Learning Automata with Name Allocation

1^{st} July 2025

Florian Frank, Stefan Milius, Jurriaan Rot
and Henning Urbat

Research Seminar

Chair for Computer Science 8 (Theoretical Computer Science)
Friedrich-Alexander-Universität Erlangen-Nürnberg

Friedrich-Alexander-Universität
Faculty of Engineering

Motivation (Register Automata & Data Languages)

- Data languages are formal languages over an infinite alphabet.

$\mathbb{A}$: admissible user IDs for a server ($\rightsquigarrow$ *infinite set*)

- Data languages are formal languages over an infinite alphabet.

$\mathbb{A}$: admissible user IDs for a server ($\rightsquigarrow$ *infinite set*)

$$\mathcal{L} = \left\{ a_1 \cdots a_n \in \mathbb{A}^\star \ : \ \left(\begin{array}{c} a_1 = a_n \wedge \\ \forall 1 < i < n.\, a_1 \neq a_i \end{array} \right) \right\}$$

'first and last user coincide and differ from any other user'

- Data languages are formal languages over an infinite alphabet.

  Standard model: Register Automata

$\mathbb{A}$: admissible user IDs for a server ($\rightsquigarrow$ *infinite set*)

$$\mathcal{L} = \left\{ a_1 \cdots a_n \in \mathbb{A}^\star \;:\; \left(\begin{array}{c} a_1 = a_n \wedge \\ \forall 1 < i < n.\, a_1 \neq a_i \end{array} \right) \right\}$$

'first and last user coincide and differ from any other user'

- Data languages are formal languages over an infinite alphabet.

  Standard model: Register Automata $\rightsquigarrow$ unfeasible for model checking (undecidable inclusion)

$\mathbb{A}$: admissible user IDs for a server ($\rightsquigarrow$ *infinite set*)

$$\mathcal{L} = \left\{ a_1 \cdots a_n \in \mathbb{A}^\star \;:\; \left(\begin{array}{c} a_1 = a_n \wedge \\ \forall 1 < i < n.\, a_1 \neq a_i \end{array} \right) \right\}$$

'first and last user coincide and differ from any other user'

- Data languages are formal languages over an infinite alphabet.

  Standard model: Register Automata $\rightsquigarrow$ unfeasible for model checking (undecidable inclusion)

- To gain decidability, we must accept restrictions in their expressivity.

$\mathbb{A}$: admissible user IDs for a server ($\rightsquigarrow$ *infinite set*)

$$\mathcal{L} = \left\{ a_1 \cdots a_n \in \mathbb{A}^\star \;:\; \left(\begin{array}{c} a_1 = a_n \wedge \\ \forall 1 < i < n.\, a_1 \neq a_i \end{array} \right) \right\}$$

'first and last user coincide and differ from any other user'

Motivation (Register Automata & Data Languages)

- Data languages are formal languages over an infinite alphabet.

  Standard model: Register Automata $\rightsquigarrow$ unfeasible for model checking (undecidable inclusion)

- To gain decidability, we must accept restrictions in their expressivity.

$\mathbb{A}$: admissible user IDs for a server ($\rightsquigarrow$ *infinite set*)

$$\mathcal{L} = \left\{ a_1 \cdots a_n \in \mathbb{A}^\star : \left(\begin{array}{c} a_1 = a_n \wedge \\ \forall 1 < i < n.\, a_1 \neq a_i \end{array} \right) \right\}$$

'first and last user coincide and differ from any other user'

Result — Schröder, Kozen, Milius, Wißmann '17

(Specific) **languages expressible by binding signatures** and their automata have decidable inclusion problems.

What are 'words with binders'?

Motivation (Register Automata & Data Languages)

- Data languages are formal languages over an infinite alphabet.

  Standard model: Register Automata $\rightsquigarrow$ unfeasible for model checking (undecidable inclusion)

- To gain decidability, we must accept restrictions in their expressivity.

$\mathbb{A}$: admissible user IDs for a server ($\rightsquigarrow$ *infinite set*)

$$\mathcal{L} = \left\{ a_1 \cdots a_n \in \mathbb{A}^\star \ : \ \left(\begin{array}{c} a_1 = a_n \wedge \\ \forall 1 < i < n.\, a_1 \neq a_i \end{array} \right) \right\}$$

'first and last user coincide and differ from any other user'

$\lambda a.(\lambda b.)^\star a$
(using shadowing)

Result — Schröder, Kozen, Milius, Wißmann '17

(Specific) **languages expressible by binding signatures** and their automata have decidable inclusion problems.

What are 'words with binders'?

Motivation

- We consider data languages with explicit binders, which we see with λ-terms without parenthesis:

$$\boxed{\lambda x.\ f\ x\ \lambda y.\ x\ y\ \lambda z.\ y\ z}$$

scope of binders is unlimited here

Motivation

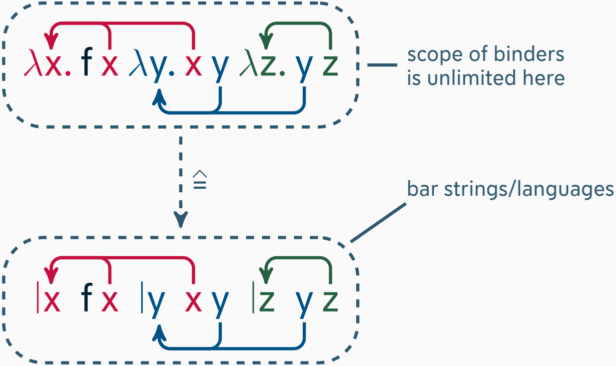
- We consider data languages with explicit binders, which we see with λ-terms without parenthesis:

$$\lambda x.\ f\ x\ \lambda y.\ x\ y\ \lambda z.\ y\ z$$

scope of binders is unlimited here

Motivation

- We consider data languages with explicit binders, which we see with λ-terms without parenthesis:



Motivation

- We consider data languages with explicit binders, which we see with λ-terms without parenthesis:

$$\lambda x.\, f\, x\, \lambda y.\, x\, y\, \lambda z.\, y\, z$$

scope of binders is unlimited here

$\hat{=}$

$$|x\ f\ x\ |y\ x\ y\ |z\ y\ z$$

bar strings/languages

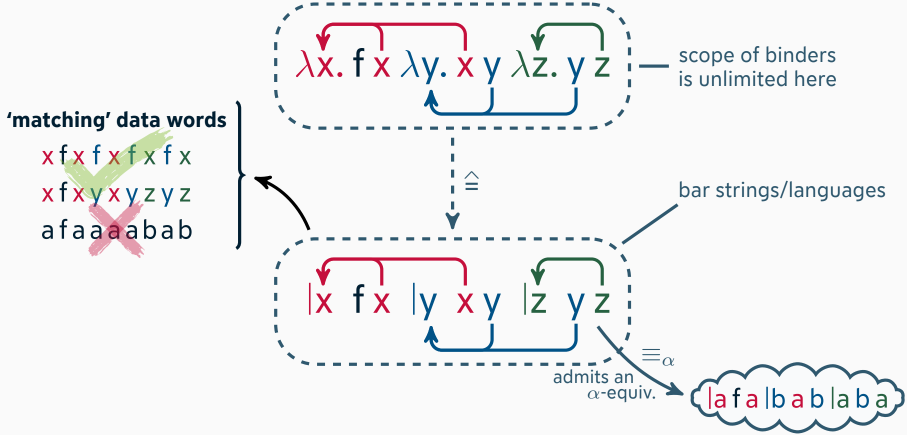
admits an α-equiv.

$\equiv_\alpha$

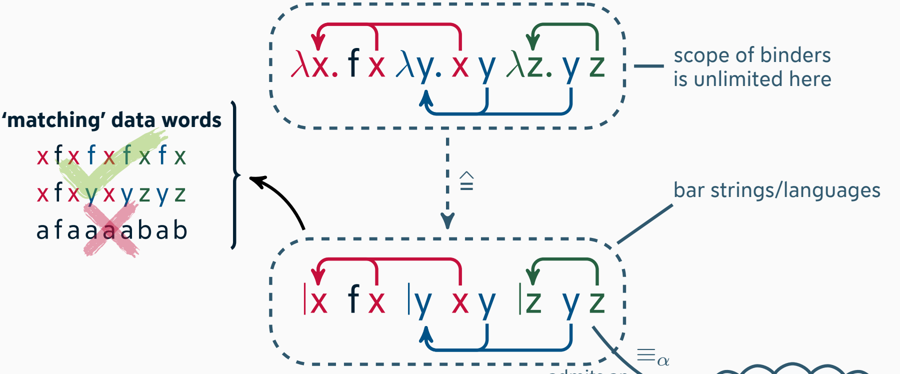
$$|a\ f\ a\ |b\ a\ b\ |a\ b\ a$$

Motivation

- We consider data languages with explicit binders, which we see with λ-terms without parenthesis:

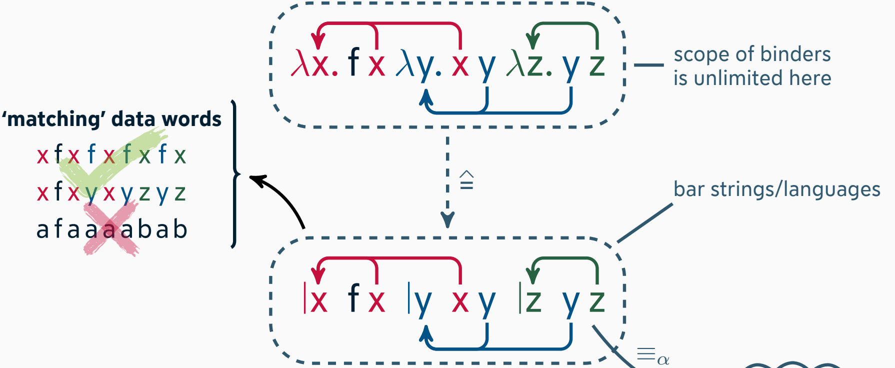


- We consider data languages with explicit binders, which we see with λ-terms without parenthesis:



- 'Match' data words by taking any representative without bars.

- We consider data languages with explicit binders, which we see with λ-terms without parenthesis:



- 'Match' data words by taking any representative without bars.
- **Our Problem:** How do we get the neccessary models for model checking from a black-box system?

Models for Bar Languages

» We consider classical automata over *finite* subalphabets $\overline{\mathbb{A}}_0 \subseteq_\mathsf{f} \overline{\mathbb{A}}$:

Definition (*Bar DFA*)

A bar DFA $\mathscr{A}$ is a DFA over a finite alphabet $\overline{\mathbb{A}}_0 \subseteq_\mathsf{f} \overline{\mathbb{A}}$.

Its *bar language* $L_\alpha(\mathscr{A}) = \left\{ w \in \overline{\mathbb{A}}^\star \,:\, w \equiv_\alpha w' \in L(\mathscr{A}) \right\}$ consists of all representatives of its α-equivalence classes.

automaton is *closed* iff $L(\mathscr{A}) = L_\alpha(\mathscr{A})$

Models for Bar Languages

- We consider classical automata over *finite* subalphabets $\overline{\mathbb{A}}_0 \subseteq_\mathsf{f} \overline{\mathbb{A}}$:

Definition (*Bar DFA*)

A bar DFA $\mathscr{A}$ is a DFA over a finite alphabet $\overline{\mathbb{A}}_0 \subseteq_\mathsf{f} \overline{\mathbb{A}}$.

Its *bar language* $L_\alpha(\mathscr{A}) = \left\{ w \in \overline{\mathbb{A}}^\star \;:\; w \equiv_\alpha w' \in L(\mathscr{A}) \right\}$ consists of all representatives of its α-equivalence classes.

automaton is *closed* iff $L(\mathscr{A}) = L_\alpha(\mathscr{A})$

- Correspond precisely to Schröder et al.'s nominal automata.

(Schröder, Kozen, Milius, Wißmann '17)

Models for Bar Languages

- We consider classical automata over *finite* subalphabets $\overline{\mathbb{A}}_0 \subseteq_\mathsf{f} \overline{\mathbb{A}}$:

Definition (*Bar DFA*)

A bar DFA $\mathscr{A}$ is a DFA over a finite alphabet $\overline{\mathbb{A}}_0 \subseteq_\mathsf{f} \overline{\mathbb{A}}$.

Its *bar language* $L_\alpha(\mathscr{A}) = \left\{ w \in \overline{\mathbb{A}}^\star \;:\; w \equiv_\alpha w' \in L(\mathscr{A}) \right\}$ consists of all representatives of its α-equivalence classes.

automaton is *closed* iff $L(\mathscr{A}) = L_\alpha(\mathscr{A})$

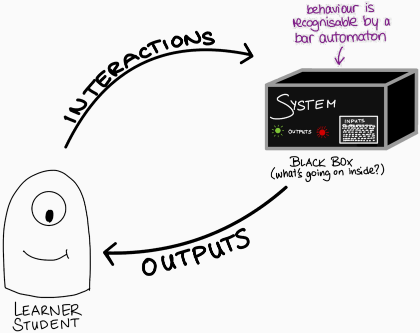
- Correspond precisely to Schröder et al.'s nominal automata.

(Schröder, Kozen, Milius, Wißmann '17)

- Expressivity (data languages):
  subclass of register automata

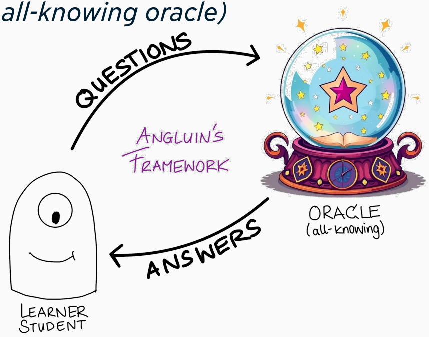
Assumption: Black-box system recognises a bar language L_T

» **Task:** Infer an automaton behaving ‘identically’ to the black-box system.



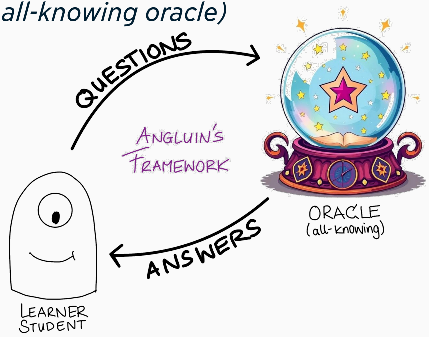
Assumption: Black-box system recognises a bar language L_T

- **Task:** Infer an automaton behaving 'identically' to the black-box system.
- Take Angluin's framework: (Angluin '87)
  (replace the black-box system by an all-knowing oracle)



Assumption: Black-box system recognises a bar language L_T

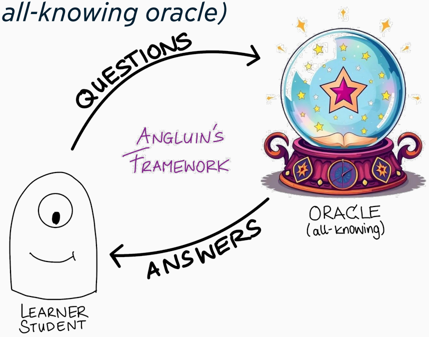
- **Task:** Infer an automaton behaving 'identically' to the black-box system.
- Take Angluin's framework: (Angluin '87)
  (replace the black-box system by an all-knowing oracle)
- Two kinds of queries:



Assumption: Black-box system recognises a bar language L_T

- **Task:** Infer an automaton behaving 'identically' to the black-box system.
- Take Angluin's framework: (Angluin '87)
  (replace the black-box system by an all-knowing oracle)
- Two kinds of queries:

MQ Given a bar string $w \in \overline{\mathbb{A}}^{\star}$, is $w \in L_T$?



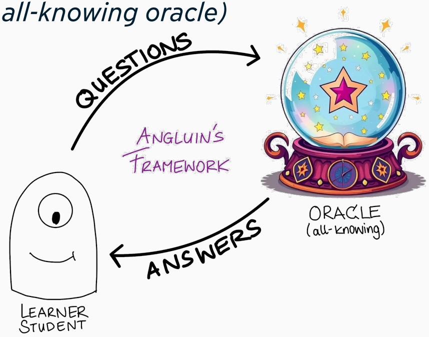
Active Automata Learning

Assumption: Black-box system recognises a bar language L_T

- **Task:** Infer an automaton behaving 'identically' to the black-box system.
- Take Angluin's framework: (Angluin '87)
  (replace the black-box system by an all-knowing oracle)
- Two kinds of queries:

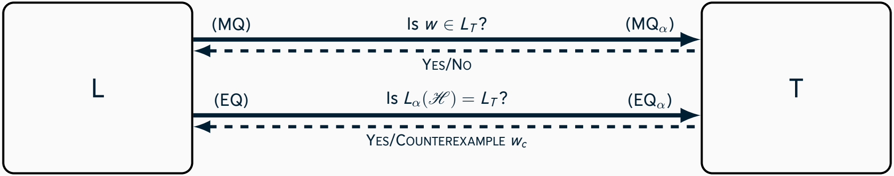
MQ Given a bar string $w \in \overline{\mathbb{A}}^\star$, is $w \in L_T$?

EQ Given a bar automaton $\mathscr{H}$, is $L_\alpha(\mathscr{H}) = L_T$?

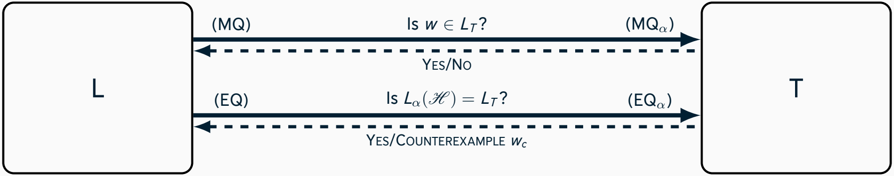


Learning Bar Automata

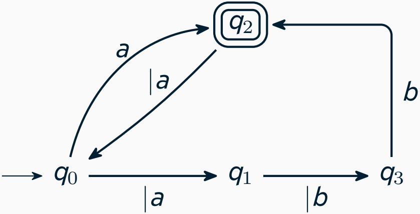
Question: Can't we just use classical learning algorithms for bar automata (as they are just DFA with additional semantics)?



Question: Can't we just use classical learning algorithms for bar automata (as they are just DFA with additional semantics)?

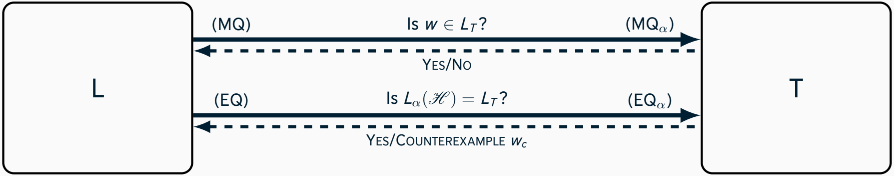


» **Problem:**

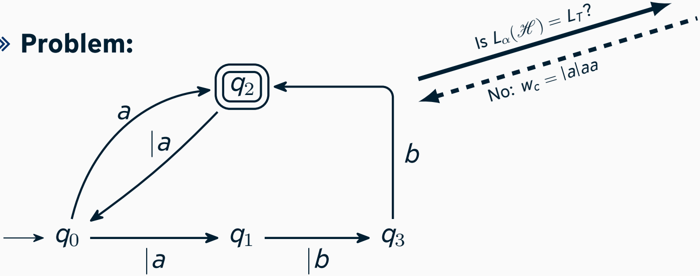


Learning Bar Automata

Question: Can't we just use classical learning algorithms for bar automata (as they are just DFA with additional semantics)?

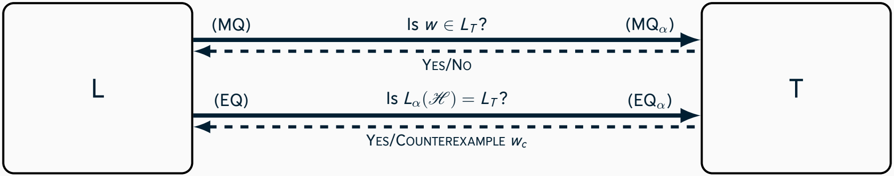


» **Problem:**

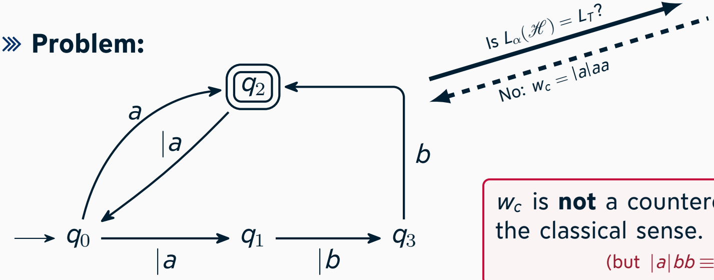


Learning Bar Automata

Question: Can't we just use classical learning algorithms for bar automata (as they are just DFA with additional semantics)?



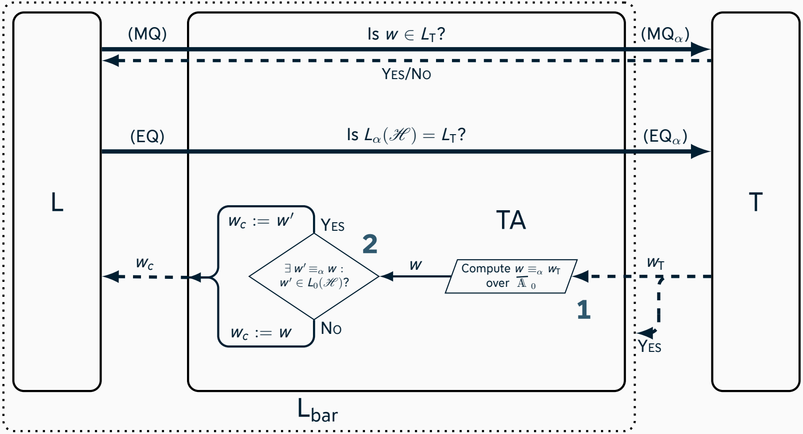
⋙ Problem:



w_c is **not** a counterexample in the classical sense.

(but $|a|bb \equiv_\alpha w_c$ would be)

» Indeed, this is the only problem with the previous approach:



Given some bar word $w_{\mathsf{T}} \in \overline{\mathbb{A}}^{\star}$, compute $w \equiv_{\alpha} w_{\mathsf{T}}$ over $\overline{\mathbb{A}}_0^{\star}$

Learning Bar Automata (Problem 1: Computing α-equivalent words)

Given some bar word $w_\top \in \overline{\mathbb{A}}^\star$, compute $w \equiv_\alpha w_\top$ over $\overline{\mathbb{A}}_0^\star$

- With λ-terms, checking if two terms are α-equivalent essentially boils down to computing a De Bruijn representation.

Given some bar word $w_T \in \overline{\mathbb{A}}^\star$, compute $w \equiv_\alpha w_T$ over $\overline{\mathbb{A}}_0^\star$

- With λ-terms, checking if two terms are α-equivalent essentially boils down to computing a De Bruijn representation.

  ($\rightsquigarrow$ apply De Bruijn representations to bar strings)

! Given some bar word $w_\mathsf{T} \in \overline{\mathbb{A}}^\star$, compute $w \equiv_\alpha w_\mathsf{T}$ over $\overline{\mathbb{A}}_0^\star$

- With λ-terms, checking if two terms are α-equivalent essentially boils down to computing a De Bruijn representation.

  ($\rightsquigarrow$ apply De Bruijn representations to bar strings)

$$1\ b\ 1\ 2\ 1\ 2\ 3\ 2\ 3 \xleftarrow[\text{normal form}]{\text{De Bruijn}} |a\ b\ a\ |y\ a\ y\ |z\ y\ z$$

Learning Bar Automata (Problem 1: Computing α-equivalent words)

Given some bar word $w_{\mathsf{T}} \in \overline{\mathbb{A}}^{\star}$, compute $w \equiv_{\alpha} w_{\mathsf{T}}$ over $\overline{\mathbb{A}}_0^{\star}$

- With λ-terms, checking if two terms are α-equivalent essentially boils down to computing a De Bruijn representation.

  ($\rightsquigarrow$ apply De Bruijn representations to bar strings)

$$1\ b\ 1\ 2\ 1\ 2\ 3\ 2\ 3 \xleftarrow[\text{normal form}]{\text{De Bruijn}} |a\ b\ a\ |y\ a\ y\ |z\ y\ z$$

- These normal forms are *unique* (per equivalence class) and computable in *polynomial time* (with linear-logarithmic space).

Learning Bar Automata (Problem 2: Finding accepted words)

Given some $w \in \overline{\mathbb{A}}_0$, is there a $w' \equiv_\alpha w$ with $w' \in L(\mathscr{H})$?

Given some $w \in \overline{\mathbb{A}}_0$, is there a $w' \equiv_\alpha w$ with $w' \in L(\mathscr{H})$?

» Easy non-deterministic algorithm (in NP):

Given some $w \in \overline{\mathbb{A}}_0$, is there a $w' \equiv_\alpha w$ with $w' \in L(\mathscr{H})$?

- Easy non-deterministic algorithm (in NP):

Algorithm

1) Guess a bar string of equal length;
2) Check for α-equivalence.

Learning Bar Automata (Problem 2: Finding accepted words)

Given some $w \in \overline{\mathbb{A}}_0$, is there a $w' \equiv_\alpha w$ with $w' \in L(\mathscr{H})$?

- Easy non-deterministic algorithm (in NP):

Algorithm

1) Guess a bar string of equal length;
2) Check for α-equivalence.

- Is a deterministic poly-time algorithm possible?

Given some $w \in \overline{\mathbb{A}}_0$, is there a $w' \equiv_\alpha w$ with $w' \in L(\mathscr{H})$?

- Easy non-deterministic algorithm (in NP):

Algorithm

1) Guess a bar string of equal length;
2) Check for α-equivalence.

- Is a deterministic poly-time algorithm possible? not exactly ...

Learning Bar Automata (Problem 2: Finding accepted words)

Given some $w \in \overline{\mathbb{A}}_0$, is there a $w' \equiv_\alpha w$ with $w' \in L(\mathscr{H})$?

- Easy non-deterministic algorithm (in NP):

Algorithm

1) Guess a bar string of equal length;
2) Check for α-equivalence.

- Is a deterministic poly-time algorithm possible? not exactly ...
  - In general: NP-completeness (via the Hamilton cycle problem) ...

Learning Bar Automata (Problem 2: Finding accepted words)

Given some $w \in \overline{\mathbb{A}}_0$, is there a $w' \equiv_\alpha w$ with $w' \in L(\mathscr{H})$?

- Easy non-deterministic algorithm (in NP):

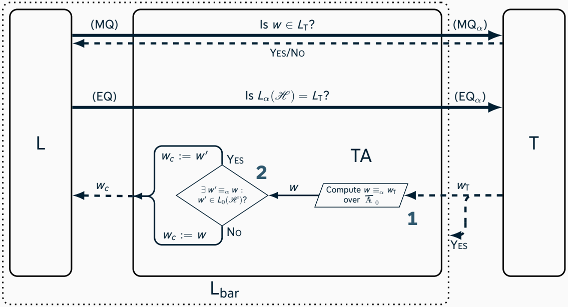
Algorithm

1) Guess a bar string of equal length;
2) Check for α-equivalence.

- Is a deterministic poly-time algorithm possible? not exactly ...
  - In general: NP-completeness (via the Hamilton cycle problem) ...
  - ... but for fixed alphabets in deterministic poly-time.

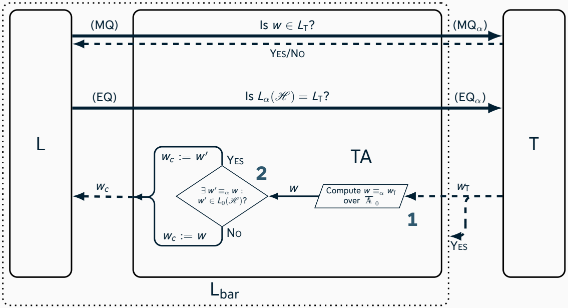
by comparing with the *closed* bar automaton.

» By solving both problems, correctness of our approach is shown.



Learning Bar Automata (Our Approach)

⋙ By solving both problems, correctness of our approach is shown.

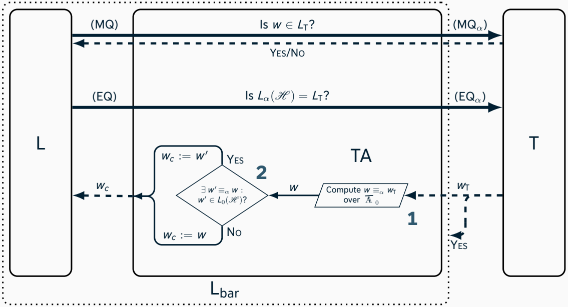


Query Complexity

$\mathrm{L}_{\mathrm{bar}}$ asks at most as many queries as L would.

Learning Bar Automata (Our Approach)

- By solving both problems, correctness of our approach is shown.



Query Complexity

$\mathrm{L_{bar}}$ asks at most as many queries as L would.

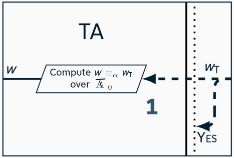
- Implicit Assumption: $\mathrm{L_{bar}}$ knows the *number of registers* (size of alphabet) needed for L_T.

How to handle an unknown Alphabet?

Idea

If L_{bar} gets stuck, just restart anew (with an extended alphabet).

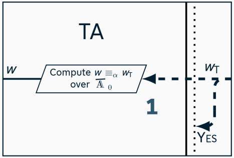
1) Start with the empty alphabet $\overline{\mathbb{A}}_0 = \emptyset$.



Idea

If L_{bar} gets stuck, just restart anew (with an extended alphabet).

1) Start with the empty alphabet $\overline{\mathbb{A}}_0 = \emptyset$.
2) **Repeat:**

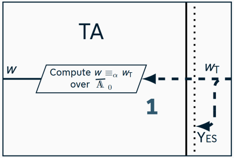


How to handle an unknown Alphabet?

Idea

If L_{bar} gets stuck, just restart anew (with an extended alphabet).

1) Start with the empty alphabet $\overline{\mathbb{A}}_0 = \emptyset$.
2) **Repeat**:
   a) If a **correct** hypothesis is found, **terminate**.

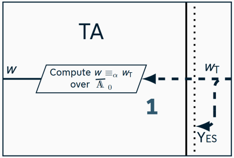


How to handle an unknown Alphabet?

Idea

If L_{bar} gets stuck, just restart anew (with an extended alphabet).

1) Start with the empty alphabet $\overline{\mathbb{A}}_0 = \emptyset$.
2) **Repeat**:
   a) If a **correct** hypothesis is found, **terminate**.
   b) If T delivers a counterexample w_T and step **1** **works**, **continue**.

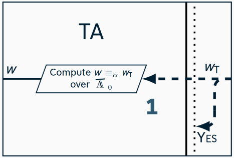


How to handle an unknown Alphabet?

Idea

If L_{bar} gets stuck, just restart anew (with an extended alphabet).

1) Start with the empty alphabet $\overline{\mathbb{A}}_0 = \emptyset$.
2) **Repeat**:
   a) If a **correct** hypothesis is found, **terminate**.
   b) If T delivers a counterexample w_T and step **1** **works**, **continue**.
   c) **Else:** Extend $\overline{\mathbb{A}}_0$ to $\overline{\mathbb{A}}_0 \subseteq \overline{\mathbb{A}}_0'$ and try again.

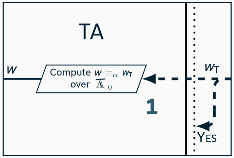


How to handle an unknown Alphabet?

Idea

If $\mathrm{L_{bar}}$ gets stuck, just restart anew (with an extended alphabet).

1) Start with the empty alphabet $\overline{\mathbb{A}}_0 = \emptyset$.
2) **Repeat**:
   a) If a **correct** hypothesis is found, **terminate**.
   b) If T delivers a counterexample w_T and step **1** **works**, **continue**.
   c) **Else:** Extend $\overline{\mathbb{A}}_0$ to $\overline{\mathbb{A}}_0 \subseteq \overline{\mathbb{A}}_0'$ and try again.



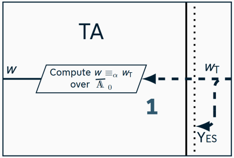
Choose extension via De Bruijn normal form.

How to handle an unknown Alphabet?

Idea

If L$_{\text{bar}}$ gets stuck, just restart anew (with an extended alphabet).

1) Start with the empty alphabet $\overline{\mathbb{A}}_0 = \emptyset$.
2) **Repeat**:
   a) If a **correct** hypothesis is found, **terminate**.
   b) If T delivers a counterexample w_T and step **1** **works**, **continue**.
   c) **Else:** Extend $\overline{\mathbb{A}}_0$ to $\overline{\mathbb{A}}_0 \subseteq \overline{\mathbb{A}}_0'$ and try again.



Choose extension via De Bruijn normal form.

Theorem (*Extension Complexity*)

L$_{\text{bar}}$ can infer a bar automaton using the minimal alphabet with at most as many queries as L would need (summed over all smaller cardinalities).

What about ω- and tree languages?

- Next to finite word languages, we extended our approach to bar ω- and bar tree languages:

What about ω- and tree languages?

- Next to finite word languages, we extended our approach to bar ω- and bar tree languages:

Alphatic Trees

As expected, everything works out analogously:

- Normal forms are computed branchwise (and thus still in *poly-time*).

What about ω- and tree languages?

- Next to finite word languages, we extended our approach to bar ω- and bar tree languages:

Alphatic Trees

As expected, everything works out analogously:

- Normal forms are computed branchwise (and thus still in *poly-time*).
- Finding accepted bar trees is NP-complete, but also in para. poly-time.

What about ω- and tree languages?

- Next to finite word languages, we extended our approach to bar ω- and bar tree languages:

Alphatic Trees

As expected, everything works out analogously:

- Normal forms are computed branchwise (and thus still in *poly-time*).
- Finding accepted bar trees is NP-complete, but also in para. poly-time.
- Handling unknown alphabets works completely identical.

What about ω- and tree languages?

- Next to finite word languages, we extended our approach to bar ω- and bar tree languages:

Alphatic Trees

As expected, everything works out analogously:

- Normal forms are computed branchwise (and thus still in *poly-time*).
- Finding accepted bar trees is NP-complete, but also in para. poly-time.
- Handling unknown alphabets works completely identical.

Challenges in the infinite word case

What about ω- and tree languages?

- Next to finite word languages, we extended our approach to bar ω- and bar tree languages:

Alphatic Trees

As expected, everything works out analogously:

- Normal forms are computed branchwise (and thus still in *poly-time*).
- Finding accepted bar trees is NP-complete, but also in para. poly-time.
- Handling unknown alphabets works completely identical.

Challenges in the infinite word case

- There is no suitable normal form for infinite strings.

(example at blackboard)

What about ω- and tree languages?

- Next to finite word languages, we extended our approach to bar ω- and bar tree languages:

Alphatic Trees

As expected, everything works out analogously:

- Normal forms are computed branchwise (and thus still in *poly-time*).
- Finding accepted bar trees is NP-complete, but also in para. poly-time.
- Handling unknown alphabets works completely identical.

Challenges in the infinite word case

- There is no suitable normal form for infinite strings.
  (example at blackboard)
- Checking α-equivalence is still in poly-time, but guessing seemingly impossible.

What about ω- and tree languages?

T.CS

- Next to finite word languages, we extended our approach to bar ω- and bar tree languages:

Alphatic Trees

As expected, everything works out analogously:

- Normal forms are computed branchwise (and thus still in *poly-time*).
- Finding accepted bar trees is NP-complete, but also in para. poly-time.
- Handling unknown alphabets works completely identical.

Challenges in the infinite word case

- There is no suitable normal form for infinite strings.
  (example at blackboard)
- Checking α-equivalence is still in poly-time, but guessing seemingly impossible.
- Expensive computation of closures is important!

Conclusion & Future Work

- Learnability of various kinds of bar languages in Angluin's framework.
- Our learner is in terms of query complexity as optimal as the underlying learner.
- Introduced efficient procedures for checking α-equivalence.

Future Work

- Is guessing possible for bar ω-languages? Can the computation of closures be removed?
- Can this approach be extended to efficiently learn data languages?
- What to do about conformance testing?

Questions?

Friedrich-Alexander-Universität
Faculty of Engineering

References

Angluin, Dana. **'Learning regular sets from queries and counterexamples'.** *Information and Computationc* 75.2 (Nov. 1987), pp. 87–106. ISSN: 0890-5401. DOI: `10.1016/0890-5401(87)90052-6`. URL: `https://doi.org/10.1016/0890-5401(87)90052-6`.

Schröder, Lutz, Dexter Kozen, Stefan Milius, Thorsten Wißmann. **'Nominal Automata with Name Binding'.** *Proc. 20th International Conference on Foundations of Software Science and Computation Structures, (FOSSACS 2017).* Vol. 10203. Lect. Notes Comput. Sci. 2017, pp. 124–142.