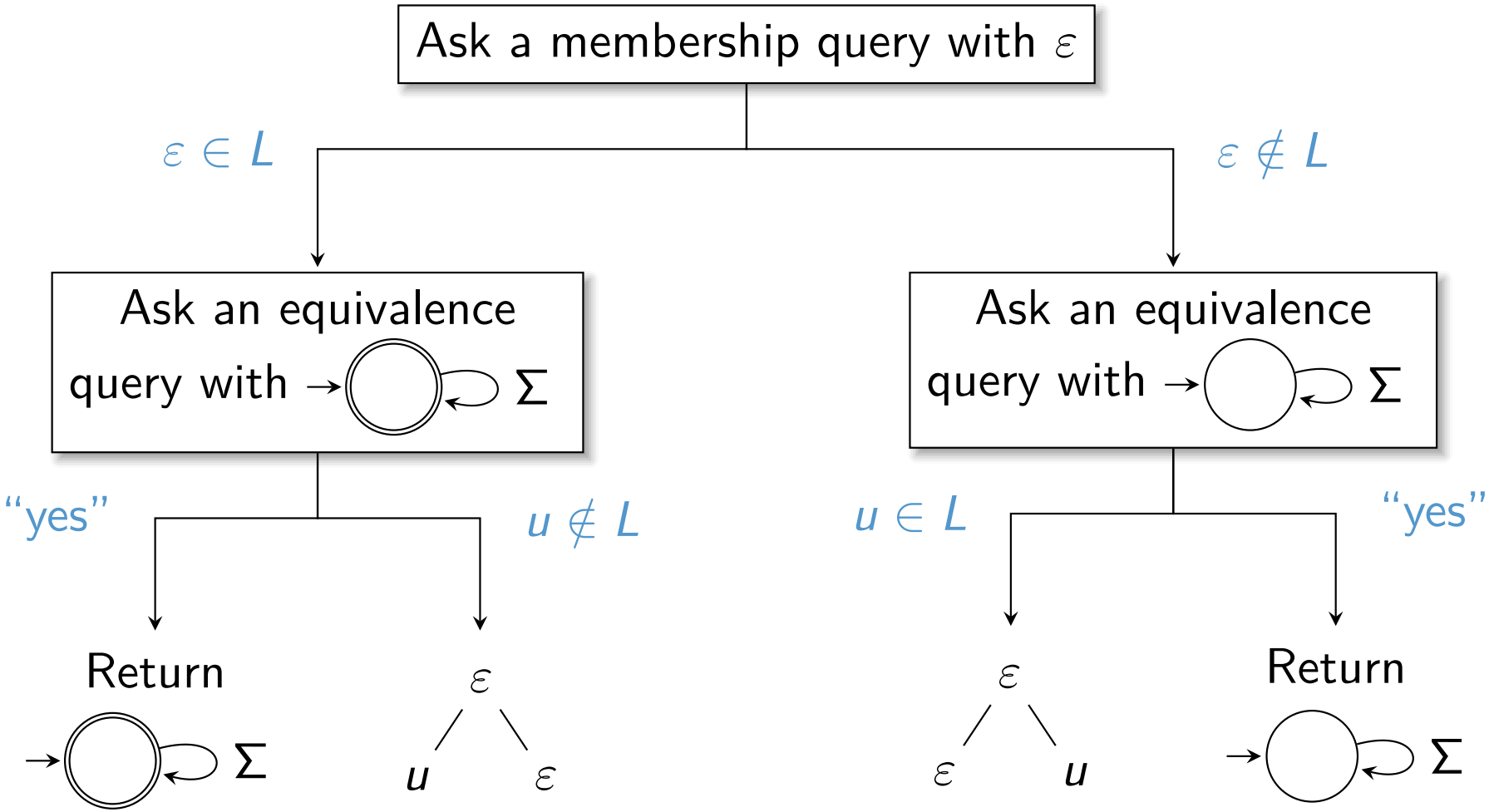
Initial Classification Tree



Initial Classification Tree



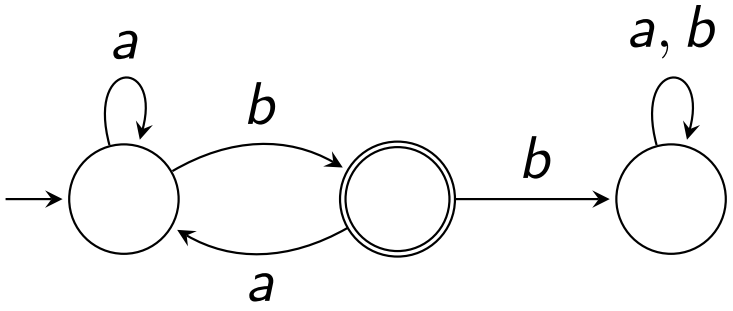
Initial Classification Tree



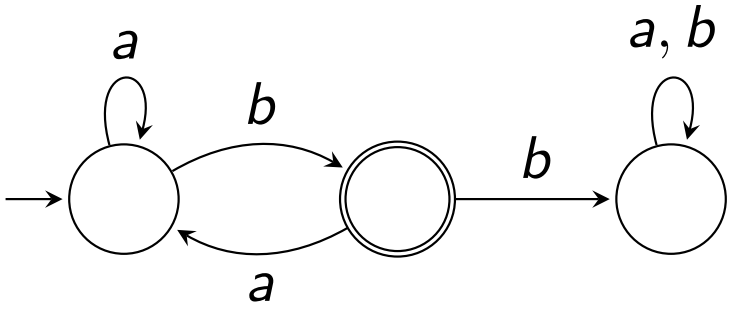
Algorithm 6: Kearns and Vazirani's Algorithm

```
Handle cases L = ∅ and L = Σ*
Initialize the classification tree t
repeat
    Construct A_t and ask an equivalence query
    if the teacher returns a counterexample u then
        Identify a breakpoint position i ∈ {1, ..., m}
        R ← R ∪ {u_{i-1} a_i}
        S ← S ∪ {a_{i+1} ... a_m}
        Update t by splitting the leaf node labeled with u_i
    end
until the teacher replies "yes" on an equivalence query
return A_t
```

Example

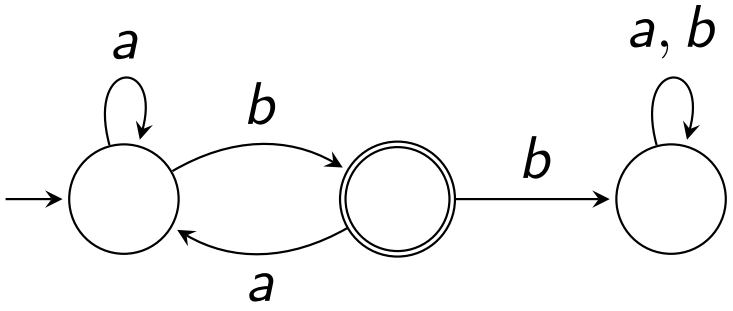


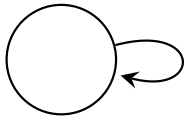
Example



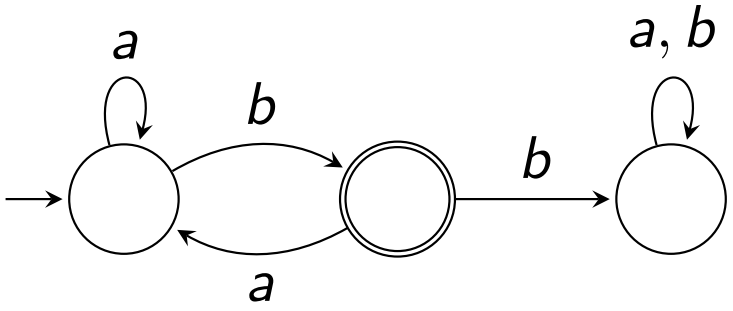
1. Ask a membership query with ε
   - Answer: $\varepsilon \notin L$

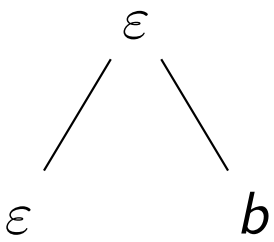
Example

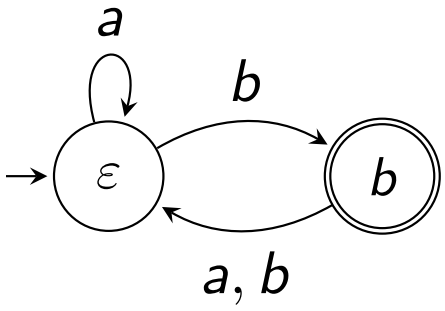


1. Ask a membership query with ε
   - Answer: $\varepsilon \notin L$
2. Ask an equivalence query with  a, b
   - Counterexample $b \in L$

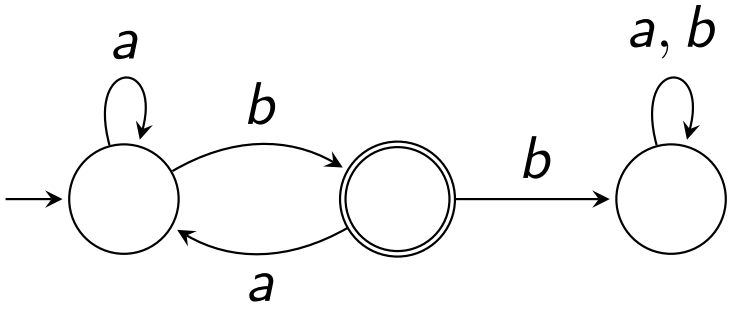
Example

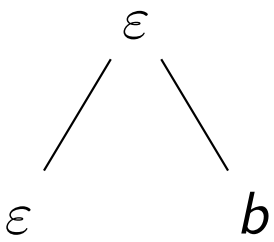


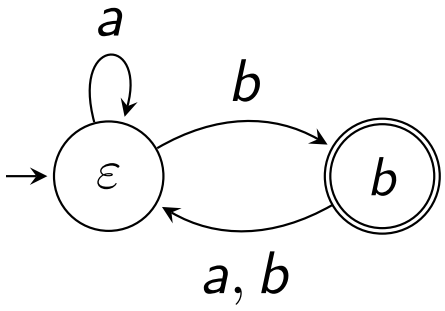




Example

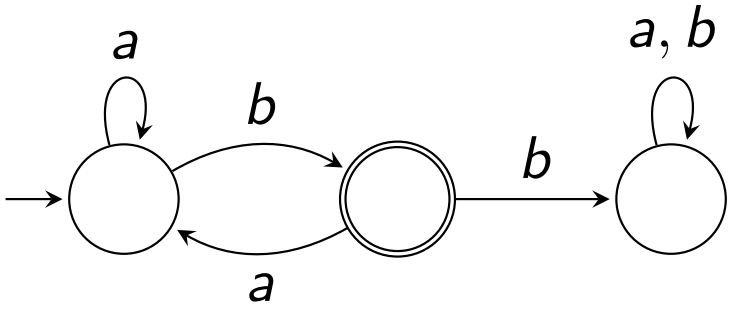




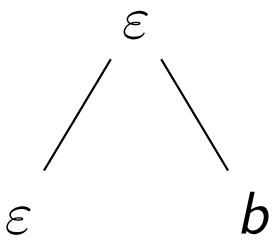


Counterexample *bbb*

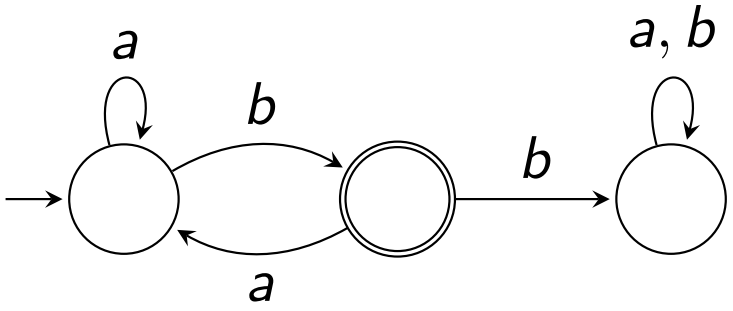
Example



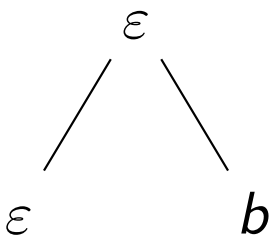
$$\mathcal{A}_t : \varepsilon \xrightarrow{b} b \xrightarrow{b} \varepsilon \xrightarrow{b} b$$



Example

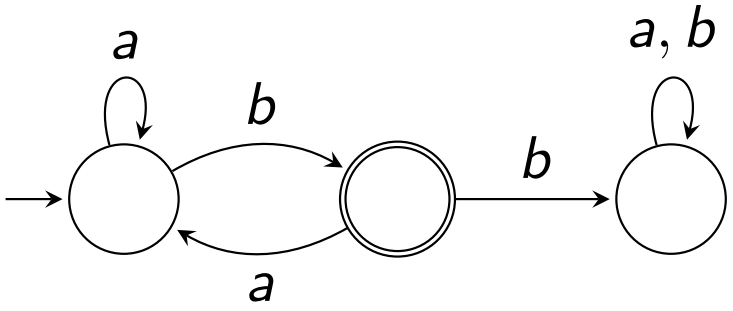


$$\mathcal{A}_t : \varepsilon \xrightarrow{b} b \xrightarrow{b} \varepsilon \xrightarrow{b} b$$

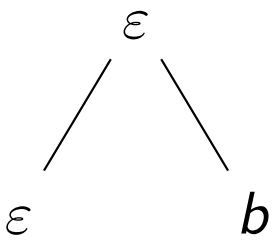


- $i = 1$:
  $u_0 bbb = bbb \notin L$ and $u_1 bb = bbb \notin L$

Example

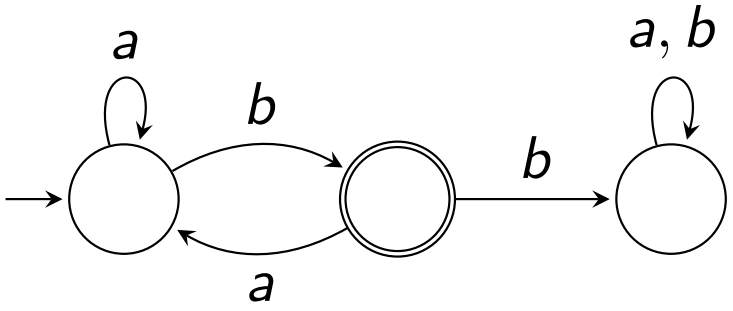


$$\mathcal{A}_t : \varepsilon \xrightarrow{b} b \xrightarrow{b} \varepsilon \xrightarrow{b} b$$

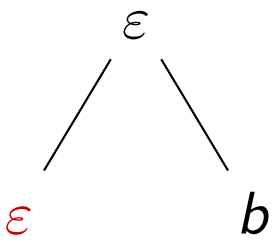


- $i = 1$:
  $u_0 bbb = bbb \notin L$ and $u_1 bb = bbb \notin L$
- $i = 2$:
  $u_1 bb = bbb \notin L$ and $u_2 b = b \in L$

Example



$$\mathcal{A}_t : \varepsilon \xrightarrow{b} b \xrightarrow{b} \varepsilon \xrightarrow{b} b$$

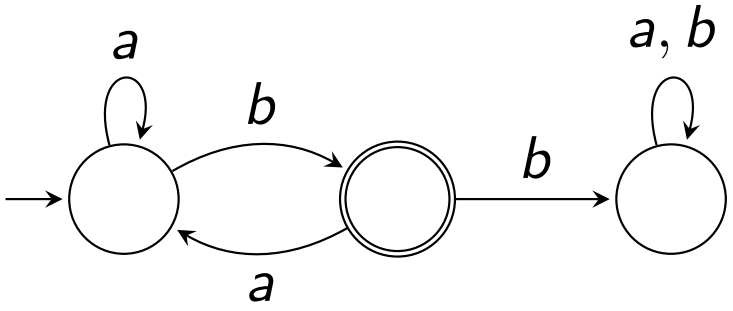


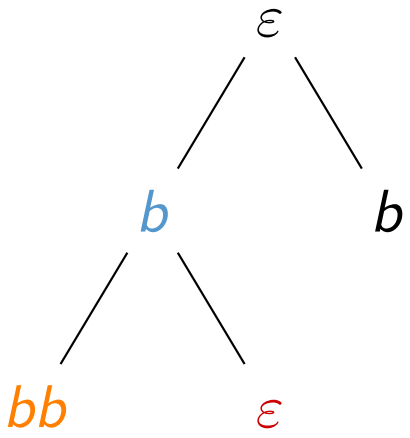
- $i = 1$:
  $u_0 bbb = bbb \notin L$ and $u_1 bb = bbb \notin L$
- $i = 2$:
  $u_1 bb = bbb \notin L$ and $u_2 b = b \in L$

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





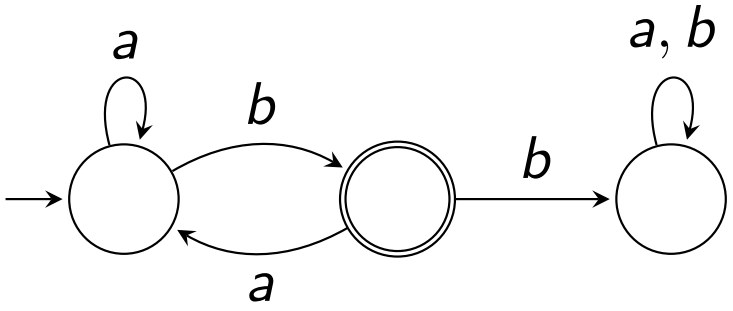
$$\mathcal{A}_t : \varepsilon \xrightarrow{b} b \xrightarrow{b} \varepsilon \xrightarrow{b} b$$

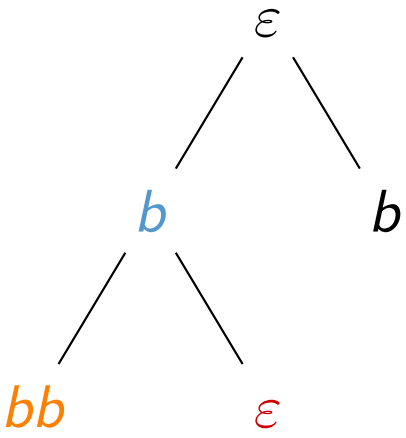
- $i = 1$:
  $u_0 bbb = bbb \notin L$ and $u_1 bb = bbb \notin L$
- $i = 2$:
  $u_1 bb = bbb \notin L$ and $u_2 b = b \in L$

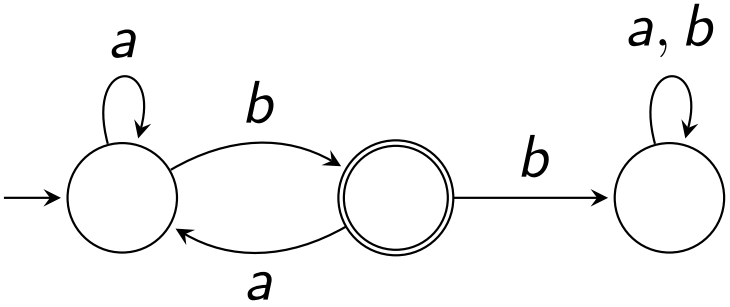
Thus, $u_1 b \not\sim_L u_2$, witnessed by b

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

Example







Kearns and Vazirani's Algorithm – Correctness

[10] M. J. Kearns and U. V. Vazirani: *Introduction to Computational Learning Theory*, 1994

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 $\mathcal{A}_L$ in time polynomial in the size n of $\mathcal{A}_L$ and the length m of the longest counterexample returned by the teacher. It asks exactly n equivalence queries and $\mathcal{O}(n^2 + n \log_2 m)$ membership queries.

Comparison of Active Learning Algorithms

Learning algorithm	*MQ*	*EQ*
Angluin's algorithm	$\mathcal{O}(n^2 m)$	$\leq n$
Kearns and Vazirani's algorithm	$\mathcal{O}(n^2 + n \log_2 m)$	n
Rivest and Schapire's algorithm	$\mathcal{O}(n^2 + n \log_2 m)$	$\leq n$

http://libalf.informatik.rwth-aachen.de/

Challenge 2

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

- $\mathcal{A}_{L_n}$ has at least n states and
- Angluin's algorithm learns L_n with a constant number of equivalence queries

3.

Applications

[11] M. Leucker: *Learning Meets Verification*, 2006

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

[11] M. Leucker: *Learning Meets Verification*, 2006

140 M. Leucker

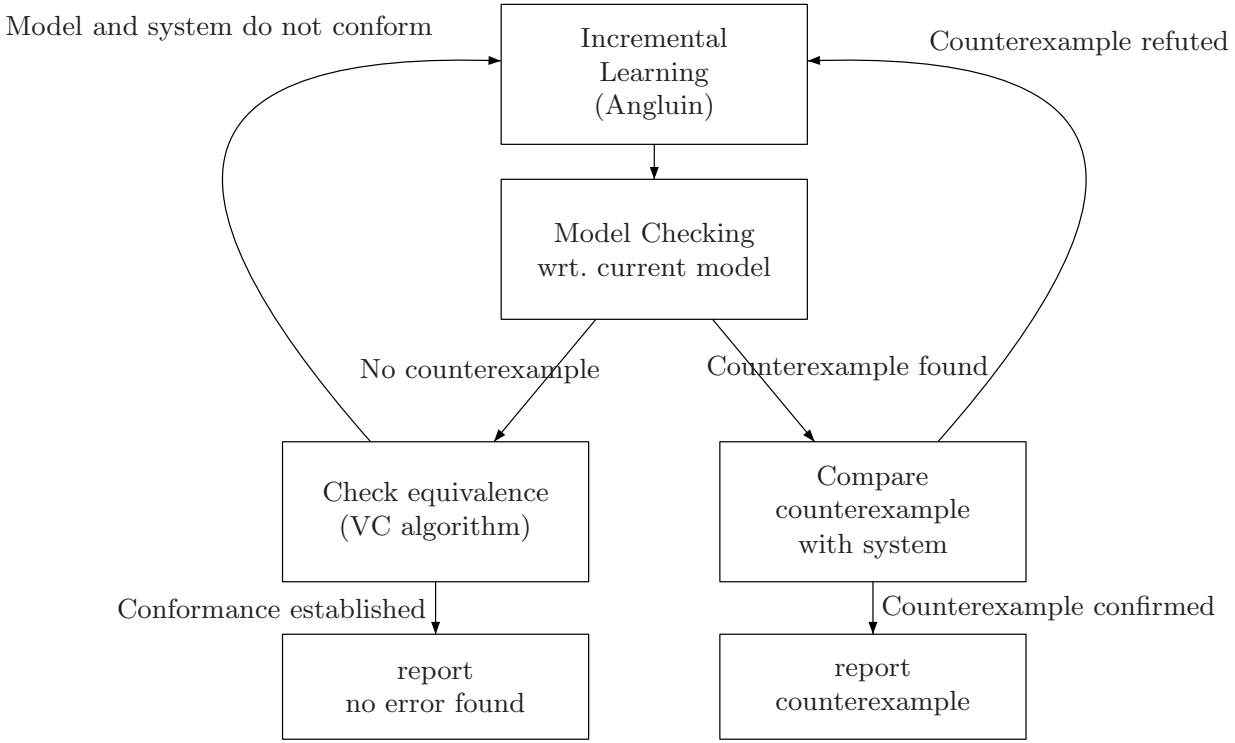


Fig. 3. Black Box Checking

[12] D. Neider and U. Topcu: *An Automaton Learning Approach to Solving Safety Games over Infinite Graphs*, 2016

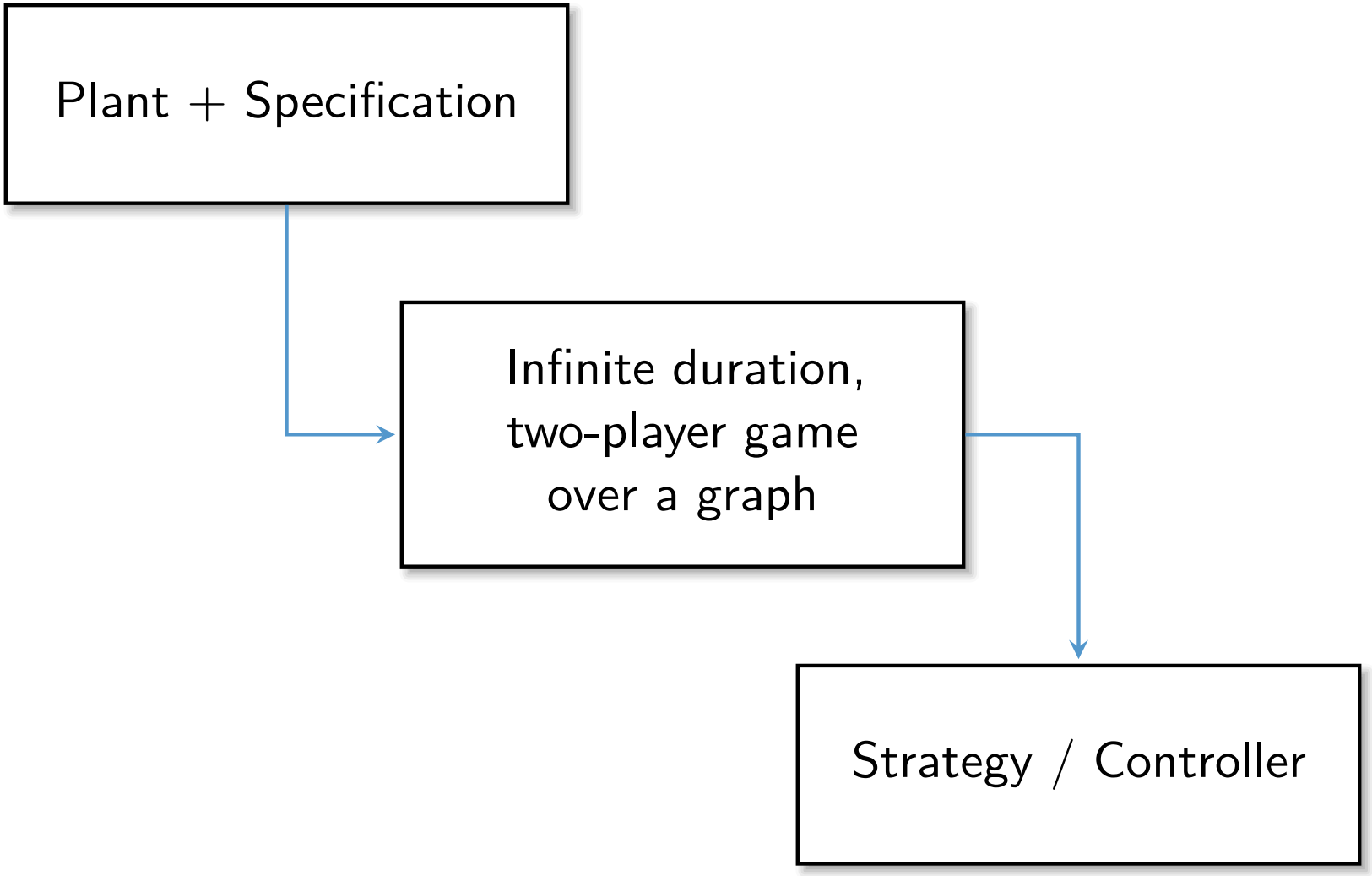
An Automaton Learning Approach to Solving Safety Games over Infinite Graphs

Daniel Neider[1] and Ufuk Topcu[2](✉)

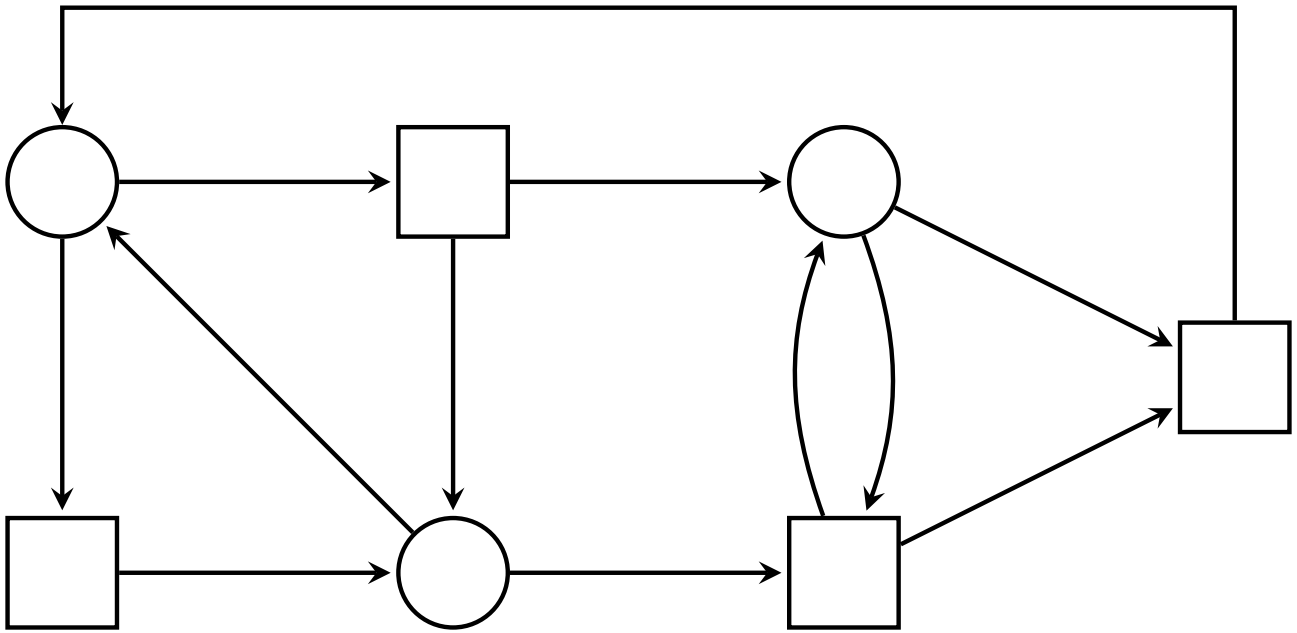
[1] University of California, Los Angeles, USA
neider@ucla.edu
[2] The University of Texas at Austin, Austin, USA
utopcu@utexas.edu

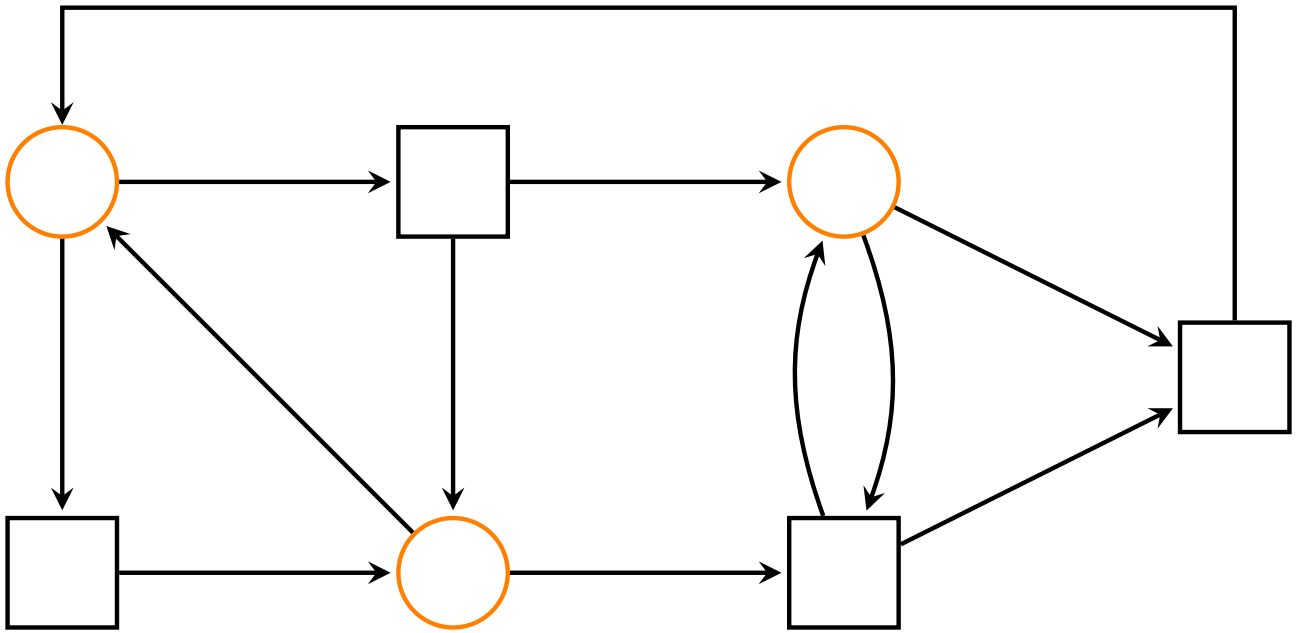
Abstract. We propose a method to construct finite-state reactive controllers for systems whose interactions with their adversarial environment are modeled by infinite-duration two-player games over (possibly) infi-

Synthesis of Reactive Controllers



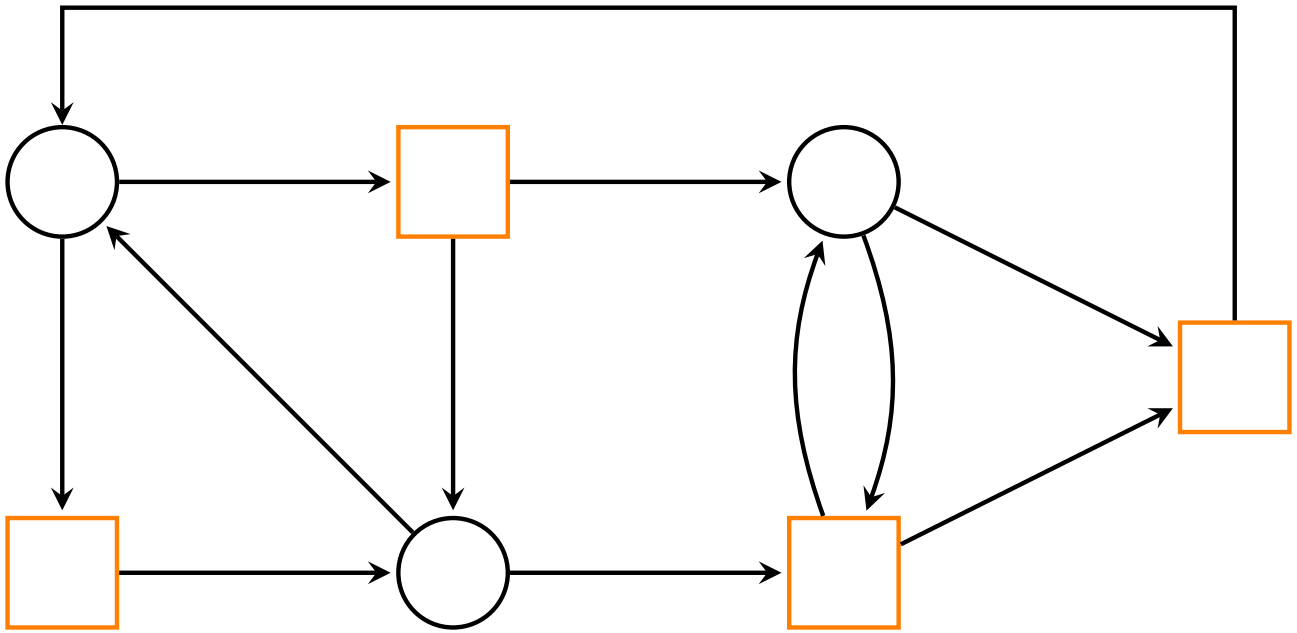
Safety Games





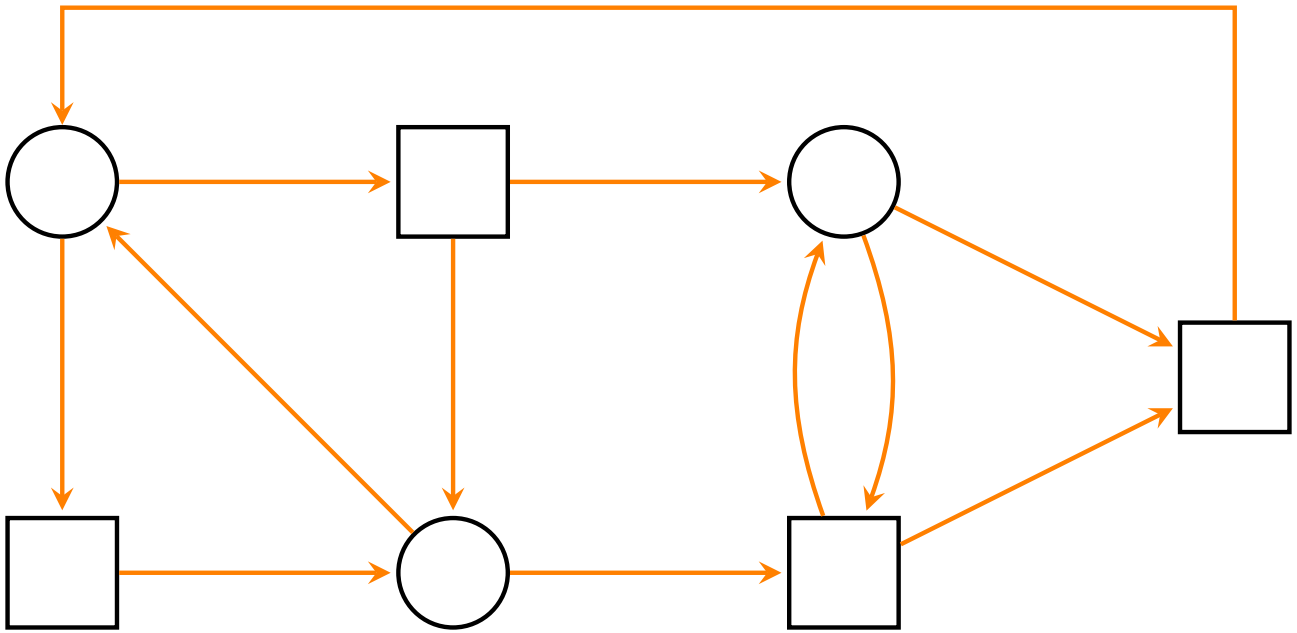
- Vertices V_0 of Player 0

Safety Games



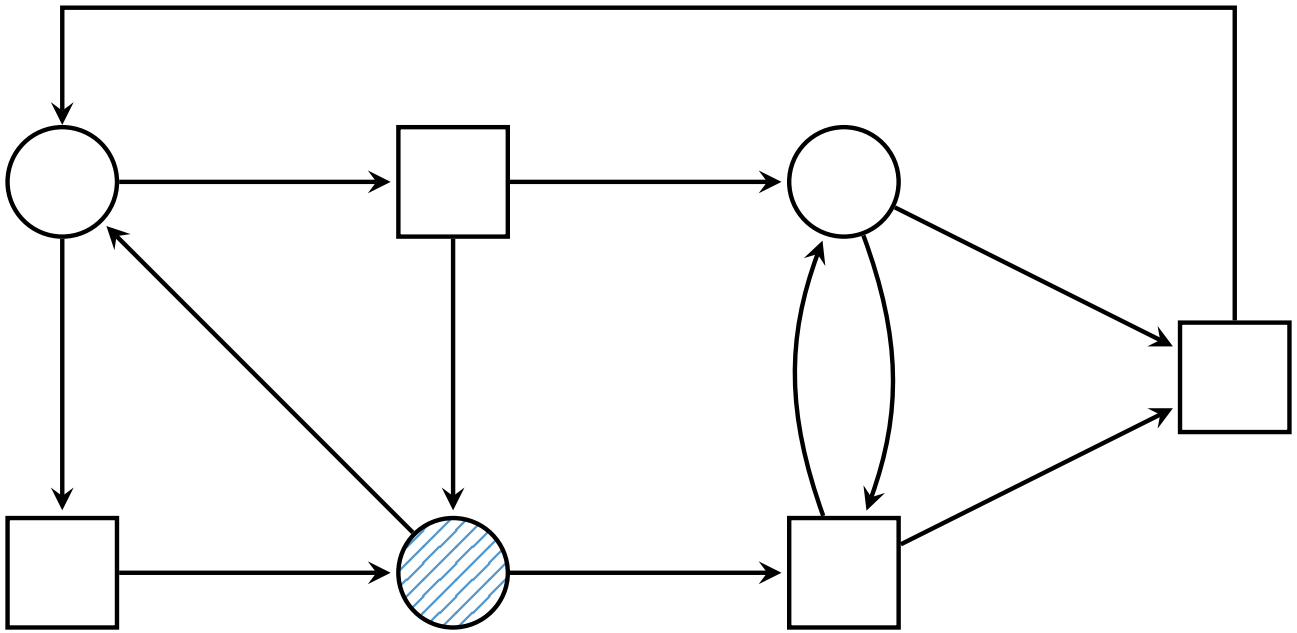
- Vertices V_0 of Player 0, vertices V_1 of Player 1

Safety Games



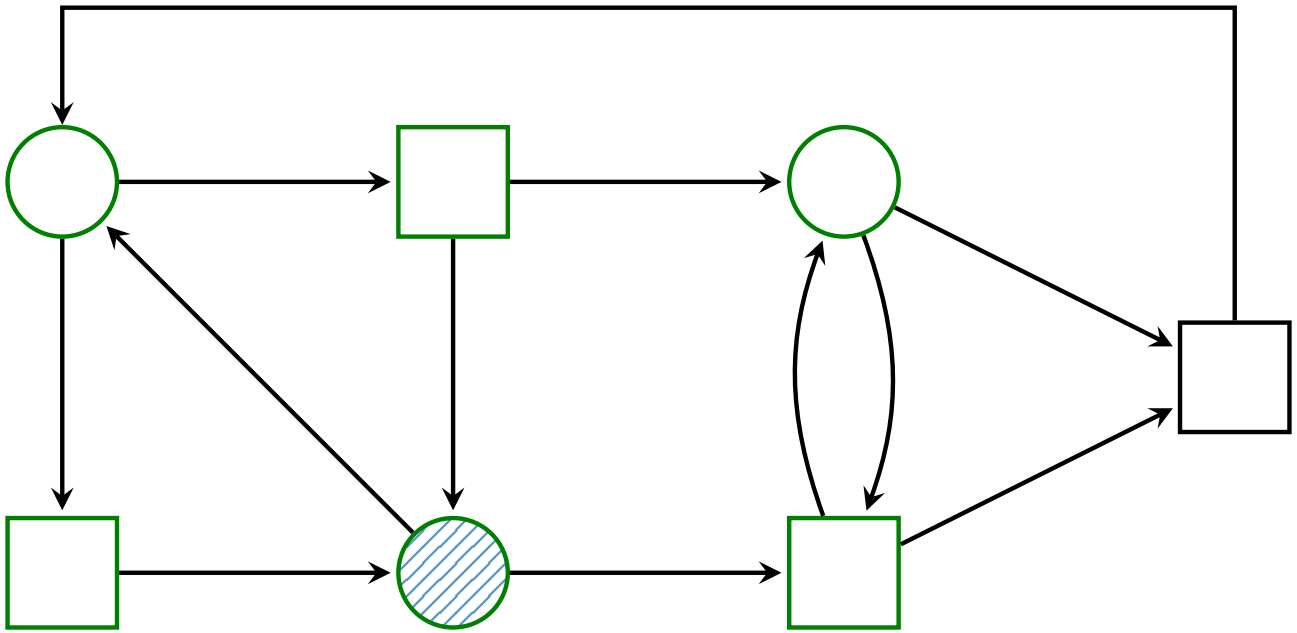
- Vertices V_0 of Player 0, vertices V_1 of Player 1
- Edges E

Safety Games



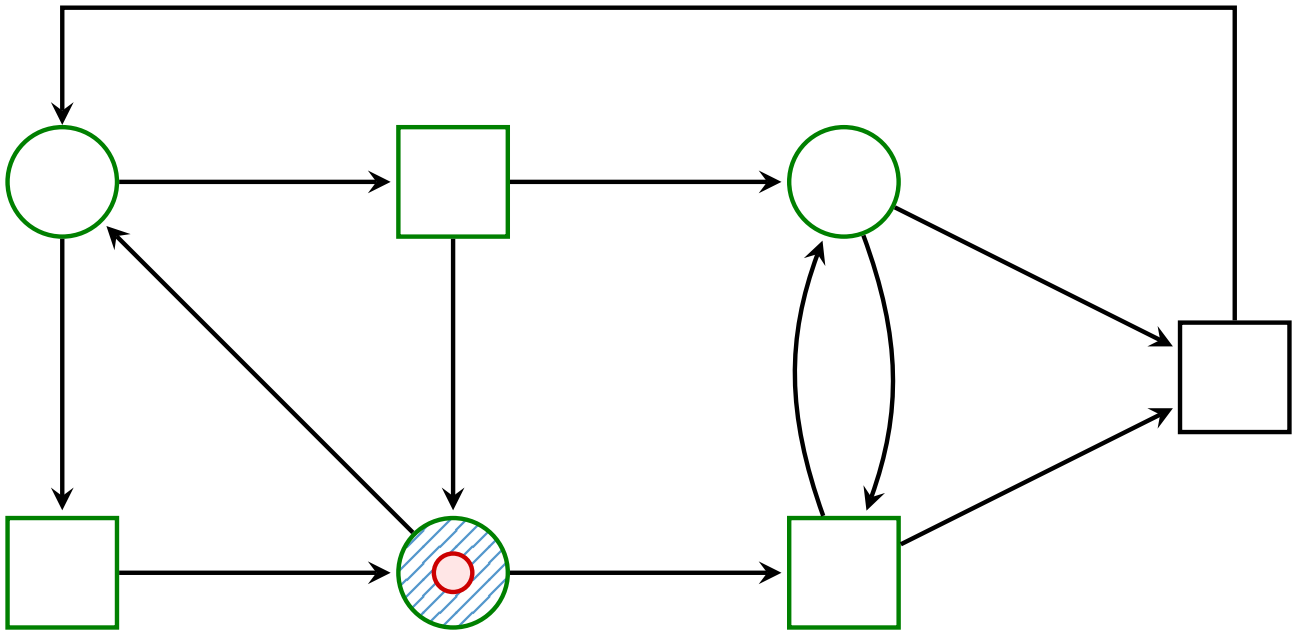
- Vertices V_0 of Player 0, vertices V_1 of Player 1
- Edges E
- Initial vertices I

Safety Games



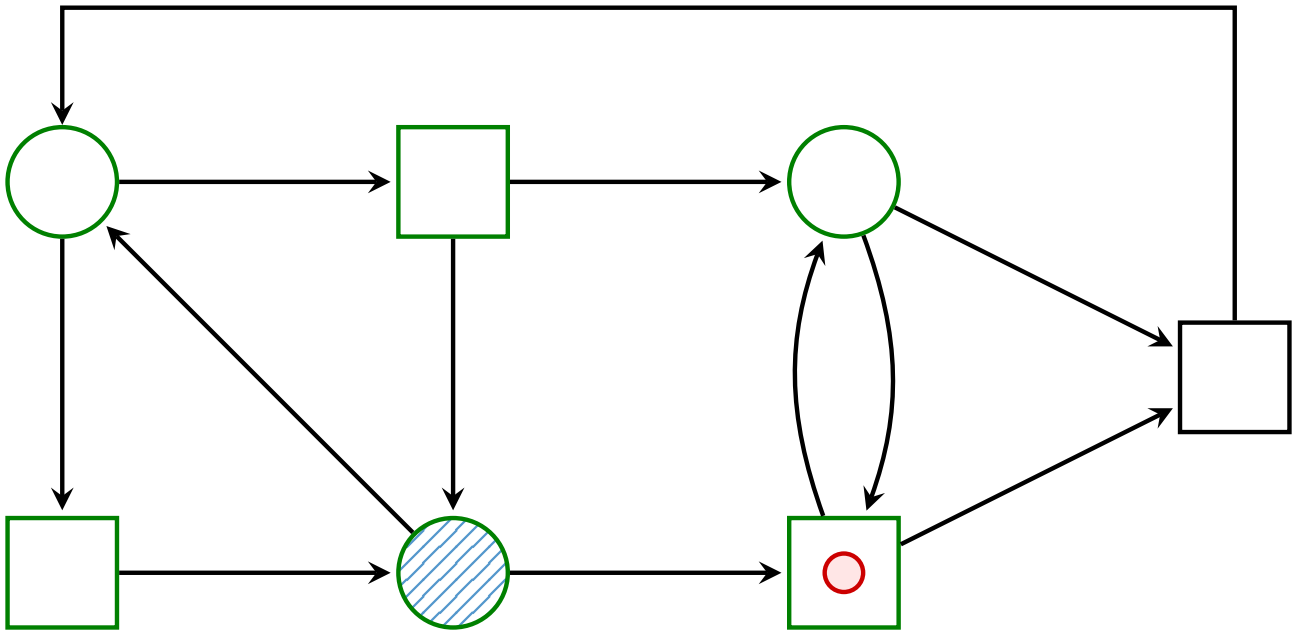
- Vertices V_0 of Player 0, vertices V_1 of Player 1
- Edges E
- Initial vertices I
- Safe vertices F

Safety Games



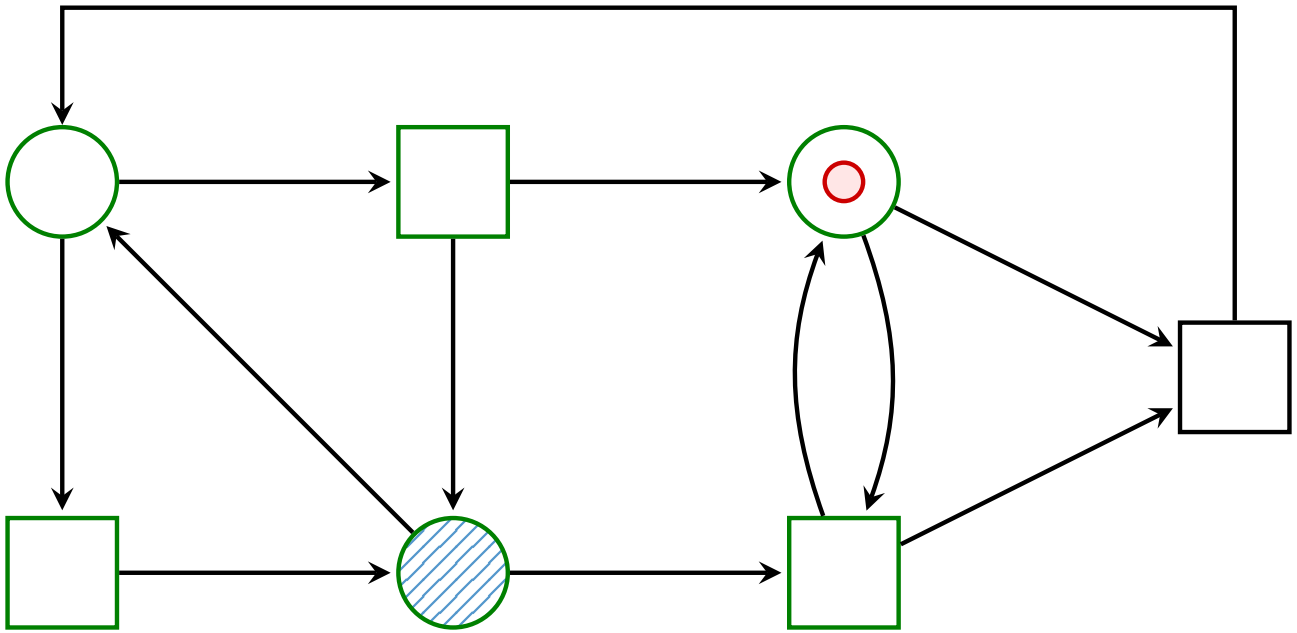
- Vertices V_0 of Player 0, vertices V_1 of Player 1
- Edges E
- Initial vertices I
- Safe vertices F

Safety Games



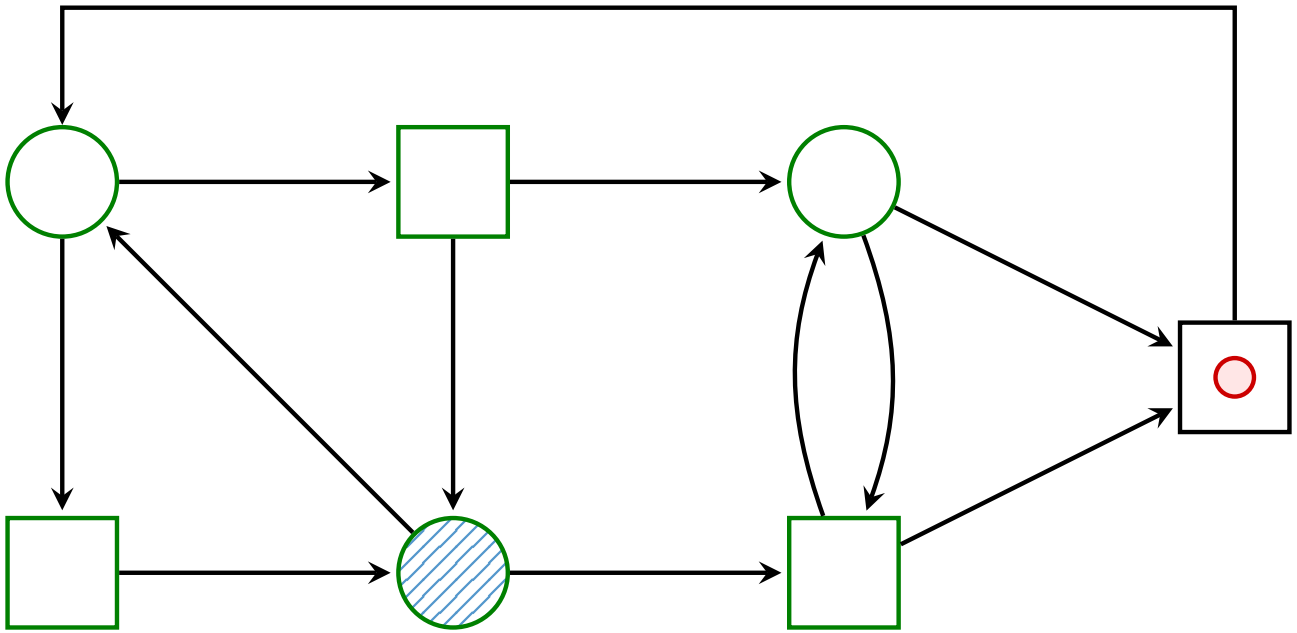
- Vertices V_0 of Player 0, vertices V_1 of Player 1
- Edges E
- Initial vertices I
- Safe vertices F

Safety Games

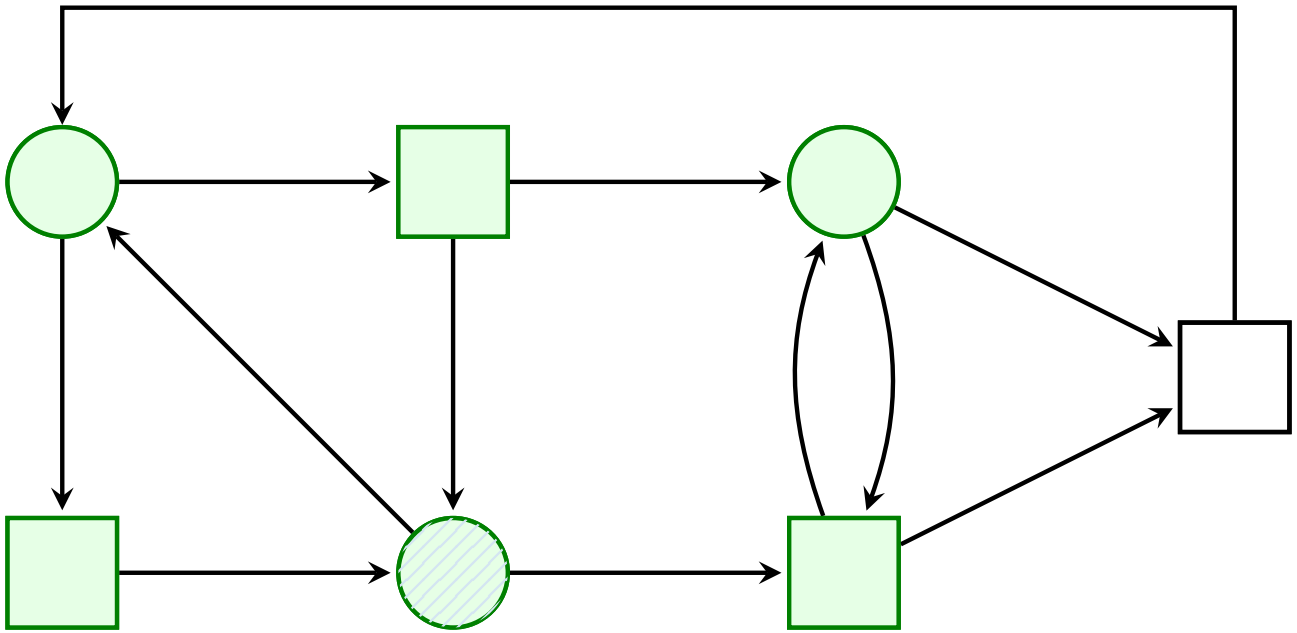


- Vertices V_0 of Player 0, vertices V_1 of Player 1
- Edges E
- Initial vertices I
- Safe vertices F

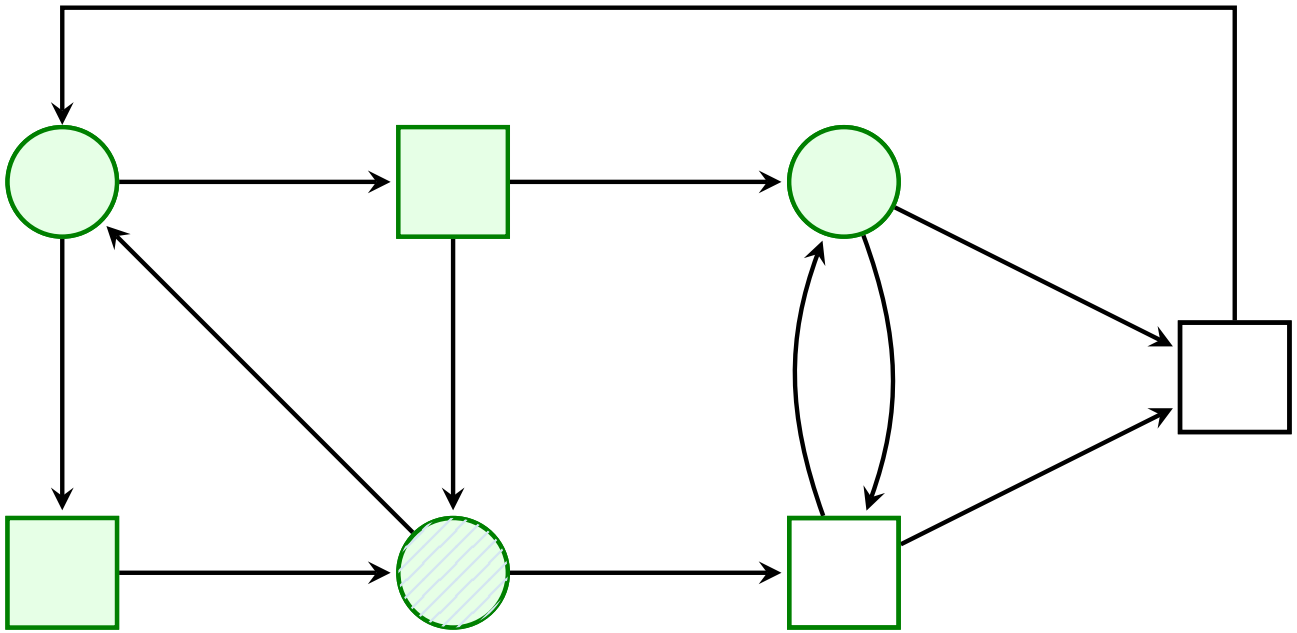
Safety Games



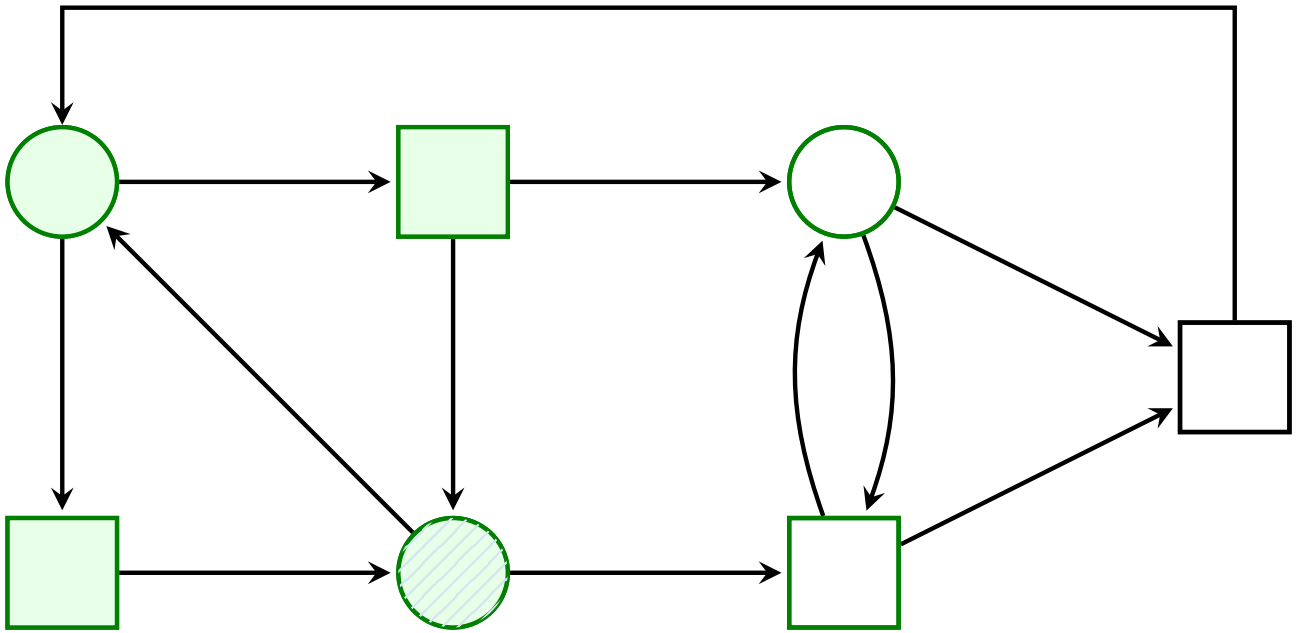
- Vertices V_0 of Player 0, vertices V_1 of Player 1
- Edges E
- Initial vertices I
- Safe vertices F



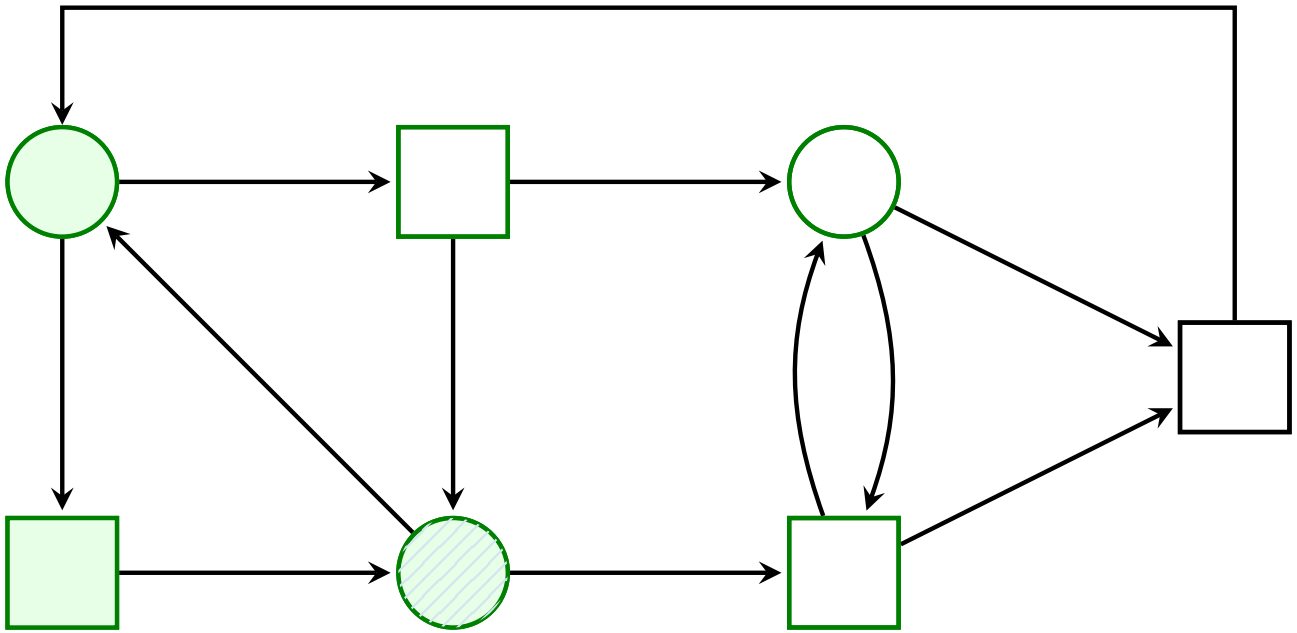
- Successively remove vertices from which a stay inside the safe vertices cannot be enforced



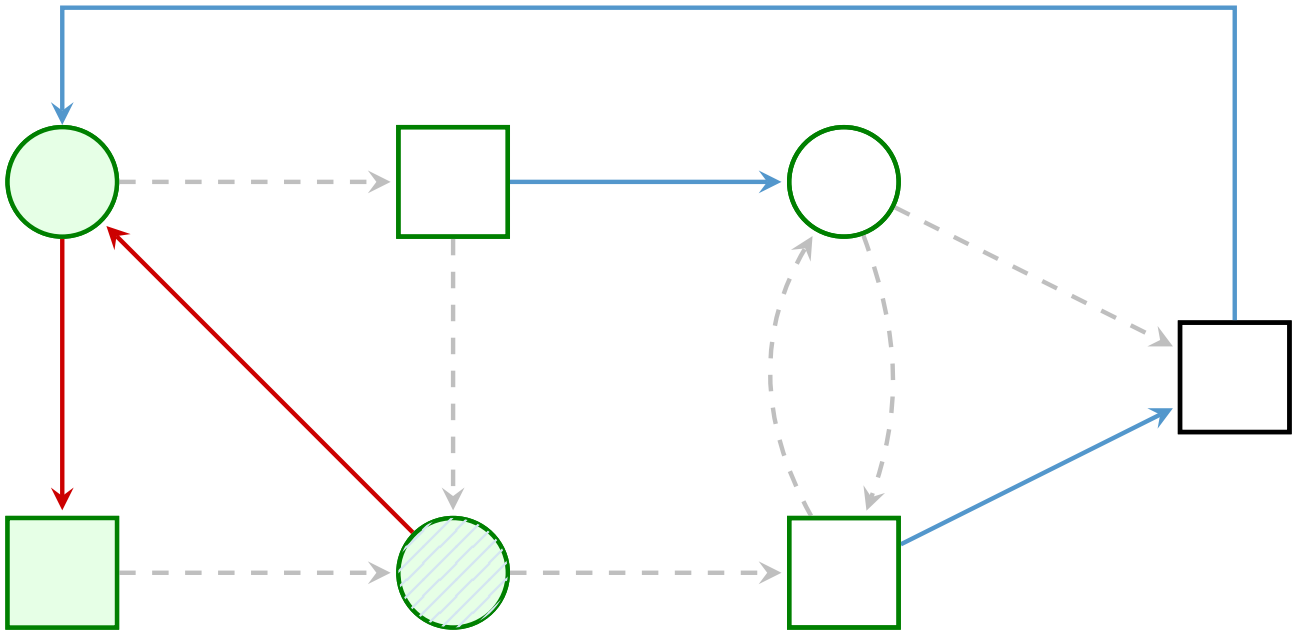
- Successively remove vertices from which a stay inside the safe vertices cannot be enforced



- Successively remove vertices from which a stay inside the safe vertices cannot be enforced

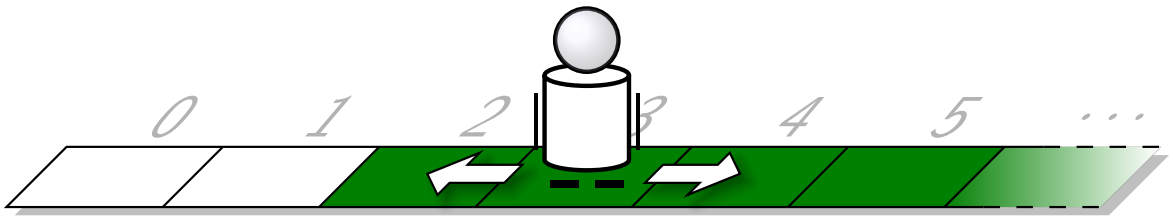


- Successively remove vertices from which a stay inside the safe vertices cannot be enforced

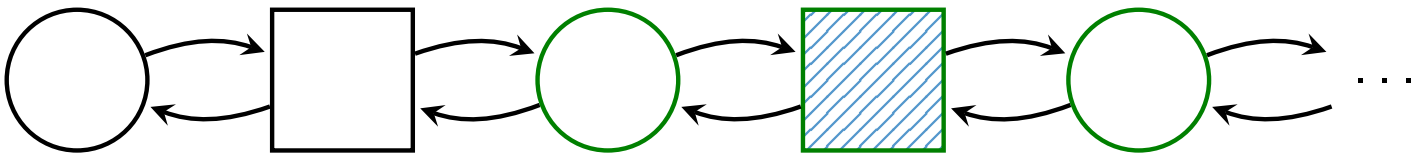


- Successively remove vertices from which a stay inside the safe vertices cannot be enforced
- Winning strategy for Player 0, winning strategy for Player 1

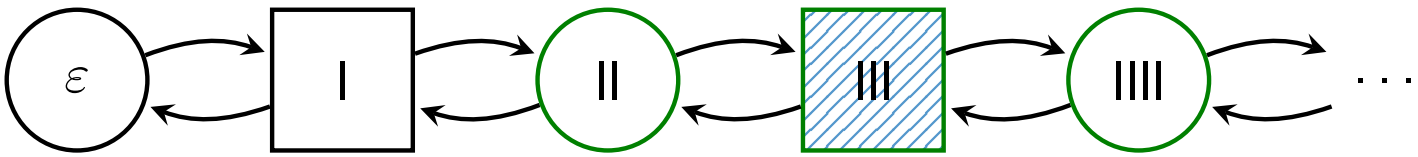
Safety Games Over Infinite Game Graphs



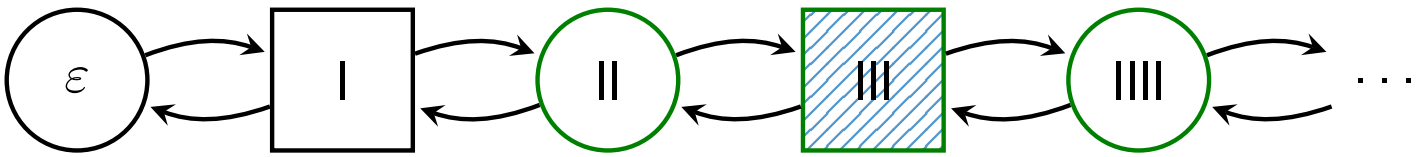
Safety Games Over Infinite Game Graphs



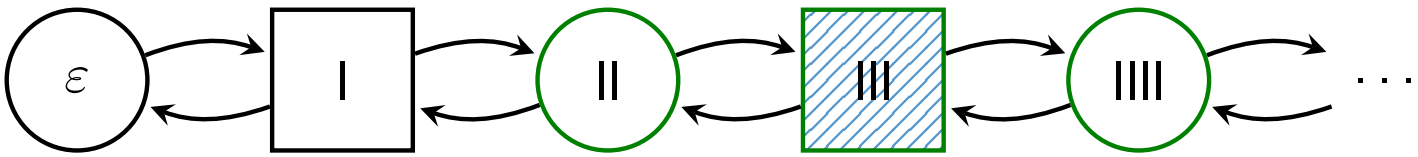
Safety Games Over Infinite Game Graphs

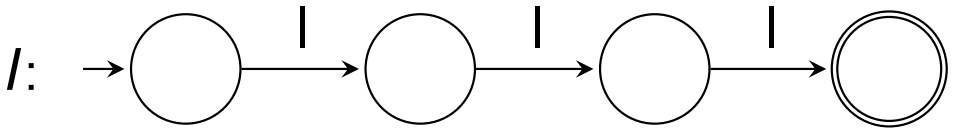


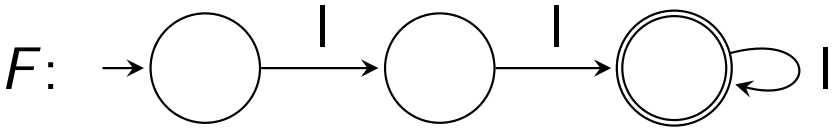
Safety Games Over Infinite Game Graphs

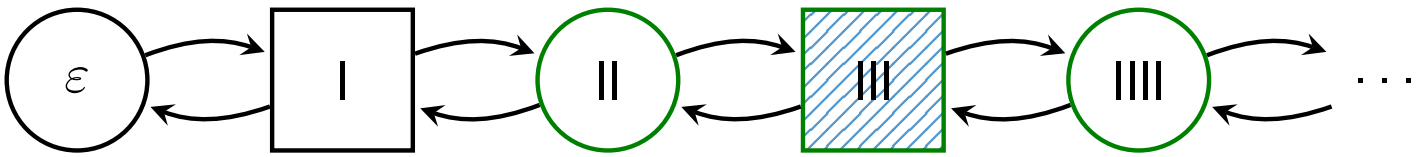


Safety Games Over Infinite Game Graphs



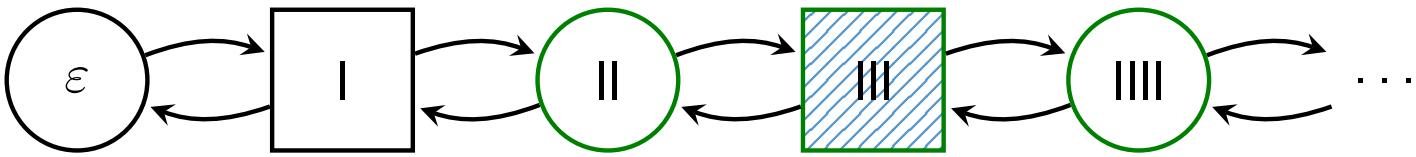


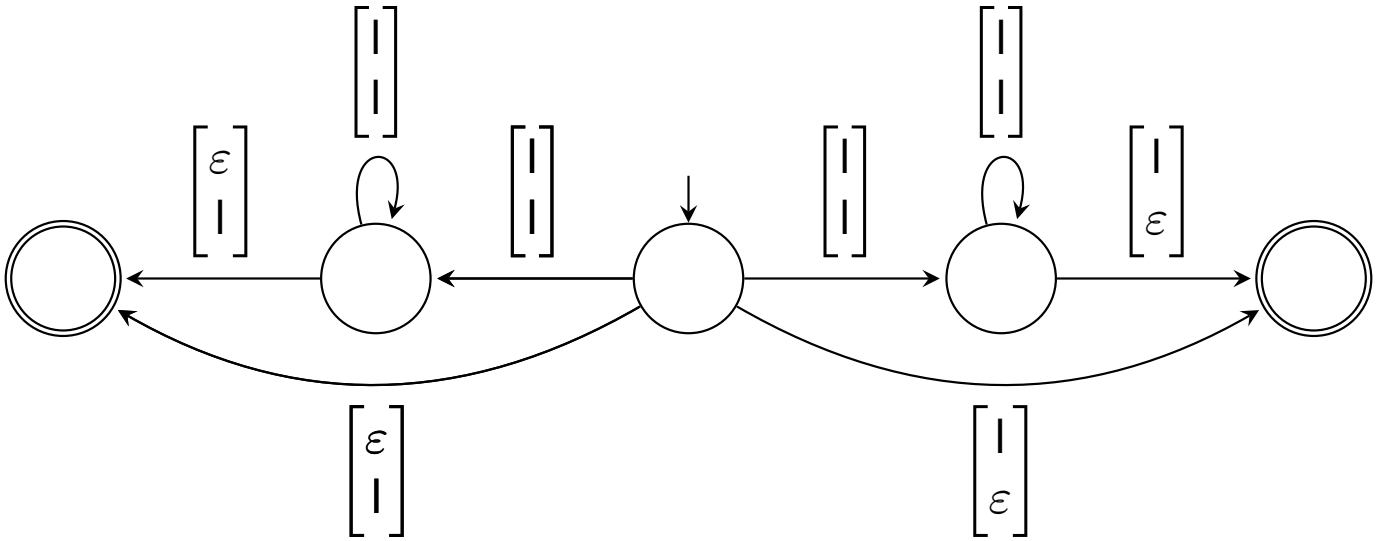




Model the edge II → III as word

Safety Games Over Infinite Game Graphs





Definition

A *rational safety games* is a five-tuple

$$\mathcal{G} = (\mathcal{A}_{V_0}, \mathcal{A}_{V_1}, \mathcal{A}_I, \mathcal{A}_F, \mathcal{T}_E)$$

consisting of

- an NFA $\mathcal{A}_{V_0}$ encoding the vertices of Player 0
- an NFA $\mathcal{A}_{V_1}$ encoding the vertices of Player 1
- an NFA $\mathcal{A}_I$ encoding the initial vertices
- an NFA $\mathcal{A}_F$ encoding the safe vertices
- a rational transducer $\mathcal{T}_E$ encoding the edges

Definition

A *rational safety games* is a five-tuple

$$\mathcal{G} = (\mathcal{A}_{V_0}, \mathcal{A}_{V_1}, \mathcal{A}_I, \mathcal{A}_F, \mathcal{T}_E)$$

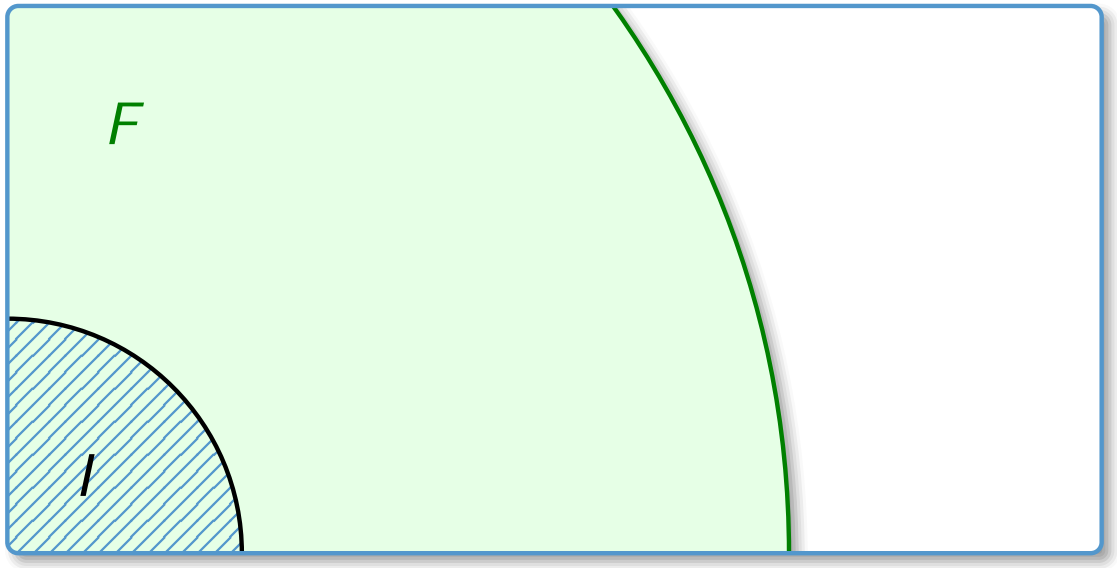
consisting of

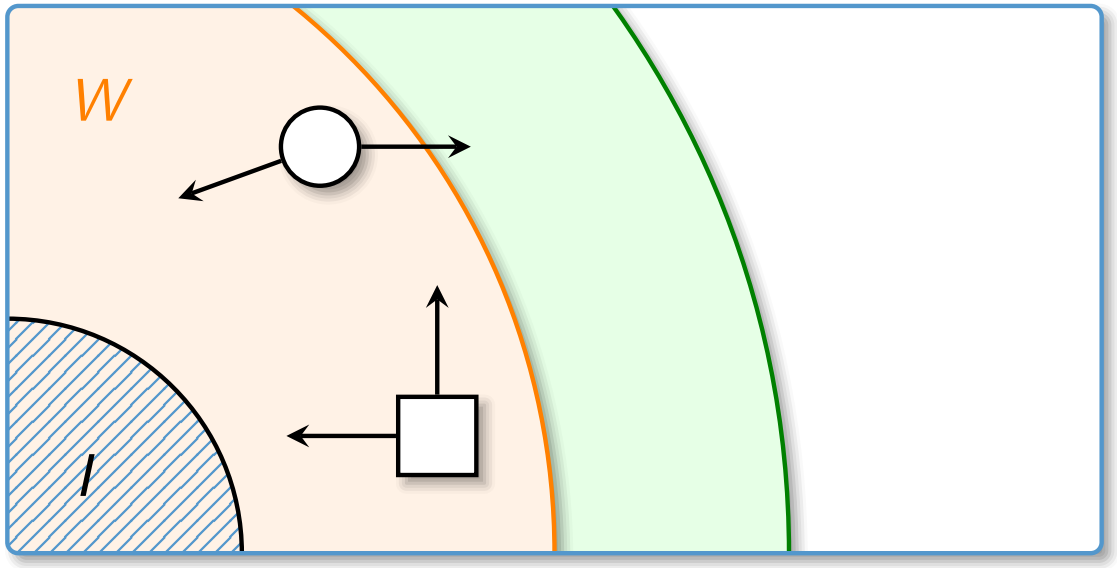
- an NFA $\mathcal{A}_{V_0}$ encoding the vertices of Player 0
- an NFA $\mathcal{A}_{V_1}$ encoding the vertices of Player 1
- an NFA $\mathcal{A}_I$ encoding the initial vertices
- an NFA $\mathcal{A}_F$ encoding the safe vertices
- a rational transducer $\mathcal{T}_E$ encoding the edges

Assumption

Each vertex has only a finite number of successors

Winning Sets



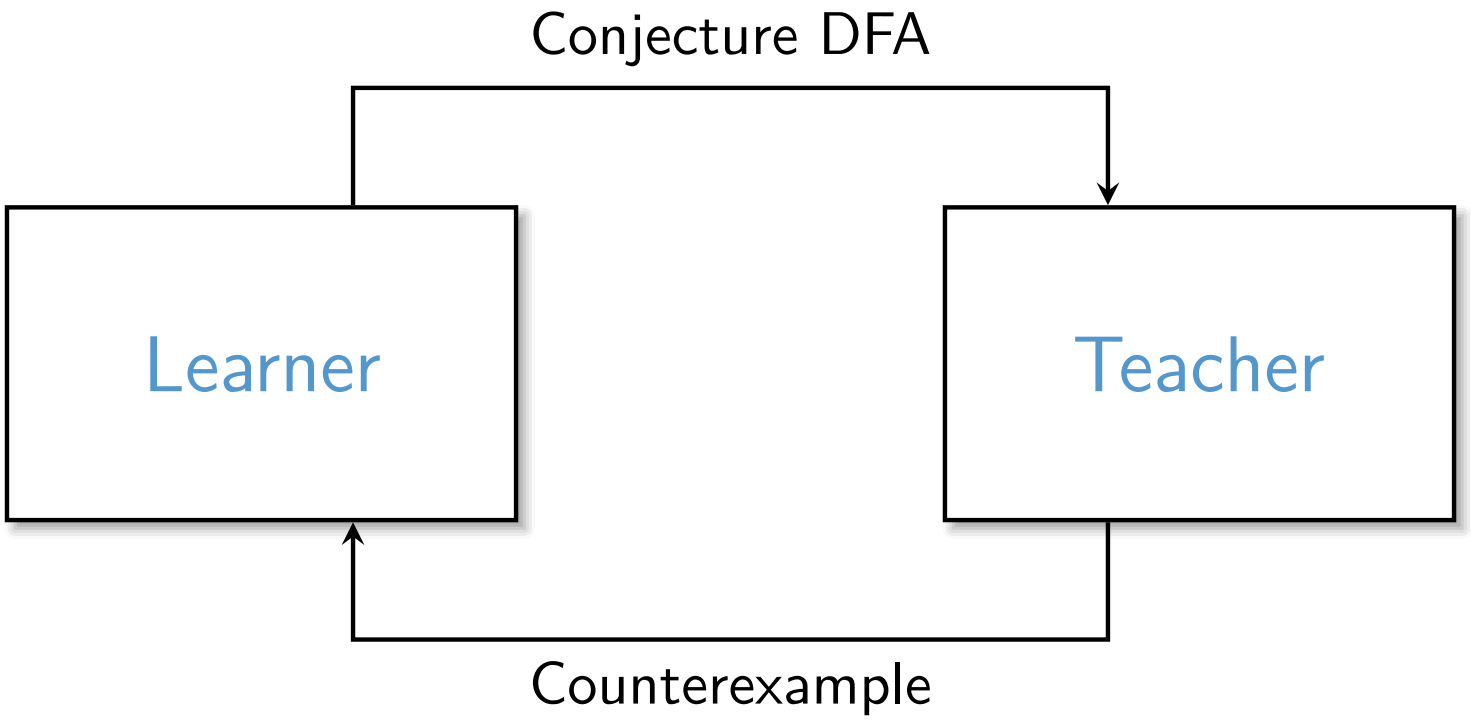


Winning Set

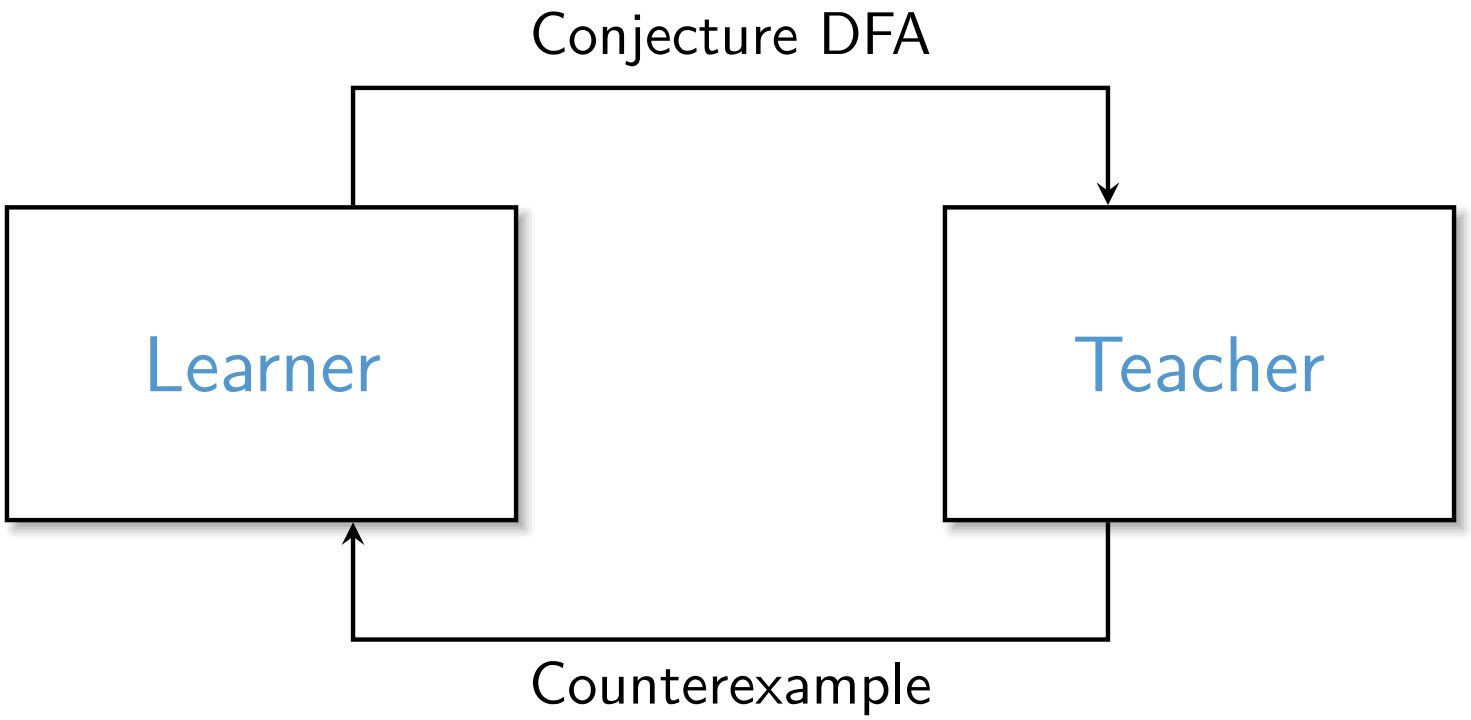
A (regular) W of vertices is a winning set if is satisfies

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

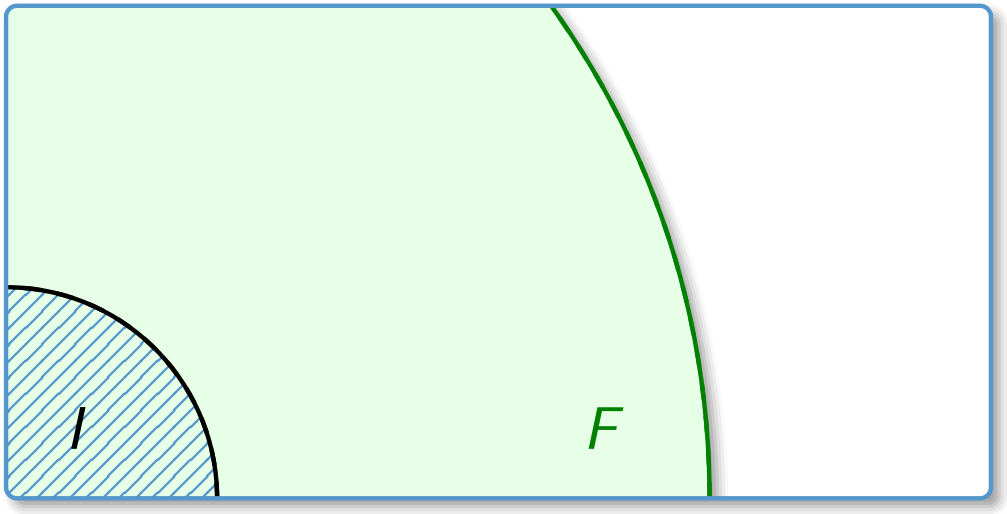


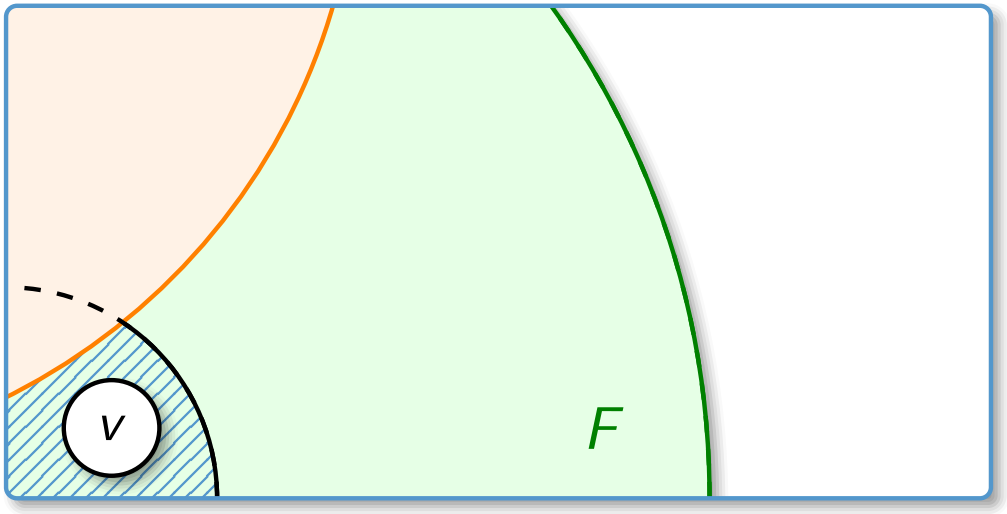
Iterative Passive Learning



Today: counterexample-guided inductive synthesis

Teacher

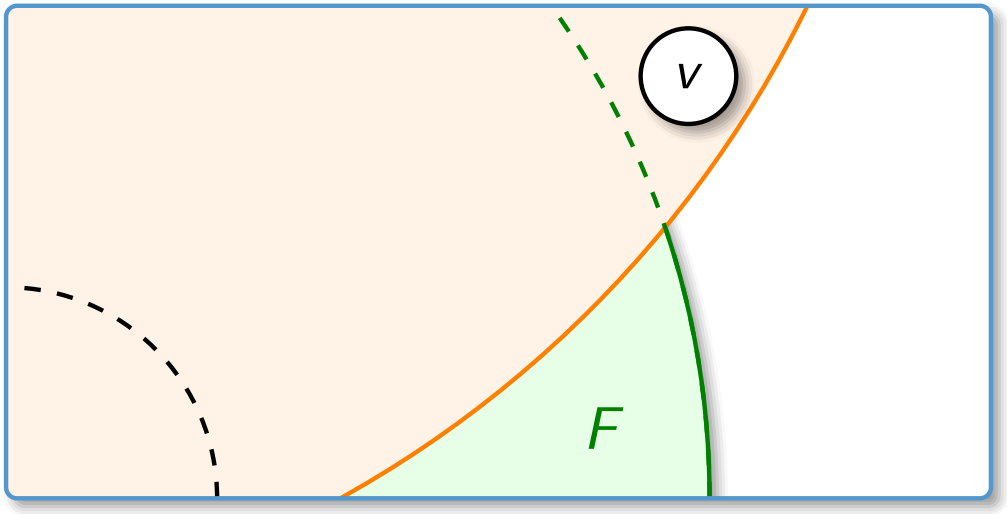




Counterexample

Let $\mathcal{C}$ be the conjectured DFA

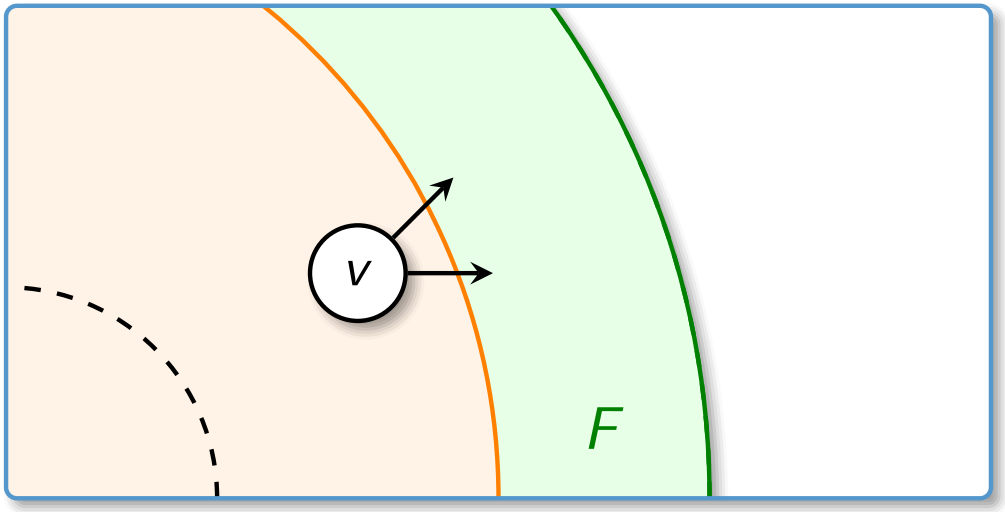
- Positive counterexample: $v \in I \setminus L(\mathcal{C})$



Counterexample

Let $\mathcal{C}$ be the conjectured DFA

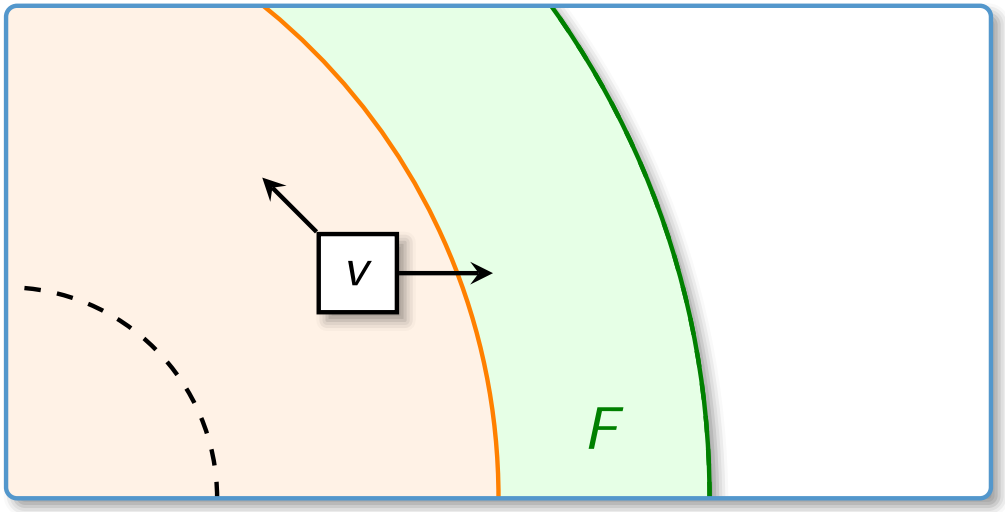
- Positive counterexample: $v \in I \setminus L(\mathcal{C})$
- Negative counterexample: $v \in L(\mathcal{C}) \setminus F$



Counterexample

Let $\mathcal{C}$ be the conjectured DFA

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



Counterexample

Let $\mathcal{C}$ be the conjectured DFA

- Positive counterexample: $v \in I \setminus L(\mathcal{C})$
- Negative counterexample: $v \in L(\mathcal{C}) \setminus F$
- Existential implication: (v, U) with $v \in V_0 \cap L(\mathcal{C})$, $U = E(v)$, and $U \cap L(\mathcal{C}) = \emptyset$
- Universal implication: (v, U) with $v \in V_1 \cap L(\mathcal{C})$, $U = E(v)$, and $U \not\subseteq L(\mathcal{C})$

The learners maintain a sample $\mathcal{S} = (S_+, S_-, I_\exists, I_\forall)$ to store counterexamples

Learning Task

Given a sample $\mathcal{S}$, construct a DFA $\mathcal{C}$ that is consistent with $\mathcal{S}$:

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

The learners maintain a sample $\mathcal{S} = (S_+, S_-, I_\exists, I_\forall)$ to store counterexamples

Learning Task

Given a sample $\mathcal{S}$, construct a DFA $\mathcal{C}$ that is consistent with $\mathcal{S}$:

1. $S_+ \subseteq L(\mathcal{C})$
2. $S_- \cap L(\mathcal{C}) = \emptyset$
3. $v \in L(C)$ implies $U \cap L(\mathcal{C}) \neq \emptyset$ for each $(v, U) \in I_\exists$
4. $v \in L(C)$ implies $U \subseteq L(\mathcal{C})$ 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 $\psi_n^{\mathcal{S}}$ that is satisfiable if and only if a consistent DFA with n states exists; increase n until $\psi_n^{\mathcal{S}}$ is satisfiable

Idea

Extend Heule and Verwer's method

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

Encoding Deterministic Automata

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

- $d_{p,a,q}$ 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 $\psi_n^{\mathcal{S}}$ that is satisfiable if and only if a consistent DFA with n states exists; increase n until $\psi_n^{\mathcal{S}}$ is satisfiable

Encoding Deterministic Automata

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

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

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

$$\bigwedge_{p \in Q} \bigwedge_{a \in \Sigma} \left(\bigvee_{q \in Q} d_{p,a,q} \right)$$

Constraints

Use Boolean variables $x_{u,q}$ where $u \in Pref(\mathcal{S})$ and $q \in Q$

- Consistency with S_+ and S_- as before

Constraints

Use Boolean variables $x_{u,q}$ where $u \in Pref(\mathcal{S})$ and $q \in Q$

- Consistency with S_+ and S_- as before
- Consistency with existential implications

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

Use Boolean variables $x_{u,q}$ where $u \in Pref(\mathcal{S})$ and $q \in Q$

- Consistency with S_+ and S_- as before
- Consistency with existential implications

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

- Consistency with universal implications

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

The Max Planck Institute for Software Systems is offering opportunities:

- Internships
- PhD positions
- PostDoc positions

https://www.mpi-sws.org

Bibliography

[1] E. M. Gold: *Complexity of automaton identification from given data*, 1978

[2] J. Oncina and P. Garcia: *Inferring regular languages in polynomial update time*, 1992

[3] K. J. Lang, B. A. Pearlmutter, and R. A. Price: *Results of the Abbadingo One DFA Learning Competition*, 1998

[4] A. W. Biermann and J. A. Feldman: *On the synthesis of finite-state machines from samples of their behavior*, 1972

[5] O. Grinchtein, M. Leucker, and N. Piterman: *Inferring network invariants automatically*, 2006

[6] M. Heule and S. Verwer: *Exact DFA Identification Using SAT Solvers*, 2010

Bibliography – Continued

[7] D. Angluin: *Negative Results for Equivalence Queries*, 1990

[8] D. Angluin: *A Note on the Number of Queries Needed to Identify Regular Language*, 1981

[9] D. Angluin: *Learning Regular Sets From Queries and Counterexamples*, 1987

[10] M. J. Kearns and U. V. Vazirani: *Introduction to Computational Learning Theory*, 1994

[11] M. Leucker: *Learning Meets Verification*, 2006

[12] D. Neider and U. Topcu: *An Automaton Learning Approach to Solving Safety Games over Infinite Graphs*, 2016

A.

Appendix

Puzzle

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

Possible Solution

Use active learning:

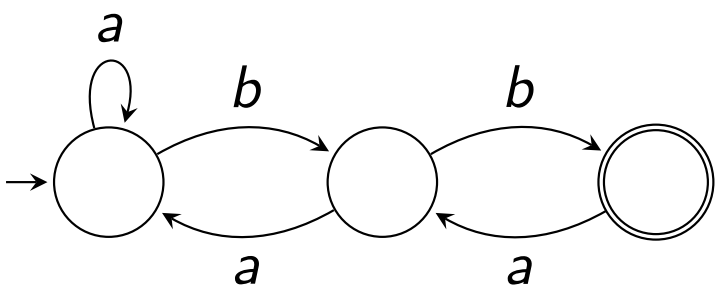
- A teacher can be constructed as follows:
  - to answer a membership query with a word $u \in \Sigma^*$, simulate the PDA $\mathcal{P}$ on u
  - to answer an equivalence query with a DFA $\mathcal{A}$, check $L(\mathcal{A}) \cap (\Sigma^* \setminus L(\mathcal{P})) = \emptyset$ and $L(\mathcal{P}) \cap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

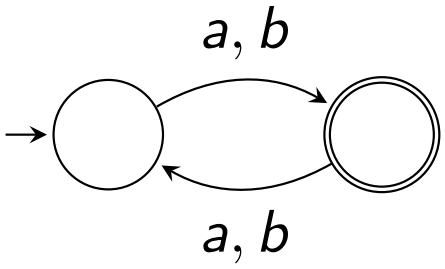
Challenge 1

Find a sample such that RPNI does <u>not</u> produce a minimal consistent DFA

Possible Solution

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

- RPNI: 

- Minimal consistent DFA: 

Challenge 2

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

- $\mathcal{A}_{L_n}$ 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 $L_n = \{a^n\}$

- $\mathcal{A}_{L_n}$ has $n + 2$ states and
- Angluin's algorithm requires 3 equivalence queries