Learning Communicating and Nondeterministic Automata

Carsten Kern

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

Oberseminar
Aachen, August 31st 2009
Main results

- New learning algorithm for NFA (NL*)
- Extension of existing learning algorithm towards learning CFMs
- Optimizations of learning algorithms
- Tools implementing these algorithms
- New software lifecycle model embedding our learning approach
Results of this thesis

Main results covered in this presentation

- New learning algorithm for NFA (NL*)
- Extension of existing learning algorithm towards learning CFMs
- Optimizations of learning algorithms
- Tools implementing these algorithms
- New software lifecycle model embedding our learning approach
Areas of application

- Formal verification (e.g., regular model checking)
- Bioinformatics (e.g., prediction of structure of proteins)
- Robotics (e.g., learning environment models)
- Computational linguistics (e.g., compiling idiom dictionaries)
- ...
Outline

1. Learning Deterministic Automata
2. Learning Nondeterministic Automata
3. Learning Communicating Automata
4. Tools
5. Conclusion
1. Learning Deterministic Automata

2. Learning Nondeterministic Automata

3. Learning Communicating Automata

4. Tools

5. Conclusion
Here, learning means:

Given exemplifying behavior of a system

Learn a *model* conforming to the given behavior
Here, learning means:

Given exemplifying behavior of a system
in terms of words

Learn a *model* conforming to the given behavior
in terms of a regular language (deterministic finite automaton, DFA)
Learning

Here, learning means:

Given exemplifying behavior of a system
in terms of words

Learn a *model* conforming to the given behavior
in terms of a regular language (deterministic finite automaton, DFA)

Active Learning

- The learner is given *positive* and *negative* examples
Learning

Here, learning means:

Given exemplifying behavior of a system
in terms of words

Learn a *model* conforming to the given behavior
in terms of a regular language (deterministic finite automaton, DFA)

Active Learning

- The learner is given positive and negative examples
- The learner can actively ask specific questions
Learning

Here, learning means:

Given exemplifying behavior of a system
in terms of words

Learn a *model* conforming to the given behavior
in terms of a regular language (deterministic finite automaton, DFA)

Active Learning

- The learner is given *positive* and *negative* examples
- The learner can actively ask specific questions

Occam’s razor:

“In case of different explanations, choose the *simplest* one.”
Learning

Here, learning means:

Given exemplifying behavior of a system in terms of words

Learn a model conforming to the given behavior in terms of a regular language (deterministic finite automaton, DFA)

Active Learning

- The learner is given positive and negative examples
- The learner can actively ask specific questions

Occam’s razor:

“In case of different explanations, choose the simplest one.”
⇒ Learn the minimal DFA conforming to given examples
Algorithm - Overview

Learner

Teacher

Oracle

L: (regular) language to learn
Algorithm - Overview

\[ L: \text{ (regular) language to learn} \]
Algorithm - Overview

- **L**: (regular) language to learn

**Teacher**
- **Membership queries**
  - Is \( w \in \Sigma^* \) a member of language \( L \)?

**Learner**
- **Membership queries**
  - Yes/No

**Oracle**
- **Equivalence queries**
  - Yes/Counterexample

**Teacher**
- **Membership queries**
  - Is \( \mathcal{H} \) a hypothesis?
  - Is \( \mathcal{H} \) equivalent to system to learn?
Algorithm - Overview

Teacher

Learner

Yes/No

Is $w \in \Sigma^*$ a member of language $L$?

Oracle

Yes/Counterexample

Let $\mathcal{H}$ be a hypothesis
Is $\mathcal{H}$ equivalent to system to learn?

Membership queries

Equivalence queries

$L$: (regular) language to learn
Algorithm - Overview

- **Teacher**
  - **Membership queries**
  - Is $w \in \Sigma^*$ a member of language $L$?

- **Learner**
  - **Yes/No**

- **Oracle**
  - **Equivalence queries**
  - Yes/Counterexample
  - Let $\mathcal{H}$ be a hypothesis
  - Is $\mathcal{H}$ equivalent to system to learn?

$L$: (regular) language to learn
$L$: (regular) language to learn
Algorithm - Overview

Teacher

Membership queries

Yes/No

Is $w \in \Sigma^*$ a member of language $L$?

Learner

Yes/Countereexample

L: (regular) language to learn

Counterexample: $w \in (L(\mathcal{H}) \setminus L) \cup (L \setminus L(\mathcal{H}))$
Let $\Sigma = \{a, b\}$

$T$

$\varepsilon$

$\varepsilon$

$a$

$b$

$aa$

$ab$

$\varepsilon \in L?$
Table-based learning

Let $\Sigma = \{a, b\}$

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

$a \in L$?
Table-based learning

Let $\Sigma = \{a, b\}$

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
</tr>
<tr>
<td>$b$</td>
<td></td>
</tr>
<tr>
<td>$aa$</td>
<td></td>
</tr>
<tr>
<td>$ab$</td>
<td></td>
</tr>
</tbody>
</table>

$b, aa, ab \in L$?
Let $\Sigma = \{a, b\}$

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$-$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
</tr>
</tbody>
</table>
Let $\Sigma = \{a, b\}$

<table>
<thead>
<tr>
<th>$\mathcal{T}$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>+</td>
</tr>
<tr>
<td>$a$</td>
<td>-</td>
</tr>
<tr>
<td>$b$</td>
<td>-</td>
</tr>
<tr>
<td>$aa$</td>
<td>-</td>
</tr>
<tr>
<td>$ab$</td>
<td>+</td>
</tr>
</tbody>
</table>

To derive an automaton:

- $\mathcal{T}$ must be **closed**, i.e., all states are derivable from $\mathcal{T}$
Let $\Sigma = \{a, b\}$

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$-$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
</tr>
</tbody>
</table>

To derive an automaton:

- $T$ must be **closed**, i.e., all states are derivable from $T$
- $T$ must be **consistent**, i.e., there are no contradicting transitions
Let $\Sigma = \{a, b\}$

<table>
<thead>
<tr>
<th>$\mathcal{T}$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>+</td>
</tr>
<tr>
<td>$a$</td>
<td>-</td>
</tr>
<tr>
<td>$b$</td>
<td>-</td>
</tr>
<tr>
<td>$aa$</td>
<td>-</td>
</tr>
<tr>
<td>$ab$</td>
<td>+</td>
</tr>
</tbody>
</table>

To derive an automaton:

- $\mathcal{T}$ must be **closed**, i.e., all states are derivable from $\mathcal{T}$
- $\mathcal{T}$ must be **consistent**, i.e., there are no contradicting transitions

To this end:

- upper rows serve to derive states
- lower rows serve to derive transitions
Let $\Sigma = \{a, b\}$

<table>
<thead>
<tr>
<th>$\mathcal{T}$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$-$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
</tr>
</tbody>
</table>

$b b \in L(\mathcal{H})$ but $b b \notin L$!
Let $\Sigma = \{a, b\}$

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$-$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
</tr>
</tbody>
</table>

Counterexample can be added to:

$bb \in L(\mathcal{H})$ but $bb \notin L!$
Table-based learning

Let $\Sigma = \{a, b\}$

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
</tr>
<tr>
<td>$bb$</td>
<td>$-$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$-$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
</tr>
<tr>
<td>$ba$</td>
<td>$-$</td>
</tr>
<tr>
<td>$bba$</td>
<td>$-$</td>
</tr>
<tr>
<td>$bbb$</td>
<td>$-$</td>
</tr>
</tbody>
</table>

Counterexample can be added to:

- the rows ($L^*$)
Table-based learning

Let $\Sigma = \{a, b\}$

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\varepsilon$</th>
<th>$bb$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>+</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>$a$</td>
<td>−</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td>$b$</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>$aa$</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>$ab$</td>
<td>+</td>
<td>−</td>
<td>−</td>
</tr>
</tbody>
</table>

Counterexample can be added to:

- the rows ($L^*$)
- the columns ($L_{col}^*$
Angluin’s Algorithm L*

Theorem (Complexity of L*)

Let:

- $n$: number of states of the minimal DFA $A_L$ for regular language $L$,
- $m$: length of the biggest counterexample

Then, $L^*$ returns after at most:

the minimal DFA $A$. 
Angluin’s Algorithm $L^*$

**Theorem (Complexity of $L^*$)**

Let:
- $n$: number of states of the minimal DFA $A_L$ for regular language $L$,
- $m$: length of the biggest counterexample

Then, $L^*$ returns after at most:
- $n$ equivalence queries and

the minimal DFA $A$. 
Angluin’s Algorithm $L^*$

**Theorem (Complexity of $L^*$)**

Let:
- $n$: number of states of the minimal DFA $A_L$ for regular language $L$,
- $m$: length of the biggest counterexample

Then, $L^*$ returns after at most:
- $n$ equivalence queries and
- $O(m|\Sigma|n^2)$ membership queries

the minimal DFA $A$. 
But there is a problem . . .

minimal DFA can be huge!
But there is a problem . . .

minimal DFA can be huge!
What about more succinct representations like NFA?

Can we learn (a certain subclass of) NFA?
But there is a problem . . .

minimal DFA can be huge!
What about more succinct representations like NFA?

Can we learn (a certain subclass of) NFA?

Yes, we can!
Motivation

- DFA can be huge
- e.g., verification is much more efficient on smaller models
Motivation

- DFA can be huge
- e.g., verification is much more efficient on smaller models

Goal

- Learn more compact representations of regular languages
Motivation
- DFA can be huge
- e.g., verification is much more efficient on smaller models

Goal
- Learn more compact representations of regular languages

How?
- Use residual finite-state automata (RFSA) for learning
Residual Finite-State Automata [Denis et al.]
Residual Finite-State Automata [Denis et al.]
Residual Finite-State Automata [Denis et al.]
Residual Finite-State Automata [Denis et al.]

\[ L_2 \]

\[ L_1 = L_2 \cup L_3 \]

\[ L_3 \]

- \( q_0 \) to \( q_1 \) via \( a \)
- \( q_2 \) to \( L_2 \)
- \( q_3 \) to \( L_3 \)
Residual Finite-State Automata [Denis et al.]
Residual Finite-State Automata [Denis et al.]

\[ L_1 = L_2 \cup L_3 \]
Residual Finite-State Automata [Denis et al.]

\[ L_1 = L_2 \cup L_3 \]

\[ L_2 \]

\[ L_3 \]

State transitions:
- \( q_0 \) to \( q_1 \) on input 'a'
- \( q_0 \) to \( q_2 \) on input 'a'
- \( q_1 \) to \( q_2 \) on input 'a'
- \( q_2 \) to \( q_3 \) on input 'a'
- \( q_3 \) to \( q_0 \) on input 'a'

Restrictions:
- No input allowed from \( q_1 \)
Residual Finite-State Automata [Denis et al.]

\[ L_1 = L_2 \cup L_3 \]
Residual Finite-State Automata [Denis et al.]
Residual Finite-State Automata

**Definition (Residual Language)**

For a language $L \subseteq \Sigma^*$ and $u \in \Sigma^*$:

$$u^{-1}L = \{v \in \Sigma^* \mid uv \in L\} \quad (u\text{-residual of } L)$$

$L'$ is a residual language of $L$ if: $\exists u \in \Sigma^*$ with $L' = u^{-1}L$.

$Res(L)$: the set of residual languages of $L$. 
Residual Finite-State Automata

\[ \varepsilon^{-1}L = \Sigma^*a\Sigma \quad (= L_{q_0}) \]

**Definition (Residual Language)**

For a language \( L \subseteq \Sigma^* \) and \( u \in \Sigma^* \):

\[ u^{-1}L = \{ v \in \Sigma^* \mid uv \in L \} \quad (u\text{-residual of } L) \]

\( L' \subseteq \Sigma^* \) is a **residual language of** \( L \) if: \( \exists u \in \Sigma^* \text{ with } L' = u^{-1}L \).

\( \text{Res}(L) \): the set of residual languages of \( L \).
Residual Finite-State Automata

Definition (Residual Language)

For a language \( L \subseteq \Sigma^* \) and \( u \in \Sigma^* \):

\[
u^{-1}L = \{ v \in \Sigma^* \mid uv \in L \} \quad (u\text{-residual of } L)
\]

\( L' \subseteq \Sigma^* \) is a residual language of \( L \) if: \( \exists u \in \Sigma^* \) with \( L' = u^{-1}L \).

\( Res(L) \): the set of residual languages of \( L \).
A residual finite-state automaton (RFSA) over $\Sigma$ is an NFA $R = (Q, Q_0, F, \delta)$ such that for each $q \in Q$, $L_q \in Res(L(R))$. 
Residual Finite-State Automata

\[
\begin{align*}
\varepsilon^{-1}L &= \Sigma^*a\Sigma \quad (= L_{q_0}) \\
(a)^{-1}L &= \Sigma^*a\Sigma \cup \Sigma \quad (= L_{q_1}) \\
(b)^{-1}L &= \Sigma^*a\Sigma \cup \Sigma \cup \{\varepsilon\} \quad (= L_{q_2}) \\
(ab)^{-1}L &= \Sigma^*a\Sigma \cup \{\varepsilon\} \quad (= L_{q_3})
\end{align*}
\]

Definition (Residual Finite-State Automaton)

A residual finite-state automaton (RFSA) over \(\Sigma\) is an NFA \(\mathcal{R} = (Q, Q_0, F, \delta)\) such that for each \(q \in Q\), \(L_q \in \text{Res}(L(\mathcal{R}))\).
Towards canonical RFSA

Definition (Prime and Composed Residuals)

Let \( L \subseteq \Sigma^* \) be a language. A residual \( L' \in Res(L) \) is called composed if there are \( L_1, \ldots, L_n \in Res(L) \setminus \{L'\} \) such that

\[
L' = L_1 \cup \ldots \cup L_n
\]

Otherwise, it is called prime.

The set of prime residuals of \( L \) is denoted by \( Primes(L) \).
Towards canonical RFSA

\[ \varepsilon^{-1} L = \Sigma^* a \Sigma \quad (= L_{q_0}) \]
\[ (a)^{-1} L = \Sigma^* a \Sigma \cup \Sigma \quad (= L_{q_1}) \]
\[ (b)^{-1} L = \Sigma^* a \Sigma \cup \Sigma \cup \{\varepsilon\} \quad (= L_{q_2}) \]
\[ (ab)^{-1} L = \Sigma^* a \Sigma \cup \{\varepsilon\} \quad (= L_{q_3}) \]

**Definition (Prime and Composed Residuals)**

Let \( L \subseteq \Sigma^* \) be a language. A residual \( L' \in Res(L) \) is called **composed** if there are \( L_1, \ldots, L_n \in Res(L) \setminus \{L'\} \) such that

\[ L' = L_1 \cup \ldots \cup L_n \]

Otherwise, it is called **prime**.

The set of prime residuals of \( L \) is denoted by \( Primes(L) \).
Towards canonical RFSA

\[ \varepsilon^{-1}L = \Sigma^*a\Sigma \quad (= L_{q_0}) \]
\[ (a)^{-1}L = \Sigma^*a\Sigma \cup \Sigma \quad (= L_{q_1}) \]
\[ (b)^{-1}L = \Sigma^*a\Sigma \cup \Sigma \cup \{\varepsilon\} \quad (= L_{q_2}) \]
\[ (ab)^{-1}L = \Sigma^*a\Sigma \cup \{\varepsilon\} \quad (= L_{q_3}) \]

Definition (Prime and Composed Residuals)

Let \( L \subseteq \Sigma^* \) be a language. A residual \( L' \in Res(L) \) is called composed if there are \( L_1, \ldots, L_n \in Res(L) \setminus \{L'\} \) such that

\[ L' = L_1 \cup \ldots \cup L_n \]

Otherwise, it is called prime.

The set of prime residuals of \( L \) is denoted by \( Primes(L) \).
Towards canonical RFSA

Let \( L \subseteq \Sigma^* \) be a language. A residual \( L' \in Res(L) \) is called composed if there are \( L_1, \ldots, L_n \in Res(L) \setminus \{L'\} \) such that

\[
L' = L_1 \cup \ldots \cup L_n
\]

Otherwise, it is called prime.

The set of prime residuals of \( L \) is denoted by \( Primes(L) \).
Canonical RFSA

$\varepsilon^{-1} L = \Sigma^* a \Sigma$ \hspace{1cm} ($= L_{q_0}$)

$(a)^{-1} L = \Sigma^* a \Sigma \cup \Sigma$ \hspace{1cm} ($= L_{q_1}$)

$(b)^{-1} L = \Sigma^* a \Sigma \cup \Sigma \cup \{\varepsilon\}$ \hspace{1cm} ($= L_{q_2}$)

$(ab)^{-1} L = \Sigma^* a \Sigma \cup \{\varepsilon\}$ \hspace{1cm} ($= L_{q_3}$)

**Definition (Canonical RFSA [Denis, Lemay, Terlutte’02])**

Let $L$ be a regular language. The **canonical RFSA** of $L$, denoted by $R(L)$, is the tuple $(Q, Q_0, F, \delta)$ where

- $Q = \text{Primes}(L)$,
- $Q_0 = \{L' \in Q \mid L' \subseteq L\}$,
- $F = \{L' \in Q \mid \varepsilon \in L'\}$, and
- $\delta(L_1, a) = \{L_2 \in Q \mid L_2 \subseteq a^{-1}L_1\}$, for $a \in \Sigma$. 

Carsten Kern   Learning Communicating and Nondet. Automata
Example: Deriving the canonical RFSA for $L = \Sigma^* a\Sigma$

Residual languages for $L = \Sigma^* a\Sigma$

$L_{q_0} = \Sigma^* a\Sigma$

$L_{q_1} = \Sigma^* a\Sigma \cup \Sigma$

$L_{q_2} = \Sigma^* a\Sigma \cup \Sigma \cup \{\varepsilon\}$

$L_{q_3} = \Sigma^* a\Sigma \cup \{\varepsilon\}$
Example: Deriving the canonical RFSA for $L = \Sigma^* a\Sigma$

States: $Q = \text{Primes}(L)$

Residual languages for $L = \Sigma^* a\Sigma$

- $L_{q_0} = \Sigma^* a\Sigma$
- $L_{q_1} = \Sigma^* a\Sigma \cup \Sigma$
- $L_{q_2} = \Sigma^* a\Sigma \cup \Sigma \cup \{\varepsilon\}$
- $L_{q_3} = \Sigma^* a\Sigma \cup \{\varepsilon\}$
Example: Deriving the canonical RFSA for $L = \Sigma^* a \Sigma$

States: $Q = \text{Primes}(L)$

Residual languages for $L = \Sigma^* a \Sigma$

$L_{q_0} = \Sigma^* a \Sigma$
$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$
$L_{q_2} = \Sigma^* a \Sigma \cup \Sigma \cup \{\varepsilon\}$
$L_{q_3} = \Sigma^* a \Sigma \cup \{\varepsilon\}$
Example: Deriving the canonical RFSA for \( L = \Sigma^* a \Sigma \)

Initial states: \( Q_0 = \{ L' \in Q \mid L' \subseteq L \} \),

Residual languages for \( L = \Sigma^* a \Sigma \)

\[
\begin{align*}
L_{q_0} &= \Sigma^* a \Sigma \quad \text{(initial state)} \\
L_{q_1} &= \Sigma^* a \Sigma \cup \Sigma \\
L_{q_2} &= \Sigma^* a \Sigma \cup \Sigma \cup \{\varepsilon\} \\
L_{q_3} &= \Sigma^* a \Sigma \cup \{\varepsilon\}
\end{align*}
\]
Example: Deriving the canonical RFSA for $L = \Sigma^* a \Sigma$

Initial states: $Q_0 = \{L' \in Q \mid L' \subseteq L\}$,

Residual languages for $L = \Sigma^* a \Sigma$

- $L_{q_0} = \Sigma^* a \Sigma$ (initial state)
- $L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$
- $L_{q_2} = \Sigma^* a \Sigma \cup \Sigma \cup \{\varepsilon\}$
- $L_{q_3} = \Sigma^* a \Sigma \cup \{\varepsilon\}$
Example: Deriving the canonical RFSA for $L = \Sigma^* a\Sigma$

Final states: $F = \{ L' \in Q \mid \epsilon \in L' \}$, and

Residual languages for $L = \Sigma^* a\Sigma$

$L_{q_0} = \Sigma^* a\Sigma$

$L_{q_1} = \Sigma^* a\Sigma \cup \Sigma$

$L_{q_2} = \Sigma^* a\Sigma \cup \Sigma \cup \{ \epsilon \}$

$L_{q_3} = \Sigma^* a\Sigma \cup \{ \epsilon \}$ (final state)
Example: Deriving the canonical RFSA for $L = \Sigma^* a \Sigma$

Final states: $F = \{ L' \in Q \mid \varepsilon \in L' \}$, and

Residual languages for $L = \Sigma^* a \Sigma$

$$L_{q_0} = \Sigma^* a \Sigma$$
$$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$$
$$L_{q_2} = \Sigma^* a \Sigma \cup \Sigma \cup \{\varepsilon\}$$
$$L_{q_3} = \Sigma^* a \Sigma \cup \{\varepsilon\} \quad \text{(final state)}$$
Example: Deriving the canonical RFSA for $L = \Sigma^* a\Sigma$

Transitions: $\delta(L_1, a) = \{L_2 \in Q \mid L_2 \subseteq a^{-1}L_1\}$, for $a \in \Sigma$.

Residual languages for $L = \Sigma^* a\Sigma$

- $L_{q_0} = \Sigma^* a\Sigma$ \quad \text{(a-transitions)}
- $L_{q_1} = \Sigma^* a\Sigma \cup \Sigma$
- $L_{q_2} = \Sigma^* a\Sigma \cup \Sigma \cup \{\varepsilon\}$
- $L_{q_3} = \Sigma^* a\Sigma \cup \{\varepsilon\}$
Example: Deriving the canonical RFSA for $L = \Sigma^* a\Sigma$

Transitions: $\delta(L_1, a) = \{L_2 \in Q \mid L_2 \subseteq a^{-1}L_1\}$, for $a \in \Sigma$.

Residual languages for $L = \Sigma^* a\Sigma$

- $L_{q_0} = \Sigma^* a\Sigma$ \hspace{1cm} (a-transitions)
- $L_{q_1} = \Sigma^* a\Sigma \cup \Sigma$
- $L_{q_2} = \Sigma^* a\Sigma \cup \Sigma \cup \{\varepsilon\}$
- $L_{q_3} = \Sigma^* a\Sigma \cup \{\varepsilon\}$
Example: Deriving the canonical RFSA for $L = \Sigma^*a\Sigma$

Transitions: $\delta(L_1, a) = \{L_2 \in Q \mid L_2 \subseteq a^{-1}L_1\}$, for $a \in \Sigma$.

Residual languages for $L = \Sigma^*a\Sigma$

- $L_{q_0} = \Sigma^*a\Sigma$ (a-transitions)
- $L_{q_1} = \Sigma^*a\Sigma \cup \Sigma$
- $L_{q_2} = \Sigma^*a\Sigma \cup \Sigma \cup \{\varepsilon\}$
- $L_{q_3} = \Sigma^*a\Sigma \cup \{\varepsilon\}$
Example: Deriving the canonical RFSA for $L = \Sigma^* a\Sigma$

Transitions: $\delta(L_1, a) = \{L_2 \in Q \mid L_2 \subseteq a^{-1}L_1\}$, for $a \in \Sigma$.

Residual languages for $L = \Sigma^* a\Sigma$

- $L_{q_0} = \Sigma^* a\Sigma$
- $L_{q_1} = \Sigma^* a\Sigma \cup \Sigma$
- $L_{q_2} = \Sigma^* a\Sigma \cup \Sigma \cup \{\varepsilon\}$
- $L_{q_3} = \Sigma^* a\Sigma \cup \{\varepsilon\}$
Let $T = (T, U, V)$ be a table. Find analogon to union of residuals

**Definition (Join Operator)**

join of two rows $r_1, r_2 \in Rows(T)$ is defined component-wise for each $v \in V$:

$$(r_1 \sqcup r_2) : V \rightarrow \{+,-\} :$$

$(r_1 \sqcup r_2)(v) = r_1(v) \sqcup r_2(v)$ where

- $- \sqcup - = -$ and
- $+ \sqcup + = + \sqcup - = - \sqcup + = +$

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aba$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$abb$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
</tbody>
</table>
Let $\mathcal{T} = (T, U, V)$ be a table. Find analogon to union of residuals.

**Definition (Join Operator)**

join of two rows $r_1, r_2 \in Rows(\mathcal{T})$ is defined component-wise for each $v \in V$:

$$(r_1 \sqcup r_2) : V \rightarrow \{+, -\}:$$

$$(r_1 \sqcup r_2)(v) = r_1(v) \sqcup r_2(v) \quad \text{where}$$

- $- \sqcup - = -$ and
- $+ \sqcup + = + \sqcup - = - \sqcup + = +$

<table>
<thead>
<tr>
<th>$\mathcal{T}$</th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aba$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$abb$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
</tbody>
</table>

**Example**

$\text{row}(a) \sqcup \text{row}(ab) = (-, +, +) \sqcup (+, -, +) = (+, +, +) = \text{row}(aa)$
Designing a table-based learning algorithm

Find analogon to *Composed and prime residuals*

**Definition (Composed and Prime Rows)**

Row \( r \in \text{Rows}(\mathcal{T}) \) is called:

- composed if there are rows \( r_1, \ldots, r_n \in \text{Rows}(\mathcal{T}) \setminus \{r\} \) such that \( r = r_1 \sqcup \ldots \sqcup r_n \).

\[
\begin{array}{c|ccc}
\mathcal{T} & \varepsilon & a & aa \\
\hline
\varepsilon & - & - & + \\
a & - & + & + \\
ab & + & - & + \\
b & - & - & + \\
aa & + & + & + \\
aba & - & + & + \\
abb & - & - & + \\
\end{array}
\]
Designing a table-based learning algorithm

Find analogon to *Composed and prime residuals*

**Definition (Composed and Prime Rows)**

Row \( r \in \text{Rows}(\mathcal{T}) \) is called:

- **composed** if there are rows \( r_1, \ldots, r_n \in \text{Rows}(\mathcal{T}) \setminus \{r\} \) such that \( r = r_1 \sqcup \ldots \sqcup r_n \).

<table>
<thead>
<tr>
<th>( \mathcal{T} )</th>
<th>( \varepsilon )</th>
<th>( a )</th>
<th>( aa )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varepsilon )</td>
<td>(-)</td>
<td>(-)</td>
<td>(+)</td>
</tr>
<tr>
<td>( a )</td>
<td>(-)</td>
<td>(+)</td>
<td>(+)</td>
</tr>
<tr>
<td>( ab )</td>
<td>(+)</td>
<td>(-)</td>
<td>(+)</td>
</tr>
<tr>
<td>( b )</td>
<td>(-)</td>
<td>(-)</td>
<td>(+)</td>
</tr>
<tr>
<td>( aa )</td>
<td>(+)</td>
<td>(+)</td>
<td>(+)</td>
</tr>
<tr>
<td>( aba )</td>
<td>(-)</td>
<td>(+)</td>
<td>(+)</td>
</tr>
<tr>
<td>( abb )</td>
<td>(-)</td>
<td>(-)</td>
<td>(+)</td>
</tr>
</tbody>
</table>

**Example**

Row \( (+, +, +) \) is composed:

\[
\text{row}(aa) = (+, +, +) = (-, +, +) \sqcup (+, -, +) = \text{row}(a) \sqcup \text{row}(ab)
\]
Find analogon to \textit{Composed and prime residuals}

\textbf{Definition (Composed and Prime Rows)}

Row $r \in \text{Rows}(\mathcal{T})$ is called:

\begin{itemize}
  \item \textbf{composed} if there are rows $r_1, \ldots, r_n \in \text{Rows}(\mathcal{T}) \setminus \{r\}$ such that $r = r_1 \sqcup \ldots \sqcup r_n$.
  \item \textbf{prime}, otherwise.
\end{itemize}

\begin{tabular}{|c|c|c|c|}
  \hline
  $\mathcal{T}$ & $\varepsilon$ & $a$ & $aa$ \\
  \hline
  $\varepsilon$ & $-$ & $-$ & $+$ \\
  $a$ & $-$ & $+$ & $+$ \\
  $ab$ & $+$ & $-$ & $+$ \\
  $b$ & $-$ & $-$ & $+$ \\
  $aa$ & $+$ & $+$ & $+$ \\
  $aba$ & $-$ & $+$ & $+$ \\
  $abb$ & $-$ & $-$ & $+$ \\
  \hline
\end{tabular}
Designing a table-based learning algorithm

Find analogon to Composed and prime residuals

**Definition (Composed and Prime Rows)**

Row $r \in \text{Rows}(T)$ is called:

- **composed** if there are rows $r_1, \ldots, r_n \in \text{Rows}(T) \setminus \{r\}$ such that $r = r_1 \sqcup \ldots \sqcup r_n$.
- **prime**, otherwise.

\[
\begin{array}{|c|c|c|c|}
\hline
T & \varepsilon & a & aa \\
\hline
\varepsilon & - & - & + \\
\hline
a & - & + & + \\
\hline
ab & + & - & + \\
\hline
b & - & - & + \\
\hline
aa & + & + & + \\
\hline
aba & - & + & + \\
\hline
abb & - & - & + \\
\hline
\end{array}
\]

**Example**

E.g. rows $(-, -, +)$, $(-, +, +)$ are prime
Designing a table-based learning algorithm

Find analogon to *Composed and prime residuals*

**Definition (Composed and Prime Rows)**

Row $r \in \text{Rows}(\mathcal{T})$ is called:

- **composed** if there are rows $r_1, \ldots, r_n \in \text{Rows}(\mathcal{T}) \setminus \{r\}$ such that $r = r_1 \sqcup \ldots \sqcup r_n$.
- **prime**, otherwise.

$\text{Primes}(\mathcal{T})$: The set of prime rows in $\mathcal{T}$ and

<table>
<thead>
<tr>
<th>$\mathcal{T}$</th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aba$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$abb$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
</tbody>
</table>

**Example**

$\text{Primes}(\mathcal{T}) = \{ \text{row}(\varepsilon), \text{row}(a), \text{row}(ab), \text{row}(b), \text{row}(aba), \text{row}(abb) \}$
Designing a table-based learning algorithm

Find analogon to *Composed and prime residuals*

**Definition (Composed and Prime Rows)**

Row \( r \in \text{Rows}(T) \) is called:
- **composed** if there are rows \( r_1, \ldots, r_n \in \text{Rows}(T) \setminus \{r\} \) such that \( r = r_1 \sqcup \ldots \sqcup r_n \).
- **prime**, otherwise.

\( \text{Primes}(T) \): The set of prime rows in \( T \) and

\( \text{Primes}_{\text{upp}}(T) = \text{Primes}(T) \cap \text{Rows}_{\text{upp}}(T) \).

**Example**

\( \text{Primes}_{\text{upp}}(T) = \{ \text{row}(\varepsilon), \text{row}(a), \text{row}(ab) \} \)
Find analogon to subset relation between residuals

**Definition (Covering Relation)**

Row \( r \in \text{Rows}(T) \) is:
- covered by row \( r' \in \text{Rows}(T) \) \( (r \sqsubseteq r') \), if for all \( v \in V \): \( r(v) = + \Rightarrow r'(v) = + \).

**Example**
- e.g., \( \text{row}(\varepsilon) \sqsubseteq \text{row}(a) \) and \( \text{row}(\varepsilon) \sqsubseteq \text{row}(abb) \)
Designing a table-based learning algorithm

Find analogon to subset relation between residuals

**Definition (Covering Relation)**

Row $r \in \text{Rows}(\mathcal{T})$ is:

- covered by row $r' \in \text{Rows}(\mathcal{T})$ ($r \sqsubseteq r'$), if for all $v \in V$: $r(v) = + \Rightarrow r'(v) = +$.

<table>
<thead>
<tr>
<th>$\mathcal{T}$</th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aba$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$abb$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
</tbody>
</table>

**Example**

- e.g., $\text{row}(\varepsilon) \sqsubseteq \text{row}(a)$ and $\text{row}(\varepsilon) \sqsubseteq \text{row}(abb)$
Designing a table-based learning algorithm

Find analogon to subset relation between residuals

**Definition (Covering Relation)**

Row \( r \in \text{Rows}(\mathcal{T}) \) is:

- covered by row \( r' \in \text{Rows}(\mathcal{T}) \) (\( r \sqsubseteq r' \)), if for all \( v \in V \): \( r(v) = + \Rightarrow r'(v) = + \).
- If moreover \( r' \neq r \), then \( r \) is strictly covered by \( r' \), denoted by \( r \sqsubset r' \).

<table>
<thead>
<tr>
<th>( \mathcal{T} )</th>
<th>( \varepsilon )</th>
<th>( a )</th>
<th>( aa )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varepsilon )</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>( a )</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( ab )</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>( b )</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>( aa )</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( aba )</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( abb )</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

**Example**

- e.g., \( \text{row}(\varepsilon) \sqsubseteq \text{row}(a) \) and \( \text{row}(\varepsilon) \sqsubseteq \text{row}(abb) \)
- e.g., \( \text{row}(\varepsilon) \sqsubset \text{row}(ab) \)
Find analogon to closedness and consistency in $L^*$

**RFSA-Closedness**

- all states identifiable from the table

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aba$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$abb$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
</tbody>
</table>
### RFSA-Closedness

- All states identifiable from the table.
- All \textit{non-composed} rows have to be in the upper part of the table.
Find analogon to *closedness and consistency* in $L^*$

### RFSA-Closedness

- all states identifiable from the table
- all *non-composed* rows have to be in the upper part of the table
- all other rows can be composed by upper rows

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>$a$</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>$ab$</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>$b$</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>$aa$</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>$aba$</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>$abb$</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>
Table properties

Find analogon to closedness and consistency in \( L^* \)

<table>
<thead>
<tr>
<th>( T )</th>
<th>( \varepsilon )</th>
<th>( a )</th>
<th>( aa )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varepsilon )</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>( a )</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( ab )</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>( b )</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>( aa )</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( aba )</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( abb )</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

**RFSA-Closedness**
- all states identifiable from the table
- all *non-composed* rows have to be in the upper part of the table
- all other rows can be composed by upper rows

**RFSA-Consistency**
- transition relation respects language inclusion
Find analogon to closedness and consistency in $L^*$

**RFSA-Closedness**
- all states identifiable from the table
- all *non-composed* rows have to be in the upper part of the table
- all other rows can be composed by upper rows

**RFSA-Consistency**
- transition relation respects language inclusion
Table properties

Find analogon to closedness and consistency in $L^*$

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aba$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$abb$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
</tbody>
</table>

RFSA-Closedness

- all states identifiable from the table
- all non-composed rows have to be in the upper part of the table
- all other rows can be composed by upper rows

RFSA-Consistency

- transition relation respects language inclusion
From Table to NFA

Definition (NFA of a Table)

For a table $T = (T, U, V)$ that is RFSA-closed and RFSA-consistent, we define an NFA $R_T = (Q, Q_0, F, \delta)$ by

- $Q = Primes_{up}(T)$,
- $Q_0 = \{ r \in Q \mid r \subseteq \text{row}(\varepsilon) \}$,
- $F = \{ r \in Q \mid r(\varepsilon) = + \}$, and
- $\delta(\text{row}(u), a) = \{ r \in Q \mid r \subseteq \text{row}(ua) \}$ ($u \in U$, $\text{row}(u) \in Q$, $a \in \Sigma$).

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$*$</td>
<td>$\varepsilon$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$*$</td>
<td>$a$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$*$</td>
<td>$ab$</td>
<td>$+$</td>
<td>$-$</td>
</tr>
<tr>
<td>$*$</td>
<td>$b$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td></td>
<td>$aa$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$*$</td>
<td>$aba$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$*$</td>
<td>$abb$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
</tbody>
</table>
From Table to NFA

**Definition (NFA of a Table)**

For a table $\mathcal{T} = (T, U, V)$ that is RFSA-closed and RFSA-consistent, we define an NFA $\mathcal{R}_T = (Q, Q_0, F, \delta)$ by

- $Q = Primes_{upp}(T)$,
- $Q_0 = \{ r \in Q \mid r \subseteq row(\varepsilon) \}$,
- $F = \{ r \in Q \mid r(\varepsilon) = + \}$, and
- $\delta(row(u), a) = \{ r \in Q \mid r \subseteq row(ua) \}$ (u ∈ U, row(u) ∈ Q, a ∈ Σ).
Definition (NFA of a Table)

For a table $\mathcal{T} = (T, U, V)$ that is RFSA-closed and RFSA-consistent, we define an NFA $\mathcal{R}_\mathcal{T} = (Q, Q_0, F, \delta)$ by

- $Q = Primes_{upp}(T)$,
- $Q_0 = \{ r \in Q \mid r \subseteq \text{row}(\varepsilon) \}$,
- $F = \{ r \in Q \mid r(\varepsilon) = + \}$, and
- $\delta(\text{row}(u), a) = \{ r \in Q \mid r \subseteq \text{row}(ua) \}$ ($u \in U$, $\text{row}(u) \in Q$, $a \in \Sigma$).

<table>
<thead>
<tr>
<th>$\mathcal{T}$</th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$*$</td>
<td>$\varepsilon$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$*$</td>
<td>$a$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$*$</td>
<td>$ab$</td>
<td>$+$</td>
<td>$-$</td>
</tr>
<tr>
<td>$*$</td>
<td>$b$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$*$</td>
<td>$aa$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$*$</td>
<td>$aba$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$*$</td>
<td>$abb$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
</tbody>
</table>

Diagram:

- $L_{q_0}$
- $L_{q_1}$
- $L_{q_3}$
From Table to NFA

Definition (NFA of a Table)

For a table \( \mathcal{T} = (T, U, V) \) that is RFSA-closed and RFSA-consistent, we define an NFA \( \mathcal{R}_\mathcal{T} = (Q, Q_0, F, \delta) \) by

- \( Q = \text{Primes}_{\text{upp}}(\mathcal{T}) \),
- \( Q_0 = \{ r \in Q \mid r \sqsubseteq \text{row}(\varepsilon) \} \),
- \( F = \{ r \in Q \mid r(\varepsilon) = + \} \), and
- \( \delta(\text{row}(u), a) = \{ r \in Q \mid r \sqsubseteq \text{row}(ua) \} \) \((u \in U, \text{row}(u) \in Q, a \in \Sigma)\).

\[
\begin{array}{c|c|c|c}
T & \varepsilon & a & aa \\
\hline
* & \varepsilon & - & - & + \\
* & a & - & + & + \\
* & ab & + & - & + \\
* & b & - & - & + \\
* & aa & + & + & + \\
* & aba & - & + & + \\
* & abb & - & - & + \\
\end{array}
\]
From Table to NFA

Definition (NFA of a Table)

For a table \( \mathcal{T} = (T, U, V) \) that is RFSA-closed and RFSA-consistent, we define an NFA \( \mathcal{R}_T = (Q, Q_0, F, \delta) \) by

- \( Q = \text{Primes}_{\text{upp}}(T) \),
- \( Q_0 = \{ r \in Q \mid r \subseteq \text{row}(\varepsilon) \} \),
- \( F = \{ r \in Q \mid r(\varepsilon) = + \} \), and
- \( \delta(\text{row}(u), a) = \{ r \in Q \mid r \subseteq \text{row}(ua) \} \) (\( u \in U \), \( \text{row}(u) \in Q \), \( a \in \Sigma \)).
For a table $\mathcal{T} = (T, U, V)$ that is RFSA-closed and RFSA-consistent, we define an NFA $\mathcal{R}_{\mathcal{T}} = (Q, Q_0, F, \delta)$ by

1. $Q = \text{Primes}_{\text{up}}(\mathcal{T})$,
2. $Q_0 = \{ r \in Q \mid r \sqsubseteq \text{row}(\varepsilon) \}$,
3. $F = \{ r \in Q \mid r(\varepsilon) = + \}$, and
4. $\delta(\text{row}(u), a) = \{ r \in Q \mid r \sqsubseteq \text{row}(ua) \}$ ($u \in U$, $\text{row}(u) \in Q$, $a \in \Sigma$).

\[ T \mid \varepsilon \mid a \mid aa \]
\[
\begin{array}{c|ccc}
* & \varepsilon & - & - & + \\
* & a & - & + & + \\
* & ab & + & - & + \\
* & b & - & - & + \\
* & aa & + & + & + \\
* & aba & - & + & + \\
* & abb & - & - & + \\
\end{array}
\]
## Summarizing: Tables in NL*

### From tables to RFSA

- we deal with tables

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aba$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$abb$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
</tbody>
</table>
### Summarizing: Tables in NL*

#### From tables to RFSA
- we deal with tables
- table rows approximate residual languages

<table>
<thead>
<tr>
<th>( \mathcal{T} )</th>
<th>( \varepsilon )</th>
<th>( a )</th>
<th>( aa )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varepsilon )</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>( a )</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( ab )</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>( b )</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>( aa )</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( aba )</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( abb )</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>
Summarizing: Tables in NL*

From tables to RFSA

- we deal with tables
- table rows approximate residual languages
- not all rows represent states
From tables to RFSA

- we deal with tables
- table rows approximate residual languages
- not all rows represent states
- as long as there is no other evidence: equal rows represent equal residual languages

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>$a$</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>$ab$</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>$b$</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>$aa$</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>$aba$</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>$abb$</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>
### Summarizing: Tables in NL*

#### From tables to RFSA

- we deal with tables
- table rows approximate residual languages
- not all rows represent states
- as long as there is no other evidence: equal rows represent equal residual languages
- **transition relation respects language inclusion**

<table>
<thead>
<tr>
<th>$\mathcal{T}$</th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$- \ - \ +$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>$- \ + \ +$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$ab$</td>
<td>$+ \ - \ +$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b$</td>
<td>$- \ - \ +$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$aa$</td>
<td>$+ \ + \ +$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$aba$</td>
<td>$- \ + \ +$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$abb$</td>
<td>$- \ - \ +$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Summarizing: Tables in NL*

<table>
<thead>
<tr>
<th>$\mathcal{T}$</th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aba$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$abb$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
</tbody>
</table>

From tables to RFSA

- we deal with tables
- table rows approximate residual languages
- not all rows represent states
- as long as there is no other evidence: equal rows represent equal residual languages
- transition relation respects language inclusion
### From tables to RFSA

- we deal with tables
- table rows approximate residual languages
- not all rows represent states
- as long as there is no other evidence: equal rows represent equal residual languages
- **transition relation respects language inclusion**
Summarizing: Tables in NL*

From tables to RFSA
- we deal with tables
- table rows approximate residual languages
- not all rows represent states
- as long as there is no other evidence: equal rows represent equal residual languages
- transition relation respects language inclusion
- treatment of counterexamples:
  - add to columns (as in $L_{col}^*$)
  - otherwise non-termination
Towards correctness

Definition (Consistency with a table)

We say that $R_T$ is consistent with the table $T$ if, for all $w \in (U \cup U\Sigma)V$, we have $T(w) = +$ iff $w \in L(R_T)$. 
Towards correctness

**Definition (Consistency with a table)**

We say that $\mathcal{R}_T$ is consistent with the table $T$ if, for all $w \in (U \cup U\Sigma)V$, we have $T(w) = +$ iff $w \in L(\mathcal{R}_T)$.

**Theorem (Correctness)**

Let $T$ be a table that is RFSA-closed and RFSA-consistent and let $\mathcal{R}_T$ be consistent with $T$. Then, $\mathcal{R}_T$ is a canonical RFSA.
Complexity issues

Theorem (Complexity of \(NL^*\))

Let:

- \(n\): number of states of minimal DFA \(A_L\) for regular language \(L\),
- \(m\): length of the biggest counterexample

Then, \(NL^*\) returns after at most:

the canonical RFSA \(\mathcal{R}(L)\).
Complexity issues

Theorem (Complexity of NL*)

Let:
- $n$: number of states of minimal DFA $A_L$ for regular language $L$,
- $m$: length of the biggest counterexample

Then, NL* returns after at most:
- $O(n^2)$ equivalence queries and

the canonical RFSA $R(L)$. 
Complexity issues

Theorem (Complexity of NL*)

Let:
- \( n \): number of states of minimal DFA \( A_L \) for regular language \( L \),
- \( m \): length of the biggest counterexample

Then, NL* returns after at most:
- \( O(n^2) \) equivalence queries and
- \( O(m|\Sigma|n^3) \) membership queries

the canonical RFSA \( R(L) \).
It’s worth considering RFSA...

Theorem

There is an infinite family of languages \( \{L_n\}_{n \in \mathbb{N}} \) for which NL* infers canonical RFSA that are exponentially more succinct than their corresponding minimal DFA.
It’s worth considering RFSA...

$L_n = \{ w \in \Sigma^* | w \text{ has an } a \text{ at the } (n+1)\text{-last position} \}$
It’s worth considering RFSA...

\[ L_n = \{ w \in \Sigma^* | w \text{ has an } a \text{ at the } (n + 1)\text{-last position} \} \]

\[ L_2 = \{ w \in \Sigma^* | w \text{ has an } a \text{ at the } 3^{rd}\text{-last position} \} \]
Minimal DFA and RFSA

Minimal DFA and RFSA for $L_2$:

Automata for language $L_n$:

- minimal DFA general case: $2^{n+1}$ states
Minimal DFA and RFSA

Minimal DFA and RFSA for $L_2$:

Automata for language $L_n$:

- minimal DFA general case: $2^{n+1}$ states
- canonical RFSA general case: $n + 2$ states
### Comparison of $L^*$, $L^*_\text{col}$, and $NL^*$

<table>
<thead>
<tr>
<th></th>
<th>Equivalence queries</th>
<th>Membership queries</th>
<th>Treatment of counterexamples</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L^*$</td>
<td>$n$</td>
<td>$O(m</td>
<td>\Sigma</td>
</tr>
<tr>
<td>$L^*_\text{col}$</td>
<td>$n$</td>
<td>$O(m</td>
<td>\Sigma</td>
</tr>
<tr>
<td>$NL^*$</td>
<td>$O(n^2)$</td>
<td>$O(m</td>
<td>\Sigma</td>
</tr>
</tbody>
</table>

*Theoretical complexity* for the number of queries is a bit worse than for learning DFA.
Algorithm - Overview

Number of states ($L^*$, $L^*_\text{col}$ vs. $NL^*$)

$\approx 3200$ reg. exp. with minimal DFA of 1 to 200 states
Algorithm - Overview

Number of membership queries ($L^*$ vs. $L^*_{col}$ vs. $NL^*$)

- $L^*$
- $L^*_{col}$
- $NL^*$

≈ 3200 reg. exp. with minimal DFA of 1 to 200 states
Algorithm - Overview

Number of equivalence queries ($L^*$ vs. $L^*_{\text{col}}$ vs. $NL^*$)

- $\approx 3200$ reg. exp. with minimal DFA of 1 to 200 states
Presentation outline

1. Learning Deterministic Automata
2. Learning Nondeterministic Automata
3. Learning Communicating Automata
4. Tools
5. Conclusion
Motivation

Requirements (incomplete)

- initial phase: requirement elicitation
  - contradicting or incomplete system description
  - common description language: sequence diagrams
Motivation

- initial phase: requirement elicitation
  - contradicting or incomplete system description
  - common description language: sequence diagrams
- goal: conforming design model
Motivation

- initial phase: requirement elicitation
  - contradicting or incomplete system description
  - common description language: sequence diagrams
- goal: conforming design model
initial phase: requirement elicitation
  - contradicting or incomplete system description
  - common description language: sequence diagrams

goal: conforming design model

closing gap between
  - requirement specification (usually incomplete) and
  - design model (complete description of system)
Our Approach

- use **learning algorithms** to synthesize models for communication protocols

**Input:** set of Message Sequence Charts
  - standardized: ITU Z.120
  - included in UML as sequence diagrams

**Output:** Communicating finite-state machine
  - distributed system fulfilling the specification
  - CFM model is close to implementation
An MSC $M = \langle \mathcal{P}, E, \{\leq_p\}_{p \in \mathcal{P}}, \langle_{\text{msg}}, l\rangle$

- $\mathcal{P}$: finite set of processes
- $E$: finite set of events ($E = \bigcup_{p \in \mathcal{P}} E_p$)
- $l : E \rightarrow \text{Act} = \{1!2(\text{req}), 1?2(\text{ack}), \ldots\}$
- for $p \in \mathcal{P}$: $\langle_p \subseteq E_p \times E_p$ is a total order on $E_p$
- $\langle_{\text{msg}}$ relates sending and receiving events
- $\leq = \left(\langle_{\text{msg}} \cup \bigcup_{p \in \mathcal{P}} \langle_p\right)^*$
An MSC \( M = \langle \mathcal{P}, E, \{\leq_p\}_{p \in \mathcal{P}}, \langle \text{msg}, l \rangle \rangle \)

- \( \mathcal{P} \): finite set of processes
- \( E \): finite set of events \( (E = \bigcup_{p \in \mathcal{P}} E_p) \)
- \( l : E \to \text{Act} = \{1!2(\text{req}), 1?2(\text{ack}), \ldots \} \)
- for \( p \in \mathcal{P} \): \( \langle p \rangle \subseteq E_p \times E_p \) is a total order on \( E_p \)
- \( \langle \text{msg} \rangle \) relates sending and receiving events
- \( \leq = \left( \langle \text{msg} \rangle \cup \bigcup_{p \in \mathcal{P}} \langle p \rangle \right)^* \)

A set of MSCs is called an MSC language

A linearization of an MSC is a total ordering of \( E \) subsuming \( \leq \)
MSCs and Linearizations

Some linearizations

- 1!2(req) 1!2(req) 2!1(ack) 1?2(ack) 2?1(req) 2?1(req)
- 1!2(req) 2!1(ack) 1!2(req) 1?2(ack) 2?1(req) 2?1(req)
- 2!1(ack) 1!2(req) 2?1(req) 1!2(req) 2?1(req) 1?2(ack)
- ...

Carsten Kern  Learning Communicating and Nondet. Automata
MSCs and Linearizations

Some linearizations

- $1!2(req)\ 1!2(req)\ 2!1(ack)\ 1?2(ack)\ 2?1(req)\ 2?1(req)$
- $1!2(req)\ 2!1(ack)\ 1!2(req)\ 1?2(ack)\ 2?1(req)\ 2?1(req)$
- $2!1(ack)\ 1!2(req)\ 2?1(req)\ 1!2(req)\ 2?1(req)\ 1?2(ack)$
- $\ldots$

- An MSC $M = MSC(w)$ is uniquely determined by any $w \in Lin(M)$
MSCs and Linearizations

Some linearizations

- $1!2(req)$ $1!2(req)$ $2!1(ack)$ $1?2(ack)$ $2?1(req)$ $2?1(req)$
- $1!2(req)$ $2!1(ack)$ $1!2(req)$ $1?2(ack)$ $2?1(req)$ $2?1(req)$
- $2!1(ack)$ $1!2(req)$ $2?1(req)$ $1!2(req)$ $2?1(req)$ $1?2(ack)$
- ...

- An MSC $M = MSC(w)$ is uniquely determined by any $w \in Lin(M)$
- Linearizations of an MSC are called equivalent
  $(\forall w, w' \in Lin(M) : w \approx w')$
A CFM consists of:

- a set of finite-state automata (*processes*) with
  - common global initial state
  - set of global final states
Communicating Finite-State Machines (CFM)

A CFM consists of:

- a set of finite-state automata (processes) with
  - common global initial state
  - set of global final states
- communication between automata through (reliable) FIFO channels
  - $p!q(a)$ appends message $a$ to buffer between $p$ and $q$
  - $q?p(a)$ removes message $a$ from buffer between $p$ and $q
CFM: An Example
CFM: An Example

![Diagram of a CFM example]

- Buffer head
- Transition: 1 → 2
- Transition: 2 → 1
CFM: An Example

![Diagram of CFM example](image)

- Transition from state 1 to state 2
- Transition from state 2 to state 1
- Buffer head
CFM: An Example

![Diagram of CFM automata with states and transitions]

Buffer head

1 → 2

2 → 1
CFM: An Example

![Diagram of CFM example]

- **Buffer head**
- **Transition rules**
  - $0 \rightarrow 1$
  - $a \rightarrow 2$

Carsten Kern  Learning Communicating and Nondet. Automata
CFM: An Example

buffer head

1 → 2

2 → 1

Carsten Kern Learning Communicating and Nondet. Automata
CFM: An Example

```
0 1 |   |
    1 → 2
buffer head

2 → 1
```
CFM: An Example

![Diagram of a CFM example](image)

1 → 2

buffer head

2 → 1
CFM: An Example

Diagram with two states:
- State 1: ?a → 1
- State 2: !a → 2

Transition arrows:
- 1 → 2 (buffer head)
- 2 → 1

Input/output symbols:
- !0, ?1, !1, ?0

Diagram showing input/output actions:
- !0, ?a, !1

Graphical representation:
- Nodes: State 1 and State 2
- Edges: Arrows indicating transitions and input/output actions
CFM: An Example

![Diagram](image)

**Buffer Head**

1 → 2

2 → 1

---

Carsten Kern  Learning Communicating and Nondet. Automata
CFM: An Example

![Diagram of a CFM example with states and transitions involving inputs and outputs.]

1 → 2

buffer head

2 → 1
Current State

- **given:** learning DFA [Angluin]
- **goal:** learning CFMs
The learning algorithm (extension of Angluin’s $L^*$)

- **Teacher**
  - $\text{MSC} \in \text{System}$
  - Yes/No

- **Learner**
  - Yes/Counterexample

- **Oracle**
  - $\mathcal{H} \equiv \text{System}$
  - Equivalence queries

- **Membership queries**
The learning algorithm (extension of Angluin’s $L^*$)

Learner

Teacher

Oracle

Membership queries

Equivalence queries

computer

user
The learning algorithm (extension of Angluin’s $L^*$)

Diagram:
- **Learner**
- **Teacher**
- **Oracle**

Actions:
- **Membership queries**
- **Equivalence queries**

Note: $\in$ System
The learning algorithm (extension of Angluin’s $L^*$)
The learning algorithm (extension of Angluin’s $L^*$)

- **Learner**
- **Teacher**
- **Oracle**

Membership queries

Equivalence queries

- computer
- user

$? \notin \text{System}$
The learning algorithm (extension of Angluin’s $L^*$)
The learning algorithm (extension of Angluin’s $L^*$)

Learner

Teacher

Oracle

Membership queries

Equivalence queries

$\mathcal{H} \equiv \text{System}$

$\mathcal{H}$

$A_1$

$A_2$

$1\!\!2\text{(req)}$

$2\!\!1\text{(req)}$

$1\!\!2\text{(ack)}$

$2\!\!1\text{(ack)}$

$2\!\!1\text{(req)}$

$1\!\!2\text{(req)}$

$\text{computer}$

$\text{user}$
The learning algorithm (extension of Angluin’s $L^*$)

- Learner
- Teacher
- Yes/Counterexample
- Oracle
- Membership queries
- Equivalence queries

1. computer
2. user

Carsten Kern   Learning Communicating and Nondet. Automata
The learning algorithm (extension of Angluin’s $L^*$)

- **Learner**
- **Teacher**
- **Oracle**

**Membership queries**
- \(? \in \text{System}\)
- Yes/No

**Equivalence queries**
- \(\mathcal{H} \equiv \text{System}\)

**Diagram:**
- Nodes labeled with
  - 1
  - 2
- Arrows indicating transitions and queries

**Legend:**
- Computer
- User

**System:**
- \(\mathcal{H} : A_1, A_2\)
- \(1\text{?}\text{req}, 1\text{?}\text{ack}, 2\text{?}\text{req}, 2\text{?}\text{req}, 2\text{?}\text{ack}\)

**Carsten Kern**
Learning Communicating and Nondet. Automata
Goal

- learning CFMs from examples (MSCs)
Goal

- learning CFMs from examples (MSCs)

Approach

- extending Angluin’s algorithm
- **Input:** linearizations of MSCs
  - positive scenarios are included in the language to learn
  - negative scenarios must not be contained
- positive and negative scenarios form system behavior
Goal
- learning CFMs from examples (MSCs)

Approach
- extending Angluin’s algorithm
- Input: linearizations of MSCs
  - positive scenarios are included in the language to learn
  - negative scenarios must not be contained
- positive and negative scenarios form system behavior

Problem
- correspondence between CFMs and regular word languages needed
Classes of MSCs

\( M \) is \( \forall B \)-bounded \((B \in \mathbb{N})\) if

all linearizations of \( M \) do not exceed buffer bound \( B \)

\( M \) is \( \exists B \)-bounded \((B \in \mathbb{N})\) if

events of \( M \) can be scheduled s.t. \( B \) is not exceeded

Fix a learning setup

- \( D \) domain over \((\forall/\exists B\)-bounded\) MSC linearizations
- \( \approx \) equivalence of \((\forall/\exists B\)-bounded\) linearizations
- \( \text{synth} \) : Synthesis function from DFA to \((\forall/\exists – B\)-bounded\) CFMs
User specification: final system should be, e.g.,

- deterministic, $\exists/\forall$-bounded (i.e., fix domain $\mathcal{D}$), deadlockfree etc.
From regular languages to CFM languages

User specification: final system should be, e.g., . . .
- deterministic, $\exists/\forall$-bounded (i.e., fix domain $\mathcal{D}$), deadlock-free etc.

Learning procedure (excerpt): a guided approach
- membership queries for equivalent words need to be answered equivalently (all-or-none law)
From regular languages to CFM languages

User specification: final system should be, e.g.,... 
- deterministic, \( \exists/\forall \)-bounded (i.e., fix domain \( \mathcal{D} \)), deadlockfree etc.

Learning procedure (excerpt): a guided approach
- membership queries for equivalent words need to be answered equivalently (\textit{all-or-none law})
- having found a hypothesis DFA \( \mathcal{H} \)
From regular languages to CFM languages

User specification: final system should be, e.g.,...

- deterministic, \( \exists / \forall \)-bounded (i.e., fix domain \( D \)), deadlockfree etc.

Learning procedure (excerpt): a guided approach

- membership queries for equivalent words need to be answered equivalently (all-or-none law)
- having found a hypothesis DFA \( \mathcal{H} \)
  - if \( L(\mathcal{H}) \not\subseteq D \), compute counterexample \( w \in L(\mathcal{H}) \setminus D \)
User specification: final system should be, e.g.,
- deterministic, $\exists/\forall$-bounded (i.e., fix domain $\mathcal{D}$), deadlockfree etc.

Learning procedure (excerpt): a guided approach
- membership queries for equivalent words need to be answered equivalently (*all-or-none* law)
- having found a hypothesis DFA $\mathcal{H}$
  1. if $L(\mathcal{H}) \not\subseteq \mathcal{D}$, compute counterexample $w \in L(\mathcal{H}) \setminus \mathcal{D}$
  2. else if $L(\mathcal{H}) \subseteq \mathcal{D}$ but $L(\mathcal{H})$ not $\approx$-closed
      - compute $w \approx w'$: $w \in L(\mathcal{H})$, $w' \notin L(\mathcal{H})$ and
      - perform membership query for MSC($w$)
From regular languages to CFM languages

User specification: final system should be, e.g.,

- deterministic, \( \exists/\forall \)-bounded (i.e., fix domain \( \mathcal{D} \)), deadlockfree etc.

Learning procedure (excerpt): a guided approach

- membership queries for equivalent words need to be answered equivalently (all-or-none law)
- having found a hypothesis DFA \( \mathcal{H} \)
  1. if \( L(\mathcal{H}) \not\subseteq \mathcal{D} \), compute counterexample \( w \in L(\mathcal{H}) \setminus \mathcal{D} \)
  2. else if \( L(\mathcal{H}) \subseteq \mathcal{D} \) but \( L(\mathcal{H}) \) not \( \approx \)-closed
     - compute \( w \approx w' \): \( w \in L(\mathcal{H}) \), \( w' \not\in L(\mathcal{H}) \) and
     - perform membership query for \( \text{MSC}(w) \)

If \( \mathcal{H} \) satisfies \( L(\mathcal{H}) \subseteq \mathcal{D} \) and \( L(\mathcal{H}) \) is \( \approx \)-closed

CFM (depending on user specification) can be derived using \textit{synth}. 

Carsten Kern  Learning Communicating and Nondet. Automata
Results:

There are synthesis functions such that the following classes of CFMs are learnable:

**Learnable classes of CFMs:**
- (deterministic) ∀-bounded CFMs
- ∃B-bounded CFMs ($B \in \mathbb{N}$)
- deterministic ∀-bounded deadlock-free weak CFMs

**Not learnable (in a guided fashion)**
- ∀-bounded weak CFMs
existentially $B$-bounded CFMs

An *existentially $B$-bounded* CFM

- Example of an $\exists B$-bounded CFM (bound $B = 1$)

\[ \mathcal{A}_1: \quad \mathcal{A}_2: \]

\[ 1!2(req) \quad 2?1(req) \]

- $D$: domain for $\exists B$-bounded words
- $\approx$: linearization equivalence for $\exists B$-bounded MSCs
- $\text{synth}$: mapping a minimal DFA to a $\exists B$-bounded CFMs
Algorithm for $∃B$-bounded CFMs

Let $\mathcal{H}$ be a minimal DFA (hypothesis)

Problems $L(\mathcal{H}) \subseteq \mathcal{D}$ and $L(\mathcal{H})$ is $≃$-closed are constructively decidable
Algorithm for \( \exists B \)-bounded CFMs

Let \( \mathcal{H} \) be a minimal DFA (hypothesis)

Problems \( L(\mathcal{H}) \subseteq D \) and \( L(\mathcal{H}) \) is \( \simeq \)-closed are constructively decidable

1. mark states of \( \mathcal{H} \) with their channel contents and always check if the channel capacity \( \leq B \)
Algorithm for $\exists B$-bounded CFMs

Let $\mathcal{H}$ be a minimal DFA (hypothesis)

Problems $L(\mathcal{H}) \subseteq D$ and $L(\mathcal{H})$ is $\approx$-closed are constructively decidable

1. mark states of $\mathcal{H}$ with their channel contents and always check if the channel capacity $\leq B$
   - sending adds a message to the corresponding channel
   - receiving removes a message from the channel head
Algorithm for $\exists B$-bounded CFMs

Let $\mathcal{H}$ be a minimal DFA (hypothesis)

Problems $L(\mathcal{H}) \subseteq D$ and $L(\mathcal{H})$ is $\cong$-closed are constructively decidable

1. mark states of $\mathcal{H}$ with their channel contents and always check if the channel capacity $\leq B$
   - sending adds a message to corresponding channel
   - receiving removes a message from channel head

2. check diamond rule

   \[
   \begin{array}{c@{\quad}c@{\quad}c}
   \sigma & \tau & (B\text{-bounded version}) \\
   \tau & \sigma & \text{for independent } \sigma, \tau
   \end{array}
   \]
Algorithm for $\exists B$-bounded CFMs

Let $\mathcal{H}$ be a minimal DFA (hypothesis)

Problems $L(\mathcal{H}) \subseteq D$ and $L(\mathcal{H})$ is $\approx$-closed are constructively decidable

1. mark states of $\mathcal{H}$ with their channel contents and always check if the channel capacity $\leq B$
   - sending adds a message to corresponding channel
   - receiving removes a message from channel head

2. check diamond rule $\begin{array}{c} \sigma \quad \tau \\ \tau \quad \sigma \end{array}$ ($B$-bounded version) for independent $\sigma$, $\tau$

3. if problems in labeling the states are encountered, or the channel capacity $> B$ a counter example can be constructed and the learning algorithm continues
Algorithm for $\exists B$-bounded CFMs

Let $\mathcal{H}$ be a minimal DFA (hypothesis)

Problems $L(\mathcal{H}) \subseteq D$ and $L(\mathcal{H})$ is $\approx$-closed are constructively decidable

1. mark states of $\mathcal{H}$ with their channel contents and always check if the channel capacity $\leq B$
   - sending adds a message to corresponding channel
   - receiving removes a message from channel head

2. check diamond rule
   - (B-bounded version) for independent $\sigma$, $\tau$

3. if problems in labeling the states are encountered, or the channel capacity $> B$ a counter example can be constructed and the learning algorithm continues

Complexity: linear in the size of $\mathcal{H}$
Complexity results

Number of equivalence queries:

- deterministic $\forall B$-bounded CFMs: $(|A| \cdot |Msg| + 1)^B \cdot |Proc|^2 + |Proc|$
- $\forall B$-bounded CFMs: $2(|A| \cdot |Msg| + 1)^B \cdot |Proc|^2 + |Proc|$
- $\exists B$-bounded CFMs: $2(|A| \cdot |Msg| + 1)^B \cdot |Proc|^2 + |Proc|$
- deterministic $\forall$-bounded deadlock-free weak CFMs: $(|A| \cdot |Msg| + 1)^B \cdot |Proc|^2$

Not learnable (in a supported fashion)

- $\forall$-bounded weak CFMs
### Some results

| Protocol                    | #membership queries | #equivalence queries | $|\mathcal{H}|$ | #rows in table | learning setup |
|-----------------------------|---------------------|----------------------|-----------|---------------|----------------|
|                            | w.o. POL | w. POL | savings | w.o. POL | w. POL | reduction |
| part of USB 1.1 continuous update negotiation | 488 | 200 | 59.0% | 14 | 1 (5) | 9 | 61 | 26 | 57.4% | 32 |
| negotiation                 | 712 | 264 | 62.9% | 21 | 1 (3) | 8 | 89 | 34 | 61.8% | 31 |
| ABP                         | 1,179 | 432 | 63.4% | 31 | 1 (3) | 9 | 131 | 49 | 62.6% | 31 |
| ABP                         | 2,286 | 697 | 69.5% | 64 | 2 (4) | 15 | 127 | 42 | 66.9% | 31 |
| ABP                         | 14,432 | 4,557 | 68.4% | 158 | 2 (13) | 25 | 451 | 131 | 71.0% | 32 |
| ABP                         | 55,131 | 19,252 | 65.1% | 407 | 2 (22) | 37 | 799 | 222 | 72.2% | 33 |
| leader elec. ($v_1$)        | 3,612 | 900 | 75.1% | 43 | 1 (2) | 13 | 301 | 76 | 74.8% | 4 |
| leader elec. ($v_2$)        | 14,704 | 6,864 | 53.3% | 196 | 2 (5) | 17 | 919 | 430 | 53.2% | 4 |
Some results

| Protocol                  | # membership queries w.o. POL | w. POL | savings | #user queries | #equivalence queries | $|\mathcal{H}|$ | # rows in table w.o. POL | w. POL | reduction | learning setup |
|--------------------------|-------------------------------|--------|---------|---------------|---------------------|--------|------------------------|--------|------------|---------------------|
| part of USB 1.1          | 488                           | 200    | 59.0%   | 14            | 1 (5)               | 9      | 61                     | 26     | 57.4%      | $\exists 2$          |
| continuous update        | 712                           | 264    | 62.9%   | 21            | 1 (3)               | 8      | 89                     | 34     | 61.8%      | $\exists 1$          |
| negotiation              | 1,179                         | 432    | 63.4%   | 31            | 1 (3)               | 9      | 131                    | 49     | 62.6%      | $\exists 1$          |
| ABP                      | 2,286                         | 697    | 69.5%   | 64            | 2 (4)               | 15     | 127                    | 42     | 66.9%      | $\exists 1$          |
| ABP                      | 14,432                        | 4,557  | 68.4%   | 158           | 2 (13)              | 25     | 451                    | 131    | 71.0%      | $\exists 2$          |
| ABP                      | 55,131                        | 19,252 | 65.1%   | 407           | 2 (22)              | 37     | 799                    | 222    | 72.2%      | $\exists 3$          |
| leader elec. ($v_1$)     | 3,612                         | 900    | 75.1%   | 43            | 1 (2)               | 13     | 301                    | 76     | 74.8%      | $\forall$            |
| leader elec. ($v_2$)     | 14,704                        | 6,864  | 53.3%   | 196           | 2 (5)               | 17     | 919                    | 430    | 53.2%      | $\forall$            |

- # membership queries: reduced by partial order learning (POL)
## Some results

| Protocol               | #membership queries | #user queries | #equivalence queries | | #rows in table | learning setup |
|------------------------|---------------------|--------------|----------------------|-----------------|----------------|
|                        | w.o. POL | w. POL | savings |                        | w.o. POL | w. POL | reduction |                      |
| part of USB 1.1        | 488     | 200    | 59.0%  | 14                  | 9         | 61     | 26        | 57.4%  | ∃2                  |
| continuous update      | 712     | 264    | 62.9%  | 21                  | 8         | 89     | 34        | 61.8%  | ∃1                  |
| negotiation            | 1,179   | 432    | 63.4%  | 31                  | 9         | 131    | 49        | 62.6%  | ∃1                  |
| ABP                    | 2,286   | 697    | 69.5%  | 64                  | 15        | 127    | 42        | 66.9%  | ∃1                  |
| ABP                    | 14,432  | 4,557  | 68.4%  | 158                 | 25        | 451    | 131       | 71.0%  | ∃2                  |
| ABP                    | 55,131  | 19,252 | 65.1%  | 407                 | 37        | 799    | 222       | 72.2%  | ∃3                  |
| leader elec. ($v_1$)  | 3,612   | 900    | 75.1%  | 43                  | 13        | 301    | 76        | 74.8%  | ∀                   |
| leader elec. ($v_2$)  | 14,704  | 6,864  | 53.3%  | 196                 | 17        | 919    | 430       | 53.2%  | ∀                   |

- # membership queries: reduced by **partial order learning (POL)**
- # equivalence queries: reduced by our learning approach
## Some results

| Protocol            | #membership queries w.o. POL | #membership queries w. POL | savings | #user queries | #equivalence queries | | | **| | | #rows in table w.o. POL | #rows in table w. POL | reduction | learning setup |
|--------------------|-----------------------------|-----------------------------|---------|---------------|---------------------|---|----------------|------------------|------------------|------------------|
| part of USB 1.1    | 488                         | 200                         | 59.0%   | 14            | 1 (5)               | 9 | 61             | 26               | 57.4%            | 32               |
| continuous update  | 712                         | 264                         | 62.9%   | 21            | 1 (3)               | 8 | 89             | 34               | 61.8%            | 31               |
| negotiation        | 1,179                       | 432                         | 63.4%   | 31            | 1 (3)               | 9 | 131            | 49               | 62.6%            | 31               |
| ABP                | 2,286                       | 697                         | 69.5%   | 64            | 2 (4)               | 15| 127            | 42               | 66.9%            | 31               |
| ABP                | 14,432                      | 4,557                       | 68.4%   | 158           | 2 (13)              | 25| 451            | 131              | 71.0%            | 32               |
| ABP                | 55,131                      | 19,252                      | 65.1%   | 407           | 2 (22)              | 37| 799            | 222              | 72.2%            | 33               |
| leader elec. ($v_1$) | 3,612                      | 900                         | 75.1%   | 43            | 1 (2)               | 13| 301            | 76               | 74.8%            | ∀                |
| leader elec. ($v_2$) | 14,704                     | 6,864                       | 53.3%   | 196           | 2 (5)               | 17| 919            | 430              | 53.2%            | ∀                |

- # membership queries: reduced by partial order learning (POL)
- # equivalence queries: reduced by our learning approach
- # user queries: reducible by employing a logic (PDL)
Related Work

Similar Approaches

- **Play-In/Play-Out** approach [Harel et al.]
  - use the more expressive language of LSCs
  - more involved treatment of negative scenarios
  - problem: detecting inconsistencies

- **MAS (Minimally Adequate Synthesizer)** [Mäkinen et al.]
  - based on Angluin’s learning approach
  - only synchronous/sequential behavior
  - implementation model is not distributed
Presentation outline

1. Learning Deterministic Automata
2. Learning Nondeterministic Automata
3. Learning Communicating Automata
4. Tools
5. Conclusion
libalf: the learning library

Features
- implements wide range of learning algorithms: $L^*$, $L^*_\text{col}$, NL*, PO learning, Biermann, RPNI, DeLeTe2, etc.
- written in C++
- approx. 13,500 lines of code
Smyle: Synthesizing Models by Learning from Examples

Features

- implements $\forall/\exists - B/\ldots$ learning setups
- written in Java 1.6
- implements partial order learning
- implements a logic (PDL) for reducing user queries
- approx. 24,000 lines of code
Smyle: Synthesizing Models by Learning from Examples

Features

- implements $\forall/\exists - B/\ldots$ learning setups
- written in Java 1.6
- implements partial order learning
- implements a logic (PDL) for reducing user queries
- approx. 24,000 lines of code

External libraries

- libalf
- GRAPPA (visualization of automata)
- JGraph (visualization of MSCs)
- MSC2000 (Parser for MSC documents)
Smyle: Synthesizing Models by Learning from Examples

Features

- implements $\forall/\exists \rightarrow B/\ldots$ learning setups
- written in Java 1.6
- implements partial order learning
- implements a logic (PDL) for reducing user queries
- approx. 24,000 lines of code

External libraries

- libalf
- GRAPPA (visualization of automata)
- JGraph (visualization of MSCs)
- MSC2000 (Parser for MSC documents)

http://www.smyle-tool.org
Presentation outline

1. Learning Deterministic Automata
2. Learning Nondeterministic Automata
3. Learning Communicating Automata
4. Tools
5. Conclusion
Results achieved:

- first active online learning algorithm for NFA: NL*
- several classes of CFMs learnable by an extension to L*
- optimizations of learning algorithms (POL, PDL, etc.)
- tools supporting the theory
- ...
Outlook

Open problems:

- applying NL* in fields like verification, robotics, etc.
- detect further learnable classes of CFMs
- learn other classes of automata (Büchi automata, alternating automata)
- ...

Carsten Kern  Learning Communicating and Nondet. Automata
List of Publications

(1) **MSCan: A tool for analyzing MSC specifications.**
    Bollig, Kern, Schlütter, Stolz.  
    *(TACAS 2006)*, LNCS.

(2) **Replaying play in and play out: Synthesis of design models from scenarios by learning.**
    Bollig, Katoen, Kern, Leucker.  
    *(TACAS 2007)*, LNCS.

(3) **Smyle: A Tool for Synthesizing Distributed Models from Scenarios by Learning.**
    Bollig, Katoen, Kern, Leucker.  
    *(CONCUR 2008)*, LNCS.

(4) **SMA—The Smyle Modeling Approach.**
    Bollig, Katoen, Kern, Leucker.  
    *(CEE-SET 2008 (IFIP))* , LNCS.

(5) **Angluin-Style Learning of NFA.**
    Bollig, Habermehl, Kern, Leucker.  
    *(IJCAI 2009)*.

(6) **SMA—The Smyle Modeling Approach.**
    Bollig, Katoen, Kern, and Leucker.  
    *(Computing and Informatics, to appear)*.

(7) **Learning Communicating Automata from MSCs.**
    Bollig, Katoen, Kern, Leucker.  
    *(Submitted to IEEE TSE)*.
Appendix

- RFSA-closedness and -consistency
- NL* in action
Designing a table-based learning algorithm

**Definition (RFSA-Closedness)**

Table $\mathcal{T} = (T, U, V)$ is called **RFSA-closed** if, for each $r \in \text{Rows}_{\text{low}}(\mathcal{T})$,

$$r = \bigcup \{ r' \in \text{Primes}_{\text{upp}}(\mathcal{T}) \mid r' \sqsubseteq r \}$$

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aba$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$abb$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
</tbody>
</table>
Designing a table-based learning algorithm

**Definition (RFSA-Closedness)**

Table $\mathcal{T} = (T, U, V)$ is called **RFSA-closed** if, for each $r \in \text{Rows}_{\text{low}}(\mathcal{T})$,

$$r = \bigcup \{ r' \in \text{Primes}_{\text{upp}}(\mathcal{T}) \mid r' \sqsubseteq r \}$$

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aba$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$abb$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
</tbody>
</table>

$\mathcal{T}$ is RFSA-closed:

- $\text{row}(aa) = \text{row}(\varepsilon) \sqcup \text{row}(a) \sqcup \text{row}(ab)$ and
Designing a table-based learning algorithm

**Definition (RFSA-Closedness)**

Table $\mathcal{T} = (T, U, V)$ is called **RFSA-closed** if, for each $r \in \text{Rows}_{\text{low}}(\mathcal{T})$, 

$$r = \bigsqcup \{ r' \in \text{Primes}_{\text{upp}}(\mathcal{T}) \mid r' \sqsubseteq r \}$$

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aba$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$abb$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
</tbody>
</table>

$\mathcal{T}$ is RFSA-closed:

- $\text{row}(aa) = \text{row}(\varepsilon) \sqcup \text{row}(a) \sqcup \text{row}(ab)$ and
- $\text{row}(b), \text{row}(aba), \text{row}(abb) \in \text{Primes}_{\text{upp}}(\mathcal{T})$
Designing a table-based learning algorithm

**Definition (RFSA-Consistency)**

A table $\mathcal{T} = (T, U, V)$ is called **RFSA-consistent** if, for all $u, u' \in U$ and $a \in \Sigma$:

$$\text{row}(u') \sqsubseteq \text{row}(u) \Rightarrow \text{row}(u'a) \sqsubseteq \text{row}(ua)$$

<table>
<thead>
<tr>
<th>$T$</th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aba$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$abb$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
</tbody>
</table>
Designing a table-based learning algorithm

**Definition (RFSA-Consistency)**

A table $\mathcal{T} = (T, U, V)$ is called **RFSA-consistent** if, for all $u, u' \in U$ and $a \in \Sigma$:

$$row(u') \sqsubseteq row(u) \Rightarrow row(u'a) \sqsubseteq row(ua)$$

<table>
<thead>
<tr>
<th>$\mathcal{T}$</th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aba$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$abb$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
</tbody>
</table>

**$\mathcal{T}$ is RFSA-consistent**

- $row(\varepsilon) \sqsubseteq row(a)$:
  - $row(a) \sqsubseteq row(aa)$ and
  - $row(b) \sqsubseteq row(ab)$
Designing a table-based learning algorithm

Definition (RFSA-Consistency)

A table $\mathcal{T} = (T, U, V)$ is called RFSA-consistent if, for all $u, u' \in U$ and $a \in \Sigma$:

$$\text{row}(u') \subseteq \text{row}(u) \Rightarrow \text{row}(u'a) \subseteq \text{row}(ua)$$

<table>
<thead>
<tr>
<th>$\mathcal{T}$</th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$aa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$a$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$+$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$b$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$aba$</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$abb$</td>
<td>$-$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
</tbody>
</table>

$\mathcal{T}$ is RFSA-consistent

1. $\text{row}(\varepsilon) \subseteq \text{row}(a)$:
   - $\text{row}(a) \subseteq \text{row}(aa)$ and
   - $\text{row}(b) \subseteq \text{row}(ab)$
Designing a table-based learning algorithm

**Definition (RFSA-Consistency)**

A table \( \mathcal{T} = (T, U, V) \) is called RFSA-consistent if, for all \( u, u' \in U \) and \( a \in \Sigma \):

\[
\text{row}(u') \subseteq \text{row}(u) \Rightarrow \text{row}(u'a) \subseteq \text{row}(ua)
\]

<table>
<thead>
<tr>
<th>( \mathcal{T} )</th>
<th>( \varepsilon )</th>
<th>( a )</th>
<th>( aa )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varepsilon )</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>( a )</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( ab )</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>( b )</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>( aa )</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( aba )</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( abb )</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

\( \mathcal{T} \) is RFSA-consistent

- \( \text{row}(\varepsilon) \subseteq \text{row}(a) \):
  - \( \text{row}(a) \subseteq \text{row}(aa) \) and
  - \( \text{row}(b) \subseteq \text{row}(ab) \)

- \( \text{row}(\varepsilon) \subseteq \text{row}(ab) \):
  - \( \text{row}(a) \subseteq \text{row}(aba) \) and
  - \( \text{row}(b) \subseteq \text{row}(abb) \)
Designing a table-based learning algorithm

Definition (RFSA-Consistency)

A table \( T = (T, U, V) \) is called RFSA-consistent if, for all \( u, u' \in U \) and \( a \in \Sigma \):

\[
row(u') \sqsubseteq row(u) \Rightarrow row(u'a) \sqsubseteq row(ua)
\]

<table>
<thead>
<tr>
<th></th>
<th>( \varepsilon )</th>
<th>( a )</th>
<th>( aa )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varepsilon )</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>( a )</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( ab )</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>( b )</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>( aa )</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( aba )</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( abb )</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

\( T \) is RFSA-consistent

- \( \mathbf{row}(\varepsilon) \sqsubseteq \mathbf{row}(a) \):
  - \( \mathbf{row}(a) \sqsubseteq \mathbf{row}(aa) \) and
  - \( \mathbf{row}(b) \sqsubseteq \mathbf{row}(ab) \)

- \( \mathbf{row}(\varepsilon) \sqsubseteq \mathbf{row}(ab) \):
  - \( \mathbf{row}(a) \sqsubseteq \mathbf{row}(aba) \) and
  - \( \mathbf{row}(b) \sqsubseteq \mathbf{row}(abb) \)
The algorithm in action

Create initial table $\mathcal{T}_1$. 

<table>
<thead>
<tr>
<th>$\mathcal{T}_1$</th>
<th>$\epsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>* $\epsilon$</td>
<td>$-$</td>
</tr>
<tr>
<td>* a</td>
<td>$-$</td>
</tr>
<tr>
<td>* b</td>
<td>$-$</td>
</tr>
</tbody>
</table>
The algorithm in action

The algorithm in action

Hypothesis $\mathcal{R}_{T_1}$:

$\Rightarrow$ Counterexample is $aaa$.

$\Rightarrow$ Add $\text{Suff}(aaa)$ to $V$. 

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$*$</td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>$*$</td>
<td>$a$</td>
</tr>
<tr>
<td>$*$</td>
<td>$b$</td>
</tr>
</tbody>
</table>
The algorithm in action

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>* $\varepsilon$</td>
<td>-</td>
</tr>
<tr>
<td>* a</td>
<td>-</td>
</tr>
<tr>
<td>* b</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$T_2$</th>
<th>$\varepsilon$</th>
<th>aaa</th>
<th>aa</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>* $\varepsilon$</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>* a</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>* b</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
$T_2$ is not closed: row($a$) $\notin$ Primes_{upp}
The algorithm in action

![Diagram of the algorithm in action with states and transitions labeled with symbols generating string 'a' and 'b'.]

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>* $\varepsilon$</td>
<td>*</td>
</tr>
<tr>
<td>* $a$</td>
<td>*</td>
</tr>
<tr>
<td>* $b$</td>
<td>*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$T_2$</th>
<th>$\varepsilon$</th>
<th>aaa</th>
<th>aa</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>* $\varepsilon$</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>* $a$</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>* $b$</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$T_3$</th>
<th>$\varepsilon$</th>
<th>aaa</th>
<th>aa</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>* $\varepsilon$</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>* $a$</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>* $b$</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>* $aa$</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>* $ab$</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>
The algorithm in action

$\mathcal{T}_3$ is not closed: $\text{row}(ab) \notin \text{Primes}_{\text{upp}}$

<table>
<thead>
<tr>
<th>$\mathcal{T}_1$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>* $\varepsilon$</td>
<td>-</td>
</tr>
<tr>
<td>* a</td>
<td>-</td>
</tr>
<tr>
<td>* b</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\mathcal{T}_2$</th>
<th>$\varepsilon$</th>
<th>aaa</th>
<th>aa</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>* $\varepsilon$</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>* a</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>* b</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\mathcal{T}_3$</th>
<th>$\varepsilon$</th>
<th>aaa</th>
<th>aa</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>* $\varepsilon$</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>* a</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>* b</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>aa</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>* ab</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>
The algorithm in action

\[ T_1 \]
\[ T_2 \]
\[ T_3 \]
\[ T_4 \]
The algorithm in action

$\mathcal{T}_4$ is not closed: $\text{row}(abb) \notin \text{Primes}_{\text{upp}}$
The algorithm in action

\[ T_1 \begin{array}{c|c} \varepsilon & - \\ \hline \varepsilon & - \\ a & - \\ b & - \end{array} \]

\[ T_2 \begin{array}{c|c|c|c} \varepsilon & \text{aaa} & \text{aa} & \text{a} \\ \hline \varepsilon & - & + & - \\ a & - & + & + \\ ab & - & + & + \end{array} \]

\[ T_3 \begin{array}{c|c|c|c|c} \varepsilon & \text{aaa} & \text{aa} & \text{a} \\ \hline \varepsilon & - & + & - \\ a & - & + & + \\ ab & - & + & + \end{array} \]

\[ T_4 \begin{array}{c|c|c|c|c} \varepsilon & \text{aaa} & \text{aa} & \text{a} \\ \hline \varepsilon & - & + & - \\ a & - & + & + \\ ab & - & + & + \\ abb & - & + & + \end{array} \]

\[ T_5 \begin{array}{c|c|c|c|c} \varepsilon & \text{aaa} & \text{aa} & \text{a} \\ \hline \varepsilon & - & + & - \\ a & - & + & + \\ ab & - & + & + \\ abba & - & + & + \end{array} \]
The algorithm in action

$T_5$ is closed and consistent: 
⇒ $R_{T_5}$ can be derived.
The algorithm in action

\[ \mathcal{T}_5 \] is closed and consistent:
\[ \Rightarrow \mathcal{R}_{\mathcal{T}_5} \] can be derived.