

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

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

Presentation outline

- 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)

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

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)

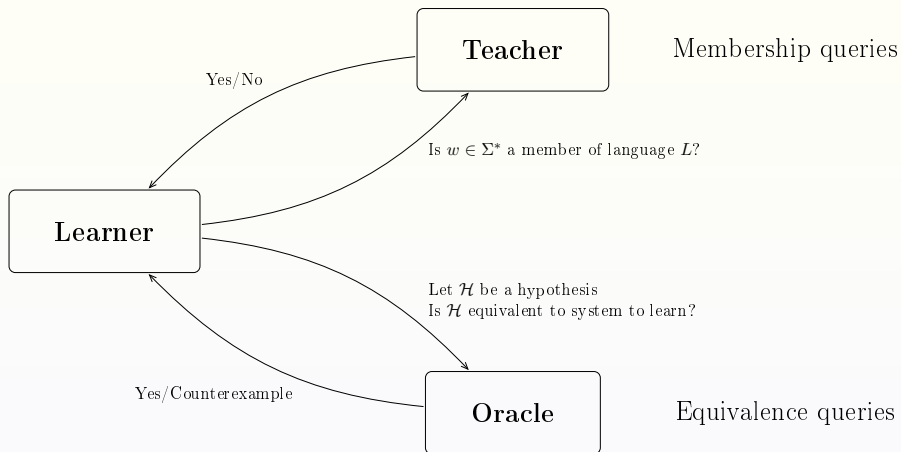
Active Learning

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

Occam's razor:

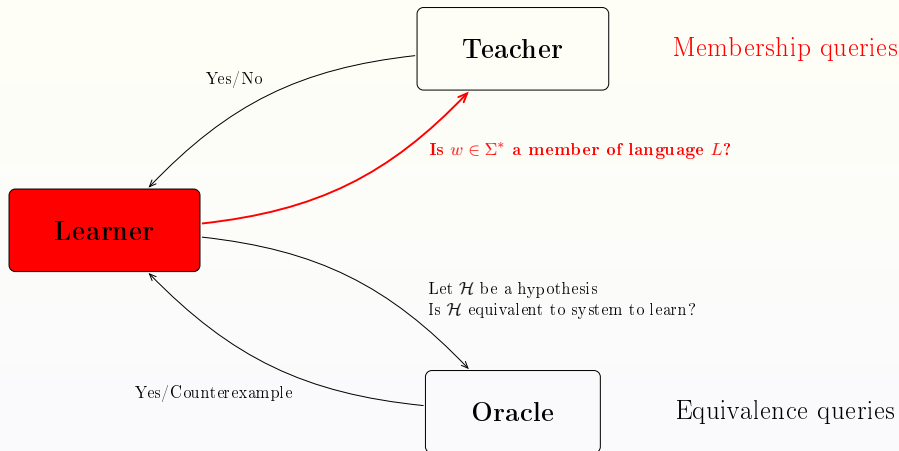
"In case of different explanations, choose the *simplest* one."
⇒ Learn the *minimal* DFA conforming to given examples

Algorithm - Overview



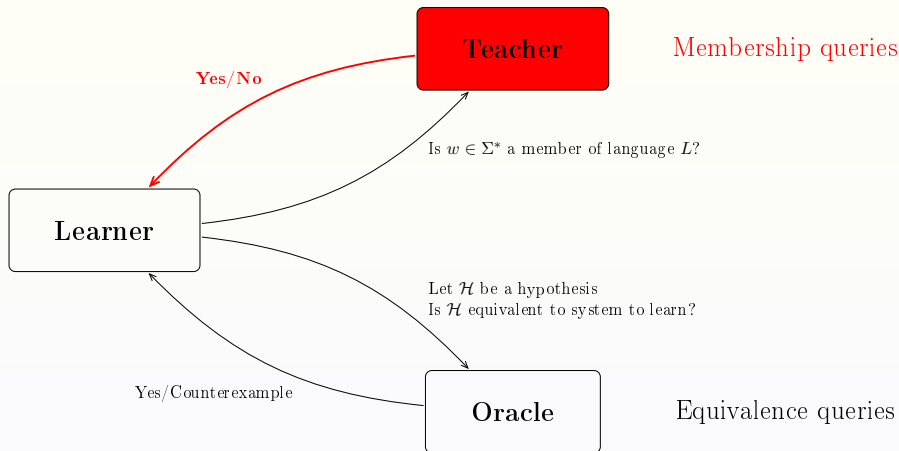
- L : (regular) language to learn

Algorithm - Overview



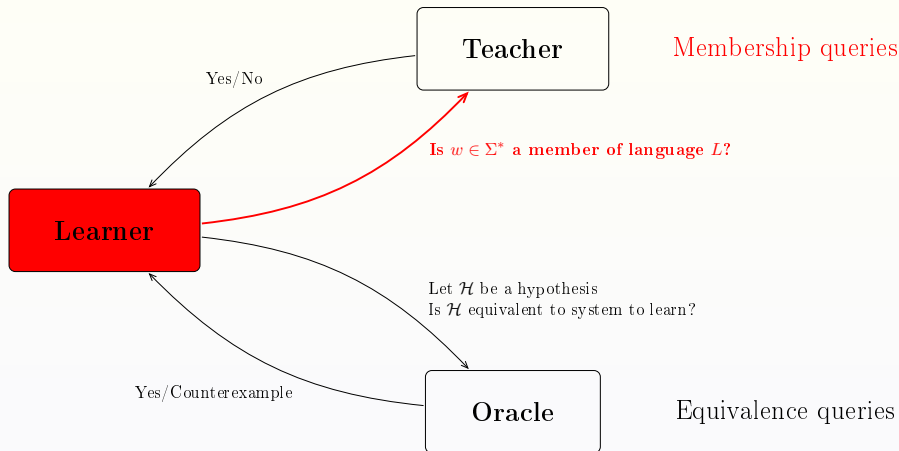
- L : (regular) language to learn

Algorithm - Overview



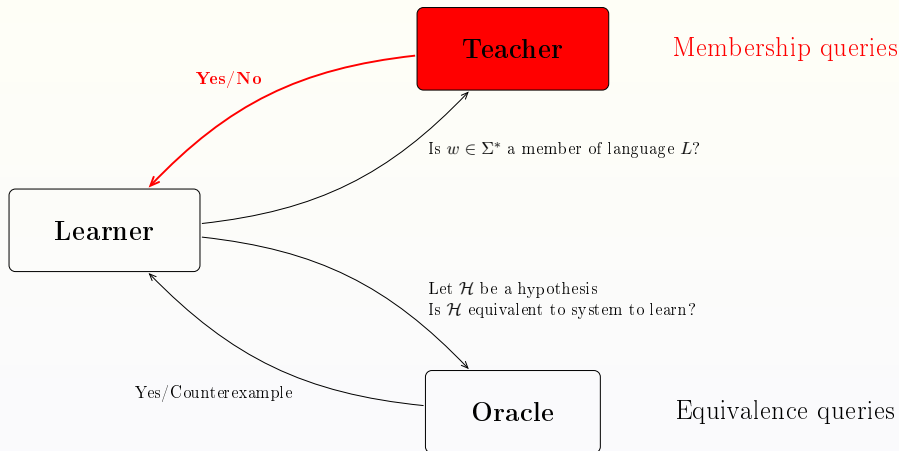
- L : (regular) language to learn

Algorithm - Overview



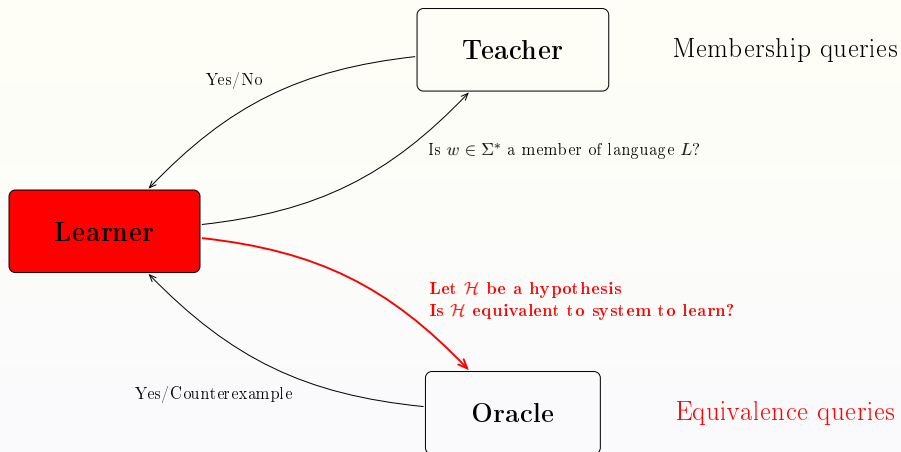
- L : (regular) language to learn

Algorithm - Overview



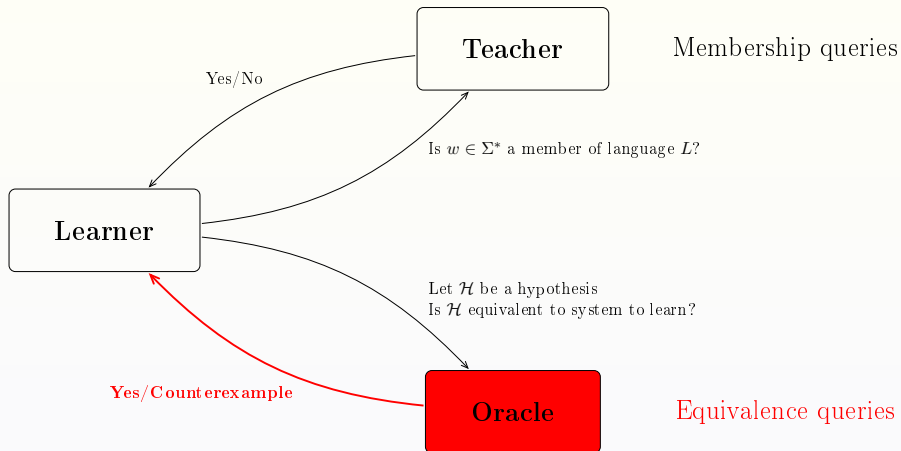
- L : (regular) language to learn

Algorithm - Overview



- L : (regular) language to learn

Algorithm - Overview



- L : (regular) language to learn
- Counterexample: $w \in (L(\mathcal{H}) \setminus L) \cup (L \setminus L(\mathcal{H}))$

Table-based learning

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

$\mathcal{T}$	ε
ε	
a	
b	
aa	
ab	

$\varepsilon \in L?$

Table-based learning

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

$\mathcal{T}$	ε
ε	+
a	
b	
aa	
ab	

$a \in L?$

Table-based learning

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

$\mathcal{T}$	ε
ε	+
a	-
b	
aa	
ab	

$b, aa, ab \in L?$

Table-based learning

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

$\mathcal{T}$	ε
ε	+
a	-
b	-
aa	-
ab	+

Table-based learning

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

$\mathcal{T}$	ε
ε	+
a	-
b	-
aa	-
ab	+

To derive an automaton:

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

Table-based learning

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

$\mathcal{T}$	ε
ε	+
a	-
b	-
aa	-
ab	+

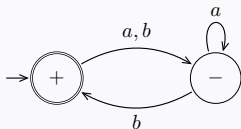
To derive an automaton:

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

Table-based learning

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

$\mathcal{T}$	ε
ε	+
a	-
b	-
aa	-
ab	+



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

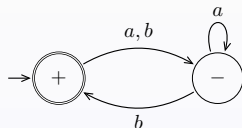
To this end:

- upper rows serve to derive states
- lower rows serve to derive transitions

Table-based learning

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

$\mathcal{T}$	ε
ε	+
a	-
b	-
aa	-
ab	+



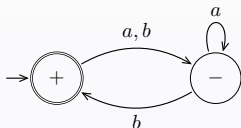
$bb \in L(\mathcal{H})$ but $bb \notin L!$

Table-based learning

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

$\mathcal{T}$	ε
ε	+
a	-
b	-
aa	-
ab	+

Counterexample can be added to:



$bb \in L(\mathcal{H})$ but $bb \notin L$!

Table-based learning

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

$\mathcal{T}$	ε
ε	+
a	-
b	-
bb	-
aa	-
ab	+
ba	-
bba	-
bbb	-

Counterexample can be added to:

- the rows (L^*)

Table-based learning

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

$\mathcal{T}$	ε	bb	b
ε	+	-	-
a	-	-	+
b	-	-	-
aa	-	-	-
ab	+	-	-

Counterexample can be added to:

- the rows (L^*)
- the columns (L_{col}^*)

Theorem (Complexity of L^*)

Let:

- n : number of states of the minimal DFA $\mathcal{A}_L$ for regular language L ,
- m : length of the biggest counterexample

Then, L^* returns after at most:

the minimal DFA $\mathcal{A}$.

Theorem (Complexity of L^*)

Let:

- n : number of states of the minimal DFA $\mathcal{A}_L$ for regular language L ,
- m : length of the biggest counterexample

Then, L^* returns after at most:

- n equivalence queries and

the minimal DFA $\mathcal{A}$.

Theorem (Complexity of L^*)

Let:

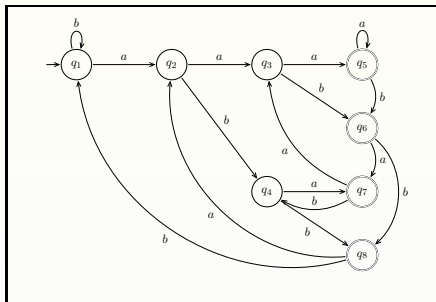
- n : number of states of the minimal DFA $\mathcal{A}_L$ for regular language L ,
- m : length of the biggest counterexample

Then, L^* returns after at most:

- n equivalence queries and
- $O(m|\Sigma|n^2)$ membership queries

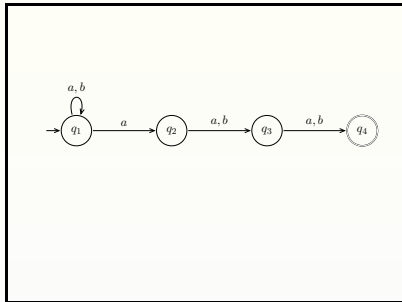
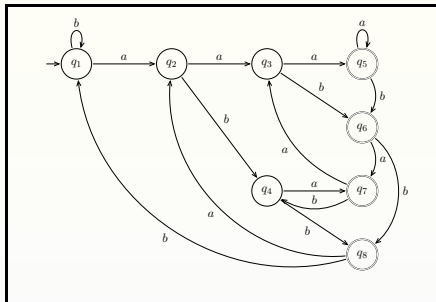
the minimal DFA $\mathcal{A}$.

But there is a problem ...



minimal DFA can be huge!

But there is a problem ...

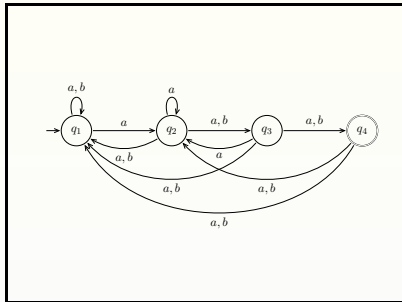
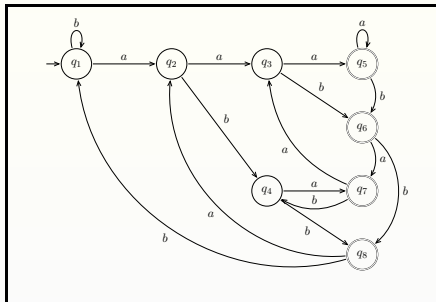


minimal DFA can be huge!

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!

Presentation outline

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

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**

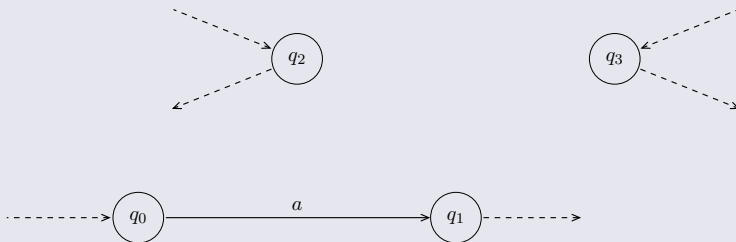
Goal

- Learn **more compact representations** of regular languages

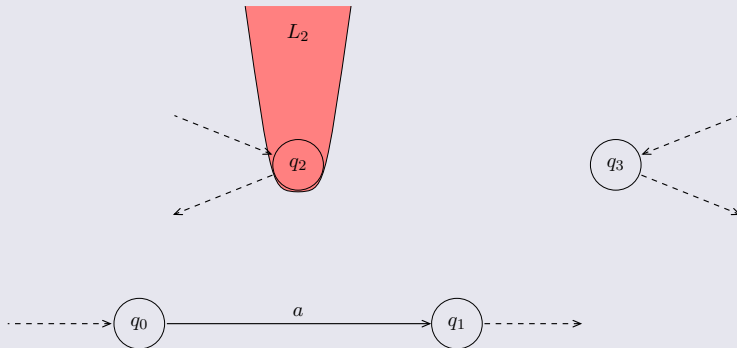
How?

- Use residual finite-state automata (RFSA) for learning

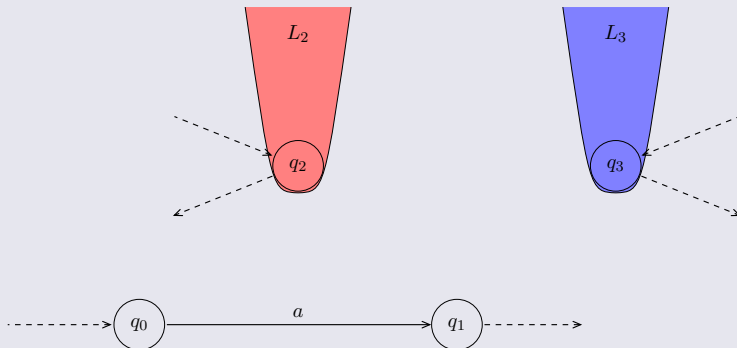
Residual Finite-State Automata [Denis et al.]



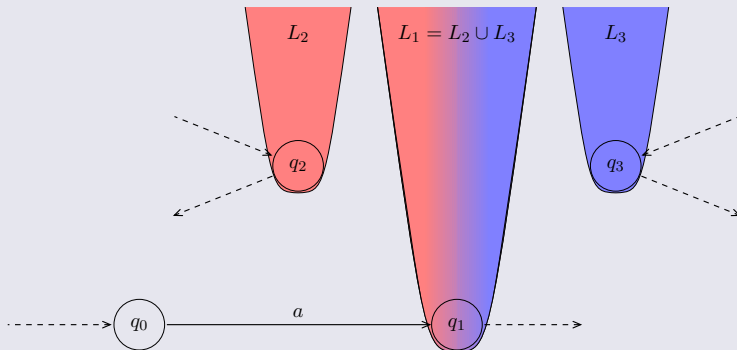
Residual Finite-State Automata [Denis et al.]



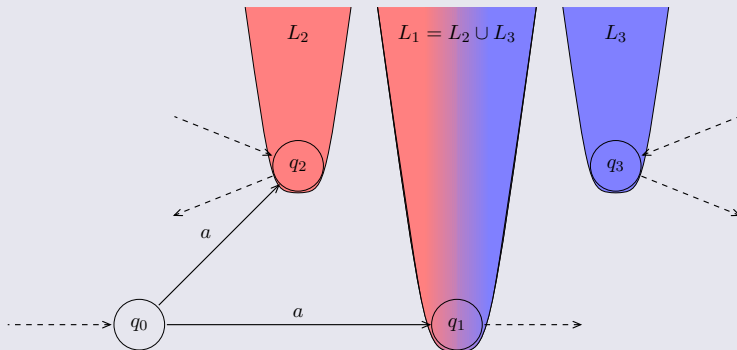
Residual Finite-State Automata [Denis et al.]



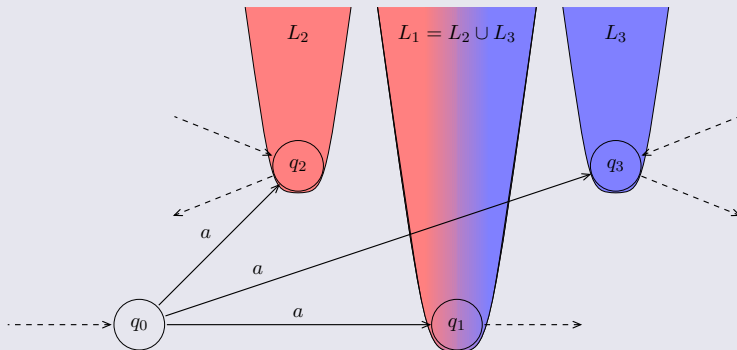
Residual Finite-State Automata [Denis et al.]



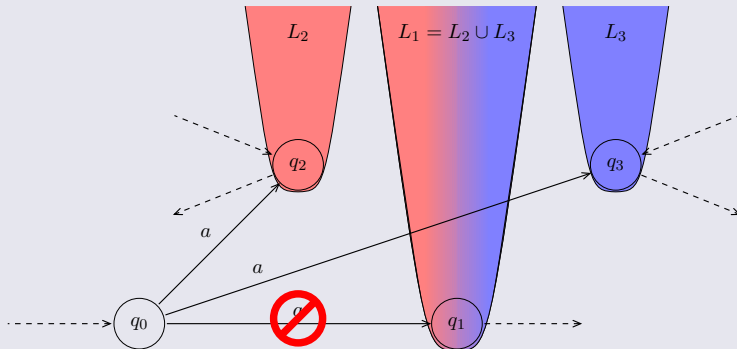
Residual Finite-State Automata [Denis et al.]



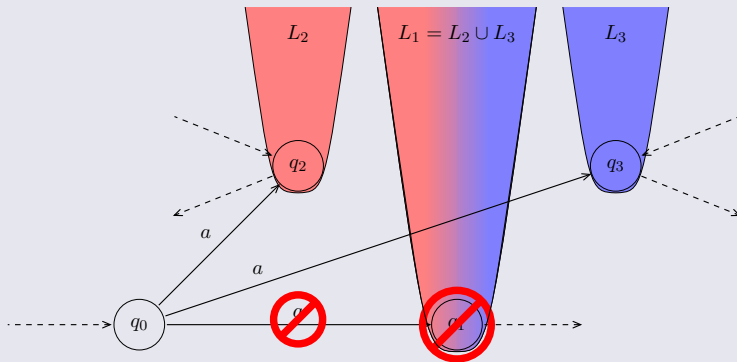
Residual Finite-State Automata [Denis et al.]



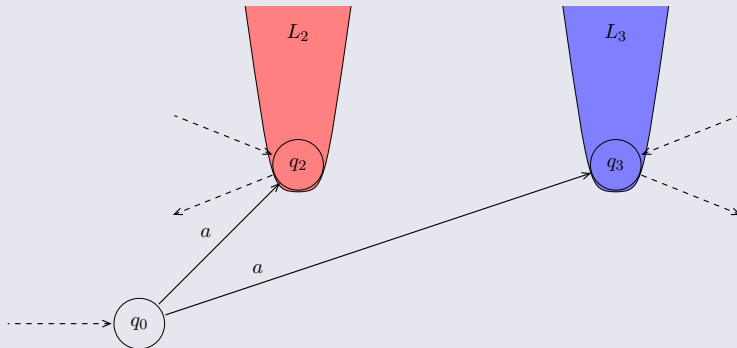
Residual Finite-State Automata [Denis et al.]



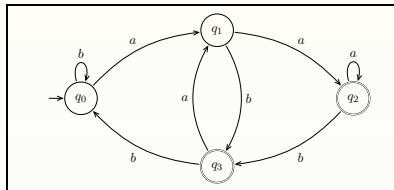
Residual Finite-State Automata [Denis et al.]



Residual Finite-State Automata [Denis et al.]



Residual Finite-State Automata



Definition (Residual Language)

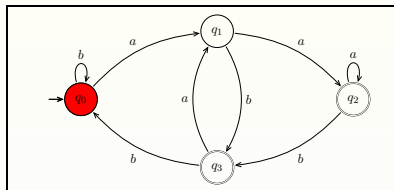
For a language $L \subseteq \Sigma^*$ and $u \in \Sigma^*$:

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

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

$\text{Res}(L)$: the set of residual languages of L .

Residual Finite-State Automata



$$\bullet \varepsilon^{-1}L = \Sigma^*a\Sigma$$

$$(= L_{q_0})$$

Definition (Residual Language)

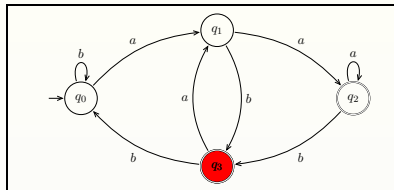
For a language $L \subseteq \Sigma^*$ and $u \in \Sigma^*$:

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

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

$\text{Res}(L)$: the set of residual languages of L .

Residual Finite-State Automata



- $\varepsilon^{-1}L = \Sigma^*a\Sigma$ ($= L_{q_0}$)
- $(ab)^{-1}L = \Sigma^*a\Sigma \cup \{\varepsilon\}$ ($= L_{q_3}$)

Definition (Residual Language)

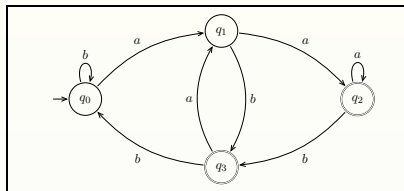
For a language $L \subseteq \Sigma^*$ and $u \in \Sigma^*$:

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

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

$\text{Res}(L)$: the set of residual languages of L .

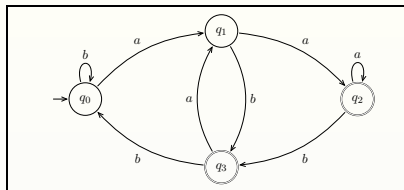
Residual Finite-State Automata



Definition (Residual Finite-State Automaton)

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

Residual Finite-State Automata

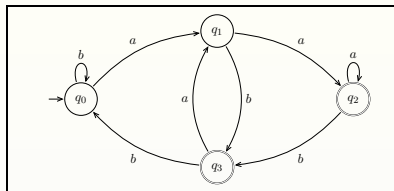


- $\varepsilon^{-1}L = \Sigma^*a\Sigma$ ($= L_{q_0}$)
- $(a)^{-1}L = \Sigma^*a\Sigma \cup \Sigma$ ($= L_{q_1}$)
- $(b)^{-1}L = \Sigma^*a\Sigma \cup \Sigma \cup \{\varepsilon\}$ ($= L_{q_2}$)
- $(ab)^{-1}L = \Sigma^*a\Sigma \cup \{\varepsilon\}$ ($= L_{q_3}$)

Definition (Residual Finite-State Automaton)

A **residual finite-state automaton** (RFSA) over Σ 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



- $\varepsilon^{-1}L = \Sigma^*a\Sigma$ ($= L_{q0}$)
- $(a)^{-1}L = \Sigma^*a\Sigma \cup \Sigma$ ($= L_{q1}$)
- $(b)^{-1}L = \Sigma^*a\Sigma \cup \Sigma \cup \{\varepsilon\}$ ($= L_{q2}$)
- $(ab)^{-1}L = \Sigma^*a\Sigma \cup \{\varepsilon\}$ ($= L_{q3}$)

Definition (Prime and Composed Residuals)

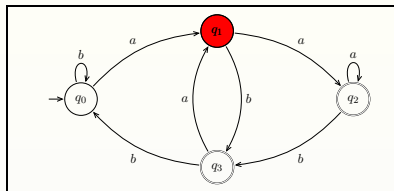
Let $L \subseteq \Sigma^*$ be a language. A **residual** $L' \in \text{Res}(L)$ is called **composed** if there are $L_1, \dots, L_n \in \text{Res}(L) \setminus \{L'\}$ such that

$$L' = L_1 \cup \dots \cup L_n$$

Otherwise, it is called **prime**.

The set of prime residuals of L is denoted by $\text{Primes}(L)$.

Towards canonical RFSA



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

Definition (Prime and Composed Residuals)

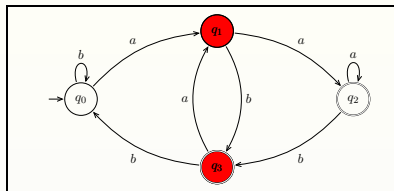
Let $L \subseteq \Sigma^*$ be a language. A **residual** $L' \in \text{Res}(L)$ is called **composed** if there are $L_1, \dots, L_n \in \text{Res}(L) \setminus \{L'\}$ such that

$$L' = L_1 \cup \dots \cup L_n$$

Otherwise, it is called **prime**.

The set of prime residuals of L is denoted by $\text{Primes}(L)$.

Towards canonical RFSA



- $\varepsilon^{-1}L = \Sigma^*a\Sigma$ ($= L_{q0}$)
- $(a)^{-1}L = \Sigma^*a\Sigma \cup \Sigma$ ($= L_{q1}$)
- $(b)^{-1}L = \Sigma^*a\Sigma \cup \Sigma \cup \{\varepsilon\}$ ($= L_{q2}$)
- $(ab)^{-1}L = \Sigma^*a\Sigma \cup \{\varepsilon\}$ ($= L_{q3}$)

Definition (Prime and Composed Residuals)

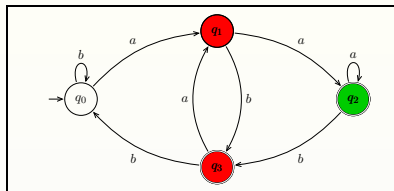
Let $L \subseteq \Sigma^*$ be a language. A **residual** $L' \in \text{Res}(L)$ is called **composed** if there are $L_1, \dots, L_n \in \text{Res}(L) \setminus \{L'\}$ such that

$$L' = L_1 \cup \dots \cup L_n$$

Otherwise, it is called **prime**.

The set of prime residuals of L is denoted by $\text{Primes}(L)$.

Towards canonical RFSA



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

Definition (Prime and Composed Residuals)

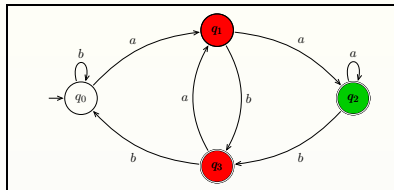
Let $L \subseteq \Sigma^*$ be a language. A **residual** $L' \in \text{Res}(L)$ is called **composed** if there are $L_1, \dots, L_n \in \text{Res}(L) \setminus \{L'\}$ such that

$$L' = L_1 \cup \dots \cup L_n$$

Otherwise, it is called **prime**.

The set of prime residuals of L is denoted by $\text{Primes}(L)$.

Canonical RFSA



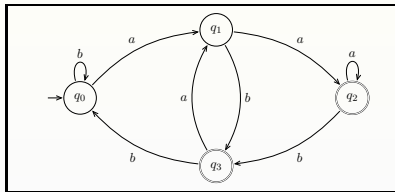
- $\varepsilon^{-1}L = \Sigma^*a\Sigma$ ($= L_{q_0}$)
- $(a)^{-1}L = \Sigma^*a\Sigma \cup \Sigma$ ($= L_{q_1}$)
- $(b)^{-1}L = \Sigma^*a\Sigma \cup \Sigma \cup \{\varepsilon\}$ ($= L_{q_2}$)
- $(ab)^{-1}L = \Sigma^*a\Sigma \cup \{\varepsilon\}$ ($= L_{q_3}$)

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

Let L be a regular language. The **canonical RFSA** of L , denoted by $\mathcal{R}(L)$, is the tuple (Q, Q_0, F, δ) 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 $a \in \Sigma$.

Example: Deriving the canonical RFSA for $L = \Sigma^* a \Sigma$



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

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

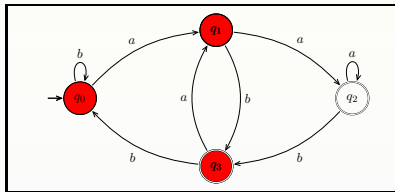
$$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$$

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

$$L_{q_3} = \Sigma^* a \Sigma \cup \{\varepsilon\}$$

Example: Deriving the canonical RFSA for $L = \Sigma^* a \Sigma$

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



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

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

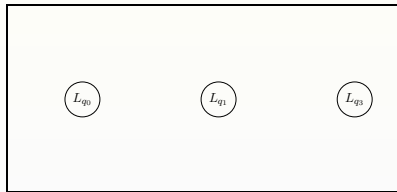
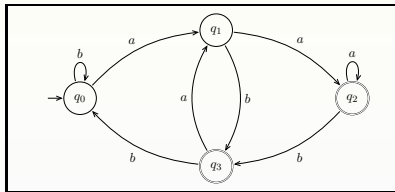
$$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$$

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

$$L_{q_3} = \Sigma^* a \Sigma \cup \{\varepsilon\}$$

Example: Deriving the canonical RFSA for $L = \Sigma^* a \Sigma$

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



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

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

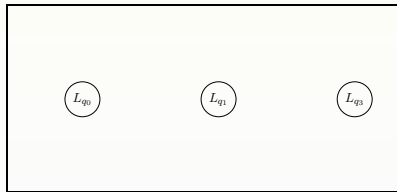
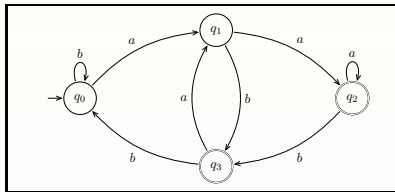
$$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$$

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

$$L_{q_3} = \Sigma^* a \Sigma \cup \{\varepsilon\}$$

Example: Deriving the canonical RFSA for $L = \Sigma^* a \Sigma$

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



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

$L_{q_0} = \Sigma^* a \Sigma$ (initial state)

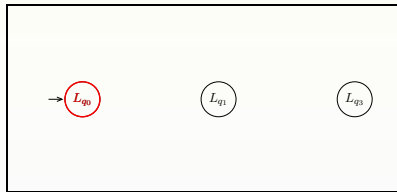
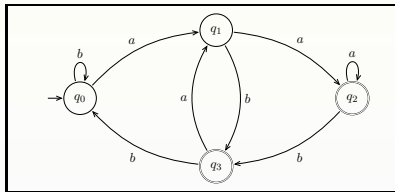
$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$

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

$L_{q_3} = \Sigma^* a \Sigma \cup \{\varepsilon\}$

Example: Deriving the canonical RFSA for $L = \Sigma^* a \Sigma$

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



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

$$L_{q_0} = \Sigma^* a \Sigma \quad (\text{initial state})$$

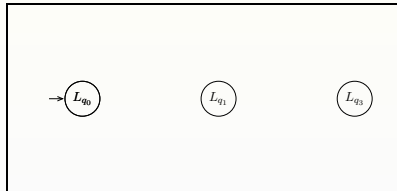
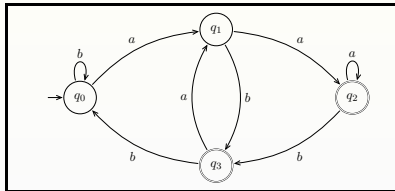
$$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$$

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

$$L_{q_3} = \Sigma^* a \Sigma \cup \{\varepsilon\}$$

Example: Deriving the canonical RFSA for $L = \Sigma^* a \Sigma$

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



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

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

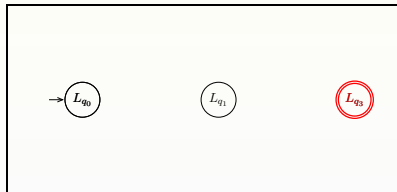
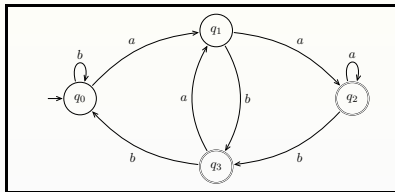
$$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$$

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

$$L_{q_3} = \Sigma^* a \Sigma \cup \{\varepsilon\} \quad (\text{final state})$$

Example: Deriving the canonical RFSA for $L = \Sigma^* a \Sigma$

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



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

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

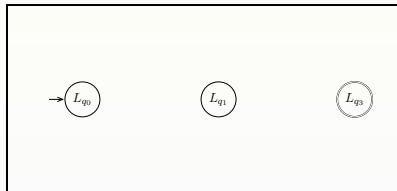
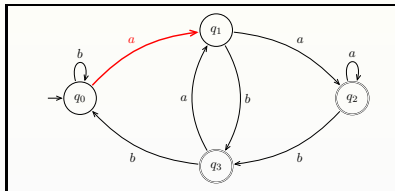
$$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$$

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

$$L_{q_3} = \Sigma^* a \Sigma \cup \{\varepsilon\} \quad (\text{final state})$$

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 $L = \Sigma^* a \Sigma$

$$L_{q_0} = \Sigma^* a \Sigma \quad (a\text{-transitions})$$

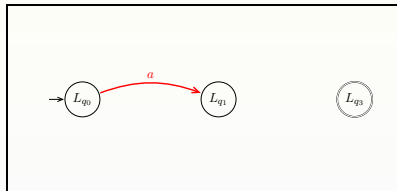
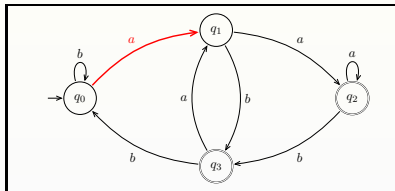
$$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$$

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

$$L_{q_3} = \Sigma^* a \Sigma \cup \{\varepsilon\}$$

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 $L = \Sigma^* a \Sigma$

$$L_{q_0} = \Sigma^* a \Sigma \quad (a\text{-transitions})$$

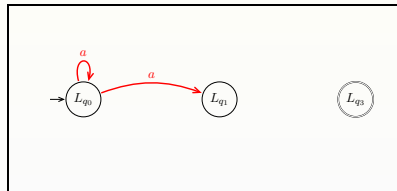
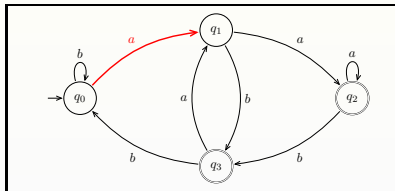
$$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$$

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

$$L_{q_3} = \Sigma^* a \Sigma \cup \{\varepsilon\}$$

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 $L = \Sigma^* a \Sigma$

$$L_{q_0} = \Sigma^* a \Sigma \quad (a\text{-transitions})$$

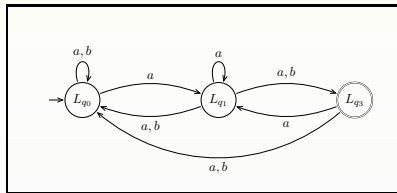
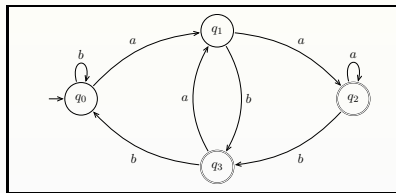
$$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$$

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

$$L_{q_3} = \Sigma^* a \Sigma \cup \{\varepsilon\}$$

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 $L = \Sigma^* a \Sigma$

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

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

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

$$L_{q_3} = \Sigma^* a \Sigma \cup \{\varepsilon\}$$

Designing a table-based learning algorithm

Let $\mathcal{T} = (T, U, V)$ be a table. Find *analogon to union of residuals*

Definition (Join Operator)

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

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

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

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

$\mathcal{T}$	ε	a	aa
ε	-	-	+
a	-	+	+
ab	+	-	+
b	-	-	+
aa	+	+	+
aba	-	+	+
abb	-	-	+

Designing a table-based learning algorithm

Let $\mathcal{T} = (T, U, V)$ be a table. Find *analogon to union of residuals*

Definition (Join Operator)

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

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

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

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

$\mathcal{T}$	ε	a	aa
ε	-	-	+
a	-	+	+
ab	+	-	+
b	-	-	+
aa	+	+	+
aba	-	+	+
abb	-	-	+

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 $r_1, \dots, r_n \in \text{Rows}(\mathcal{T}) \setminus \{r\}$ such that $r = r_1 \sqcup \dots \sqcup r_n$.

$\mathcal{T}$	ε	a	aa
ε	−	−	+
a	−	+	+
ab	+	−	+
b	−	−	+
aa	+	+	+
aba	−	+	+
abb	−	−	+

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 $r_1, \dots, r_n \in \text{Rows}(\mathcal{T}) \setminus \{r\}$ such that $r = r_1 \sqcup \dots \sqcup r_n$.

$\mathcal{T}$	ε	a	aa
ε	−	−	+
a	−	+	+
ab	+	−	+
b	−	−	+
aa	+	+	+
aba	−	+	+
abb	−	−	+

Example

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

$$\text{row}(aa) = (+, +, +) = (-, +, +) \sqcup (+, -, +) = \text{row}(a) \sqcup \text{row}(ab)$$

Designing a table-based learning algorithm

Find analogon to *Composed and prime residuals*

Definition (Composed and Prime Rows)

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

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

$\mathcal{T}$	ε	a	aa
ε	−	−	+
a	−	+	+
ab	+	−	+
b	−	−	+
aa	+	+	+
aba	−	+	+
abb	−	−	+

Designing a table-based learning algorithm

Find analogon to *Composed and prime residuals*

Definition (Composed and Prime Rows)

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

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

$\mathcal{T}$	ε	a	aa
ε	-	-	+
a	-	+	+
ab	+	-	+
b	-	-	+
aa	+	+	+
aba	-	+	+
abb	-	-	+

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

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

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

$\mathcal{T}$	ε	a	aa
ε	—	—	+
a	—	+	+
ab	+	—	+
b	—	—	+
aa	+	+	+
aba	—	+	+
abb	—	—	+

Example

$Primes(\mathcal{T}) = \{row(\varepsilon), row(a), row(ab), row(b), row(aba), row(abb)\}$

Designing a table-based learning algorithm

Find analogon to *Composed and prime residuals*

Definition (Composed and Prime Rows)

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

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

$Primes(\mathcal{T})$: The set of prime rows in $\mathcal{T}$ and
 $Primes_{\text{upp}}(\mathcal{T}) = Primes(\mathcal{T}) \cap Rows_{\text{upp}}(\mathcal{T})$.

$\mathcal{T}$	ε	a	aa
ε	—	—	+
a	—	+	+
ab	+	—	+
b	—	—	+
aa	+	+	+
aba	—	+	+
abb	—	—	+

Example

$$Primes_{\text{upp}}(\mathcal{T}) = \{row(\varepsilon), row(a), row(ab)\}$$

Designing a table-based learning algorithm

Find analogon to *subset relation between residuals*

Definition (Covering Relation)

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

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

$\mathcal{T}$	ε	a	aa
ε	—	—	+
a	—	+	+
ab	+	—	+
b	—	—	+
aa	+	+	+
aba	—	+	+
abb	—	—	+

Example

- e.g., $row(\varepsilon) \sqsubseteq row(a)$ and $row(\varepsilon) \sqsubseteq row(abb)$

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 $v \in V$: $r(v) = + \Rightarrow r'(v) = +$.

$\mathcal{T}$	ε	a	aa
ε	—	—	+
a	—	+	+
ab	+	—	+
b	—	—	+
aa	+	+	+
aba	—	+	+
abb	—	—	+

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 $r \sqsubset r'$.

$\mathcal{T}$	ε	a	aa
ε	—	—	+
a	—	+	+
ab	+	—	+
b	—	—	+
aa	+	+	+
aba	—	+	+
abb	—	—	+

Example

- e.g., $\text{row}(\varepsilon) \sqsubseteq \text{row}(a)$ and $\text{row}(\varepsilon) \sqsubseteq \text{row}(abb)$
- e.g., $\text{row}(\varepsilon) \sqsubset \text{row}(ab)$

Table properties

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

$\mathcal{T}$	ε	a	aa
ε	−	−	+
a	−	+	+
ab	+	−	+
b	−	−	+
aa	+	+	+
aba	−	+	+
abb	−	−	+

RFSA-Closedness

- all states identifiable from the table

Table properties

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

$\mathcal{T}$	ε	a	aa
ε	−	−	+
a	−	+	+
ab	+	−	+
b	−	−	+
aa	+	+	+
aba	−	+	+
abb	−	−	+

RFSA-Closedness

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

Table properties

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

$\mathcal{T}$	ε	a	aa
ε	-	-	+
a	-	+	+
ab	+	-	+
b	-	-	+
aa	+	+	+
aba	-	+	+
abb	-	-	+

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 properties

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

$\mathcal{T}$	ε	a	aa
ε	—	—	+
a	—	+	+
ab	+	—	+
b	—	—	+
aa	+	+	+
aba	—	+	+
abb	—	—	+

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^*

$\mathcal{T}$	ε	a	aa
ε	-	-	+
a	-	+	+
ab	+	-	+
b	-	-	+
aa	+	+	+
aba	-	+	+
abb	-	-	+

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

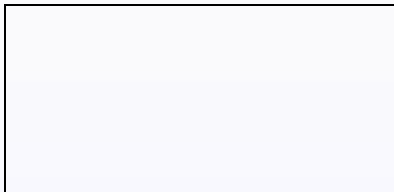
From Table to NFA

Definition (NFA of a Table)

For a table $\mathcal{T} = (T, U, V)$ 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
- $\delta(\text{row}(u), a) = \{r \in Q \mid r \sqsubseteq \text{row}(ua)\}$ ($u \in U$, $\text{row}(u) \in Q$, $a \in \Sigma$).

$\mathcal{T}$	ε	a	aa
* ε	−	−	+
* a	−	+	+
* ab	+	−	+
* b	−	−	+
aa	+	+	+
* aba	−	+	+
* abb	−	−	+



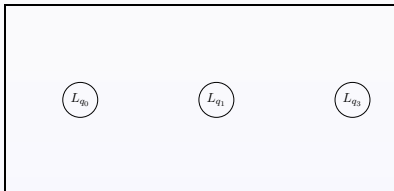
From Table to NFA

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
- $\delta(\text{row}(u), a) = \{r \in Q \mid r \sqsubseteq \text{row}(ua)\}$ ($u \in U$, $\text{row}(u) \in Q$, $a \in \Sigma$).

$\mathcal{T}$	ε	a	aa
* ε	−	−	+
* a	−	+	+
* ab	+	−	+
* b	−	−	+
aa	+	+	+
* aba	−	+	+
* abb	−	−	+



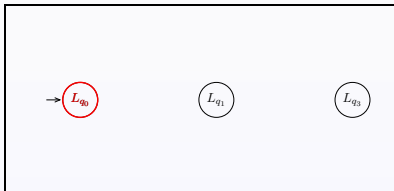
From Table to NFA

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
- $\delta(\text{row}(u), a) = \{r \in Q \mid r \sqsubseteq \text{row}(ua)\}$ ($u \in U$, $\text{row}(u) \in Q$, $a \in \Sigma$).

$\mathcal{T}$	ε	a	aa
* ε	—	—	+
* a	—	+	+
* ab	+	—	+
* b	—	—	+
aa	+	+	+
* aba	—	+	+
* abb	—	—	+



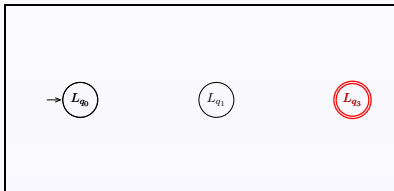
From Table to NFA

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
- $\delta(\text{row}(u), a) = \{r \in Q \mid r \sqsubseteq \text{row}(ua)\}$ ($u \in U$, $\text{row}(u) \in Q$, $a \in \Sigma$).

$\mathcal{T}$	ε	a	aa
* ε	−	−	+
* a	−	+	+
* ab	+	−	+
* b	−	−	+
aa	+	+	+
* aba	−	+	+
* abb	−	−	+



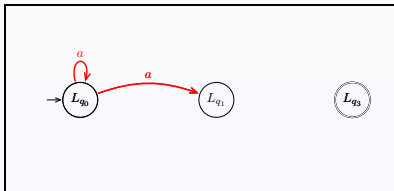
From Table to NFA

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
- $\delta(\text{row}(u), a) = \{r \in Q \mid r \sqsubseteq \text{row}(ua)\}$ ($u \in U$, $\text{row}(u) \in Q$, $a \in \Sigma$).

$\mathcal{T}$	ε	a	aa
* ε	—	—	+
* a	—	+	+
* ab	+	—	+
* b	—	—	+
aa	+	+	+
* aba	—	+	+
* abb	—	—	+



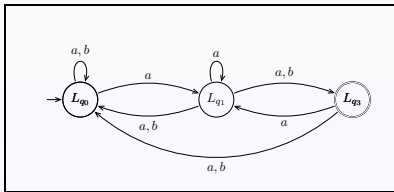
From Table to NFA

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
- $\delta(\text{row}(u), a) = \{r \in Q \mid r \sqsubseteq \text{row}(ua)\}$ ($u \in U$, $\text{row}(u) \in Q$, $a \in \Sigma$).

$\mathcal{T}$	ε	a	aa
* ε	—	—	+
* a	—	+	+
* ab	+	—	+
* b	—	—	+
aa	+	+	+
* aba	—	+	+
* abb	—	—	+



Summarizing: Tables in NL^*

$\mathcal{T}$	ε	a	aa
ε	−	−	+
a	−	+	+
ab	+	−	+
b	−	−	+
aa	+	+	+
aba	−	+	+
abb	−	−	+

From tables to RFSA

- we deal with tables

Summarizing: Tables in NL^*

$\mathcal{T}$	ε	a	aa
ε	−	−	+
a	−	+	+
ab	+	−	+
b	−	−	+
aa	+	+	+
aba	−	+	+
abb	−	−	+

From tables to RFSA

- we deal with tables
- table rows approximate residual languages

Summarizing: Tables in NL*

$\mathcal{T}$	ε	a	aa
ε	—	—	+
a	—	+	+
ab	+	—	+
b	—	—	+
aa	+	+	+
aba	—	+	+
abb	—	—	+

From tables to RFSA

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

Summarizing: Tables in NL*

$\mathcal{T}$	ε	a	aa
ε	-	-	+
a	-	+	+
ab	+	-	+
b	-	-	+
aa	+	+	+
aba	-	+	+
abb	-	-	+

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

Summarizing: Tables in NL*

$\mathcal{T}$	ε	a	aa
ε	-	-	+
a	-	+	+
ab	+	-	+
b	-	-	+
aa	+	+	+
aba	-	+	+
abb	-	-	+

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*

$\mathcal{T}$	ε	a	aa
ε	-	-	+
a	-	+	+
ab	+	-	+
b	-	-	+
aa	+	+	+
aba	-	+	+
abb	-	-	+

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*

$\mathcal{T}$	ε	a	aa
ε	-	-	+
a	-	+	+
ab	+	-	+
b	-	-	+
aa	+	+	+
aba	-	+	+
abb	-	-	+

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^*

$\mathcal{T}$	ε	a	aa	...
ε	−	−	+	...
a	−	+	+	...
ab	+	−	+	...
b	−	−	+	...
aa	+	+	+	...
aba	−	+	+	...
abb	−	−	+	...

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

Definition (Consistency with a table)

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

Definition (Consistency with a table)

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

Theorem (Correctness)

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

Theorem (Complexity of NL^*)

Let:

- n : number of states of minimal DFA $\mathcal{A}_L$ for regular language L ,
- m : length of the biggest counterexample

Then, NL^* returns after at most:

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

Theorem (Complexity of NL^*)

Let:

- n : number of states of minimal DFA $\mathcal{A}_L$ 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 $\mathcal{R}(L)$.

Theorem (Complexity of NL^*)

Let:

- n : number of states of minimal DFA $\mathcal{A}_L$ 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 $\mathcal{R}(L)$.

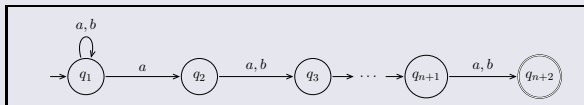
It's worth considering RFSA...

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**.*

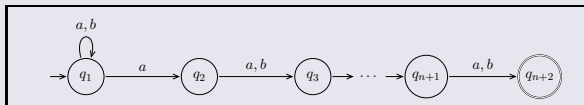
It's worth considering RFSA...

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

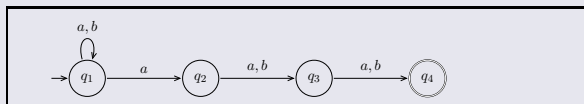


It's worth considering RFSA...

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

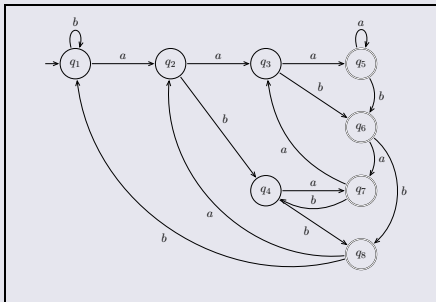


$$L_2 = \{w \in \Sigma^* \mid w \text{ has an } a \text{ at the 3}^{\text{rd}}\text{-last position}\}$$



Minimal DFA and RFSA

Minimal DFA and RFSA for L_2 :

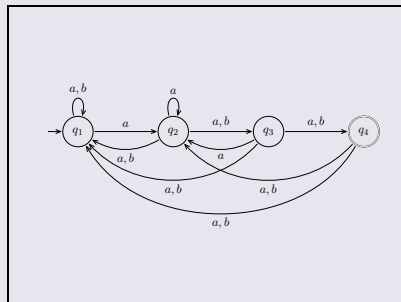
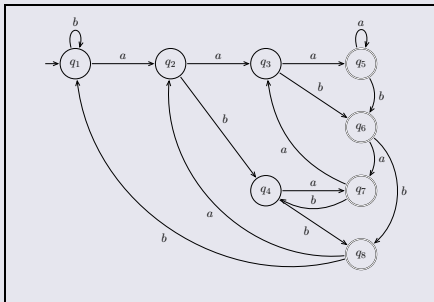


Automata for language L_n :

- 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

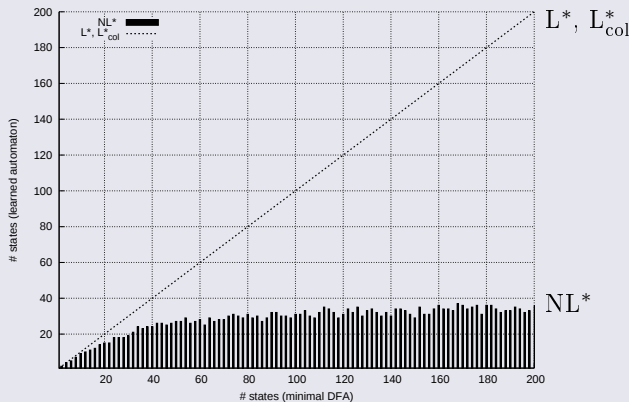
Comparison of L^* , L_{col}^* , and NL^*

	Equivalence queries	Membership queries	Treatment of counterexamples
L^*	n	$\mathcal{O}(m \Sigma n^2)$	to rows
L_{col}^*	n	$\mathcal{O}(m \Sigma n^2)$	to columns
NL^*	$\mathcal{O}(n^2)$	$\mathcal{O}(m \Sigma n^3)$	to columns

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

Algorithm - Overview

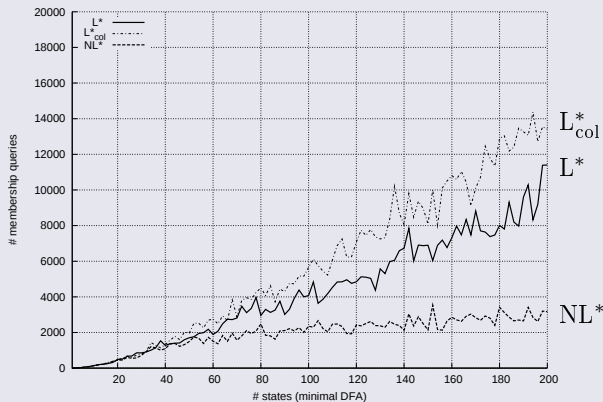
Number of states (L^* , L_{col}^* vs. NL^*)



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

Algorithm - Overview

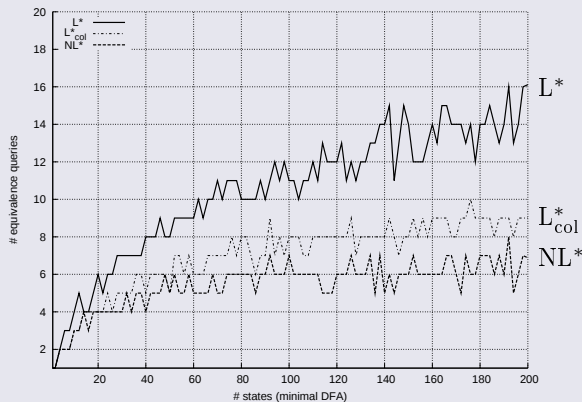
Number of membership queries (L^* vs. L_{col}^* vs. NL^*)



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

Algorithm - Overview

Number of equivalence queries (L^* vs. L_{col}^* vs. NL^*)



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

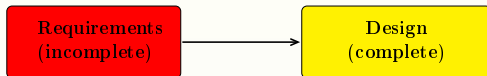
Presentation outline

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

Requirements (incomplete)

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

Motivation



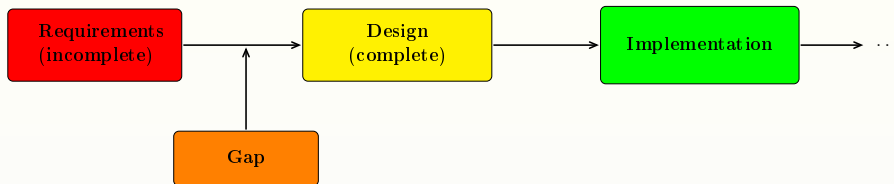
- initial phase: requirement elicitation
 - contradicting or incomplete system description
 - common description language: sequence diagrams
- goal: conforming design model

Motivation

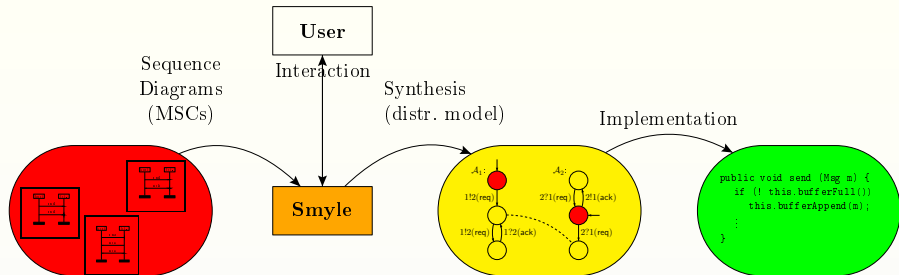


- initial phase: requirement elicitation
 - contradicting or incomplete system description
 - common description language: sequence diagrams
- goal: conforming design model

Motivation



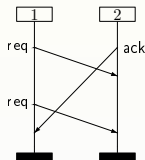
- initial phase: requirement elicitation
 - contradicting or incomplete system description
 - common description language: sequence diagrams
- goal: conforming design model
- closing gap between
 - requirement specification (usually incomplete) and
 - design model (complete description of system)



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

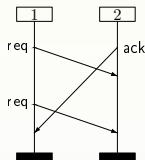
Message Sequence Chart



An MSC $M = \langle \mathcal{P}, E, \{\leq_p\}_{p \in \mathcal{P}}, <_{msg}, l \rangle$

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

Message Sequence Chart



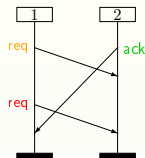
An MSC $M = \langle \mathcal{P}, E, \{\leq_p\}_{p \in \mathcal{P}}, <_{msg}, l \rangle$

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

A set of MSCs is called an *MSC language*

A *linearization* of an MSC is a total ordering of E subsuming $\leq$

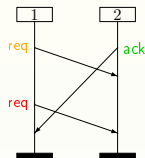
MSCs and Linearizations



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)
- ...

MSCs and Linearizations

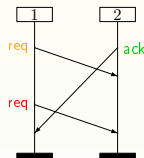


Some linearizations

- 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 $M = \text{MSC}(w)$ is uniquely determined by any $w \in \text{Lin}(M)$

MSCs and Linearizations



Some linearizations

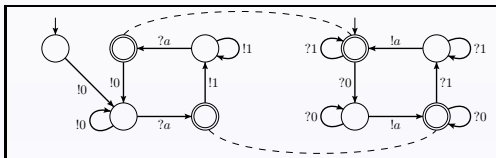
- 1!2(req) 1!2(req) 2!1(ack) 1?2(ack) 2?1(req) 2?1(req)
- 1!2(req) 2!1(ack) 1!2(req) 1?2(ack) 2?1(req) 2?1(req)
- 2!1(ack) 1!2(req) 2?1(req) 1!2(req) 2?1(req) 1?2(ack)
- ...

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

Communicating Finite-State Machines (CFM)

A CFM consists of:

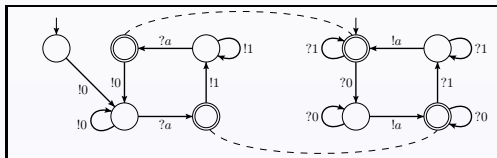
- a set of finite-state automata (*processes*) with
 - common global initial state
 - set of global final states



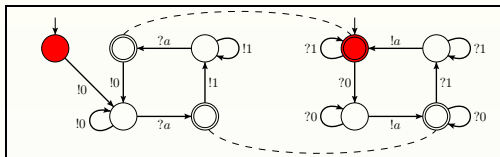
Communicating Finite-State Machines (CFM)

A CFM consists of:

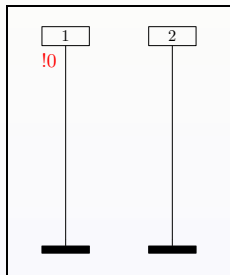
- a set of finite-state automata (*processes*) with
 - common global initial state
 - set of global final states
- communication between automata through (reliable) FIFO channels
 - $p!q(a)$ appends message a to buffer between p and q
 - $q?p(a)$ removes message a from buffer between p and q



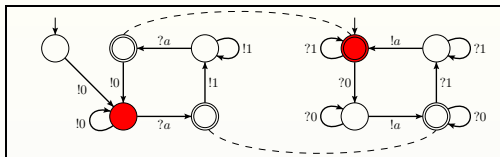
CFM: An Example



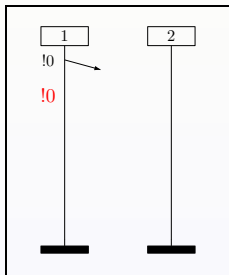
buffer head



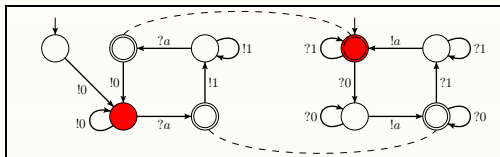
CFM: An Example



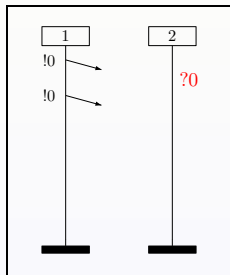
buffer head



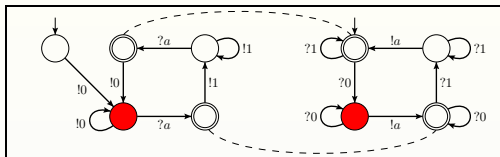
CFM: An Example



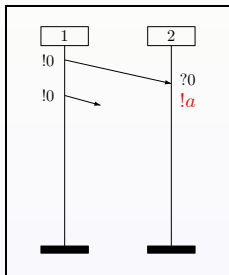
buffer head



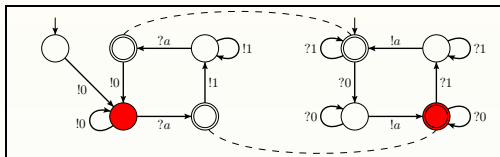
CFM: An Example



buffer head



CFM: An Example



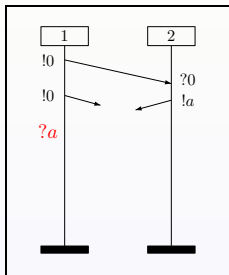
0

 $1 \rightarrow 2$

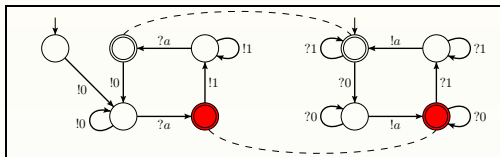
buffer head

a

 $2 \rightarrow 1$



CFM: An Example



0

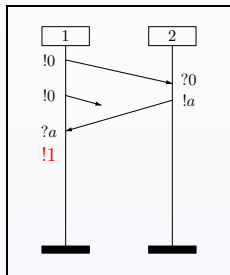
--	--	--	--	--

 $1 \rightarrow 2$

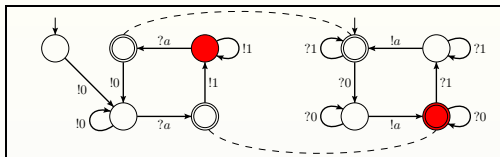
buffer head

--	--	--	--	--

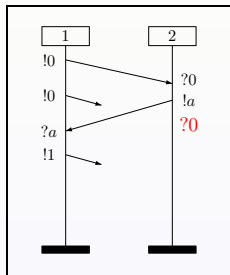
 $2 \rightarrow 1$



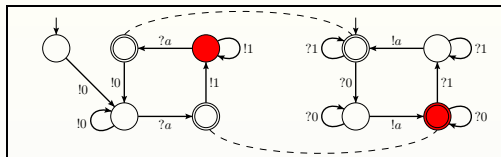
CFM: An Example



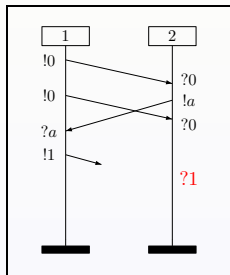
buffer head



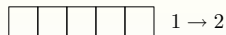
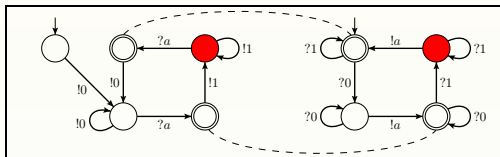
CFM: An Example



buffer head



CFM: An Example

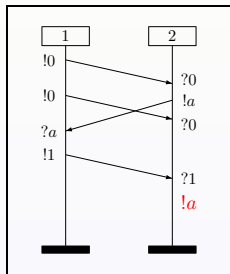


$1 \rightarrow 2$

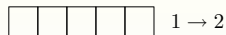
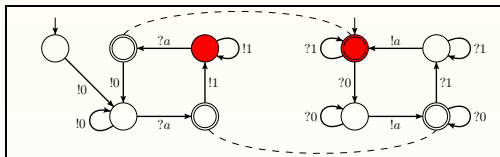
buffer head



$2 \rightarrow 1$



CFM: An Example

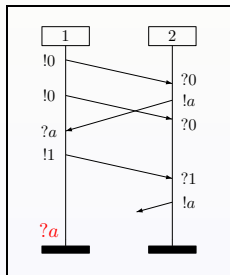


$1 \rightarrow 2$

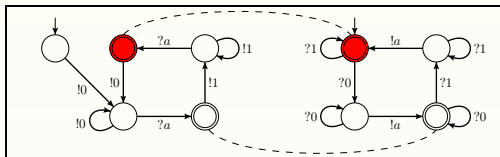
buffer head



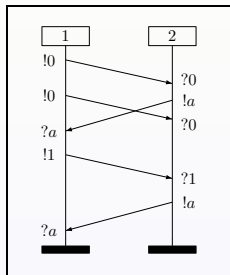
$2 \rightarrow 1$



CFM: An Example



buffer head

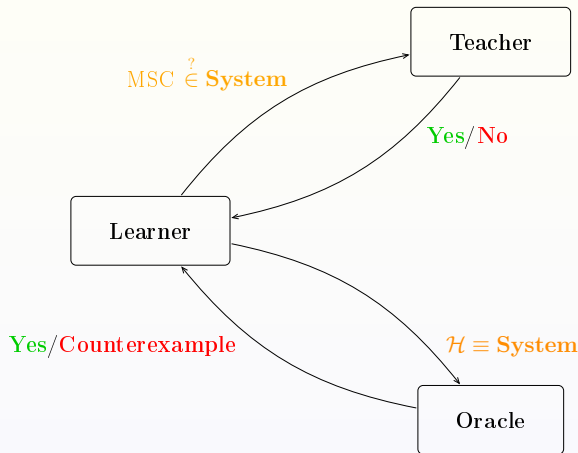


Current State

Current State

- **given:** *learning DFA* [Angluin]
- **goal:** *learning CFMs*

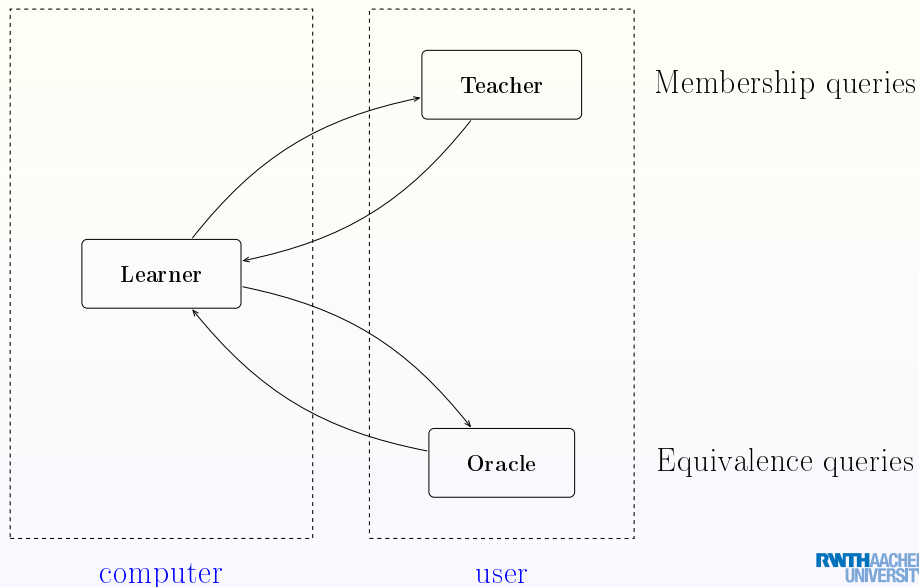
The learning algorithm (extension of Angluin's L^*)



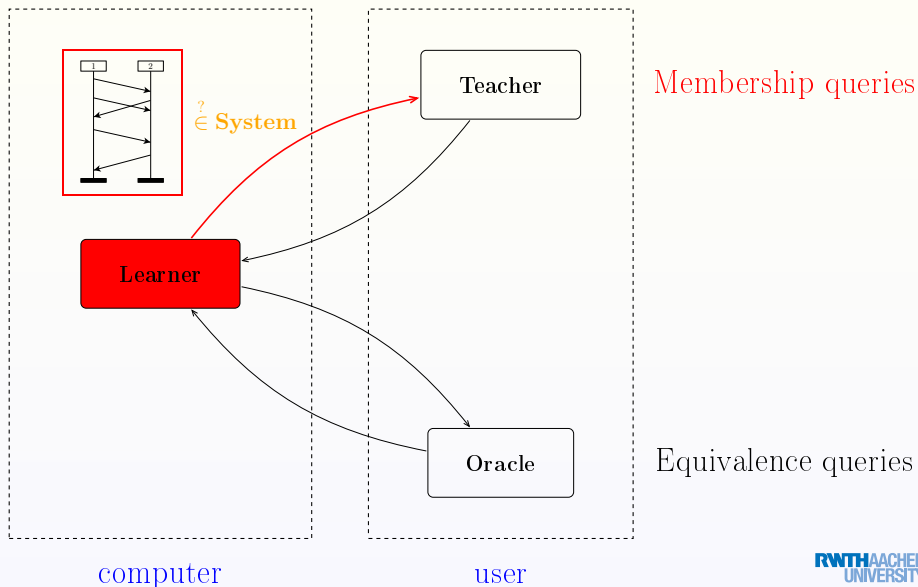
Membership queries

Equivalence queries

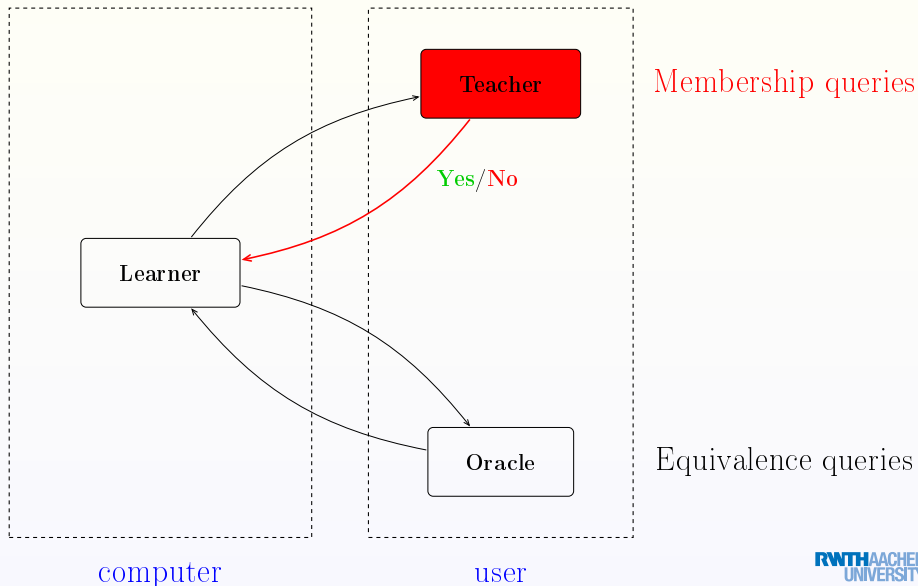
The learning algorithm (extension of Angluin's L^*)



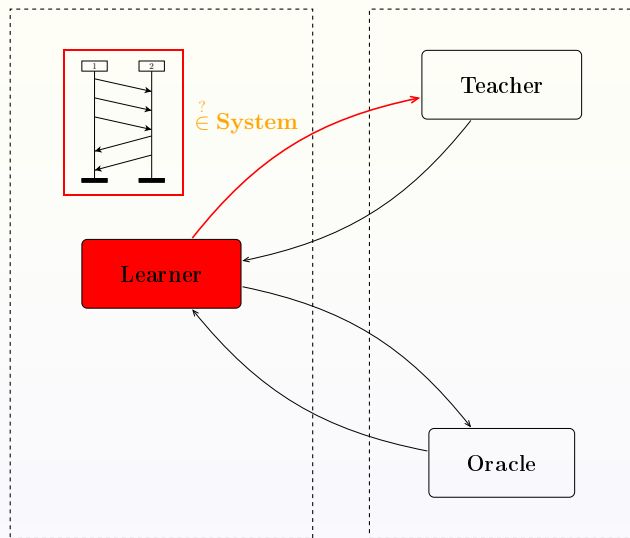
The learning algorithm (extension of Angluin's L^*)



The learning algorithm (extension of Angluin's L^*)



The learning algorithm (extension of Angluin's L^*)



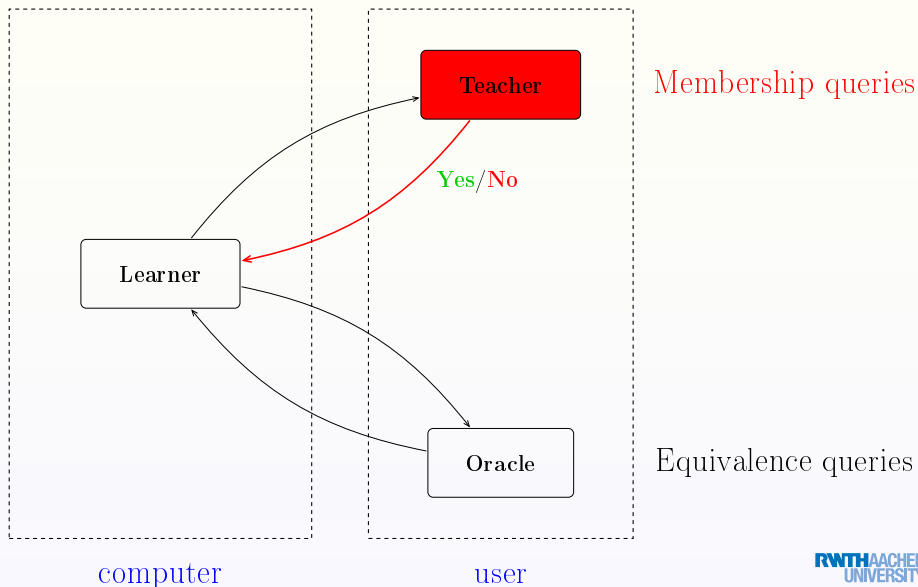
Membership queries

Equivalence queries

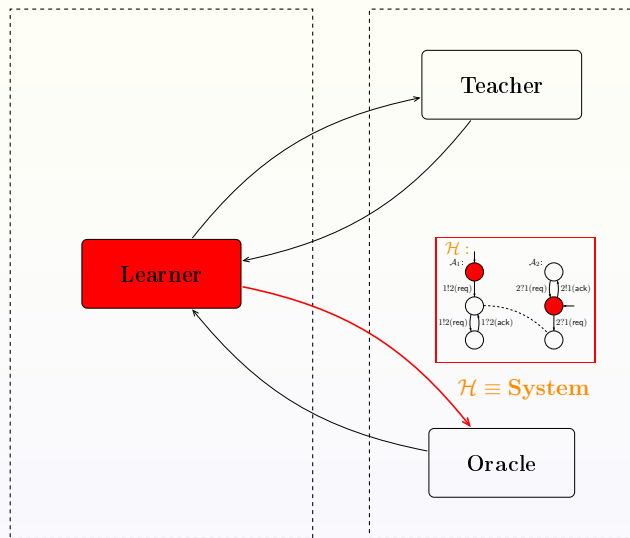
computer

user

The learning algorithm (extension of Angluin's L^*)



The learning algorithm (extension of Angluin's L^*)



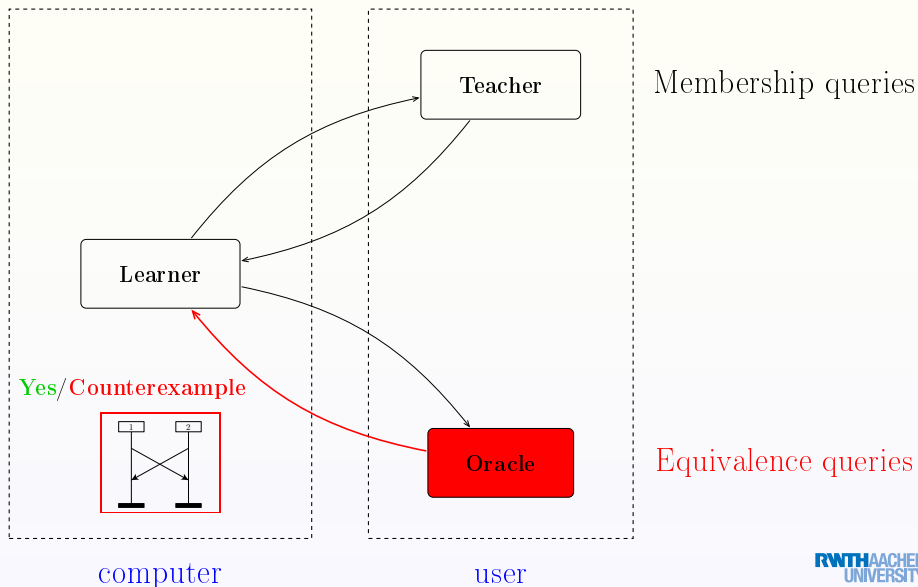
Membership queries

Equivalence queries

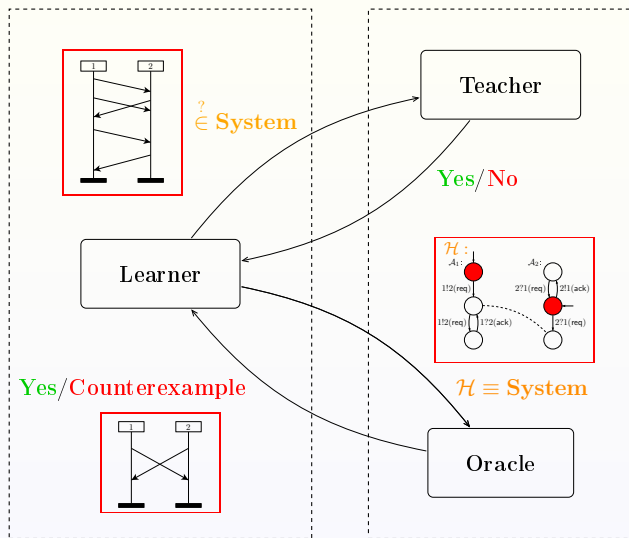
computer

user

The learning algorithm (extension of Angluin's L^*)



The learning algorithm (extension of Angluin's L^*)



Membership queries

Equivalence queries

computer

user

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

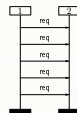
Classes of MSCs

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*

- $\mathcal{D}$ domain over ($\forall/\exists B$ -bounded) MSC linearizations
- $\approx$: equivalence of ($\forall/\exists B$ -bounded) linearizations
- *synth* : Synthesis function from DFA to ($\forall/\exists - B$ -bounded) CFMs

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.

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*)

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 $\mathcal{H}$

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 $w \in L(\mathcal{H}) \setminus \mathcal{D}$

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 $w \approx w'$: $w \in L(\mathcal{H})$, $w' \notin L(\mathcal{H})$ and
 - perform membership query for $\text{MSC}(w)$

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 $w \approx w'$: $w \in L(\mathcal{H})$, $w' \notin L(\mathcal{H})$ and
 - perform membership query for $MSC(w)$

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 *synth.*

Results:

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

Learnable classes of CFMs:

- (deterministic) $\forall$ -bounded CFMs
- $\exists B$ -bounded CFMs ($B \in \mathbb{N}$)
- deterministic $\forall$ -bounded deadlock-free weak CFMs

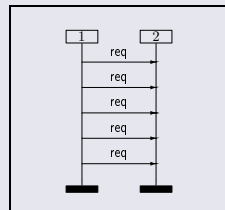
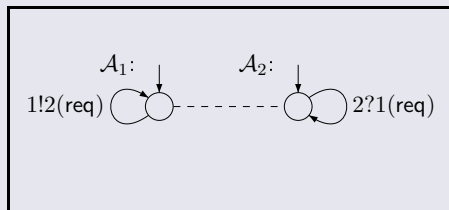
Not learnable (in a guided fashion)

- $\forall$ -bounded weak CFMs

existentially B -bounded CFMs

An *existentially B -bounded* CFM

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



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

Algorithm for $\exists B$ -bounded CFMs

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

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

Algorithm for $\exists B$ -bounded CFMs

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

Problems $L(\mathcal{H}) \subseteq \mathcal{D}$ 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 $\leq B$

Algorithm for $\exists B$ -bounded CFMs

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

Problems $L(\mathcal{H}) \subseteq \mathcal{D}$ 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

Algorithm for $\exists B$ -bounded CFMs

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

Problems $L(\mathcal{H}) \subseteq \mathcal{D}$ 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 σ, τ

Algorithm for $\exists B$ -bounded CFMs

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

Problems $L(\mathcal{H}) \subseteq \mathcal{D}$ 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 σ, τ

- 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 $L(\mathcal{H}) \subseteq \mathcal{D}$ 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 σ, τ

- 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 $\mathcal{H}$

Number of equivalence queries:

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

Not learnable (in a supported fashion)

- $\forall$ -bounded weak CFMs

Some results

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

Some results

Protocol	#membership queries			#user queries	#equivalence queries	$ \mathcal{H} $	#rows in table			learning setup
	w.o. POL	w. POL	savings				w.o. POL	w. POL	reduction	
part of <i>USB 1.1</i>	488	200	59.0%	14	1 (5)	9	61	26	57.4%	$\exists 2$
<i>continuous update</i>	712	264	62.9%	21	1 (3)	8	89	34	61.8%	$\exists 1$
<i>negotiation</i>	1,179	432	63.4%	31	1 (3)	9	131	49	62.6%	$\exists 1$
<i>ABP</i>	2,286	697	69.5%	64	2 (4)	15	127	42	66.9%	$\exists 1$
<i>ABP</i>	14,432	4,557	68.4%	158	2 (13)	25	451	131	71.0%	$\exists 2$
<i>ABP</i>	55,131	19,252	65.1%	407	2 (22)	37	799	222	72.2%	$\exists 3$
<i>leader elec. (v₁)</i>	3,612	900	75.1%	43	1 (2)	13	301	76	74.8%	$\forall$
<i>leader elec. (v₂)</i>	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	$ \mathcal{H} $	#rows in table			learning setup
	w.o. POL	w. POL	savings				w.o. POL	w. POL	reduction	
part of <i>USB 1.1</i>	488	200	59.0%	14	1 (5)	9	61	26	57.4%	$\exists 2$
<i>continuous update</i>	712	264	62.9%	21	1 (3)	8	89	34	61.8%	$\exists 1$
<i>negotiation</i>	1,179	432	63.4%	31	1 (3)	9	131	49	62.6%	$\exists 1$
<i>ABP</i>	2,286	697	69.5%	64	2 (4)	15	127	42	66.9%	$\exists 1$
<i>ABP</i>	14,432	4,557	68.4%	158	2 (13)	25	451	131	71.0%	$\exists 2$
<i>ABP</i>	55,131	19,252	65.1%	407	2 (22)	37	799	222	72.2%	$\exists 3$
<i>leader elec. (v_1)</i>	3,612	900	75.1%	43	1 (2)	13	301	76	74.8%	$\forall$
<i>leader elec. (v_2)</i>	14,704	6,864	53.3%	196	2 (5)	17	919	430	53.2%	$\forall$

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

Some results

Protocol	#membership queries			#user queries	#equivalence queries	$ \mathcal{H} $	#rows in table			learning setup
	w.o. POL	w. POL	savings				w.o. POL	w. POL	reduction	
part of <i>USB 1.1</i>	488	200	59.0%	14	1 (5)	9	61	26	57.4%	$\exists 2$
<i>continuous update</i>	712	264	62.9%	21	1 (3)	8	89	34	61.8%	$\exists 1$
<i>negotiation</i>	1,179	432	63.4%	31	1 (3)	9	131	49	62.6%	$\exists 1$
<i>ABP</i>	2,286	697	69.5%	64	2 (4)	15	127	42	66.9%	$\exists 1$
<i>ABP</i>	14,432	4,557	68.4%	158	2 (13)	25	451	131	71.0%	$\exists 2$
<i>ABP</i>	55,131	19,252	65.1%	407	2 (22)	37	799	222	72.2%	$\exists 3$
<i>leader elec. (v₁)</i>	3,612	900	75.1%	43	1 (2)	13	301	76	74.8%	$\forall$
<i>leader elec. (v₂)</i>	14,704	6,864	53.3%	196	2 (5)	17	919	430	53.2%	$\forall$

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

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

Features

- implements wide range of learning algorithms:
 L^* , L^*_{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 $\forall/\exists - B/\dots$ 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 $\forall/\exists - B/\dots$ 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 $\forall/\exists - B/\dots$ 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>

Smile

The screenshot shows the Smite software interface with several components and numbered annotations:

- 1**: Points to the "Learning setup" section in the "Example No: 1" pane, which includes settings like "Type of automaton to learn: EXISTENTIALLY_BOUNDED", "Learning Bound b = 8", and "Always add all linearizations of a given NSC: true".
- 2**: Points to the "user queries" and "membership queries" tabs in the top right pane.
- 3**: Points to the "diagram" section at the bottom of the interface, which displays a graph with nodes and edges.
- 4**: Points to the "Cutting new counter examples" section in the "Example No: 1" pane, which shows a sequence of nodes (p, c, b, a, e, n) and their corresponding actions (a, b, c, d, e, f).
- 5**: Points to the "PDL Filter conforms action of last matching f" dialog box, which contains a table with "Action" and "Formula" columns.

The "PDL Filter conforms action of last matching f" dialog box contains the following data:

Action	Formula
path=direct	TRUE

The "diagram" section at the bottom displays a graph with nodes and edges, and a list of queries:

```

1 [6], 2 [fontsize=16], 3 [fontsize=16], 4 [fontsize=16], 5
->8 [label="c?b?d?"] 6->7 [label="p?b?d?"] 0->10 [label="p?c?d?"]
[label="p?c?d?"] 7->6 [label="p?c?d?"] 4->5 [label="c?b?d?"]
  
```

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
- ...

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)
- ...

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 $r \in Rows_{low}(\mathcal{T})$,

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

$\mathcal{T}$	ε	a	aa
ε	−	−	+
a	−	+	+
ab	+	−	+
b	−	−	+
aa	+	+	+
aba	−	+	+
abb	−	−	+

Designing a table-based learning algorithm

Definition (RFSA-Closedness)

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

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

$\mathcal{T}$	ε	a	aa
ε	−	−	+
a	−	+	+
ab	+	−	+
b	−	−	+
aa	+	+	+
aba	−	+	+
abb	−	−	+

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

- $row(aa) = row(\varepsilon) \sqcup row(a) \sqcup row(ab)$ and

Designing a table-based learning algorithm

Definition (RFSA-Closedness)

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

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

$\mathcal{T}$	ε	a	aa
ε	−	−	+
a	−	+	+
ab	+	−	+
b	−	−	+
aa	+	+	+
aba	−	+	+
abb	−	−	+

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

- $row(aa) = row(\varepsilon) \sqcup row(a) \sqcup row(ab)$ and
- $row(b), row(aba), row(abb) \in Primes_{upp}(\mathcal{T})$

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 $a \in \Sigma$:

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

$\mathcal{T}$	ε	a	aa
ε	−	−	+
a	−	+	+
ab	+	−	+
b	−	−	+
aa	+	+	+
aba	−	+	+
abb	−	−	+

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 $a \in \Sigma$:

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

$\mathcal{T}$	ε	a	aa
ε	−	−	+
a	−	+	+
ab	+	−	+
b	−	−	+
aa	+	+	+
aba	−	+	+
abb	−	−	+

$\mathcal{T}$ is RFSA-consistent

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

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 $a \in \Sigma$:

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

$\mathcal{T}$	ε	a	aa
ε	−	−	+
a	−	+	+
ab	+	−	+
b	−	−	+
aa	+	+	+
aba	−	+	+
abb	−	−	+

$\mathcal{T}$ is RFSA-consistent

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

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 $a \in \Sigma$:

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

$\mathcal{T}$	ε	a	aa
ε	−	−	+
a	−	+	+
ab	+	−	+
b	−	−	+
aa	+	+	+
aba	−	+	+
abb	−	−	+

$\mathcal{T}$ is RFSA-consistent

- $\text{row}(\varepsilon) \sqsubseteq \text{row}(a)$:
 - $\text{row}(a) \sqsubseteq \text{row}(aa)$ and
 - $\text{row}(b) \sqsubseteq \text{row}(ab)$
- $\text{row}(\varepsilon) \sqsubseteq \text{row}(ab)$:
 - $\text{row}(a) \sqsubseteq \text{row}(aba)$ and
 - $\text{row}(b) \sqsubseteq \text{row}(abb)$

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 $a \in \Sigma$:

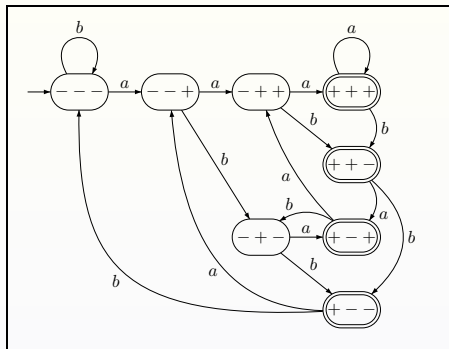
$$row(u') \sqsubseteq row(u) \Rightarrow row(u'a) \sqsubseteq row(ua)$$

$\mathcal{T}$	ε	a	aa
ε	—	—	+
a	—	+	+
ab	+	—	+
b	—	—	+
aa	+	+	+
aba	—	+	+
abb	—	—	+

$\mathcal{T}$ is RFSA-consistent

- $row(\varepsilon) \sqsubseteq row(a)$:
 - $row(a) \sqsubseteq row(aa)$ and
 - $row(b) \sqsubseteq row(ab)$
- $row(\varepsilon) \sqsubseteq row(ab)$:
 - $row(a) \sqsubseteq row(aba)$ and
 - $row(b) \sqsubseteq row(abb)$

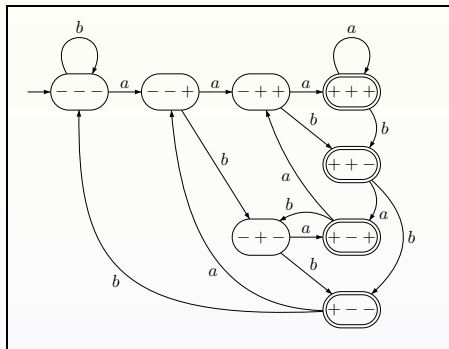
The algorithm in action



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

	$\mathcal{T}_1$	ϵ
*	ϵ	—
*	a	—
*	b	—

The algorithm in action



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

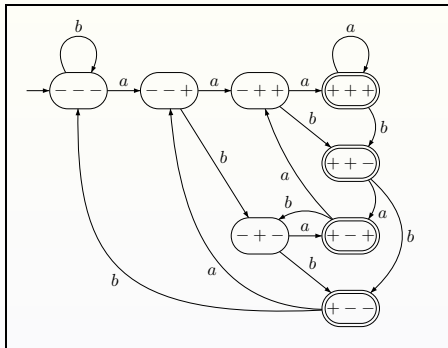


$\Rightarrow$ Counterexample is aaa .

$\Rightarrow$ Add $Suff(aaa)$ to V .

	T_1	ϵ
*	ϵ	—
*	a	—
*	b	—

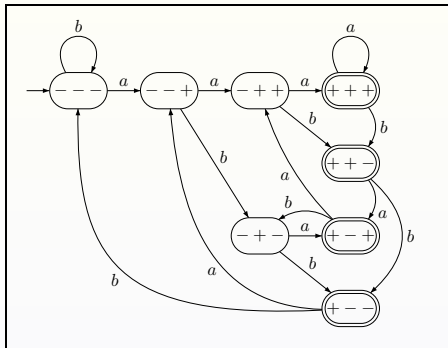
The algorithm in action



T_1		ϵ				
*	ϵ	-				
*	a	-				
*	b	-				

T_2		ϵ	aaa	aa	a
*	ϵ	-	+	-	-
*	a	-	+	+	-
*	b	-	+	-	-

The algorithm in action

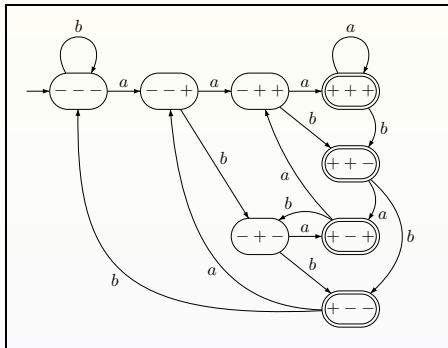


$\mathcal{T}_2$ is not closed:
 $row(a) \notin Primes_{\text{upp}}$

$\mathcal{T}_1$	ϵ
*	ϵ
*	ϵ
*	ϵ
*	ϵ

$\mathcal{T}_2$	ϵ	aaa	aa	a
*	ϵ	-	+	-
*	ϵ	-	+	-
*	ϵ	-	+	-
*	ϵ	-	+	-

The algorithm in action

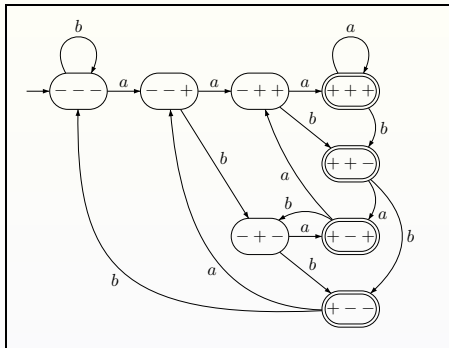


T_1	ϵ
* ϵ	-
* a	-
* b	-

T_2	ϵ	aaa	aa	a
* ϵ	-	+	-	-
* a	-	+	+	-
* b	-	+	-	-

T_3	ϵ	aaa	aa	a
* ϵ	-	+	-	-
* a	-	+	+	-
* b	-	+	-	-
* aa	-	+	+	+
* ab	-	+	-	+

The algorithm in action



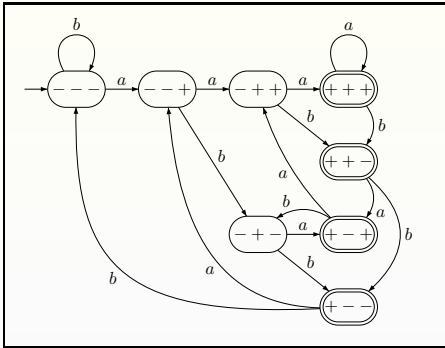
$\mathcal{T}_3$ is not closed:
 $\text{row}(ab) \notin \text{Primes}_{\text{upp}}$

$\mathcal{T}_1$	ϵ
* ϵ	-
* a	-
* b	-

$\mathcal{T}_2$	ϵ	aaa	aa	a
* ϵ	-	+	-	-
* a	-	+	+	-
* b	-	+	-	-

$\mathcal{T}_3$	ϵ	aaa	aa	a
* ϵ	-	+	-	-
* a	-	+	+	-
* b	-	+	-	-
aa	-	+	+	+
* ab	-	+	-	+

The algorithm in action



	$\mathcal{T}_1$	ϵ
*	ϵ	—

*	a	—
*	b	—

$\mathcal{T}_2$	ϵ	aaa	aa	a
*	ϵ	—	+	—
*	a	—	+	—
*	b	—	+	—

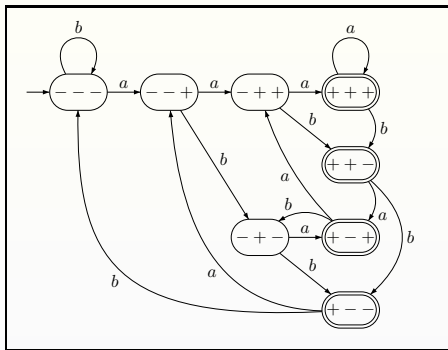
	$\mathcal{T}_3$	ϵ	aaa	aa	a
*	ϵ	—	+	—	—
*	a	—	+	+	—

*	b	—	+	—	—
	aa	—	+	+	+
*	ab	—	+	—	+

	$\mathcal{T}_4$	ε	aaa	aa	a
*	ε	—	+	—	—
*	a	—	+	+	—
*	ab	—	+	—	+

*	b	—	+	—	—
	aa	—	+	+	+
	aba	+	+	+	—
*	abb	+	+	—	—

The algorithm in action



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

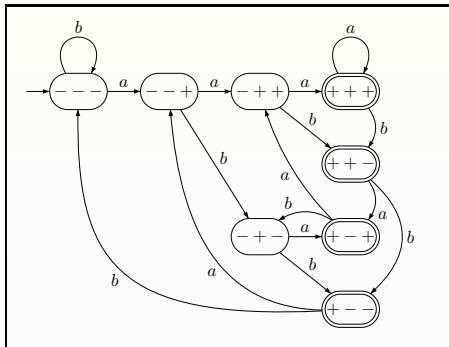
$\mathcal{T}_1$	ε
* ε	-
* a	-
* b	-

$\mathcal{T}_2$	ε	aaa	aa	a
* ε	-	+	-	-
* a	-	+	+	-
* b	-	+	-	-

$\mathcal{T}_3$	ε	aaa	aa	a
* ε	-	+	-	-
* a	-	+	+	-
* b	-	+	-	-
aa	-	+	+	+
* ab	-	+	-	+

$\mathcal{T}_4$	ε	aaa	aa	a
* ε	-	+	-	-
* a	-	+	+	-
* ab	-	+	-	+
* b	-	+	-	-
aa	-	+	+	+
aba	+	+	+	-
* abb	+	+	-	-

The algorithm in action



T_1	ϵ
* ϵ	-
* a	-
* b	-

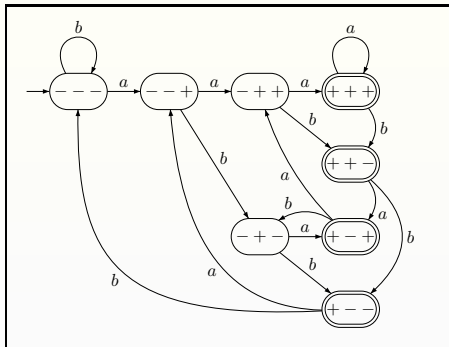
T_2	ϵ	aaa	aa	a
* ϵ	-	+	-	-
* a	-	+	+	-
* b	-	+	-	-

T_3	ϵ	aaa	aa	a
* ϵ	-	+	-	-
* a	-	+	+	-
* b	-	+	-	-
* aa	-	+	+	+
* ab	-	+	-	+

T_4	ϵ	aaa	aa	a
* ϵ	-	+	-	-
* a	-	+	+	-
* ab	-	+	-	+
* b	-	+	-	-
* aa	-	+	+	+
* aba	+	+	+	-
* abb	+	+	-	-

T_5	ϵ	aaa	aa	a
* ϵ	-	+	-	-
* a	-	+	+	-
* ab	-	+	-	+
* abb	+	+	-	-
* b	-	+	-	-
* aa	-	+	+	+
* aba	+	+	+	-
* abba	-	+	+	-
* abbb	-	+	-	-

The algorithm in action



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

$\mathcal{T}_1$	ϵ
* ϵ	-
* a	-
* b	-

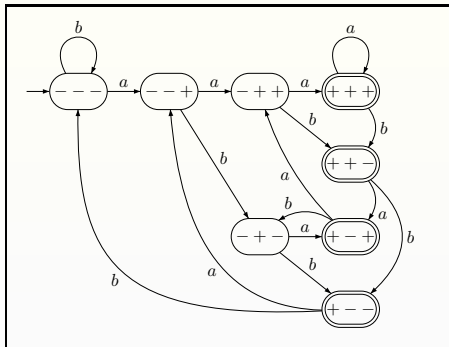
$\mathcal{T}_2$	ϵ	aaa	aa	a
* ϵ	-	+	-	-
* a	-	+	+	-
* b	-	+	-	-

$\mathcal{T}_3$	ϵ	aaa	aa	a
* ϵ	-	+	-	-
* a	-	+	+	-
* b	-	+	-	-
* aa	-	+	+	+
* ab	-	+	-	+

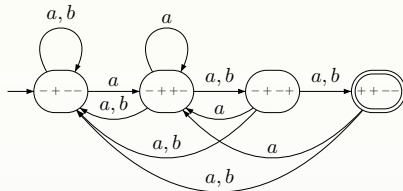
$\mathcal{T}_4$	ϵ	aaa	aa	a
* ϵ	-	+	-	-
* a	-	+	+	-
* ab	-	+	-	+
* b	-	+	-	-
* aa	-	+	+	+
* aba	+	+	+	-
* abb	+	+	-	-

$\mathcal{T}_5$	ϵ	aaa	aa	a
* ϵ	-	+	-	-
* a	-	+	+	-
* ab	-	+	-	+
* abb	+	+	-	-
* b	-	+	-	-
* aa	-	+	+	+
* aba	+	+	+	-
* abba	-	+	+	-
* abbb	-	+	-	-

The algorithm in action



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



$\mathcal{T}_1$	ϵ
* ϵ	-
* a	-
* b	-

$\mathcal{T}_2$	ϵ	aaa	aa	a
* ϵ	-	+	-	-
* a	-	+	+	-
* b	-	+	-	-

$\mathcal{T}_3$	ϵ	aaa	aa	a
* ϵ	-	+	-	-
* a	-	+	+	-
* b	-	+	-	-
* aa	-	+	+	+
* ab	-	+	-	+

$\mathcal{T}_4$	ϵ	aaa	aa	a
* ϵ	-	+	-	-
* a	-	+	+	-
* ab	-	+	-	+
* b	-	+	-	-
* aa	-	+	+	+
* aba	+	+	+	-
* abb	+	+	-	-

$\mathcal{T}_5$	ϵ	aaa	aa	a
* ϵ	-	+	-	-
* a	-	+	+	-
* ab	-	+	-	+
* abb	+	+	-	-
* b	-	+	-	-
* aa	-	+	+	+
* aba	+	+	+	-
* abba	-	+	+	-
* abbb	-	+	-	-