# In Practice

- https://www.microsoft.com/en-us/springfield/
- http://www.economist.com/news/science-and-technology/21647269-automating-search-loopholes-software-hacking-hackers
- https://www.wired.com/2016/08/security-bots-show-hacking-isnt-just-humans/
- http://www.digitaltrends.com/computing/darpa-mayhem-bot/
- …

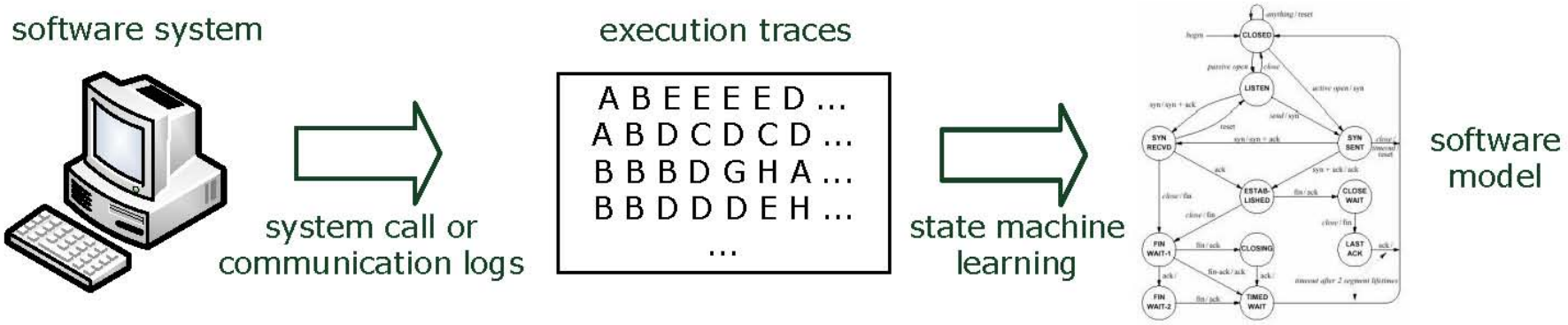
# Teminology

- Black-box (fuzz/random) testing:
  - Not interpreting the software, using it as a “black-box”
  - Boundary value analysis
  - Brute-force
- White-box (fuzz) testing:
  - Symbolic Execution
  - Unit Testing
- Grey-box testing:
  - Some parts are black-box, other white-box
  - Concolic, DART (?)
  - Regression testing

# Today: State Machine Learning

Myhill-Nerode
language Hankel matrix
graph coloring
active and passive learning

# 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

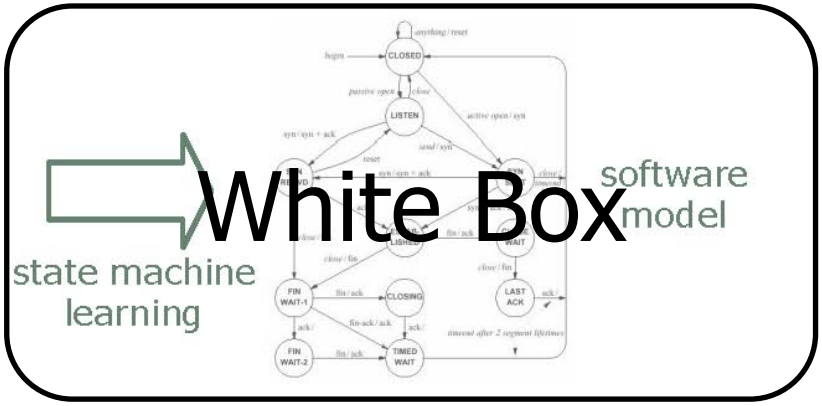
execution traces

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 ...
...

state machine learning

White Box

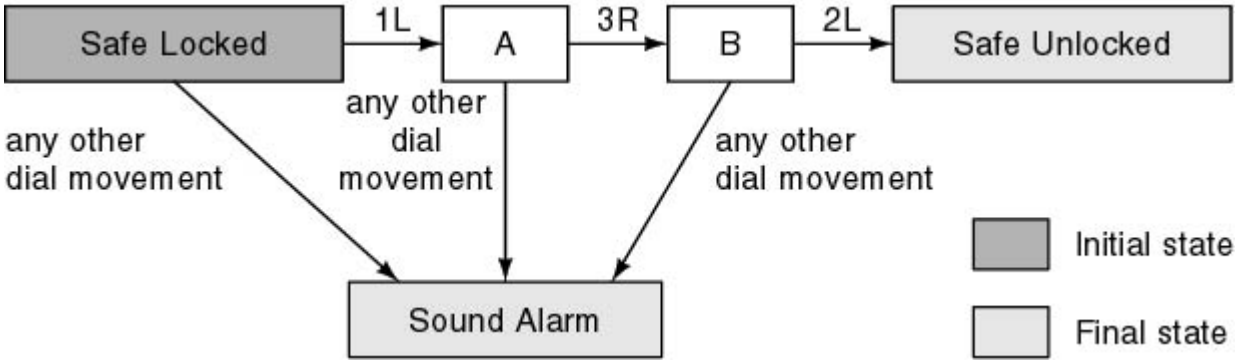
software model



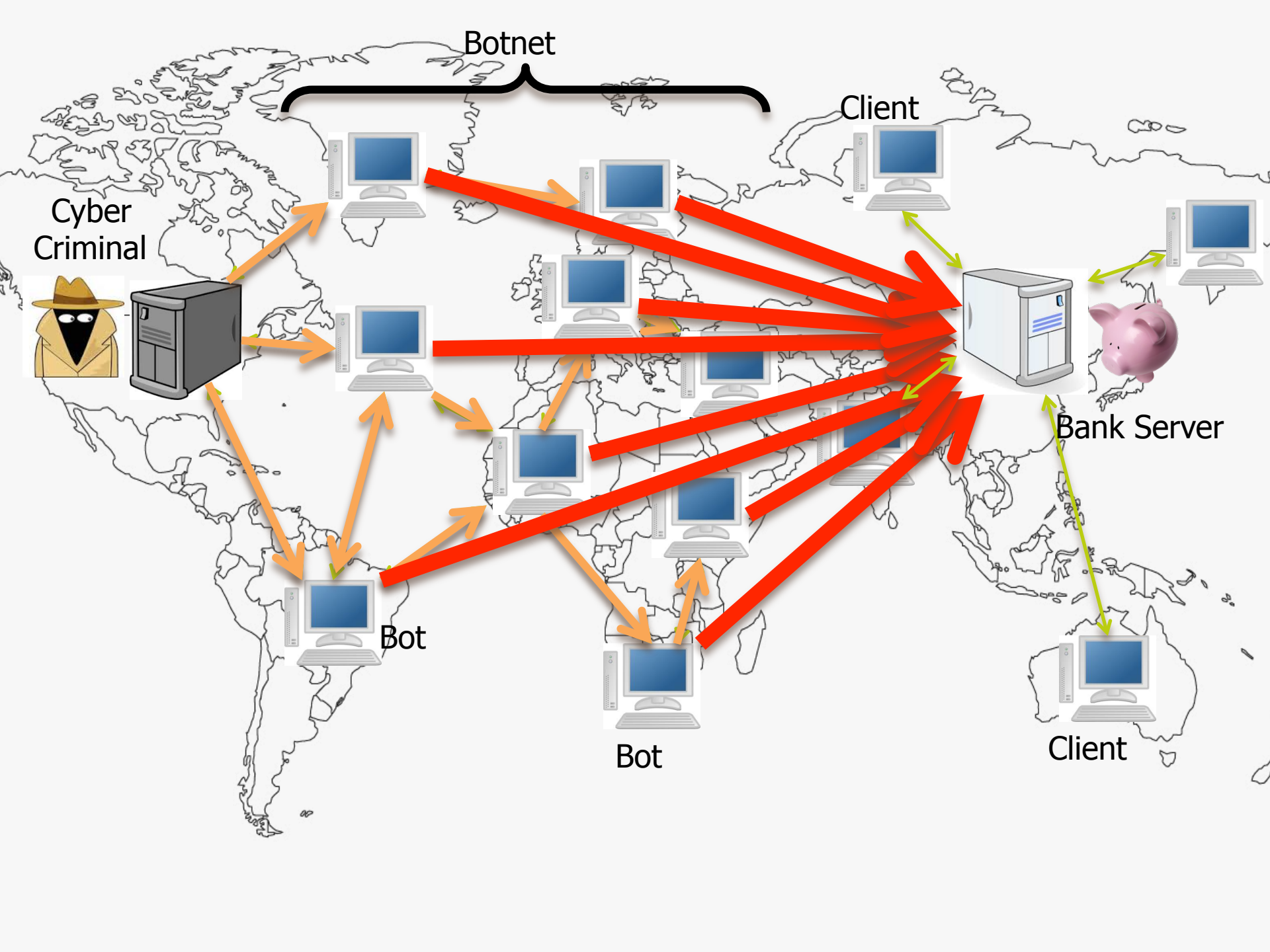

TU Delft

# State machines

- State machines are key models for the design and analysis of software systems
- They model the states of a system and the transitions between them:

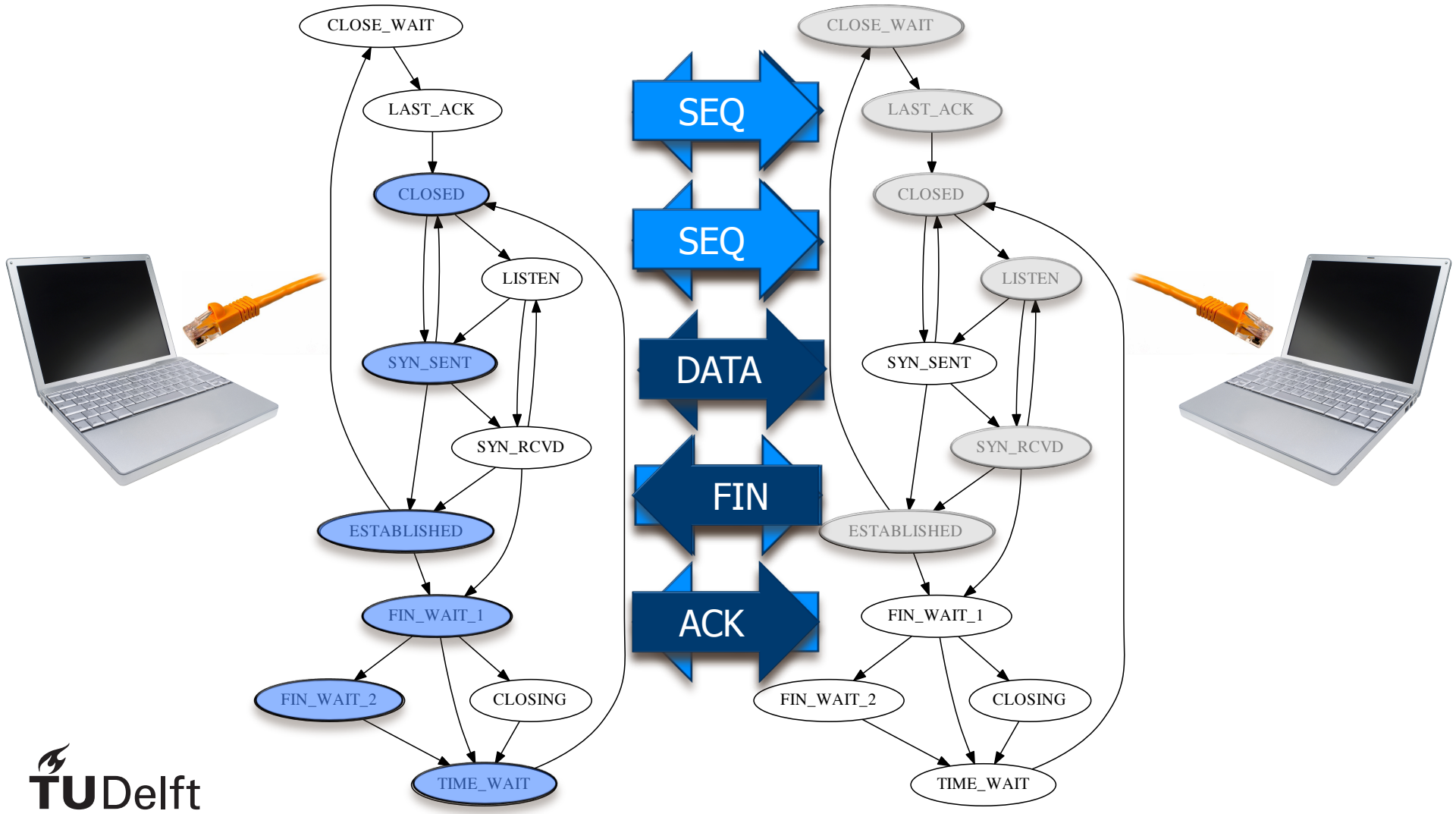



- Used for visualization, verification, testing, control, …


Botnet
Client
Cyber Criminal
Bank Server
Bot
Bot
Client

# We observe internet traffic

- SYN, SYN-ACK, ACK, SEQ, ACK, SEQ, ACK, SEQ, ACK, FIN-ACK, ACK, FIN-ACK
- SYN, SYN-ACK, ACK, FIN-ACK, ACK, FIN-ACK
- SYN, SYN-ACK, ACK, SEQ, ACK, FIN-ACK, ACK, FIN-ACK
- …..


CLOSE_WAIT
LAST_ACK
CLOSED
LISTEN
SYN_SENT
SYN_RCVD
ESTABLISHED
FIN_WAIT_1
FIN_WAIT_2
CLOSING
TIME_WAIT
SEQ
SEQ
DATA
FIN
ACK
CLOSE_WAIT
LAST_ACK
CLOSED
LISTEN
SYN_SENT
SYN_RCVD
ESTABLISHED
FIN_WAIT_1
FIN_WAIT_2
CLOSING
TIME_WAIT

TUDelft

# Learning (reverse-engineering)

- Execution logs form a big data set from which can be used to gain information about a system or protocol

This can help to

- analyze your own code and hunt for bugs, or
- reverse-engineer someone else's unknown protocol, eg. a botnet, to fingerprint or to analyze (and attack)

# We observe internet traffic

- SYN, SYN-ACK, ACK, SEQ, ACK, SEQ, ACK, SEQ, ACK, FIN-ACK, ACK, FIN-ACK
- SYN, SYN-ACK, ACK, FIN-ACK, ACK, FIN-ACK
- SYN, SYN-ACK, ACK, SEQ, ACK, FIN-ACK, ACK, FIN-ACK
- …..

# First: make abstract

- A, B, C, D, C, D, C, D, C, E, C, E
- A, B, C, E, C, E
- A, B, C, D, C, E, C, E
- …..

# Then: look for patterns!

- A, B, C, D, C, D, C, D, C, E, C, E
- A, B, C, E, C, E
- A, B, C, D, C, E, C, E
- …..

# Then: look for patterns!

- A, B, C, D, C, D, C, D, C, E, C, E
- A, B, C, E, C, E
- A, B, C, D, C, E, C, E
- …..

# Then: look for patterns!

- A, B, C, D, C, D, C, D, C, E, C, E
- A, B, C, E, C, E
- A, B, C, D, C, E, C, E
- …..

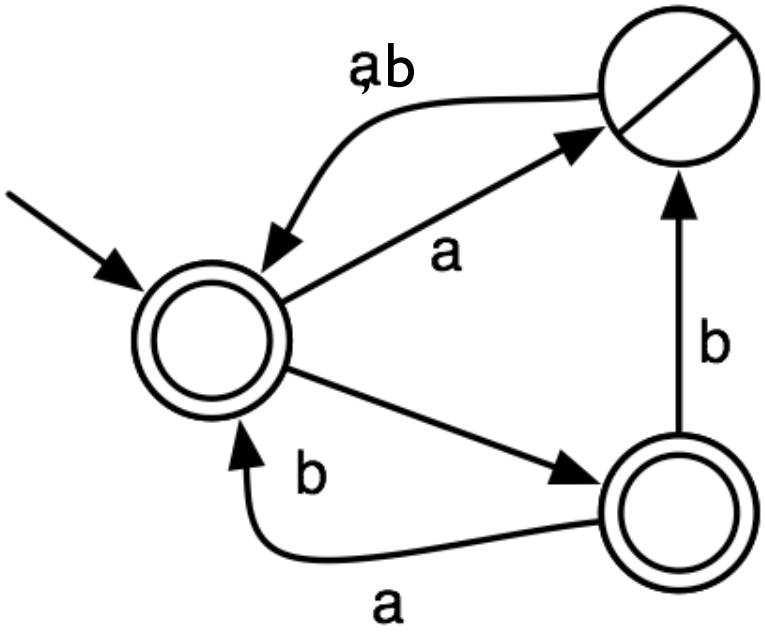
# Then: look for patterns!

- A, B, C, D, C, D, C, D, C, E, C, E
- A, B, C, E, C, E
- A, B, C, D, C, E, C, E
- …..

# Then: look for patterns!

- A, B, C, D, C, D, C, D, C, E, C, E
- A, B, C, E, C, E
- A, B, C, D, C, E, C, E
- …..

# Deterministic finite state automata (DFA)




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

# State machine learning

- Input:
  - Data, e.g., (a,-),(ab,+),(aa,+),(b,+),(bb,-),(bab,+)
- Goal: Find a smallest DFA that is consistent with the data
- Size is measured by the number of states/transitions
  - Occam's razor
- Consistency is:
  - Accepts all positive, rejects all negative

# Computational complexity

- NP
  - Class of problems verifiable in polynomial time
- NP-hard
  - At least as difficult to solve as any other problem in NP
  - Solving in polynomial time is widely believed to be impossible (even with quantum computers)
- NP-complete
  - NP-hard and in NP, e.g., satisfiability, traveling salesman
- Undecidable
  - Impossible to solve using a computer, equivalent to the halting problem

# 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

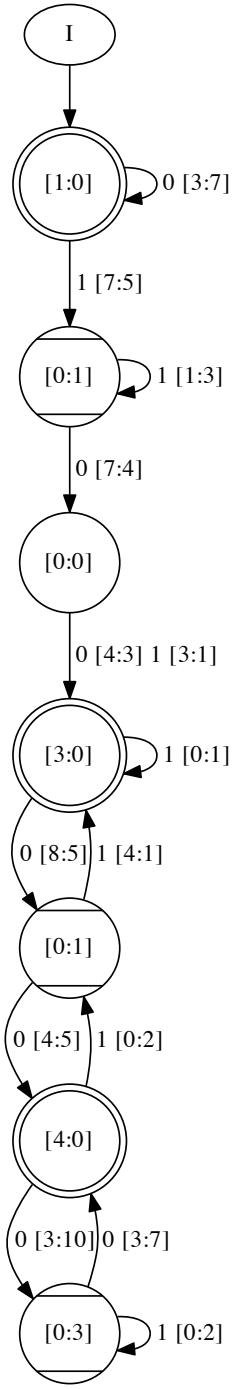
What is the smallest state machine that accepts the top and rejects the bottom strings?

**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




TU Delft

# 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:

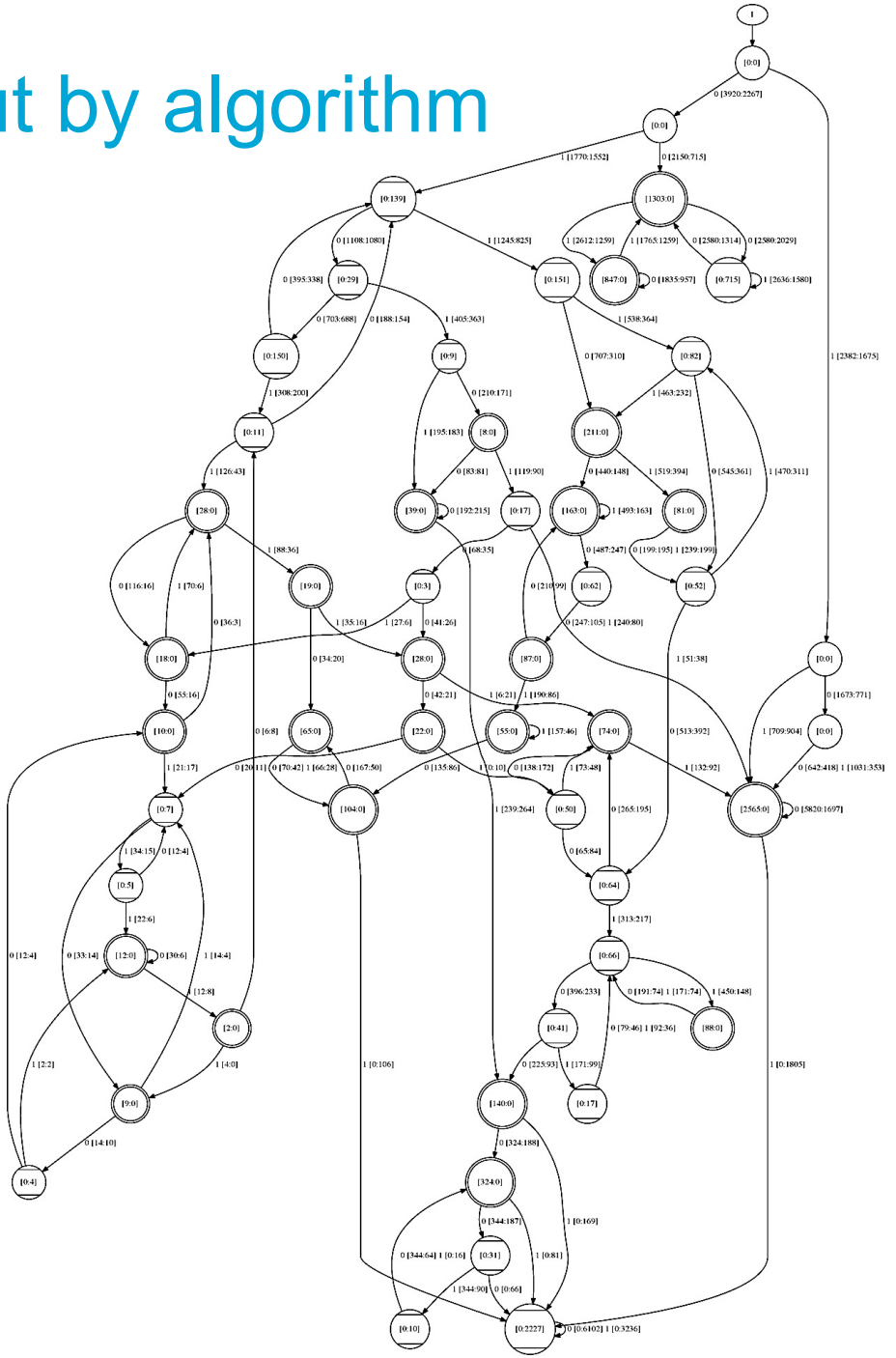
0 1 0 0 0

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

0 0 0 0 0 1

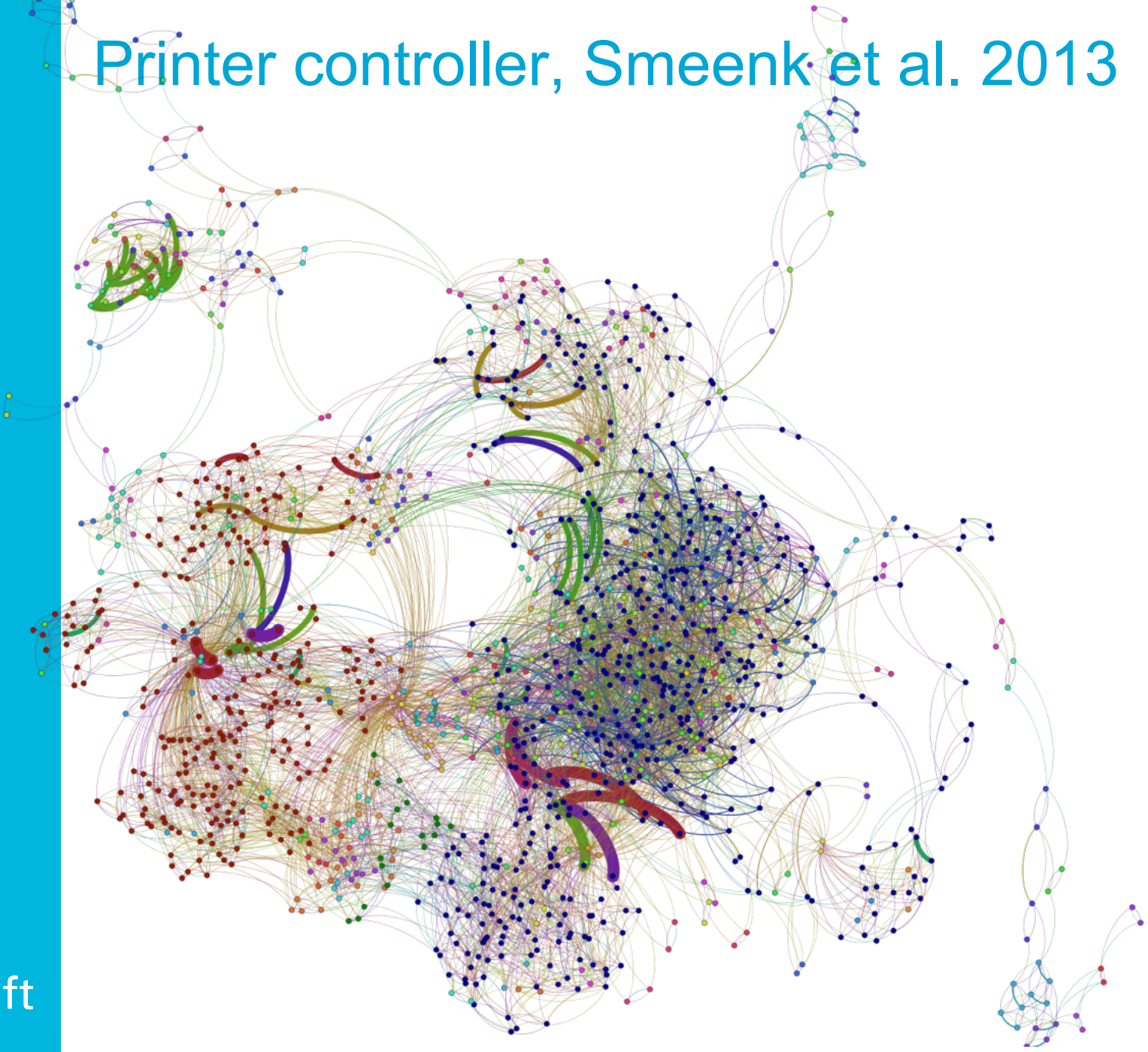
1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 1 0 0 0

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



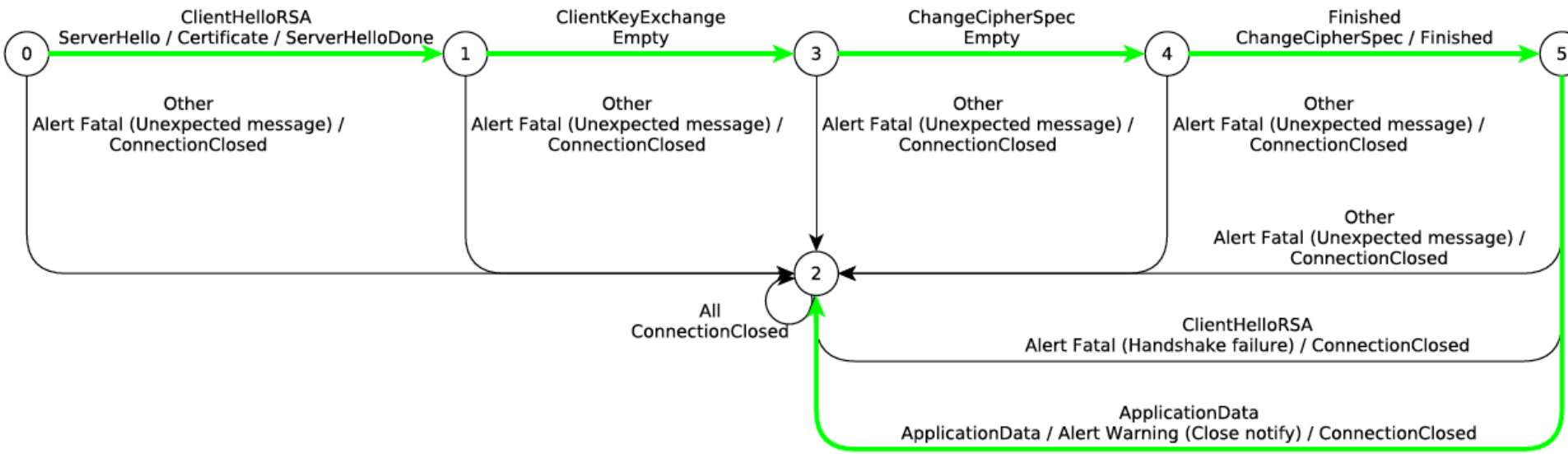
TUDelft … 10244 strings in total

# Printer controller, Smeenk et al. 2013

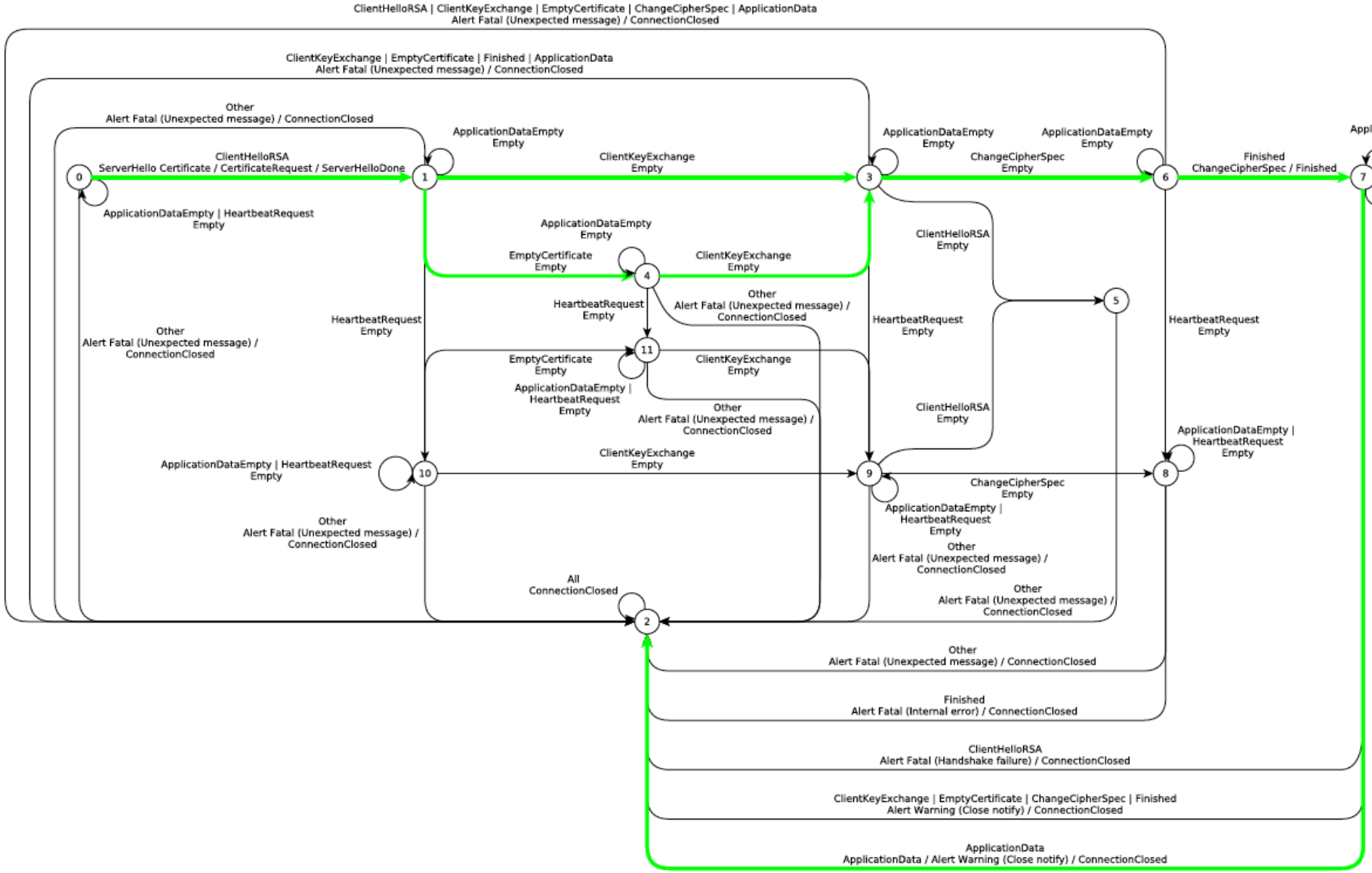


TU Delft

# Use Case: TLS RSA BSAFE



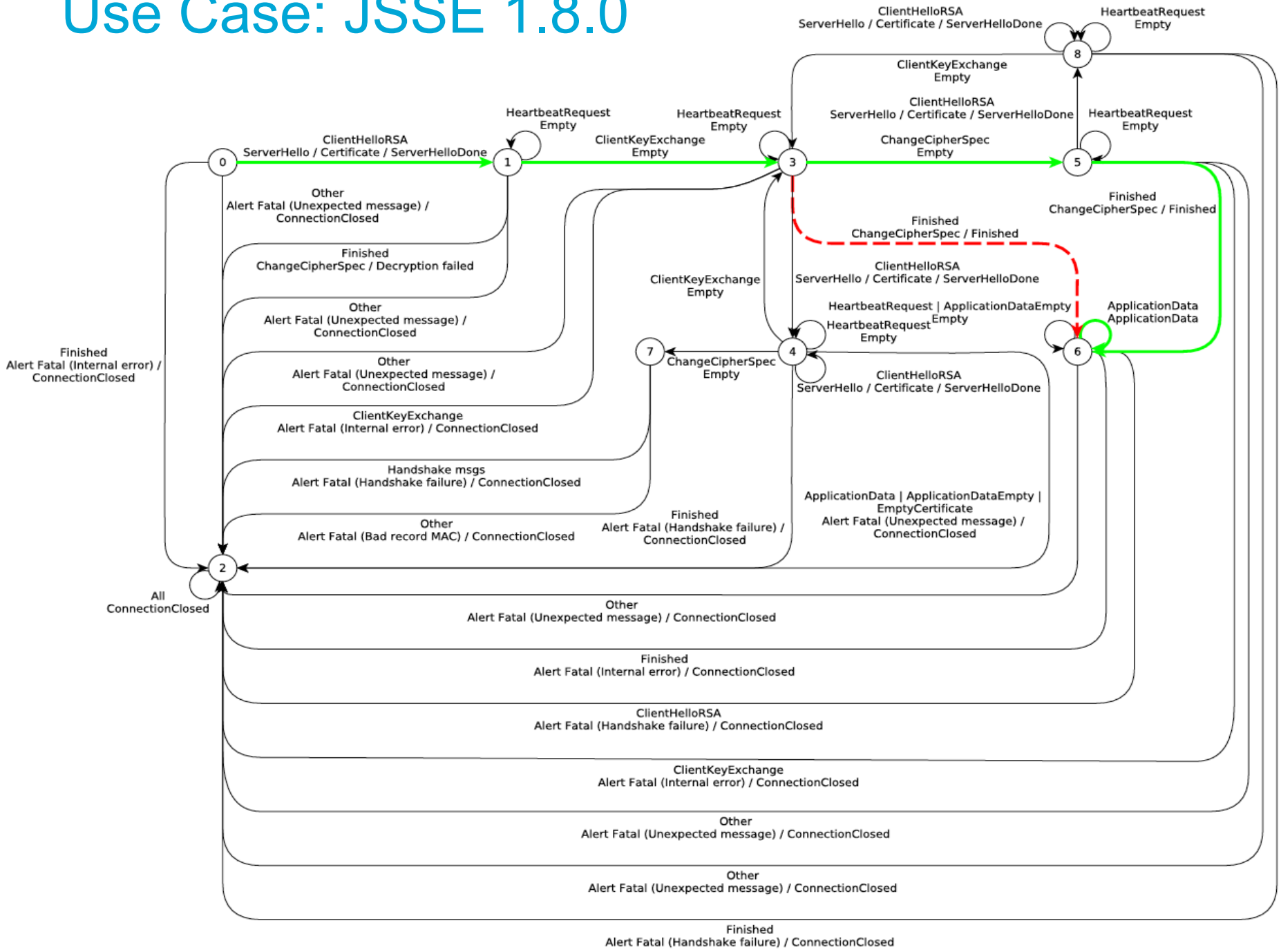
# Use Case: GNU TLS 3.3.8



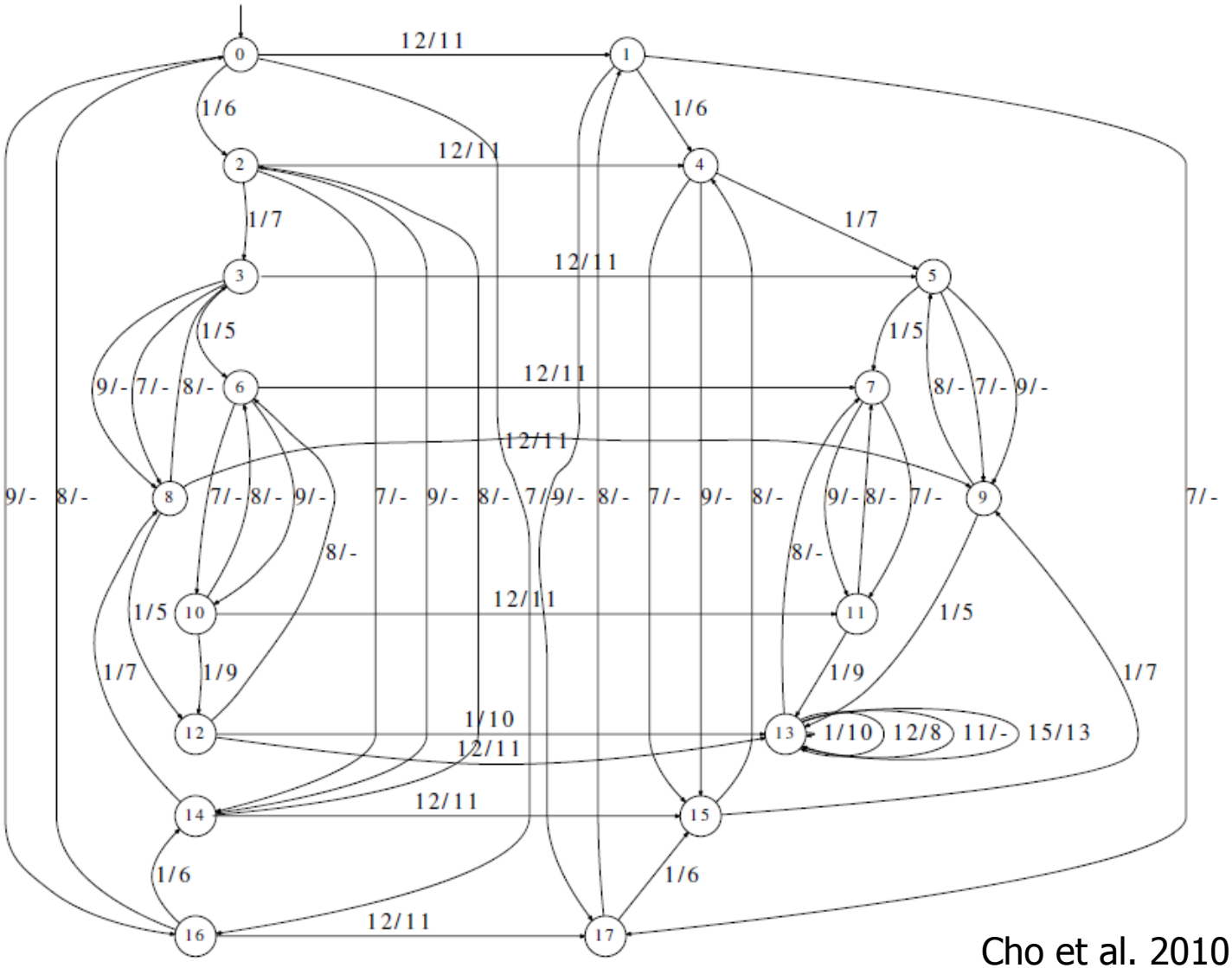

TUDelft

Joeri de Ruiter & Erik Poll 2015

# Use Case: JSSE 1.8.0

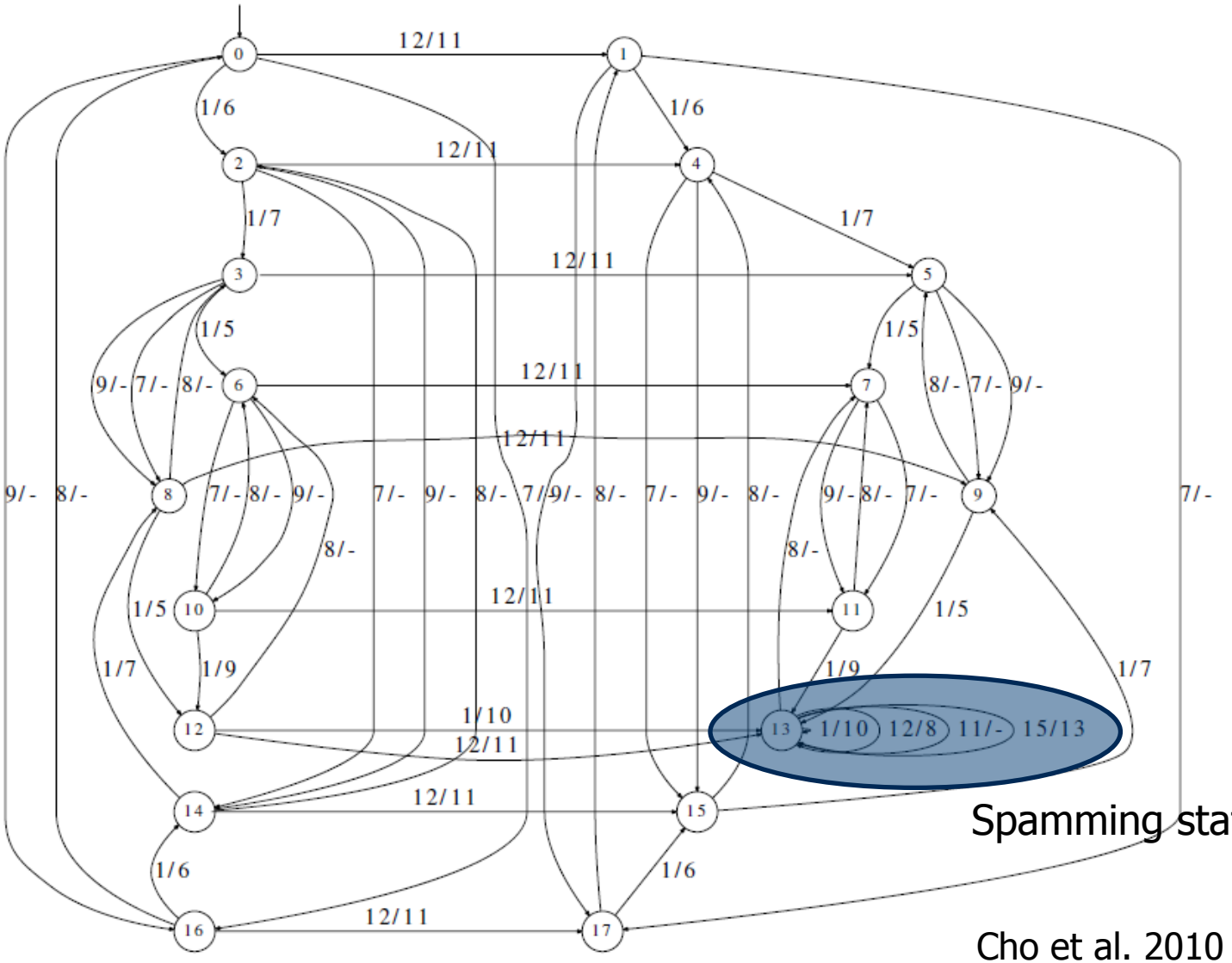


# Use Case: MegaD botnet protocol




Cho et al. 2010

# Use Case: MegaD botnet protocol




Spamming state

Cho et al. 2010

# MYHILL-NERODE AND THE HANKEL MATRIX

# 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*.

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

| | _ | 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 |
| **b** | | | | | | | | | | |
| **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 |

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 |
| **b** | | | | | | | | | | |
| **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

# Constructing a state machine

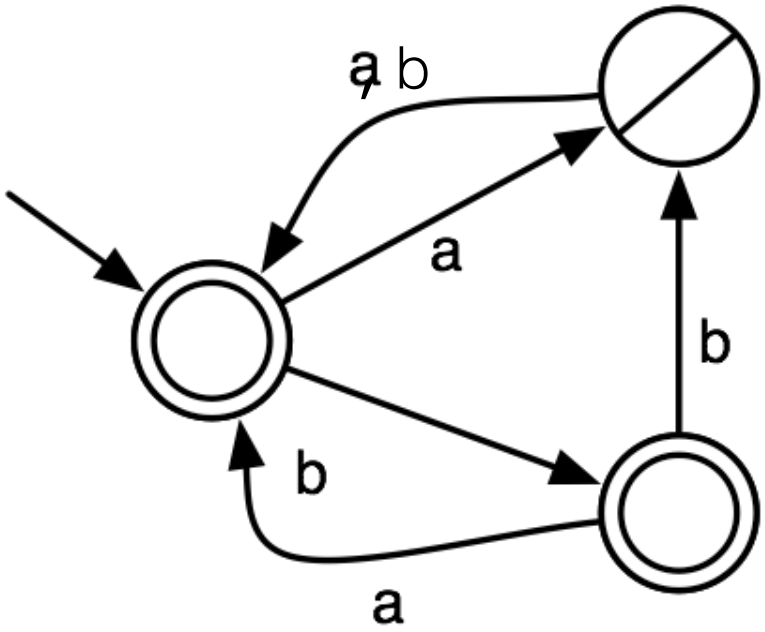
- Given a Hankel matrix for L
- What is the smallest state machine (DFA) for L?

# Q: what does this state machine look like?

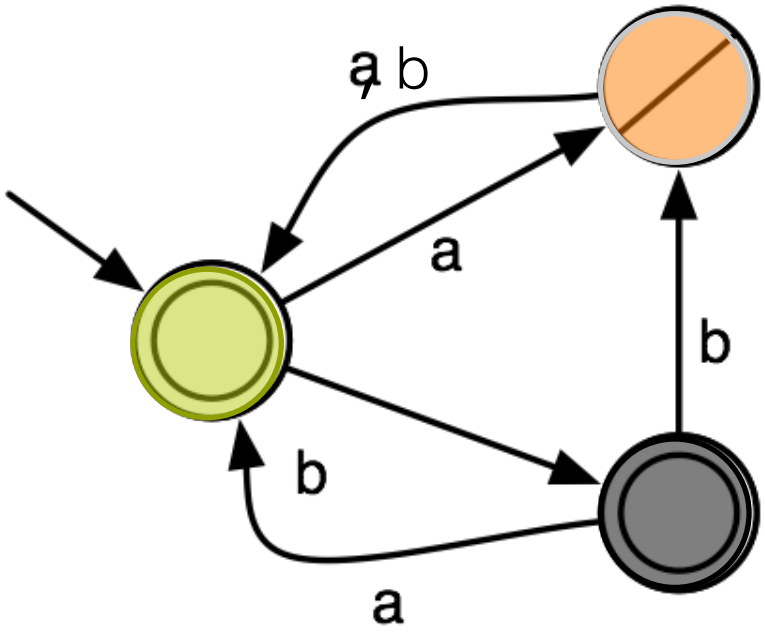
| | _ | a | b | aa | ab | ba | bb | aaa | aab | aba |
|---|---|---|---|---|---|---|---|---|---|---|
| _ | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| **a** | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| **b** | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| **aa** | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| **ab**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

| | _ | 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 | 1 | 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 |

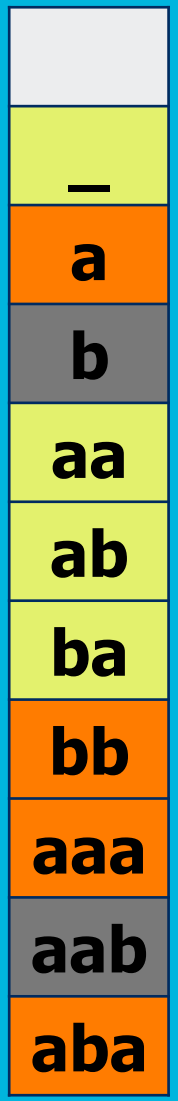
# Answer



# Answer



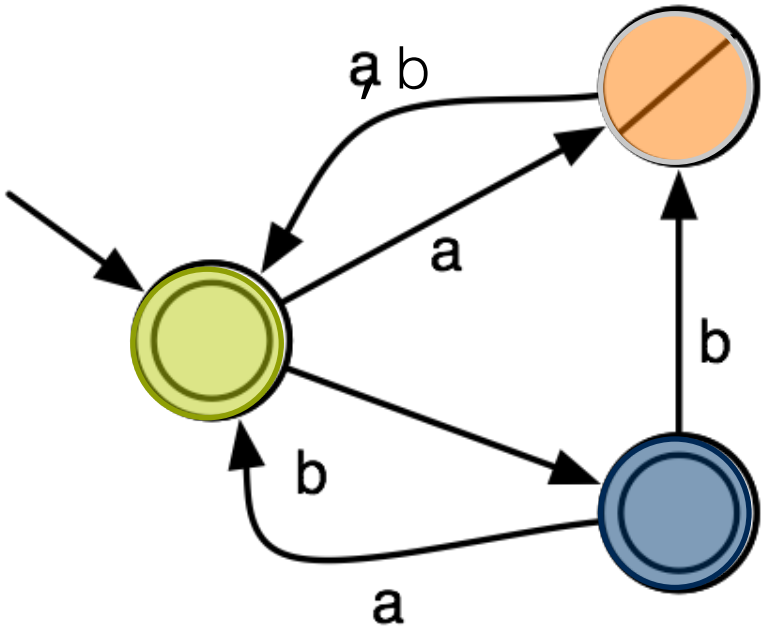
| | _ | a | b | aa | ab | ba | bb | aaa | aab | aba |
|---|---|---|---|---|---|---|---|---|---|---|
| _ | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| a | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| b | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| aa | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| ab | 1 | 0 | 1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

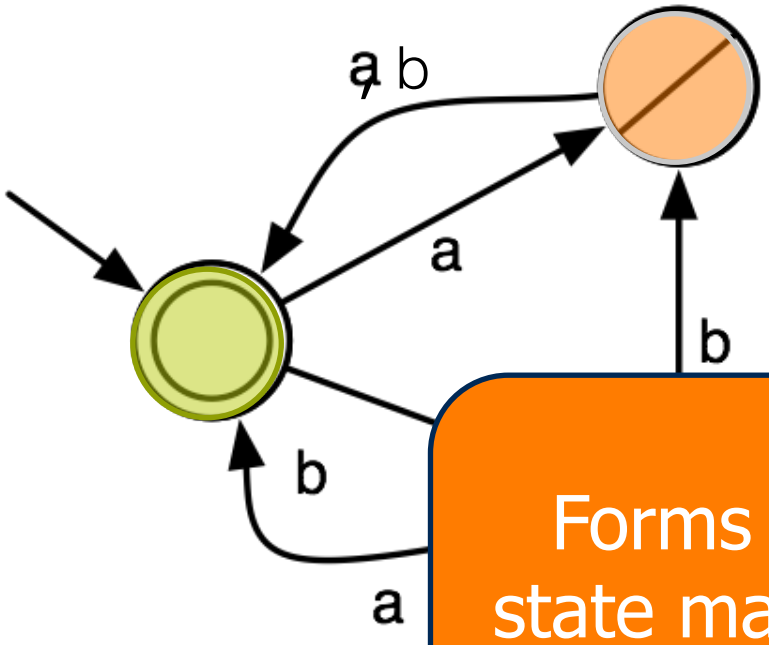
| |
|---|
| _ |
| a |
| b |
| aa |
| ab |
| ba |
| bb |

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



| |
|---|
| _ |
| a |
| b |
| aa |
| ab |
| ba |
| bb |

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




Forms the basis for state machine learning algorithms

# 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,+)

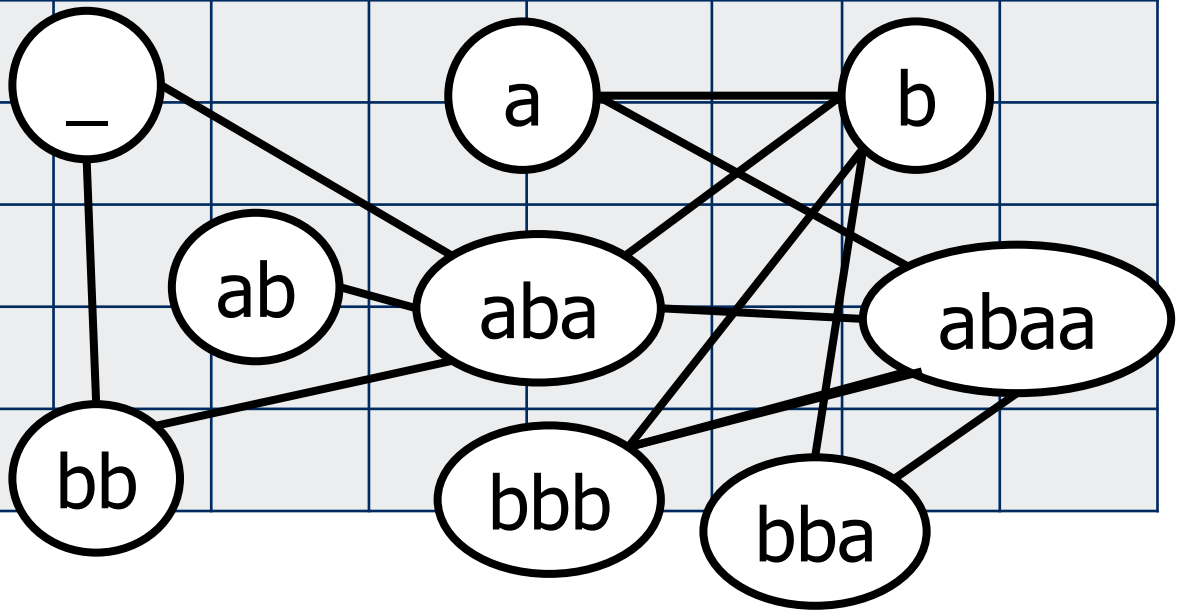
| | _ | a | b | ba | aba | aa | baa | abaa | bb | bbb | bba |
|---|---|---|---|---|---|---|---|---|---|---|---|
| _ | | 1 | 0 | | 1 | | | 0 | | 1 | 1 |
| a | 1 | | | 1 | | | 0 | | | | |
| b | 0 | | | 1 | | | | | 1 | | |
| ab | | 1 | | | | 0 | | | | | |
| aba | 1 | 0 | | | | | | | | | |
| abaa | 0 | |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



coloring such a matrix is NP-hard!

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

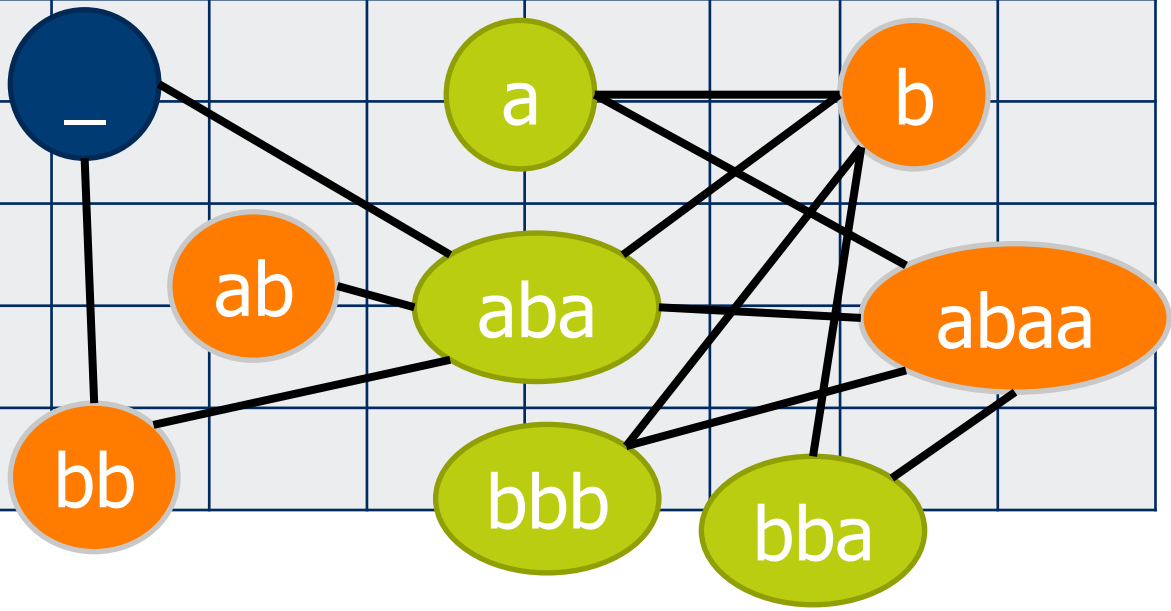
| | _ | a | b | ba | aba | aa | baa | abaa | bb | bbb | bba |
|---|---|---|---|---|---|---|---|---|---|---|---|
| _ | | 1 | 0 | | 1 | | | 0 | | 1 | 1 |
| a | 1 | | | 1 | | | 0 | | | | |
| b | 0 | | | 1 | | | | | 1 | | |
| ab | | 1 | | | | 0 | | | | | |
| aba | 1 | 0 | | | | | | | | | |
| abaa | 0 | |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




TUDelft

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

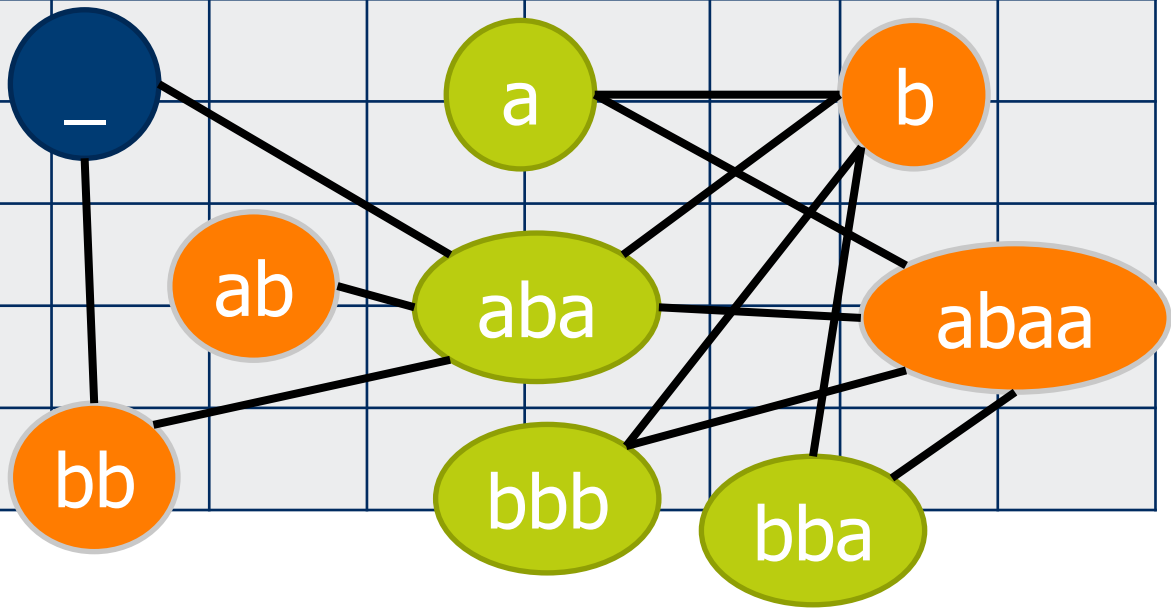
| | _ | a | b | ba | aba | aa | baa | abaa | bb | bbb | bba |
|---|---|---|---|---|---|---|---|---|---|---|---|
| _ | | 1 | 0 | | 1 | | | 0 | | 1 | 1 |
| a | 1 | | | 1 | | | 0 | | | | |
| b | 0 | | | 1 | | | | | 1 | | |
| ab | | 1 | | | | 0 | | | | | |
| aba | 1 | 0 | | | | | | | | | |
| abaa | 0 | |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

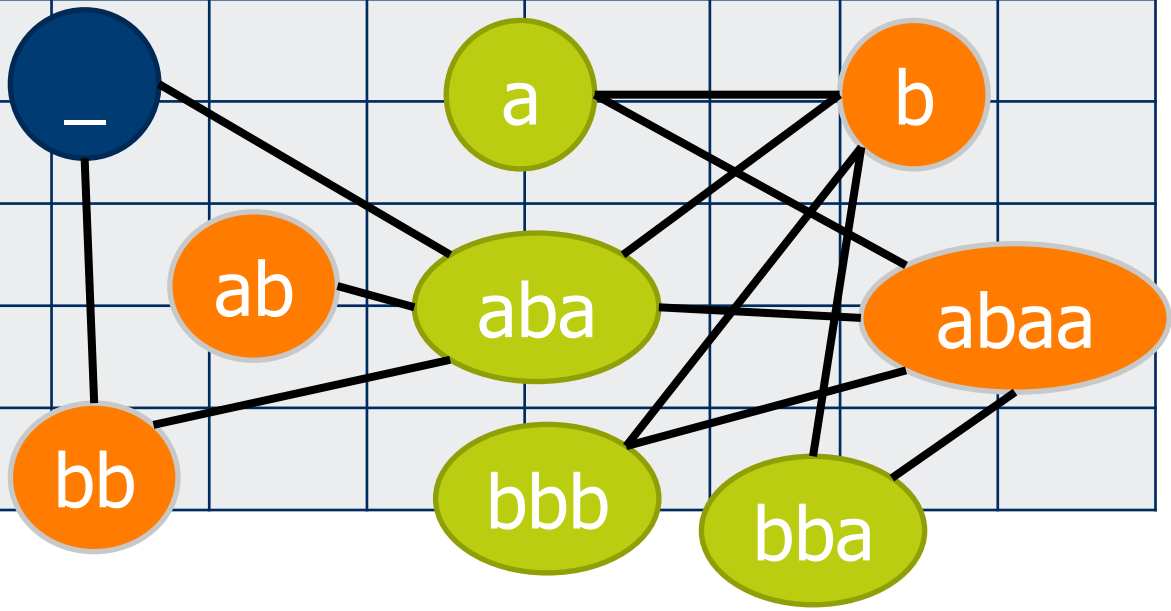



TU Delft

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

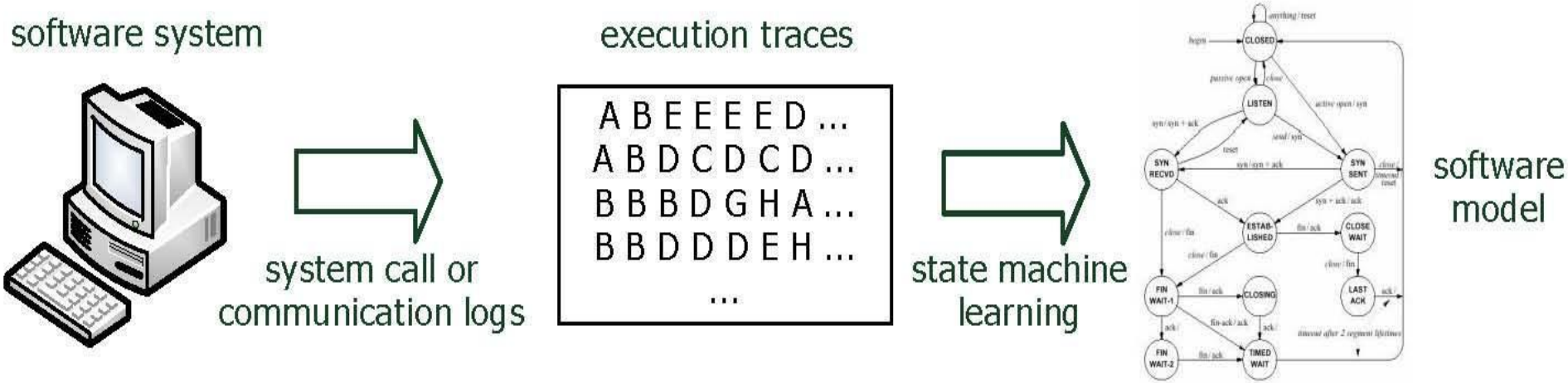
| | | | | | | | baa | abaa | bb | bbb | bba |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | 0 | | 1 | 1 |
| | | | | | | | 0 | | | | |
| | | | | | | | | | 1 | | |
| ab | | 1 | | | | 0 | | | | | |
| aba | 1 | 0 | | | | | | | | | |
| abaa | 0 | | | | | | | | | | |
| bb | | 1 | 1 | | | | | | | | |
| 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? →

| | _ |
|---|---|
| _ | 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!

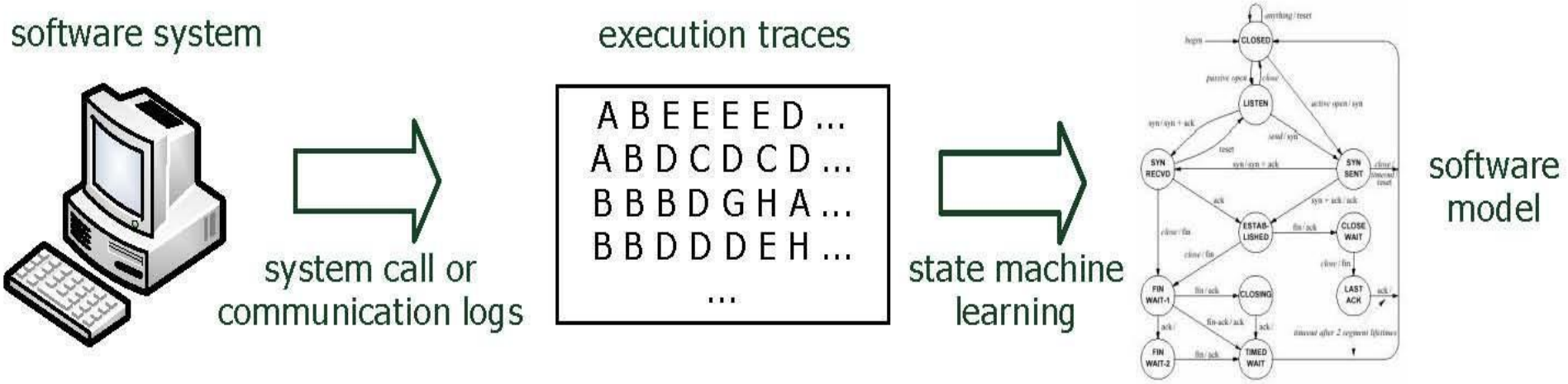
complete

| | _ | 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

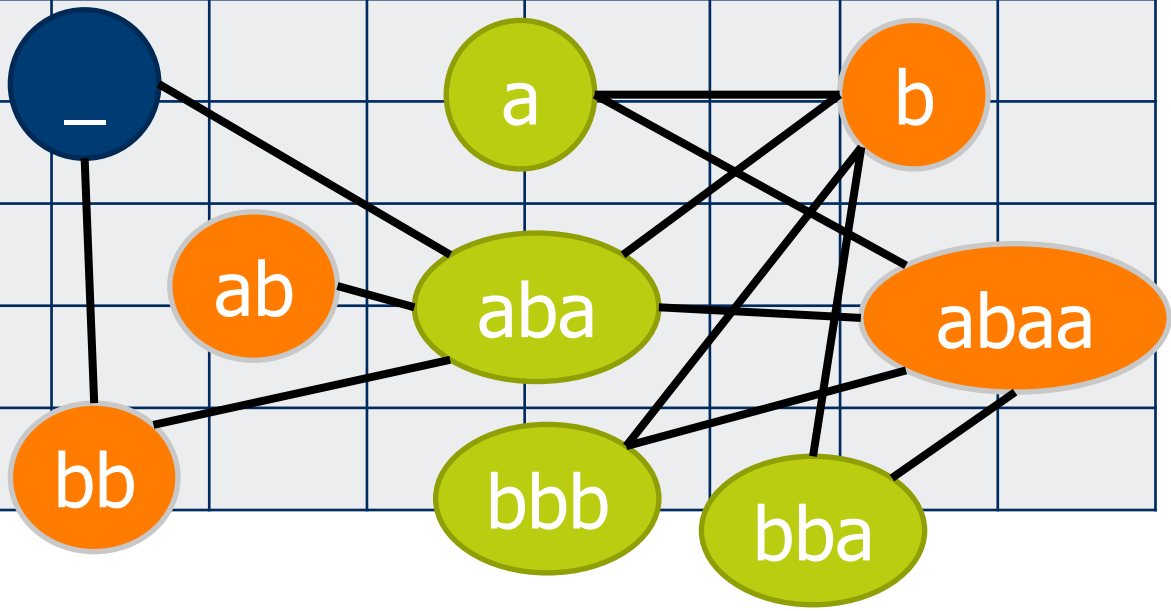
# Use the information we have…




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

# Solution: state merging

- Very effective greedy method
  - Initially assume a large DFA
  - Iteratively combine similar states
  - Until no more state can be combined
- State-of-the-art since 1998

# 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

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

# 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 |

# 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
  - …

| | _ | 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 |

| | _ | 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 |

| | _ | 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 | | | | | | |

Copying values is less important

Merged rows are statistically close anyway…

| | _ | 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

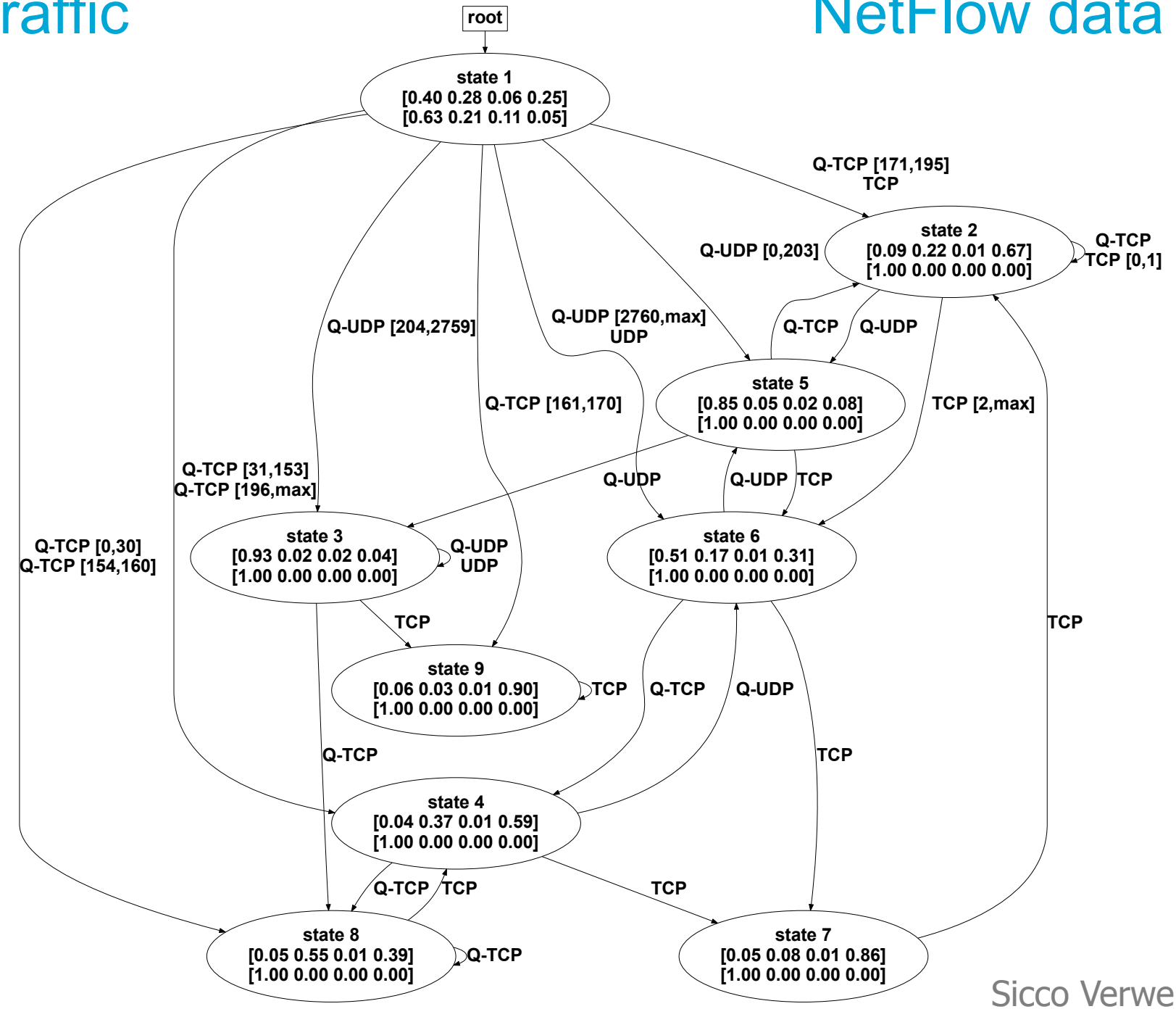
- Problem: the matrix M is fully specified, but very noisy
- Three steps:
  1. Compute singular values of M
  2. Keep the n largest singular values, recompute M
  3. Color M assuming it is exact
- Step 2 compresses M to an n-state object, removing noise
  - Given sufficient data, this turns out to be a non-deterministic PFA!
  - If not, what is learned it is not really defined…
- Step 3 is not hard because M is fully specified

# Key ingredients

- When are two states inconsistent? (both active and passive)
  - Positive gets merged with negative
  - Input/output does not match
  - Other events are possible
  - Statistics do not match
  - If the Markov property is violated
  - …
- What to use as merge score? (passive)
  - Number of positive-positive, negative-negative merges
  - Overlap in sets of possible events
  - Statistical distance
  - …
- Should we ensure correctness at all? (spectral, graph coloring)

# Botnet traffic

# NetFlow data



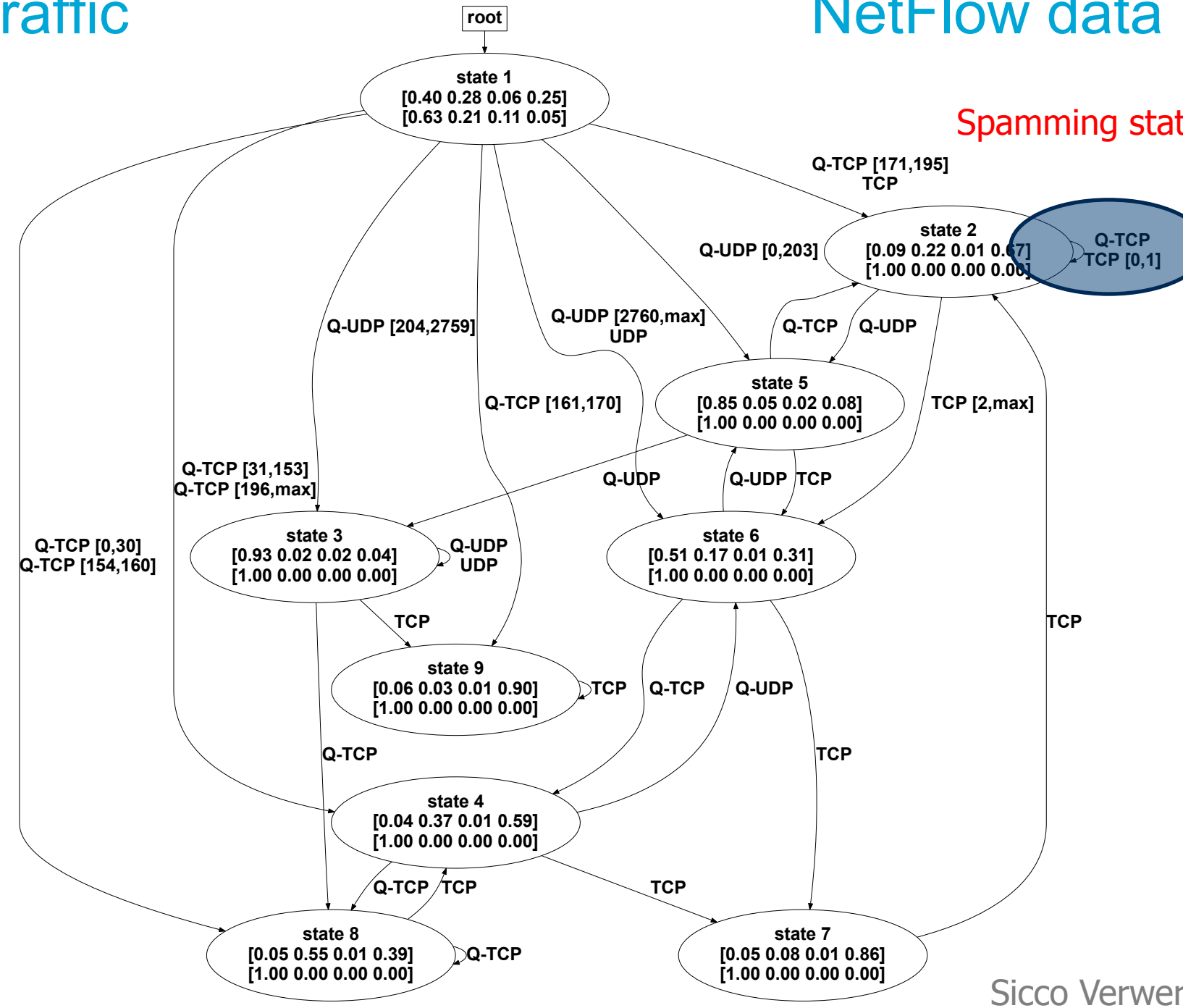

TU Delft

Sicco Verwer

# Botnet traffic

# NetFlow data

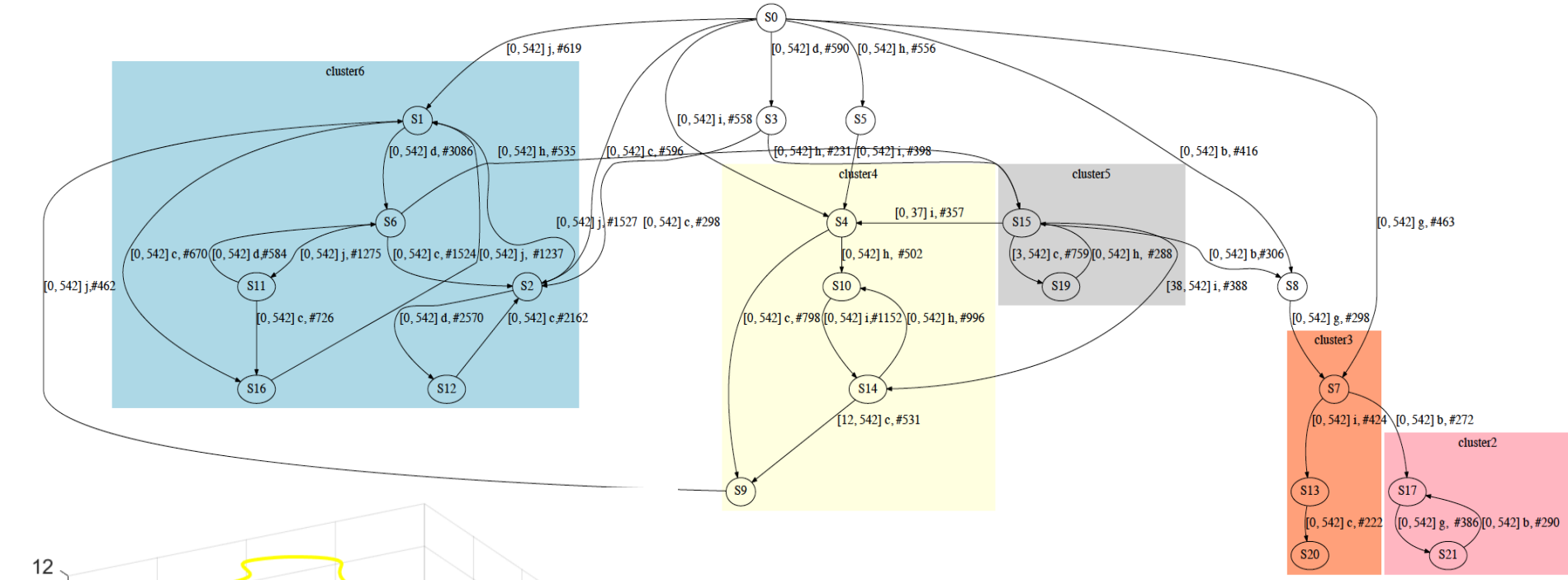
Spamming stat

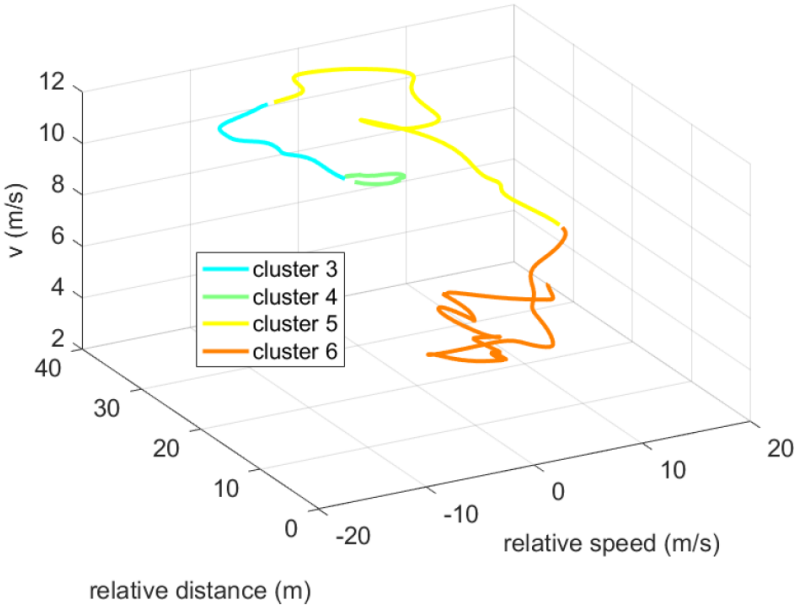



TU Delft

Sicco Verwer

# Automated driving




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

# Near Future

- First method for systematically analyzing network data
  1. Connect to a monitor
  2. Leave alone for a week
  3. Analyze the learned state machines
- Develop learning tool

*fast & flexible*