京都大学
KYOTO UNIVERSITY

Active Learning of Symbolic Mealy Automata

Kengo Irie[1], **Masaki Waga[1,2]**, and Kohei Suenaga[1]

Kyoto University[1] , National Institute of Informatics[2]

2025 November 27th, ICTAC 2025

京都大学
KYOTO UNIVERSITY

Active Learning of Symbolic Mealy Automata

Kengo Irie[1], **Masaki Waga[1,2]**, and Kohei Suenaga[1]

Kyoto University[1] , National Institute of Informatics[2]

2025 November 27th, ICTAC 2025

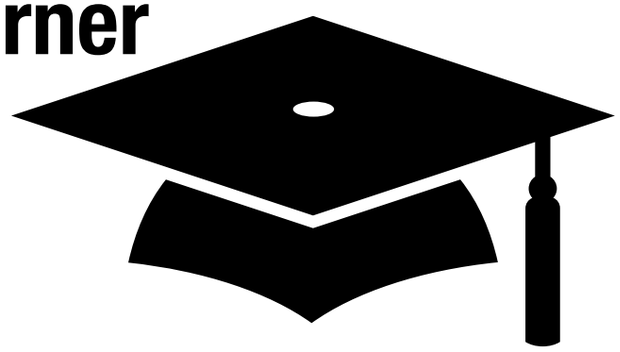
Active DFA Learning

[Angluin, Inf. Comput.'87]

Active DFA Learning

[Angluin, Inf. Comput.'87]

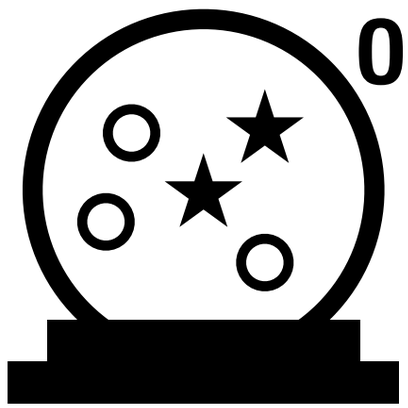
Learner



Active DFA Learning

[Angluin, Inf. Comput.'87]

Learner



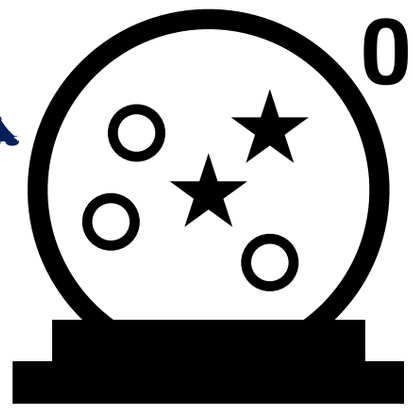
Oracle

Active DFA Learning

[Angluin, Inf. Comput.'87]

Questions: $w \overset{?}{\in} L_{\text{target}}$, $\mathscr{L}(\mathscr{A}_{\text{hyp}}) \overset{?}{=} L_{\text{target}}$

Learner



Oracle

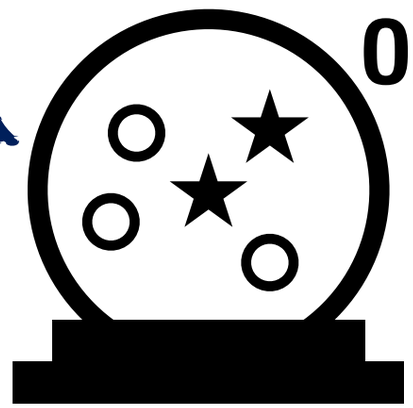
Active DFA Learning

[Angluin, Inf. Comput.'87]

Membership

Questions: $w \overset{?}{\in} L_{\text{target}}$, $\mathscr{L}(\mathscr{A}_{\text{hyp}}) \overset{?}{=} L_{\text{target}}$

Learner



Oracle

Active DFA Learning

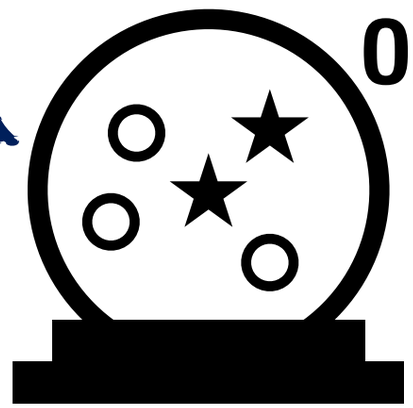
[Angluin, Inf. Comput.'87]

Membership

Equivalence

Questions: $w \overset{?}{\in} L_{\text{target}}$, $\mathscr{L}(\mathscr{A}_{\text{hyp}}) \overset{?}{=} L_{\text{target}}$

Learner



Oracle

Active DFA Learning

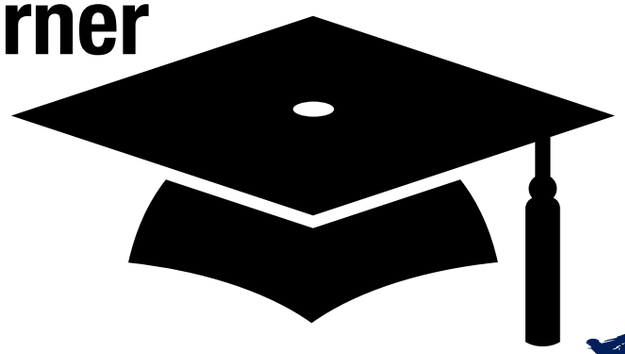
[Angluin, Inf. Comput.'87]

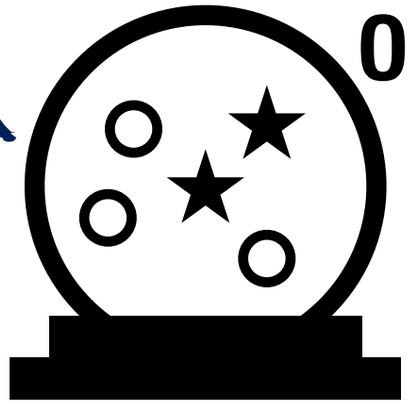
Membership

Equivalence

Questions: $w \stackrel{?}{\in} L_{\text{target}}$, $\mathscr{L}(\mathscr{A}_{\text{hyp}}) \stackrel{?}{=} L_{\text{target}}$

Learner





Oracle

Answers: Yes, No, Evidence $w \in \mathscr{L}(\mathscr{A}_{\text{hyp}}) \triangle L_{\text{target}}$

Active DFA Learning

[Angluin, Inf. Comput.'87]

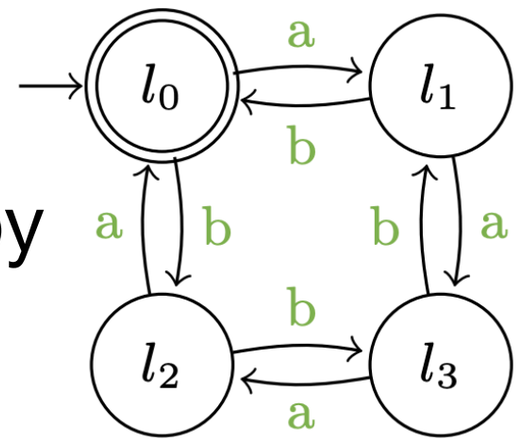
Membership **Equivalence**

Questions: $w \overset{?}{\in} L_{\text{target}}$, $\mathscr{L}(\mathscr{A}_{\text{hyp}}) \overset{?}{=} L_{\text{target}}$

Learner **Oracle**

Answers: Yes, No, Evidence $w \in \mathscr{L}(\mathscr{A}_{\text{hyp}}) \triangle L_{\text{target}}$

L_{target} is recognized by



Active DFA Learning

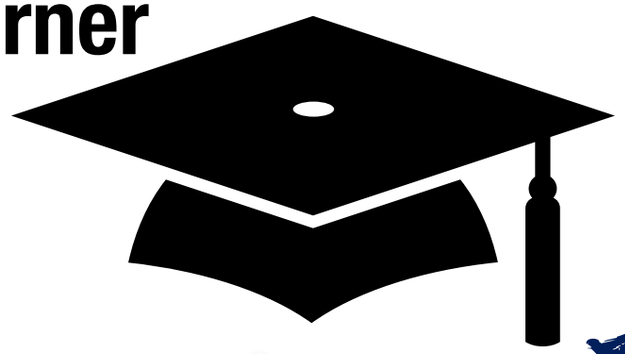
[Angluin, Inf. Comput.'87]

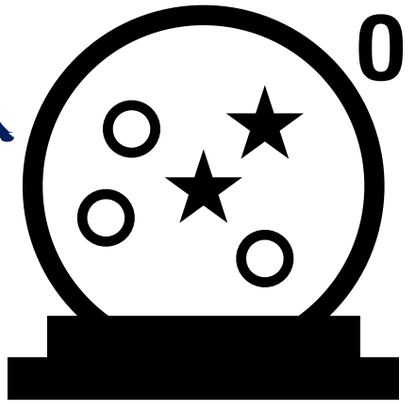
Membership

Equivalence

Questions: $w \overset{?}{\in} L_{\text{target}}$, $\mathscr{L}(\mathscr{A}_{\text{hyp}}) \overset{?}{=} L_{\text{target}}$

Learner

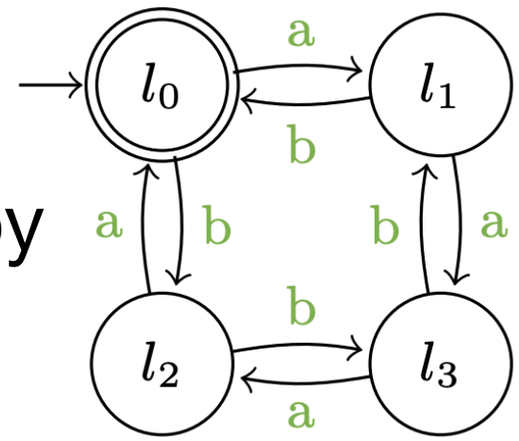




Oracle

Answers: Yes, No, Evidence $w \in \mathscr{L}(\mathscr{A}_{\text{hyp}}) \triangle L_{\text{target}}$

L_{target} is recognized by



Studied from both AI/ML and FM/SE/PL

Extensions of Active Automata Learning

- Learning with fewer queries in practice/theory
  - e.g. better analysis of counterexamples, better data structures, …
- Learning automata with multiple outputs
  - e.g. Mealy/Moore machines
- Learning automata over infinite alphabets
  - e.g. symbolic automata, nominal automata, …
- …

Extensions of Active Automata Learning

- Learning with fewer queries in practice/theory
  - e.g. better analysis of counterexamples, better data structures, …
- Learning automata with multiple outputs
  - e.g. Mealy/Moore machines
- Learning automata over infinite alphabets
  - e.g. symbolic automata, nominal automata, …
- …

e.g. Approx. black-box systems as automata

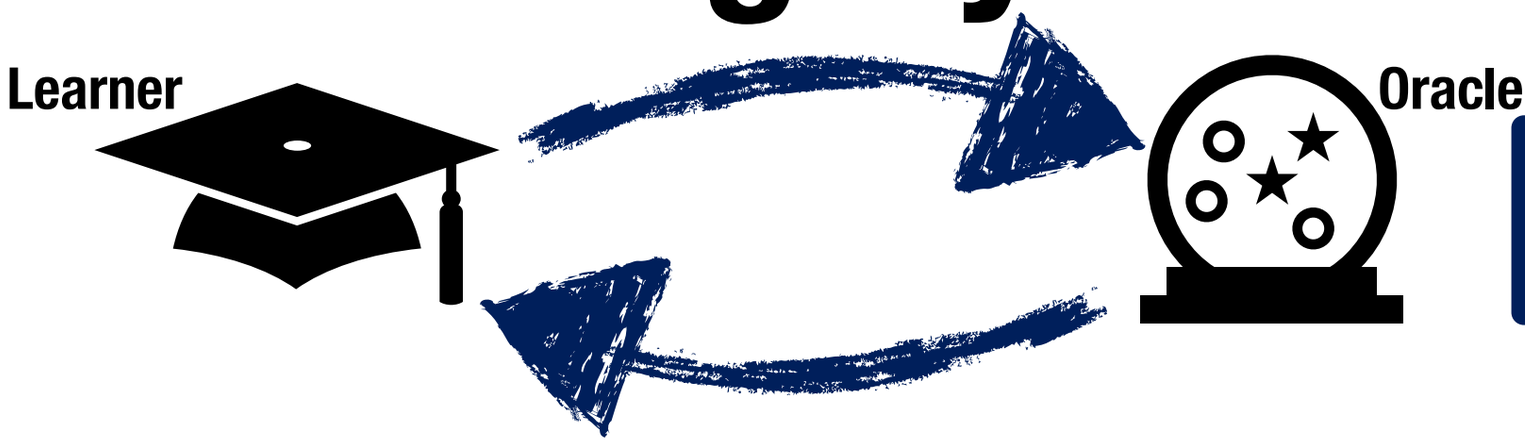
Learning Symbolic Automata

[Drews & D'Antoni, TACAS'17]

Automata w/ predicates on trans.
for infinite/huge alphabets, e.g. $\mathbb{N}$

Learning Symbolic Automata

[Drews & D'Antoni, TACAS'17]



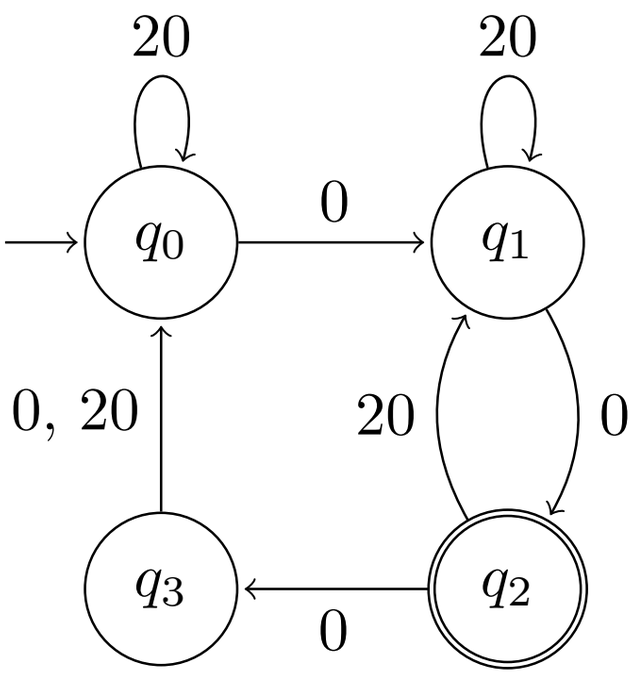
Automata w/ predicates on trans.
for infinite/huge alphabets, e.g. $\mathbb{N}$

Learning Symbolic Automata

[Drews & D'Antoni, TACAS'17]

Automata w/ predicates on trans.
for infinite/huge alphabets, e.g. $\mathbb{N}$

Learn a DFA over
a part of the alphabet
e.g. $\subsetneq \mathbb{N}$

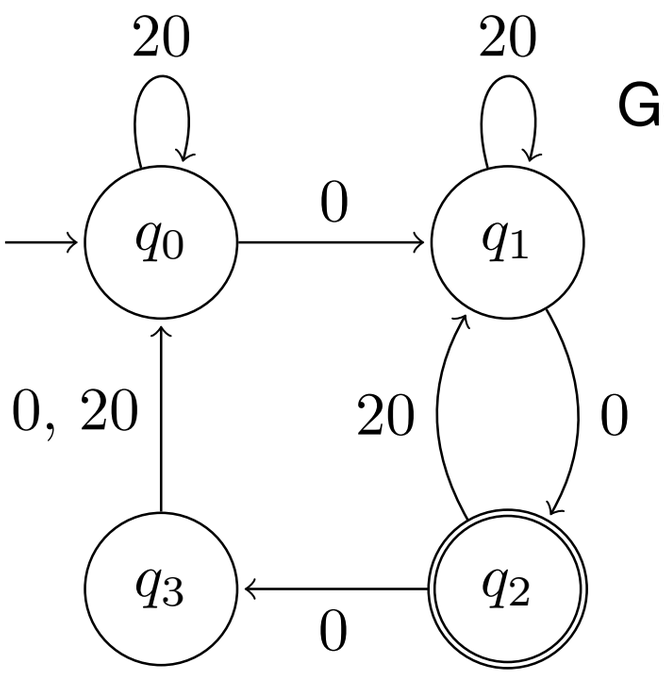


Learning Symbolic Automata

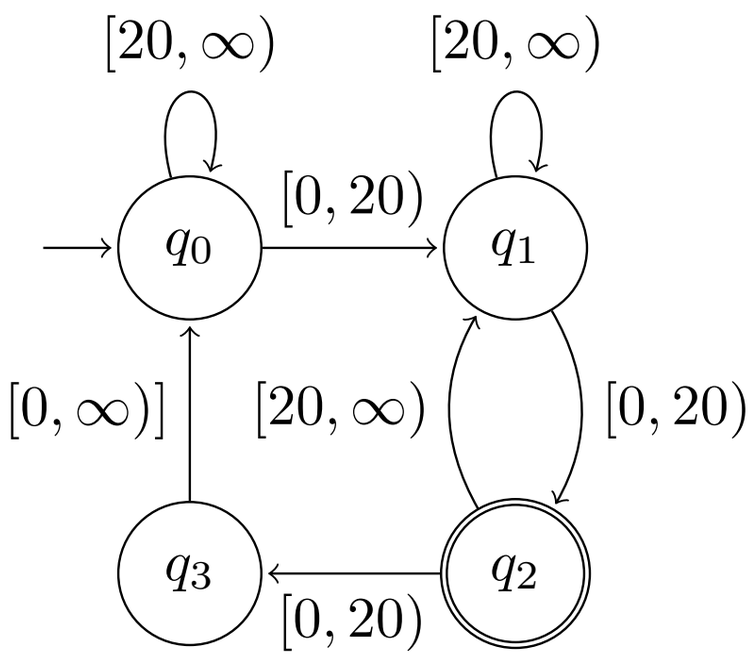
[Drews & D'Antoni, TACAS'17]

Automata w/ predicates on trans.
for infinite/huge alphabets, e.g. $\mathbb{N}$

Learn a DFA over
a part of the alphabet
e.g. $\subsetneq \mathbb{N}$



Generalize concrete characters
into predicates
e.g. intervals over $\mathbb{N}$

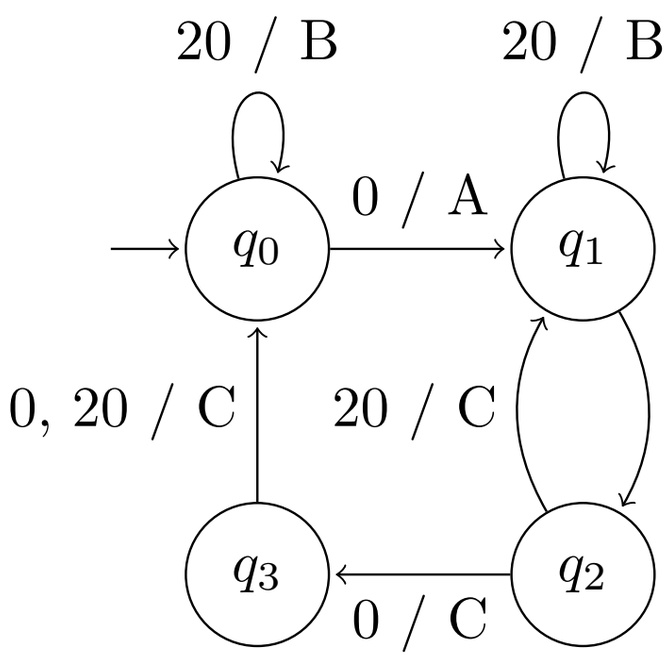


Learning Symbolic Mealy Automata

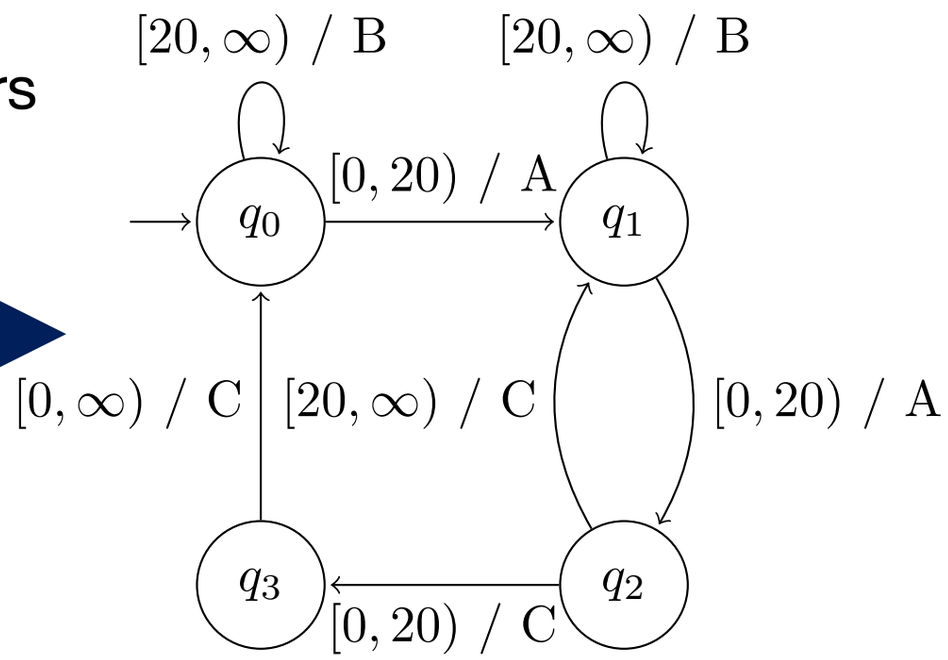
[Contributions]

Symbolic automata w/
transition-labeled outputs

Learn a *Mealy machine* over
a part of the alphabet
e.g. $\subsetneq \mathbb{N}$



Generalize concrete characters
into predicates
e.g. intervals over $\mathbb{N}$



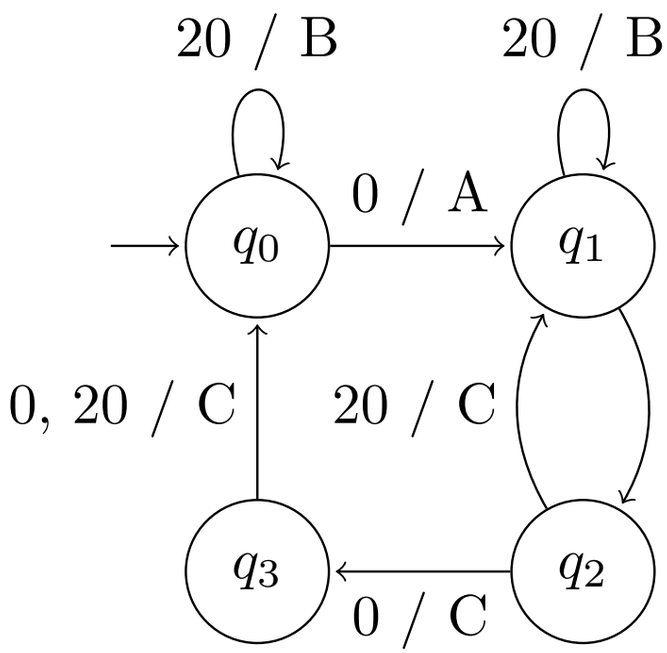
Learning Symbolic Mealy Automata

[Contributions]

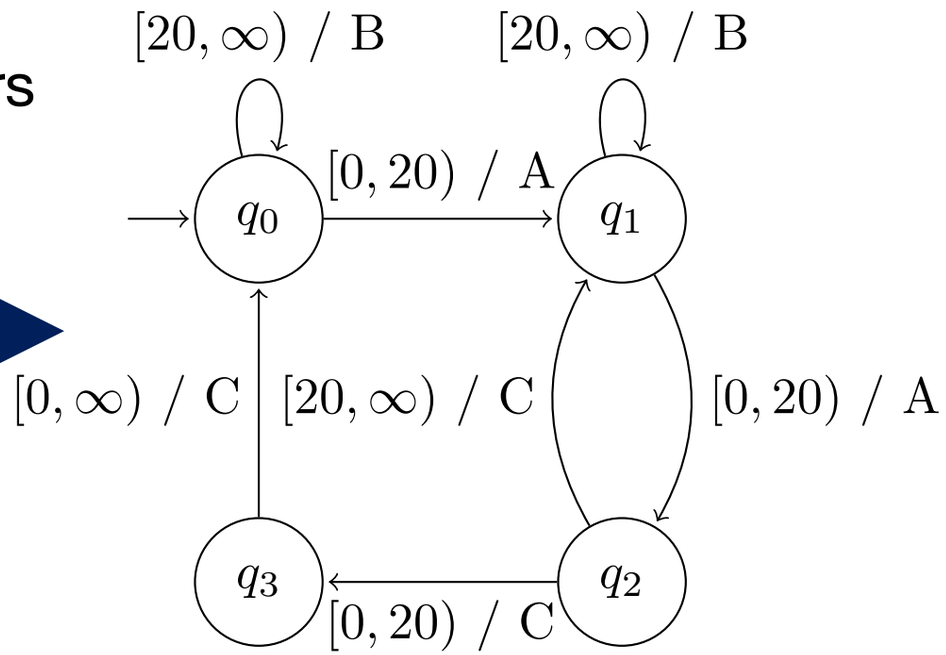
Symbolic automata w/
transition-labeled outputs

Learn a *Mealy machine* over
a part of the alphabet
e.g. $\subsetneq \mathbb{N}$

Naively, chars on learned transition/output functions may be different
→ Successor/output of some “transitions” may be undefined!!



Generalize concrete characters
into predicates
e.g. intervals over $\mathbb{N}$



Idea: Explicitly Learn "Essential" Characters

Σ_E: Finite set of current essential characters

- Σ_E incrementally increases
- New characters are found via equivalence queries

1. Learn a Mealy machine $\mathscr{M}^e$ over Σ_E ($\subsetneq \Sigma$)
2. Generalize the characters in $\mathscr{M}^e$ to predicates to obtain a symbolic Mealy automaton $\mathscr{M}$

Contributions

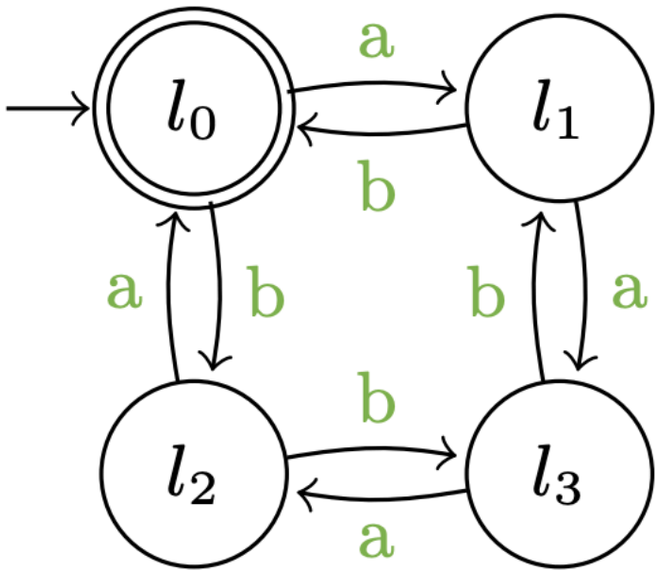
- Learning algorithm for symbolic Mealy automata
  - Idea: Explicitly learn “essential” characters Σ_E
- Complexity analysis of our algorithm
  - Worst case query complexity
  - An example to demonstrate the tightness
- Implementation + Experiments
  - Typically much more efficient than theoretical bound

Outline

- Preliminaries: Algorithms for active automata learning
  - Idea: Identify good prefixes/suffixes
  - Data structure: observation table
  - Learning algorithm for symbolic automata
- Our learning algorithm: Learning symbolic Mealy automata
  - Idea: Explicitly learn “essential” characters
- Experiments

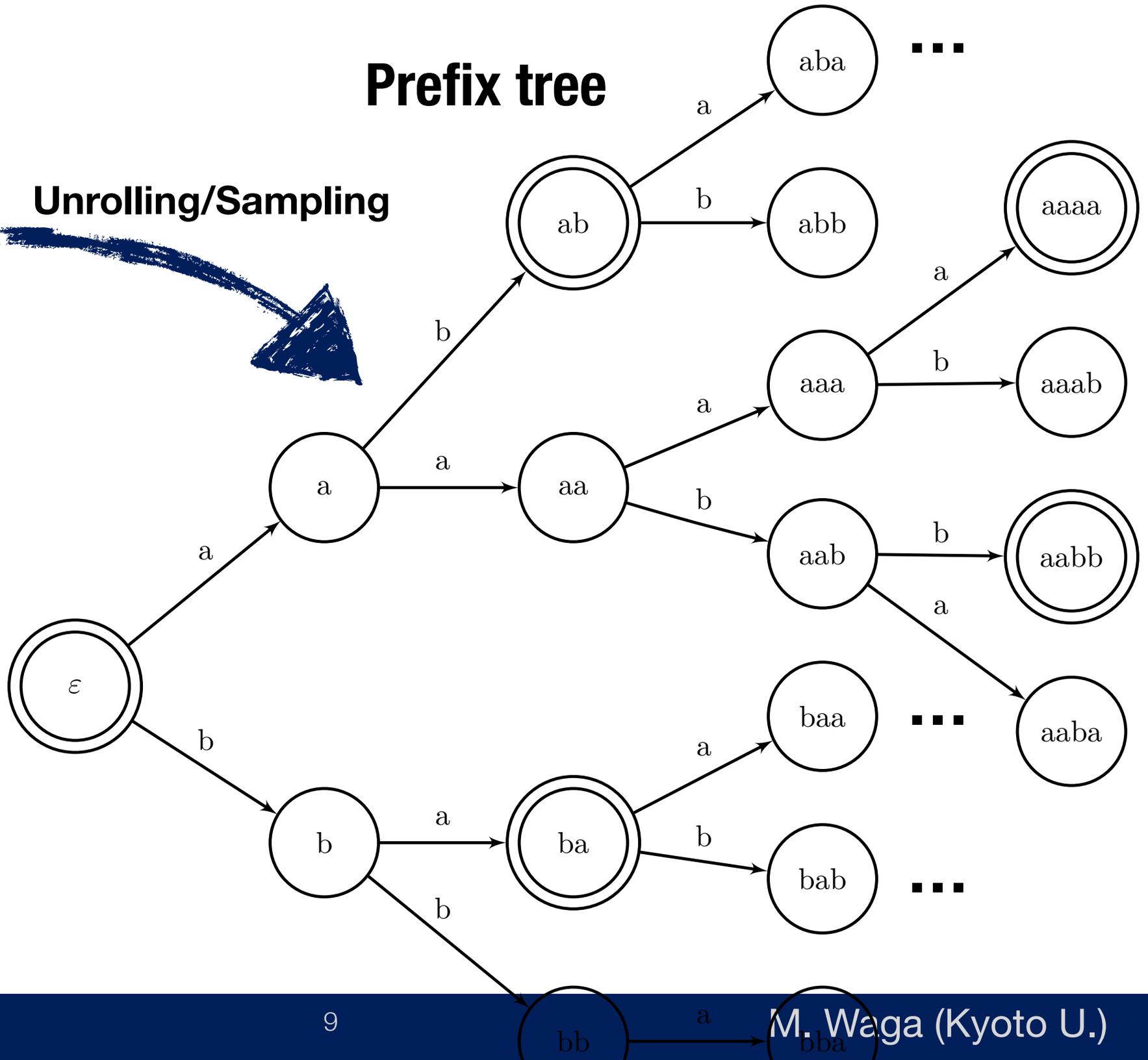
Idea: Find Good Prefixes/Suffixes

Target DFA



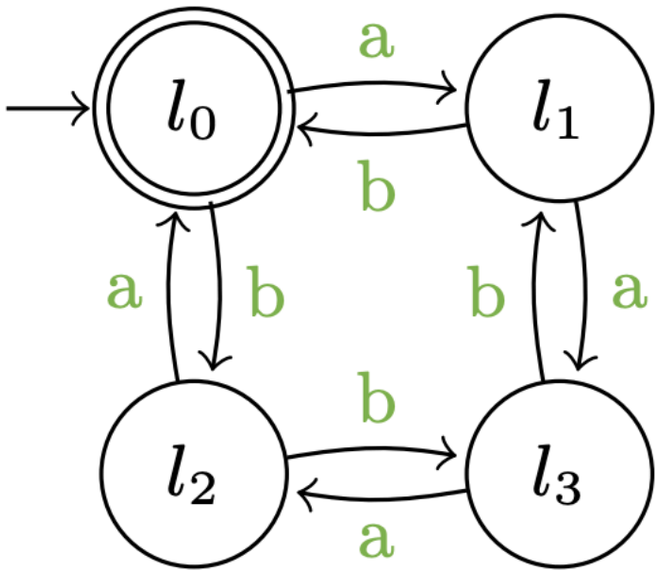
Unrolling/Sampling

Prefix tree



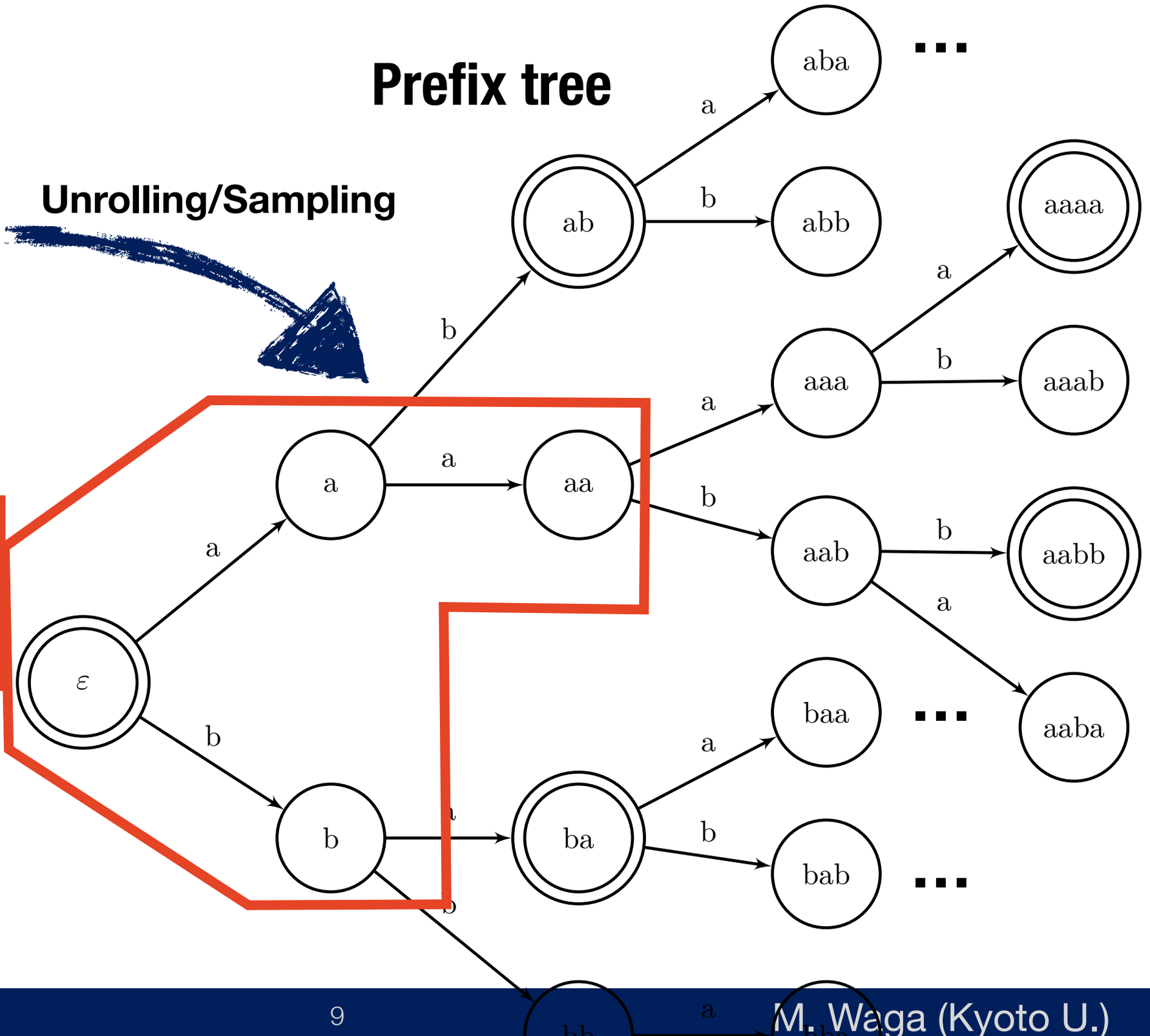
Idea: Find Good Prefixes/Suffixes

Target DFA



Unrolling/Sampling

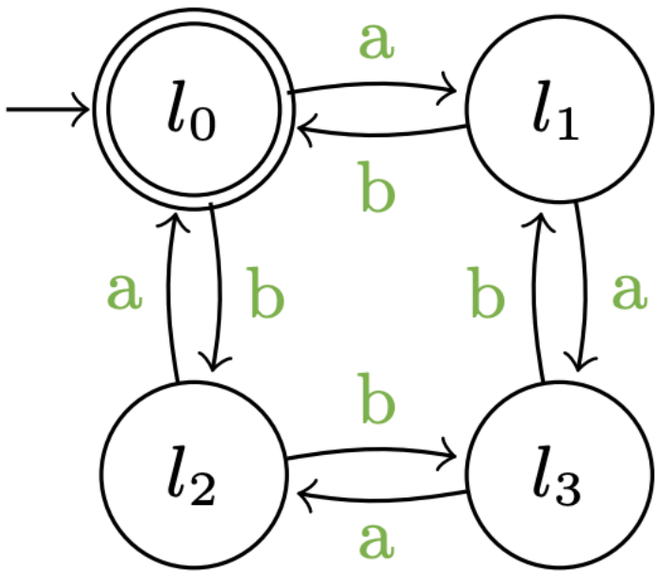
Prefix tree



Prefixes S to cover states

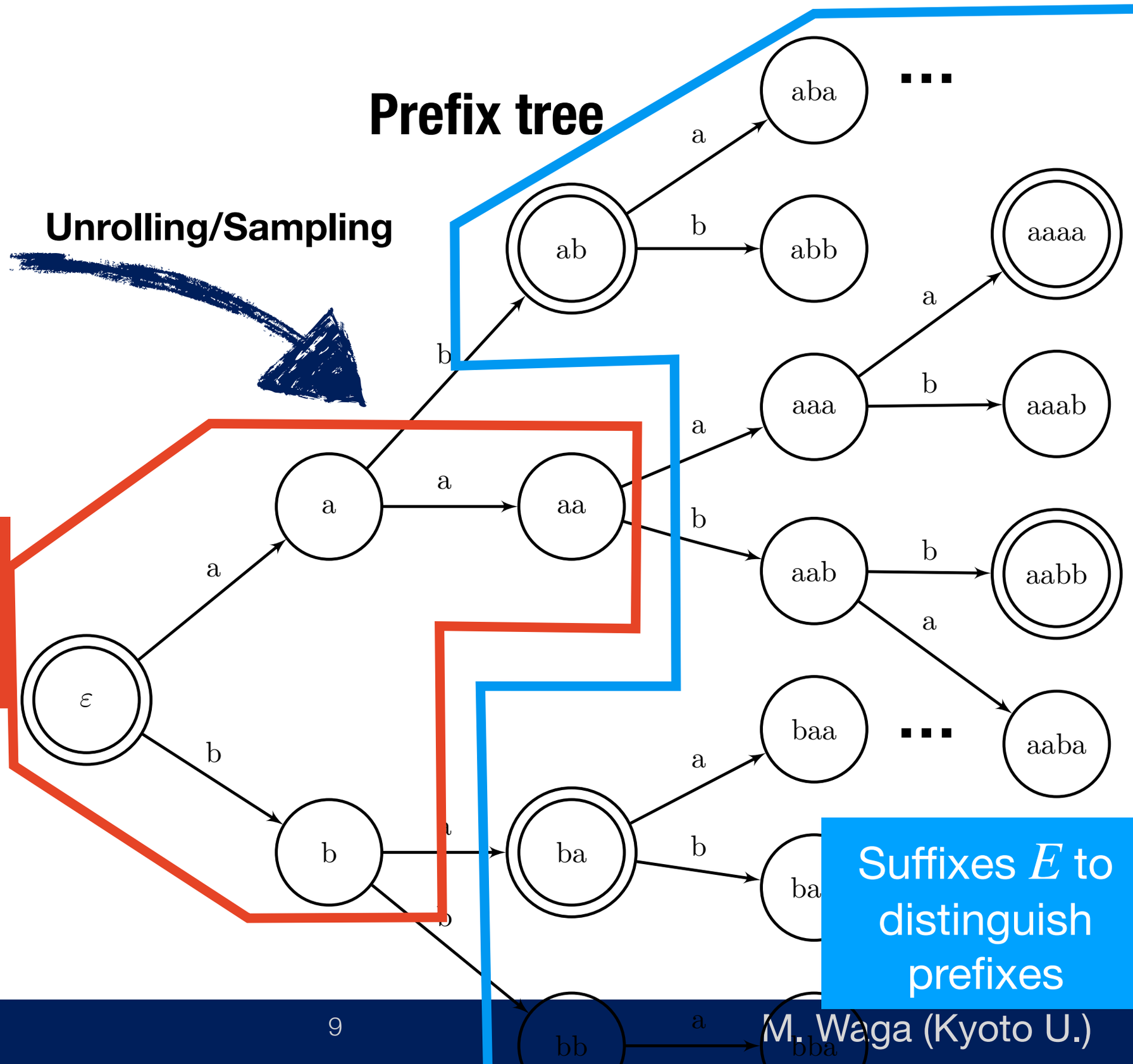
Idea: Find Good Prefixes/Suffixes

Target DFA



Unrolling/Sampling

Prefix tree

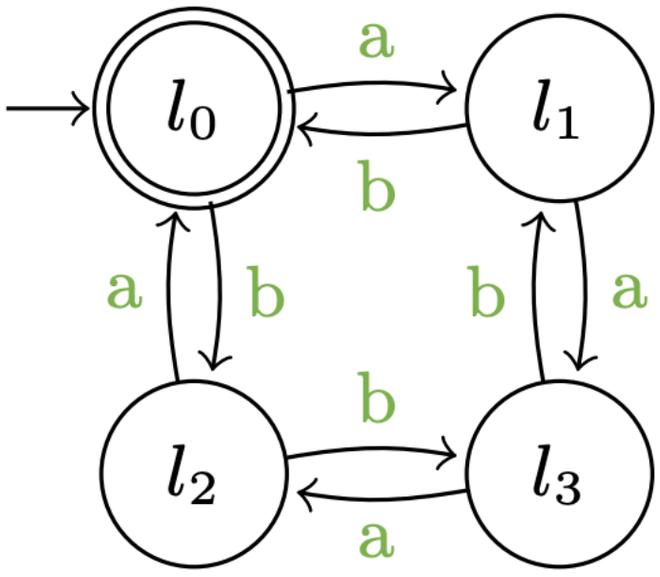


Prefixes S to cover states

Suffixes E to distinguish prefixes

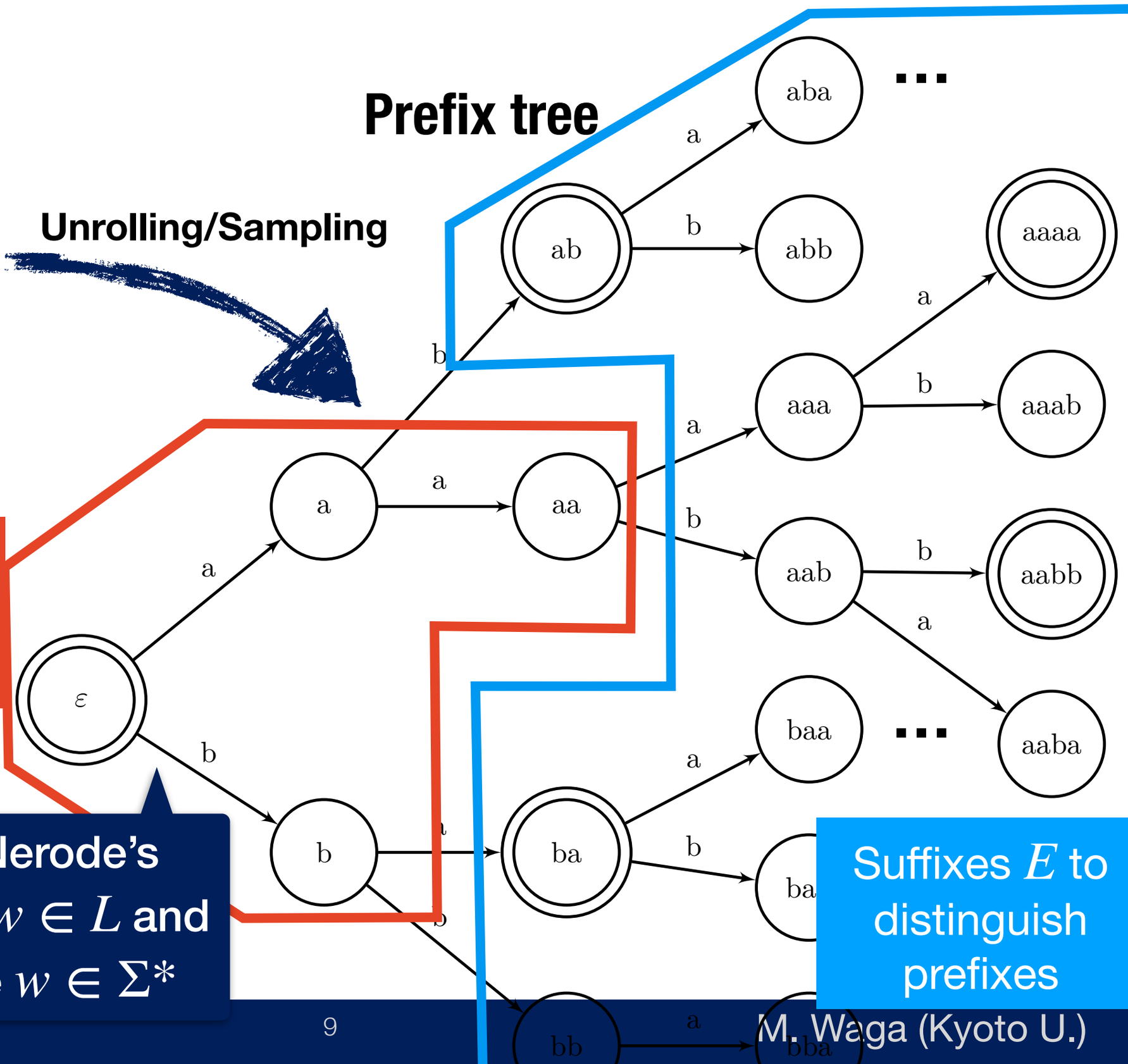
Idea: Find Good Prefixes/Suffixes

Target DFA



Unrolling/Sampling

Prefix tree



Prefixes S to cover states

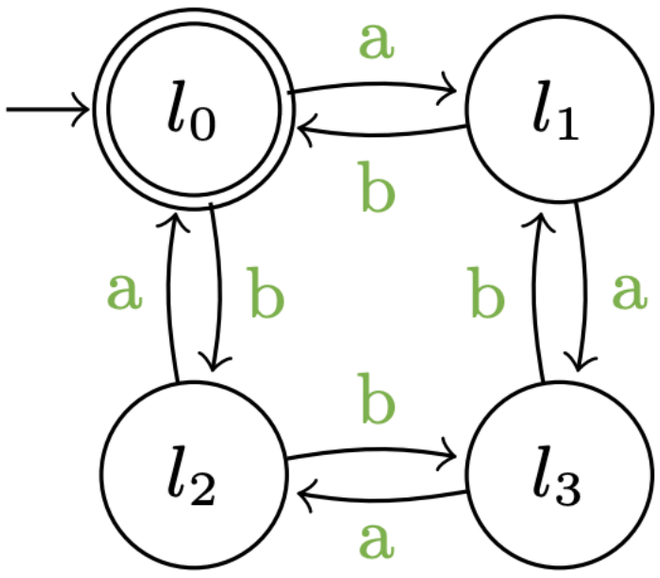
Suffixes E to distinguish prefixes

Inequivalent wrt. Nerode's congruence, i.e., $s \cdot w \in L$ and $s' \cdot w \notin L$ for some $w \in \Sigma^*$

Idea: /Suffixes

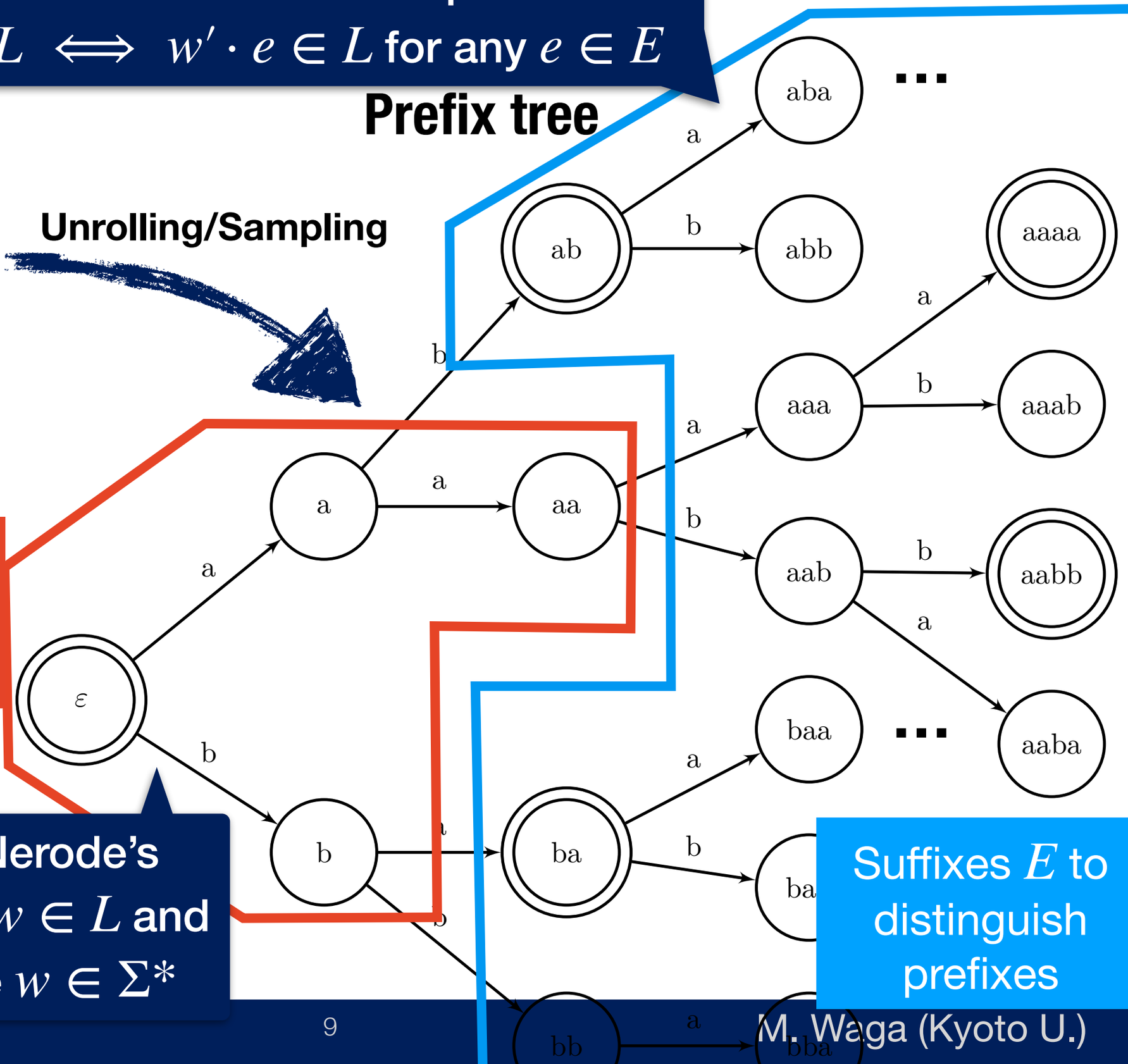
Test Nerode-equivalence, i.e.,
deem w and w' are Nerode-equivalent if
$w \cdot e \in L \iff w' \cdot e \in L$ for any $e \in E$

Target DFA



Unrolling/Sampling

Prefix tree



Prefixes S to cover states

Inequivalent wrt. Nerode's congruence, i.e., $s \cdot w \in L$ and $s' \cdot w \notin L$ for some $w \in \Sigma^*$

Suffixes E to distinguish prefixes

Observation Table for DFA Learning

[Angluin, Inf. Comput.'87]

Table showing the membership of concatenated words

Shows if $s \cdot e \in L$
e.g. if $\mathrm{a} \cdot \varepsilon = \mathrm{a} \in L$

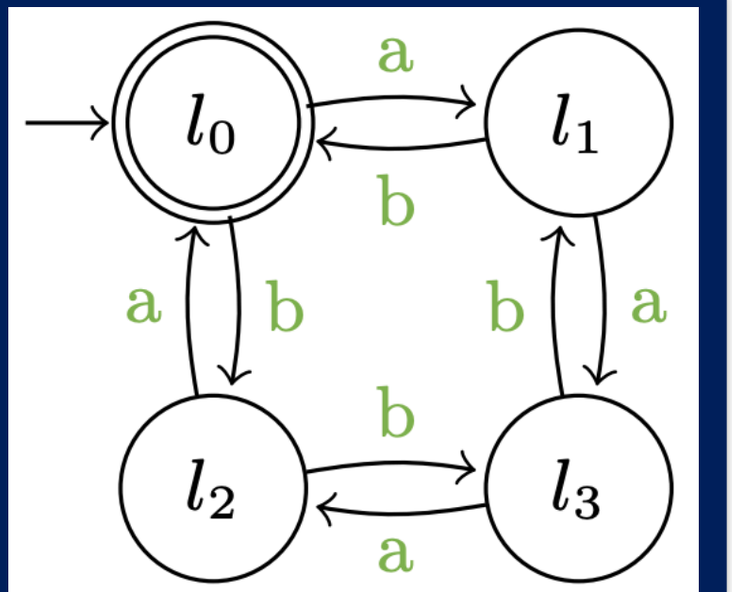
Suffixes E to distinguish prefixes

Prefixes S to cover states

$S \cdot \Sigma \backslash S$

	ε	a	b
ε	$\top$	$\bot$	$\bot$
a	$\bot$	$\bot$	$\top$
b	$\bot$	$\top$	$\bot$
aa	$\bot$	$\bot$	$\bot$
ab	$\top$	$\bot$	$\bot$
ba	$\top$	$\bot$	$\bot$
bb	$\bot$	$\bot$	$\bot$
aaa	$\bot$	$\top$	$\bot$
aab	$\bot$	$\bot$	$\top$

Table for



Observation Table for DFA Learning

[Angluin, Inf. Comput.'87]

Table showing the membership of concatenated words

Shows if $s \cdot e \in L$
e.g. if $\mathrm{a} \cdot \varepsilon = \mathrm{a} \in L$

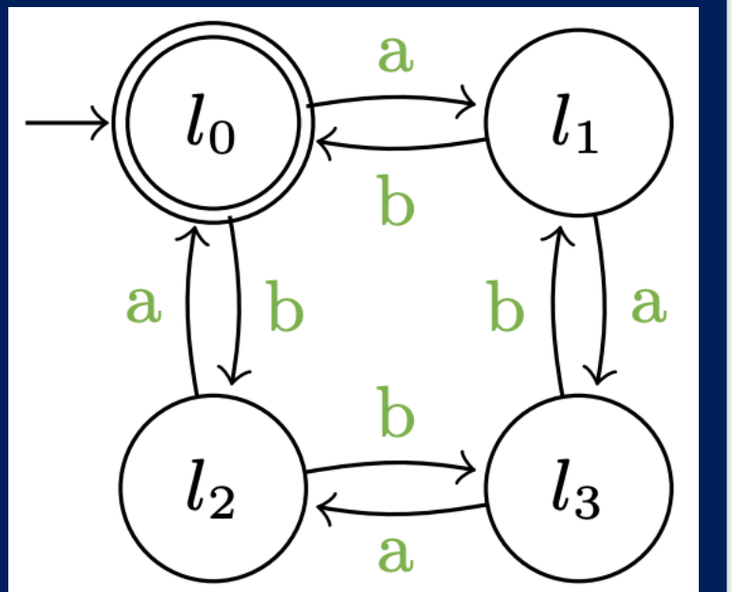
Suffixes E to distinguish prefixes

Prefixes S to cover states

$S \cdot \Sigma \backslash S$

	ε	a	b
ε	$\top$	$\bot$	$\bot$
a	$\bot$	$\bot$	$\top$
b	$\bot$	$\top$	$\bot$
aa	$\bot$	$\bot$	$\bot$
ab	$\top$	$\bot$	$\bot$
ba	$\top$	$\bot$	$\bot$
bb	$\bot$	$\bot$	$\bot$
aaa	$\bot$	$\top$	$\bot$
aab	$\bot$	$\bot$	$\top$

Table for



Each $s \in S$ corresponds to one state

Observation Table for DFA Learning

[Angluin, Inf. Comput.'87]

Table showing the membership of concatenated words

Shows if $s \cdot e \in L$
e.g. if $\mathrm{a} \cdot \varepsilon = \mathrm{a} \in L$

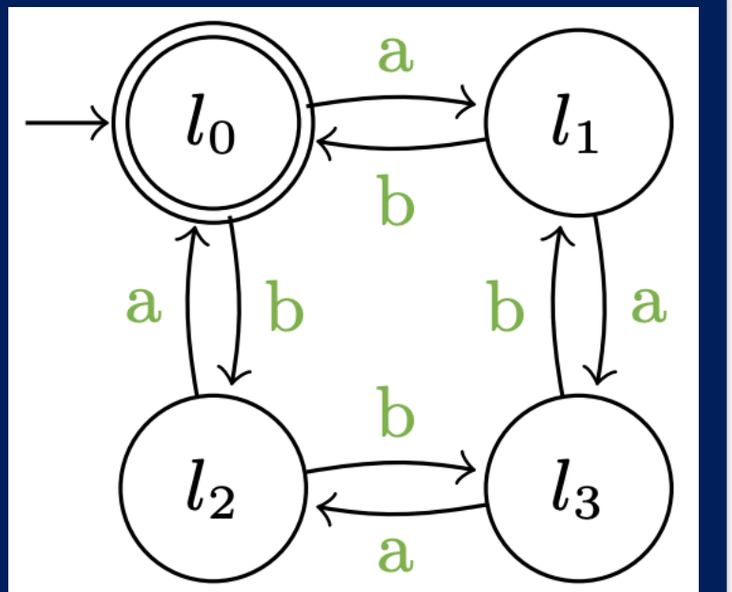
Suffixes E to distinguish prefixes

Prefixes S to cover states

	ε	a	b
ε	$\top$	$\bot$	$\bot$
a	$\bot$	$\bot$	$\top$
b	$\bot$	$\top$	$\bot$
aa	$\bot$	$\bot$	$\bot$
ab	$\top$	$\bot$	$\bot$
ba	$\top$	$\bot$	$\bot$
bb	$\bot$	$\bot$	$\bot$
aaa	$\bot$	$\top$	$\bot$
aab	$\bot$	$\bot$	$\top$

$S \cdot \Sigma \backslash S$

Table for



Each $s \in S$ corresponds to one state

Used to define successors

Observation Table for DFA Learning

[Angluin, Inf. Comput.'87]

Table showing the membership of concatenated words

Shows if $s \cdot e \in L$
e.g. if $\mathrm{a} \cdot \varepsilon = \mathrm{a} \in L$

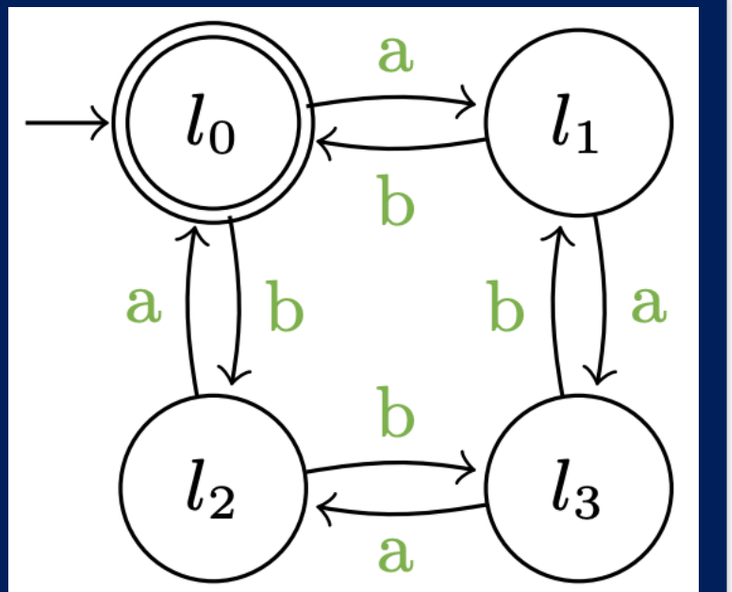
Suffixes E to distinguish prefixes

Prefixes S to cover states

	ε	a	b
ε	$\top$	$\bot$	$\bot$
a	$\bot$	$\bot$	$\top$
b	$\bot$	$\top$	$\bot$
aa	$\bot$	$\bot$	$\bot$
ab	$\top$	$\bot$	$\bot$
ba	$\top$	$\bot$	$\bot$
bb	$\bot$	$\bot$	$\bot$
aaa	$\bot$	$\top$	$\bot$
aab	$\bot$	$\bot$	$\top$

$S \cdot \Sigma \backslash S$

Table for



Each $s \in S$ corresponds to one state

Used to define successors

Observation Table for DFA Learning

[Angluin, Inf. Comput.'87]

Table showing the membership of concatenated words

Shows if $s \cdot e \in L$
e.g. if $\mathrm{a} \cdot \varepsilon = \mathrm{a} \in L$

Suffixes E to distinguish prefixes

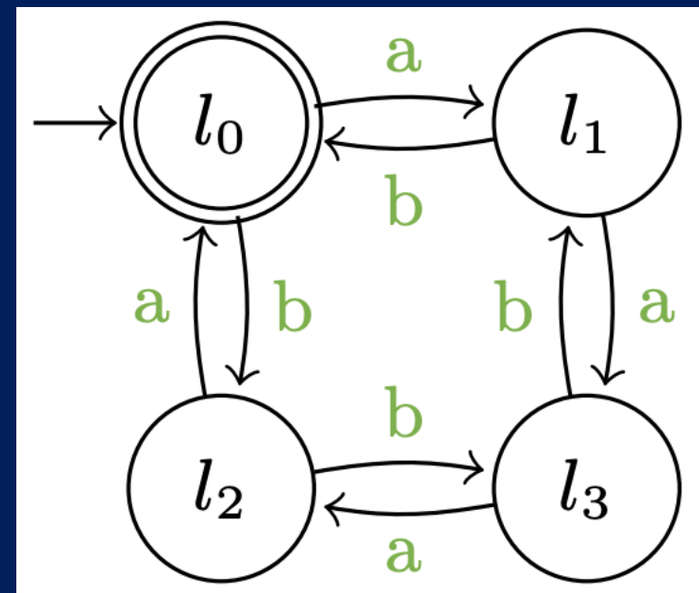
Prefixes S to cover states

Successor with a

$S \cdot \Sigma \backslash S$

	ε	a	b
ε	$\top$	$\bot$	$\bot$
a	$\bot$	$\bot$	$\top$
b	$\bot$	$\top$	$\bot$
aa	$\bot$	$\bot$	$\bot$
ab	$\top$	$\bot$	$\bot$
ba	$\top$	$\bot$	$\bot$
bb	$\bot$	$\bot$	$\bot$
aaa	$\bot$	$\top$	$\bot$
aab	$\bot$	$\bot$	$\top$

Table for



Each $s \in S$ corresponds to one state

Used to define successors

Observation Table for DFA Learning

[Angluin, Inf. Comput.'87]

Table showing the membership of concatenated words

Shows if $s \cdot e \in L$
e.g. if $\mathrm{a} \cdot \varepsilon = \mathrm{a} \in L$

Suffixes E to distinguish prefixes

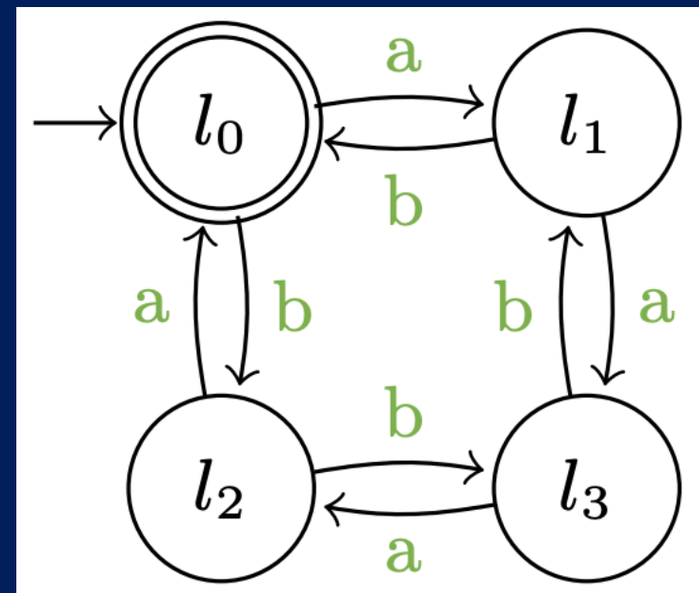
Prefixes S to cover states

Successor with b

$S \cdot \Sigma \backslash S$

	ε	a	b
ε	$\top$	$\bot$	$\bot$
a	$\bot$	$\bot$	$\top$
b	$\bot$	$\top$	$\bot$
aa	$\bot$	$\bot$	$\bot$
ab	$\top$	$\bot$	$\bot$
ba	$\top$	$\bot$	$\bot$
bb	$\bot$	$\bot$	$\bot$
aaa	$\bot$	$\top$	$\bot$
aab	$\bot$	$\bot$	$\top$

Table for



Each $s \in S$ corresponds to one state

Used to define successors

Observation Table for DFA Learning

[Angluin, Inf. Comput.'87]

Table showing the membership of concatenated words

Shows if $s \cdot e \in L$
e.g. if $\mathrm{a} \cdot \varepsilon = \mathrm{a} \in L$

Suffixes E to distinguish prefixes

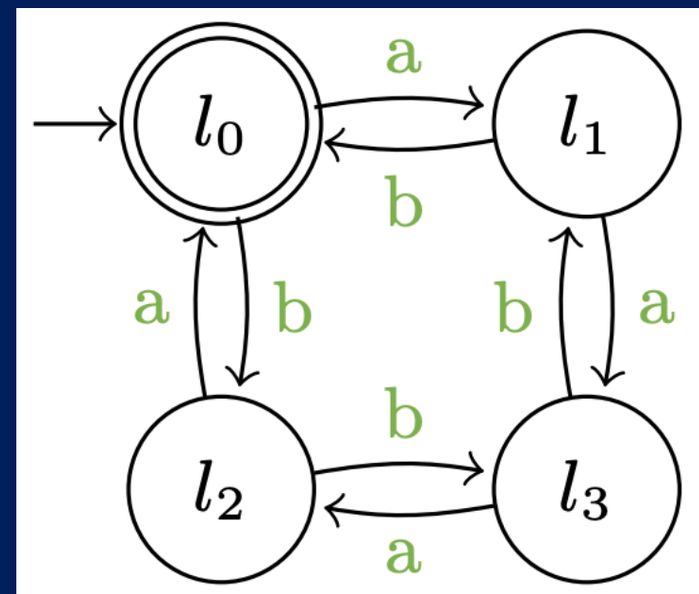
Prefixes S to cover states

Successor with b

$S \cdot \Sigma \backslash S$

	ε	a	b
ε	$\top$	$\bot$	$\bot$
a	$\bot$	$\bot$	$\top$
b	$\bot$	$\top$	$\bot$
aa	$\bot$	$\bot$	$\bot$
ab	$\top$	$\bot$	$\bot$
ba	$\top$	$\bot$	$\bot$
bb	$\bot$	$\bot$	$\bot$
aaa	$\bot$	$\top$	$\bot$
aab	$\bot$	$\bot$	$\top$

Table for



Equivalent!

Each $s \in S$ corresponds to one state

Used to define successors

Observation Table for Mealy Machines

[Niese, 2003]

Idea: Require $E \supseteq \Sigma$ to learn output function for each state

Output of $s \cdot e$
e.g. $\mathscr{M}(\mathrm{a} \cdot \varepsilon) = \mathscr{M}(\mathrm{a})$

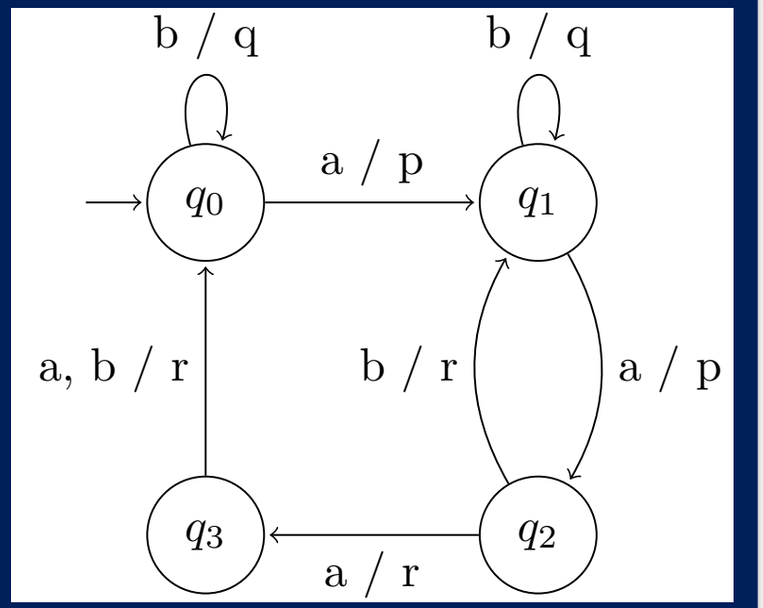
Suffixes E to distinguish prefixes

Prefixes S to cover states

$S \cdot \Sigma \backslash S$

	a	b	aa
ε	p	q	p
a	p	q	r
aa	r	r	r
aaa	r	r	p
b	p	q	p
ab	p	q	r
aab	p	q	r
aaaa	p	q	p
aaab	p	q	p

Table for



Observation Table for Mealy Machines

[Niese, 2003]

Idea: Require $E \supseteq \Sigma$ to learn output function for each state

Output of $s \cdot e$
e.g. $\mathscr{M}(\mathrm{a} \cdot \varepsilon) = \mathscr{M}(\mathrm{a})$

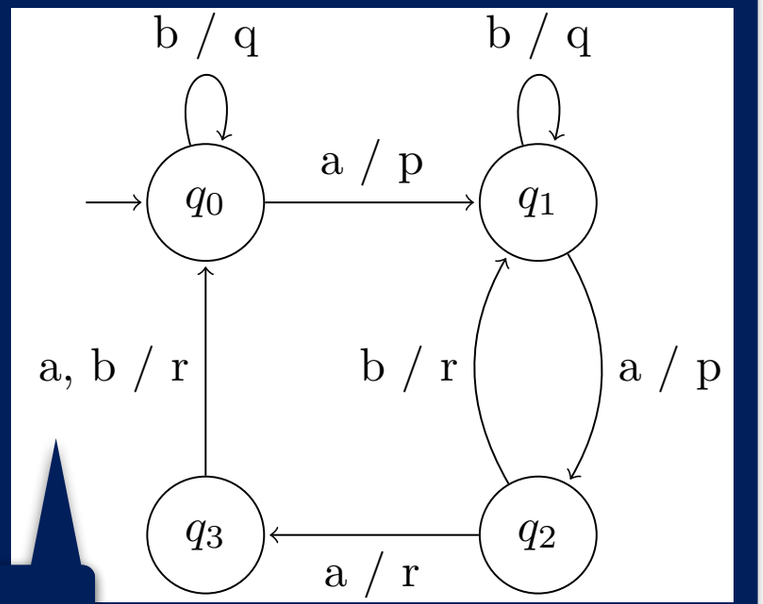
Suffixes E to distinguish prefixes

Prefixes S to cover states

$S \cdot \Sigma \backslash S$

	a	b	aa
ε	p	q	p
a	p	q	r
aa	r	r	r
aaa	r	r	p
b	p	q	p
ab	p	q	r
aab	p	q	r
aaaa	p	q	p
aaab	p	q	p

Table for



Output fun. for q_3

Observation Table for Symbolic Automata

[Drews & D'Antoni, TACAS'17]

Use finite R containing some of $S \cdot \Sigma$ to learn partial successors

Shows if $s \cdot e \in L$
e.g. if $0 \cdot \varepsilon = 0 \in L$

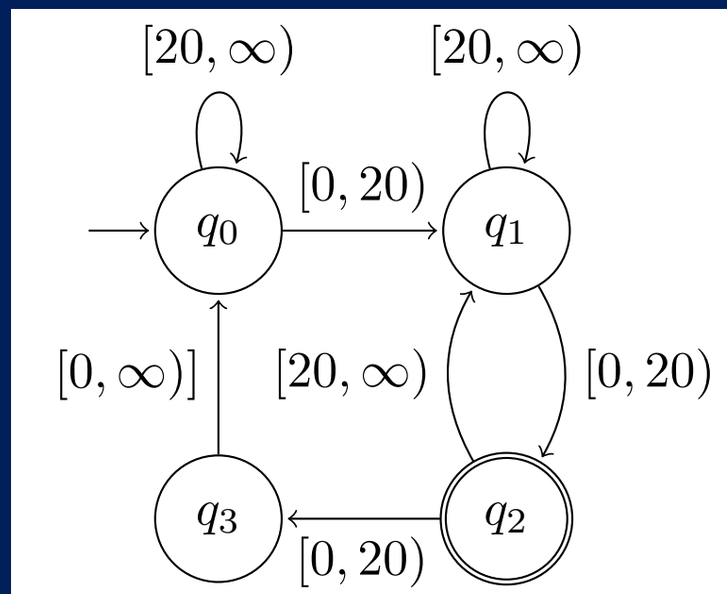
Suffixes E to distinguish prefixes

Prefixes S to cover states

R contains some of $S \cdot \Sigma \backslash S$

	ε	0	20
ε	$\top$	$\bot$	$\bot$
0	$\bot$	$\bot$	$\top$
20	$\bot$	$\top$	$\bot$
$0,0$	$\bot$	$\bot$	$\bot$
$0,20$	$\top$	$\bot$	$\bot$
$20,0$	$\top$	$\bot$	$\bot$
$20,20$	$\bot$	$\bot$	$\bot$
$0,0,0$	$\bot$	$\top$	$\bot$
$0,0,20$	$\bot$	$\bot$	$\top$

Table for



Observation Table for Symbolic Automata

[Drews & D'Antoni, TACAS'17]

Use finite R containing some of $S \cdot \Sigma$ to learn partial successors

Shows if $s \cdot e \in L$
e.g. if $0 \cdot \varepsilon = 0 \in L$

Suffixes E to distinguish prefixes

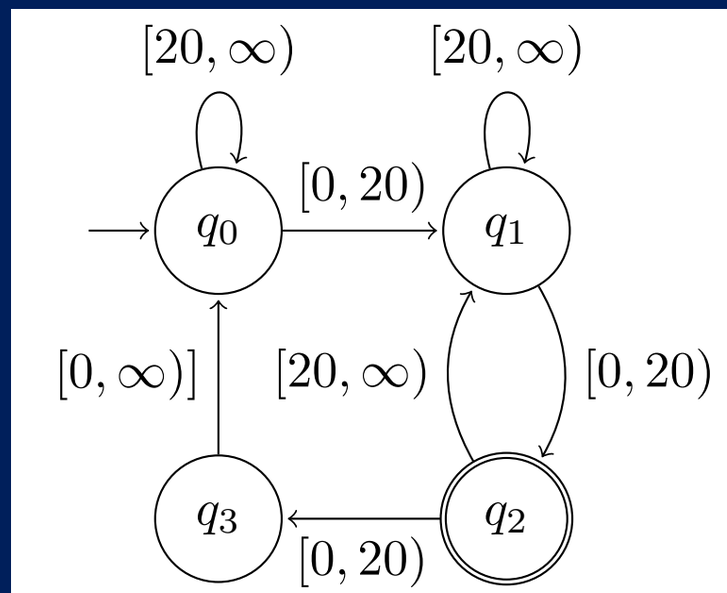
Prefixes S to cover states

	ε	0	20
ε	$\top$	$\bot$	$\bot$
0	$\bot$	$\bot$	$\top$
20	$\bot$	$\top$	$\bot$
$0,0$	$\bot$	$\bot$	$\bot$
$0,20$	$\top$	$\bot$	$\bot$
$20,0$	$\top$	$\bot$	$\bot$
$20,20$	$\bot$	$\bot$	$\bot$
$0,0,0$	$\bot$	$\top$	$\bot$
$0,0,20$	$\bot$	$\bot$	$\top$

R contains some of $S \cdot \Sigma \backslash S$

Construct a partial transition function

Table for



Summary so far…

- Learn automata by identifying “essential” prefixes/suffixes
  - Data structure: observation table
- For Mealy machines:
  - Successors: Ensure row indices include $S \cdot \Sigma$
  - Outputs: Ensure column indices include Σ
- For symbolic automata, relax requirements on totality to
  1. learn a partial transition function
  2. generalize characters to predicates

Outline

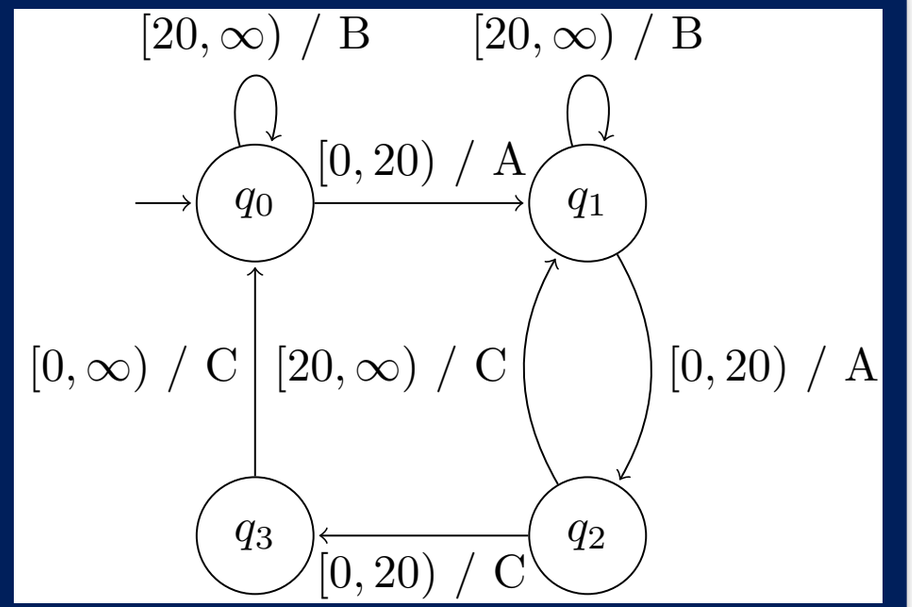
- Preliminaries: Algorithms for active automata learning
  - Idea: Identify good prefixes/suffixes
  - Data structure: observation table
  - Learning algorithm for symbolic automata
- Our learning algorithm: Learning symbolic Mealy automata
  - Idea: Explicitly learn “essential” characters
- Experiments

Constructing Symbolic Mealy Automata from Observation Tables

Issue: Successors/outputs may be undefined

	0	0,0
ε	A	A
0	A	C
0,0	C	C
0,0,0	C	A
20	A	A
0,20	A	C
0,0,20	A	C
0,0,0,0	A	A
0,0,0,20	A	A

Table for

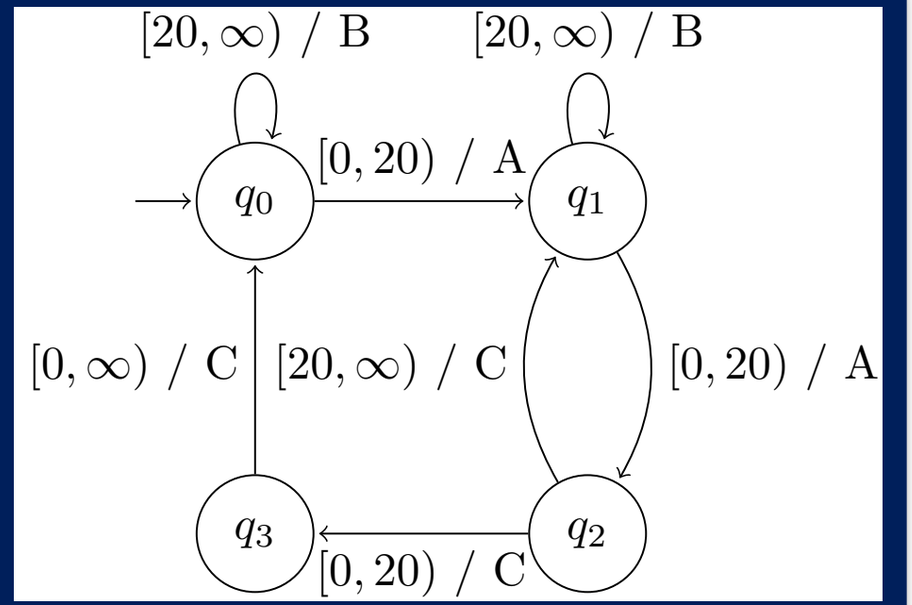


Constructing Symbolic Mealy Automata from Observation Tables

Issue: Successors/outputs may be undefined

	0	0,0
ε	A	A
0	A	C
0,0	C	C
0,0,0	C	A
20	A	A
0,20	A	C
0,0,20	A	C
0,0,0,0	A	A
0,0,0,20	A	A

Table for



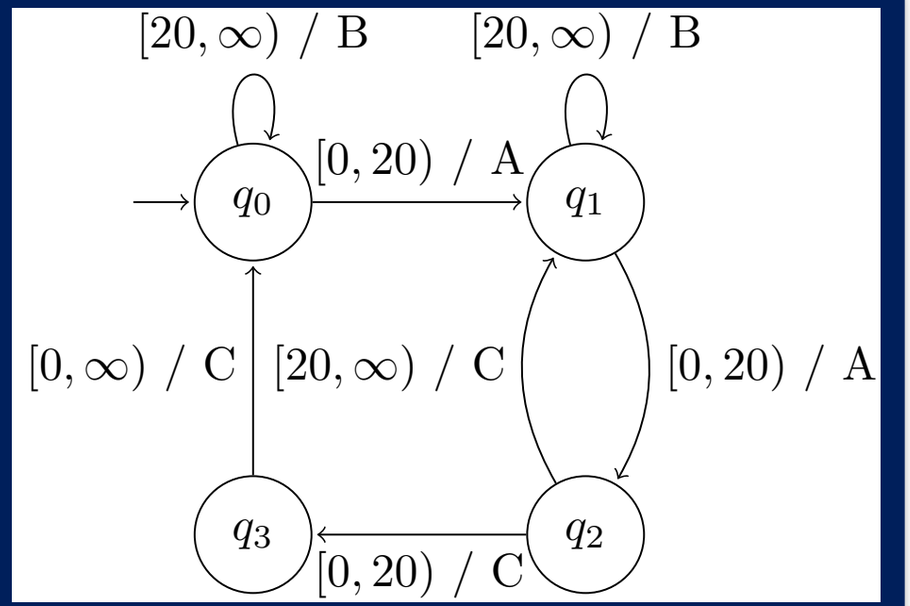
Constructing Symbolic Mealy Automata from Observation Tables

Issue: Successors/outputs may be undefined

	0	0,0
ε	A	A
0	A	C
0,0	C	C
0,0,0	C	A
20	A	A
0,20	A	C
0,0,20	A	C
0,0,0,0	A	A
0,0,0,20	A	A

Successor with 20

Table for



Constructing Symbolic Mealy Automata from Observation Tables

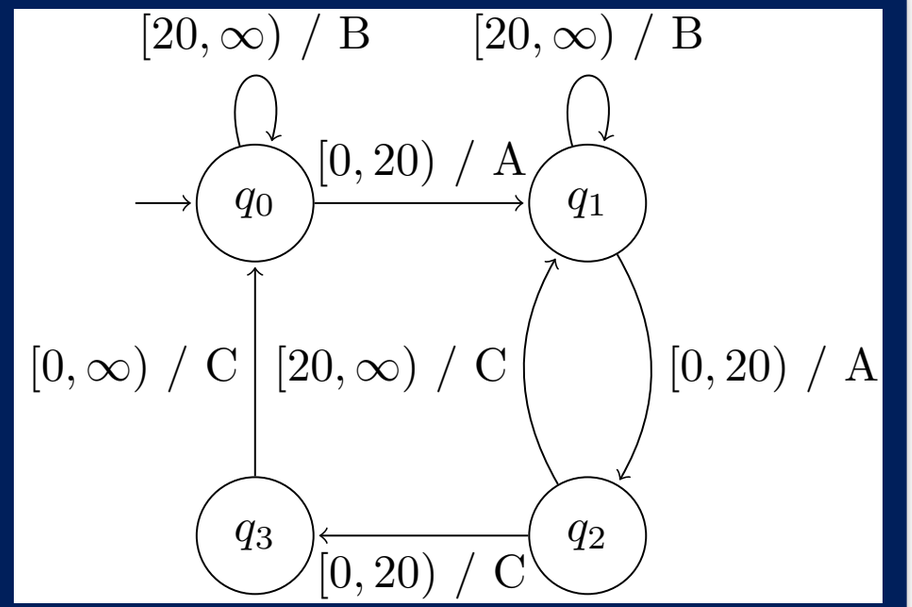
Issue: Successors/outputs may be undefined

Output for 20 is undefined

	0	0,0
ε	A	A
0	A	C
0,0	C	C
0,0,0	C	A
20	A	A
0,20	A	C
0,0,20	A	C
0,0,0,0	A	A
0,0,0,20	A	A

Successor with 20

Table for



Observation Table for Symbolic Mealy Autom.

[Contributions]

Idea: Explicitly identify essential characters Σ_E

Essential characters Σ_E

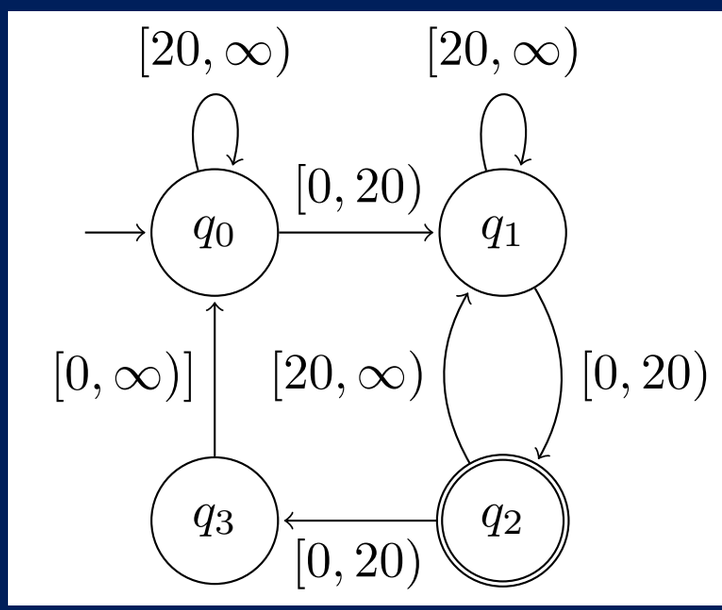
Suffixes E to distinguish prefixes

Prefixes S to cover states

R contains some of $S \cdot \Sigma \backslash S$

	0	20	0,0
ε	A	B	A
0	A	B	C
0,0	C	C	C
0,0,0	C	C	A
20	A	B	A
0,20	A	B	C
0,0,20	A	B	C
0,0,0,0	A	B	A
0,0,0,20	A	B	A

Table for



Output Closedness

[Contributions]

Definition

An observation table for symbolic Mealy automata is *output-closed* if for any $s, s \cdot a \in S \cup R, a \in \Sigma_E$ holds

Lemma

Observation table can be made output-closed by increasing Σ_E

If the successor from $s \in S$ with $a \in \Sigma$ is defined in an *output-closed* observation table, the output from s with a is also defined

→ We can construct a Mealy machine over Σ_E

Outline of Symbolic Mealy Learning

1. Initialize observation table
2. Make observation table satisfy certain properties, including output closedness
3. Construct a Mealy machine $\mathscr{M}^e$ over Σ_E
4. Generalize characters into predicates to construct a symbolic Mealy automaton $\mathscr{M}$
5. Ask equivalence query with $\mathscr{M}$
   - If $\mathscr{M}$ is equivalent to target, return $\mathscr{M}$
   - Otherwise, add prefixes of the counterexample to R → Go back to Step 2

Correctness of Our Algorithm

[Contributions]

Theorem

If our algorithm for learning symbolic Mealy automata terminates, the returned symbolic Mealy automaton represents the same function as the target one.

The proof is immediate because the algorithm terminates only if the equivalence oracle returns “Yes”.

Complexity Analysis with Σ_E^{final}

- Σ_E^{final}: Σ_E at the end of learning

- New characters in Σ_E are found by equiv. queries
  → Σ_E^{final} depends on the "smartness" of the oracle

Note: Σ_E^{final} may not be finite in general, i.e., the learning algorithm may not terminate

- Example: for $\Sigma = \mathbb{Q}$, oracle can keep returning a new character that is closer to the actual threshold

Upper Bound of # of Queries

Theorem

The number of queries in our algorithms is bounded by

- $(|\Sigma_E^{final}| + m + 1) \times n^2 +$
  $(2m + |\Sigma_E^{final}| + 1) \times |\Sigma_E^{final}| \times n + m \times |\Sigma_E^{final}|^2$ for membership queries
- $n + |\Sigma_E^{final}|$ for equivalence queries

Polynomial queries (similar to classical results)

Complexity in state space and characters can independently make learning difficult

where

- m: maximum length of the counterexample in equiv. query
- n: number of states of the target symbolic Mealy automata

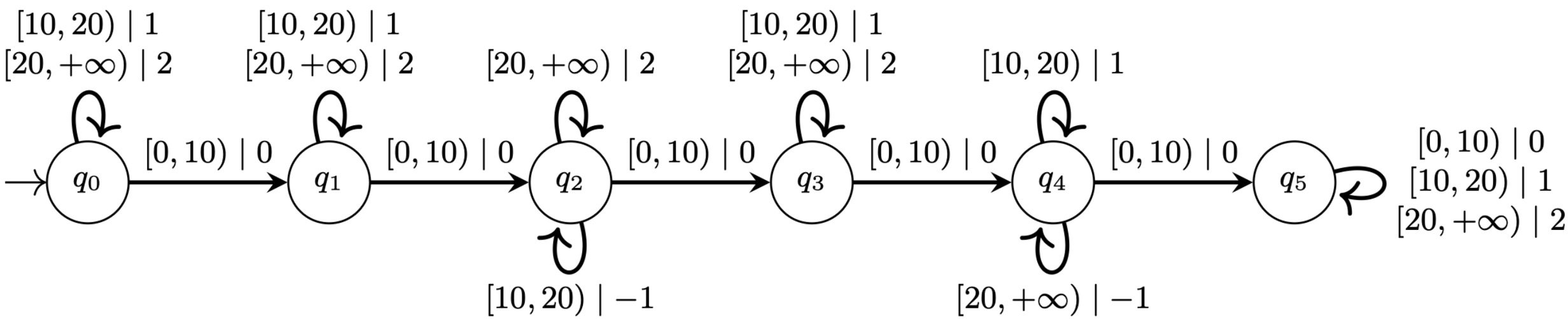
A Lower Bound of # of Queries

Theorem

For any $n, k \in \mathbb{N}$ s.t. $k \geq n \geq 2$, there is a target symbolic Mealy automaton with $2n$ such that $|\Sigma_E^{final}| = k$ and our algorithm requires at least $n + k$ equivalence queries.

Concrete example of the independent contribution of state space and characters to the learning complexity

Example ($n = k = 3$)



Outline

- Preliminaries: Algorithms for active automata learning
  - Idea: Identify good prefixes/suffixes
  - Data structure: observation table
  - Learning algorithm for symbolic automata
- Our learning algorithm: Learning symbolic Mealy automata
  - Idea: Explicitly learn “essential” characters
- Experiments

Implementation & Experiments

- Implemented our learning algorithm in Java
  - https://github.com/SoftwareFoundationGroupAtKyotoU/learningsma
- Two kinds of benchmarks
  - Random examples to observe the scalability
    - Generated 10 examples for each configuraton
  - “Practical” examples from MATLAB/Simulink models
    - Mars Helicopter (MH) and Automatic Transmission Gear System (ATGS)
- Intel Xeon E5-2687W v3, 251 GiB RAM, with Ubuntu 22.04.4 LTS

Scalability wrt Number of States

# of output queries scales almost linearly wrt # of states

← $|E|$ is almost 0 and quadratic term is ≈ 0

	# of states = 10	# of states = 20	# of states = 40	# of states = 80
$\lvert\Sigma_E^{final}\rvert$	10	10	10	10
# of output queries	1015.6	2035.95	4010.9	8186.59
# of eq. queries	10	10.1	10	10.18
$\lvert R\rvert$	91.56	181.57	361.09	722.55
$\lvert E\rvert$	0.0	0.1	0.0	0.2

Words of length 1 are typically enough to distinguish states

→ E is not used

Scalability wrt $|\Sigma_E^{final}|$

of output queries scales almost linearly wrt $|\Sigma_E^{final}|$

← $|E|$ is almost 0 and quadratic term is ≈ 0

	$\|\Sigma_E^{final}\| = 10$	$\|\Sigma_E^{final}\| = 20$	$\|\Sigma_E^{final}\| = 40$	$\|\Sigma_E^{final}\| = 80$
# of states	10	10	10	10
# of output queries	1015.6	4075.6	9104.7	16161.6
# of eq. queries	10	20	29.99	40
$\|R\|$	91.56	193.78	293.49	394.04
$\|E\|$	0.0	0.0	0.0	0.0

“Practical” Examples

Query complexity seems better than the worst-case bound, also for “practical” examples.

	Mars Helicopter	Automatic Transmission Gear System
# of output queries	6516	86446.8
# of equivalence queries	36	66
$\lvert \Sigma_E^{final} \rvert$	36	68
# of states	5	16
Max. length of cex.	4	12
$\lvert E \rvert$	0	6
Boolean dimensions in Σ	1	0
Numeric dimensions in Σ	3	2

“Practical” Examples

Query complexity seems better than the worst-case bound, also for “practical” examples.

	Mars Helicopter	Automatic Transmission Gear System
# of output queries	6516	86446.8
# of equivalence queries	36	66
$\lvert \Sigma_E^{final} \rvert$	36	68
# of states	5	16
Max. length of cex.	4	12
$\lvert E \rvert$	0	6
Boolean dimensions in Σ	1	0
Numeric dimensions in Σ	3	2

$\lvert E \rvert$ is small

“Practical” Examples

Query complexity seems better than the worst-case bound, also for “practical” examples.

	Mars Helicopter	Automatic Transmission Gear System
# of output queries	6516	86446.8
# of equivalence queries	36	66
$\lvert \Sigma_E^{final} \rvert$	36	68
# of states	5	16
Max. length of cex.	4	12
$\lvert E \rvert$	0	6
Boolean dimensions in Σ	1	0
Numeric dimensions in Σ	3	2

Less than half of the upper bound (14,309 and 177,408)

“Practical” Examples

Query complexity seems better than the worst-case bound, also for “practical” examples.

	Mars Helicopter	Automatic Transmission Gear System
# of output queries	6516	86446.8
# of equivalence queries	36	66
$\lvert \Sigma_E^{final} \rvert$	36	68
# of states	5	16
Max. length of cex.	4	12
$\lvert E \rvert$	0	6
Boolean dimensions in Σ	1	0
Numeric dimensions in Σ	3	2

Much better than the worst case
(# eq. $\leq \lvert \Sigma_E^{final} \rvert$ + # of states)

Conclusions

- Learning algorithm for symbolic Mealy automata
  - Idea: Explicitly learn “essential” characters Σ_E
- Complexity analysis of our algorithm
  - Worst case query complexity
  - An example to demonstrate the tightness
- Implementation + Experiments
  - Typically much more efficient than theoretical bound

Future Directions

- Better handling of counterexamples based on [Rivest & Schapire, ’93]
- Learning symbolic Mealy automata based on active learning of predicates, e.g., [Argyros & D’Antoni, CAV’18]
- Testing black-box systems by combining with model checking, e.g., [Peled+, FORTE’99]

Appendix

Nerode's congruence

Nerode's congruence ($\equiv_L$): For

- language: $L \subseteq \Sigma^*$
- prefixes: $s, s' \in \Sigma^*$

$s \equiv_L s'$ iff. $\forall e \in \Sigma^* . s \cdot e \in L \iff s' \cdot e \in L$

e.g. for $L = (\mathtt{ab})^*$, $\varepsilon \equiv_L \mathtt{ab}$, $\mathtt{a} \equiv_L \mathtt{aba}$, ...

Accepted iff $s \in (\mathtt{ab})^*$

Accepted iff $s \in \mathtt{b}(\mathtt{ab})^*$

✔: Easy to construct $\mathscr{A}$ s.t. $\mathscr{L}(\mathscr{A}) = L$ from $\equiv_L$

- **Idea**: Use $\Sigma^* / \equiv_L$ as the state space

✘: Requires ***infinite*** comparison to check if $s \equiv_L s'$

Other Requirements for Making Mealy Machines

Definition (closedness)

An observation table for symbolic Mealy automata is *closed* if for any $r \in R$ there is $s \in S$ with the same row

Definition (consistency)

An observation table for symbolic Mealy automata is *consistent* if for any $s, s' \in S \cup R$ and $s \cdot a, s' \cdot a \in S \cup R$, $row(s) = row(s')$ implies $row(s \cdot a) = row(s' \cdot a)$

Other Requirements for Making Mealy Machines

Definition (evidence-closedness)

An observation table for symbolic Mealy automata is *evidence-closed* if for any $s \in S$ and $e \in \Sigma_E$,
$s \cdot e \in S \cup R$ holds