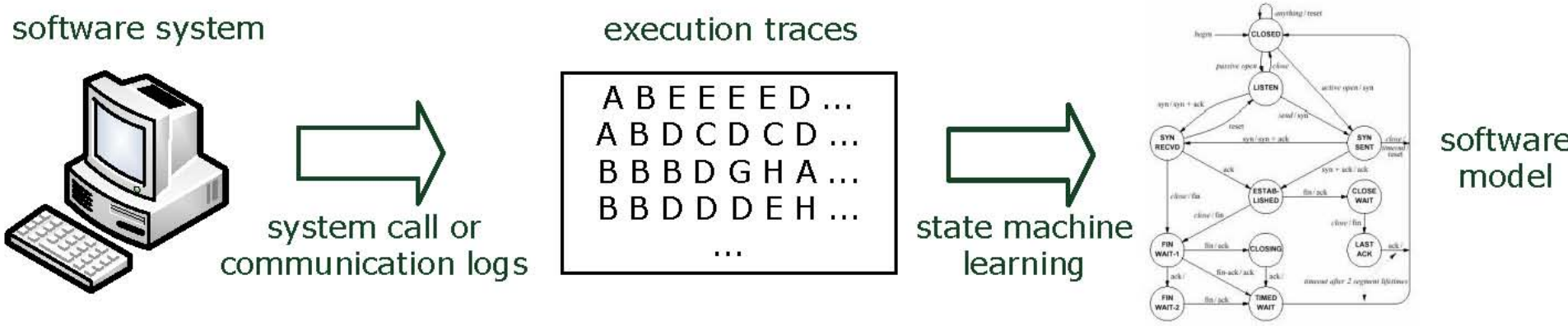
Learning State Machines

Myhill-Nerode, the Hankel matrix, Active and Passive learning

Sicco Verwer
http://cybersecurity.tudelft.nl/users/sicco-verwer

System identification and analysis



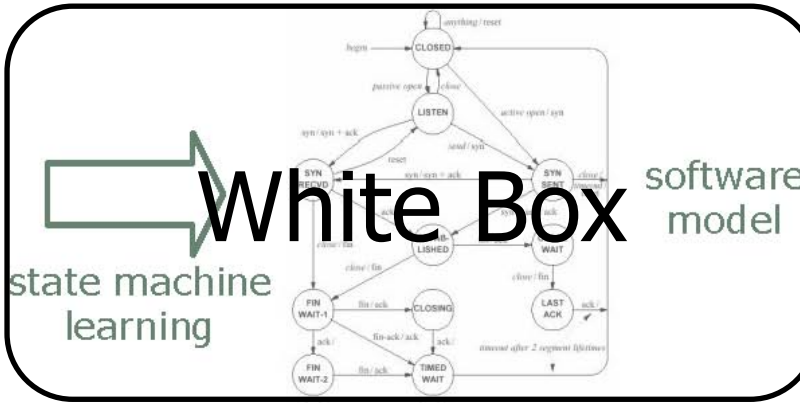
- Software leaves traces
- A state machine is a logical model describing these traces
  - Classification – is a new trace generated by the same software?
  - Prediction – what trace is most likely to occur next?
  - Analysis – is the software deadlock-free, secure, malicious?

System identification and analysis

Black Box

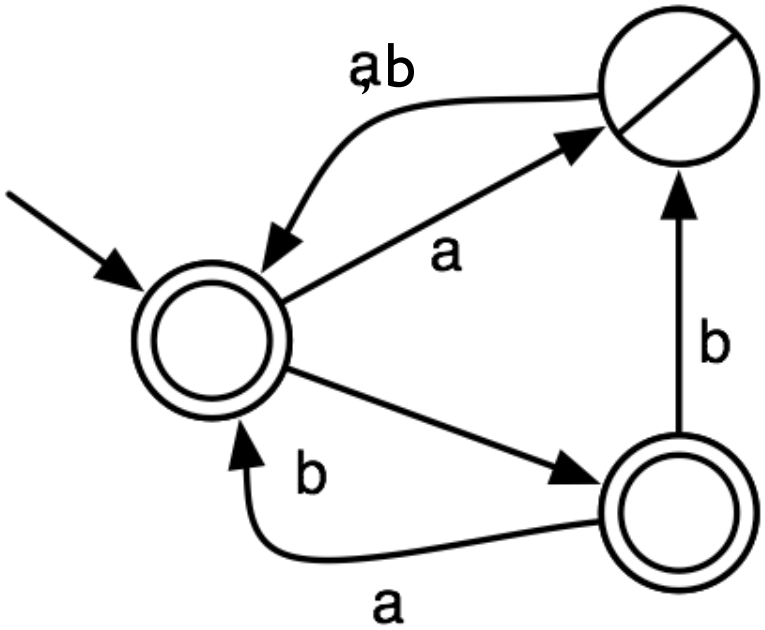
execution traces

A B E E E E D ...
A B D C D C D ...
B B B D G H A ...
B B D D D E H ...
...



- Software leaves traces
- A state machine is a logical model describing these traces
  - Classification – is a new trace generated by the same software?
  - Prediction – what trace is most likely to occur next?
  - Analysis – is the software deadlock-free, secure, malicious?

Deterministic finite state automata (DFA)



(a,-),(ab,+),(aa,+),(b,+),(bb,-),(bab,+)

State machine learning

- Input:
  - Data, e.g., (a,+) (b,-) (aba,+) (abaa,-) (bbb,+) (bba,+)
- Goal: Find a smallest DFA that is consistent with the data
- Size is measured by the number of states/transitions
  - Occam's razor:

  Among competing hypotheses, the one with the fewest assumptions should be selected.
- Consistency is:
  - Accepts all positive, rejects all negative

Example

accept:
1 0 0 0 0 0 0 0 0
0 1 1 0 0 0 1 0 1 0 1 0 1 0 0
0 0
1 0 1 0 0
1 0 0
1 0 1
1 0 1
1 0 0 0 0 0 0

reject:
0 1 0 0 0
1 1 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 1 0 0 0
0 1 1 1 0 1 0 0 0 0 0 0

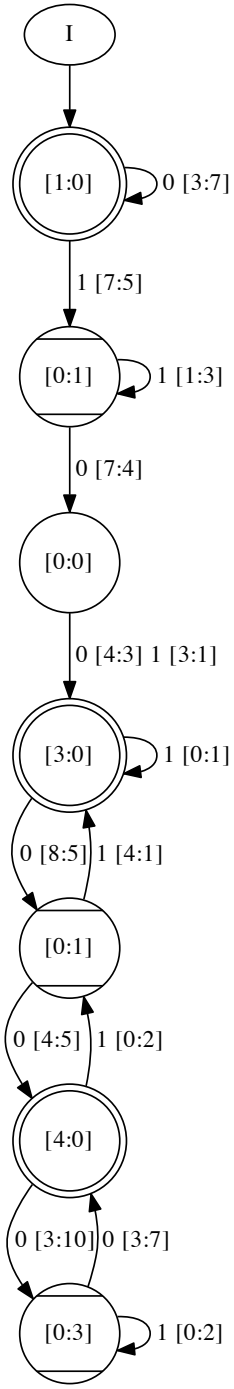
What is the smallest state machine that accepts the top and rejects the bottom strings?

Guess: how many states are needed?

Example

accept:
1 0 0 0 0 0 0 0 0
0 1 1 0 0 0 1 0 1 0 1 0 1 0 0
0 0
1 0 1 0 0
1 0 0
1 0 1
1 0 1
1 0 0 0 0 0 0

reject:
0 1 0 0 0
1 1 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 1 0 0 0
0 1 1 1 0 1 0 0 0 0 0 0 0



Example output by algorithm

accept:

1 0 0 0 0 0 0 0 0

0 1 1 0 0 0 1 0 1 0 1 0 1 0 0

0 0

1 0 1 0 0

1 0 0

1 0 1

1 0 1

1 0 0 0 0 0 0

reject:

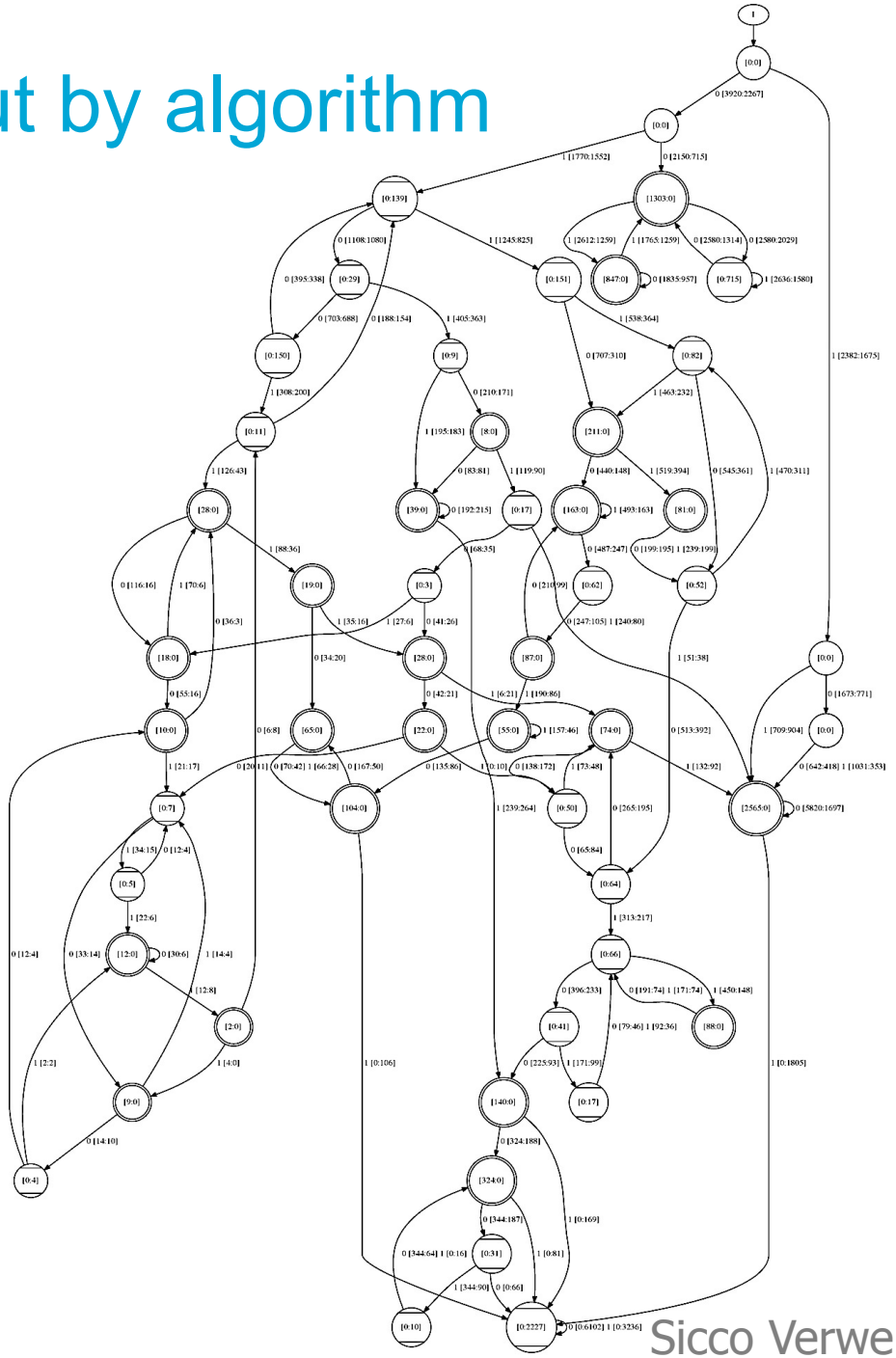
0 1 0 0 0

1 1 0 0 0 0 0 0 0 1 0 0

0 0 0 0 0 1

1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 1 0 0 0

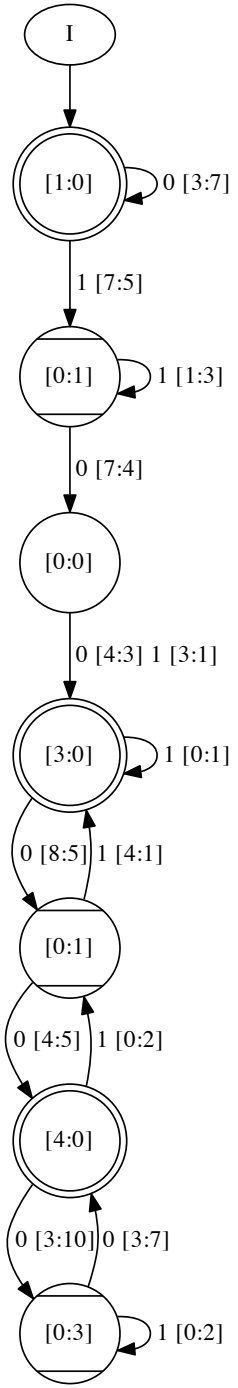
0 1 1 1 0 1 0 0 0 0 0 0



TU Delft … 10244 strings in total

Sicco Verwer

What does a state mean?



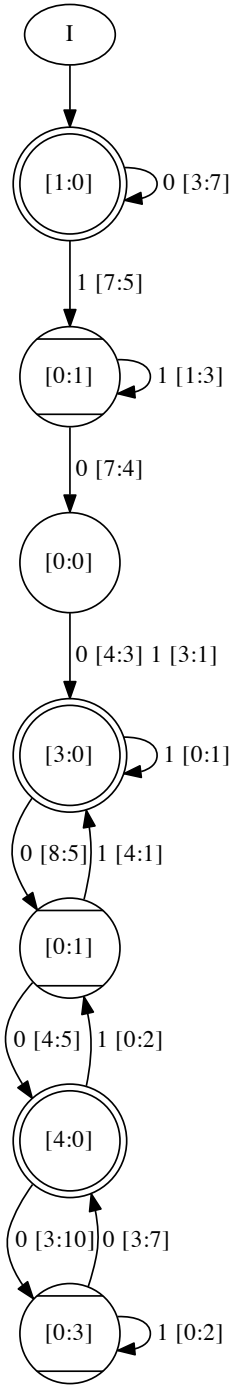
What does a state mean?

1. A state q represents possible futures
   all sequences starting in q

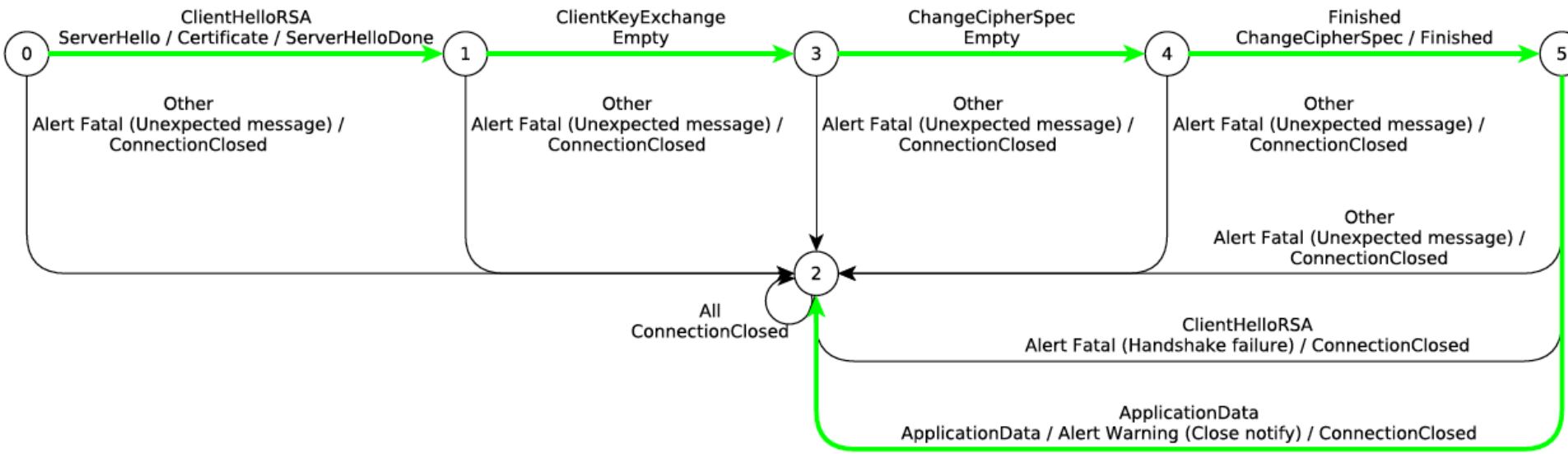
2. A state q is independent of the past
   no matter how q is reached, the futures remain the same

Intuitively, a state means the condition of a system, e.g.:

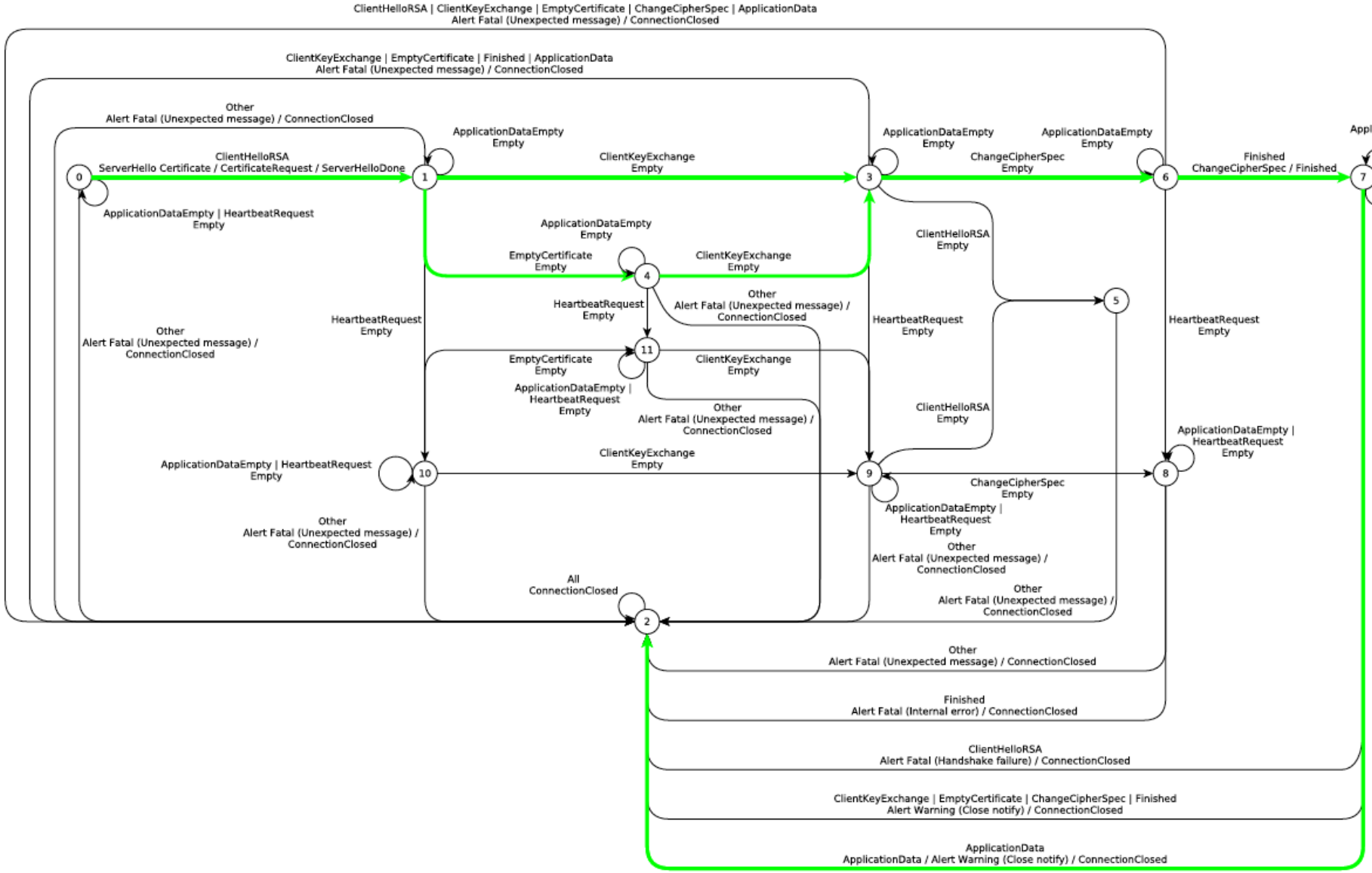
- initial states
- working states
- error states
- …



Use Case: TLS RSA BSAFE



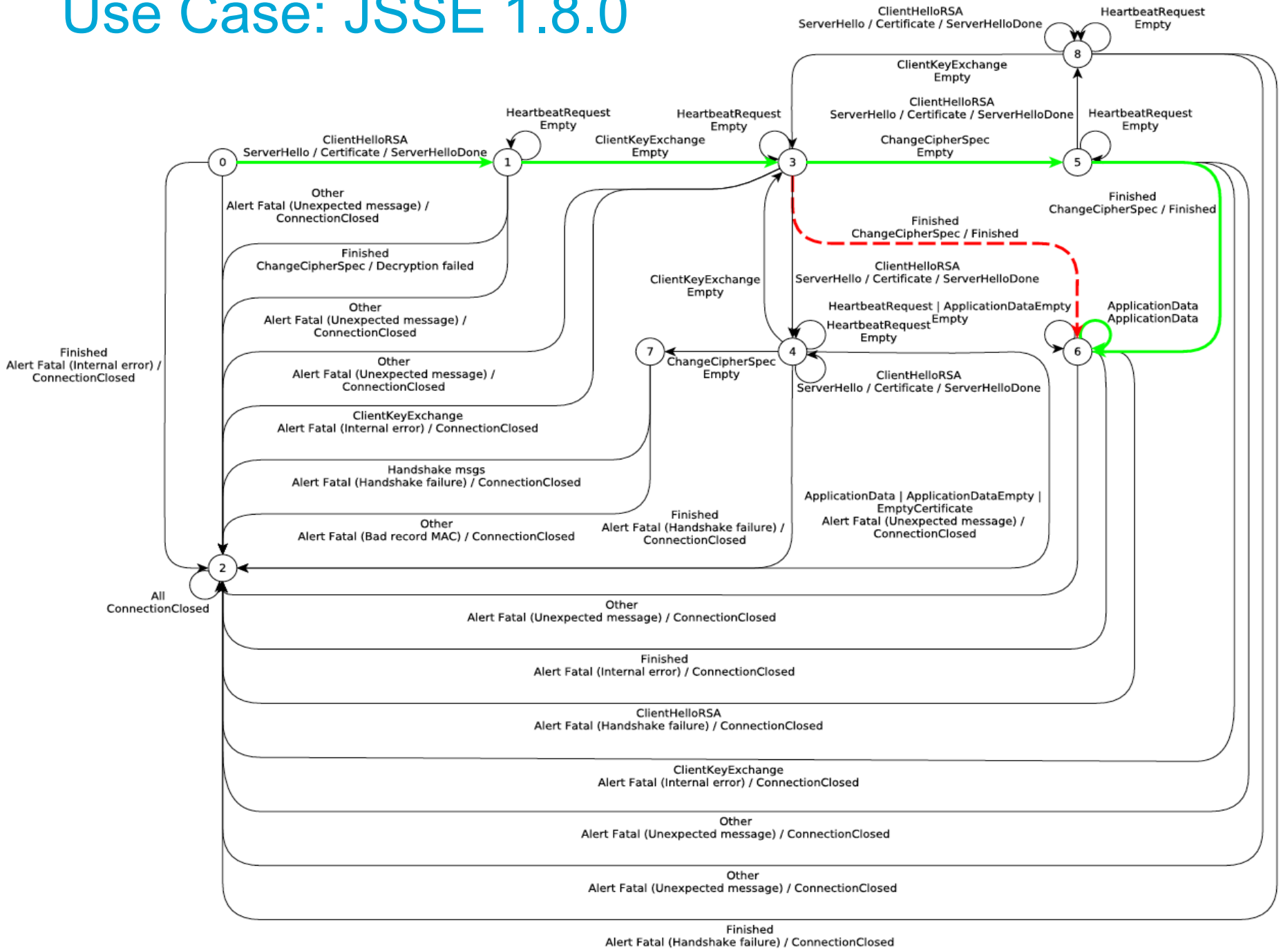
Use Case: GNU TLS 3.3.8



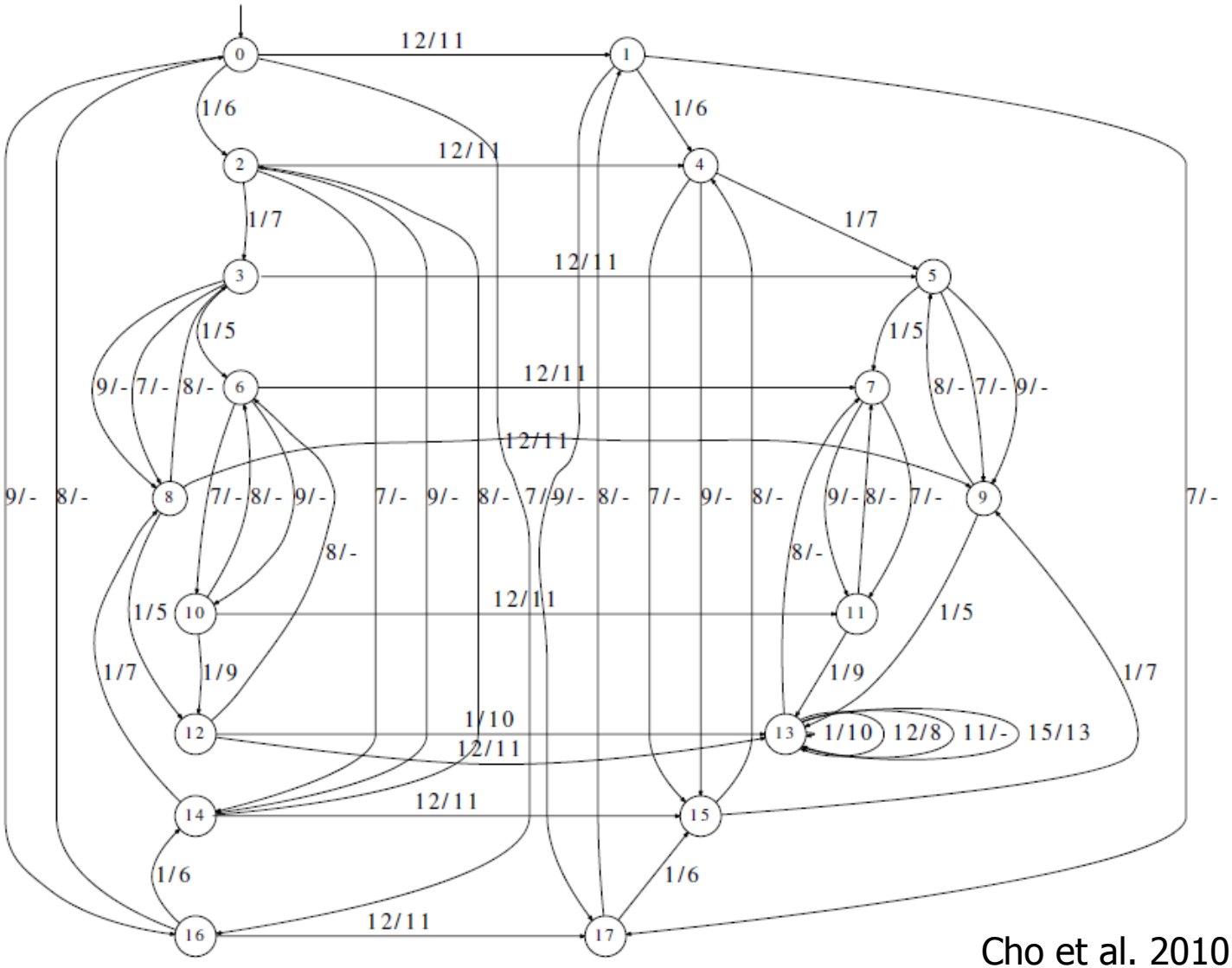
TUDelft

Joeri de Ruiter & Erik Poll 2015

Use Case: JSSE 1.8.0

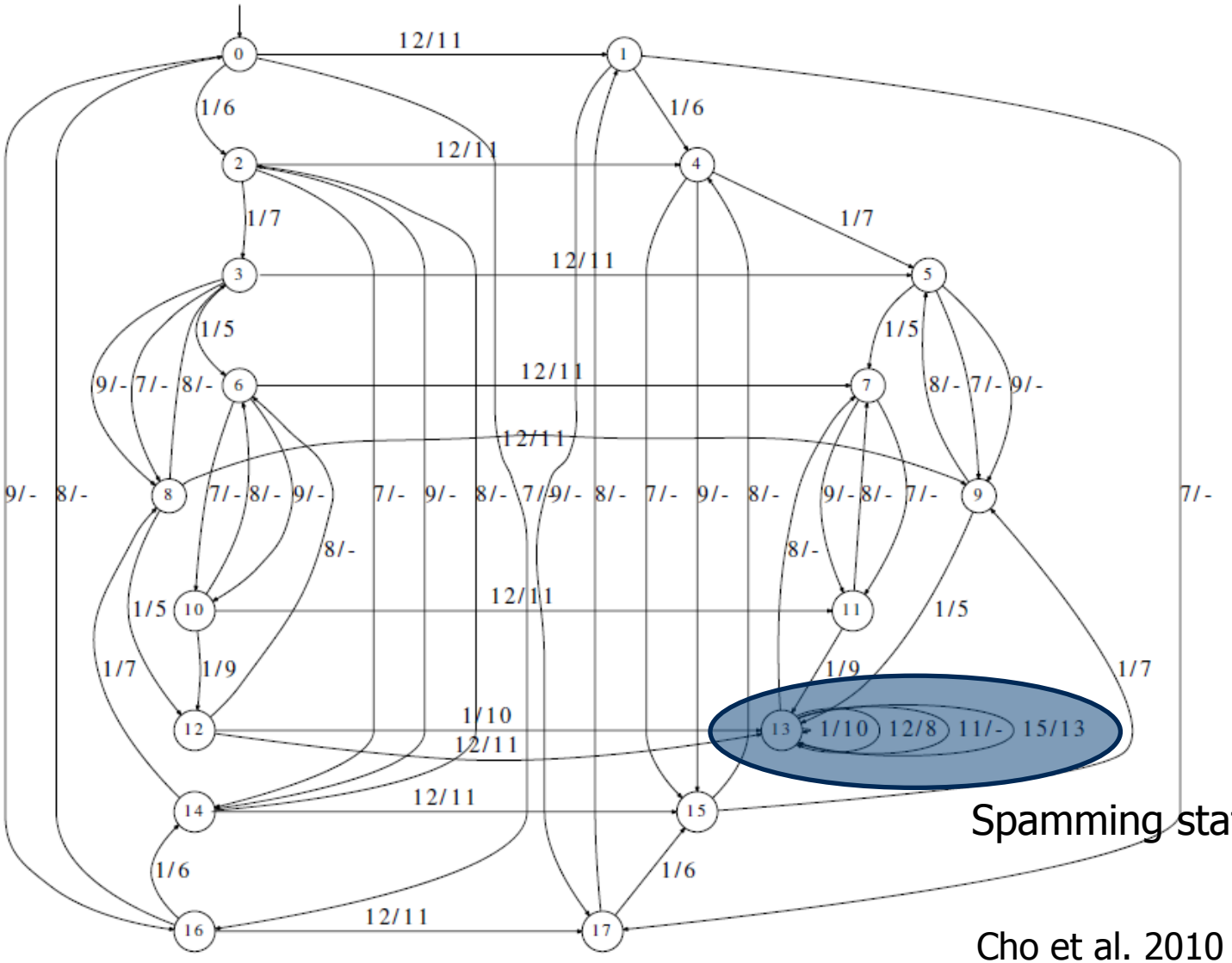


Use Case: MegaD botnet protocol



Cho et al. 2010

Use Case: MegaD botnet protocol



Spamming state

Cho et al. 2010

Today – passive learning

- Key theory: Myhill Nerode
  - Language Hankel Matrix
- Formalizing the problem: state merging
  - Basic algorithm for learning state machines
- What to do when data is unlabeled?
  - Replace Myhill-Nerode by Markov
- Tools, open issues, problems, questions, …

Key Theory: Myhill-Nerode

- A necessary and sufficient condition (if and only if) for a language to be regular
  - Characterises when finite state machines exist
  - We use it for learning such state machines

Myhill-Nerode

- Given a language L, and a pair of strings x and y, define a distinguishing extension to be a string z such that exactly one of the two strings xz and yz belong to L.
- Define a relation R on the strings by the rule that
    - x R y
    - if there is no distinguishing extension for x and y.
- It is easy to show that R is an equivalence relation.
- L is regular if and only if R has a finite number of equivalence classes (sets of R-equivalent strings).
- Moreover, the number of states of the smallest deterministic finite state automaton recognizing L is equal to the number of equivalence classes in R.

Myhill-Nerode

- Given a language L, and a pair of strings x and y, define a distinguishing extension to be a string z such that exactly one of the two strings xz and yz belong to L.
- Define a relation R on the strings by the rule that
  - x R y
  - if there is no distinguishing extension for x and y.
- It is easy to show that R is an equivalence relation.
- L is regular if and only if R has a finite number of equivalence classes (sets of R-equivalent strings).
- Moreover, the number of states of the smallest deterministic finite state automaton recognizing L is equal to the number of equivalence classes in R.

Visualization

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
aa										
ba										

- x = aa
- y = ba
- z = column

two strings aa and ba
and possible distinguishing extensions

Visualization

a(a|b)*b

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
aa										
ba										

Visualization

a(a|b)*b

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
aa	0									
ba										

Visualization

a(a|b)*b

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
aa	0									
ba	0									

Visualization

a(a|b)*b

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
aa	0	0								
ba	0	0								

Visualization

a(a|b)*b

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
aa	0	0	1							
ba	0	0	0							

Visualization

a(a|b)*b

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
aa	0	0	1	0	1	0	1	0	1	0
ba	0	0	0	0	0	0	0	0	0	0

Myhill-Nerode

- Given a language L, and a pair of strings x and y, define a distinguishing extension to be a string z such that exactly one of the two strings xz and yz belong to L.
- Define a relation R on the strings by the rule that
  - x R y
  - if there is no distinguishing extension for x and y.
- It is easy to show that R is an equivalence relation.
- L is regular if and only if R has a finite number of equivalence classes (sets of R-equivalent strings).
- Moreover, the number of states of the smallest deterministic finite state automaton recognizing L is equal to the number of equivalence classes in R.

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0	0	0	0	1	0	0	0	1	0
a	0	0	1	0	1	0	1	0	1	0
b	0	0	0	0	0	0	0	0	0	0
aa	0	0	1	0	1	0	1	0	1	0
ab	1	0	1	0	1	0	1	0	1	0
ba	0	0	0	0	0	0	0	0	0	0
bb	0	0	0	0	0	0	0	0	0	0
aaa	0	0	1	0	1	0	1	0	1	0
aab	1	0	1	0	1	0	1	0	1	0
aba	0	0	1	0	1	0	1	0	1	0

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0	0	0	0	1	0	0	0	1	0
a	0	0	1	0	1	0	1	0	1	0
b	0	0	0	0	0	0	0	0	0	0
aa	0								1	0
ab	1								1	0
ba	0								0	0
bb	0	0	0	0	0	0	0	0	0	0
aaa	0	0	1	0	1	0	1	0	1	0
aab	1	0	1	0	1	0	1	0	1	0
aba	0	0	1	0	1	0	1	0	1	0

Hankel matrix of language L

Hankel matrix

- Rows are prefixes

  pasts

- Columns are suffixes

  futures

- Row-column (x,y) is:
  - 1 if xy in L
  - 0 if xy not in L

  outputs

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0	0	0	0	1	0	0	0	1	0
a	0	0	1	0	1	0	1	0	1	0
b	0	0	0	0	0	0	0	0	0	0
aa	0	0	1	0	1	0	1	0	1	0
ab	1	0	1	0	1	0	1	0	1	0
ba	0	0	0	0	0	0	0	0	0	0
bb	0	0	0	0	0	0	0	0	0	0
aaa	0	0	1	0	1	0	1	0	1	0
aab	1	0	1	0	1	0	1	0	1	0
aba	0	0	1	0	1	0	1	0	1	0

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0	0	0	0	1	0	0	0	1	0
a	0	0	1	0	1	0	1	0	1	0
b	0	0	0	0	0	0	0	0	0	0
aa	0	0	1	0	1	0	1	0	1	0
ab	1	0	1	0	1	0	1	0	1	0
ba	0	0	0	0	0	0	0	0	0	0
bb	0	0	0	0	0	0	0	0	0	0
aaa	0	0	1	0	1	0	1	0	1	0
aab	1	0	1	0	1	0	1	0	1	0
aba	0	0	1	0	1	0	1	0	1	0

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0	0	0	0	1	0	0	0	1	0
a	0	0	1	0	1	0	1	0	1	0
b	0	0	0	0	0	0	0	0	0	0
aa	0	0	1	0	1	0	1	0	1	0
ab	1	0	1	0	1	0	1	0	1	0
ba	0	0	0	0	0	0	0	0	0	0
bb	0	0	0	0	0	0	0	0	0	0
aaa	0	0	1	0	1	0	1	0	1	0
aab	1	0	1	0	1	0	1	0	1	0
aba	0	0	1	0	1	0	1	0	1	0

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0	0	0	0	1	0	0	0	1	0
a	0	0	1	0	1	0	1	0	1	0
b	0	0	0	0	0	0	0	0	0	0
aa	0	0	1	0	1	0	1	0	1	0
ab	1	0	1	0	1	0	1	0	1	0
ba	0	0	0	0	0	0	0	0	0	0
bb	0	0	0	0	0	0	0	0	0	0
aaa	0	0	1	0	1	0	1	0	1	0
aab	1	0	1	0	1	0	1	0	1	0
aba	0	0	1	0	1	0	1	0	1	0

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0	0	0	0	1	0	0	0	1	0
a	0	0	1	0	1	0	1	0	1	0
b	0	0	0	0	0	0	0	0	0	0
aa	0	0	1	0	1	0	1	0	1	0
ab	1	0	1	0	1	0	1	0	1	0
ba	0	0	0	0	0	0	0	0	0	0
bb	0	0	0	0	0	0	0	0	0	0
aaa	0	0	1	0	1	0	1	0	1	0
aab	1	0	1	0	1	0	1	0	1	0
aba	0	0	1	0	1	0	1	0	1	0

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0	0	0	0	1	0	0	0	1	0
a	0	0	1	0	1	0	1	0	1	0
b	0	0	0	0	0	0	0	0	0	0
aa										0
ab										0
ba										0
bb	0	0	0	0	0	0	0	0	0	0
aaa	0	0	1	0	1	0	1	0	1	0
aab	1	0	1	0	1	0	1	0	1	0
aba	0	0	1	0	1	0	1	0	1	0

x R y
if row x and row y have the same color

Myhill-Nerode

- Given a language L, and a pair of strings x and y, define a distinguishing extension to be a string z such that exactly one of the two strings xz and yz belong to L.
- Define a relation R on the strings by the rule that
  - x R y
  - if there is no distinguishing extension for x and y.
- It is easy to show that R is an equivalence relation.
- L is regular if and only if R has a finite number of equivalence classes (sets of R-equivalent strings).
- Moreover, the number of states of the smallest deterministic finite state automaton recognizing L is equal to the number of equivalence classes in R.

Myhill-Nerode

- Given a language L, and a pair of strings x and y, define a distinguishing extension to be a string z such that exactly one of the two strings xz and yz belong to L.
- Define a relation R on the strings by the rule that
    - x R y
    - if there is no distinguishing extension for x and y.
- It is easy to show that R is an equivalence relation.
- L is regular if and only if R has a finite number of equivalence classes (sets of R-equivalent strings).
- Moreover, the number of states of the smallest deterministic finite state automaton recognizing L is equal to the number of equivalence classes in R.

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0	0	0	0	1	0	0	0	1	0
a	0	0	1	0	1	0	1	0	1	0
b	0	0	0	0	0	0	0	0	0	0
aa	0	0	1	0	1	0	1	0	1	0
ab	1	0	1	0	1	0	1	0	1	0
ba	0	0	0	0	0	0	0	0	0	0
bb	0	0	0	0	0	0	0	0	0	0
aaa	0	0	1	0	1	0	1	0	1	0
aab	1	0	1	0	1	0	1	0	1	0
aba	0	0	1	0	1	0	1	0	1	0

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0	0	0	0	1	0	0	0	1	0
a	0	0	1	0	1	0	1	0	1	0
aaa	0	0	1	0	1	0	1	0	1	0
aab	1	0	1	0	1	0	1	0	1	0
aba	0	0	1	0	1	0	1	0	1	0

A language is regular if and only if the Hankel matrix contains a finite number of distinct rows, i.e., colors

Myhill-Nerode

- Given a language L, and a pair of strings x and y, define a distinguishing extension to be a string z such that exactly one of the two strings xz and yz belong to L.
- Define a relation R on the strings by the rule that
  - x R y
  - if there is no distinguishing extension for x and y.
- It is easy to show that R is an equivalence relation.
- L is regular if and only if R has a finite number of equivalence classes (sets of R-equivalent strings).
- Moreover, the number of states of the smallest deterministic finite state automaton recognizing L is equal to the number of equivalence classes in R.

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0	0	0	0	1	0	0	0	1	0
a	0	0	1	0	1	0	1	0	1	0
a										
a										
b										
b										
aaa	0	0	1	0	1	0	1	0	1	0
aab	1	0	1	0	1	0	1	0	1	0
aba	0	0	1	0	1	0	1	0	1	0

The number of state of the smallest DFA for L is
the number of colors in the Hankel matrix

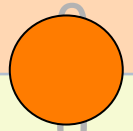
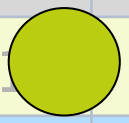
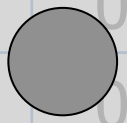
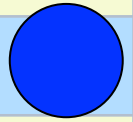
TUDelft

Q: what does this state machine look like?

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0	0	0	0	1	0	0	0	1	0
a	0	0	1	0	1	0	1	0	1	0
b	0	0	0	0	0	0	0	0	0	0
aa	0	0	1	0	1	0	1	0	1	0
ab	1	0	1	0	1	0	1	0	1	0
ba	0	0	0	0	0	0	0	0	0	0
bb	0	0	0	0	0	0	0	0	0	0
aaa	0	0	1	0	1	0	1	0	1	0
aab	1	0	1	0	1	0	1	0	1	0
aba	0	0	1	0	1	0	1	0	1	0

Q: what does this state machine look like?

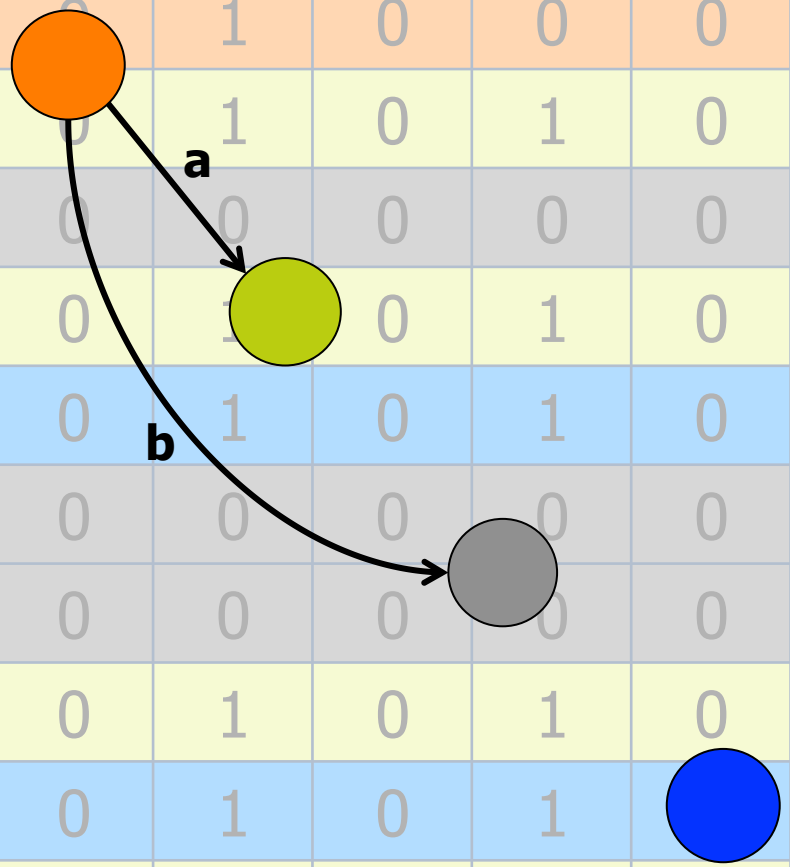
	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0	0	0	0	1	0	0	0	1	0
a	0	0	1	0	1	0	1	0	1	0
b	0	0	0	0	0	0	0	0	0	0
aa	0	0	1	0	1	0	1	0	1	0
ab	1	0	1	0	1	0	1	0	1	0
ba	0	0	0	0	0	0	0	0	0	0
bb	0	0	0	0	0	0	0	0	0	0
aaa	0	0	1	0	1	0	1	0	1	0
aab	1	0	1	0	1	0	1	0	1	0
aba	0	0	1	0	1	0	1	0	1	0

TU Delft

Q: what does this state machine look like?

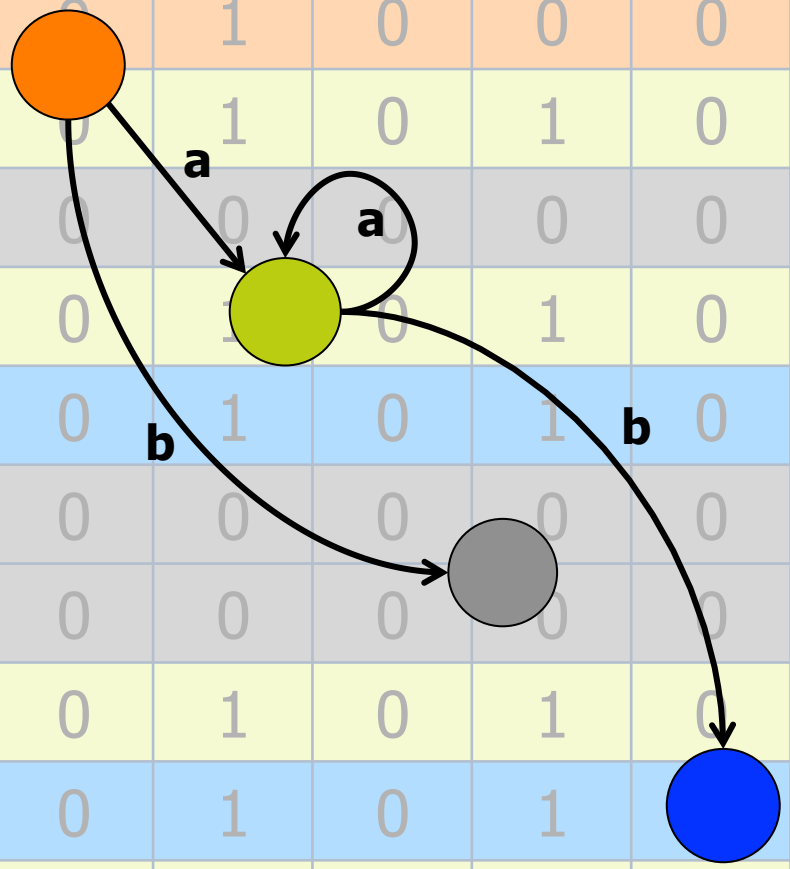
	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0	0	0	0	1	0	0	0	1	0
a	0	0	1	0	1	0	1	0	1	0
b	0	0	0	0	0	0	0	0	0	0
aa	0	0	1	0	1	0	1	0	1	0
ab	1	0	1	0	1	0	1	0	1	0
ba	0	0	0	0	0	0	0	0	0	0
bb	0	0	0	0	0	0	0	0	0	0
aaa	0	0	1	0	1	0	1	0	1	0
aab	1	0	1	0	1	0	1	0	1	0
aba	0	0	1	0	1	0	1	0	1	0



TU Delft

Q: what does this state machine look like?

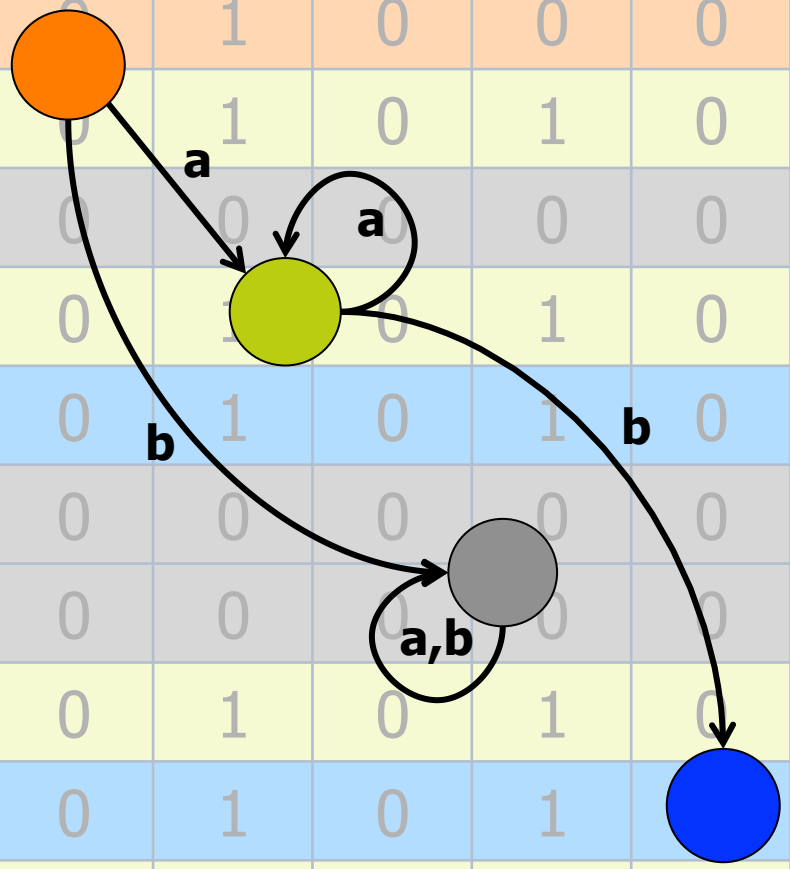
	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0	0	0	0	1	0	0	0	1	0
a	0	0	1	0	1	0	1	0	1	0
b	0	0	0	0	0	0	0	0	0	0
aa	0	0	1	0	1	0	1	0	1	0
ab	1	0	1	0	1	0	1	0	1	0
ba	0	0	0	0	0	0	0	0	0	0
bb	0	0	0	0	0	0	0	0	0	0
aaa	0	0	1	0	1	0	1	0	1	0
aab	1	0	1	0	1	0	1	0	1	0
aba	0	0	1	0	1	0	1	0	1	0



TUDelft

Q: what does this state machine look like?

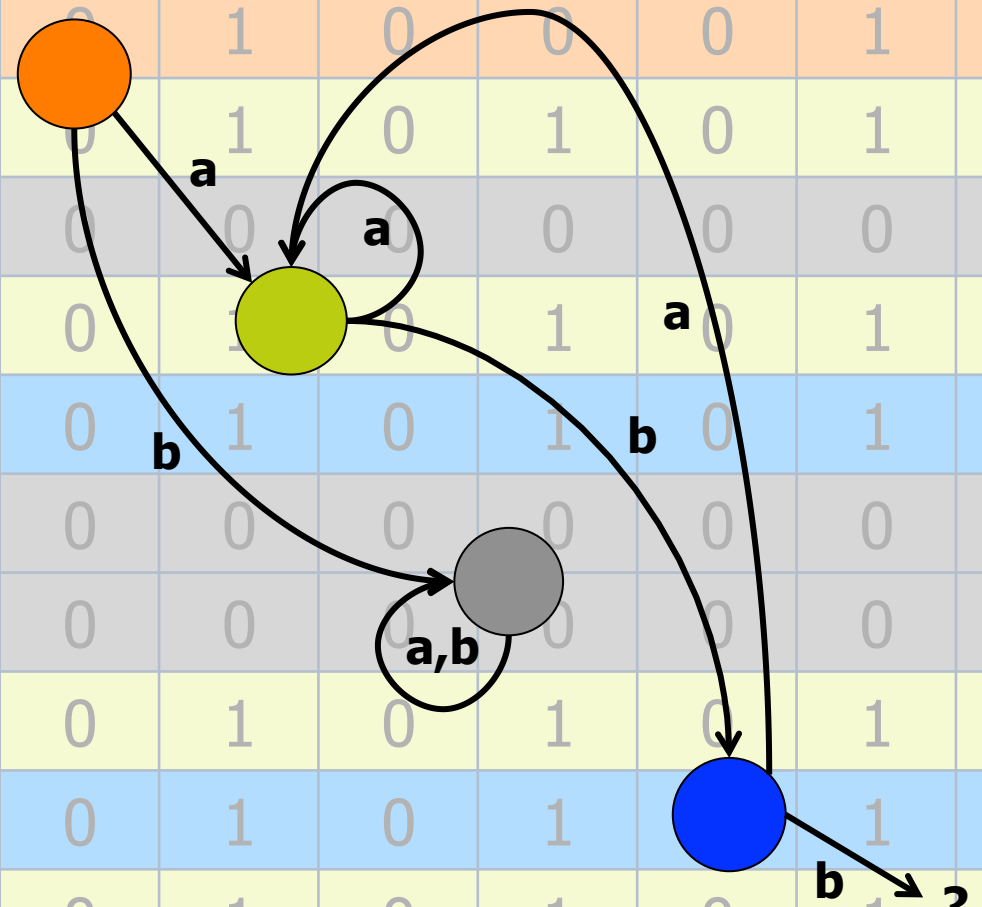
	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0	0	0	0	1	0	0	0	1	0
a	0	0	1	0	1	0	1	0	1	0
b	0	0	0	0	0	0	0	0	0	0
aa	0	0	1	0	1	0	1	0	1	0
ab	1	0	1	0	1	0	1	0	1	0
ba	0	0	0	0	0	0	0	0	0	0
bb	0	0	0	0	0	0	0	0	0	0
aaa	0	0	1	0	1	0	1	0	1	0
aab	1	0	1	0	1	0	1	0	1	0
aba	0	0	1	0	1	0	1	0	1	0



TU Delft

Q: what does this state machine look like?

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0	0	0	0	1	0	0	0	1	0
a	0	0	1	0	1	0	1	0	1	0
b	0	0	0	0	0	0	0	0	0	0
aa	0	0	1	0	1	0	1	0	1	0
ab	1	0	1	0	1	0	1	0	1	0
ba	0	0	0	0	0	0	0	0	0	0
bb	0	0	0	0	0	0	0	0	0	0
aaa	0	0	1	0	1	0	1	0	1	0
aab	1	0	1	0	1	0	1	0	1	0
aba	0	0	1	0	1	0	1	0	1	0



TUDelft

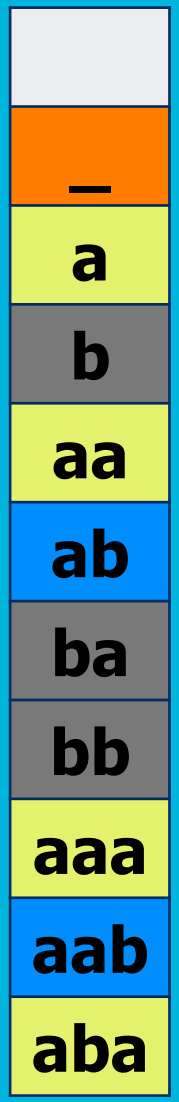
Using state machines

- Analyzing the models by hand, or with model checker, for flaws
  - to see if *all paths* are correct & secure
- Fuzzing or model-based testing
  - using the diagram as basis for “deeper” fuzz testing
- Program verification
  - *proving* that there is no functionality beyond that in the diagram, which using testing you can never establish
- Using it when doing a manual code review

Q: what does this state machine look like?

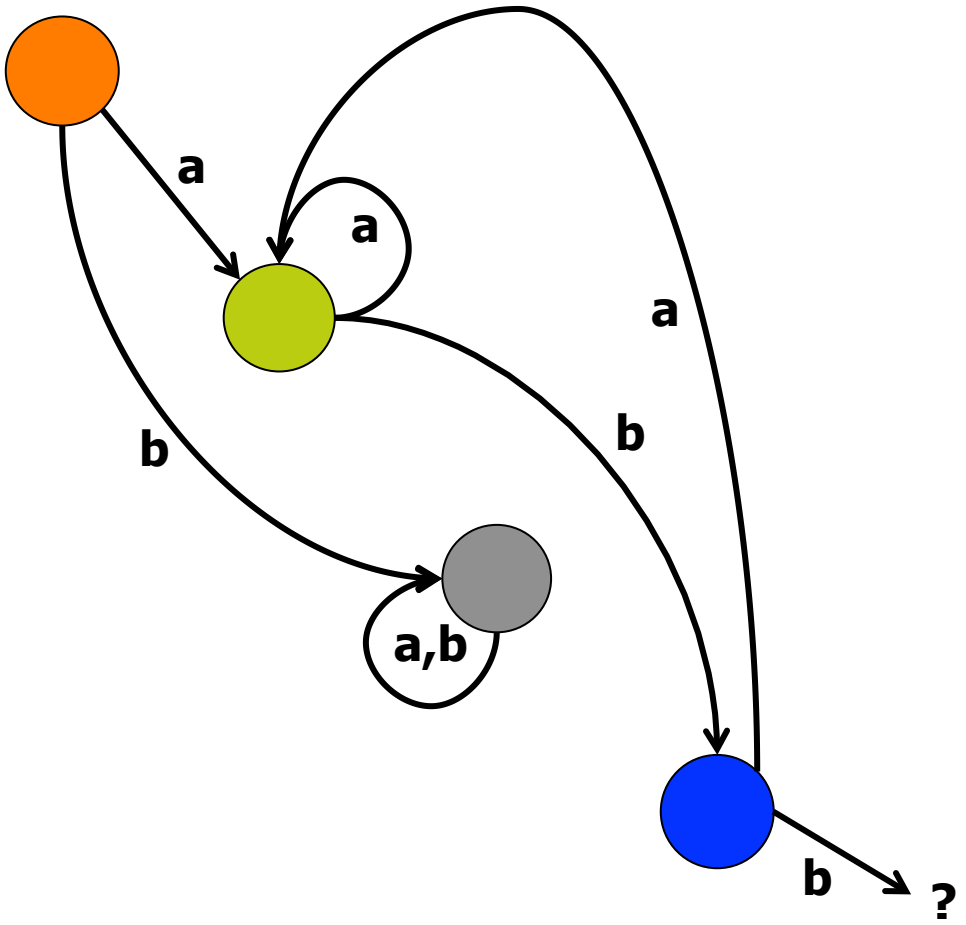
	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0	0	0	0	1	0	0	0	1	0
a	0	0	1	0	1	0	1	0	1	0
b	0	0	0	0	0	0	0	0	0	0
aa	0	0	1	0	1	0	1	0	1	0
ab	1	0	1	0	1	0	1	0	1	0
ba	0	0	0	0	0	0	0	0	0	0
bb	0	0	0	0	0	0	0	0	0	0
aaa	0	0	1	0	1	0	1	0	1	0
aab	1	0	1	0	1	0	1	0	1	0
aba	0	0	1	0	1	0	1	0	1	0

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	1	0	1	1	1	1	0	0	1	0
a	0	1	1	0	1	0	1	1	1	1
b	1	1	0	0	1	1	1	1	1	1
aa	1	0	1	1	1	1	0	0	1	0
ab	1	0	1	1	1	1	0	0	1	0
ba	1	0	1	1	1	1	0	0	1	0
bb	0	1	1	0	1	0	1	0	1	1
aaa	0	1	1	0	1	0	1	1	1	1
aab	1	1	0	0	1	1	1	1	1	1
aba	0	1	1	0	1	0	1	1	1	1



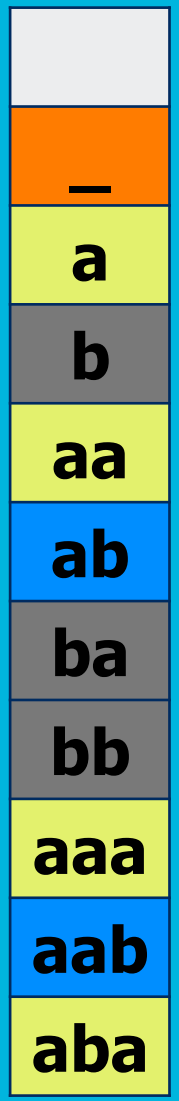
Look at colors of Hankel Matrix rows

access strings and *one-letter extensions* provide sufficient information



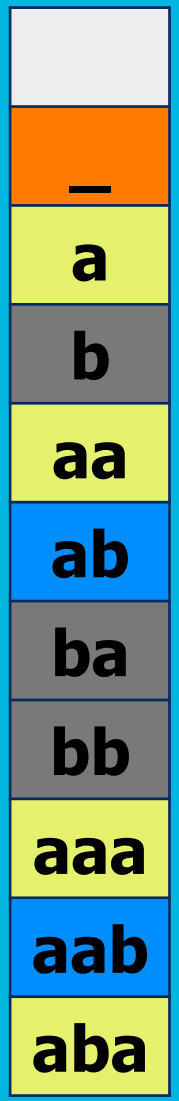
access string: shortest path reaching a state

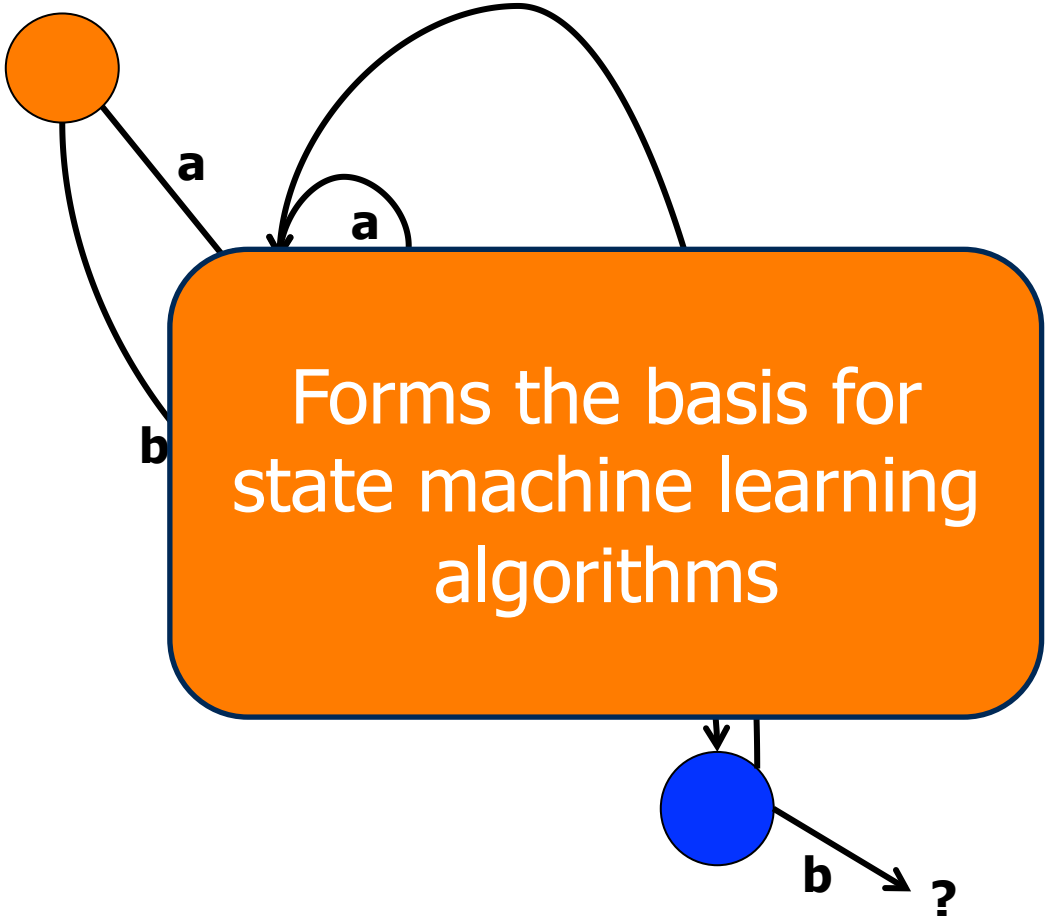
one-letter extension: access string + a single symbol



TU Delft

access strings and *one-letter extensions* provide sufficient information





access string: shortest path reaching a state

one-letter extension: access string + a single symbol

TU Delft

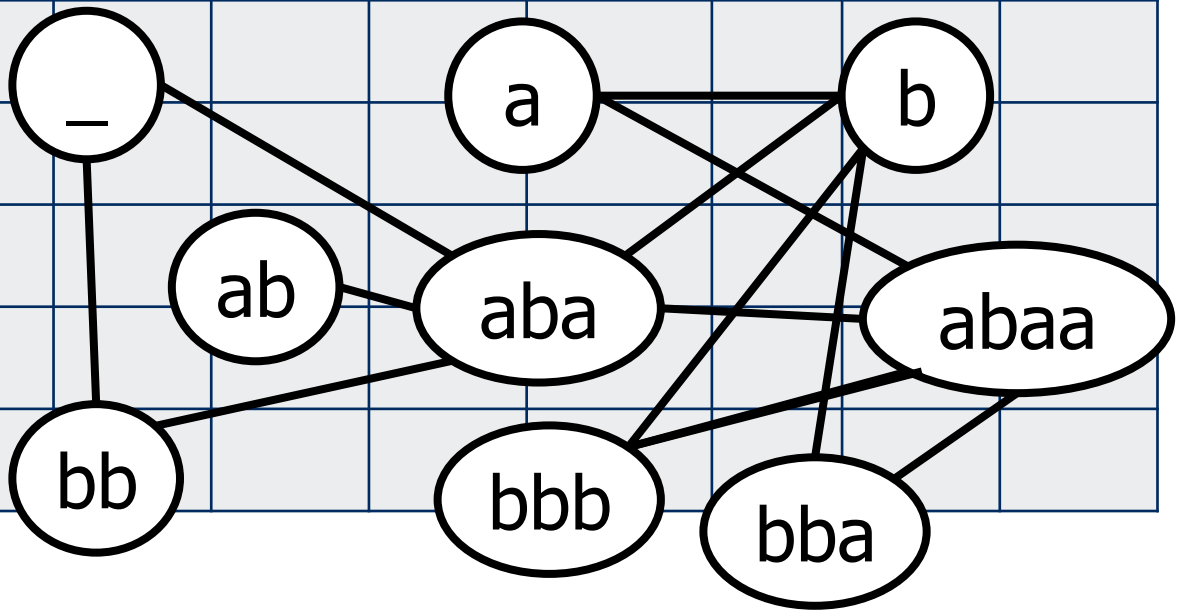
STATE MACHINE LEARNING

(a,+) (b,-) (aba,+) (abaa,-) (bbb,+) (bba,+)

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0			1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										

(a,+) (b,-) (aba,+) (abaa,-) (bbb,+) (bba,+)

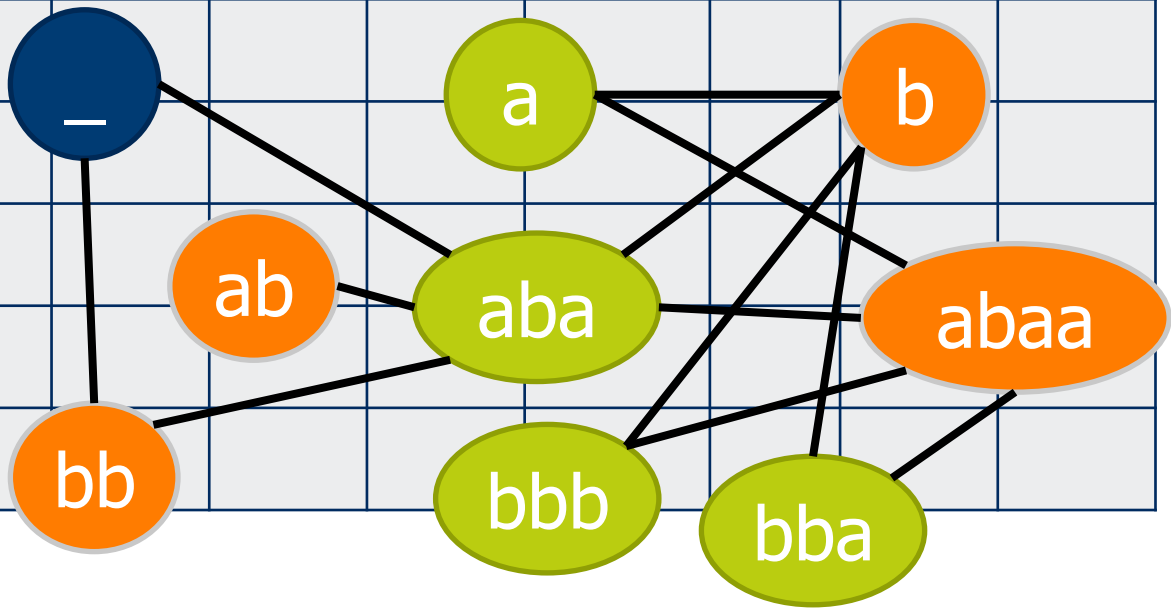
	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0			1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										



TUDelft

(a,+) (b,-) (aba,+) (abaa,-) (bbb,+) (bba,+)

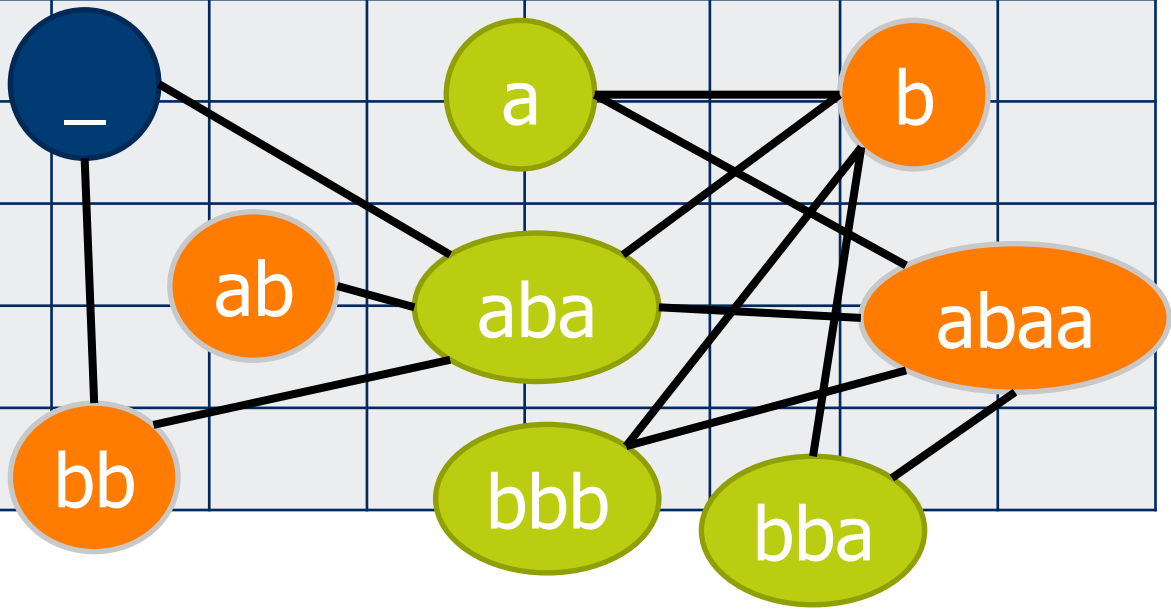
	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0			1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										



(a,+) (b,-) (aba,+) (abaa,-) (bbb,+) (bba,+)

						aa	baa	abaa	bb	bbb	bba
								0		1	1
							0				
									1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										

Try to make state machine...

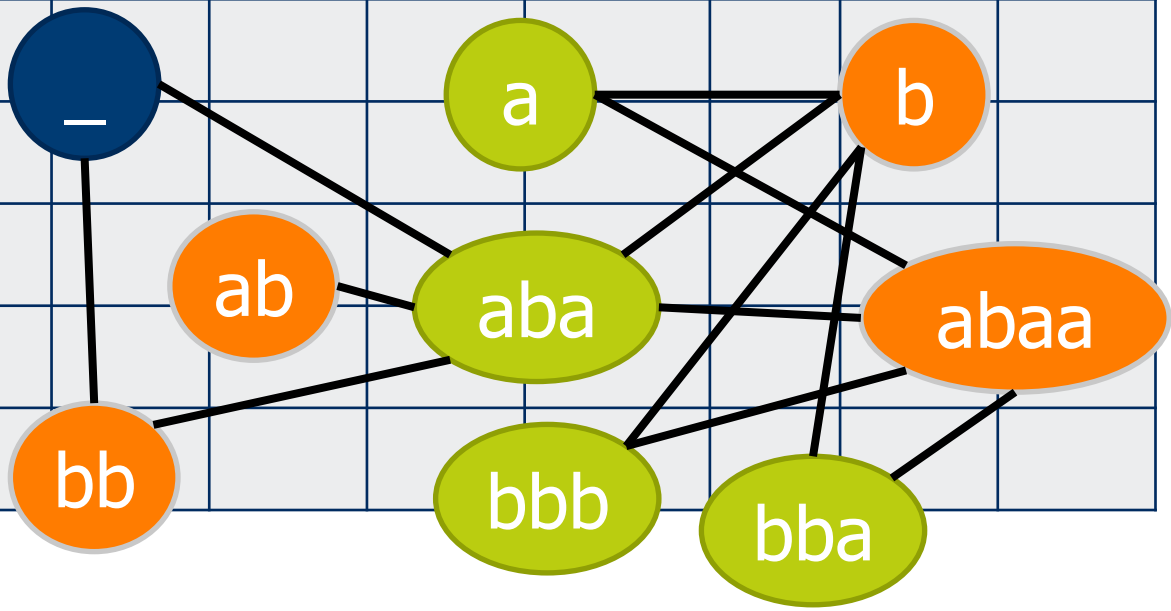


TU Delft

(a,+) (b,-) (aba,+) (abaa,-) (bbb,+) (bba,+)

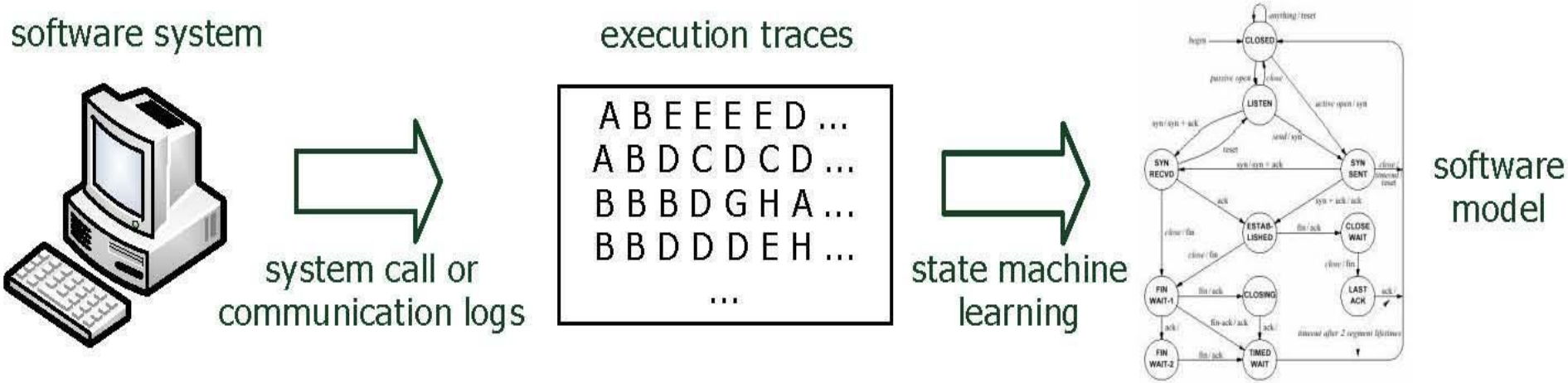
						aa	baa	abaa	bb	bbb	bba
								0		1	1
							0				
									1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										

Try to make state machine...
It is inconsistent!
(b and bb end in same state, but bbb does not)



TU Delft

We need more information…



	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0			1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										

What to ask?

Ask: bb

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0			1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										

Ask: bb

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0		1	1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb	1	1	1								
bbb	1										
bba	1										

If + _ distinguishes between b and bb

Ask: bb

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0		0	1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb	0	1	1								
bbb	1										
bba	1										

If - b distinguishes between b and bb

TUDelft

Active learning with L*

- Aims to discover:
  - access strings, smallest strings for every color
  - one-letter extensions, and
  - their distinguishing suffixes
- Maintains a distinguishing table
- Which distinguishing suffixes is determined during learning

	_	a
_	1	0
a	0	1
b	1	1
aa	1	0
ab	1	0
ba	1	0
bb	0	1

Active learning with L*

- Start with an empty table, with one access string and one distinguishing extension:
  - _ and _
- Ask membership queries for _ and its one letter extensions:
  - _ in L? → yes
  - a in L? → no
  - b in L? → yes

	_
_	1
a	0
b	1

Active learning with L*

- Can we construct a state machine?
  - No!
  - a points to a new state
- Ask membership queries for its one-letter extensions:
  - aa in L? → yes
  - ab in L? → yes

	_
_	1
a	0
b	1
aa	1
ab	1

Active learning with L*

- Can we construct a state machine?
  - Yes!
- Ask an equivalence query:
  - L(M) = L? → no, M for instance accepts aabbaa

	_
_	1
a	0
b	1
aa	1
ab	1

Active learning with L*

- C = aabbaa, -
- Where does M make a mistake?
  - Replace c prefixes with reached access strings
- Ask membership queries:
  - _bbaa in L? → no
  - _baa in L? → no
  - _aa in L? → yes

	_
_	1
a	0
b	1
aa	1
ab	1

Active learning with L*

- C = aabbaa, -
- Where does M make a mistake?
  - Replace c prefixes with reached access strings
- Ask membership queries:
  - _bbaa in L? → no
  - _baa in L? → no
  - _aa in L? → yes

	_
_	1
a	0
b	1
aa	1
ab	1

aa is a new distinguishing extension!

Active learning with L*

- Add the column aa
- Ask membership queries to complete the table

	_	aa
_	1	1
a	0	0
b	1	0
aa	1	1
ab	1	1

Active learning with L*

- Add the column aa
- Ask membership queries to complete the table
- b points to a new state
- Add its one-letter extensions, ask membership queries to complete

	_	aa
_	1	1
a	0	0
b	1	0
aa	1	1
ab	1	1
ba	1	1
bb	0	0

Active learning with L*

- Add the column aa
- Ask membership queries

The table defines a state machine, ask an equivalence query → yes!

Done

complete

	_	aa
_	1	1
a	0	0
b	1	0
aa	1	1
ab	1	1
ba	1	1
bb	0	0

Active learning with L*

1. Initialize the distinguishing table T
2. Complete T until it defines a state machine M
3. While(Eq(M) gives a counterexample c)
    1. Use Mem() and c to find a new distinguishing extension d
    2. Add d to T
    3. Complete T until it defines a state machine M

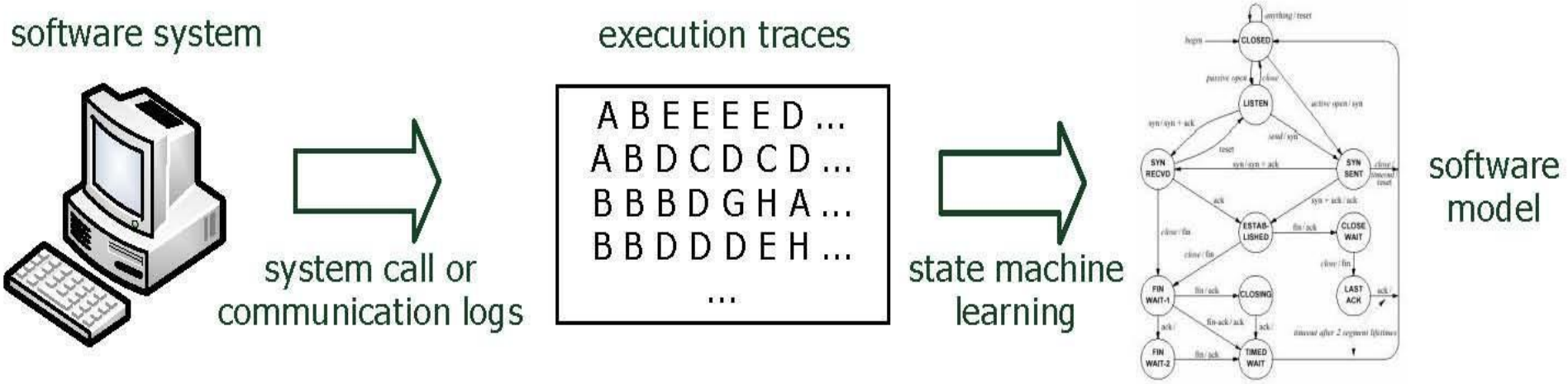
L* - some theory

- L* always finds a state machine for the target language
  - A distinguishing extension can always be found using the counterexample
  - Completing T terminates since we only have to add one-letter extensions once for every color
  - Once all states are identified, Eq(M) will return yes
- L* runs in polynomial time and from polynomial data!
  - Both $O(n^2)$, n number of states of smallest target

Active automaton learning

- Learnlib is an L* implementation that can be applied to software
  - Uses model-based testing methods for answering equivalence queries
  - We are combining this with fuzzing for security purposes
- The learned models allow to:
  - Investigate the model for any implementation flaws
  - Compare models from different implementations
  - …
- Demo…

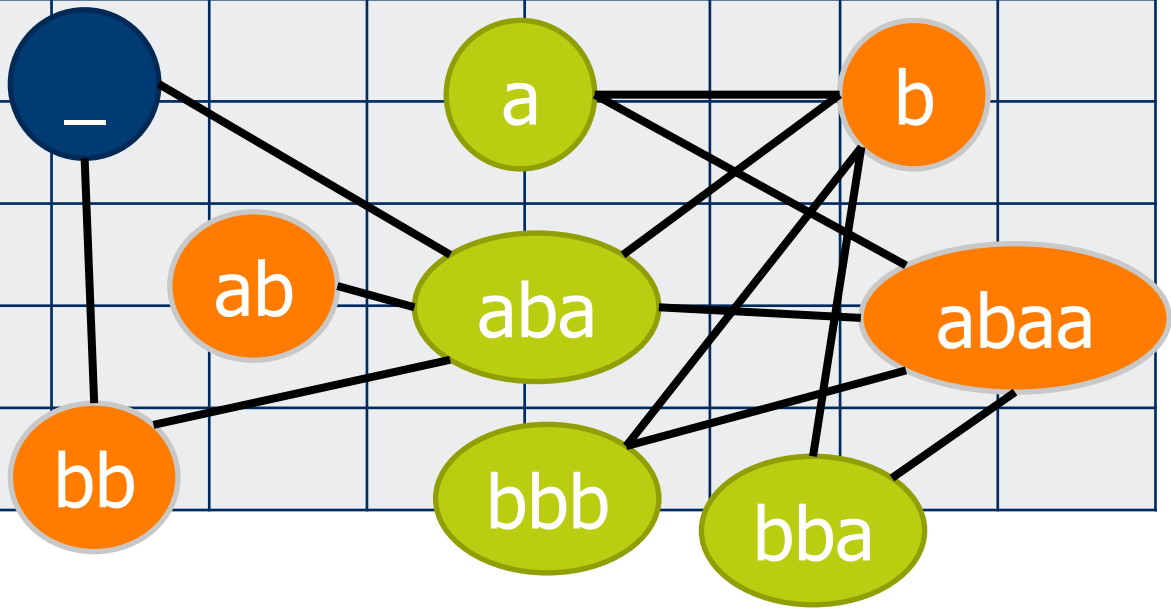
Use the information we have…



Passive learning

Key issue: coloring leads to inconsistencies

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0			1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										

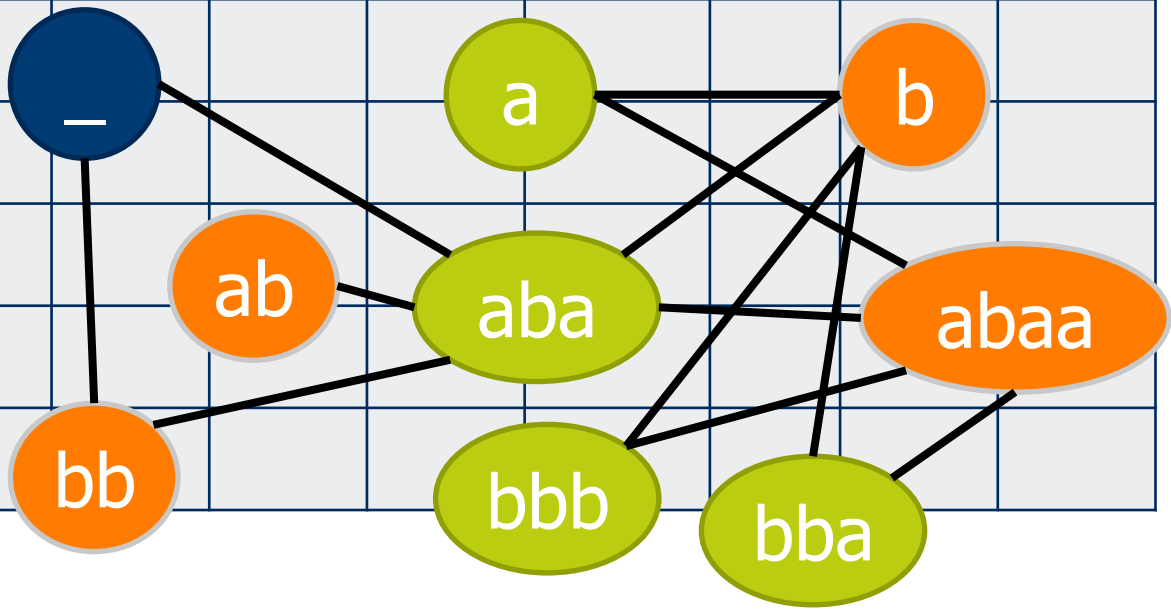


TUDelft

Key issue: coloring leads to inconsistencies

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
									1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										

Oh, and it is NP-hard…



TUDelft

Formalization

- Graph/row coloring works when the Hankel matrix is infinite (proof via Myhill-Nerode)
  - otherwise it gives a lower bound

- Missing:

  - *row i and j have the same color implies:*

    1. *if i has a 0 in column x, then j has a 0 in column x*
    2. *if i has a 1 in column x, then j has a 1 in column x*

    3. ***for all a, states reached from i and j by symbol a also have the same color!***

Formalization: exact solutions

1. Encode everything in Satisfiability

   DFASAT (Heule and Verwer 2010)

2. Encode the Hankel matrix with suffixes of size up to k

   kTails (Biermann and Feldman 1972)

3. Search all colorings using iterative deepening

   exbar (Lang 1999)

4. Other (inexact) search strategies
   - Beam search

     (Bugalho and Oliveira 2005)
   - Evolutionary algorithms

     (Lucas and Reynolds 2003)
   - Ant colony optimization

     (Buzhinsky et al. 2014)

Formalization: approximations

1. State merging, color iteratively, smaller prefix first

   RPNI (Oncina and Garcia1994)

2. State merging, largest evidence first, red-blue fringe

   EDSM (Lang 1998)

3. State merging, ignore negative traces

   DFASAT (Heule and Verwer 1999)

4. State merging, flexible evidence functions

   Flexfringe (Verwer and Hammerschmidt 2017)

   also contains DFASAT translation and search wrappers

Approximation: state merging

- Very effective greedy method
  - Initially assume a large DFA
  - Iteratively combine similar states
  - Until no more state can be combined
- Guarantees to find the target state machine in the limit
  - given a polynomial amount of data in the size of the target machine
  - using run-time polynomial in the input data size

color iteratively, copy values, force determinism!

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0			1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										

color iteratively, copy values, force determinism!

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0			1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										

color iteratively, copy values, force determinism!

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0			1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										

can a be merged with _?

color iteratively, copy values, force determinism!

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0			1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										

if a and _ are the same,
ab and b are the same

color iteratively, copy values, force determinism!

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0			1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										

a and _ can be merged,
but there is 0 overlap in values

evidence suggests not to merge
(RPNI would merge EDSM not)

color iteratively, copy values, force determinism!

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0			1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										

can b be merged with _?

color iteratively, copy values, force determinism!

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0			1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										

no!

color iteratively, copy values, force determinism!

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0			1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										

color iteratively, copy values, force determinism!

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0			1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										

there is evidence suggesting ab and _ to be the same

color iteratively, copy values, force determinism!

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0			1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										

even more in determinization

color iteratively, copy values, force determinism!

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1	0		0		1	1
a	1	0		1			0				
b	0			1					1		
ab		1	0		1	0		0		1	1
aba	1	0		1			0				
abaa	0										
bb		1	1								
bbb	1										
bba	1										

merge and copy values!

color iteratively, copy values, force determinism!

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1	0		0		1	1
a	1	0		1			0				
b	0			1					1		
ab		1	0		1	0		0		1	1
aba	1	0		1			0				
abaa	0			1					1		
bb		1	1								
bbb	1										
bba	1										

continue merging
maximizing evidence

TU Delft

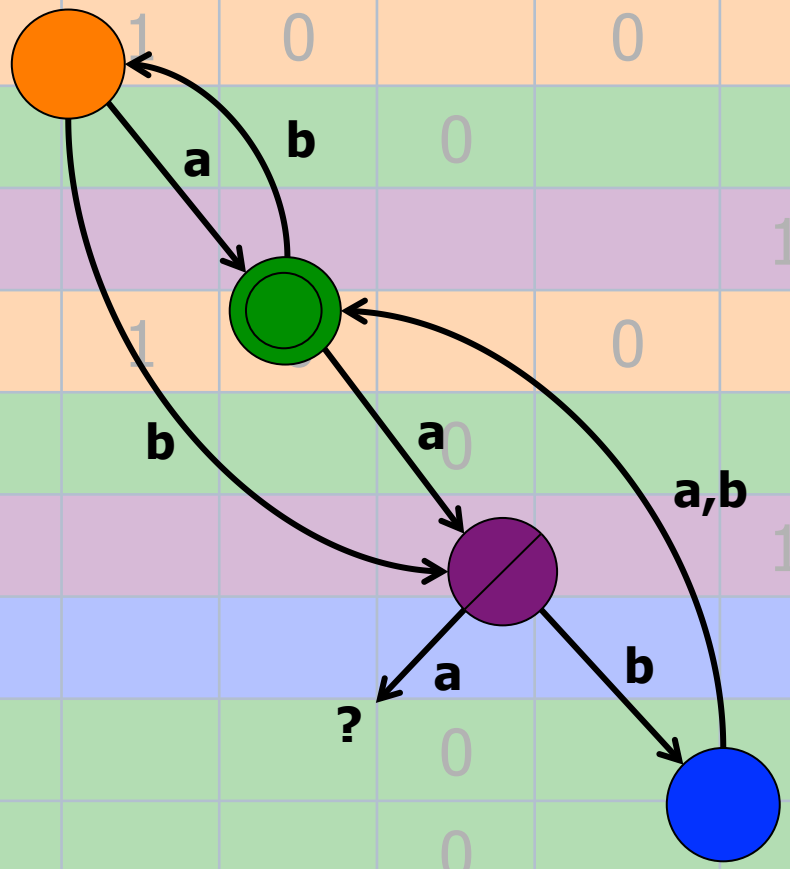
color iteratively, copy values, force determinism!

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1	0		0		1	1
a	1	0		1			0				
b	0			1					1		
ab		1	0		1	0		0		1	1
aba	1	0		1			0				
abaa	0			1					1		
bb		1	1								
bbb	1	0		1			0				
bba	1	0		1							

continue merging
maximizing evidence

color iteratively, copy values, force determinism!

	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1	0		0		1	1
a	1	0		1			0				
b	0			1					1		
ab		1	0		1			0		1	1
aba	1	0		1			0				
abaa	0			1					1		
bb		1	1								
bbb	1	0		1			0				
bba	1	0		1			0				



obtaining a state machine

Red-blue fringe state merging

- *To improve speed, use trees!*

1. Start from a prefix tree acceptor
2. Color the initial state red, and it's children blue
3. For all consistent red-blue state pairs R-B
    1. Merge B with R
    2. Compute evidence score
    3. Undo merge
4. If there is a high scoring and consistent R-B merge
    1. Merge B with R
5. Else
    1. Color B with most occurrences (or most shallow first, ..) red
6. Color all children of red states blue
7. Goto 3

Red-blue fringe state merging

- *To improve speed, use trees!*

1. Start from a prefix tree acceptor
2. Color the initial state red, and it's children blue
3. For all consistent red-blue state pairs R-B
    1. Merge B with R
    2. Compute evidence score
    3. Undo merge
4. If there is a high scoring and consistent R-B merge
    1. Merge B with R
5. Else
    1. Color B with most occurrences (or most shallow first, ..) red
6. Color all children of
7. Goto 3

efficient active learning algorithms use trees as well

or… solve the NP-hard problem by SAT

- Generate the prefix-tree T
- Initialize the set of colors L
- Construct a SAT formula F using T and L
- Solve F using a SAT solver
- If F is unsatisfiable add a color to L, goto 3
- Return the DFA found in step 4

Satisfiability encoding

Variables	Range	Meaning
$x_{v,i}$	$v \in V; i \in C$	$x_{v,i} \equiv 1$ iff state v has color i
$y_{a,i,j}$	$a \in \Sigma; i,j \in C$	$y_{a,i,j} \equiv 1$ iff parents of states with color j and incoming label a have color i
z_i	$i \in C$	$z_i \equiv 1$ iff an accepting state has color i

Clauses	Range	Meaning
$(x_{v,1} \vee x_{v,2} \vee \cdots \vee x_{v,k})$	$v \in V$	every state has at least one color
$(\neg x_{v,i} \vee z_i) \wedge (\neg x_{w,i} \vee \neg z_i)$	$v \in V_+; w \in V_-; i \in C$	accepting states cannot have the same color as rejecting states
$(y_{l(v),i,j} \vee \neg x_{p(v),i} \vee \neg x_{v,j})$	$v \in V; i,j \in C$	a parent relation is set when a state and its parent are colored
$(\neg y_{a,i,h} \vee \neg y_{a,i,j})$	$a \in \Sigma; h,i,j \in C; h < j$	each parent relation can target at most one color

Redundant Clauses	Range	Meaning
$(\neg x_{v,i} \vee \neg x_{v,j})$	$v \in V; i,j \in C; i < j$	every state has at most one color
$(y_{a,i,1} \vee y_{a,i,2} \vee \cdots \vee y_{a,i,k})$	$a \in \Sigma; i \in C$	each parent relation must target at least one color
$(\neg y_{l(v),i,j} \vee \neg x_{p(v),i} \vee x_{v,j})$	$v \in V; i,j \in C$	a parent relation forces a state once the parent is colored
$(\neg x_{v,i} \vee \neg x_{w,i})$	$i \in C; (v,w) \in E$	all determinization conflicts explicitly added as clauses

WHEN DATA IS UNLABELED

Hankel Matrix

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	1	0	1	1	1	1	0	0	1	0
a	0	1	1	0	1	0	1	1	1	1
b	1	1	0	0	1	1	1	1	1	1
aa	1	0	1	1	1	1	0	0	1	0
ab	1	0	1	1	1	1	0	0	1	0
ba	1	0	1	1	1	1	0	0	1	0
bb	0	1	1	0	1	0	1	0	1	1
aaa	0	1	1	0	1	0	1	1	1	1
aab	1	1	0	0	1	1	1	1	1	1
aba	0	1	1	0	1	0	1	1	1	1

Probabilistic/Weighted Hankel Matrix

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0.46	0	0.08	0.08	0.03	0.07	0	0	0.01	0
a	0	0.08	0.03	0	0.01	0	0	0.01	0	0.01
b	0.08	0.07	0	0	0.01	0.03	0.01	0.01	0	0.01
aa	0.08	0	0.01	0.01	0	0.01	0	0	0	0
ab	0.03	0	0	0	0	0	0	0	0	0
ba	0.07	0	0.01	0.01	0	0.01	0	0	0	0
bb	0	0.03	0.01	0	0	0	0	0	0	0
aaa	0	0.01	0	0	0	0	0	0	0	0
aab	0.01	0.01	0	0	0	0	0	0	0	0
aba	0	0	0	0	0	0	0	0	0	0

cells contains the probability of observing prefix.suffix

Replacing Myhill-Nerode by Markov

- Given an unlabeled dataset
  - Put the sample probabilities in a Hankel matrix
  - Determine the minimal coloring of the matrix
- Two rows r, r' can have the same color if
  - r = c * r', c a constant
  - i.e., *if the suffix probability is independent of the prefix!*
- Use, i.e.:
  - State merging

    ALERGIA (Carrasco and Oncina 1994)
  - Using likelihood ratio test

    RTI (Verwer et al. 2010)

    both implemented in Flexfringe (Verwer 2017)
  - Singular value decomposition

    Spectral Learning (Hsu 2009)

Probabilistic/Weighted Hankel Matrix

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0.46	0	0.08	0.08	0.03	0.07	0	0	0.01	0
a	0	0.08	0.03	0	0.01	0	0	0.01	0	0.01
b	0.08	0.07	0	0	0.01	0.03	0.01	0.01	0	0.01
aa	0.08	0	0.01	0.01	0	0.01	0	0	0	0
ab	0.03	0	0	0	0	0	0	0	0	0
ba	0.07	0	0.01	0.01	0	0.01	0	0	0	0
bb	0	0.03	0.01	0	0	0	0	0	0	0
aaa	0	0.01	0	0	0	0	0	0	0	0
aab	0.01	0.01	0	0	0	0	0	0	0	0
aba	0	0	0	0	0	0	0	0	0	0

Probabilistic/Weighted Hankel Matrix

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0.46	0	0.08	0.08	0.03	0.07	0	0	0.01	0
a	0	0.08	0.03	0	0.01	0	0	0.01	0	0.01
b	0.08	0.07	0	0	0.01	0.03	0.01	0.01	0	0.01
aa	0.08	0	0.01	0.01	0	0.01	0	0	0	0
ab	0.03	0	0	0	0	0	0	0	0	0
ba	0.07	0	0.01	0.01	0	0.01	0	0	0	0
bb	0	0.03	0.01	0	0	0	0	0	0	0
aaa	0	0.01	0	0						
aab	0.01	0.01	0	0						
aba	0	0	0	0						

The Hankel matrix is fully specified given the input data!

This is used by spectral learning

Spectral learning: find the rank, color the matrix

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0.46	0	0.08	0.08	0.03	0.07	0	0	0.01	0
a	0	0.08	0.03	0	0.01	0	0	0.01	0	0.01
b	0.08	0.07	0	0	0.01	0.03	0.01	0.01	0	0.01
aa	0.08	0	0.01	0.01	0	0.01	0	0	0	0
ab	0.03	0	0	0	0	0	0	0	0	0
ba	0.07	0	0.01	0.01	0	0.01	0	0	0	0
bb	0	0.03	0.01	0	0	0	0	0	0	0
aaa	0	0.01	0	0	0	0	0	0	0	0
aab	0.01	0.01	0	0	0	0	0	0	0	0
aba	0	0	0	0	0	0	0	0	0	0

Spectral learning: find the rank, color the matrix

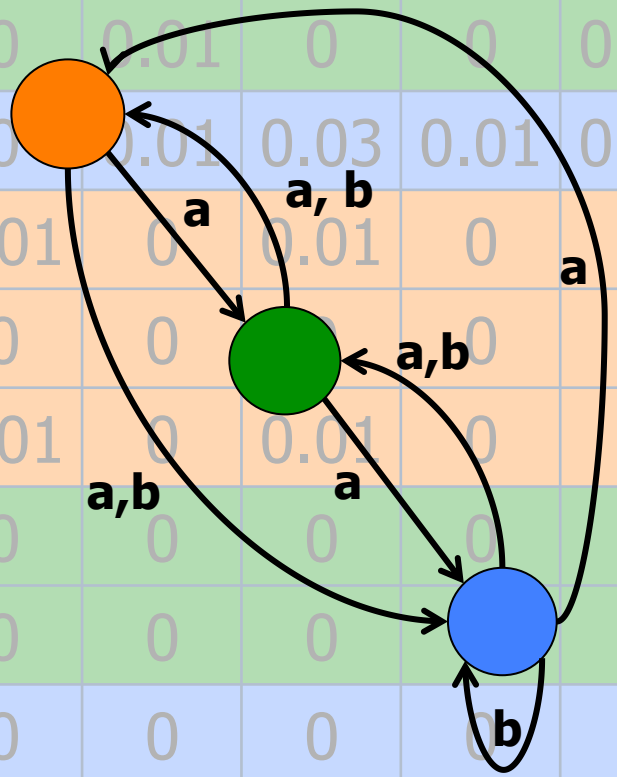
	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0.46	0	0.08	0.08	0.03	0.07	0	0	0.01	0
a	0	0.08	0.03	0	0.01	0	0	0.01	0	0.01
b	0.08	0.07	0	0	0.01	0.03	0.01	0.01	0	0.01
aa	0.08	0	0.01	0.01	0	0.01	0	0	0	0
ab	0.03	0	0	0	0	0	0	0	0	0
ba	0.07	0	0.01	0.01	0	0.01	0	0	0	0
bb	0	0.03	0.01	0	0	0	0	0	0	0
aaa	0	0.01	0	0	0	0	0	0	0	0
aab	0.01	0.01	0	0	0					
aba	0	0	0	0	0					

this algorithm is polynomial time!

TU Delft

and it gives a state machine

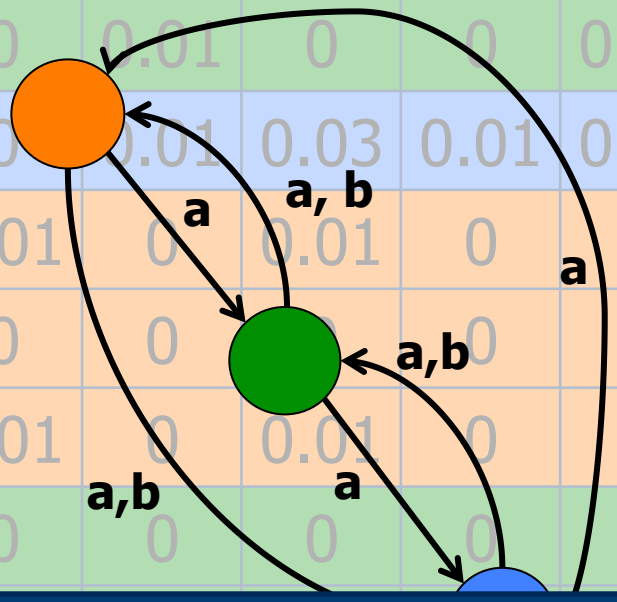
	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0.46	0	0.08	0.08	0.03	0.07	0	0	0.01	0
a	0	0.08	0.03	0	0.01	0	0	0.01	0	0.01
b	0.08	0.07	0	0	0.01	0.03	0.01	0.01	0	0.01
aa	0.08	0	0.01	0.01	0	0.01	0	0	0	0
ab	0.03	0	0	0	0	0	0	0	0	0
ba	0.07	0	0.01	0.01	0	0.01	0	0	0	0
bb	0	0.03	0.01	0	0	0	0	0	0	0
aaa	0	0.01	0	0	0	0	0	0	0	0
aab	0.01	0.01	0	0	0	0	0	0	0	0
aba	0	0	0	0	0	0	0	0	0	0



TUDelft

and it gives a state machine

	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	0.46	0	0.08	0.08	0.03	0.07	0	0	0.01	0
a	0	0.08	0.03	0	0.01	0	0	0.01	0	0.01
b	0.08	0.07	0	0	0.01	0.03	0.01	0.01	0	0.01
aa	0.08	0	0.01	0.01	0	0.01	0	0	0	0
ab	0.03	0	0	0	0	0	0	0	0	0
ba	0.07	0	0.01	0.01	0	0.01	0	0	0	0
bb	0	0.03	0.01	0	0	0	0	0	0	0
a										
a										
a										



Model is non-deterministic and typically not very insightful but gives accurate predictions…

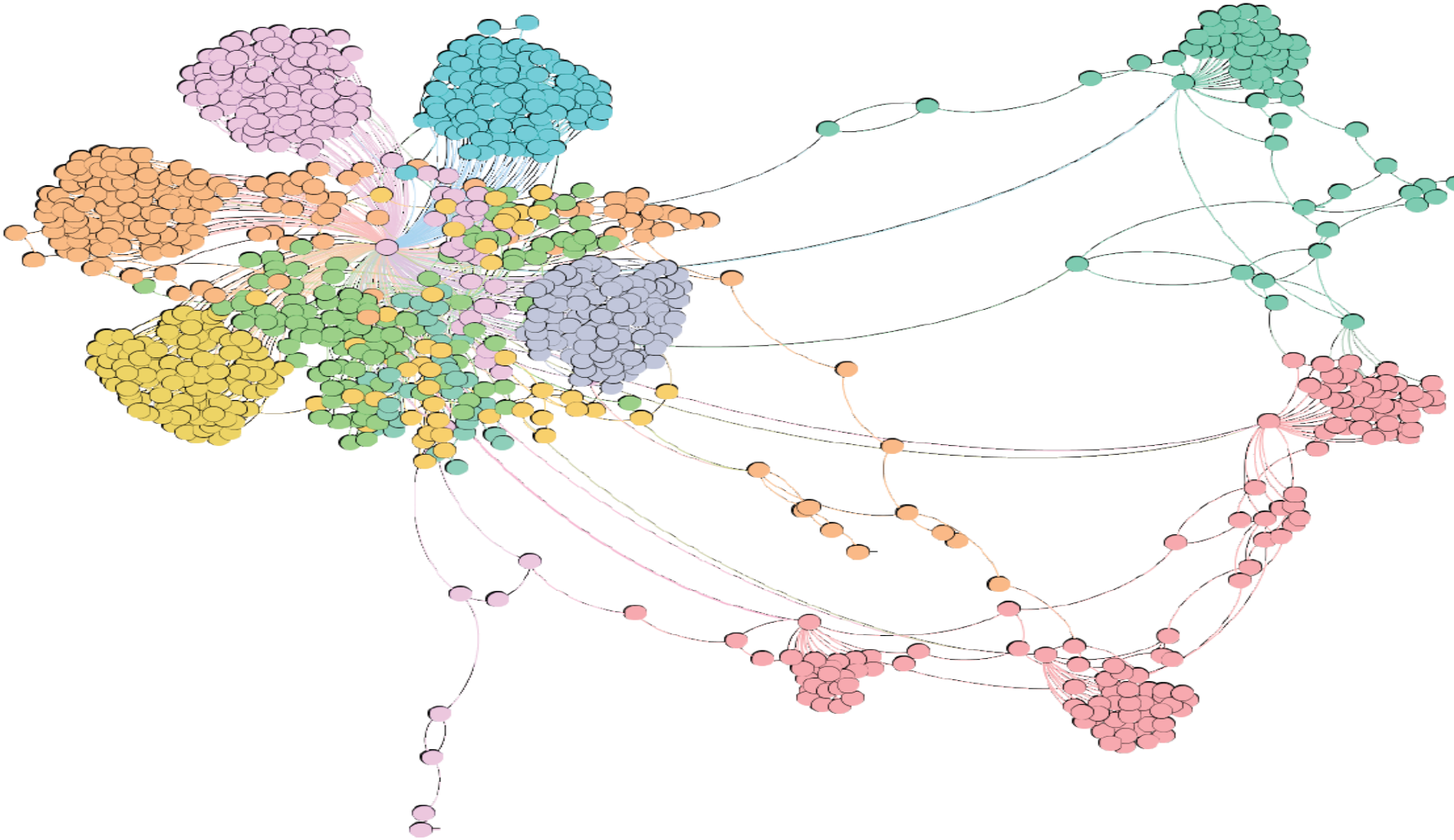
Running ALERGIA typically gives a larger but deterministic model

Replacing Myhill-Nerode by Markov

- Two rows r, r' can have the same color if
  - r = c * r', c a constant
  - i.e., *if the suffix probability is independent of the prefix!*
- Given an unlabeled dataset
  - Put the sample probabilities in a Hankel matrix
  - Determine the minimal coloring of the matrix
- Using, i.e.:
  - State merging (ALERGIA)
  - Singular value decomposition (spectral learning)
  - satisfiability or graph coloring
  - …

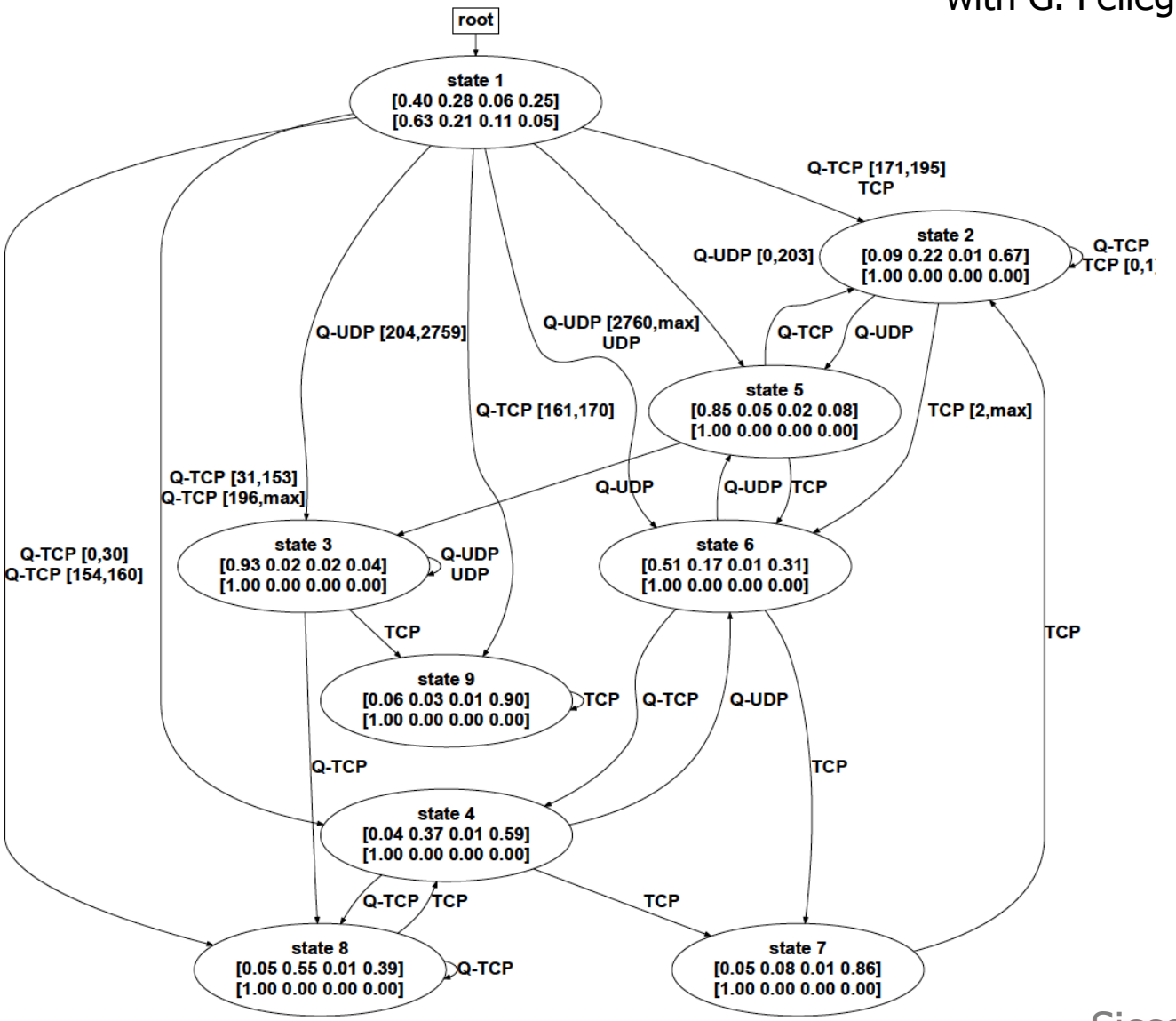
Use Case: Reactive Systems

with M. Janssen



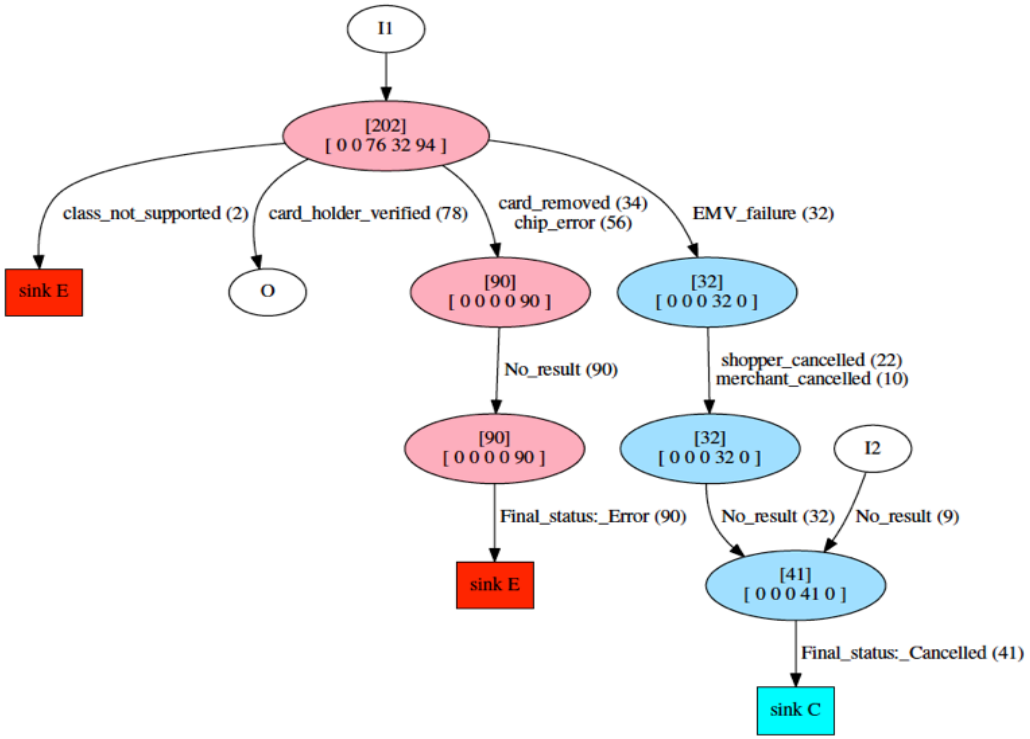
A model for malware

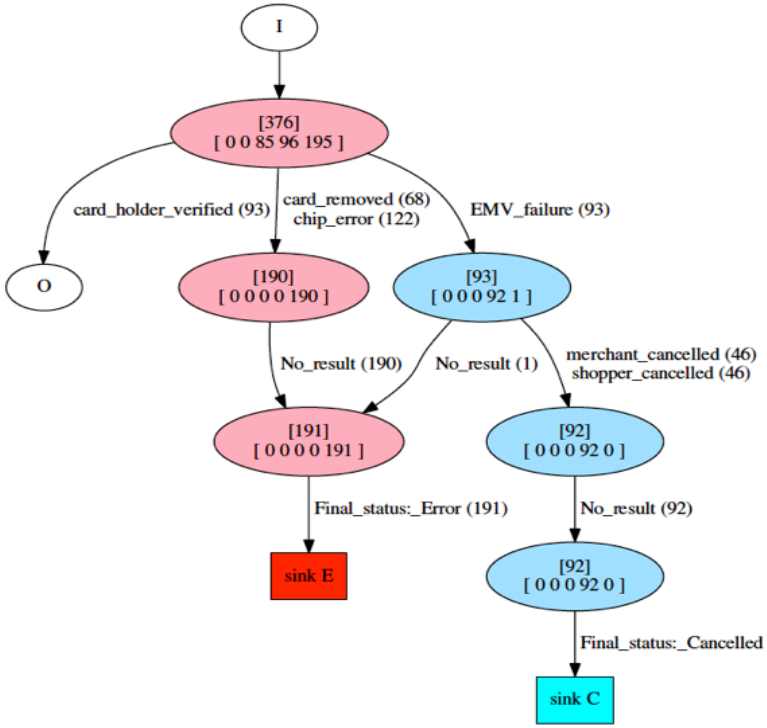
with G. Pellegrino



Automated Testing of Payment Devices

with R. Wieman,
C. Hammerschmi





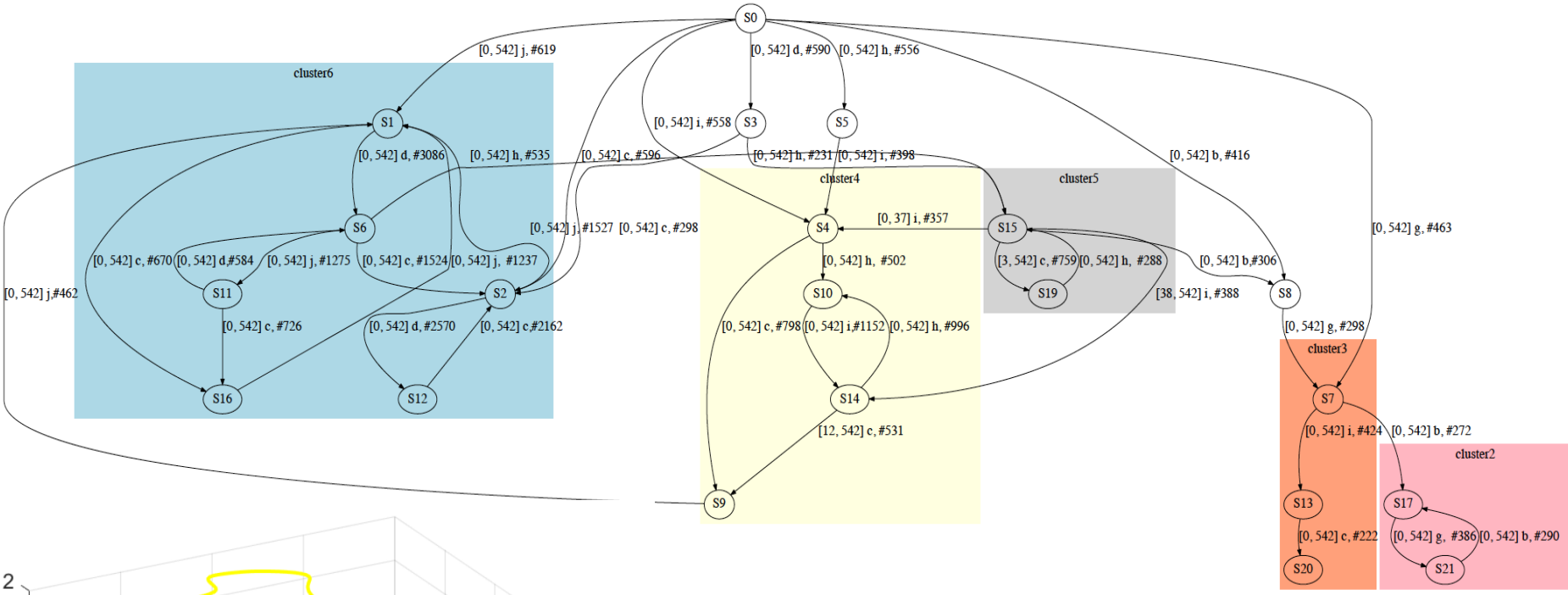
TU Delft

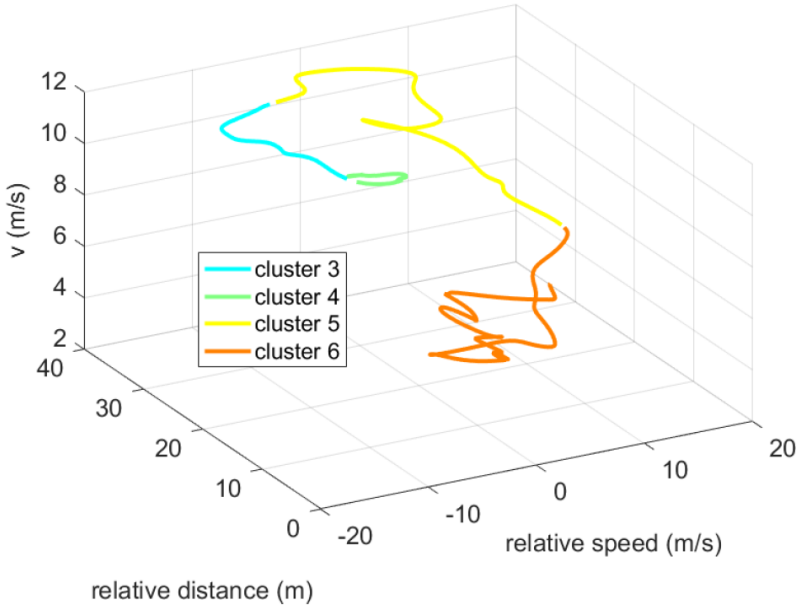
adyen

Sicco Verwer

Automated driving

with Q. Lin



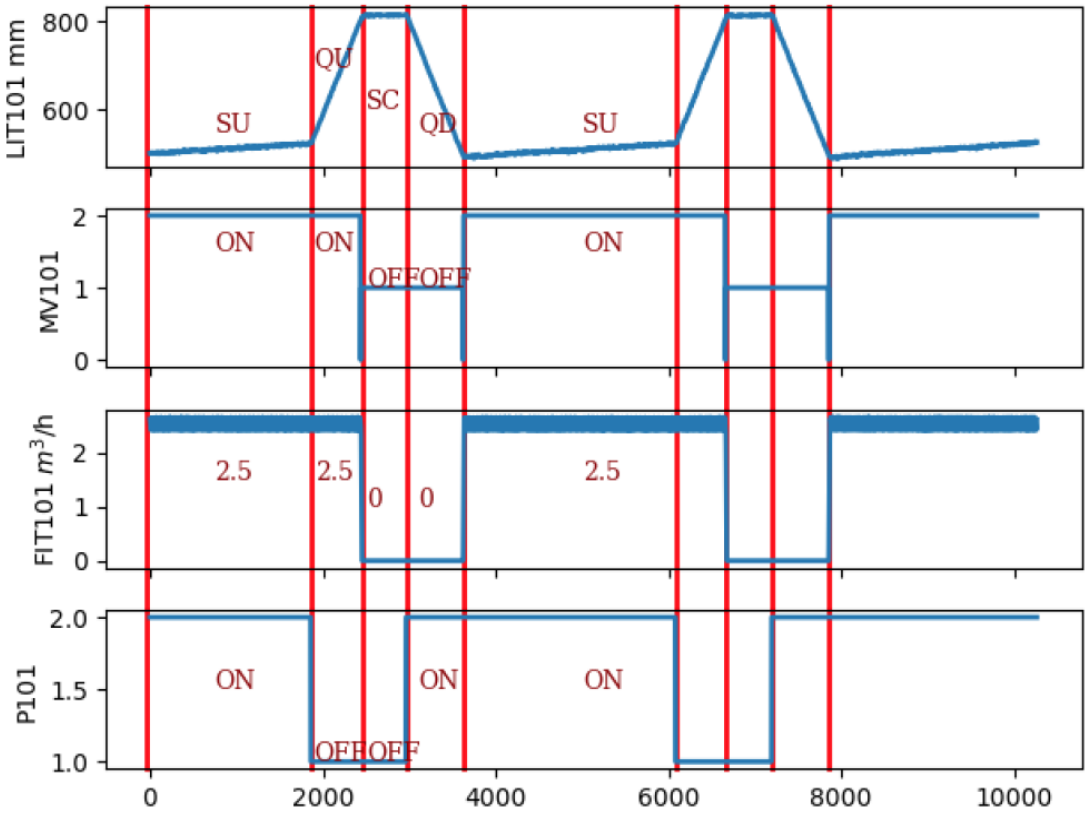


- Learning behavioral models
- Learning a human-like cruise-control (finite-state controller)

Sicco Verwer

Industrial Control System Security

with Q. Lin



- Sensor and actuator data
- Highly discrete and predictable

TU Delft

Sicco Verwer

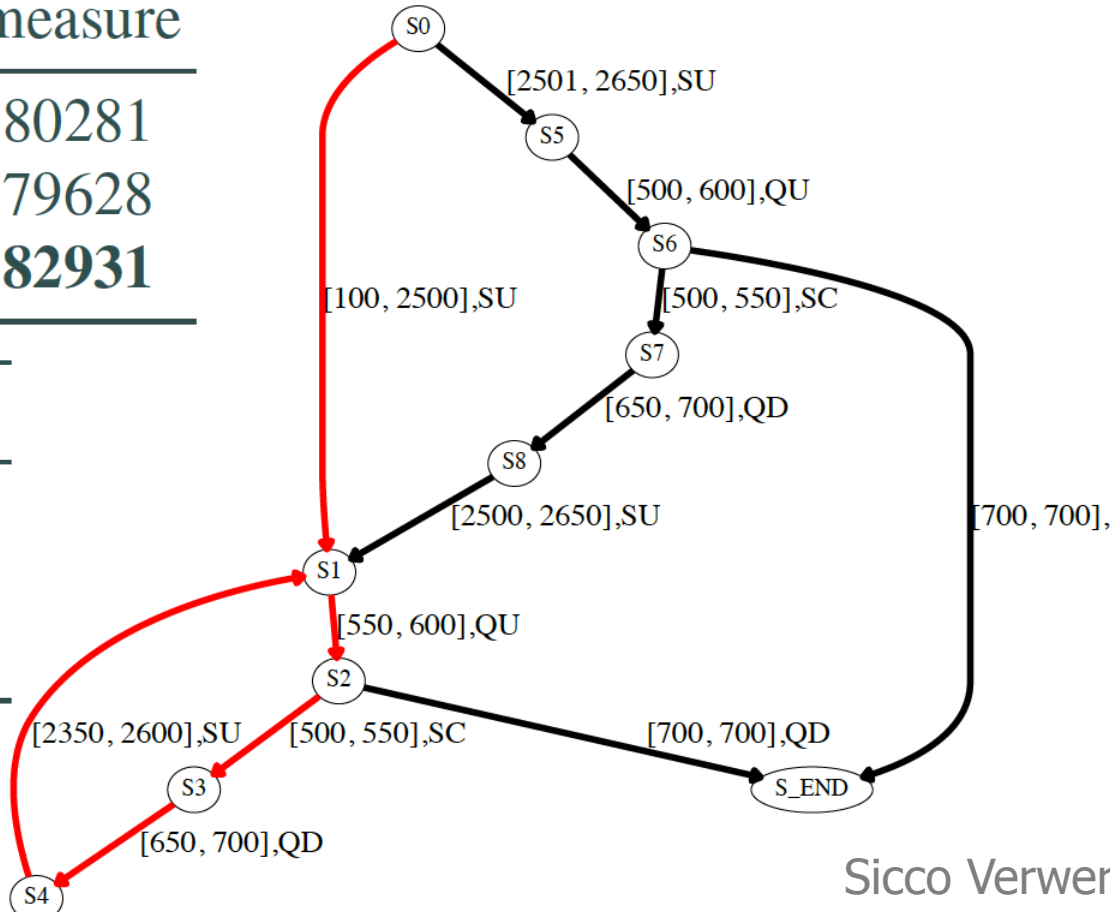
ICS Intrusion detection results

with Q. Lin

- 40 attacks on SWaT system (Singapore SUTD)
- We learn a simple, fast, and accurate model

Method	Precision	Recall	F measure
DNN	**0.98295**	0.67847	0.80281
SVM	0.92500	0.69901	0.79628
TABOR	0.84702	**0.81233**	**0.82931**

Model Number	Training	Testing
DNN	2 weeks	8 hours
SVM	30 min	10 min
TABOR	**214 s**	**33** s



TUDelft

Sicco Verwer