Friedrich-Alexander-Universität
Technische Fakultät

Learning Automata with Name Allocation

Florian Frank[1], Stefan Milius[1], Jurriaan Rot[2] and Henning Urbat[1]

11th April 2026

18th International Workshop on Coalgebraic Methods in Computer Science; Turin

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

[2] Department of Software Science
Radboud Universiteit Nijmegen

Data Languages

» **Data languages**: **Infinite alphabet** and **assertions of equality and inequality**

Data Languages

» **Data languages**: **Infinite alphabet** and **assertions of equality and inequality**

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

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

Data Languages

» **Data languages**: **Infinite alphabet** and **assertions of equality and inequality**

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

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

infinite set of data values

$\mathbb{D} = \{\text{all possible user IDs}\}$

Data Languages

» **Data languages**: **Infinite alphabet** and **assertions of equality and inequality**

» **Formal:** Closure under permutation of data values

$$\forall \pi : \mathbb{D} \xrightarrow{\simeq} \mathbb{D}, a_1 \cdots a_n \in \mathbb{D}^\star :$$
$$a_1 \cdots a_n \in L \iff \pi(a_1) \cdots \pi(a_n) \in L.$$

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

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

infinite set of data values

$$\mathbb{D} = \{\text{all possible user IDs}\}$$

Data Languages

» **Data languages**: **Infinite alphabet** and **assertions of equality and inequality**

» **Formal:** Closure under permutation of data values

$$\forall \pi : \mathbb{D} \xrightarrow{\simeq} \mathbb{D}, a_1 \cdots a_n \in \mathbb{D}^\star : \\ a_1 \cdots a_n \in L \iff \pi(a_1) \cdots \pi(a_n) \in L.$$

» Use **nominal sets** for implicit data value handling.

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

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

infinite set of data values

$$\mathbb{D} = \{\text{all possible user IDs}\}$$

Data Languages

» **Data languages**: **Infinite alphabet** and **assertions of equality and inequality**

» **Formal:** Closure under permutation of data values

$$\forall \pi : \mathbb{D} \xrightarrow{\simeq} \mathbb{D}, a_1 \cdots a_n \in \mathbb{D}^\star : \\ a_1 \cdots a_n \in L \iff \pi(a_1) \cdots \pi(a_n) \in L.$$

» Use **nominal sets** for implicit data value handling.

» **Idea:** set + elements depend on data values.

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

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

infinite set of data values

$\mathbb{D} = \{\text{all possible user IDs}\}$

Data Languages

» **Data languages**: **Infinite alphabet** and **assertions of equality and inequality**

» **Formal:** Closure under permutation of data values

$$\forall \pi : \mathbb{D} \xrightarrow{\simeq} \mathbb{D}, a_1 \cdots a_n \in \mathbb{D}^\star : \\ a_1 \cdots a_n \in L \iff \pi(a_1) \cdots \pi(a_n) \in L.$$

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

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

infinite set of data values

$\mathbb{D} = \{\text{all possible user IDs}\}$

» Use **nominal sets** for implicit data value handling.

- » **Idea:** set + elements depend on data values.
- » **Change** the data values of an element using permutations.
  $(a\ b) \cdot accab = bccba$ (switching data values a and b)

Data Languages

- **Data languages**: **Infinite alphabet** and **assertions of equality and inequality**
- **Formal:** Closure under permutation of data values

$$\forall \pi : \mathbb{D} \xrightarrow{\simeq} \mathbb{D}, a_1 \cdots a_n \in \mathbb{D}^\star : \\ a_1 \cdots a_n \in L \iff \pi(a_1) \cdots \pi(a_n) \in L.$$

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

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

infinite set of data values

$$\mathbb{D} = \{\text{all possible user IDs}\}$$

- Use **nominal sets** for implicit data value handling.
  - **Idea:** set + elements depend on data values.
  - **Change** the data values of an element using permutations.
    $(a\ b) \cdot accab = bccba$ (switching data values a and b)
- **Problem** with 'unrestricted' data languages: **Unfeasability** for practical use.

Name Allocation in Data Languages

Motivation and Example

- **Restriction:** Making equality and inequality **explicit** through binders
  (λ-Terms without parenthesis)
- **Previous Work:** Automata for these languages have nice properties.

Name Allocation in Data Languages

Motivation and Example

- **Restriction:** Making equality and inequality **explicit** through binders
  (λ-Terms without parenthesis)
- **Previous Work:** Automata for these languages have nice properties.

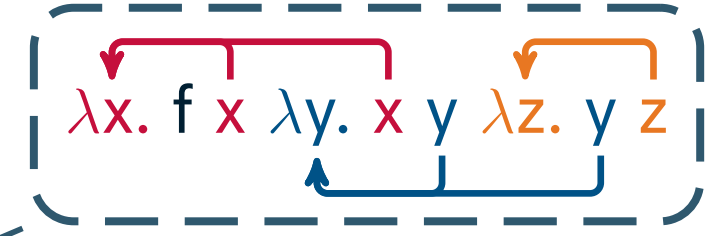
λx. f x λy. x y λz. y z

scope of binders
is unlimited here

Name Allocation in Data Languages

Motivation and Example

- **Restriction:** Making equality and inequality **explicit** through binders
  (λ-Terms without parenthesis)
- **Previous Work:** Automata for these languages have nice properties.

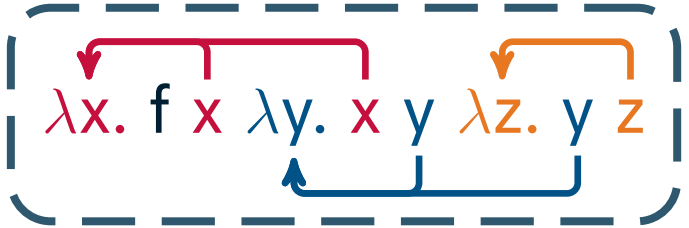


scope of binders
is unlimited here

Name Allocation in Data Languages

Motivation and Example

- **Restriction:** Making equality and inequality **explicit** through binders
  (λ-Terms without parenthesis)
- **Previous Work:** Automata for these languages have nice properties.
- Introduce **'variables'** to words/strings:



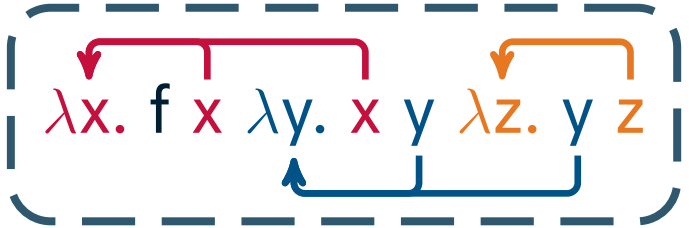
$$\overline{\mathbb{D}} := \mathbb{D} \cup \{ \mathsf{I}d \ : \ d \in \mathbb{D} \} \cong \mathbb{D} + \mathbb{D}$$

data values or 'variable names'

Name Allocation in Data Languages

Motivation and Example

- **Restriction:** Making equality and inequality **explicit** through binders
  (λ-Terms without parenthesis)
- **Previous Work:** Automata for these languages have nice properties.
- Introduce **'variables'** to words/strings:



$$\overline{\mathbb{D}} := \mathbb{D} \cup \{ \boxed{d} : \ d \in \mathbb{D} \} \cong \mathbb{D} + \mathbb{D}$$

introduces 'variable' d

data values or 'variable names'

Name Allocation in Data Languages

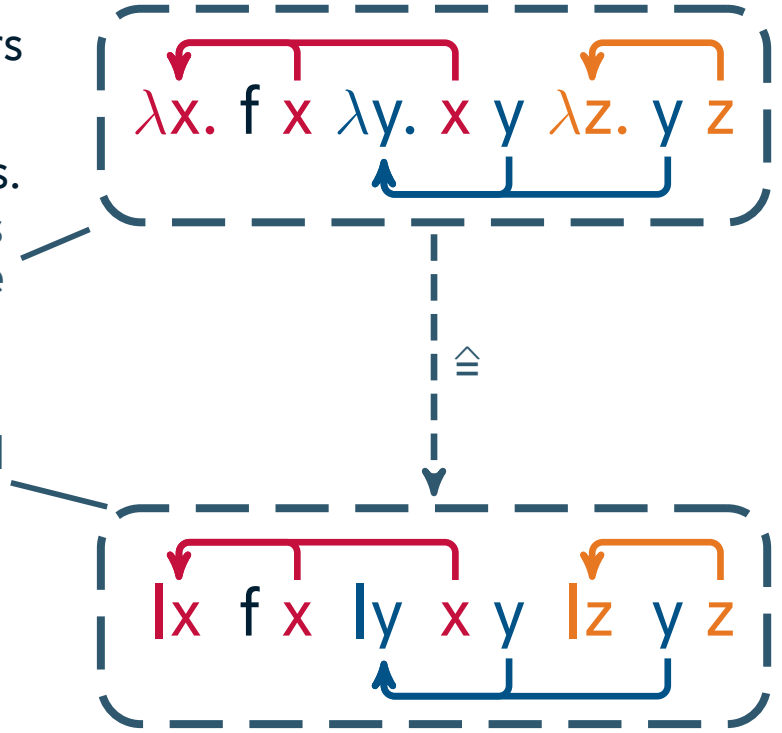
Motivation and Example

- **Restriction:** Making equality and inequality **explicit** through binders
  (λ-Terms without parenthesis)
- **Previous Work:** Automata for these languages have nice properties.
- Introduce **'variables'** to words/strings:

introduces 'variable' d

$$\overline{\mathbb{D}} := \mathbb{D} \cup \{ \boxed{|}d : \ d \in \mathbb{D} \} \cong \mathbb{D} + \mathbb{D}$$

data values or 'variable names'



Name Allocation in Data Languages

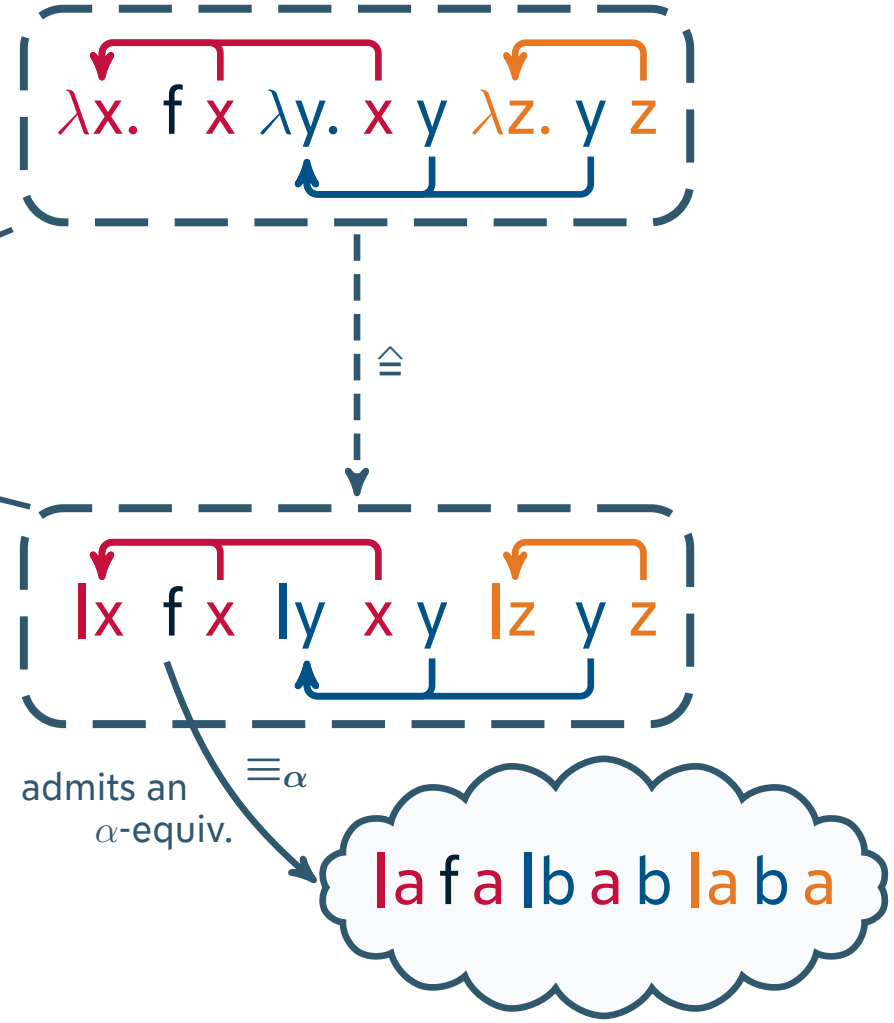
Motivation and Example

- **Restriction:** Making equality and inequality **explicit** through binders
  (λ-Terms without parenthesis)
- **Previous Work:** Automata for these languages have nice properties.
- Introduce **'variables'** to words/strings:

introduces 'variable' d

$$\overline{\mathbb{D}} := \mathbb{D} \cup \{ |d : d \in \mathbb{D} \} \cong \mathbb{D} + \mathbb{D}$$

data values or 'variable names'



Name Allocation in Data Languages

Motivation and Example

- **Restriction:** Making equality and inequality **explicit** through binders
  (λ-Terms without parenthesis)
- **Previous Work:** Automata for these languages have nice properties.
- Introduce **'variables'** to words/strings:

introduces 'variable' d

$$\overline{\mathbb{D}} := \mathbb{D} \cup \{ |d : d \in \mathbb{D} \} \cong \mathbb{D} + \mathbb{D}$$

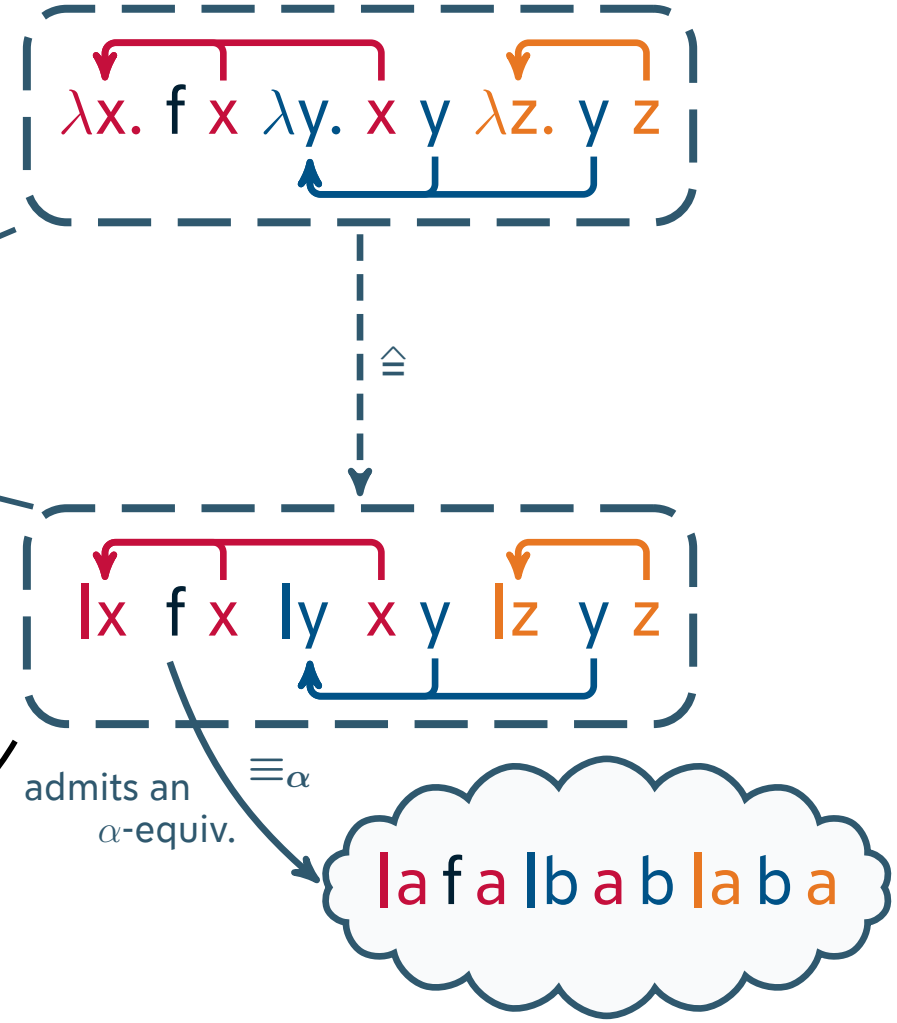
data values or 'variable names'

bar strings and bar languages

- **'Corresponding' data words:** Representatives without bars.

'matching' data words

- x f x f x f x f x
- x f x y x y z y z
- a f a a a a b a b



Name Allocation in Data Languages

Motivation and Example

- **Restriction:** Making equality and inequality **explicit** through binders
  (λ-Terms without parenthesis)
- **Previous Work:** Automata for these languages have nice properties.
- Introduce **'variables'** to words/strings:

introduces 'variable' d

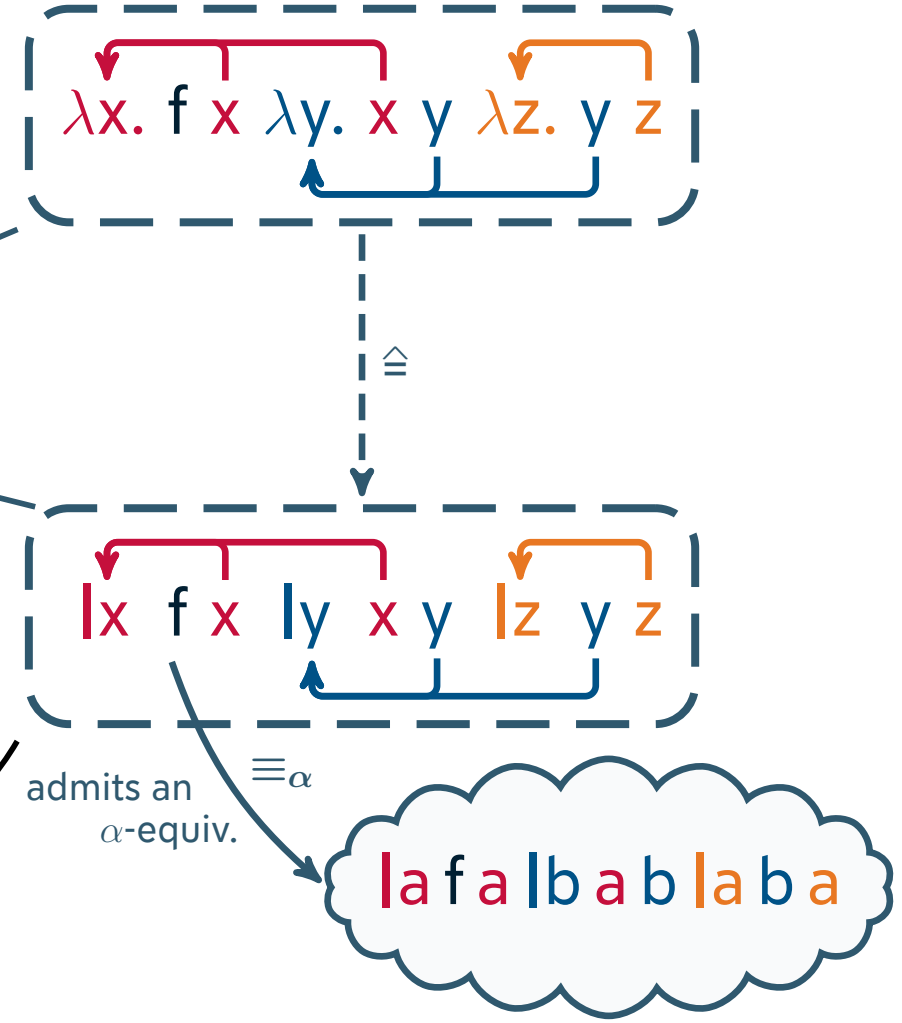
$$\overline{\mathbb{D}} := \mathbb{D} \cup \{ |d : d \in \mathbb{D} \} \cong \mathbb{D} + \mathbb{D}$$

data values or 'variable names'

- **'Corresponding' data words:** Representatives without bars.
- Three kinds of languages:
  - **Data Languages** (over $\mathbb{D}^\star$)
  - **Literal Languages** (over $\overline{\mathbb{D}}^\star$)
  - **Bar Languages** (closed under $\equiv_\alpha$)

'matching' data words

- x f x f x f x f x ✓
- x f x y x y z y z
- a f a a a a b a b ✗



Name Allocation in Data Languages

Motivation and Example

- **Restriction:** Making equality and inequality **explicit** through binders
  (λ-Terms without parenthesis)
- **Previous Work:** Automata for these languages have nice properties.
- Introduce **'variables'** to words/strings:

introduces 'variable' d

$$\overline{\mathbb{D}} := \mathbb{D} \cup \{ |d : d \in \mathbb{D} \} \cong \mathbb{D} + \mathbb{D}$$

data values or 'variable names'

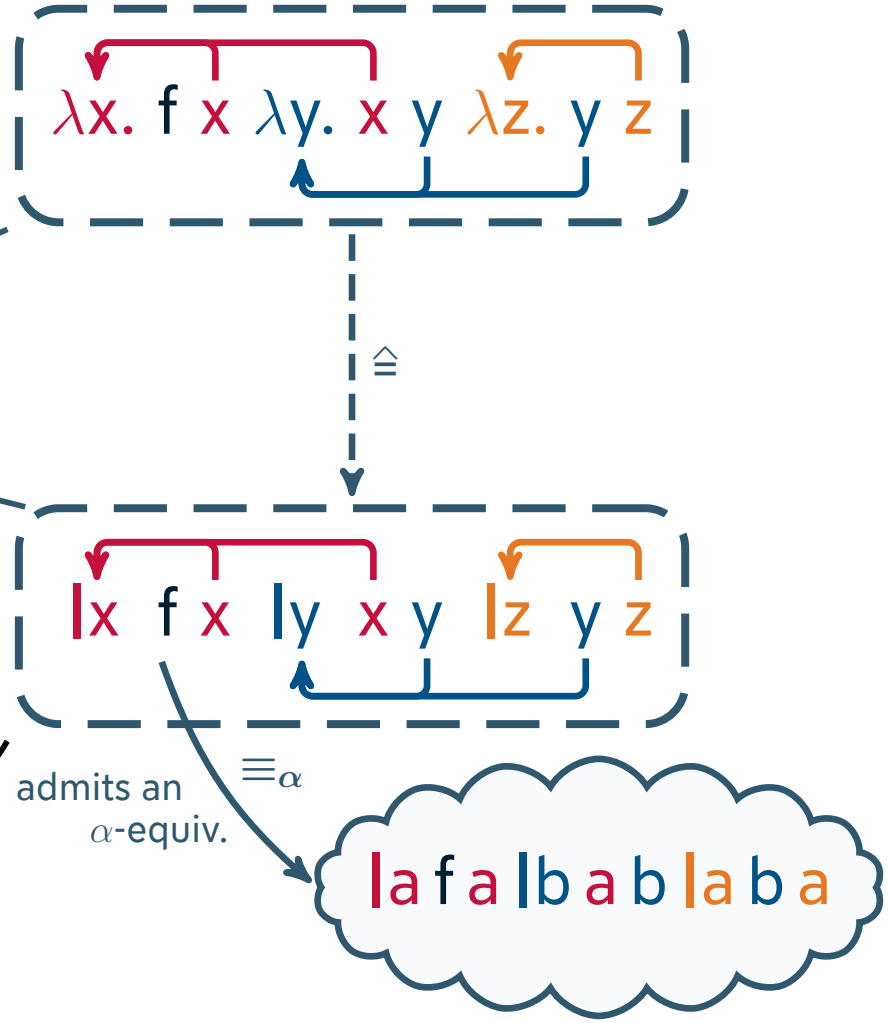
bar strings and bar languages

- **'Corresponding' data words:** Representatives without bars.
- Three kinds of languages:
  - **Data Languages** (over $\mathbb{D}^\star$)
  - **Literal Languages** (over $\overline{\mathbb{D}}^\star$)
  - **Bar Languages** (closed under $\equiv_\alpha$)

Conversion

'matching' data words

- x f x f x f x f x
- x f x y x y z y z
- a f a a a a b a b



Name Allocation in Data Languages

Extensions, Automata and Acceptance

- Urbat, Hausmann, Milius, Schröder '21 introduced **infinite bar strings**. ($\overline{\mathbb{D}}^{\omega}$)
  - **α-equivalence:** all finite prefixes are α-equivalent.
  - **Acceptance:** Büchi conditions
  - **Reduction** to ultimately periodic strings possible (using quotients)

Name Allocation in Data Languages

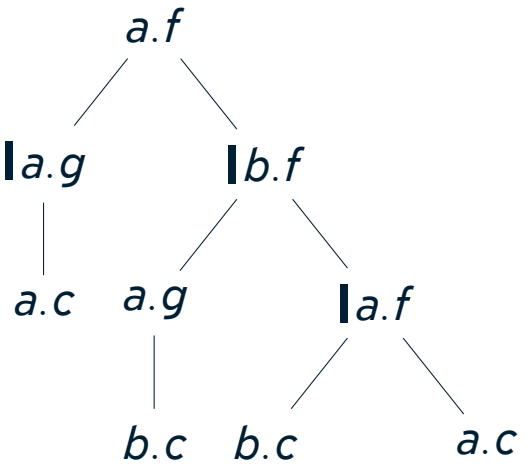
Extensions, Automata and Acceptance

- Urbat, Hausmann, Milius, Schröder '21 introduced **infinite bar strings**. ($\overline{\mathbb{D}}^{\omega}$)
  - **α-equivalence:** all finite prefixes are α-equivalent.
  - **Acceptance:** Büchi conditions
  - **Reduction** to ultimately periodic strings possible (using quotients)
- Prucker, Schröder '24 introduced **bar tree languages** over a signature. ($\mathcal{T}_{\overline{\mathbb{D}}}(\Sigma)$)
  - **α-equivalence:** pathwise α-equivalence.
  - **Acceptance:** Tree automata

Name Allocation in Data Languages

Extensions, Automata and Acceptance

- Urbat, Hausmann, Milius, Schröder '21 introduced **infinite bar strings**. ($\overline{\mathbb{D}}^{\omega}$)
  - **α-equivalence:** all finite prefixes are α-equivalent.
  - **Acceptance:** Büchi conditions
  - **Reduction** to ultimately periodic strings possible (using quotients)
- Prucker, Schröder '24 introduced **bar tree languages** over a signature. ($\mathcal{T}_{\overline{\mathbb{D}}}(\Sigma)$)
  - **α-equivalence:** pathwise α-equivalence.
  - **Acceptance:** Tree automata



Name Allocation in Data Languages

Extensions, Automata and Acceptance

- Urbat, Hausmann, Milius, Schröder '21 introduced **infinite bar strings**. ($\overline{\mathbb{D}}^{\omega}$)
  - **α-equivalence:** all finite prefixes are α-equivalent.
  - **Acceptance:** Büchi conditions
  - **Reduction** to ultimately periodic strings possible (using quotients)
- Prucker, Schröder '24 introduced **bar tree languages** over a signature. ($\mathcal{T}_{\overline{\mathbb{D}}}(\Sigma)$)
  - **α-equivalence:** pathwise α-equivalence.
  - **Acceptance:** Tree automata

RNTA

Acceptance via nominal Automata

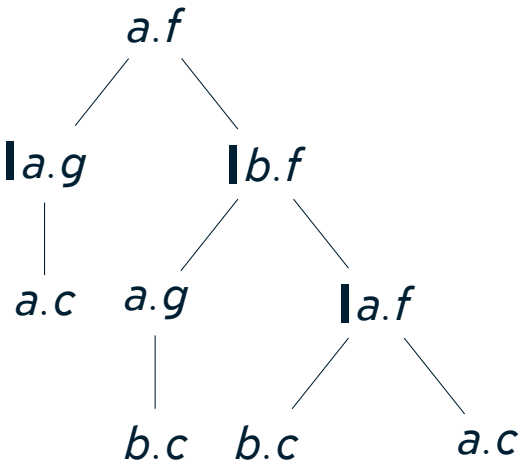
(Büchi) RNNA

DFA NFA NFTA

Acceptance via 'classical' Automata

Büchi Automata DFTA

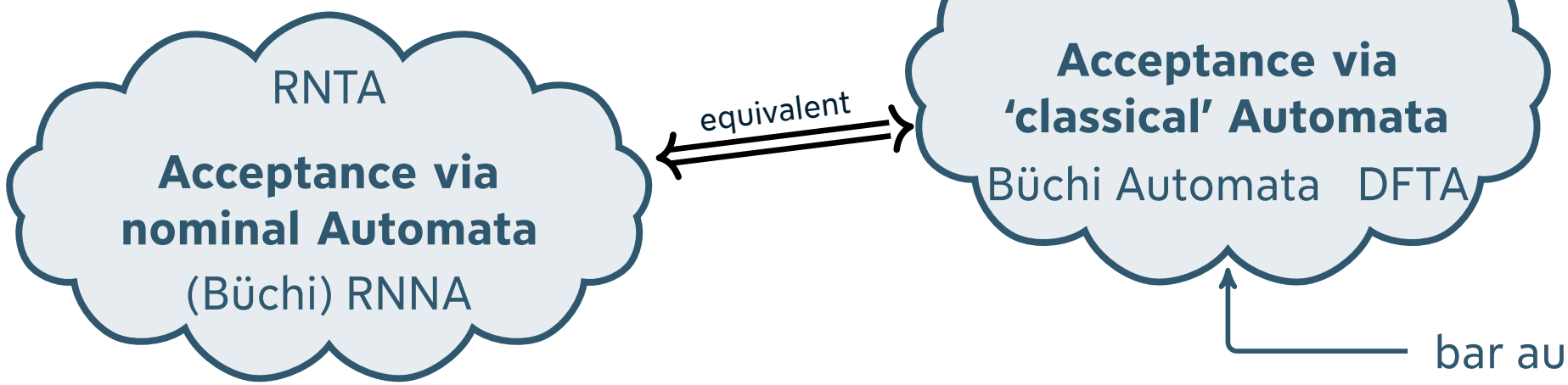
bar automata

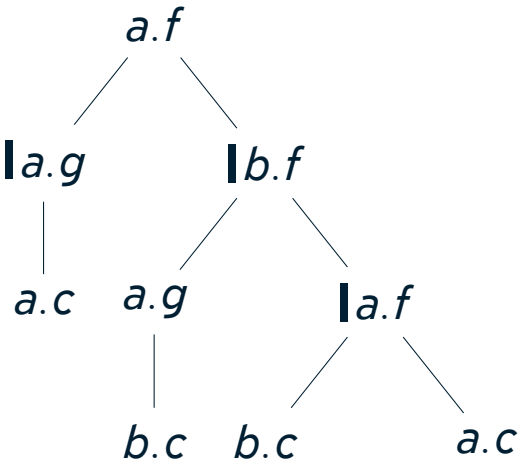


Name Allocation in Data Languages

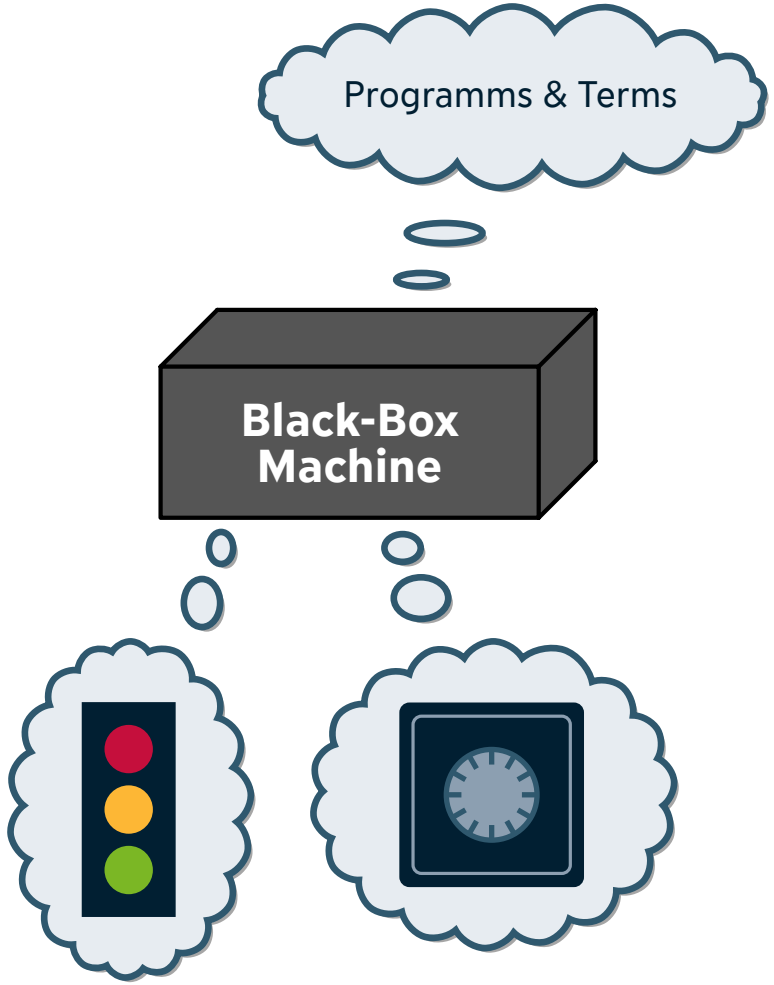
Extensions, Automata and Acceptance

- Urbat, Hausmann, Milius, Schröder '21 introduced **infinite bar strings**. ($\overline{\mathbb{D}}^{\omega}$)
  - **α-equivalence:** all finite prefixes are α-equivalent.
  - **Acceptance:** Büchi conditions
  - **Reduction** to ultimately periodic strings possible (using quotients)
- Prucker, Schröder '24 introduced **bar tree languages** over a signature. ($\mathcal{T}_{\overline{\mathbb{D}}}(\Sigma)$)
  - **α-equivalence:** pathwise α-equivalence.
  - **Acceptance:** Tree automata

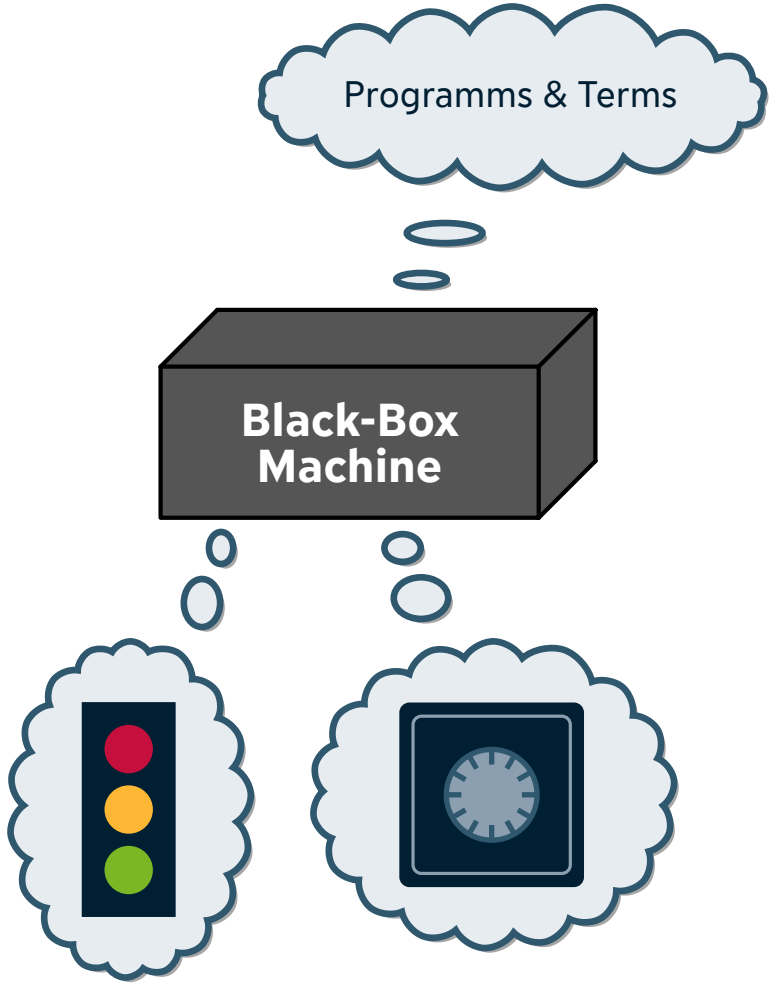


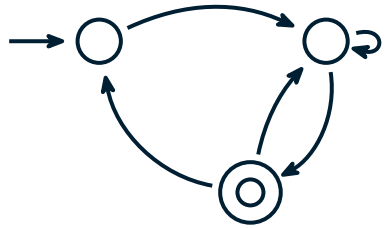


Why Automata Learning?



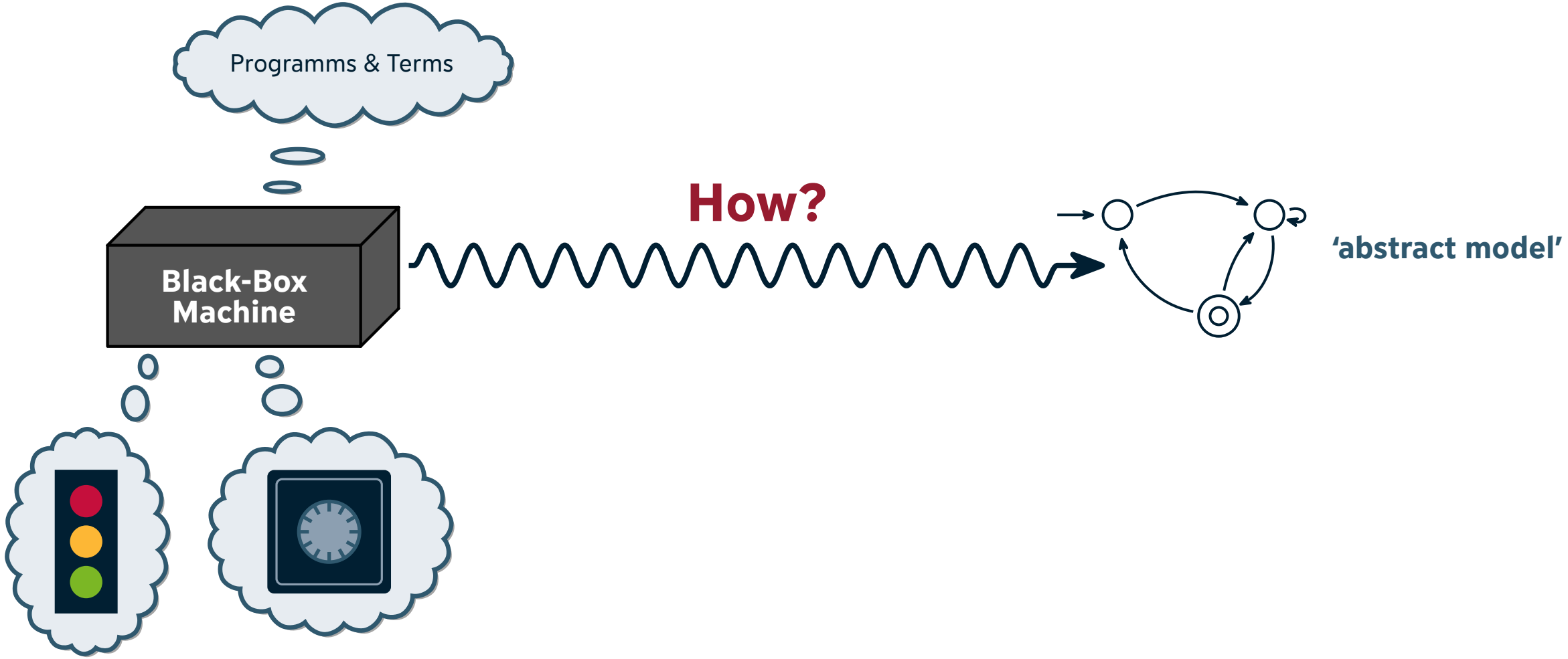
Why Automata Learning?





‘abstract model’

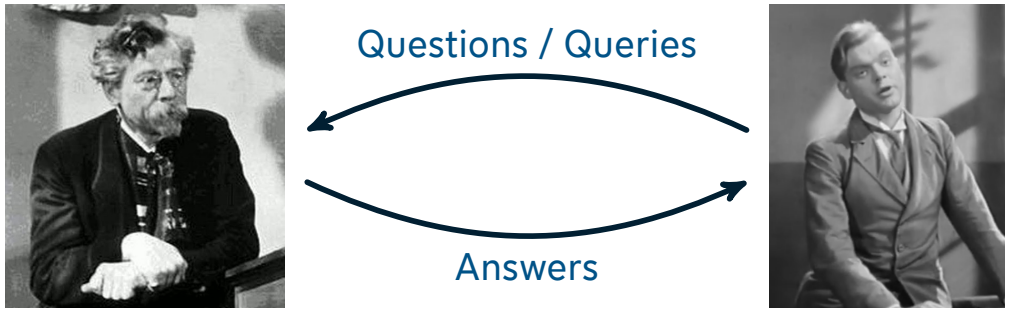
Why Automata Learning?



Learning Bar Automata

Model Generation in Angluin's Framework

» **Automata Learning:** Infer a model for L_T using only:



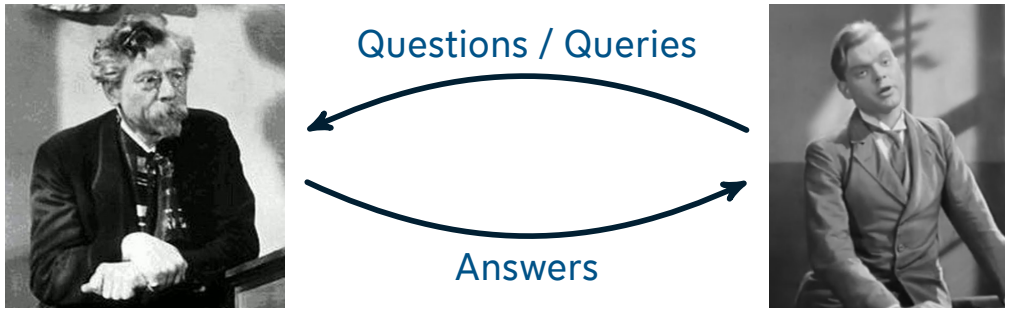
Teacher
(Die Feuerzangenbowle, 1944)

Student
(Die Feuerzangenbowle, 1944)

Learning Bar Automata

Model Generation in Angluin's Framework

- **Automata Learning:** Infer a model for L_T using only:
  - **Membership Queries (MQ):**
    Given a word $w \in \Sigma^*$, is $w \in L_T$?
    (**Answers:** Yes/No)



Teacher
(Die Feuerzangenbowle, 1944)

Student
(Die Feuerzangenbowle, 1944)

Learning Bar Automata

Model Generation in Angluin's Framework

- **Automata Learning:** Infer a model for L_T using only:
  - **Membership Queries (MQ):**
    Given a word $w \in \Sigma^*$, is $w \in L_T$?
    (**Answers:** Yes/No)
  - **Equivalence Queries (EQ):**
    Given a hypothesis $\mathcal{H}$, is $L(\mathcal{H}) = L_T$?
    (**Answers:** Yes/Counterexample)

Questions / Queries

Answers

Teacher
(Die Feuerzangenbowle, 1944)

Student
(Die Feuerzangenbowle, 1944)

Learning Bar Automata

Model Generation in Angluin's Framework

- **Automata Learning:** Infer a model for L_T using only:
  - **Membership Queries (MQ):**
    Given a word $w \in \Sigma^*$, is $w \in L_T$?
    (**Answers:** Yes/No)
  - **Equivalence Queries (EQ):**
    Given a hypothesis $\mathcal{H}$, is $L(\mathcal{H}) = L_T$?
    (**Answers:** Yes/Counterexample)

Teacher
(Die Feuerzangenbowle, 1944)

Questions / Queries

Answers

Student
(Die Feuerzangenbowle, 1944)

- Active learning algorithms available for DFAs, NFAs, Büchi automata, DFTAs, ...

Learning Bar Automata

Model Generation in Angluin's Framework

- **Automata Learning:** Infer a model for L_T using only:
  - **Membership Queries (MQ):**
    Given a word $w \in \Sigma^*$, is $w \in L_T$?
    (**Answers:** Yes/No)
  - **Equivalence Queries (EQ):**
    Given a hypothesis $\mathcal{H}$, is $L(\mathcal{H}) = L_T$?
    (**Answers:** Yes/Counterexample)

Teacher
(Die Feuerzangenbowle, 1944)

Questions / Queries

Answers

Student
(Die Feuerzangenbowle, 1944)

- Active learning algorithms available for DFAs, NFAs, Büchi automata, DFTAs, ...
- For **Data Languages**: active field of research
  - Bollig, Habermehl, Leucker, Monmege '14 learn *session automata*. ($\approx$ equivalent)
  - Sakamoto '97 learns deterministic register automata by modifying classical L^*. (incomparable)

Learning Bar Automata

Model Generation in Angluin's Framework

- **Automata Learning:** Infer a model for L_T using only:
  - **Membership Queries (MQ):**
    Given a word $w \in \Sigma^*$, is $w \in L_T$?
    (**Answers:** Yes/No)
  - **Equivalence Queries (EQ):**
    Given a hypothesis $\mathcal{H}$, is $L(\mathcal{H}) = L_T$?
    (**Answers:** Yes/Counterexample)

Teacher
(Die Feuerzangenbowle, 1944)

Questions / Queries

Answers

Student
(Die Feuerzangenbowle, 1944)

- Active learning algorithms available for DFAs, NFAs, Büchi automata, DFTAs, ...
- For **Data Languages**: active field of research
  - Bollig, Habermehl, Leucker, Monmege '14 learn *session automata*. ($\approx$ equivalent)
  - Sakamoto '97 learns deterministic register automata by modifying classical L^*. (incomparable)

Fact (*Finitisation*)

A **finite alphabet** is enough for bar languages.
(α**-equivalence**)

Learning Bar Automata

Model Generation in Angluin's Framework

- **Automata Learning:** Infer a model for L_T using only:
  - **Membership Queries (MQ):**
    Given a word $w \in \Sigma^*$, is $w \in L_T$?
    (**Answers:** Yes/No)
  - **Equivalence Queries (EQ):**
    Given a hypothesis $\mathcal{H}$, is $L(\mathcal{H}) = L_T$?
    (**Answers:** Yes/Counterexample)

Teacher
(Die Feuerzangenbowle, 1944)

Questions / Queries

Answers

Student
(Die Feuerzangenbowle, 1944)

- Active learning algorithms available for DFAs, NFAs, Büchi automata, DFTAs, ...
- For **Data Languages**: active field of research
  - Bollig, Habermehl, Leucker, Monmege '14 learn *session automata*. ($\approx$ equivalent)
  - Sakamoto '97 learns deterministic register automata by modifying classical L^*. (incomparable)
- **Aim:** Learning algorithms for bar languages.

Fact (*Finitisation*)

A **finite alphabet** is enough for bar languages.
(α**-equivalence**)

Learning Bar Automata

Queries for Bar Languages in Angluin's Framework

T.CS FAU

» For bar languages $L_T \subseteq \overline{\mathbb{D}}^\star$ queries **change**:

Aim (*Modular Learning Algorithm* **)**

Use classical learning algorithms for bar languages.

Learning Bar Automata

Queries for Bar Languages in Angluin's Framework

- For bar languages $L_T \subseteq \overline{\mathbb{D}}^\star$ queries **change**:
  - **Membership Queries (MQ_α):**
    Given a bar word $w \in \overline{\mathbb{D}}^\star$, is $w \in L_T$?
    (**ANSWERS:** YES/NO)

Aim (*Modular Learning Algorithm* **)**

Use classical learning algorithms for bar languages.

Learning Bar Automata

Queries for Bar Languages in Angluin's Framework

- For bar languages $L_T \subseteq \overline{\mathbb{D}}^\star$ queries **change**:
  - **Membership Queries (MQ_α):**
    Given a bar word $w \in \overline{\mathbb{D}}^\star$, is $w \in L_T$?
    (**Answers:** Yes/No)
  - **Equivalence Queries (EQ_α):**
    Given a hypothesis $\mathcal{H}$, is $L_\alpha(\mathcal{H}) = L_T$?
    (**Answers:** Yes/Counterexample)

Aim (*Modular Learning Algorithm* **)**

Use classical learning algorithms for bar languages.

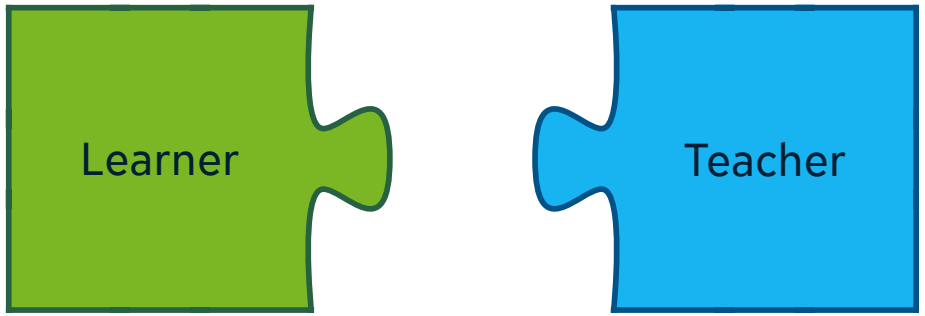
Learning Bar Automata

Queries for Bar Languages in Angluin's Framework

- For bar languages $L_T \subseteq \overline{\mathbb{D}}^\star$ queries **change**:
  - **Membership Queries (MQ_α):**
    Given a bar word $w \in \overline{\mathbb{D}}^\star$, is $w \in L_T$?
    (**Answers:** Yes/No)
  - **Equivalence Queries (EQ_α):**
    Given a hypothesis $\mathcal{H}$, is $L_\alpha(\mathcal{H}) = L_T$?
    (**Answers:** Yes/Counterexample)

Aim (*Modular Learning Algorithm* **)**

Use classical learning algorithms for bar languages.



Learning Bar Automata

Example

$\overline{\mathbb{D}}_0 = \{|a, |b, b\}$

$L_T = [|a|ba]_\alpha$

Learner

Teacher

Learning Bar Automata

Example

$\overline{\mathbb{D}}_0 = \{|a, |b, b\}$

$L_T = [|a|ba]_\alpha$

Learner

Teacher

MQ $w = \varepsilon$

Learning Bar Automata

Example

$\overline{\mathbb{D}}_0 = \{|a, |b, b\}$

$L_T = [|a|ba]_\alpha$

Learner

Teacher

MQ $w = \varepsilon$

No!

Learning Bar Automata

Example

$\overline{\mathbb{D}}_0 = \{\mathsf{l}a, \mathsf{l}b, b\}$

$L_T = [\mathsf{l}a|\mathsf{l}ba]_\alpha$

Learner

Teacher

MQ $w = \mathsf{l}a$

Learning Bar Automata

Example

$\overline{\mathbb{D}}_0 = \{\mathsf{l}a, \mathsf{l}b, b\}$

$L_T = [\mathsf{l}a\mathsf{l}ba]_\alpha$

Learner

Teacher

MQ $w = \mathsf{l}a$

No!

Learning Bar Automata

Example

$\overline{\mathbb{D}}_0 = \{|a, |b, b\}$

$L_T = [|a|ba]_\alpha$

Learner

Teacher

MQ ...

Learning Bar Automata

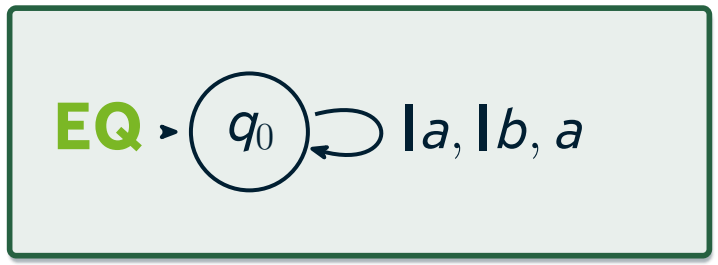
Example

$\overline{\mathbb{D}}_0 = \{|a, |b, b\}$

$L_T = [|a|ba]_\alpha$

Learner

Teacher



Learning Bar Automata

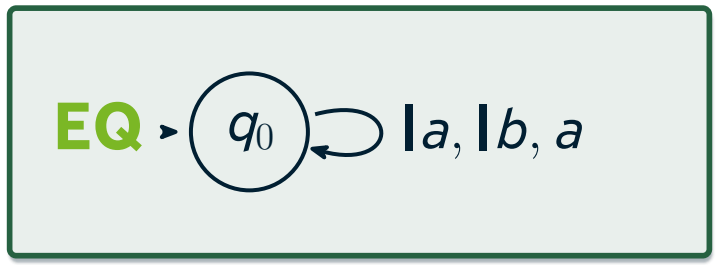
Example

$\overline{\mathbb{D}}_0 = \{|a, |b, b\}$

$L_T = [|a|ba]_\alpha$

Learner

Teacher



No, $w_T = |ff$ ↯

Learning Bar Automata

Working with Counterexamples

» There are a few **problems** with returned counterexamples:

Learning Bar Automata

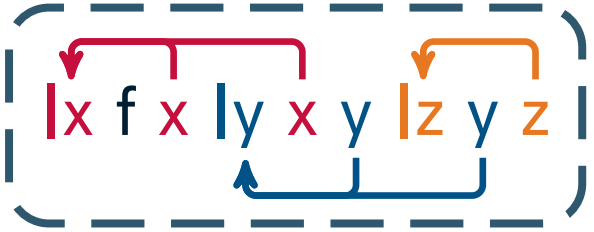
Working with Counterexamples

- There are a few **problems** with returned counterexamples:
  - What if the counterexample is not over the finite alphabet of the learner?

Learning Bar Automata

Working with Counterexamples

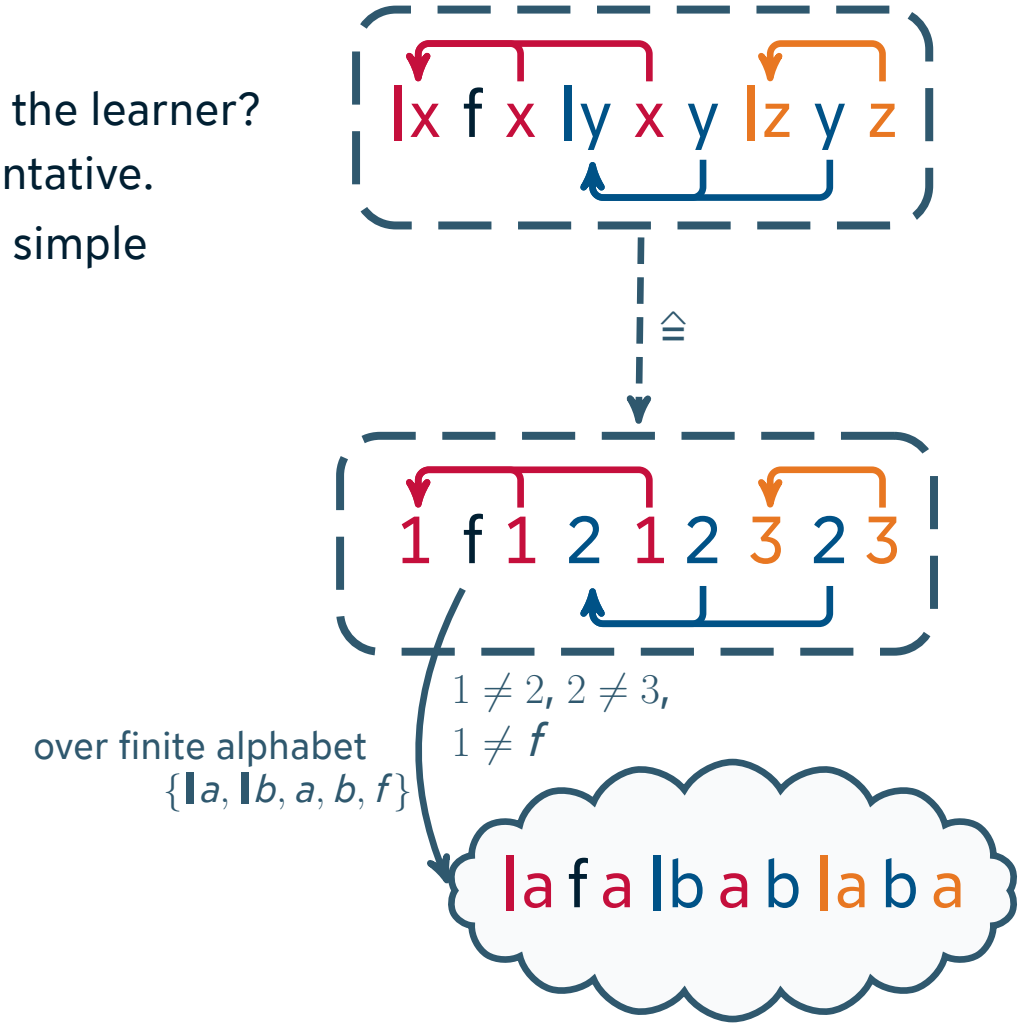
- There are a few **problems** with returned counterexamples:
  - What if the counterexample is not over the finite alphabet of the learner?
    - Using **normal forms**, we can easily find a ‘correct’ representative.
    - For finite bar strings, this works in **polynomial time** using simple constraints on the normal form and finite alphabets.



Learning Bar Automata

Working with Counterexamples

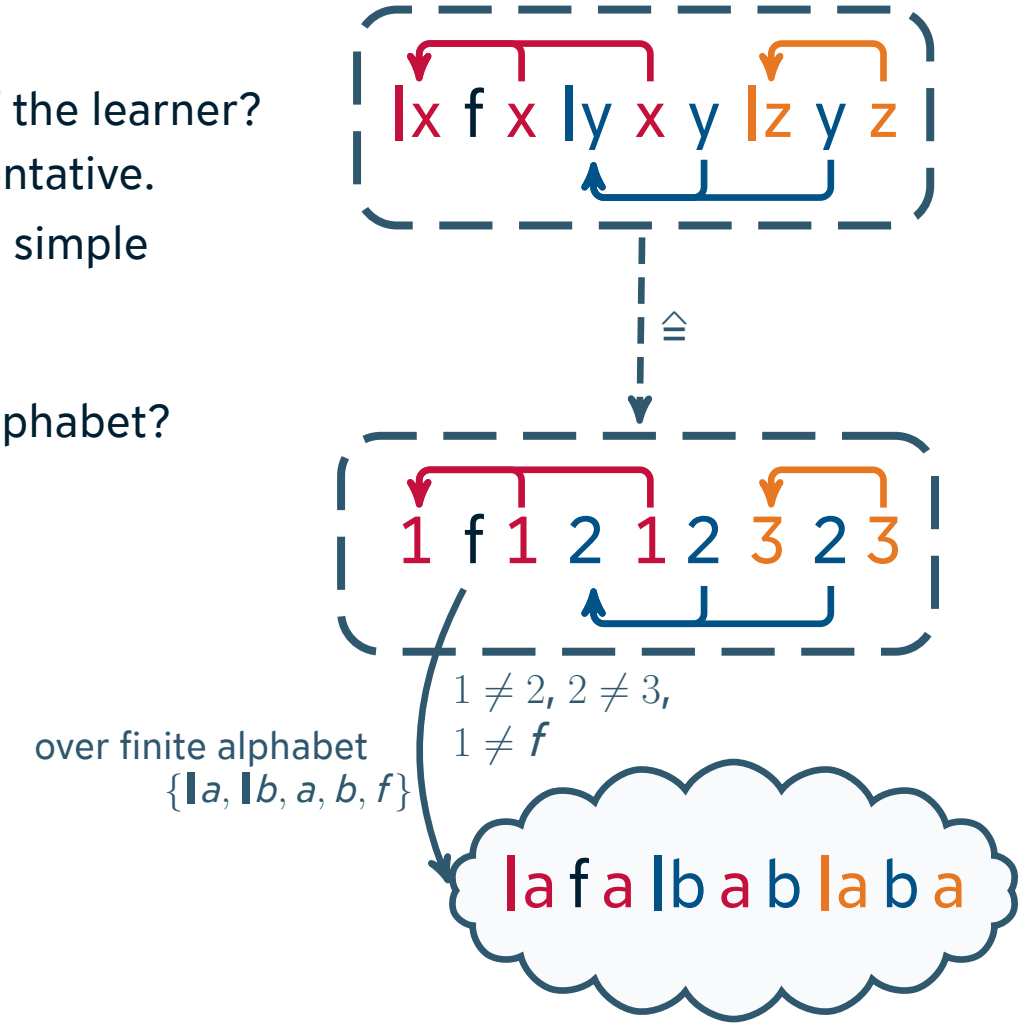
- There are a few **problems** with returned counterexamples:
  - What if the counterexample is not over the finite alphabet of the learner?
    - Using **normal forms**, we can easily find a ‘correct’ representative.
    - For finite bar strings, this works in **polynomial time** using simple constraints on the normal form and finite alphabets.



Learning Bar Automata

Working with Counterexamples

- There are a few **problems** with returned counterexamples:
  - What if the counterexample is not over the finite alphabet of the learner?
    - Using **normal forms**, we can easily find a 'correct' representative.
    - For finite bar strings, this works in **polynomial time** using simple constraints on the normal form and finite alphabets.
  - What if there is no suitable counterexample over the finite alphabet?
    - ⇝ solve via iterative process



Learning Bar Automata

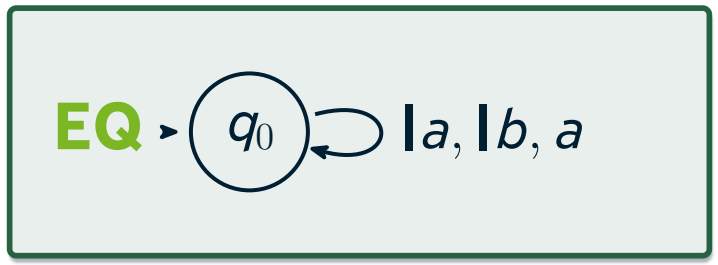
Example

$L_T = [\mathbf{l}a\mathbf{l}ba]_\alpha$

$\overline{\mathbb{D}}_0 = \{\mathbf{l}a, \mathbf{l}b, b\}$

Learner

Teacher



No, $w_T = \mathbf{l}ff \longrightarrow \mathbf{l}aa$

Learning Bar Automata

Example

$\overline{\mathbb{D}}_0 = \{\mathsf{l}a, \mathsf{l}b, b\}$

$L_T = [\mathsf{l}a\mathsf{l}ba]_\alpha$

Learner

Teacher

MQ ...

Learning Bar Automata

Example

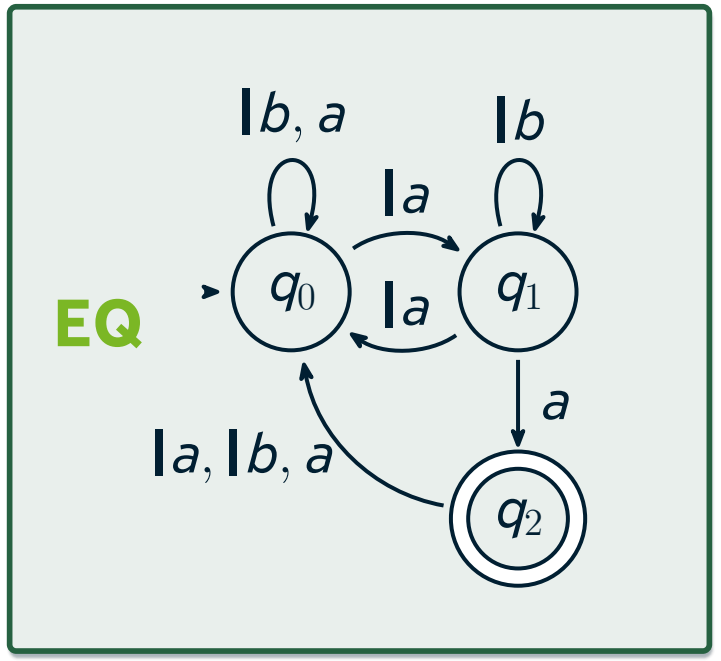
$\overline{\mathbb{D}}_0 = \{|a, |b, b\}$

$L_T = [|a|ba]_\alpha$

Learner

Teacher

EQ



Learning Bar Automata

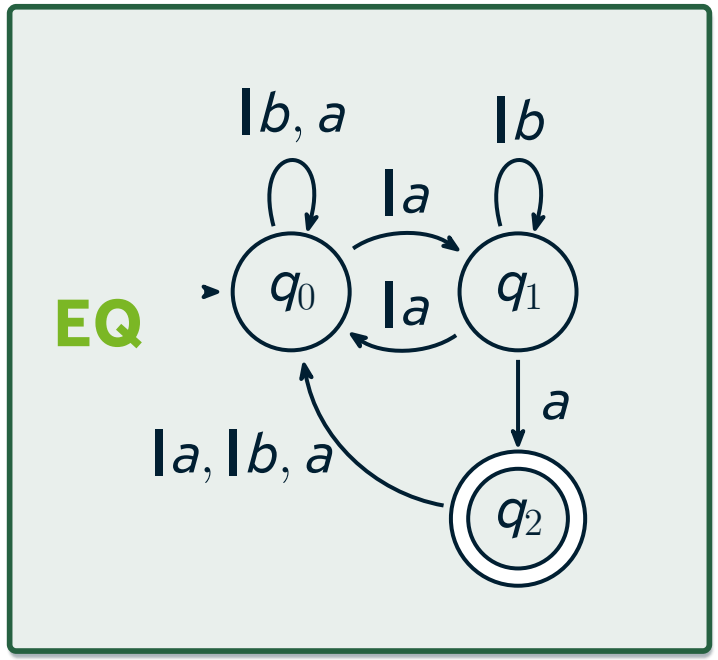
Example

$L_T = [\mathsf{I}a\mathsf{I}ba]_\alpha$

$\overline{\mathbb{D}}_0 = \{\mathsf{I}a, \mathsf{I}b, b\}$

Learner

Teacher

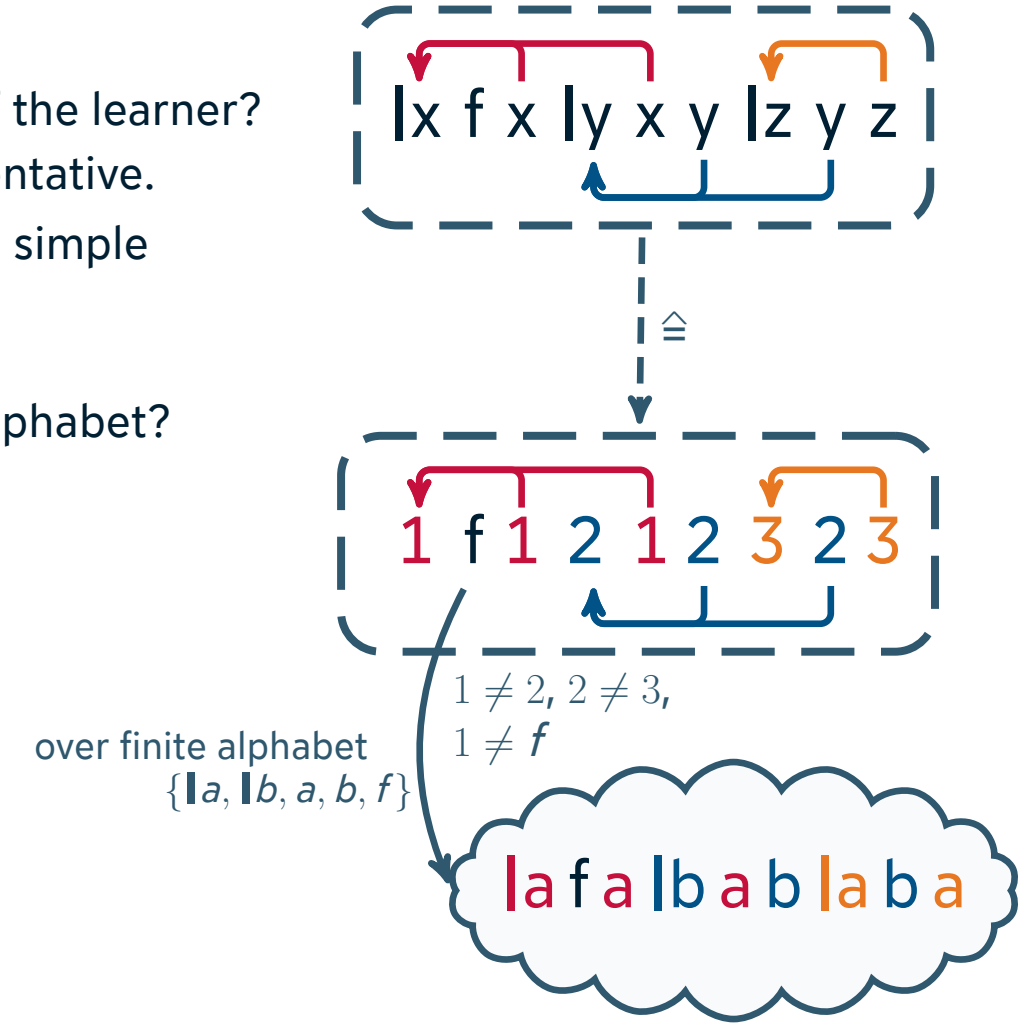


No, $w_T = \mathsf{I}a\mathsf{I}aa$

Learning Bar Automata

Working with Counterexamples

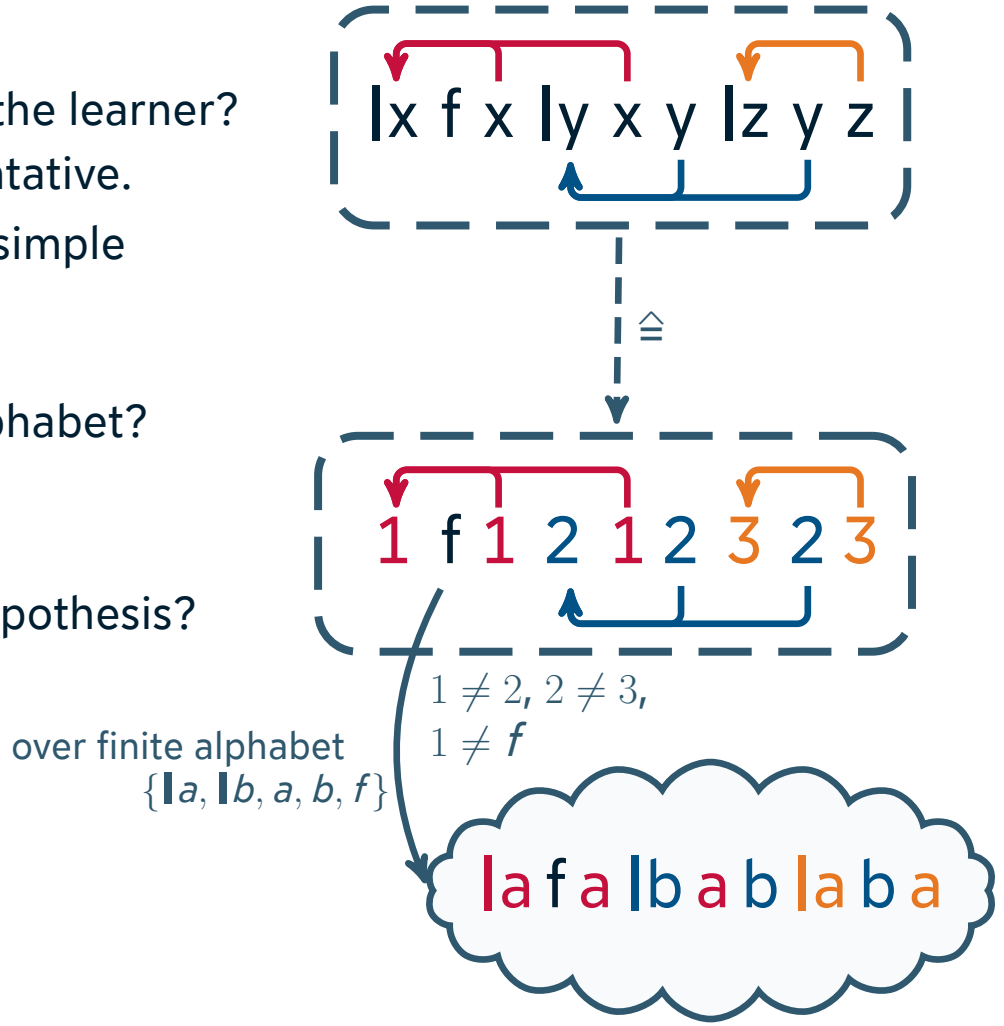
- There are a few **problems** with returned counterexamples:
  - What if the counterexample is not over the finite alphabet of the learner?
    - Using **normal forms**, we can easily find a 'correct' representative.
    - For finite bar strings, this works in **polynomial time** using simple constraints on the normal form and finite alphabets.
  - What if there is no suitable counterexample over the finite alphabet?
    - ⇝ solve via iterative process



Learning Bar Automata

Working with Counterexamples

- There are a few **problems** with returned counterexamples:
  - What if the counterexample is not over the finite alphabet of the learner?
    - Using **normal forms**, we can easily find a ‘correct’ representative.
    - For finite bar strings, this works in **polynomial time** using simple constraints on the normal form and finite alphabets.
  - What if there is no suitable counterexample over the finite alphabet?
    - ⇝ solve via iterative process
  - What if the counterexample is not directly accepted by the hypothesis?

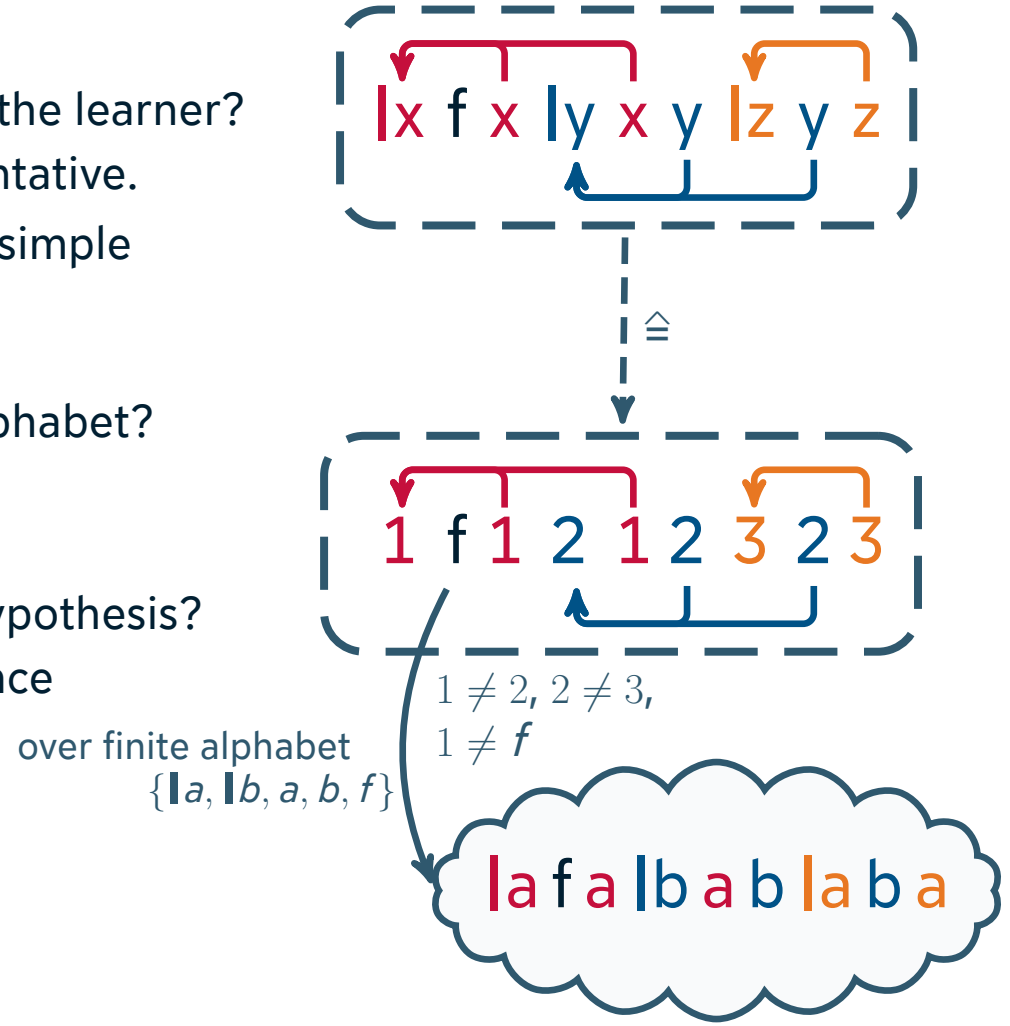


Learning Bar Automata

Working with Counterexamples

- There are a few **problems** with returned counterexamples:
  - What if the counterexample is not over the finite alphabet of the learner?
    - Using **normal forms**, we can easily find a 'correct' representative.
    - For finite bar strings, this works in **polynomial time** using simple constraints on the normal form and finite alphabets.
  - What if there is no suitable counterexample over the finite alphabet?

    ⇝ solve via iterative process
  - What if the counterexample is not directly accepted by the hypothesis?
    - We need to solve the problem of bar acceptance/acceptance modulo α-equivalence.
    - For finite bar strings, this problem is **NP-complete**.



Learning Bar Automata

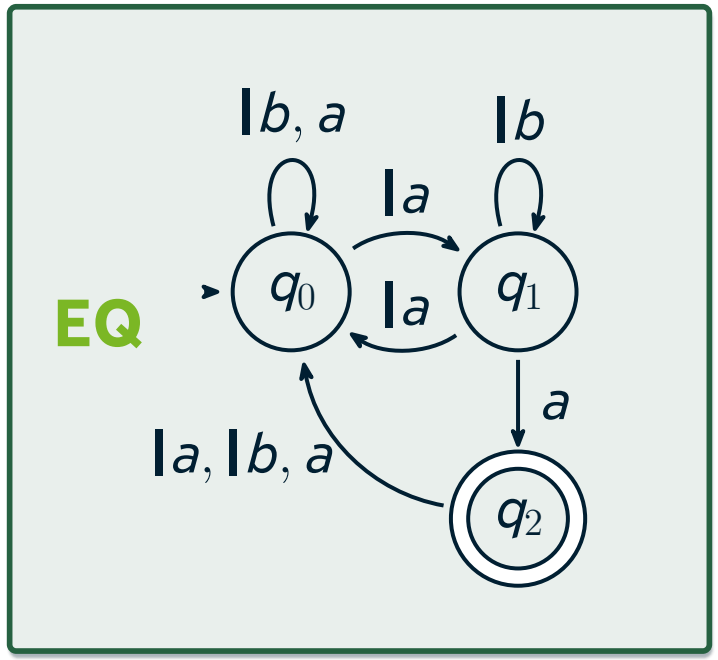
Example

$\overline{\mathbb{D}}_0 = \{|a, |b, b\}$

$L_T = [|a|ba]_\alpha$

Learner

Teacher



No, $w_T = |a|aa \to |b|aa$

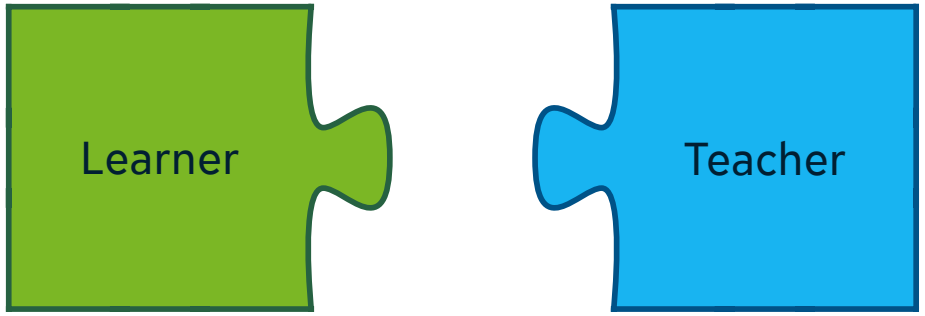
Learning Bar Automata

Queries for Bar Languages in Angluin's Framework

- For bar languages $L_T \subseteq \overline{\mathbb{D}}^\star$ queries **change**:
  - **Membership Queries (MQ_α):**
    Given a bar word $w \in \overline{\mathbb{D}}^\star$, is $w \in L_T$?
    (**Answers:** Yes/No)
  - **Equivalence Queries (EQ_α):**
    Given a hypothesis $\mathcal{H}$, is $L_\alpha(\mathcal{H}) = L_T$?
    (**Answers:** Yes/Counterexample)

Aim (*Modular Learning Algorithm* **)**

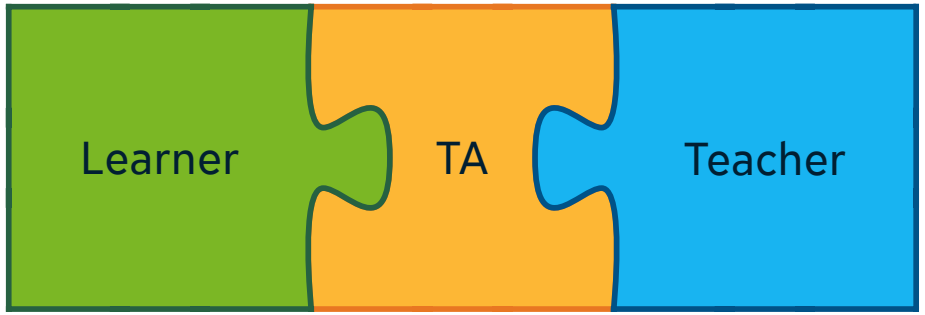
Use classical learning algorithms for bar languages.



Learning Bar Automata

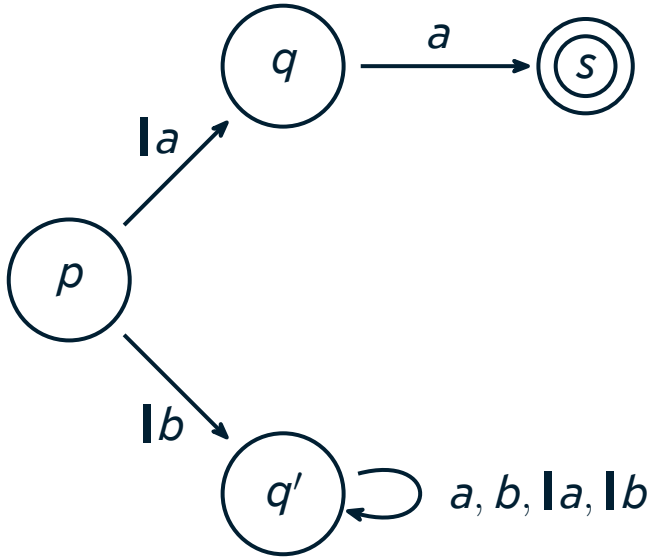
Queries for Bar Languages in Angluin's Framework

- For bar languages $L_T \subseteq \overline{\mathbb{D}}^\star$ queries **change**:
  - **Membership Queries (MQ_α):**
    Given a bar word $w \in \overline{\mathbb{D}}^\star$, is $w \in L_T$?
    (**Answers:** Yes/No)
  - **Equivalence Queries (EQ_α):**
    Given a hypothesis $\mathcal{H}$, is $L_\alpha(\mathcal{H}) = L_T$?
    (**Answers:** Yes/Counterexample)



Aim (*Modular Learning Algorithm* **)**

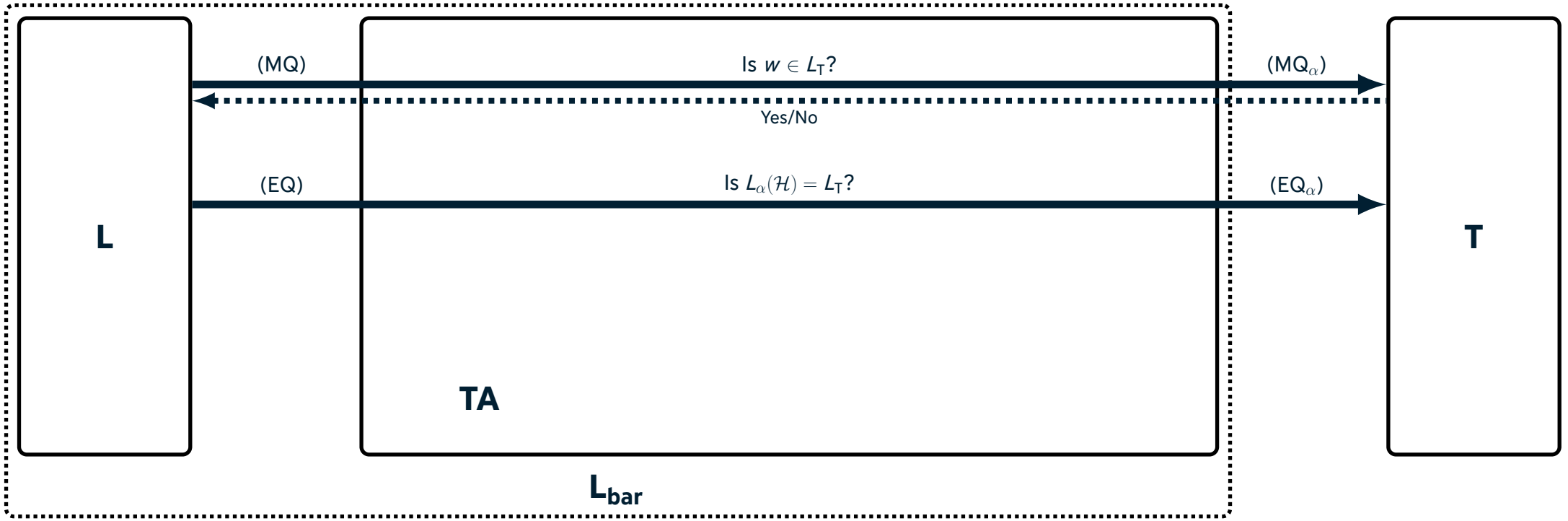
Use classical learning algorithms for bar languages.



Counterexample: $|bb \in L_\alpha(\mathcal{H}) \setminus L(\mathcal{H})$ ↯

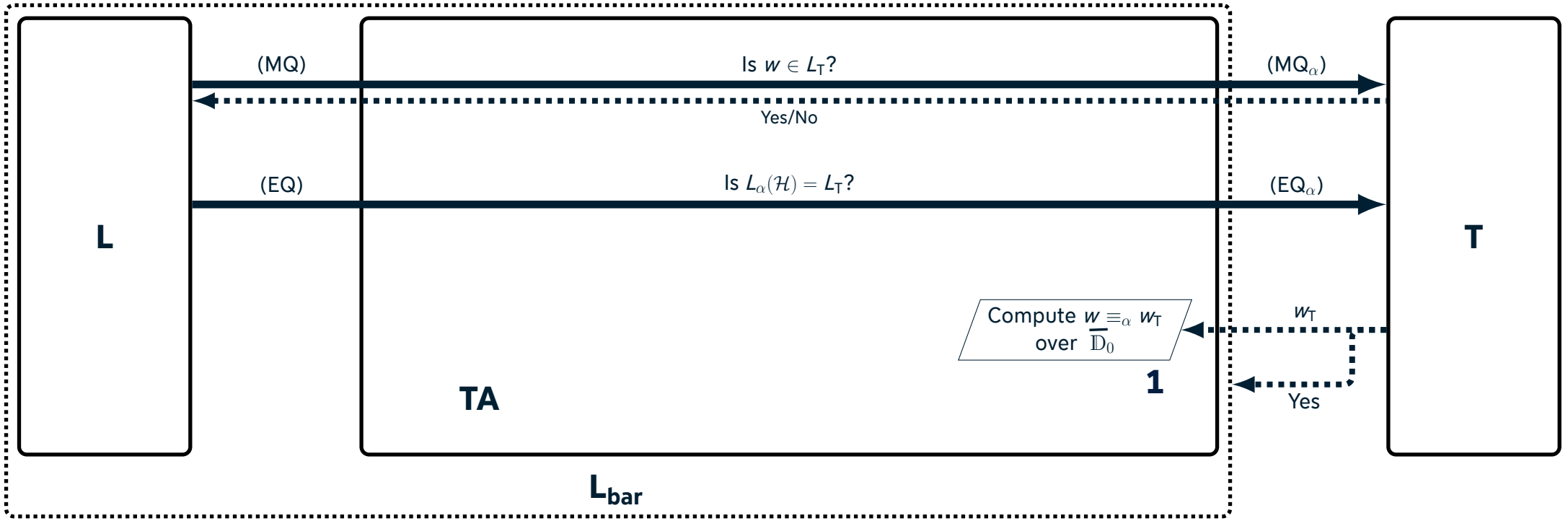
Learning Bar Automata

Modular Algorithm for Bar Automata



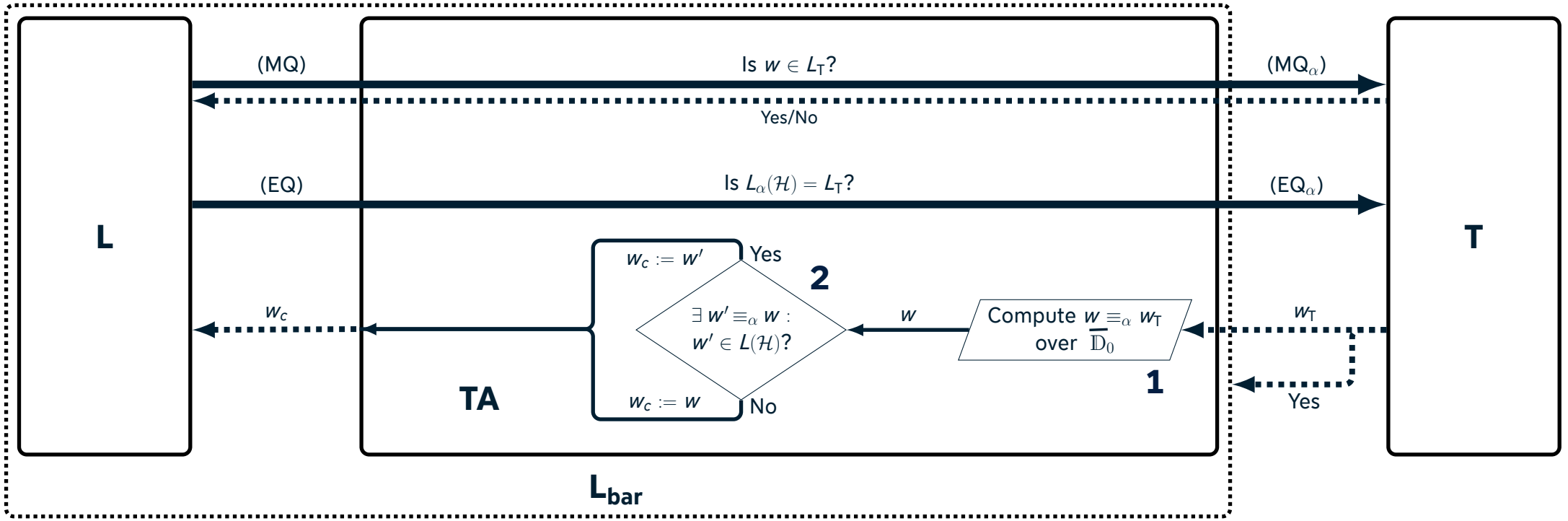
Learning Bar Automata

Modular Algorithm for Bar Automata



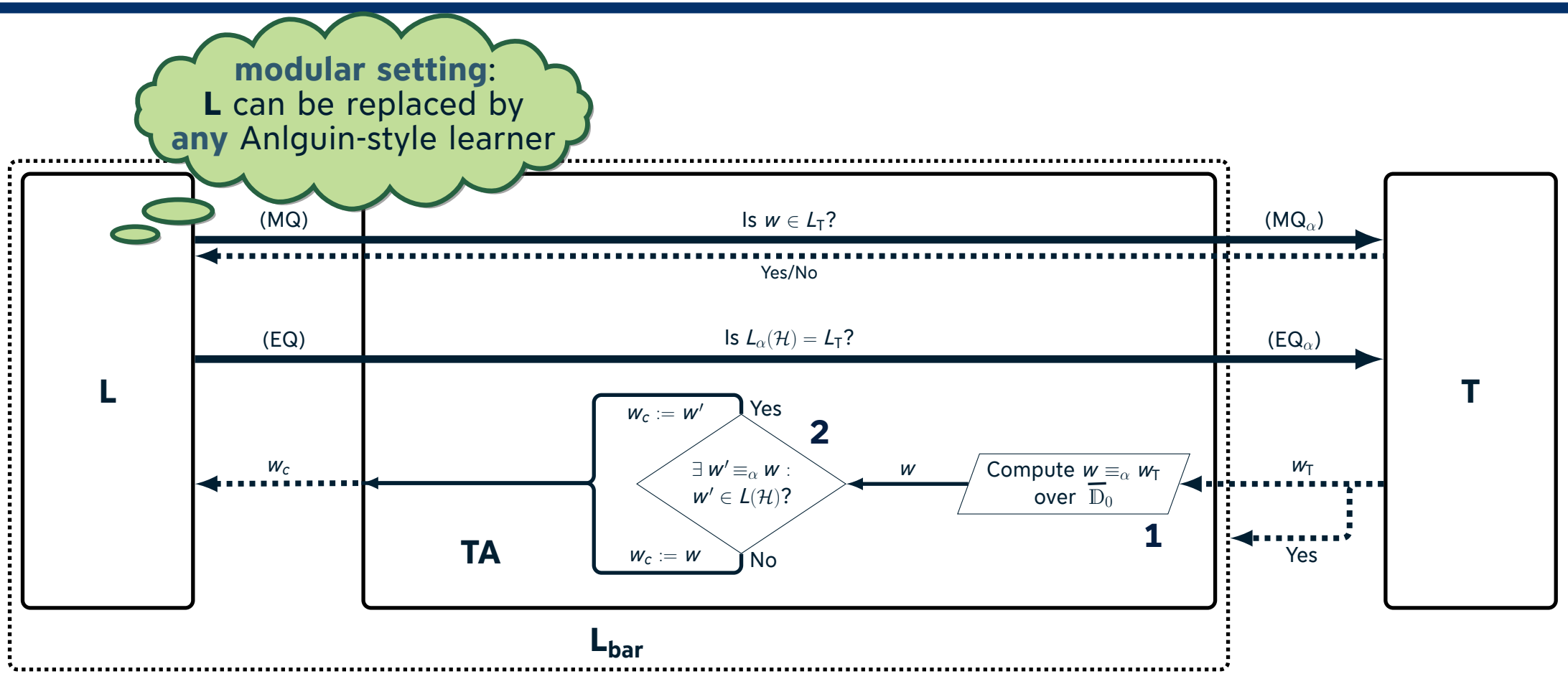
Learning Bar Automata

Modular Algorithm for Bar Automata



Learning Bar Automata

Modular Algorithm for Bar Automata



Learning Bar Languages

Correctness, Termination and Complexity

» **Correctness** of our algorithm follows directly from correctness of **L**.

Learning Bar Languages

Correctness, Termination and Complexity

- **Correctness** of our algorithm follows directly from correctness of **L**.
- What about the number of queries we ask/forward to the teacher? What about **termination**?

Learning Bar Languages

Correctness, Termination and Complexity

- **Correctness** of our algorithm follows directly from correctness of **L**.
- What about the number of queries we ask/forward to the teacher? What about **termination**?

Theorem (*Closure-Construction* **)**

For every bar language of a bar automaton, there is a bar automaton accepting it **directly**.

Learning Bar Languages

Correctness, Termination and Complexity

- **Correctness** of our algorithm follows directly from correctness of **L**.
- What about the number of queries we ask/forward to the teacher? What about **termination**?

Theorem (*Closure-Construction* **)**

For every bar language of a bar automaton, there is a bar automaton accepting it **directly**.

- This gives us an upper bound (again by **L**):

We ask **at most** as many queries as **L** would with a classical teacher.

Learning Bar Languages

Correctness, Termination and Complexity

- **Correctness** of our algorithm follows directly from correctness of **L**.
- What about the number of queries we ask/forward to the teacher? What about **termination**?

Theorem (*Closure-Construction* **)**

For every bar language of a bar automaton, there is a bar automaton accepting it **directly**.

- This gives us an upper bound (again by **L**):

We ask **at most** as many queries as **L** would with a classical teacher.

- This **need not** be a lower bound!

Learning Bar Languages

Problems with Counterexamples

What happens if we cannot find a suitable counterexample over the finite alphabet?

- Typically: Number of ‘registers’ **unknown**.

Learning Bar Languages

Problems with Counterexamples

What happens if we cannot find a suitable counterexample over the finite alphabet?

- Typically: Number of ‘registers’ **unknown**.
- **Solution:** Extend alphabet until representative is found.

(restart $\mathbf{L_{bar}}$ from scratch)

Learning Bar Languages

Problems with Counterexamples

What happens if we cannot find a suitable counterexample over the finite alphabet?

- Typically: Number of ‘registers’ **unknown**.
- **Solution:** Extend alphabet until representative is found.

(restart $\mathbf{L_{bar}}$ from scratch)

Extension can be calculated from **normal form**!

Learning Bar Languages

Problems with Counterexamples

What happens if we cannot find a suitable counterexample over the finite alphabet?

- Typically: Number of ‘registers’ **unknown**.
- **Solution:** Extend alphabet until representative is found.

(restart $\mathbf{L_{bar}}$ from scratch)

Extension can be calculated from **normal form**!

Theorem (*Query Complexity*)

Number of queries **add up** over all restarts!

Learning Bar Languages

Problems with Counterexamples

What happens if we cannot find a suitable counterexample over the finite alphabet?

- Typically: Number of ‘registers’ **unknown**.
- **Solution:** Extend alphabet until representative is found.

(restart $\mathbf{L_{bar}}$ from scratch)

Extension can be calculated from **normal form**!

Theorem (*Query Complexity* **)**

Number of queries **add up** over all restarts!

No normal form? For counterexample $w_T \in \overline{\mathbb{D}}^\star$, test new $\overline{\mathbb{D}}'_\mathsf{L}$ **by cardinality**:

$$\overline{\mathbb{D}}_\mathsf{L} \subseteq \overline{\mathbb{D}}'_\mathsf{L} \subseteq (\overline{\mathbb{D}}_\mathsf{cex} \cup \overline{\mathbb{D}}_\mathsf{L})$$

Generalisation: Requirements of L_{bar}

What is necessary for the arguments to go through?

Generalisation: Requirements of L_{bar}

What is necessary for the arguments to go through?

1 **Finitisation- and Closure-Principle**

Generalisation: Requirements of L_{bar}

What is necessary for the arguments to go through?

1 **Finitisation- and Closure-Principle**

2 **Checking representatives over finite alphabets**

Generalisation: Requirements of L_{bar}

What is necessary for the arguments to go through?

1 **Finitisation- and Closure-Principle**

2 **Checking representatives over finite alphabets**

3 **Decidability of Bar Acceptance Problem**

Generalisation: Requirements of L_{bar}

What is necessary for the arguments to go through?

1 **Finitisation- and Closure-Principle**

T Bar tree automata satisfy **all** requirements!

2 **Checking representatives over finite alphabets**

3 **Decidability of Bar Acceptance Problem**

Generalisation: Requirements of L_{bar}

What is necessary for the arguments to go through?

1 **Finitisation- and Closure-Principle**

T Bar tree automata satisfy **all** requirements!

Büchi automata satisfy **all** requirements:

B **no** normal form available

more difficult processes used here

2 **Checking representatives over finite alphabets**

3 **Decidability of Bar Acceptance Problem**

Generalisation: Requirements of L_{bar}

What is necessary for the arguments to go through?

1 **Finitisation- and Closure-Principle**

T Bar tree automata satisfy **all** requirements!

Büchi automata satisfy **all** requirements:

B **no** normal form available

more difficult processes used here

2 **Checking representatives over finite alphabets**

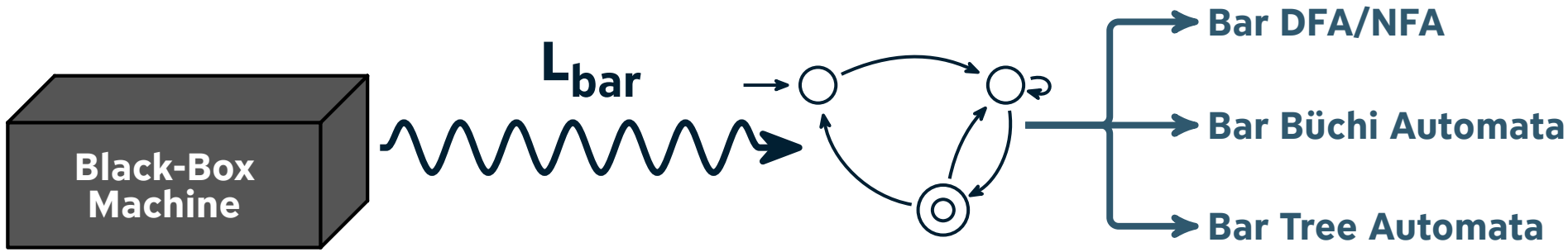
3 **Decidability of Bar Acceptance Problem**

G Under these conditions, our results generalise!

Learning Bar Automata

Conclusions and Future Work

- We developed a modular learning algorithm for bar languages with a good query complexity:



- The algorithm is general in the system type under few **conditions**.

- Some remaining problems:
  - **Generalisations to other types/coalgebras**
  - **Learn data languages**
  - **Conformance Testing/Passive Learning**

References

- Bollig, Benedikt, Peter Habermehl, Martin Leucker, Benjamin Monmege. **'A Robust Class of Data Languages and an Application to Learning'.** *Logical Methods in Computer Science* 10.4 (2014). doi: 10.2168/LMCS-10(4:19)2014.
- Prucker, Simon, Lutz Schröder. **'Nominal Tree Automata with Name Allocation'.** *35th International Conference on Concurrency Theory (CONCUR 2024)*. Ed. by Rupak Majumdar, Alexandra Silva. Vol. 311. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 35:1–35:17. doi: 10.4230/LIPIcs.CONCUR.2024.35.
- Sakamoto, Hiroshi. **'Learning simple deterministic finite-memory automata'.** *Algorithmic Learning Theory. ALT 1997*. Ed. by Ming Li, Akira Maruoka. Vol. 1316. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 1997, pp. 416–431. doi: 10.1007/3-540-63577-7_58.
- Urbat, Henning, Daniel Hausmann, Stefan Milius, Lutz Schröder. **'Nominal Büchi Automata with Name Allocation'.** *32nd International Conference on Concurrency Theory (CONCUR 2021)*. Ed. by Serge Haddad, Daniele Varacca. Vol. 203. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 4:1–4:16. doi: 10.4230/LIPIcs.CONCUR.2021.4.