Learning Communicating and Nondeterministic Automata

Carsten Kern

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

Oberseminar
Aachen, August 31st 2009

Results of this thesis

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

Learning

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

Presentation outline

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

Learning

Here, learning means:

Given exemplifying behavior of a system

Learn a *model* conforming to the given behavior

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)

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)

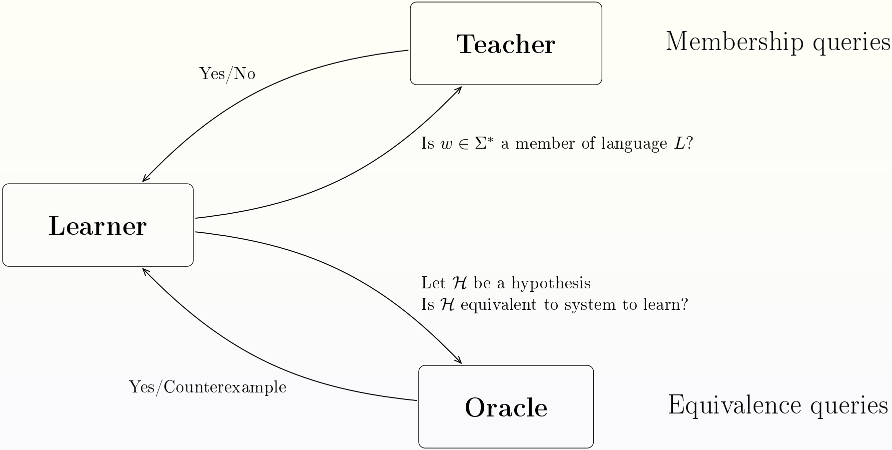
Active Learning

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

Occam's razor:

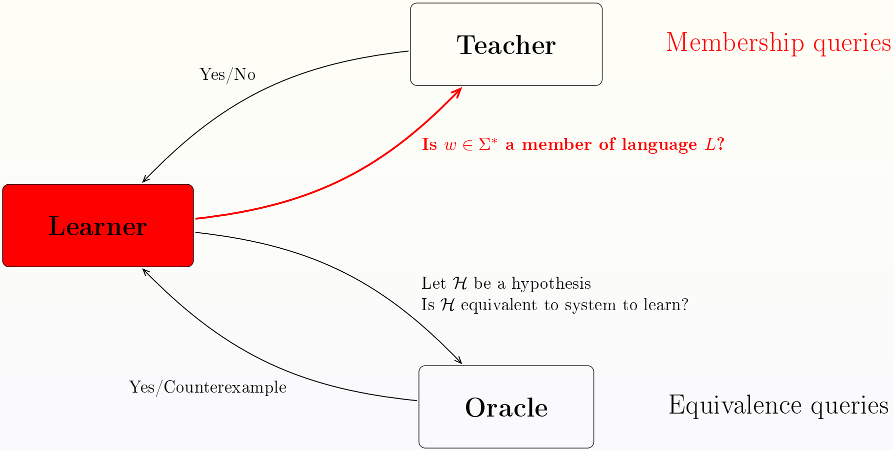
"In case of different explanations, choose the *simplest* one."
$\Rightarrow$ Learn the *minimal* DFA conforming to given examples

Algorithm - Overview



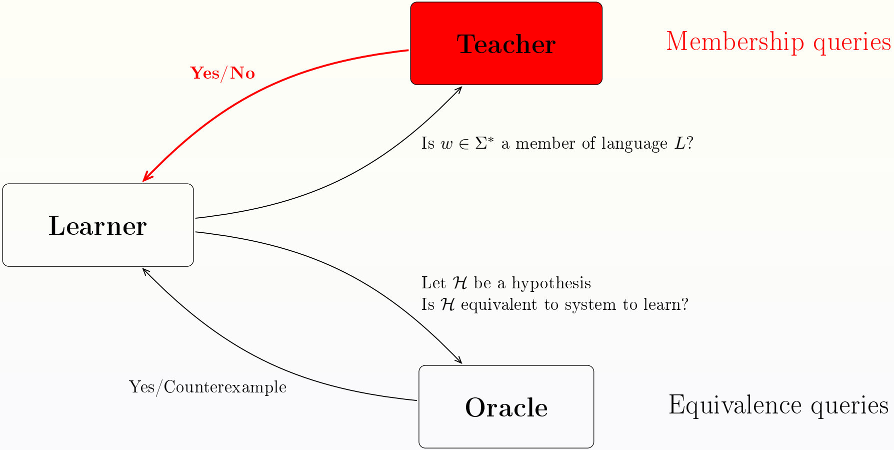
- *L*: (regular) language to learn

Algorithm - Overview



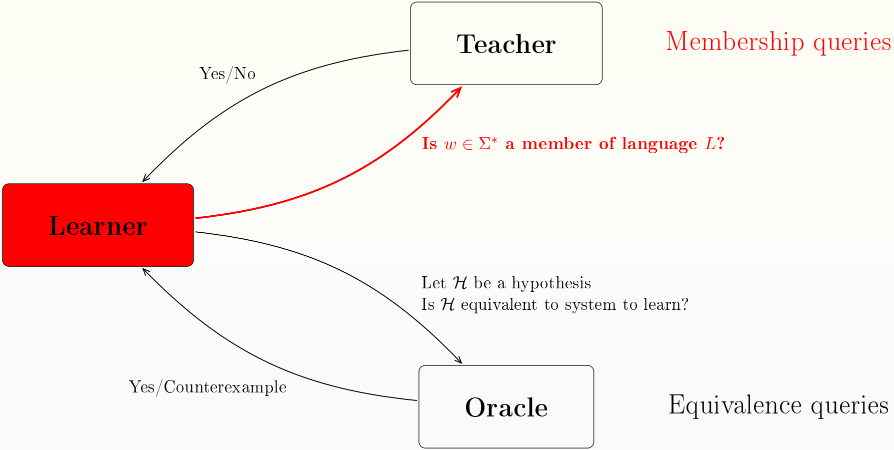
- L: (regular) language to learn

Algorithm - Overview



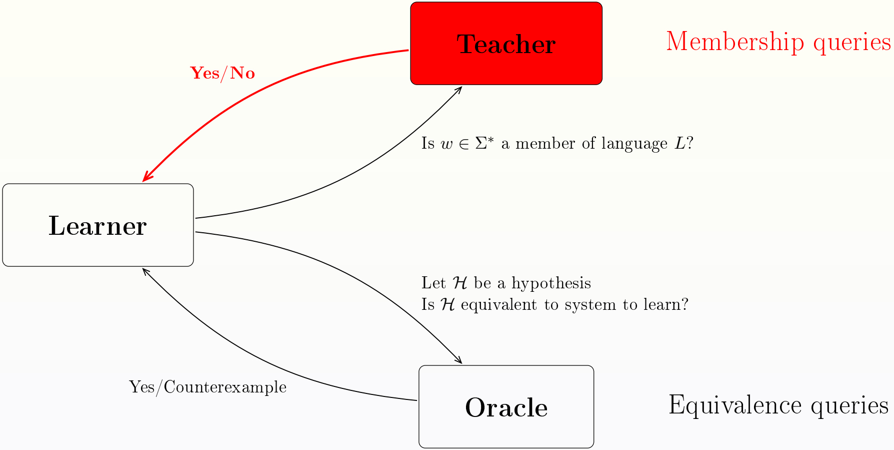
- L: (regular) language to learn

Algorithm - Overview



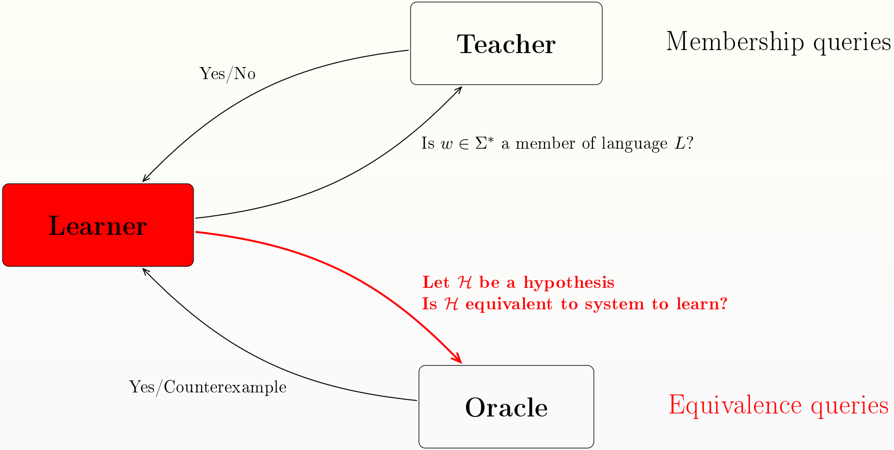
- L: (regular) language to learn

Algorithm - Overview



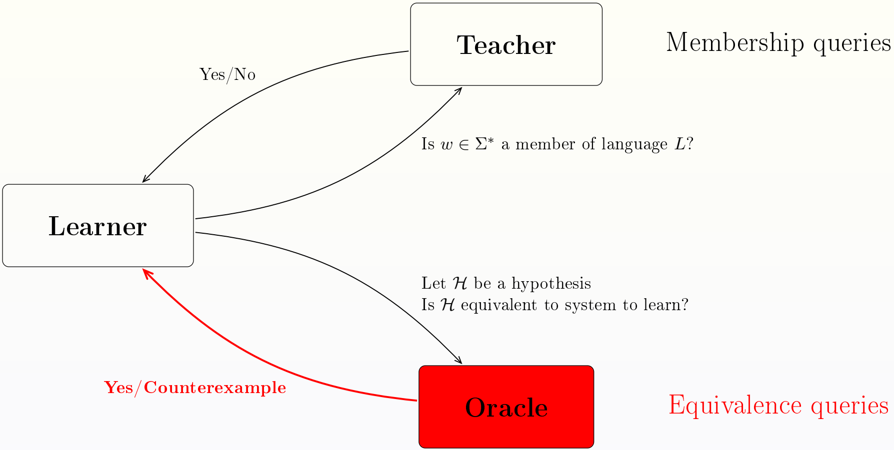
- L: (regular) language to learn

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	$+$

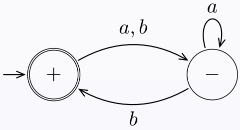
To derive an automaton:

- $\mathcal{T}$ 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

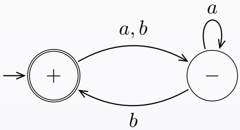
To this end:

- 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	$+$

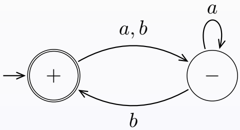


$bb \in L(\mathcal{H})$ but $bb \notin L$!

Table-based learning

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

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



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

Counterexample can be added to:

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 $\mathcal{A}$.

Angluin's Algorithm L*

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:

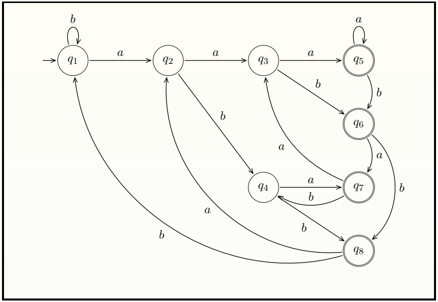
- n: number of states of the minimal DFA $\mathcal{A}_L$ for regular language L,
- m: length of the biggest counterexample

Then, L* returns after at most:

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

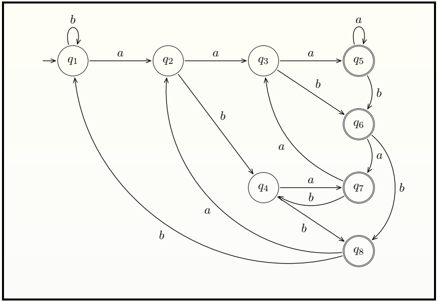
the minimal DFA $\mathcal{A}$.

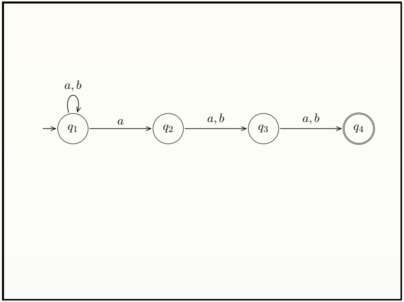
But there is a problem . . .



minimal DFA can be huge!

But there is a problem . . .



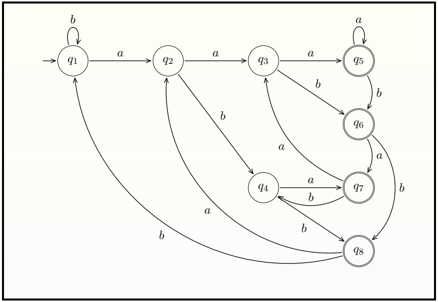


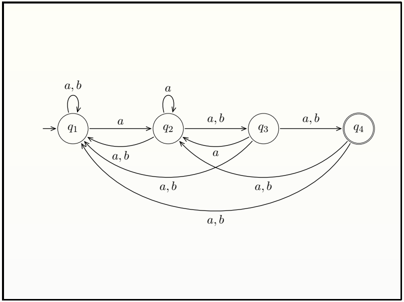
minimal DFA can be huge!

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

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

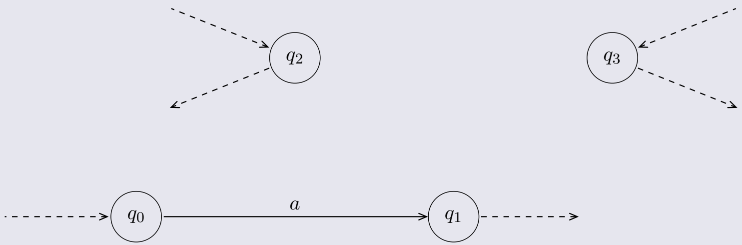
Goal

- Learn more compact representations of regular languages

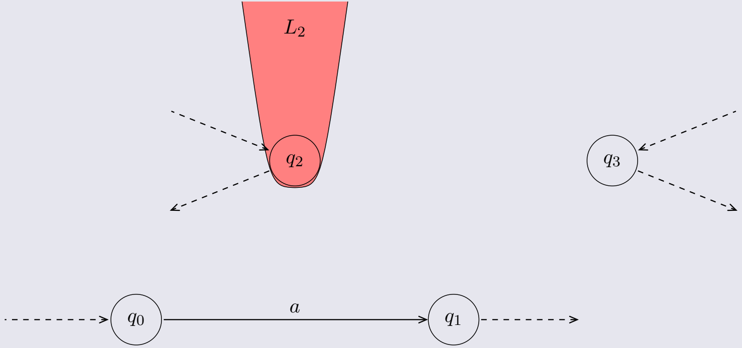
How?

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

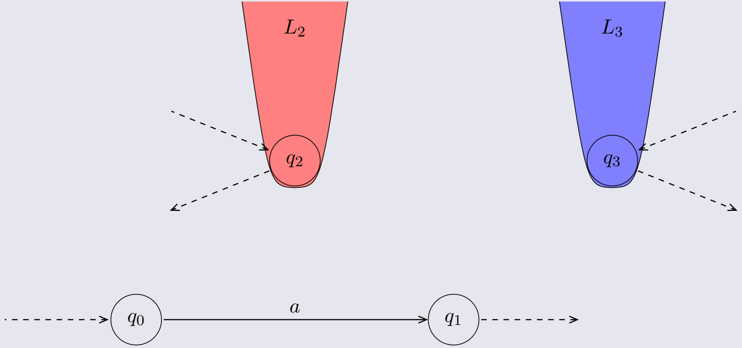
Residual Finite-State Automata [Denis et al.]



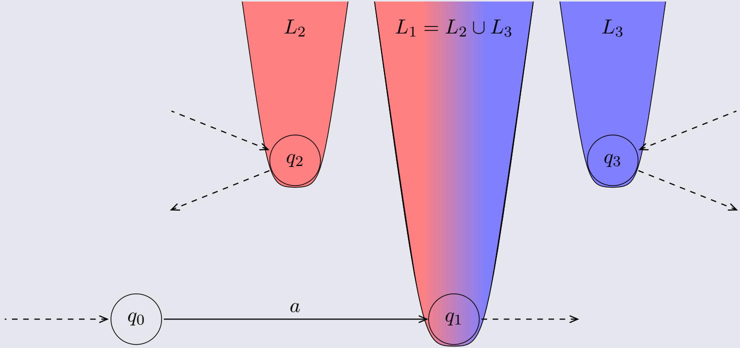
Residual Finite-State Automata [Denis et al.]



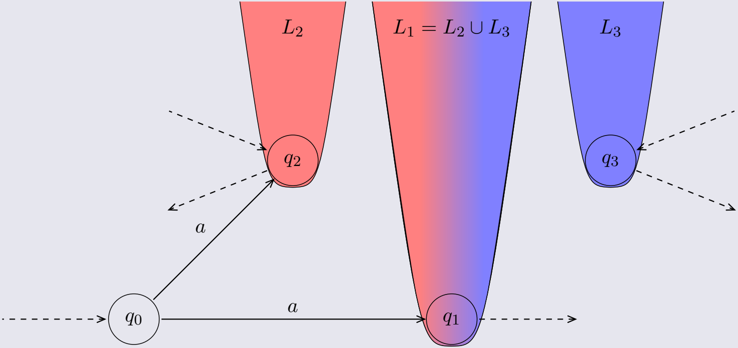
Residual Finite-State Automata [Denis et al.]



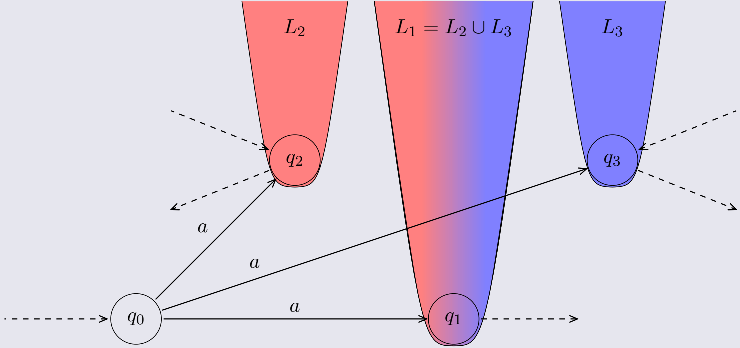
Residual Finite-State Automata [Denis et al.]



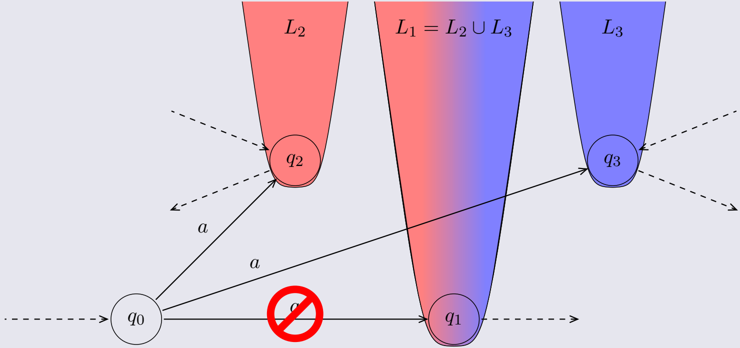
Residual Finite-State Automata [Denis et al.]



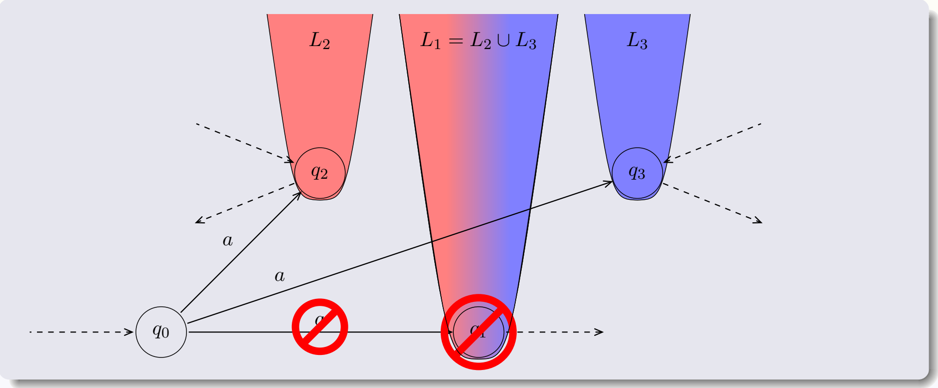
Residual Finite-State Automata [Denis et al.]



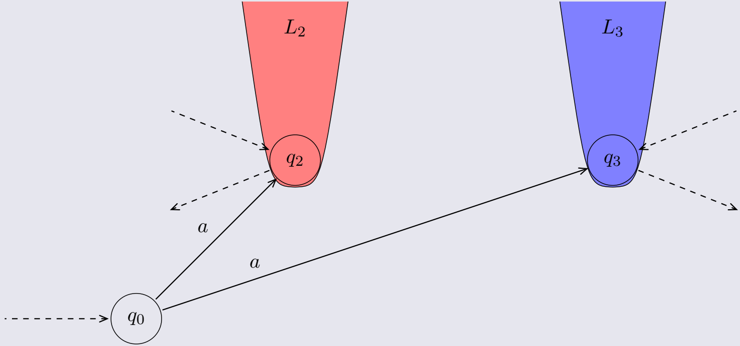
Residual Finite-State Automata [Denis et al.]



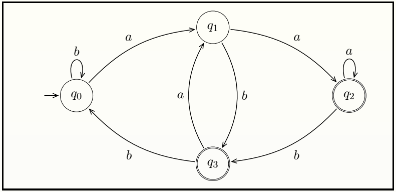
Residual Finite-State Automata [Denis et al.]



Residual Finite-State Automata [Denis et al.]



Residual Finite-State Automata

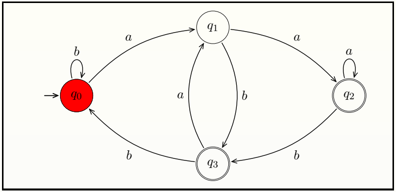


Definition (Residual Language)

For a language $L \subseteq \Sigma^*$ and $u \in \Sigma^*$:
$u^{-1}L = \{v \in \Sigma^* \mid uv \in L\}$ (u-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.



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

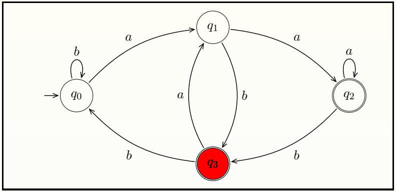
Definition (Residual Language)

For a language $L \subseteq \Sigma^*$ and $u \in \Sigma^*$:
$u^{-1}L = \{v \in \Sigma^* \mid uv \in L\}$ $\quad$ (u-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.

Residual Finite-State Automata



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

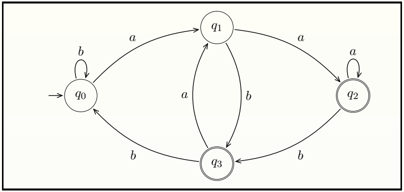
Definition (Residual Language)

For a language $L \subseteq \Sigma^*$ and $u \in \Sigma^*$:
$u^{-1}L = \{v \in \Sigma^* \mid uv \in L\}$ $\quad$ (u-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.

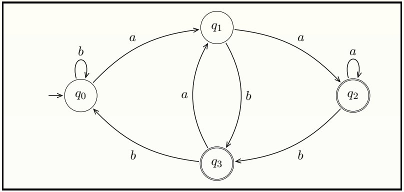
Residual Finite-State Automata



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 $q \in Q$, $L_q \in \mathit{Res}(L(\mathcal{R}))$.

Residual Finite-State Automata

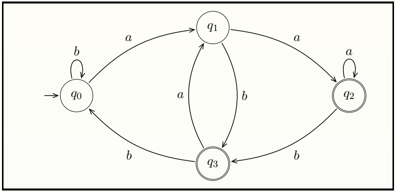


- $\varepsilon^{-1}L = \Sigma^* a \Sigma$ $\quad (= L_{q_0})$
- $(a)^{-1}L = \Sigma^* a \Sigma \cup \Sigma$ $\quad (= L_{q_1})$
- $(b)^{-1}L = \Sigma^* a \Sigma \cup \Sigma \cup \{\varepsilon\}$ $\quad (= L_{q_2})$
- $(ab)^{-1}L = \Sigma^* a \Sigma \cup \{\varepsilon\}$ $\quad (= L_{q_3})$

Definition (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 $q \in Q$, $L_q \in Res(L(\mathcal{R}))$.

Towards canonical RFSA



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

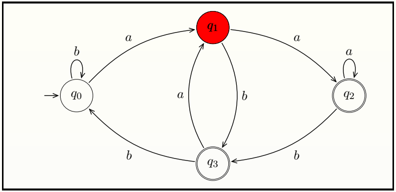
Definition (Prime and Composed Residuals)

Let $L \subseteq \Sigma^*$ be a language. A residual $L' \in Res(L)$ 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



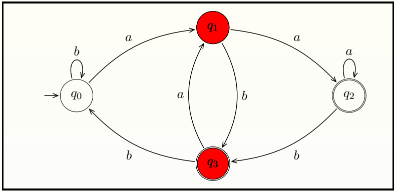
- $\varepsilon^{-1}L = \Sigma^* a \Sigma$ $\quad (= L_{q_0})$
- $(a)^{-1}L = \Sigma^* a \Sigma \cup \Sigma$ $\quad (= L_{q_1})$
- $(b)^{-1}L = \Sigma^* a \Sigma \cup \Sigma \cup \{\varepsilon\}$ $\quad (= L_{q_2})$
- $(ab)^{-1}L = \Sigma^* a \Sigma \cup \{\varepsilon\}$ $\quad (= L_{q_3})$

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 $Primes(L)$.



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

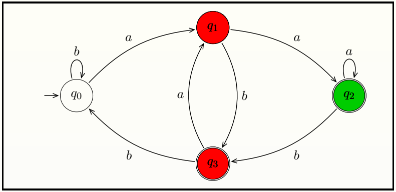
Definition (Prime and Composed Residuals)

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

Towards canonical RFSA



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

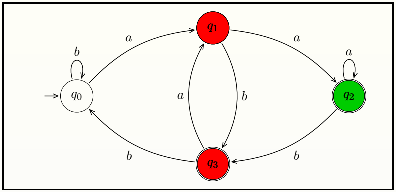
Definition (Prime and Composed Residuals)

Let $L \subseteq \Sigma^*$ be a language. A residual $L' \in Res(L)$ 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



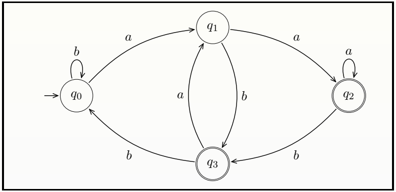
- $\varepsilon^{-1}L = \Sigma^* a \Sigma$ $(= L_{q_0})$
- $(a)^{-1}L = \Sigma^* a \Sigma \cup \Sigma$ $(= L_{q_1})$
- $(b)^{-1}L = \Sigma^* a \Sigma \cup \Sigma \cup \{\varepsilon\}$ $(= L_{q_2})$
- $(ab)^{-1}L = \Sigma^* a \Sigma \cup \{\varepsilon\}$ $(= L_{q_3})$

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

- $Q = \mathit{Primes}(L)$,
- $Q_0 = \{L' \in Q \mid L' \subseteq L\}$,
- $F = \{L' \in Q \mid \varepsilon \in L'\}$, 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 $L = \Sigma^* a \Sigma$

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

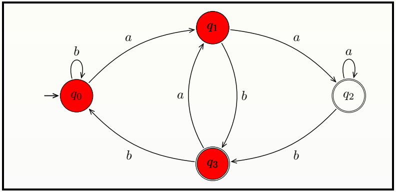
$$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$$

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

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

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

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



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

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

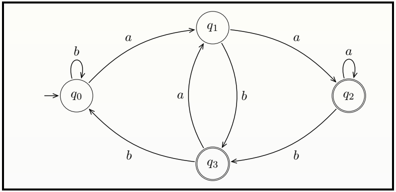
$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$

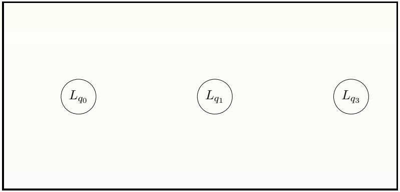
$L_{q_2} = \Sigma^* a \Sigma \cup \Sigma \cup \{\varepsilon\}$

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

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

States: $Q = Primes(L)$



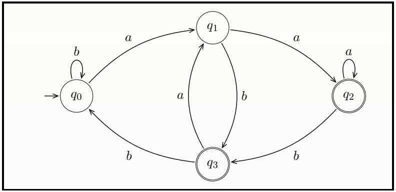


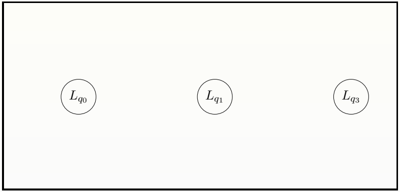
Residual languages for $L = \Sigma^* a \Sigma$

$$L_{q_0} = \Sigma^* a \Sigma$$
$$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$$
$$L_{q_2} = \Sigma^* a \Sigma \cup \Sigma \cup \{\varepsilon\}$$
$$L_{q_3} = \Sigma^* a \Sigma \cup \{\varepsilon\}$$

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)

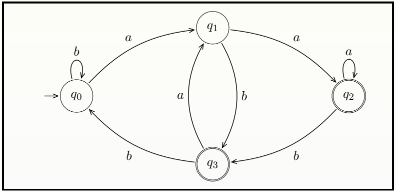
$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$

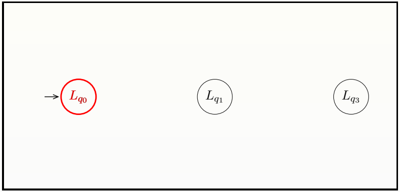
$L_{q_2} = \Sigma^* a \Sigma \cup \Sigma \cup \{\varepsilon\}$

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

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

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





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

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

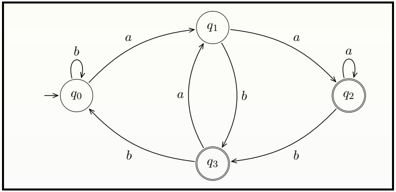
$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$

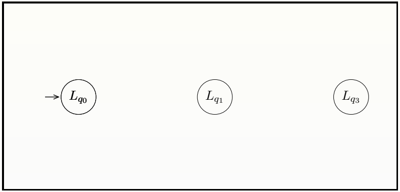
$L_{q_2} = \Sigma^* a \Sigma \cup \Sigma \cup \{\varepsilon\}$

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

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

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



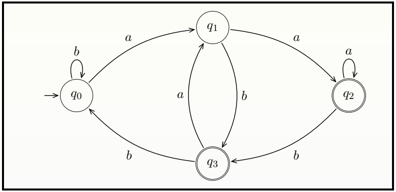


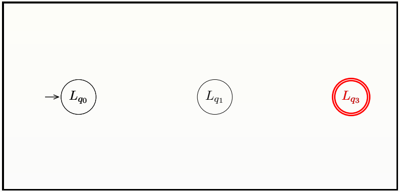
Residual languages for $L = \Sigma^* a \Sigma$

$$L_{q_0} = \Sigma^* a \Sigma$$
$$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$$
$$L_{q_2} = \Sigma^* a \Sigma \cup \Sigma \cup \{\varepsilon\}$$
$$L_{q_3} = \Sigma^* a \Sigma \cup \{\varepsilon\} \qquad \text{(final state)}$$

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

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



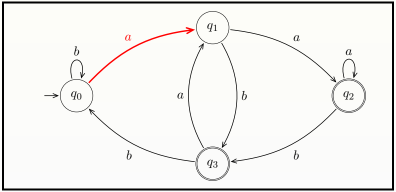


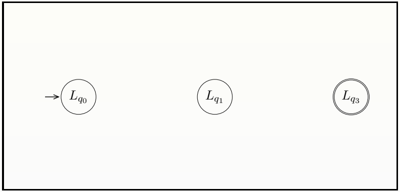
Residual languages for $L = \Sigma^* a \Sigma$

$$L_{q_0} = \Sigma^* a \Sigma$$
$$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$$
$$L_{q_2} = \Sigma^* a \Sigma \cup \Sigma \cup \{\varepsilon\}$$
$$L_{q_3} = \Sigma^* a \Sigma \cup \{\varepsilon\} \qquad \text{(final state)}$$

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 $L = \Sigma^* a \Sigma$

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

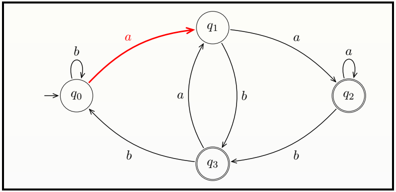
$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$

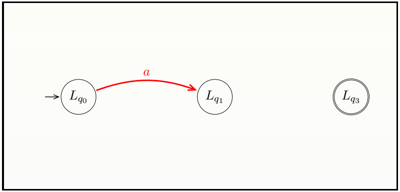
$L_{q_2} = \Sigma^* a \Sigma \cup \Sigma \cup \{\varepsilon\}$

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

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)

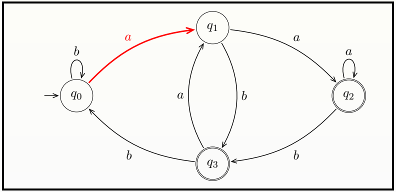
$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$

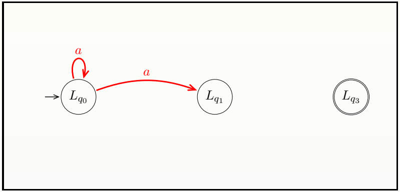
$L_{q_2} = \Sigma^* a \Sigma \cup \Sigma \cup \{\varepsilon\}$

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

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 $L = \Sigma^* a \Sigma$

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

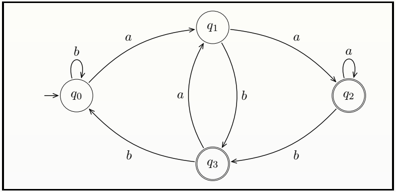
$L_{q_1} = \Sigma^* a \Sigma \cup \Sigma$

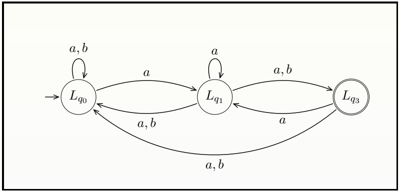
$L_{q_2} = \Sigma^* a \Sigma \cup \Sigma \cup \{\varepsilon\}$

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

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 $L = \Sigma^* a \Sigma$

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

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

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

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

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

$(r_1 \sqcup r_2) : V \to \{+, -\}$:
$(r_1 \sqcup r_2)(v) = r_1(v) \sqcup r_2(v)$ 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 $r_1, r_2 \in Rows(\mathcal{T})$ is defined component-wise for each $v \in V$:

$(r_1 \sqcup r_2) : V \to \{+, -\}$:
$(r_1 \sqcup r_2)(v) = r_1(v) \sqcup r_2(v)$ where

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

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

Example

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

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

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

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

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

Example

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

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

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 $r_1, \ldots, r_n \in Rows(\mathcal{T}) \setminus \{r\}$ such that $r = r_1 \sqcup \ldots \sqcup r_n$.
- prime, otherwise.

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

RWTH AACHEN UNIVERSITY

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 $r_1, \ldots, r_n \in Rows(\mathcal{T}) \setminus \{r\}$ such that $r = r_1 \sqcup \ldots \sqcup r_n$.
- 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 $r_1, \ldots, r_n \in Rows(\mathcal{T}) \setminus \{r\}$ such that $r = r_1 \sqcup \ldots \sqcup r_n$.
- 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 $r_1, \ldots, r_n \in Rows(\mathcal{T}) \setminus \{r\}$ such that $r = r_1 \sqcup \ldots \sqcup r_n$.
- 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 $r \in \mathit{Rows}(\mathcal{T})$ is:

- covered by row $r' \in \mathit{Rows}(\mathcal{T})$ ($r \sqsubseteq r'$), 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., $\mathit{row}(\varepsilon) \sqsubseteq \mathit{row}(a)$ and $\mathit{row}(\varepsilon) \sqsubseteq \mathit{row}(abb)$

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 $v \in V$: $r(v) = + \Rightarrow r'(v) = +$.
- 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., $row(\varepsilon) \sqsubseteq row(a)$ and $row(\varepsilon) \sqsubseteq row(abb)$
- e.g., $row(\varepsilon) \sqsubset row(ab)$

RWTH AACHEN UNIVERSITY

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

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

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

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

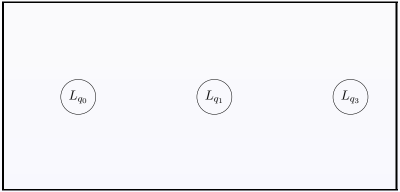
From Table to NFA

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

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

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



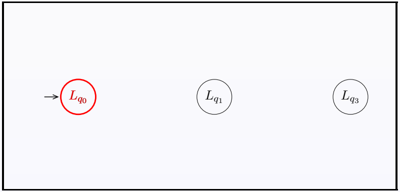
From Table to NFA

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

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

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



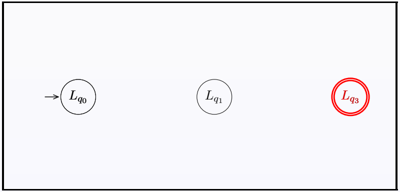
From Table to NFA

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

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

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



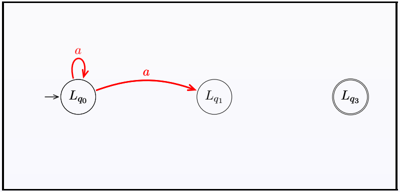
From Table to NFA

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

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

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



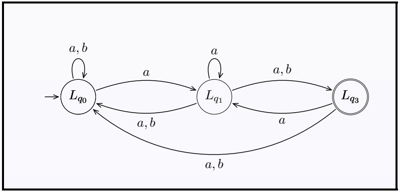
From Table to NFA

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

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

	$\mathcal{T}$	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

Towards correctness

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 $w \in L(\mathcal{R}_\mathcal{T})$.

Towards correctness

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.

Complexity issues

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 $\mathcal{R}(L)$.

Complexity issues

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 $\mathcal{R}(L)$.

Complexity issues

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 $\mathcal{R}(L)$.

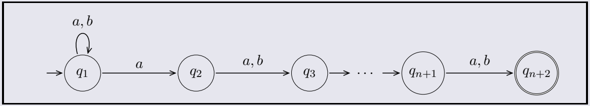
It's worth considering RFSA...

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.*

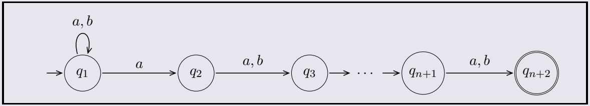
It's worth considering RFSA...

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

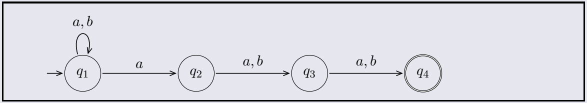


It's worth considering RFSA...

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

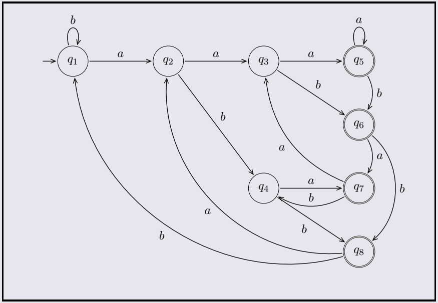


$L_2 = \{w \in \Sigma^* | w$ has an a at the 3rd-last position$\}$



Minimal DFA and RFSA

Minimal DFA and RFSA for L_2:

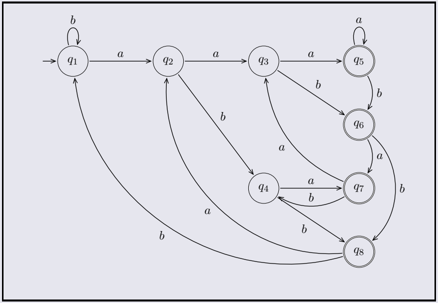


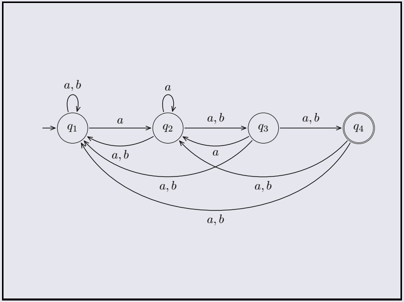
Automata for language L_n:

- 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

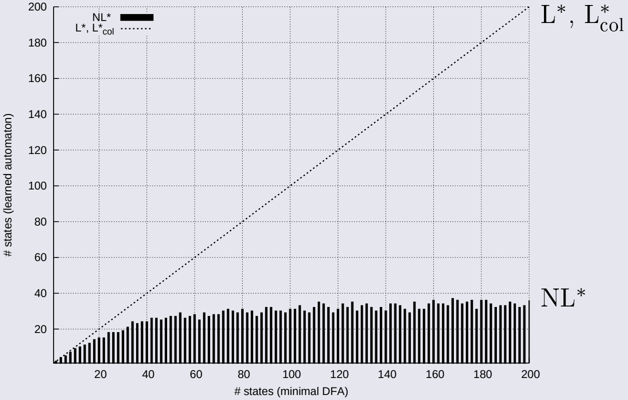
Comparison of L^*, L^*_{col}, and NL^*

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

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

Algorithm - Overview

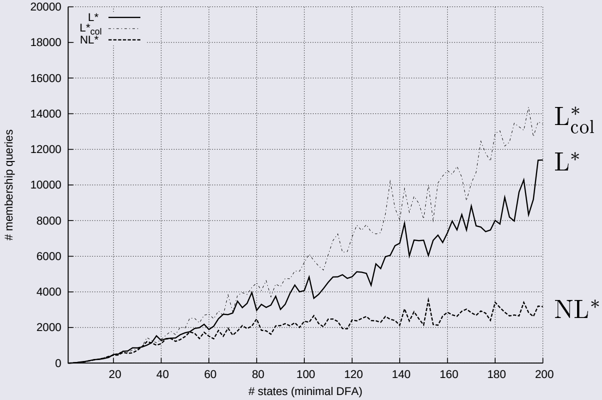
Number of states (L^*, L^*_{col} vs. NL^*)



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

Algorithm - Overview

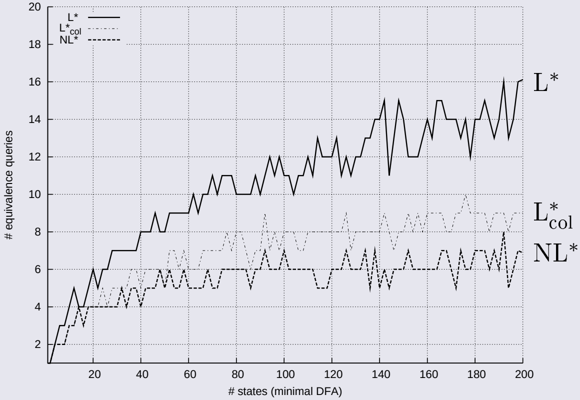
Number of membership queries (L^* vs. L^*_{col} vs. NL^*)



- ≈ 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

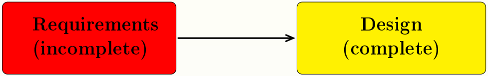
Presentation outline

Motivation

Requirements (incomplete)

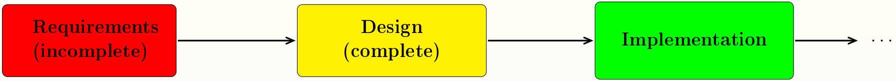
- initial phase: requirement elicitation
  - contradicting or incomplete system description
  - common description language: sequence diagrams

Motivation



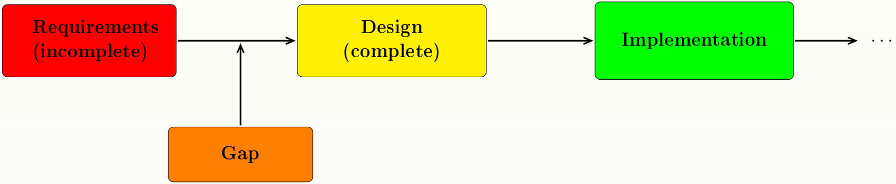
- initial phase: requirement elicitation
  - contradicting or incomplete system description
  - common description language: sequence diagrams
- goal: conforming design model

Motivation

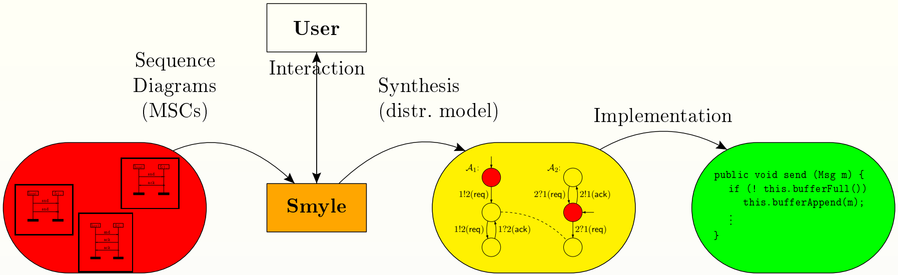


- initial phase: requirement elicitation
  - contradicting or incomplete system description
  - common description language: sequence diagrams
- goal: conforming design model

Motivation



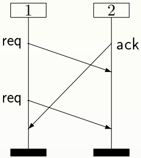
- initial phase: requirement elicitation
  - contradicting or incomplete system description
  - common description language: sequence diagrams
- goal: conforming design model
- closing gap between
  - requirement specification (usually incomplete) and
  - design model (complete description of system)



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

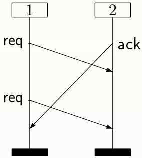
Message Sequence Chart



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

- $\mathcal{P}$: finite set of processes
- E: finite set of events ($E = \bigcup_{p \in \mathcal{P}} E_p$)
- $l : E \to Act = \{1!2(\mathsf{req}), 1?2(\mathsf{ack}), \dots\}$
- 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



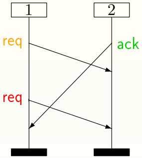
An MSC $M = \langle \mathcal{P}, E, \{\leq_p\}_{p\in\mathcal{P}}, <_{\mathsf{msg}}, l\rangle$

- $\mathcal{P}$: finite set of processes
- E: finite set of events ($E = \bigcup_{p\in\mathcal{P}} E_p$)
- $l : E \rightarrow Act = \{1!2(\mathsf{req}), 1?2(\mathsf{ack}), \dots\}$
- 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 $\leq$

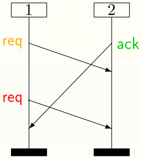
MSCs and Linearizations



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

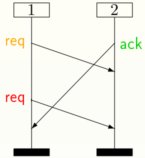


Some linearizations

- $1!2(\text{req})\ 1!2(\text{req})\ 2!1(\text{ack})\ 1?2(\text{ack})\ 2?1(\text{req})\ 2?1(\text{req})$
- $1!2(\text{req})\ 2!1(\text{ack})\ 1!2(\text{req})\ 1?2(\text{ack})\ 2?1(\text{req})\ 2?1(\text{req})$
- $2!1(\text{ack})\ 1!2(\text{req})\ 2?1(\text{req})\ 1!2(\text{req})\ 2?1(\text{req})\ 1?2(\text{ack})$
- ...

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

MSCs and Linearizations



Some linearizations

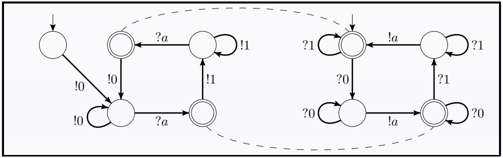
- $1!2$(req) $1!2$(req) $2!1$(ack) $1?2$(ack) $2?1$(req) $2?1$(req)
- $1!2$(req) $2!1$(ack) $1!2$(req) $1?2$(ack) $2?1$(req) $2?1$(req)
- $2!1$(ack) $1!2$(req) $2?1$(req) $1!2$(req) $2?1$(req) $1?2$(ack)
- ...

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

Communicating Finite-State Machines (CFM)

A CFM consists of:

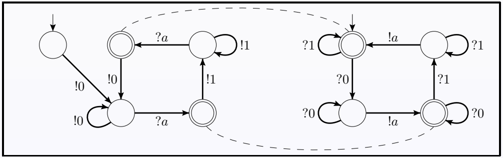
- a set of finite-state automata (*processes*) with
  - common global initial state
  - set of global final states



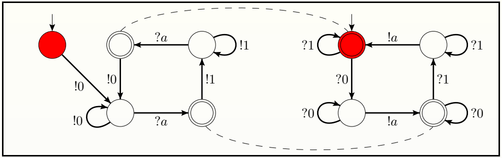
Communicating Finite-State Machines (CFM)

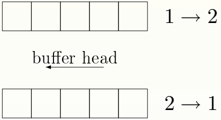
A CFM consists of:

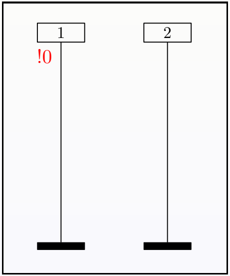
- a set of finite-state automata (*processes*) with
  - common global initial state
  - set of global final states
- communication between automata through (reliable) FIFO channels
  - $p!q(a)$ appends message a to buffer between p and q
  - $q?p(a)$ removes message a from buffer between p and q



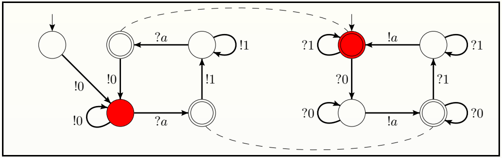
CFM: An Example

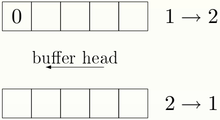


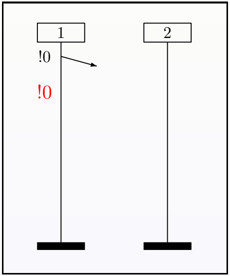




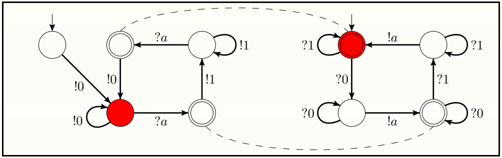
CFM: An Example

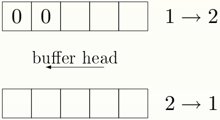


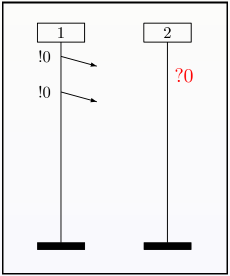




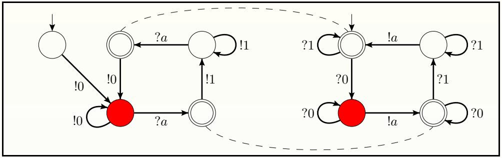
CFM: An Example

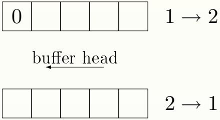


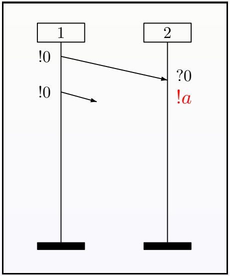




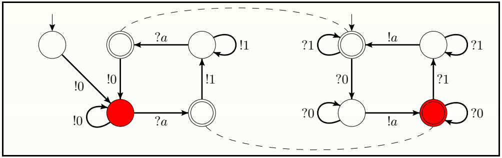
CFM: An Example

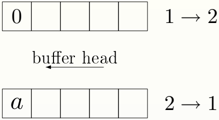


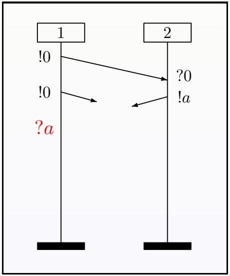




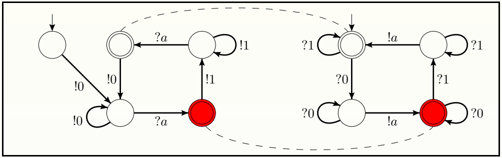
CFM: An Example

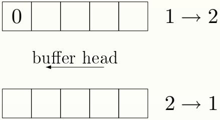


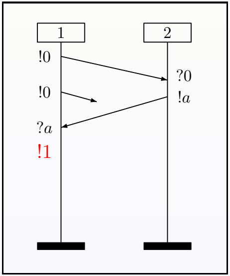




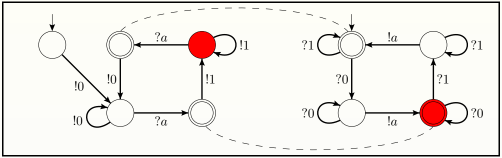
CFM: An Example

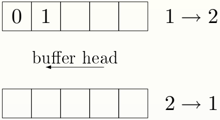


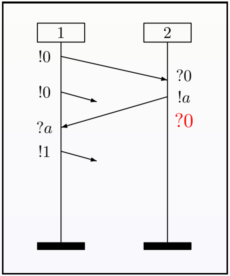




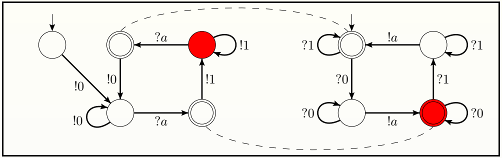
CFM: An Example

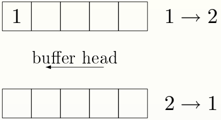


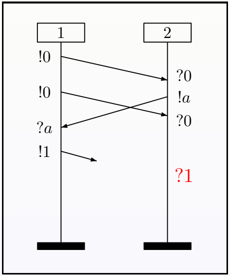




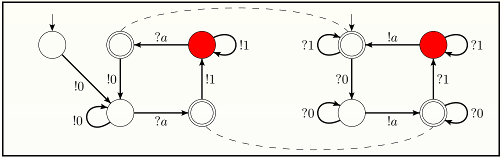
CFM: An Example

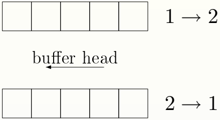


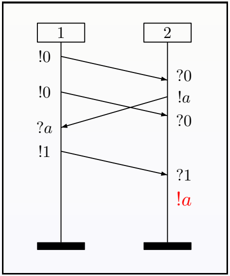




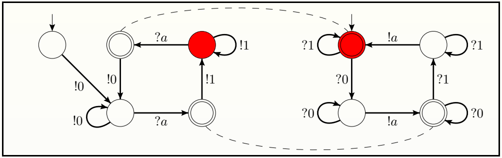
CFM: An Example

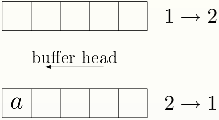


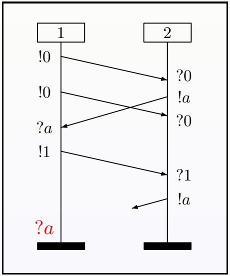




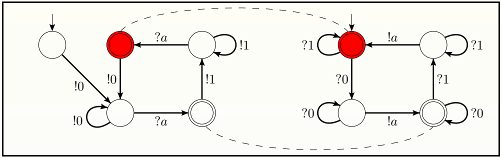
CFM: An Example

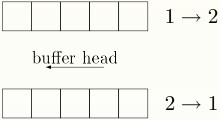


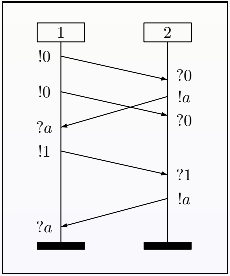




CFM: An Example





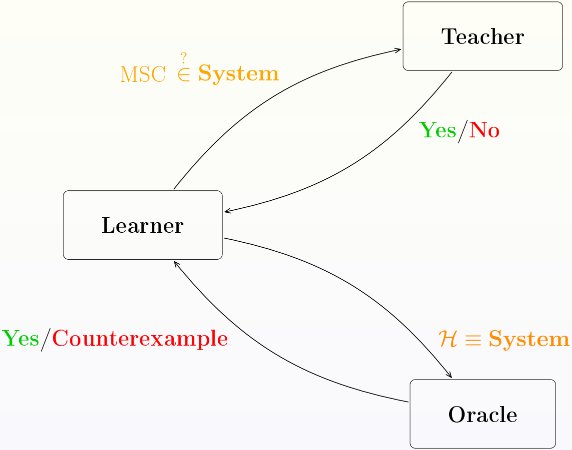


Current State

Current State

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

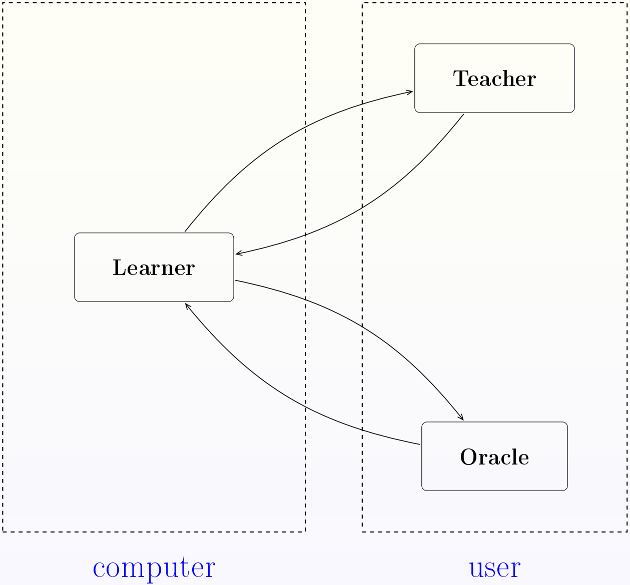
The learning algorithm (extension of Angluin's L^*)



Membership queries

Equivalence queries

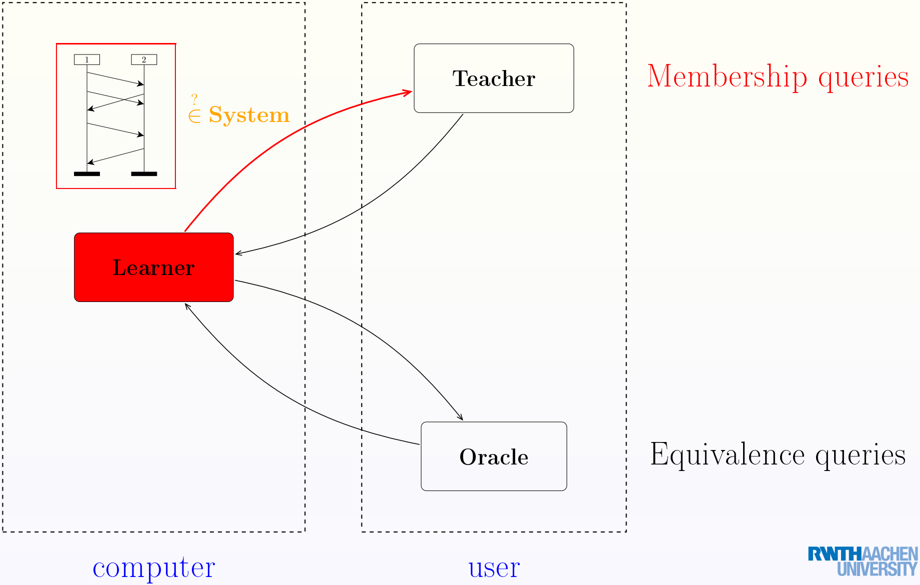
The learning algorithm (extension of Angluin's L^*)



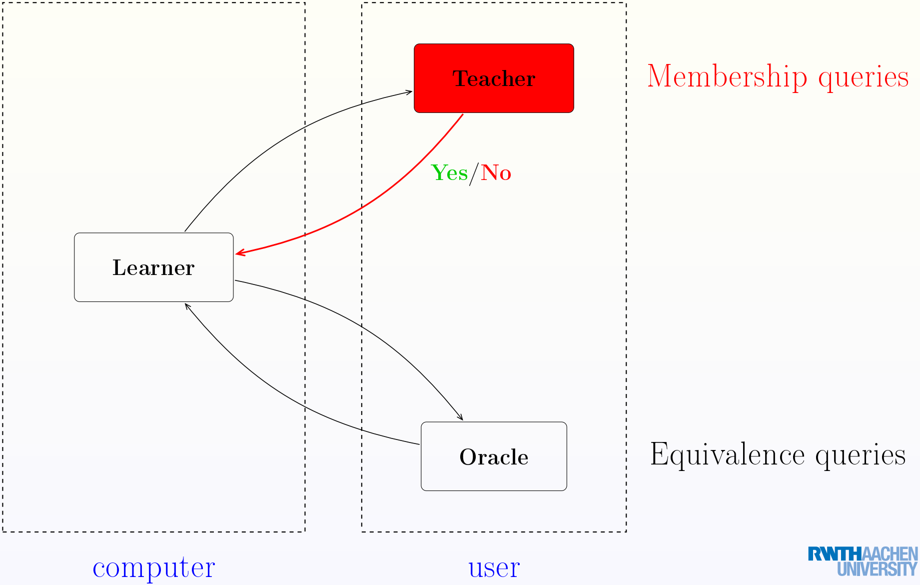
Membership queries

Equivalence queries

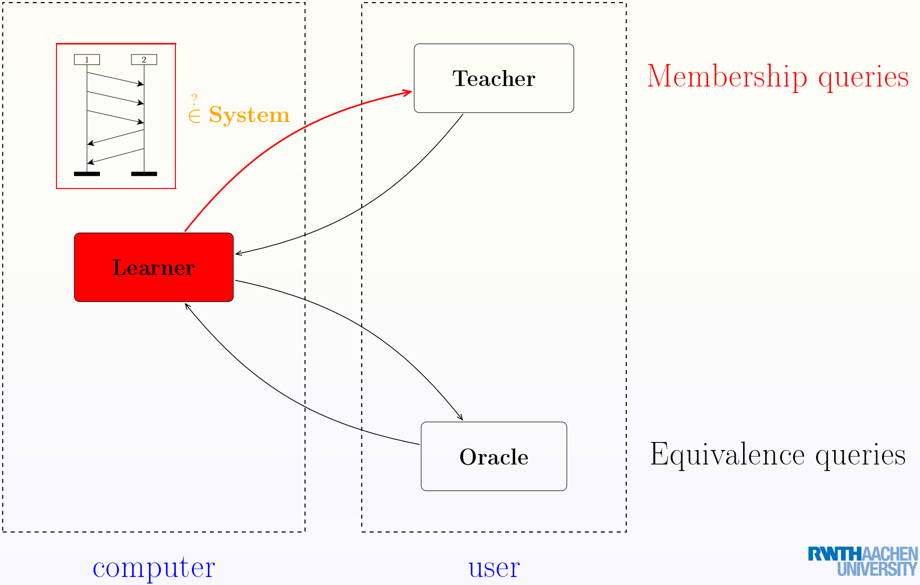
The learning algorithm (extension of Angluin's L^*)



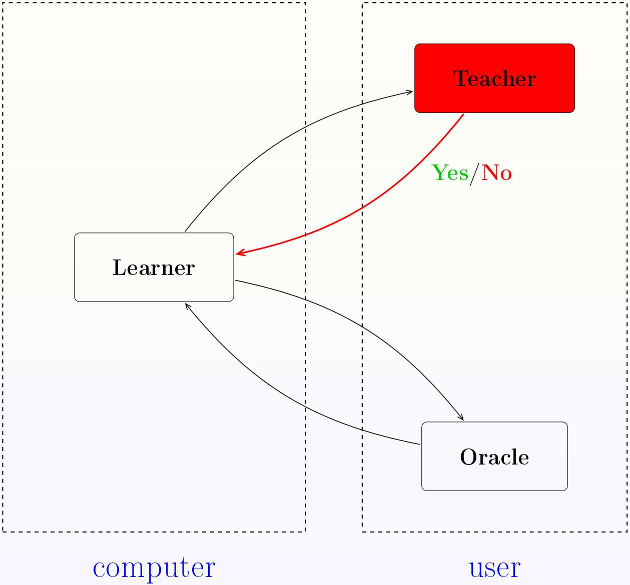
The learning algorithm (extension of Angluin's L^*)



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



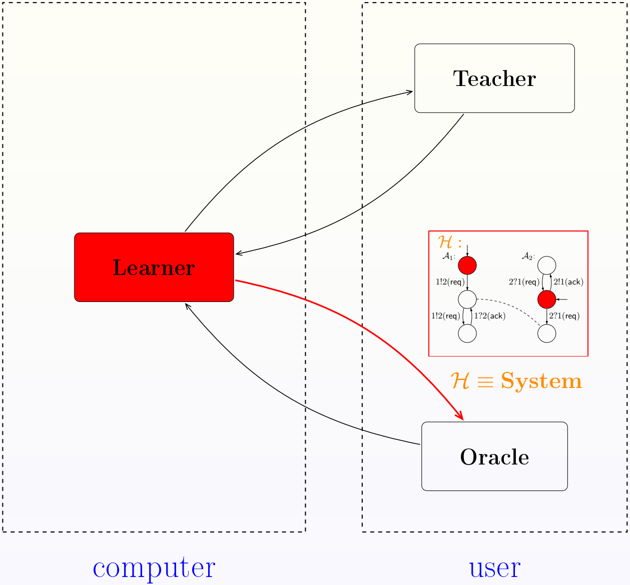
The learning algorithm (extension of Angluin's L^*)



Membership queries

Equivalence queries

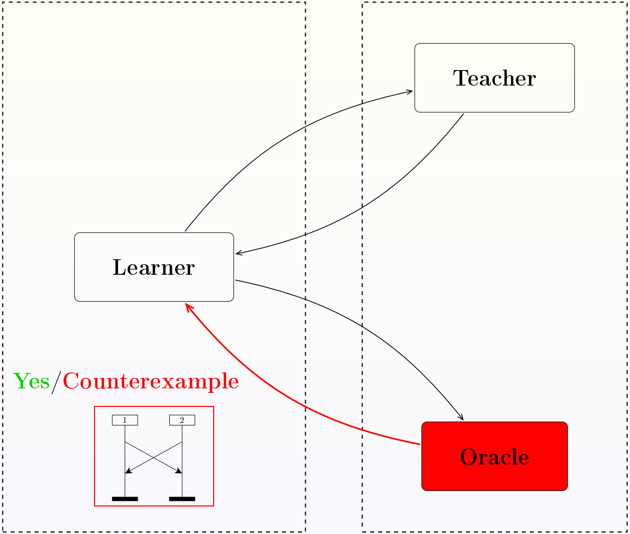
The learning algorithm (extension of Angluin's L^*)



Membership queries

Equivalence queries

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



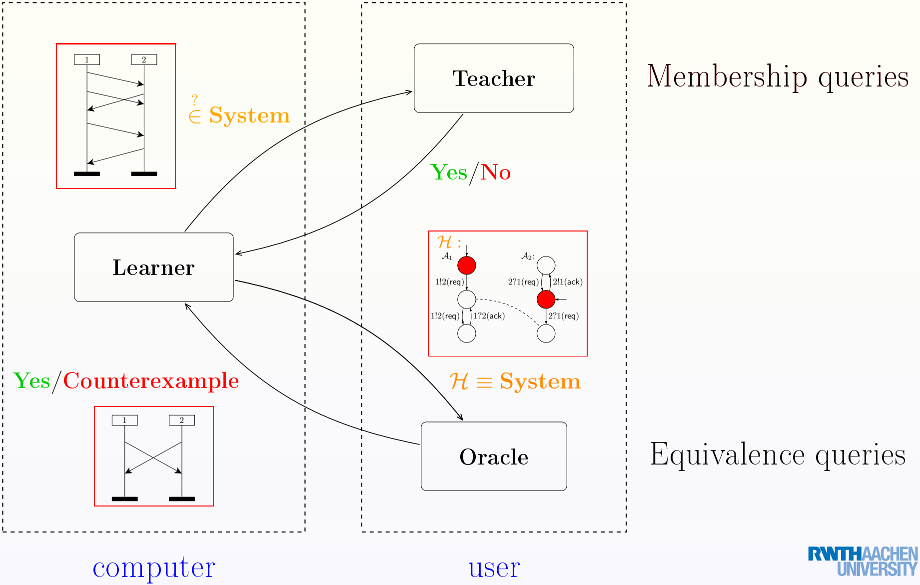
Membership queries

Equivalence queries

computer

user

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



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

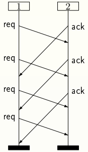
Problem

- correspondence between CFMs and regular word languages needed

Classes of MSCs

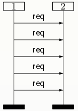
M is $\forall B$-bounded ($B \in \mathbb{N}$) if

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 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:

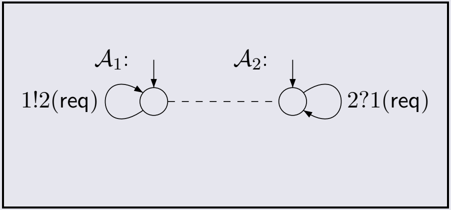
- (deterministic) $\forall$-bounded CFMs
- $\exists B$-bounded CFMs ($B \in \mathbb{N}$)
- deterministic $\forall$-bounded deadlock-free weak CFMs

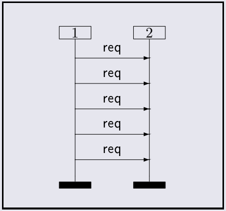
Not learnable (in a guided fashion)

- $\forall$-bounded weak CFMs

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*

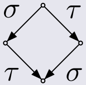
1. mark states of $\mathcal{H}$ with their channel contents and always check if the channel capacity $\leq B$
   - sending adds a message to corresponding channel
   - receiving removes a message from channel head

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

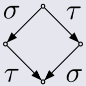


(B-bounded version) for independent σ, τ

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*

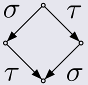
1. mark states of $\mathcal{H}$ with their channel contents and always check if the channel capacity $\leq B$
   - sending adds a message to corresponding channel
   - receiving removes a message from channel head
2. check diamond rule 

(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 $\mathcal{H}$

Complexity results

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 w.o. POL	w. POL	savings	#user queries	#equivalence queries	$\lvert\mathcal{H}\rvert$	#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%	∃2
continuous update	712	264	62.9%	21	1 (3)	8	89	34	61.8%	∃1
negotiation	1,179	432	63.4%	31	1 (3)	9	131	49	62.6%	∃1
ABP	2,286	697	69.5%	64	2 (4)	15	127	42	66.9%	∃1
ABP	14,432	4,557	68.4%	158	2 (13)	25	451	131	71.0%	∃2
ABP	55,131	19,252	65.1%	407	2 (22)	37	799	222	72.2%	∃3
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%	∀

RWTH AACHEN UNIVERSITY

Some results

Protocol	#membership queries w.o. POL	#membership queries w. POL	#membership queries savings	#user queries	#equivalence queries	$\|\mathcal{H}\|$	#rows in table w.o. POL	#rows in table w. POL	#rows in table reduction	learning setup
part of *USB 1.1*	488	200	59.0%	14	1 (5)	9	61	26	57.4%	∃2
continuous update	712	264	62.9%	21	1 (3)	8	89	34	61.8%	∃1
negotiation	1,179	432	63.4%	31	1 (3)	9	131	49	62.6%	∃1
ABP	2,286	697	69.5%	64	2 (4)	15	127	42	66.9%	∃1
ABP	14,432	4,557	68.4%	158	2 (13)	25	451	131	71.0%	∃2
ABP	55,131	19,252	65.1%	407	2 (22)	37	799	222	72.2%	∃3
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**)

Some results

Protocol	#membership queries w.o. POL	#membership queries w. POL	#membership queries savings	#user queries	#equivalence queries	$\|\mathcal{H}\|$	#rows in table w.o. POL	#rows in table w. POL	#rows in table reduction	learning setup
part of *USB 1.1*	488	200	59.0%	14	1 (5)	9	61	26	57.4%	∃2
continuous update	712	264	62.9%	21	1 (3)	8	89	34	61.8%	∃1
negotiation	1,179	432	63.4%	31	1 (3)	9	131	49	62.6%	∃1
ABP	2,286	697	69.5%	64	2 (4)	15	127	42	66.9%	∃1
ABP	14,432	4,557	68.4%	158	2 (13)	25	451	131	71.0%	∃2
ABP	55,131	19,252	65.1%	407	2 (22)	37	799	222	72.2%	∃3
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

Some results

Protocol	#membership queries w.o. POL	#membership queries w. POL	#membership queries savings	#user queries	#equivalence queries	$\|\mathcal{H}\|$	#rows in table w.o. POL	#rows in table w. POL	#rows in table reduction	learning setup
part of *USB 1.1*	488	200	59.0%	14	1 (5)	9	61	26	57.4%	∃2
continuous update	712	264	62.9%	21	1 (3)	8	89	34	61.8%	∃1
negotiation	1,179	432	63.4%	31	1 (3)	9	131	49	62.6%	∃1
ABP	2,286	697	69.5%	64	2 (4)	15	127	42	66.9%	∃1
ABP	14,432	4,557	68.4%	158	2 (13)	25	451	131	71.0%	∃2
ABP	55,131	19,252	65.1%	407	2 (22)	37	799	222	72.2%	∃3
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

`libalf`: the learning library

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 $\forall/\exists - B/\ldots$ 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

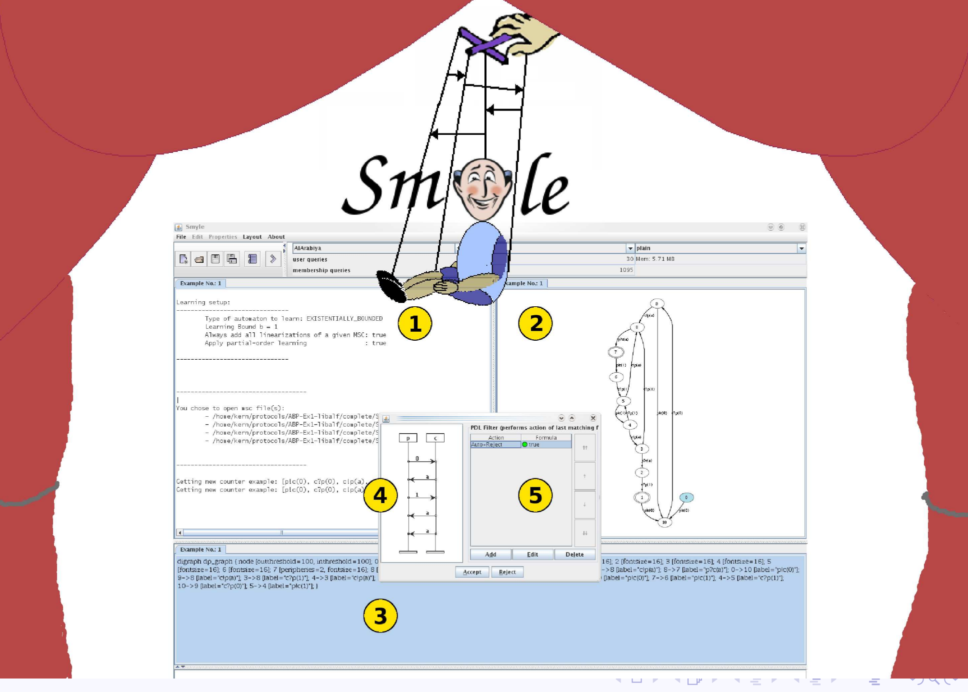
Features

- implements $\forall/\exists - B/\ldots$ 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`


Smile
Smyle
File Edit Properties Layout About
AlArabiya
plain
user queries
30
Mem: 5.71 MB
membership queries
1095
Example No.: 1
Learning setup:
Type of automaton to learn: EXISTENTIALLY_BOUNDED
Learning Bound b = 1
Always add all linearizations of a given MSC: true
Apply partial-order learning : true
You chose to open msc file(s):
Getting new counter example: [p!c(0), c?p(0), c!p(a), ...
PDL Filter (performs action of last matching f
Action
Formula
Auto-Reject
true
Add
Edit
Delete
Accept
Reject
1
2
3
4
5

Presentation outline

Summary

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)
- ...

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

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

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

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

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

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

$\mathcal{T}$	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
- $row(b), row(aba), row(abb) \in Primes_{\text{upp}}(\mathcal{T})$

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 $a \in \Sigma$:

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

$\mathcal{T}$	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 $a \in \Sigma$:

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

$\mathcal{T}$	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
  - $row(b) \sqsubseteq row(ab)$

Designing a table-based learning algorithm

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

Definition (RFSA-Consistency)

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

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

$\mathcal{T}$ 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

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

Definition (RFSA-Consistency)

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

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

$\mathcal{T}$ 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
  - $row(b) \sqsubseteq row(abb)$

Designing a table-based learning algorithm

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

Definition (RFSA-Consistency)

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

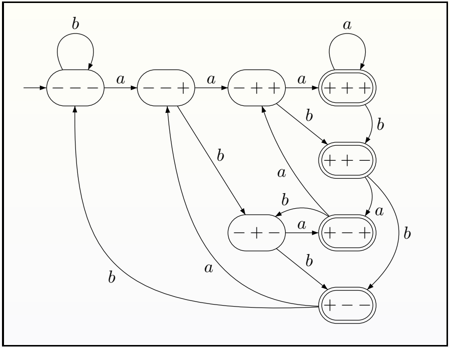
$$row(u') \sqsubseteq row(u) \Rightarrow row(u'a) \sqsubseteq row(ua)$$

$\mathcal{T}$ 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
  - $row(b) \sqsubseteq row(abb)$

return

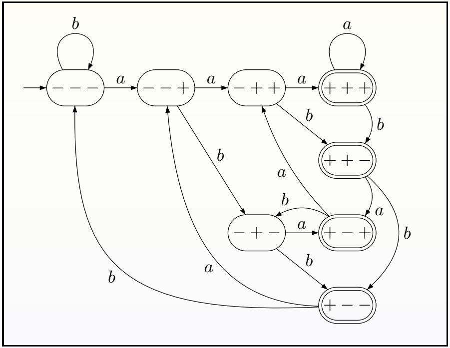
The algorithm in action



	$\mathcal{T}_1$	ε
*	ε	−
*	a	−
*	b	−

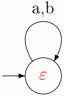
Create initial table $\mathcal{T}_1$.

The algorithm in action



$\mathcal{T}_1$	ε
* ε	−
* a	−
* b	−

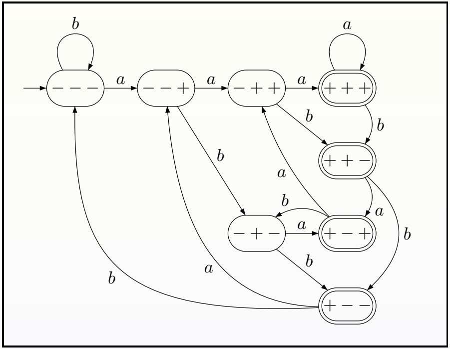
Hypothesis $\mathcal{R}_{\mathcal{T}_1}$:



$\Rightarrow$ Counterexample is aaa.

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

The algorithm in action

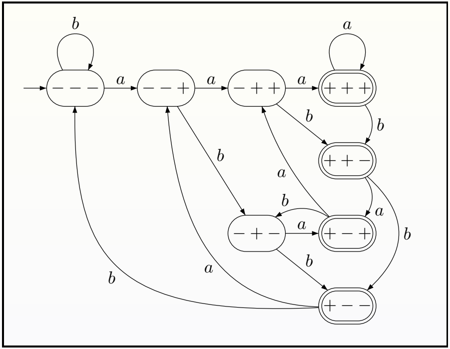


$\mathcal{T}_1$	ε
* ε	−
* a	−
* b	−

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

RWTH AACHEN UNIVERSITY

The algorithm in action

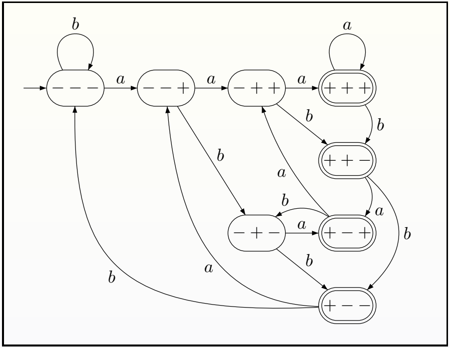


$\mathcal{T}_1$	ε
* ε	−
* a	−
* b	−

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

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

The algorithm in action

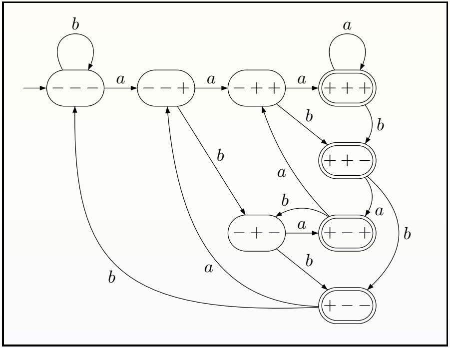


	$\mathcal{T}_1$	ε
*	ε	−
*	a	−
*	b	−

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

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

The algorithm in action



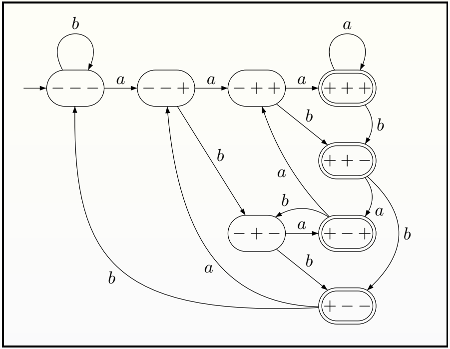
$\mathcal{T}_3$ is not closed:
$row(ab) \notin Primes_{\text{upp}}$

	$\mathcal{T}_1$	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



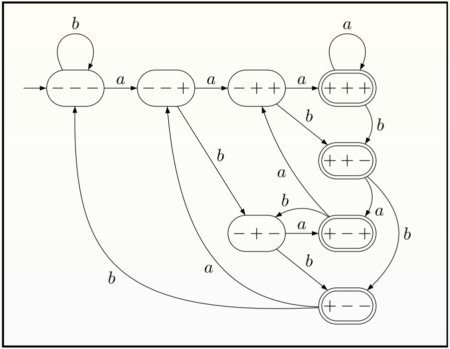
$\mathcal{T}_1$	ε
* ε	−
* a	−
* b	−

$\mathcal{T}_2$	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:
$row(abb) \notin Primes_{\text{upp}}$

	$\mathcal{T}_1$	ε
*	ε	−
*	a	−
*	b	−

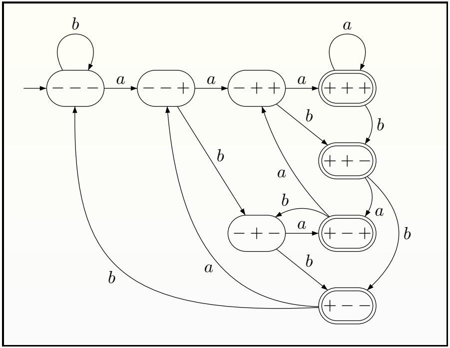
	$\mathcal{T}_2$	ε	aaa	aa	a
*	ε	−	+	−	−
*	a	−	+	+	−
*	b	−	+	−	−

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

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

RWTH AACHEN UNIVERSITY

The algorithm in action



	$\mathcal{T}_1$	ε
*	ε	−
*	a	−
*	b	−

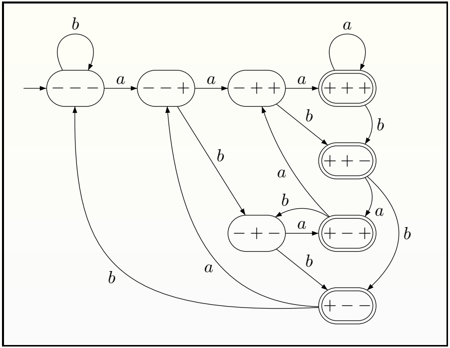
	$\mathcal{T}_2$	ε	aaa	aa	a
*	ε	−	+	−	−
*	a	−	+	+	−
*	b	−	+	−	−

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

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

	$\mathcal{T}_5$	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	−	+	−	−

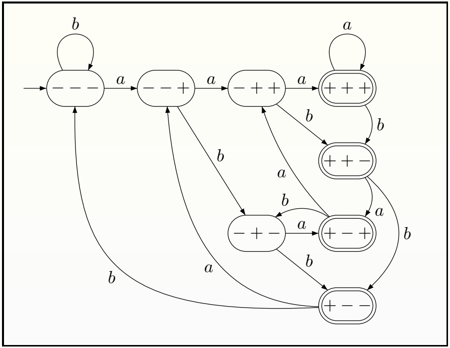
	$\mathcal{T}_3$	ε	aaa	aa	a
*	ε	−	+	−	−
*	a	−	+	+	−
*	b	−	+	−	−
	aa	−	+	+	+
*	ab	−	+	−	+

	$\mathcal{T}_4$	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	−	+	−	−

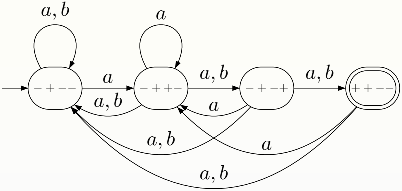
RWTH AACHEN UNIVERSITY

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	−	+	−	−