Learning State Machines

Learning in the limit, Myhill-Nerode, the Hankel matrix, Active and Passive learning

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

TU Delft

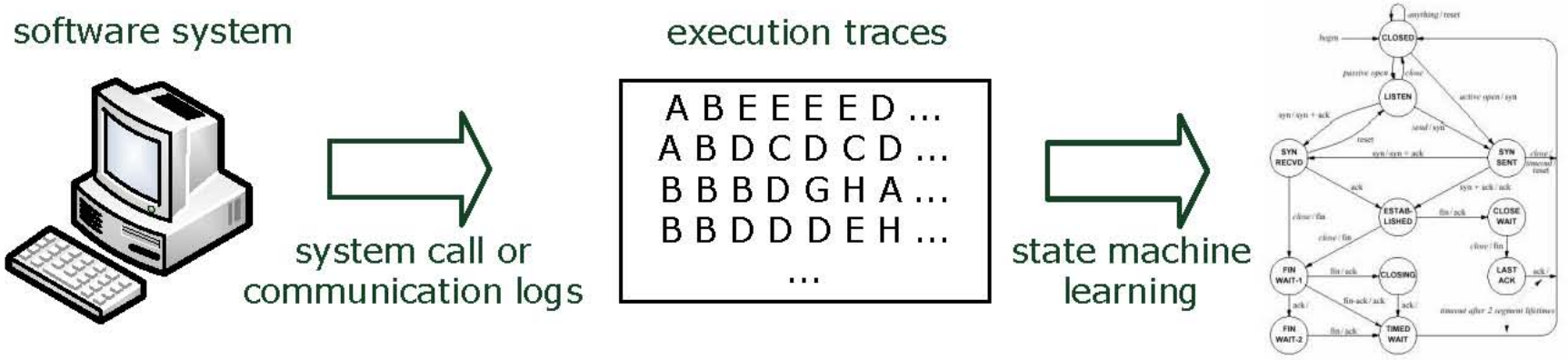
Learning (reverse-engineering)

- 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)

State machine learning



- 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?

Learning DFAs from labeled data

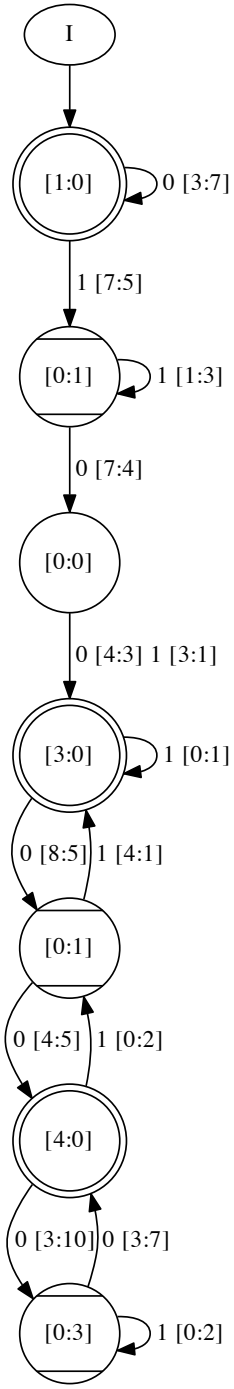
- Input:
  - Labeled data, positive and negative traces
- Goal:
  - Find the smallest DFA that is consistent with the data

Learning DFAs from labeled data

- Input:
  - Labeled data, positive and negative traces
- Goal:
  - Find the smallest DFA that is consistent with the data
- Occam's razor
  - *Among competing hypotheses, the one with the fewest assumptions should be selected.*

Example

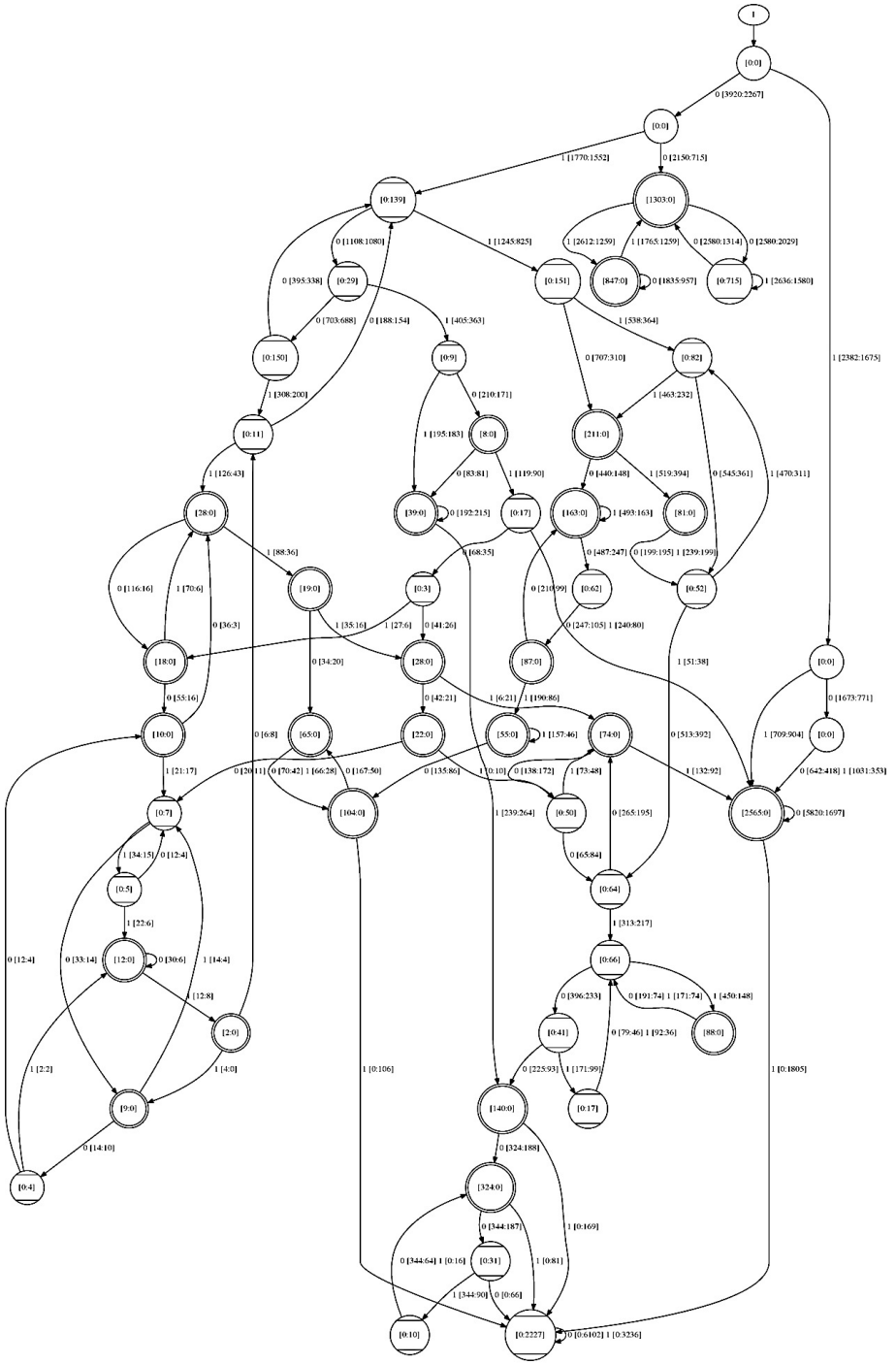
1 9 1 0 0 0 0 0 0 0 0
1 15 0 1 1 0 0 0 1 0 1 0 1 0 1 0 0
0 5 0 1 0 0 0
1 2 0 0
0 12 1 1 0 0 0 0 0 0 0 1 0 0
1 5 1 0 1 0 0
1 3 1 0 0
0 6 0 0 0 0 0 1
1 3 1 0 1
0 20 1 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0
0
0 13 0 1 1 1 0 1 0 0 0 0 0 0 0
1 3 1 0 1
1 7 1 0 0 0 0 0 0



TU Delft

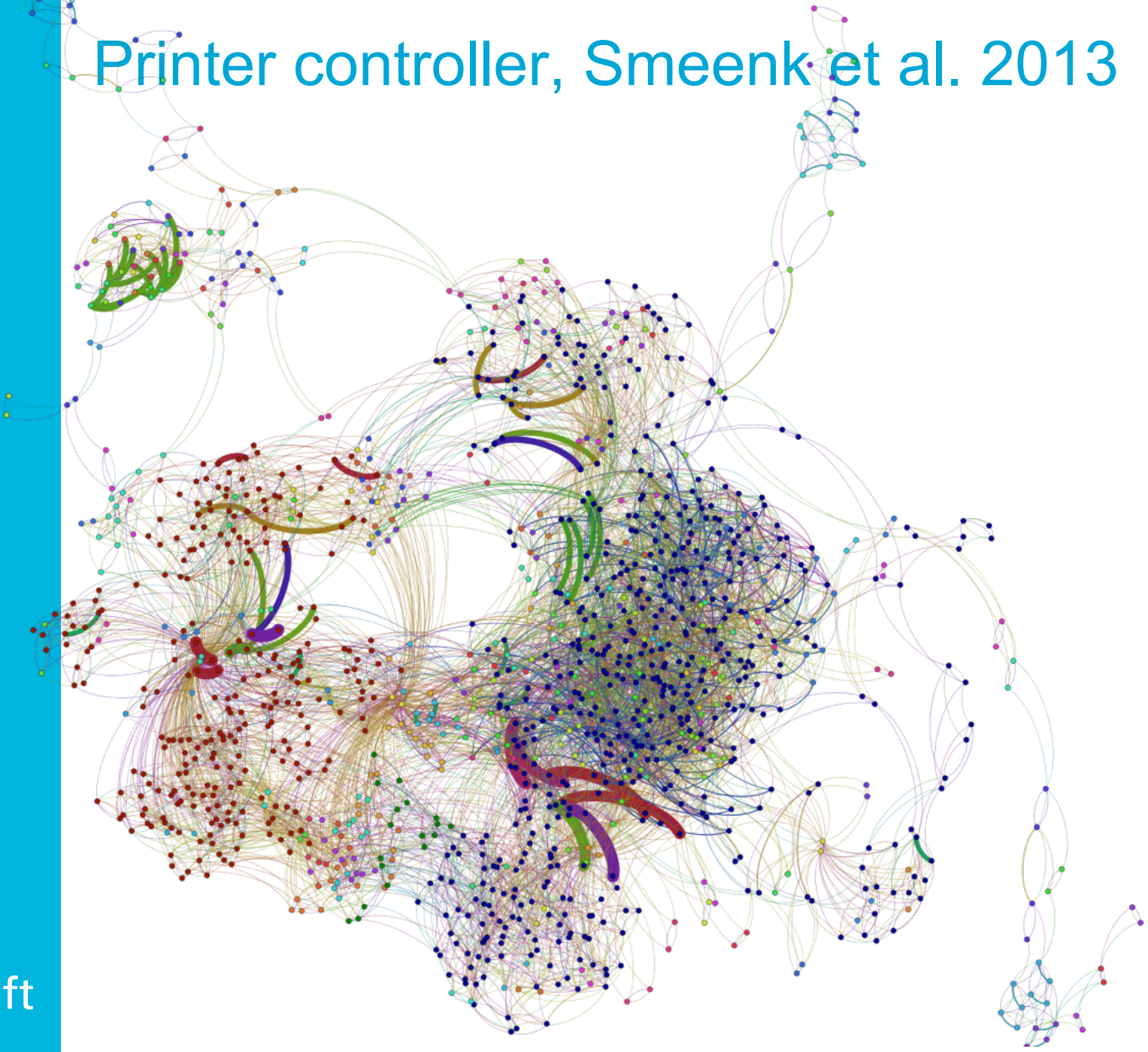
Example

1 9 1 0 0 0 0 0 0 0 0
1 15 0 1 1 0 0 0 1 0 1 0 1 0 1 0 0
0 5 0 1 0 0 0
1 2 0 0
0 12 1 1 0 0 0 0 0 0 0 1 0 0
1 5 1 0 1 0 0
1 3 1 0 0
0 6 0 0 0 0 0 0 1
1 3 1 0 1
0 20 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 1 0 0 0
0 13 0 1 1 1 0 1 0 0 0 0 0 0 0
1 3 1 0 1
1 7 1 0 0 0 0 0 0
....
10244 strings in total



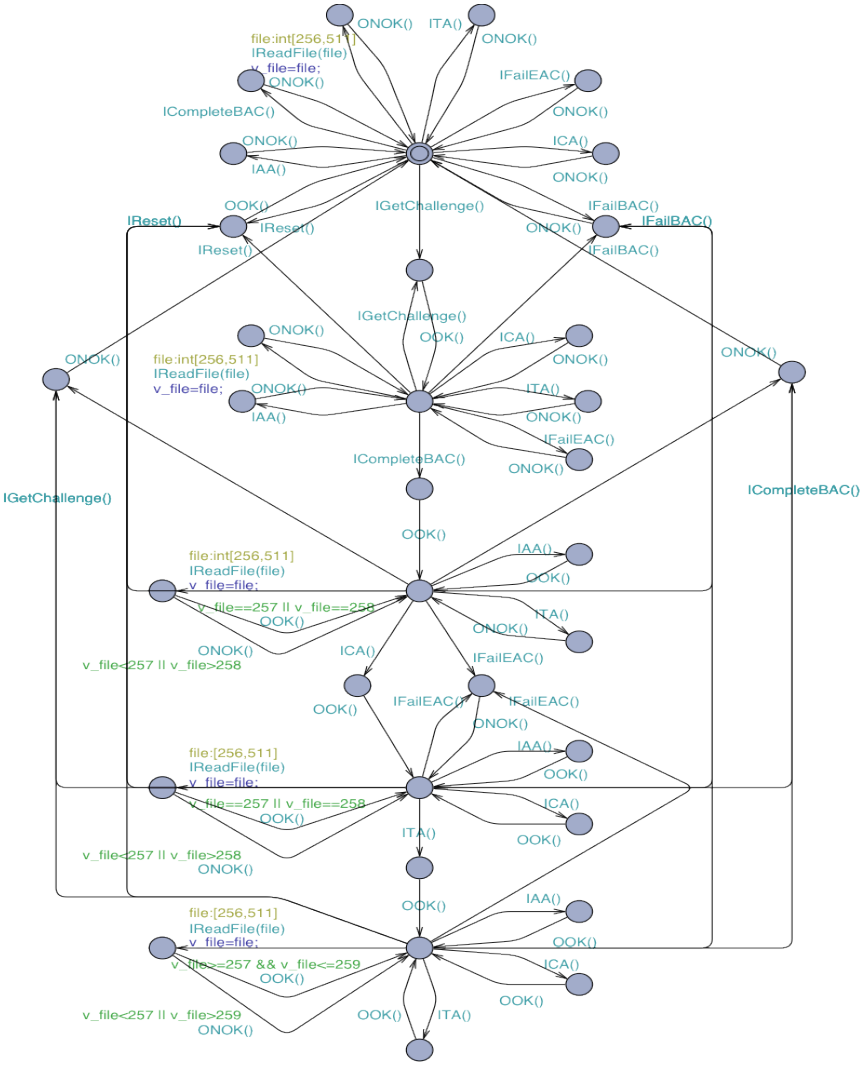
TU Delft

Printer controller, Smeenk et al. 2013



TU Delft

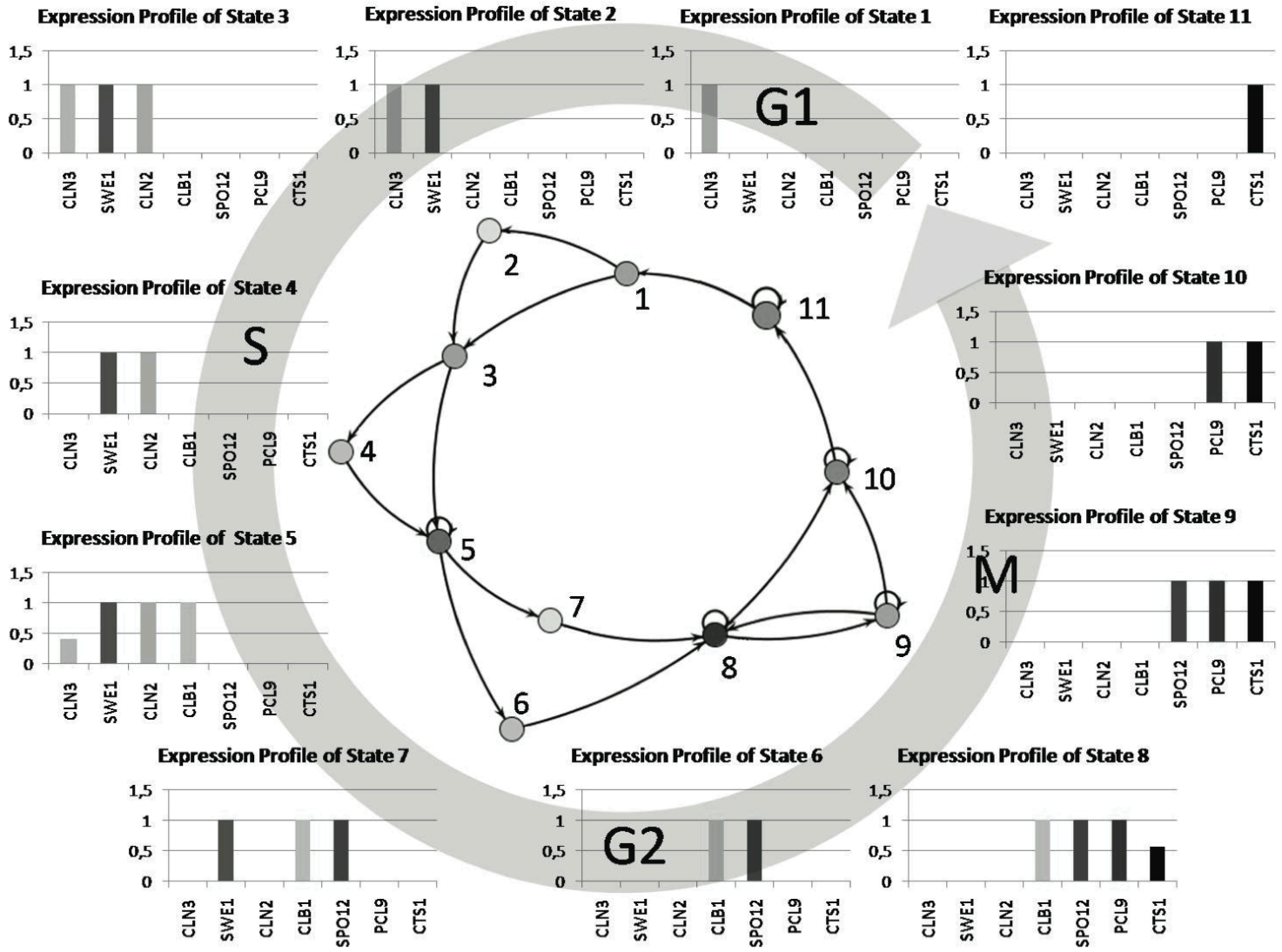
Biometric Passport, Aarts et al. 2010



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

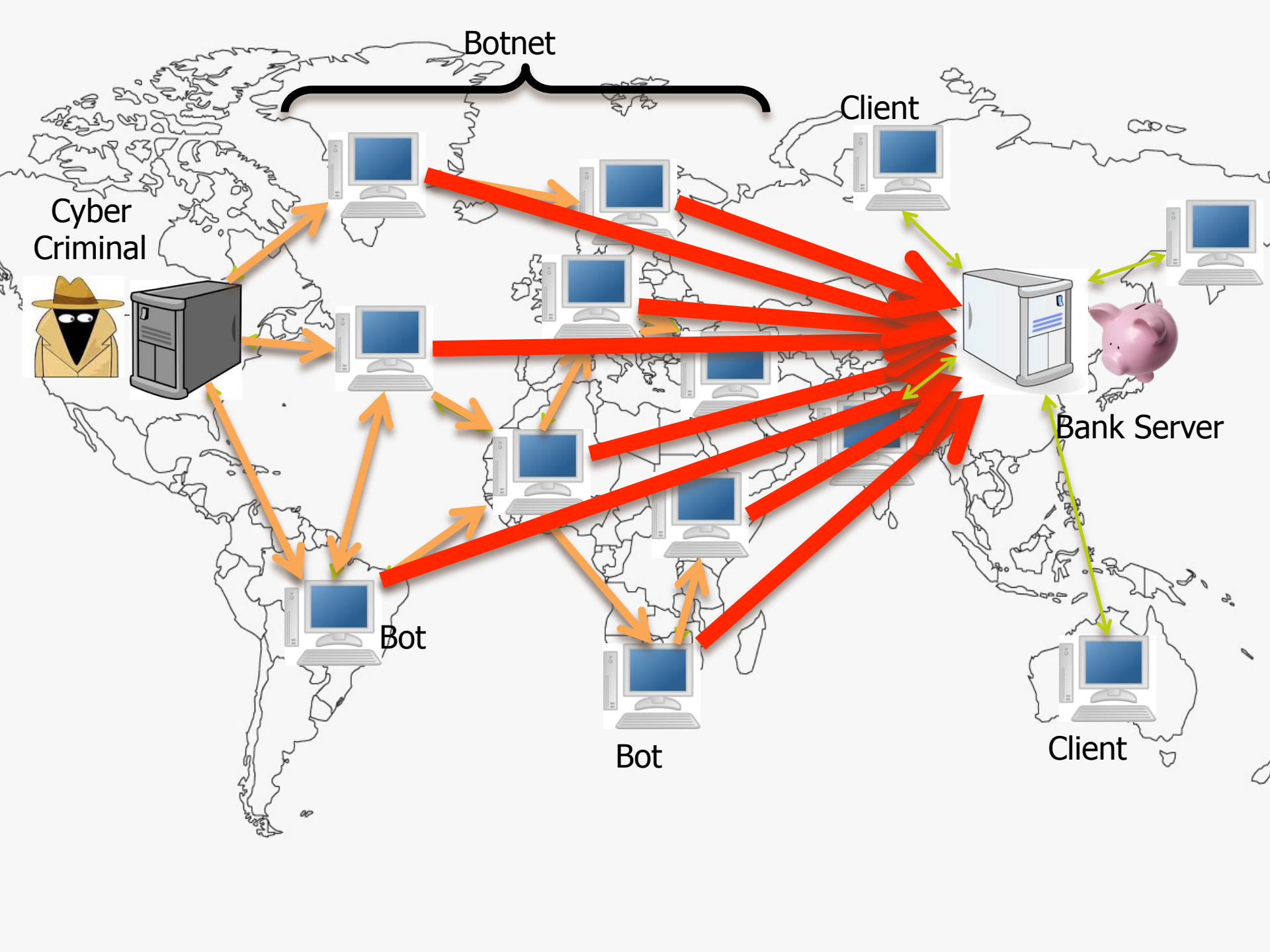
Yeast cycle, Schmidt et al. 2013



TU Delft

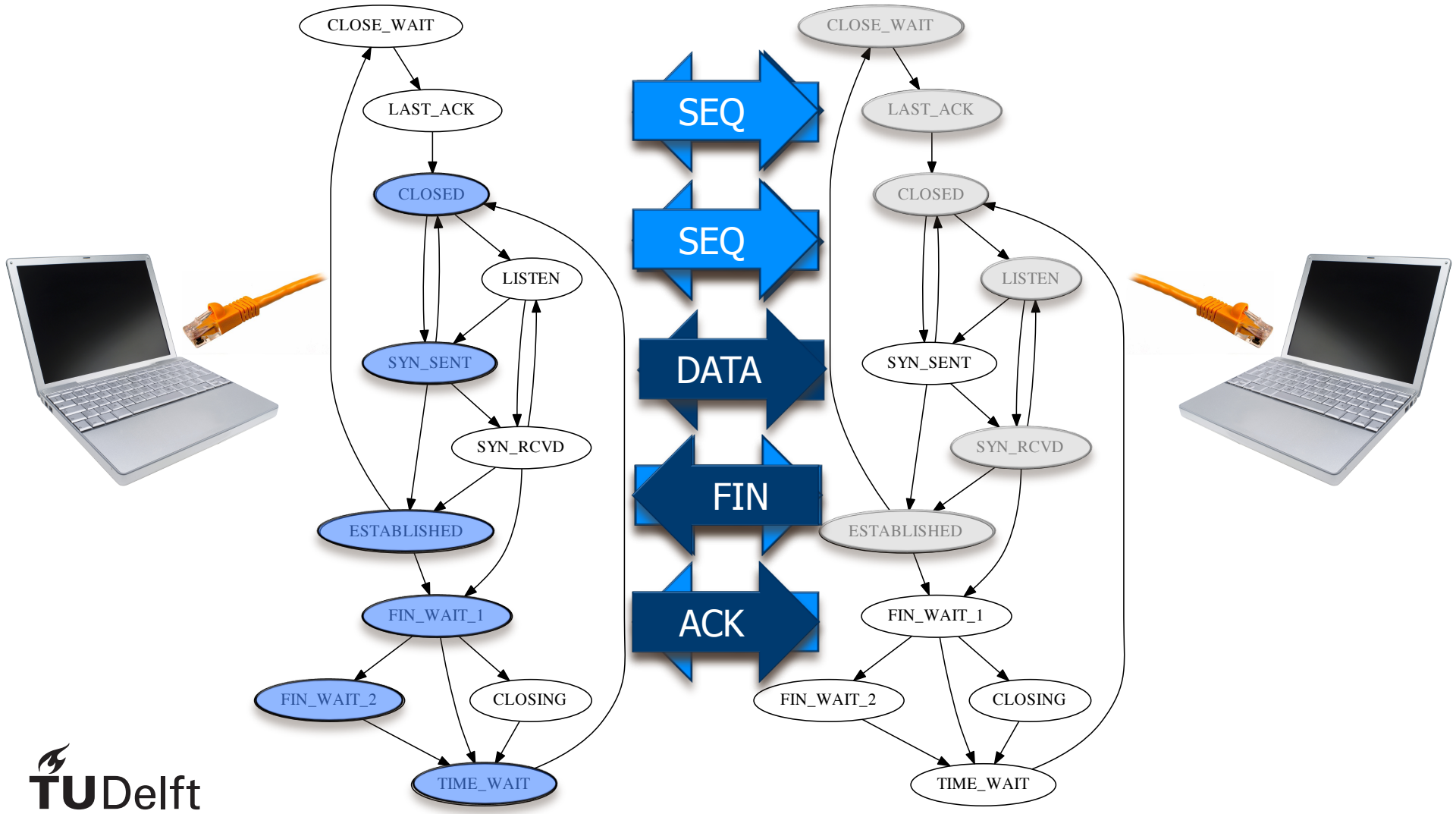
Preventing botnets

- Traditional
  - code analysis and finding malware fingerprints
- Code/binary analysis is mostly manual and increasingly harder
  - Code obfuscation
  - Encryption
  - Self-modifying
- Behavior-based analysis is much harder to thwart
  - Bots need to communicate!


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!

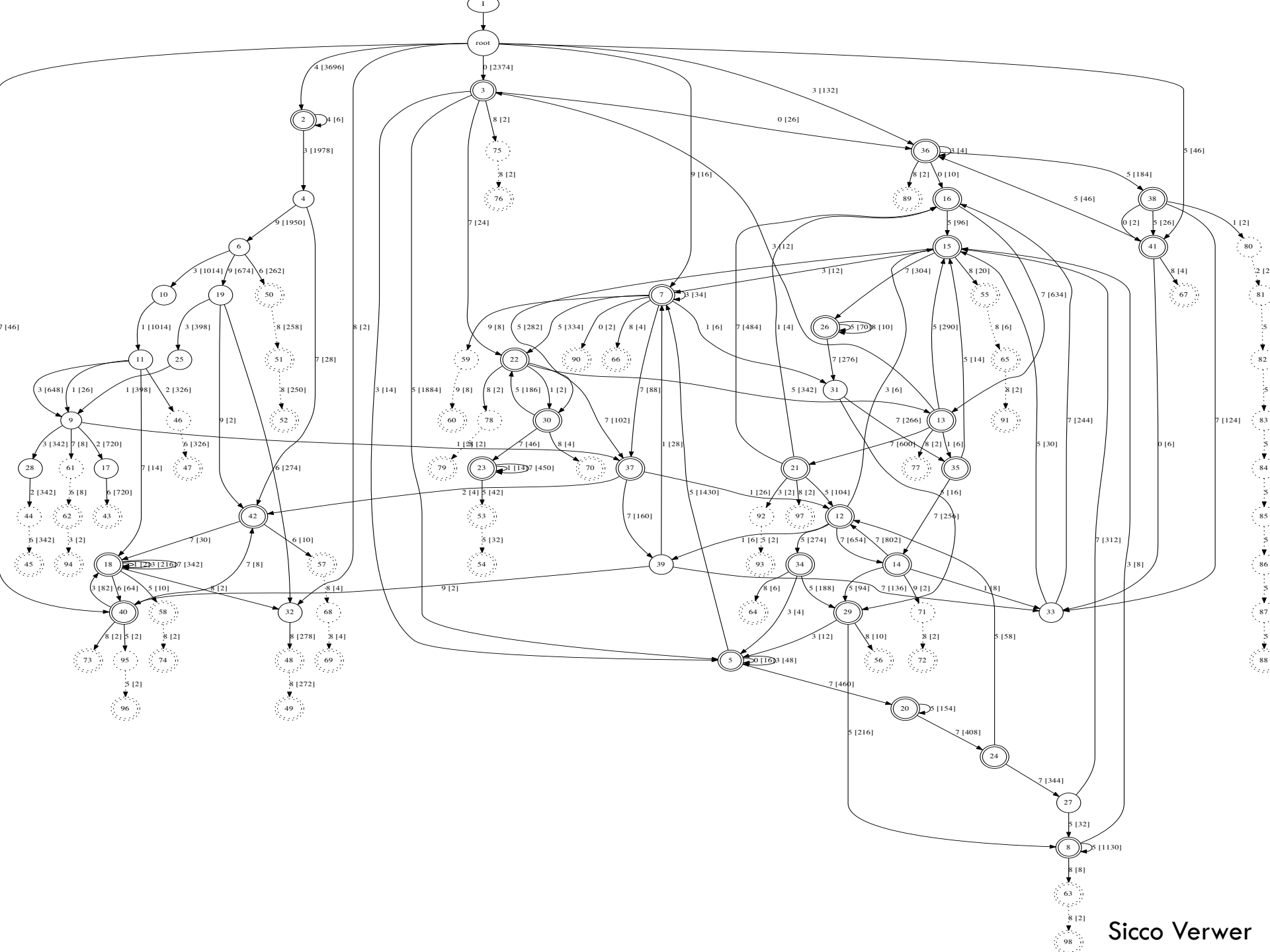
- A, B, C, D, C, D, C, D, C, E, C, E
- A, B, C, E, C, E
- A, B, C, D, C, E, C, E
- …..

Then: look for patterns!

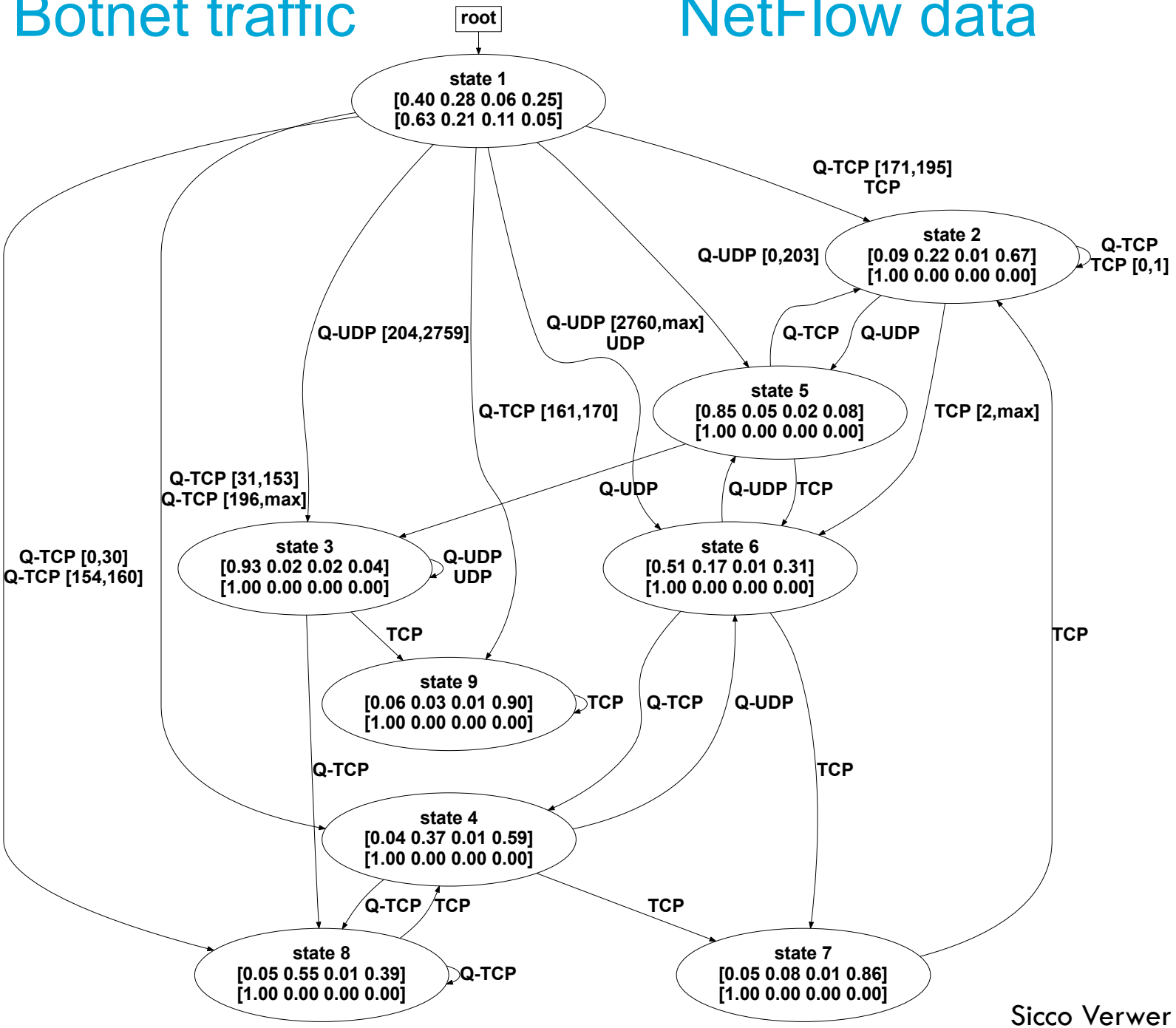
- A, B, C, D, C, D, C, D, C, E, C, E
- A, B, C, E, C, E
- A, B, C, D, C, E, C, E
- …..

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
- …..


1
root
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
Sicco Verwer

Botnet traffic NetFlow data



TU Delft

Sicco Verwer

NetFlow data --- fingerprinting

- 4 scenarios: click-fraud, spamming, DDoS, …
- For each scenario:
  - Learn state machine model from 1 malicious host
  - Behavioral difference is small → malicious

TP	TN	FP	FN
9	377	0	0
9	153	0	0
1	35	0	1
0	7	0	2

NetFlow data --- anomaly detection

- 4 scenarios: click-fraud, spamming, DDoS, …
- For each scenario:
  - Learn state machine model from a large set of benign hosts
  - Behavioral difference to every model is large → malicious

TP	TN	FP	FN
10	561	3	0
10	225	6	0
2	50	4	1
0	8	1	3

WHAT IS LEARNING?

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

Learning theory

- Field that studies learning problems, or how to find a model for data
  - computational complexity
  - approximation algorithms
  - statistical guarantees
  - convergence properties
  - ...
- One of the oldest fields in computer science, contributed to many innovations
  - neural networks, support vector machines, minimum description length
  - grammatical inference, state machine learning

Learning in the limit

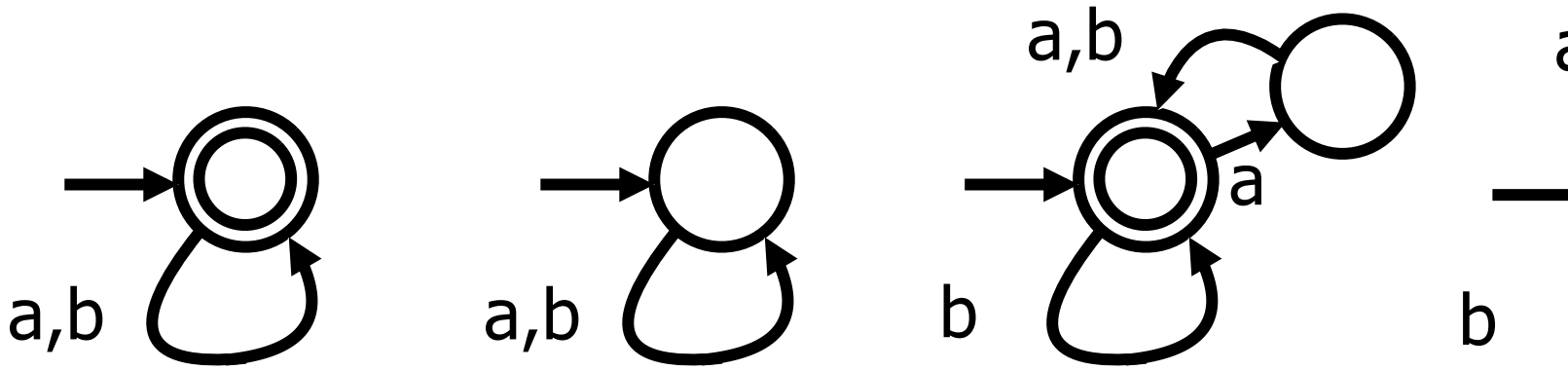
- Studies learning from infinite data
- Views learning as an ongoing process
  1. learner receives some data from a target concept C
  2. updates its hypothesis H, then goes to 1.
- Learning in the limit is successful if it converges:
  - L(H) = L(C), at some point in time t
  - and L(H) = L(C) at all time points > t

  - All data will be presented to the learner
  - Learner does not need to know when it converges

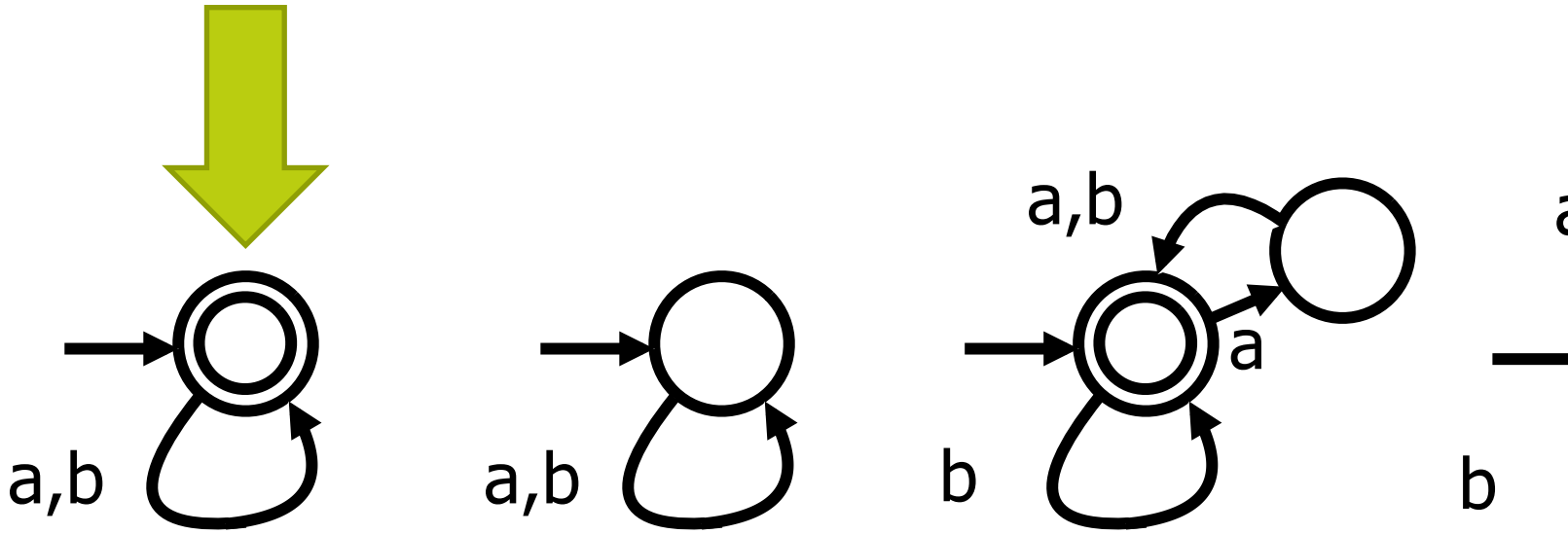
Learning by enumeration

- A simple learning algorithm:
  - Given an enumeration H_1, H_2, H_3, ...
  - Assume H_1 until it is inconsistent with the data
  - Then assume the first H_n such that there is no H_m with $m < n$ that is consistent with the data
  - Read more data and continue
- Many learning algorithms are of this type, but traverse the enumeration in different (smarter) ways

Learning by enumeration

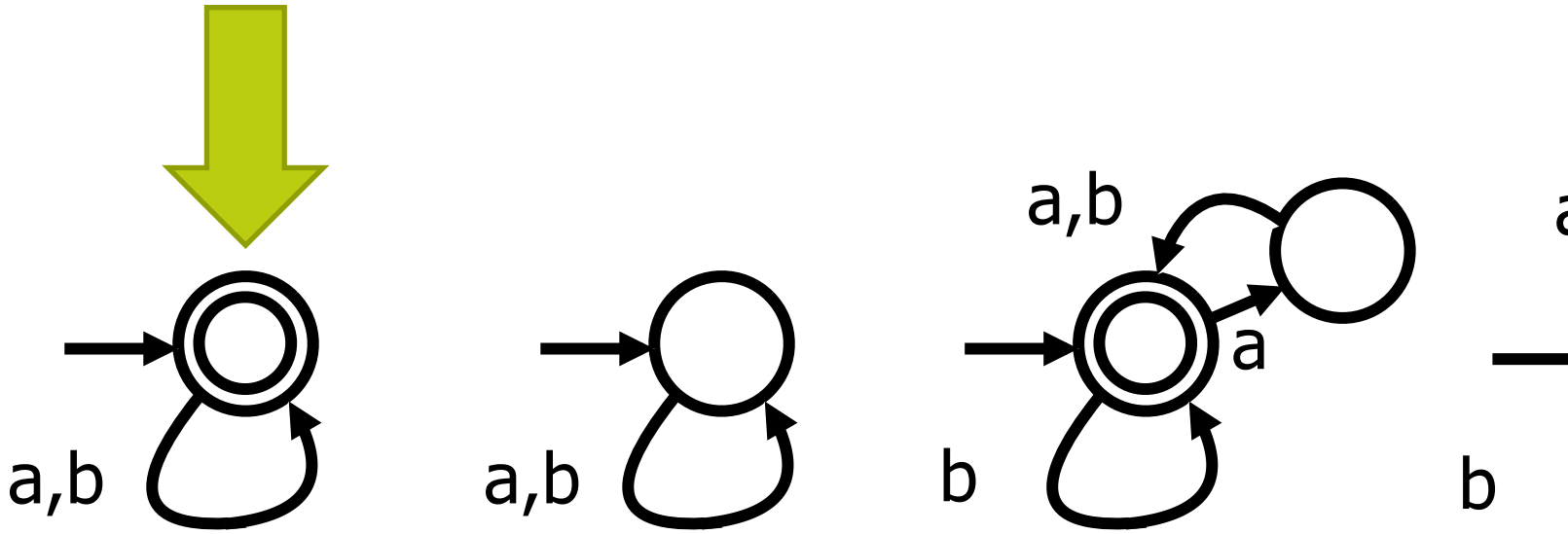


Learning by enumeration



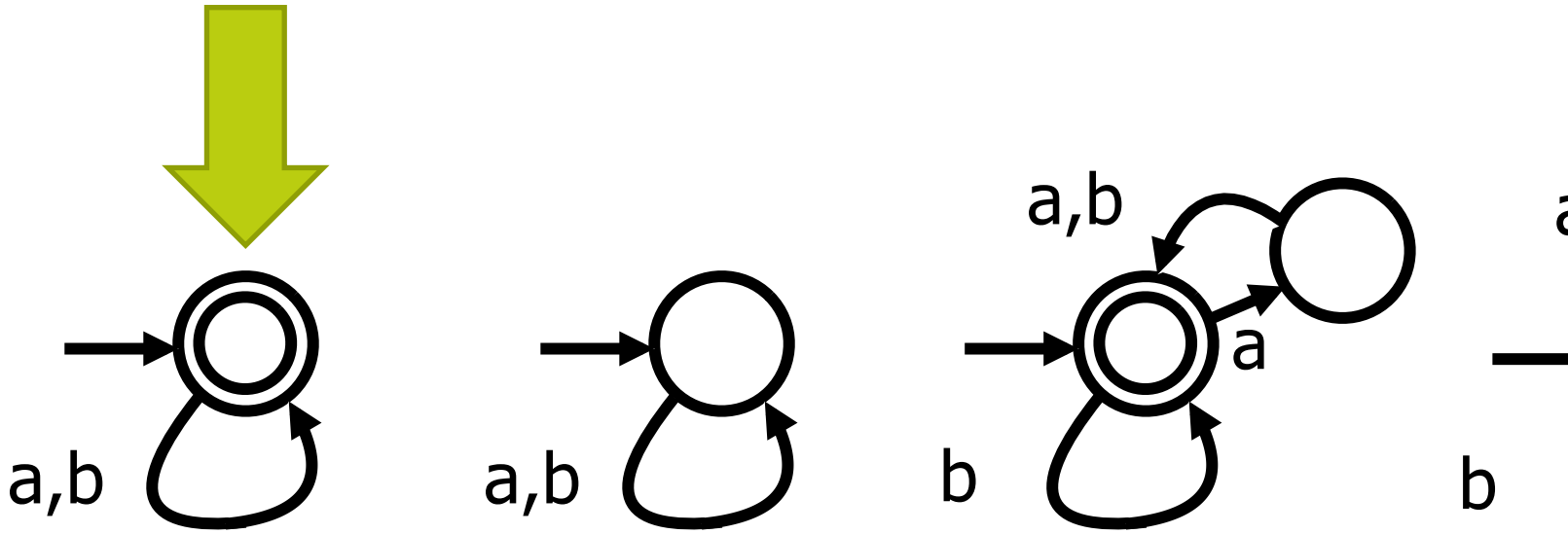
Examples: (a,+)

Learning by enumeration



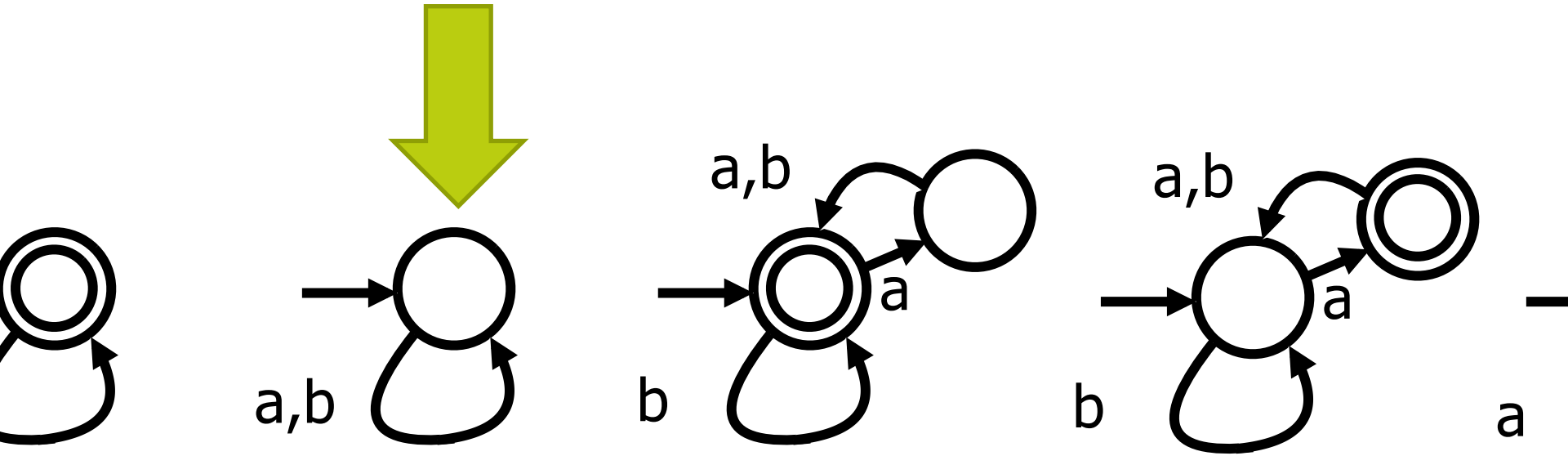
Examples: (a,+)(aa,+)

Learning by enumeration



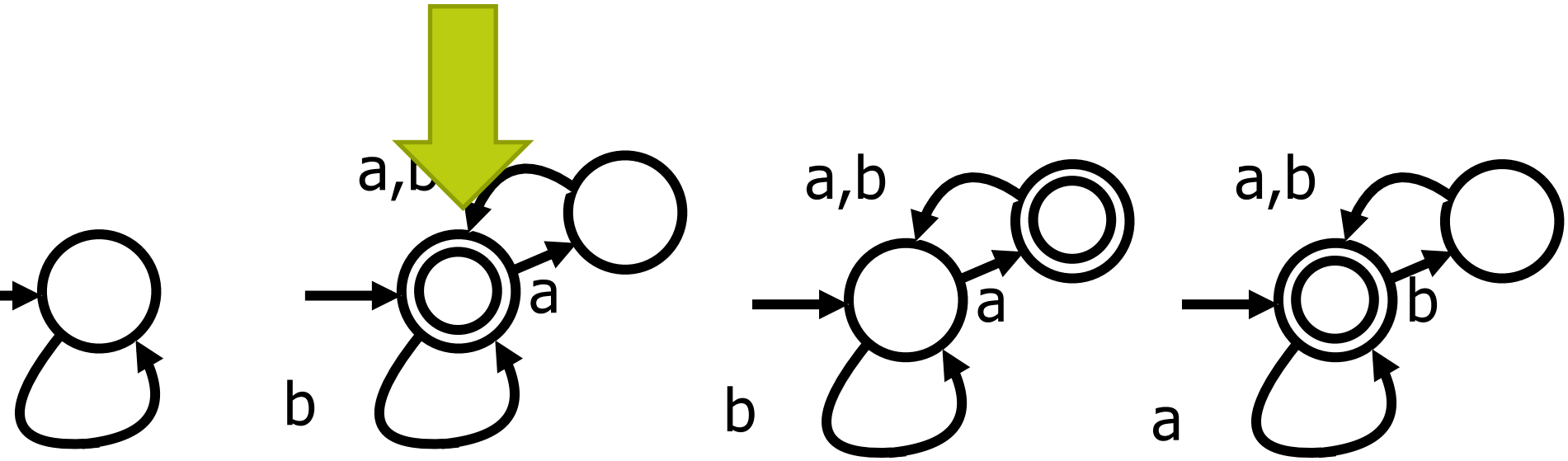
Examples: (a,+)(aa,+)(b,-)

Learning by enumeration



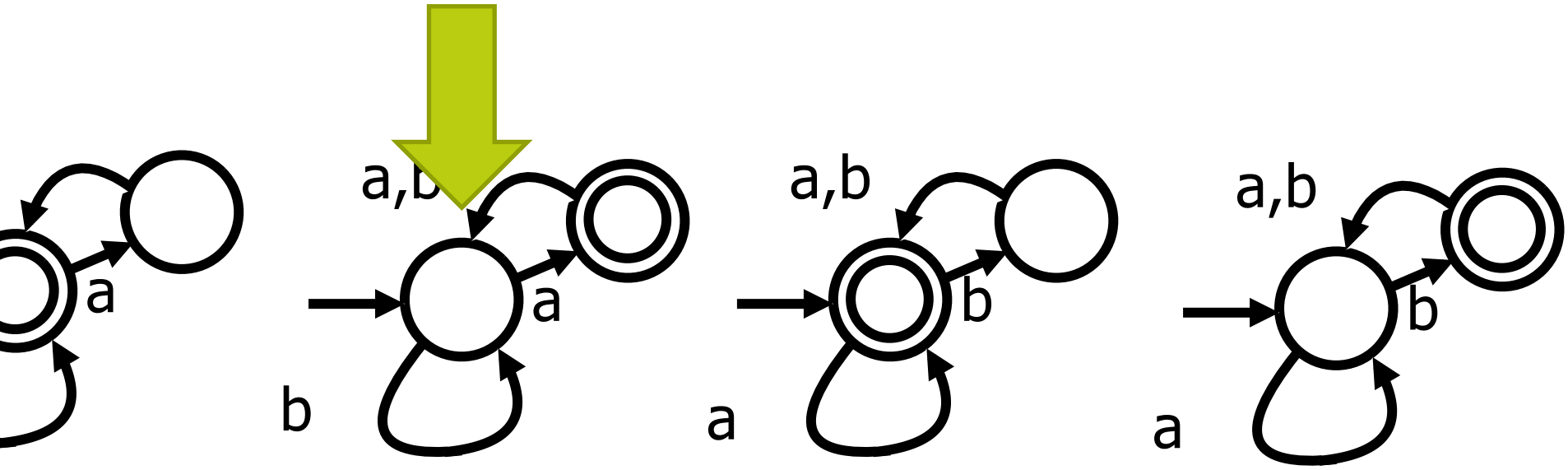
Examples: (a,+)(aa,+)(b,-)

Learning by enumeration



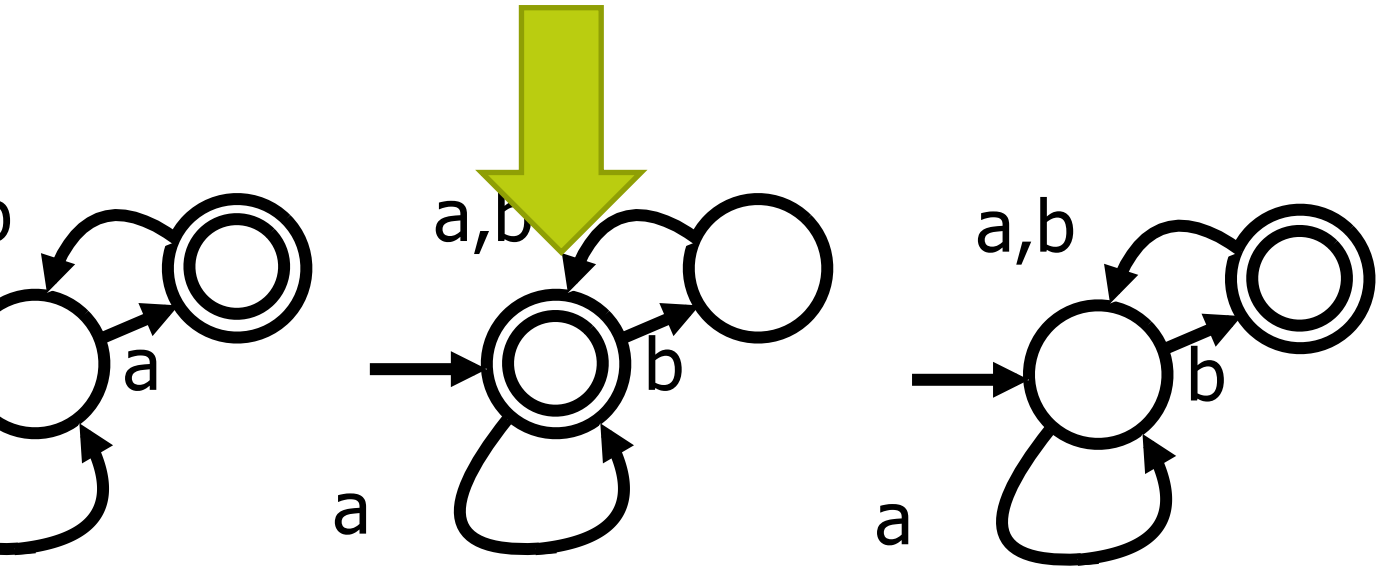
Examples: (a,+)(aa,+)(b,-)

Learning by enumeration



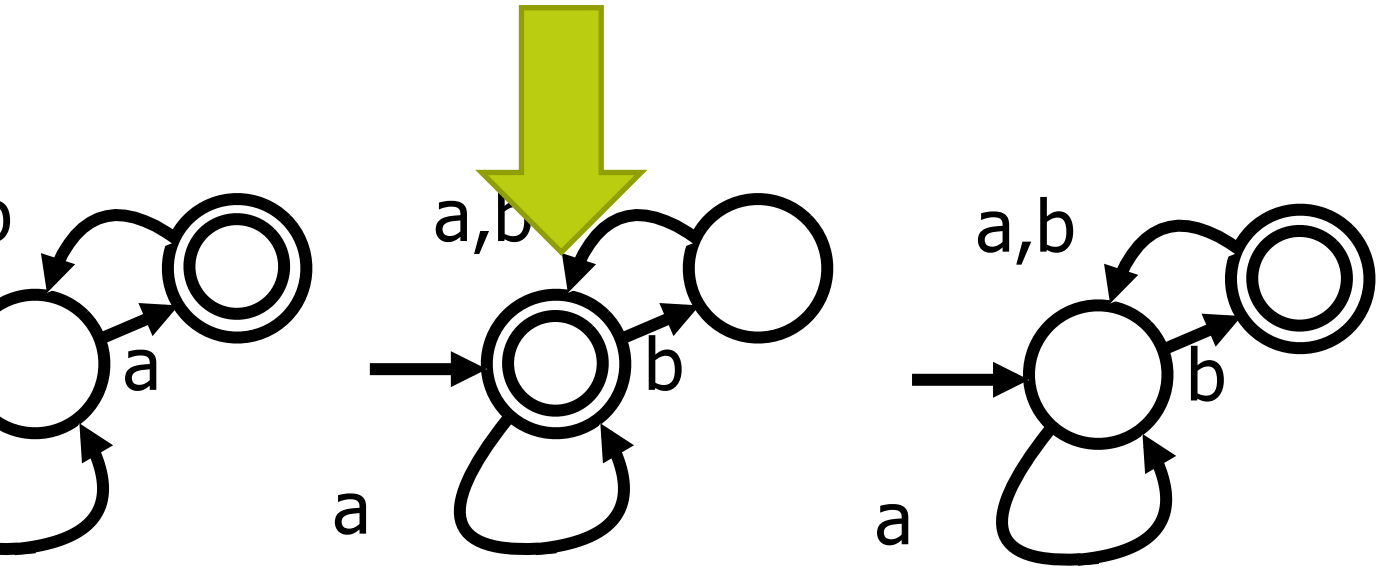
Examples: (a,+)(aa,+)(b,-)

Learning by enumeration



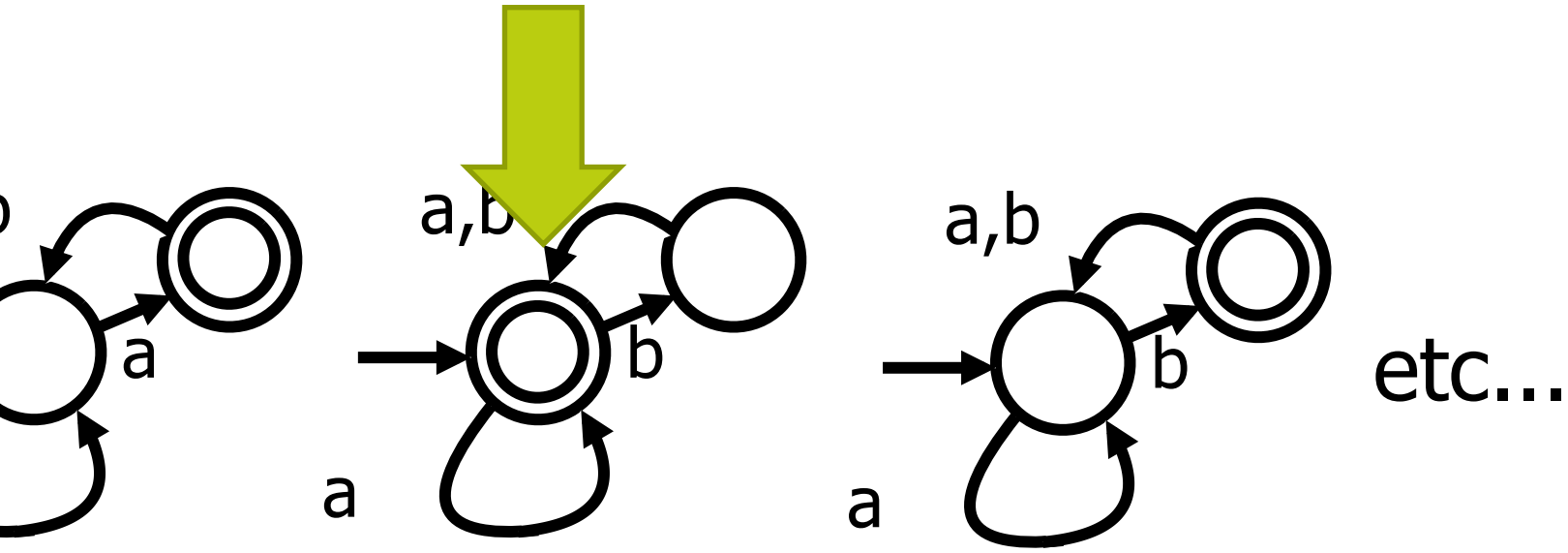
Examples: (a,+)(aa,+)(b,-)

Learning by enumeration



Examples: (a,+)(aa,+)(b,-)(bbaa,+)

Learning by enumeration



Examples: (a,+)(aa,+)(b,-)(bbaa,+)(bb,-)

Learning DFAs

- DFAs are learnable in the limit from labeled data
  - by enumeration using Occam's razor, smallest DFA first
- Are DFAs learnable in the limit from unlabeled data?
  - try learning {a,b}* (all strings) and all finite languages over {a,b}

Learning DFAs

- DFAs cannot be learned in the limit from unlabeled data!
  - Limit learners should at some point converge to C
  - {a,b}* contains all finite languages over {a,b}
  - Suppose a learner converges to {a,b}* at some point
  - At that point, the learner has seen a finite data set D
  - D is a finite language
  - The learner can never converge to D
  - Contradiction

Learning DFAs from labeled data

- Input:
  - Labeled data, positive and negative traces
- Goal:
  - Find the smallest DFA that is consistent with the data

- Learnable in the limit, but NP-complete
  - proof from graph coloring

- Solutions:
  - Rely on data to be nice
  - Learn actively, by asking questions to a teacher (oracle)
  - Assume statistical properties, e.g., a Markov property
  - Translate the problem and use powerful solvers

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.

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

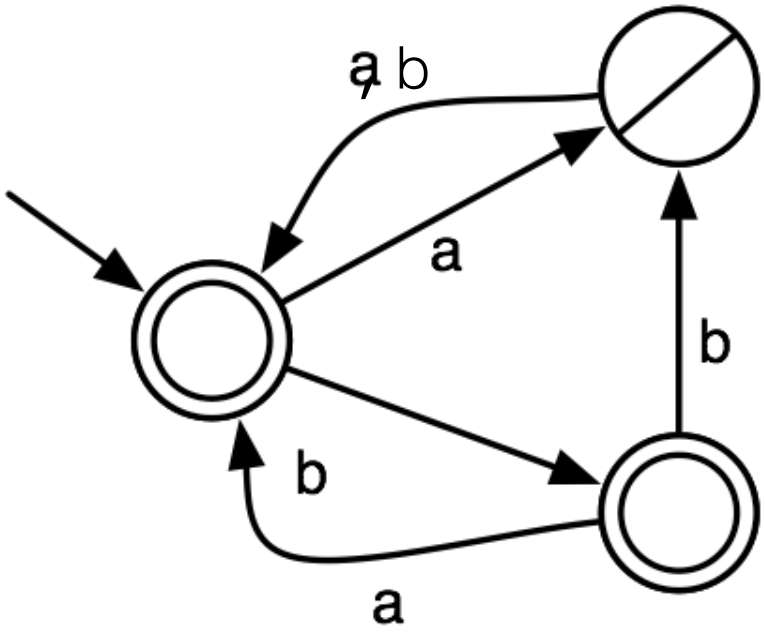
- Given a Hankel matrix for L
- What is the smallest state machine (DFA) for L?

Q: what does this state machine look like?

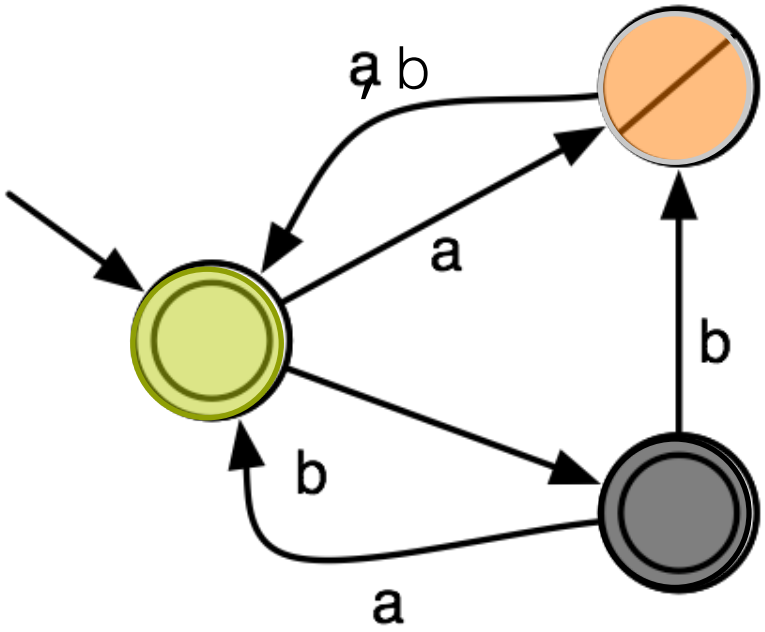
	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	1	0	1	1	1	1	0	0	1	0
a	0	1	1	0	1	0	1	1	1	1
b	1	1	0	0	1	1	1	1	1	1
aa	1	0	1	1	1	1	0	0	1	0
ab	1	0	1	1	1	1	0	0	1	0
ba	1	0	1	1	1	1	0	0	1	0
bb	0	1	1	0	1	0	1	0	1	1
aaa	0	1	1	0	1	0	1	1	1	1
aab	1	1	0	0	1	1	1	1	1	1
aba	0	1	1	0	1	0	1	1	1	1

	_	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	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

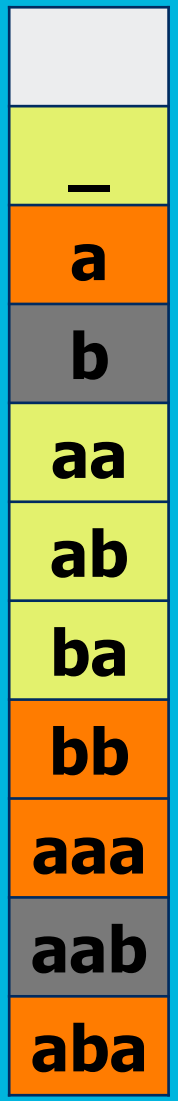
Answer



Answer



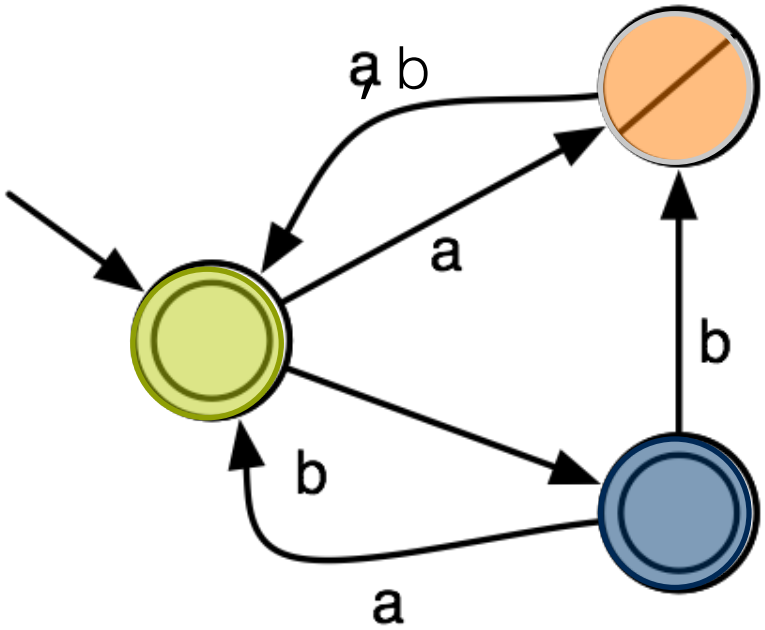
	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	1	0	1	1	1	1	0	0	1	0
a	0	1	1	0	1	0	1	1	1	1
b	1	1	0	0	1	1	1	1	1	1
aa	1	0	1	1	1	1	0	0	1	0
ab	1	0	1	1	1	1	0	0	1	0
ba	1	0	1	1	1	1	0	0	1	0
bb	0	1	1	0	1	0	1	0	1	1
aaa	0	1	1	0	1	0	1	1	1	1
aab	1	1	0	0	1	1	1	1	1	1
aba	0	1	1	0	1	0	1	1	1	1



Look at colors of Hankel Matrix rows

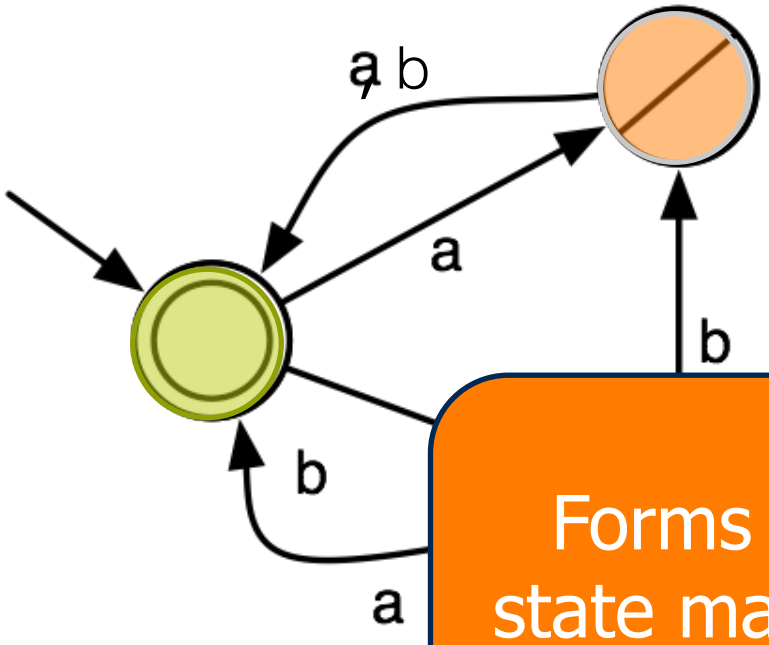
_
a
b
aa
ab
ba
bb

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

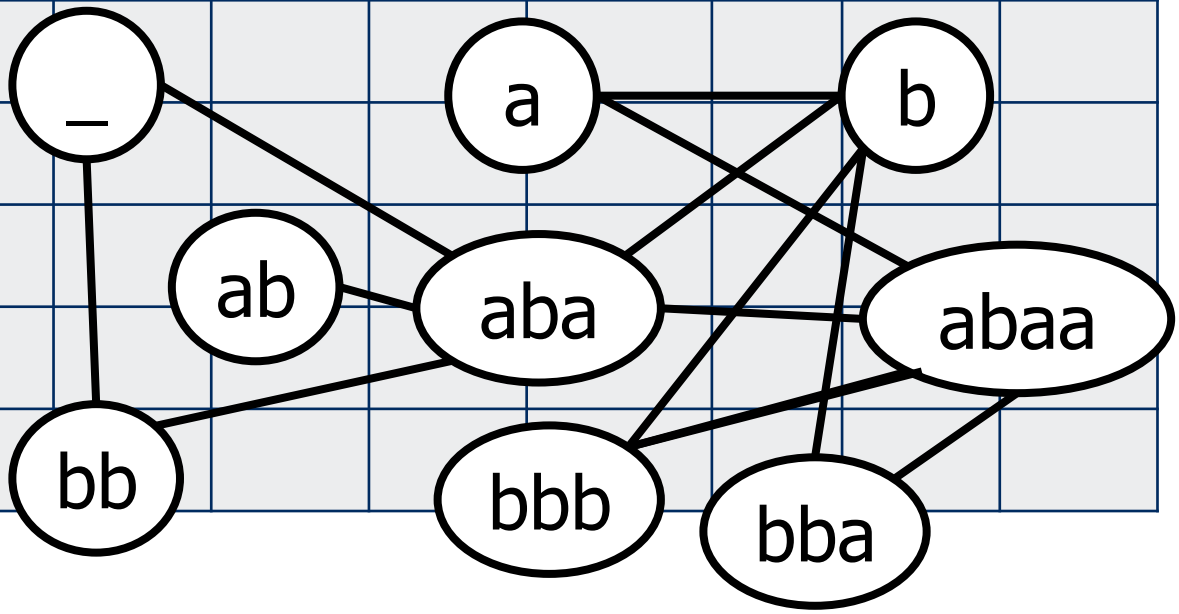
STATE MACHINE LEARNING

(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,+)

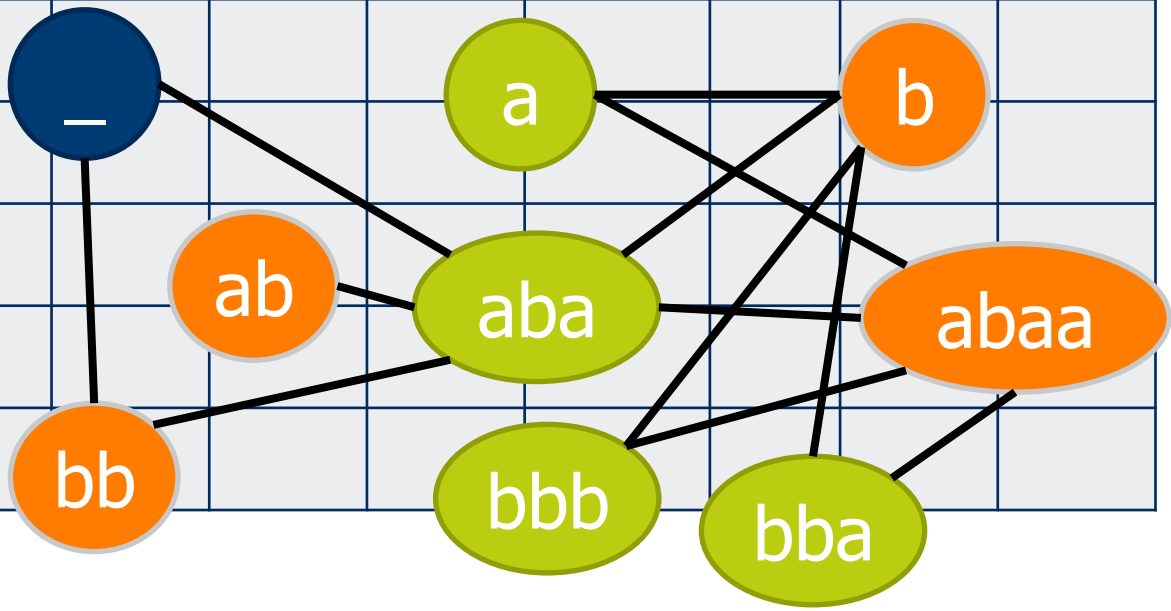
	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0			1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										



TUDelft

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

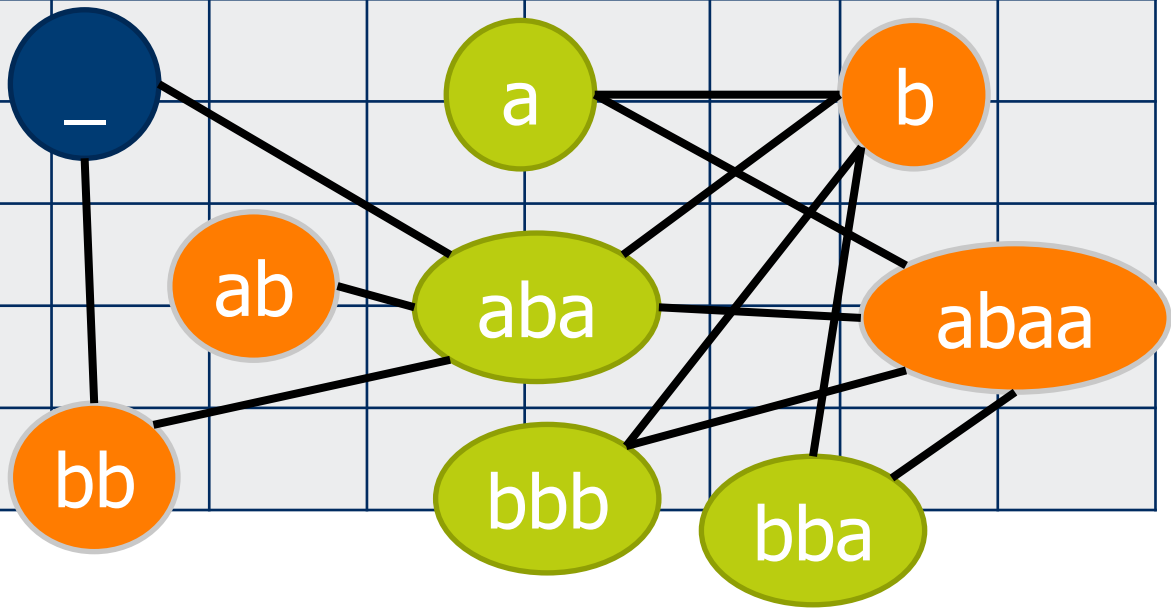
	_	a	b	ba	aba	aa	baa	abaa	bb	bbb	bba
_		1	0		1			0		1	1
a	1			1			0				
b	0			1					1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										



(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...

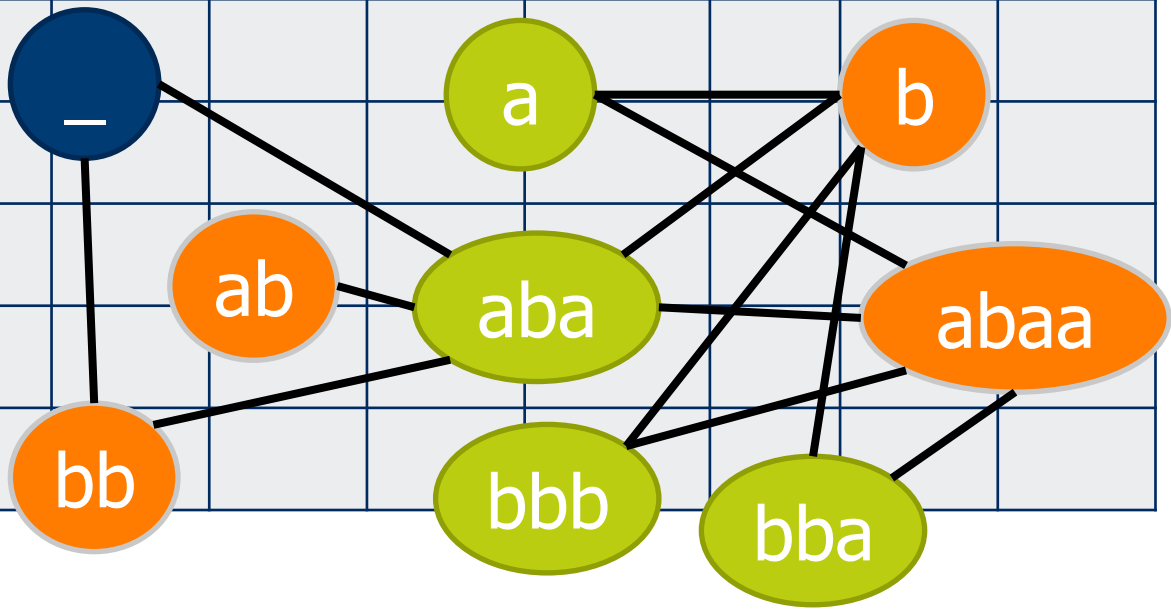


TU Delft

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

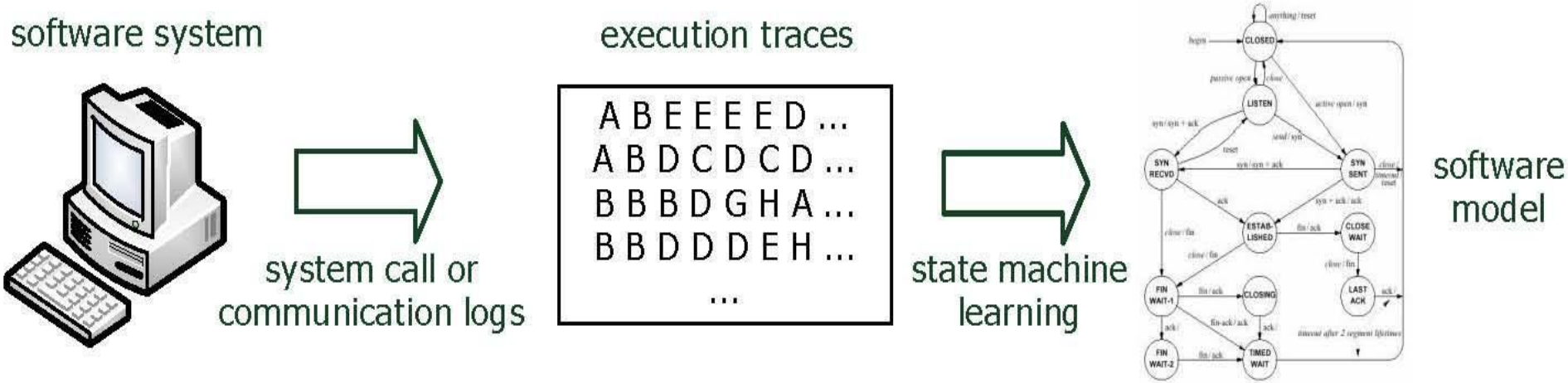
						aa	baa	abaa	bb	bbb	bba
								0		1	1
							0				
									1		
ab		1				0					
aba	1	0									
abaa	0										
bb		1	1								
bbb	1										
bba	1										

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

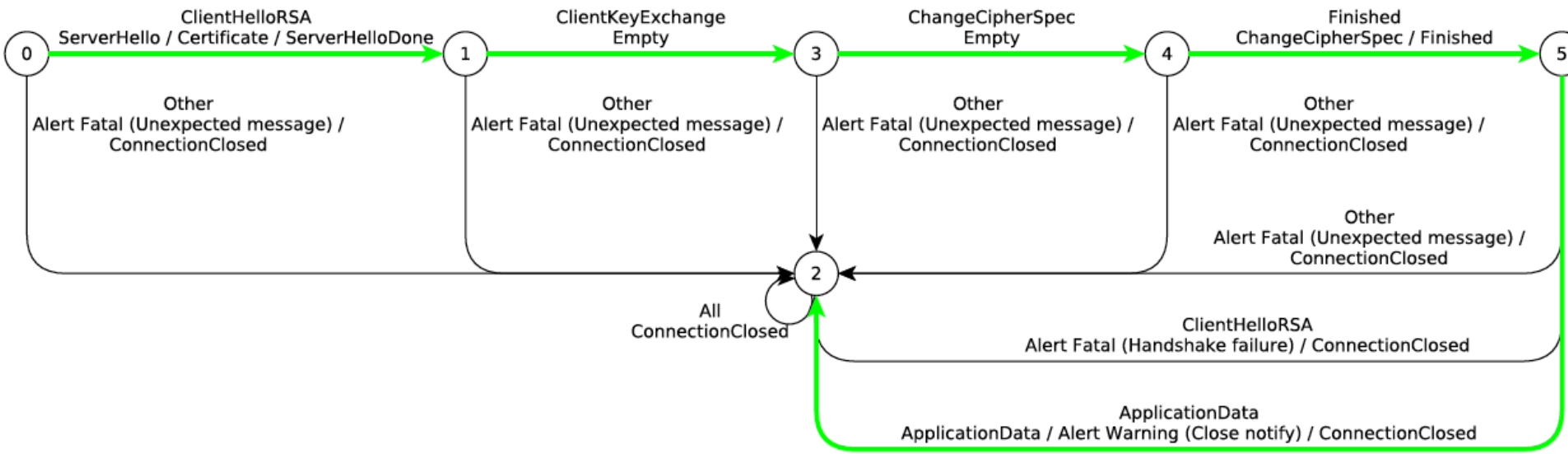
L* - some theory

- 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

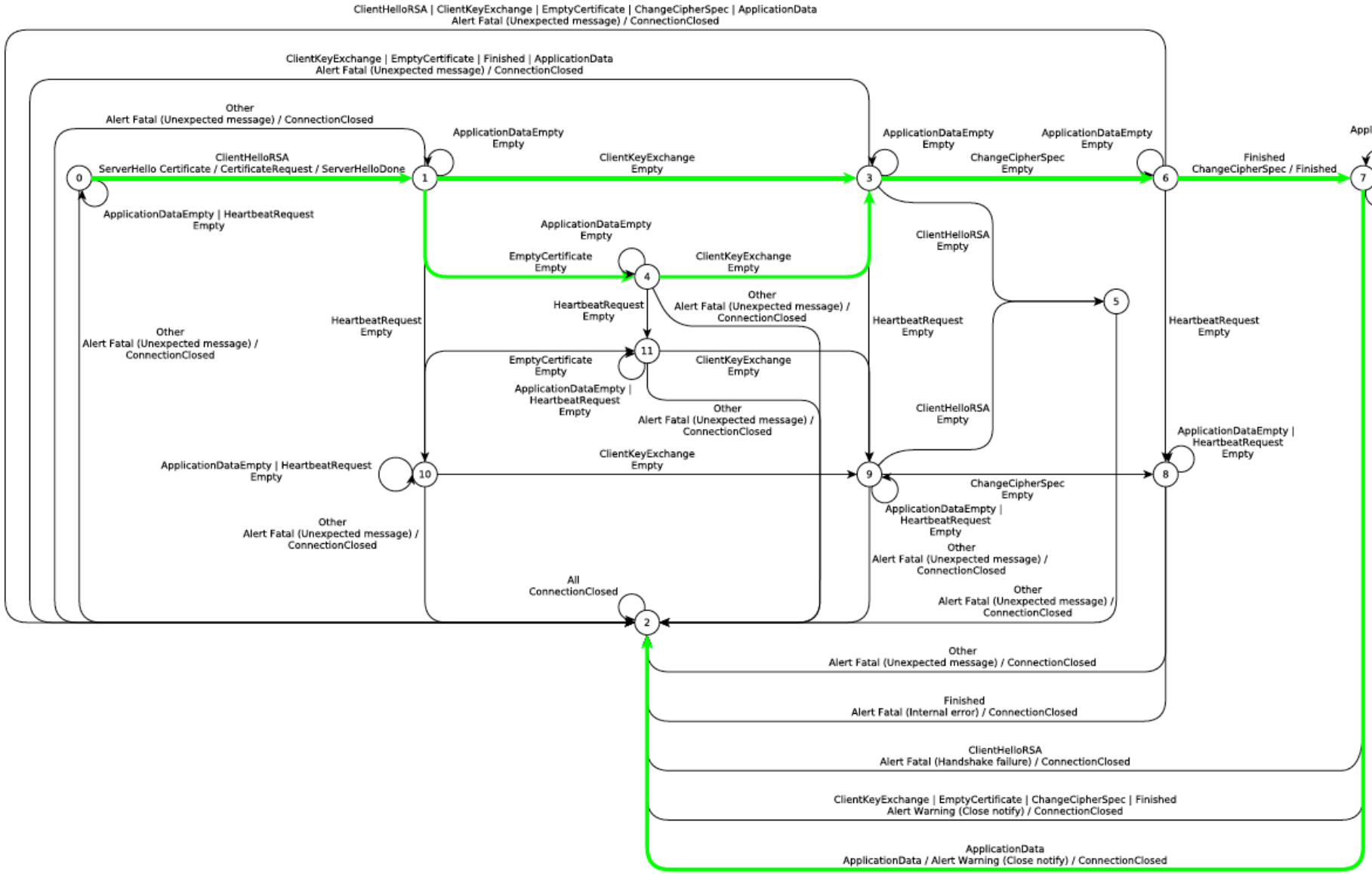
- Learnlib is an L* implementation that can be applied to software
  - Uses model-based testing methods for answering equivalence queries
  - We are currently combining this with fuzzing for security purposes
- The learned models allow to:
  - Investigate the model for any implementation flaws
  - Compare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

TLS RSA BSAFE



TUDelft

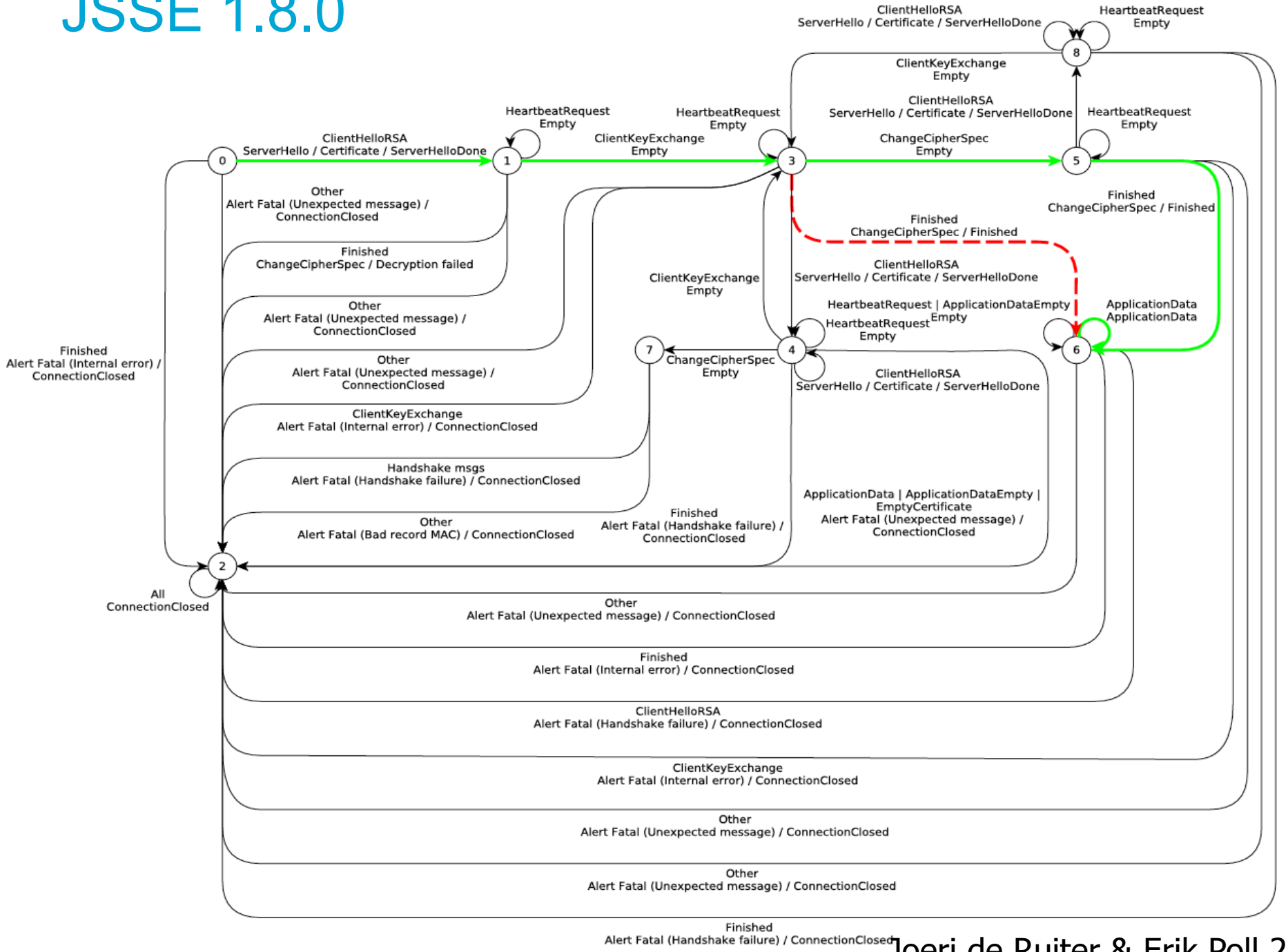
GNU TLS 3.3.8



TUDelft

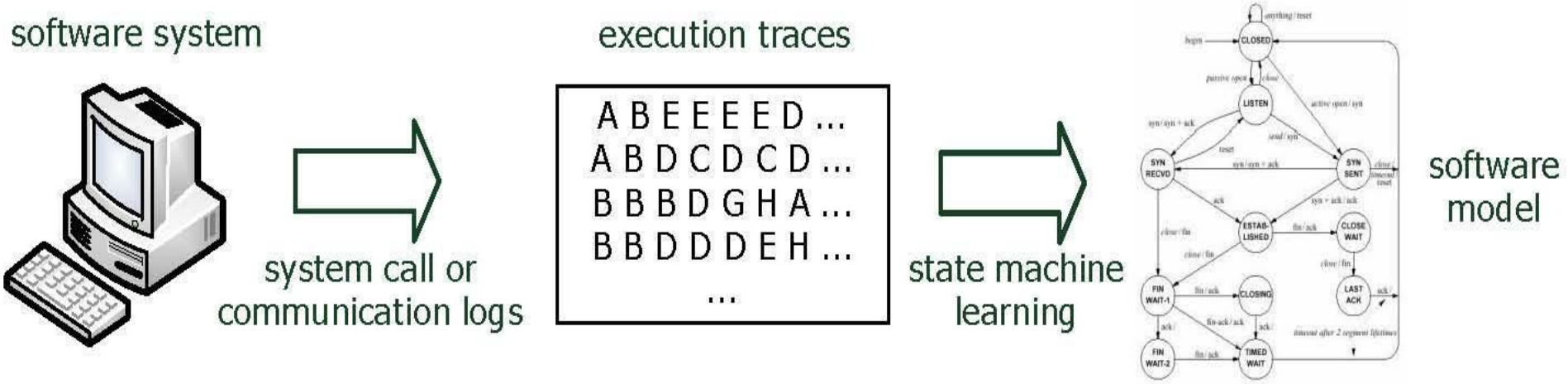
Joeri de Ruiter & Erik Poll 201

JSSE 1.8.0



Joeri de Ruiter & Erik Poll 201

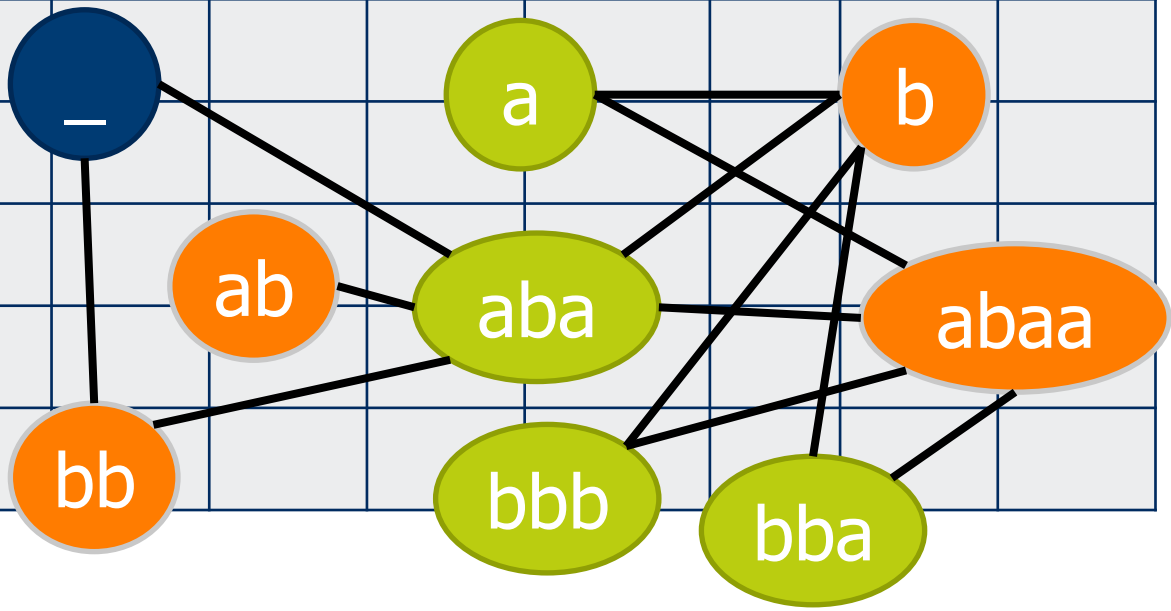
Use the information we have…



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										

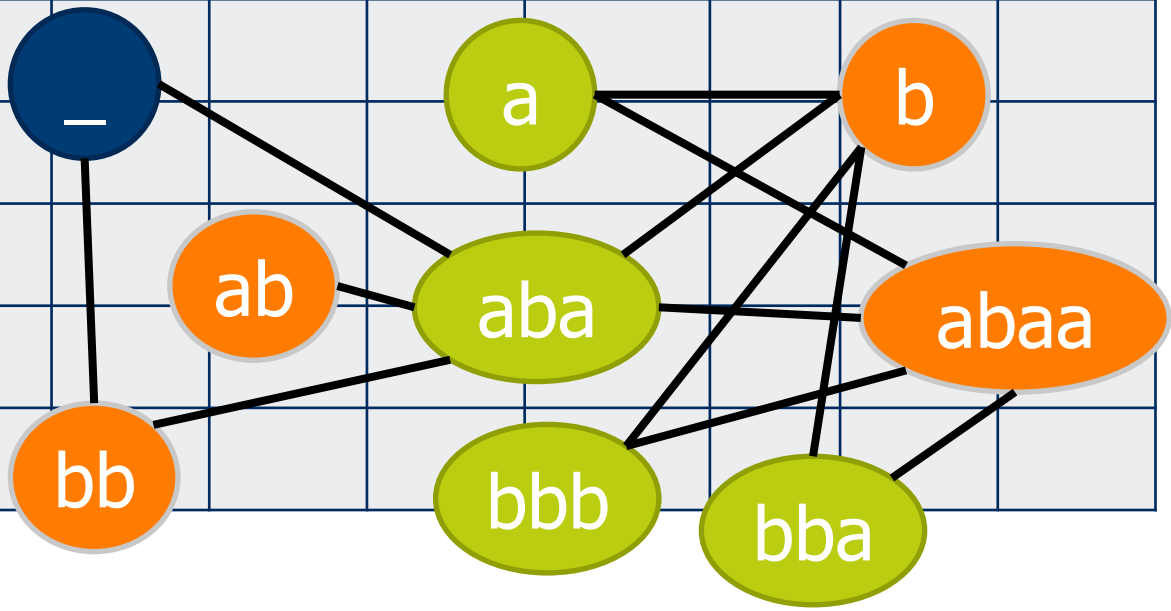


TUDelft

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…

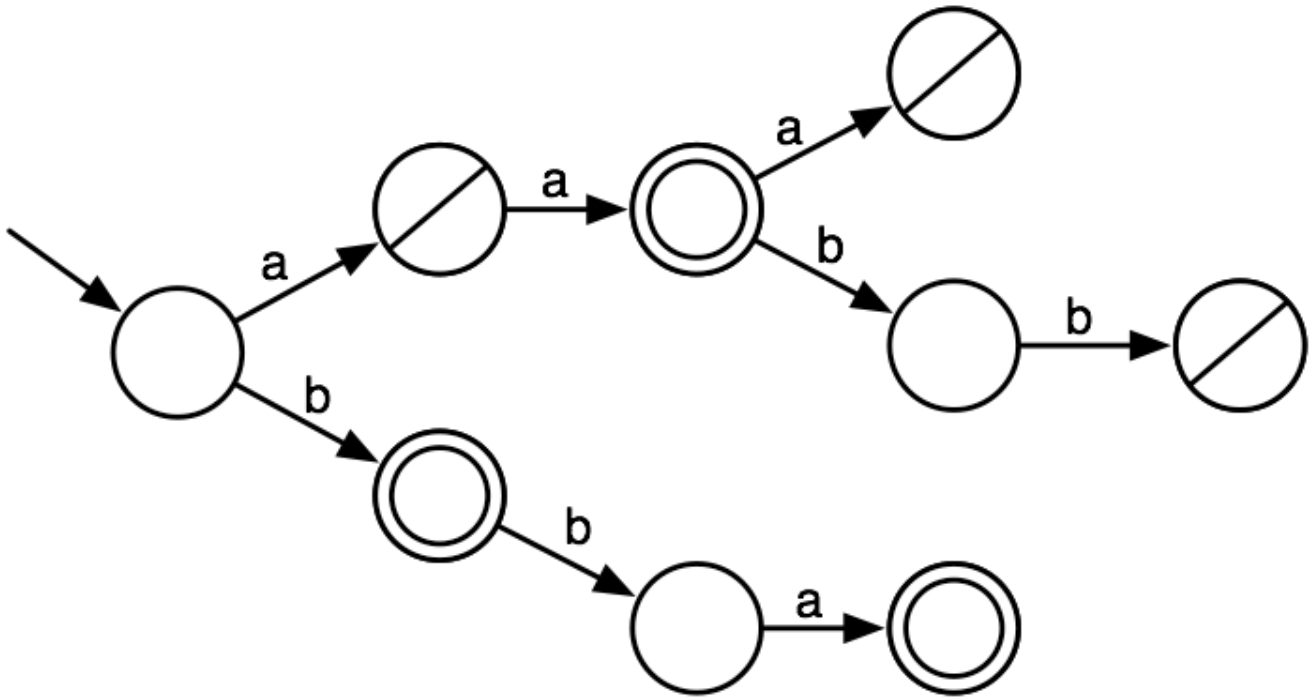


TUDelft

Solution: state merging

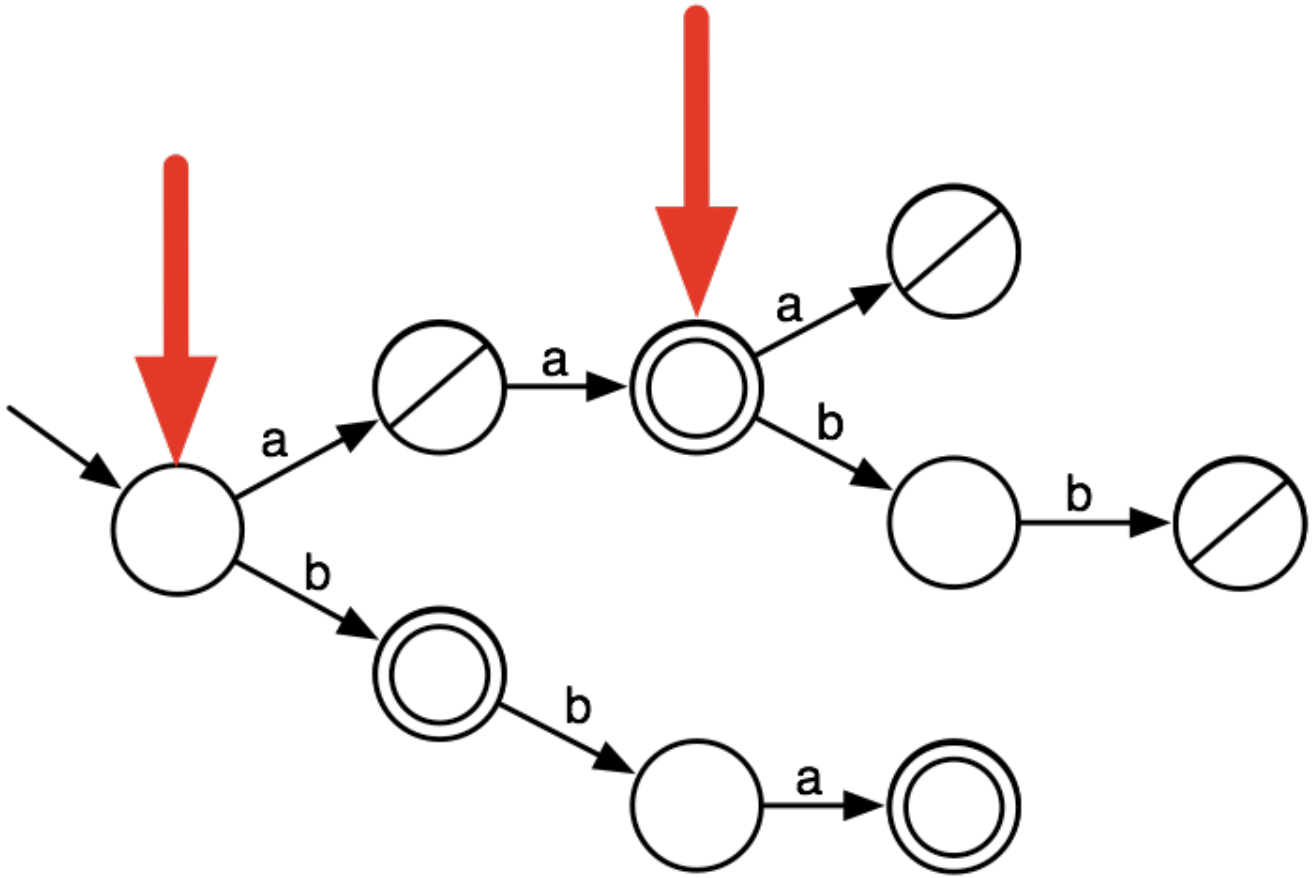
- Very effective greedy method
  - Initially assume a large DFA
  - Iteratively combine similar states
  - Until no more state can be combined
- State-of-the-art since 1998

State merging



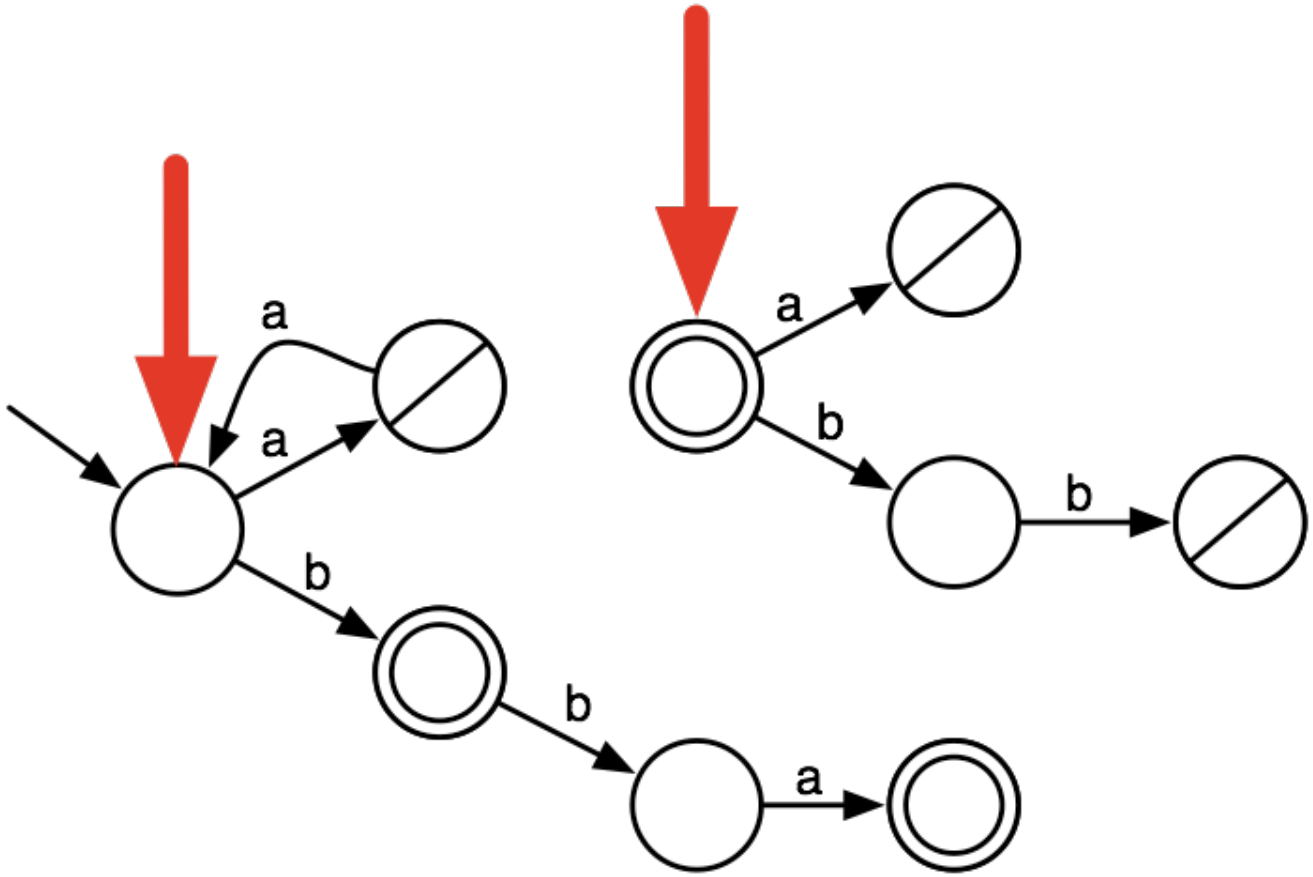
positive data: aa, b, bba; negative data: a, aaa, aabb
represented as a prefix tree

State merging



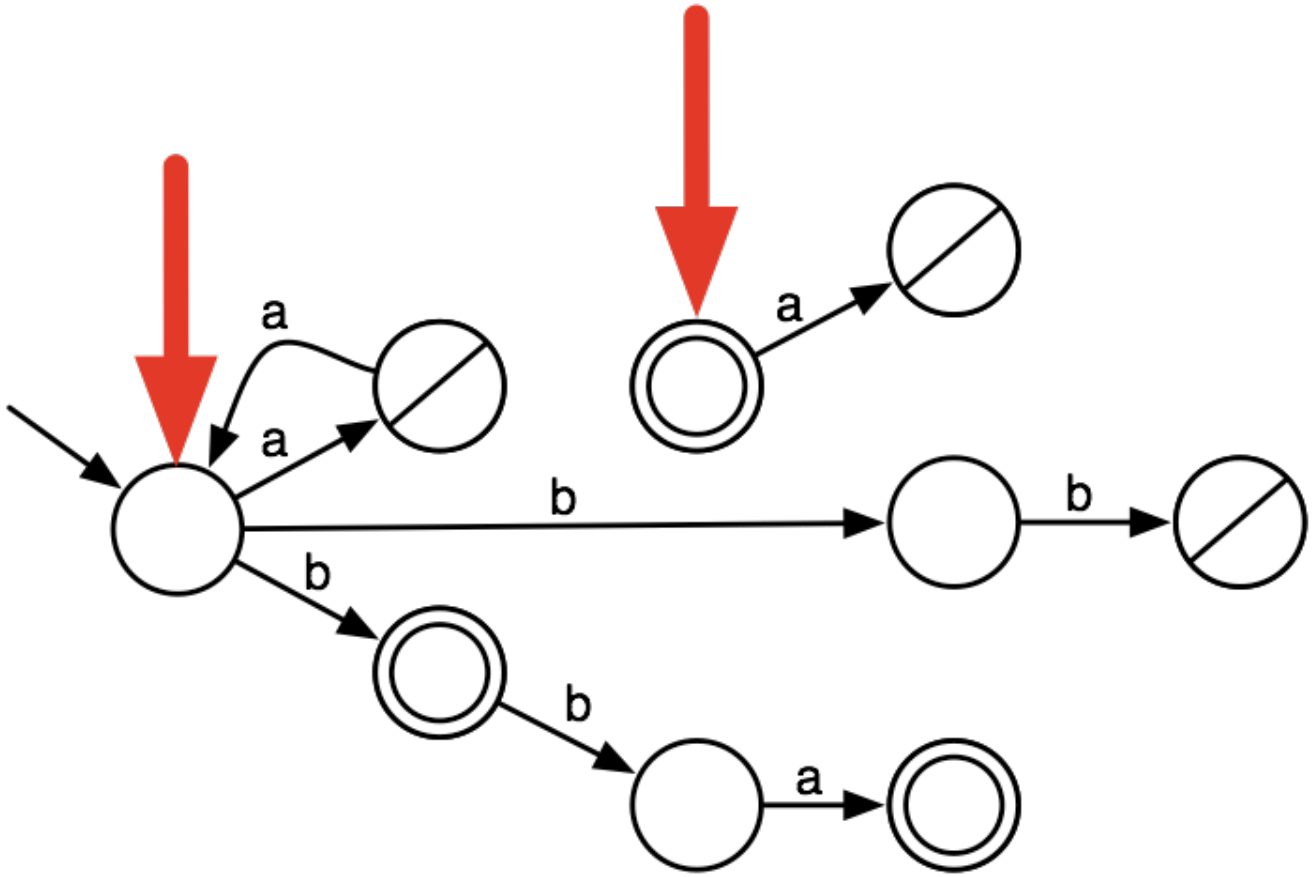
State merging:
select two nodes

State merging



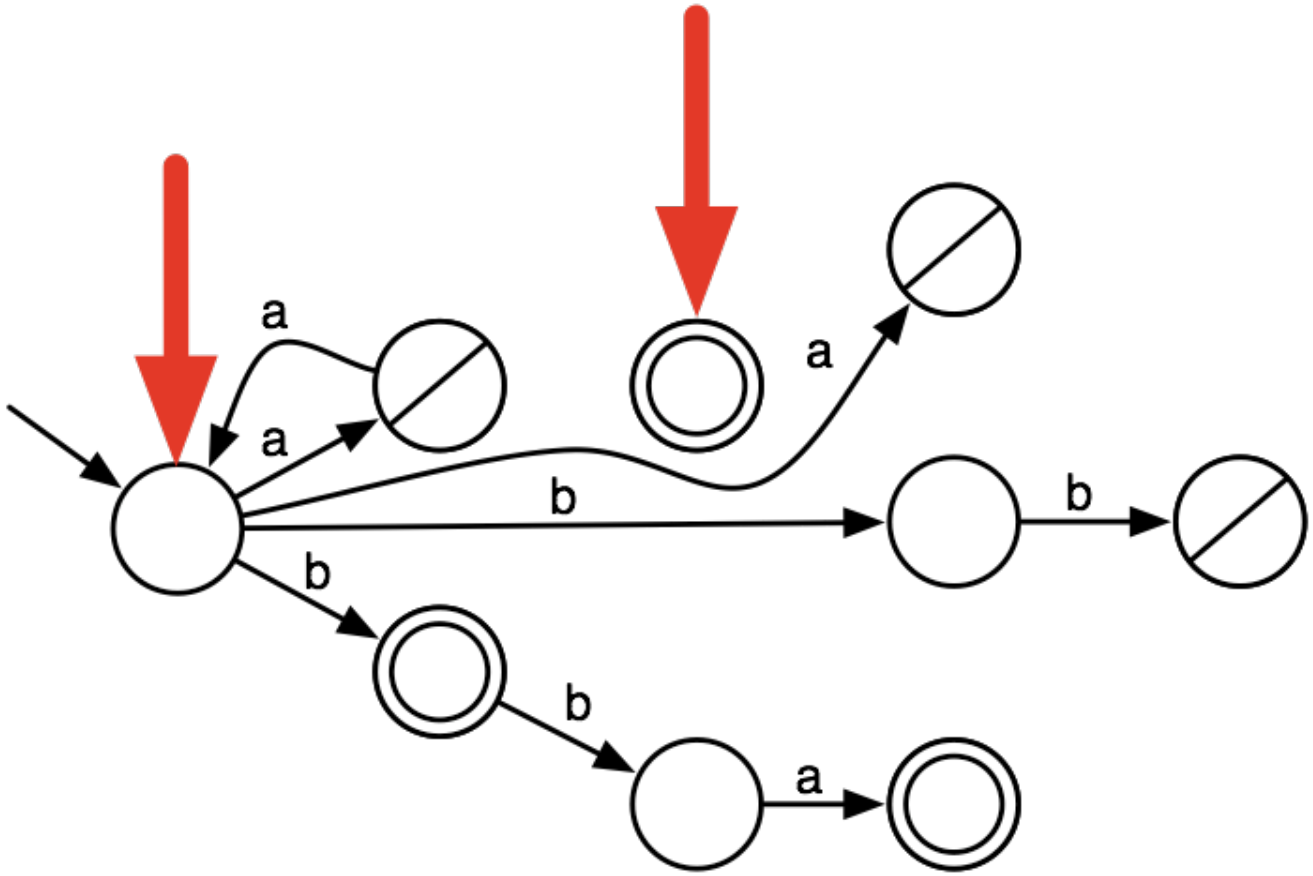
State merging:
move input transitions from one state to the other

State merging



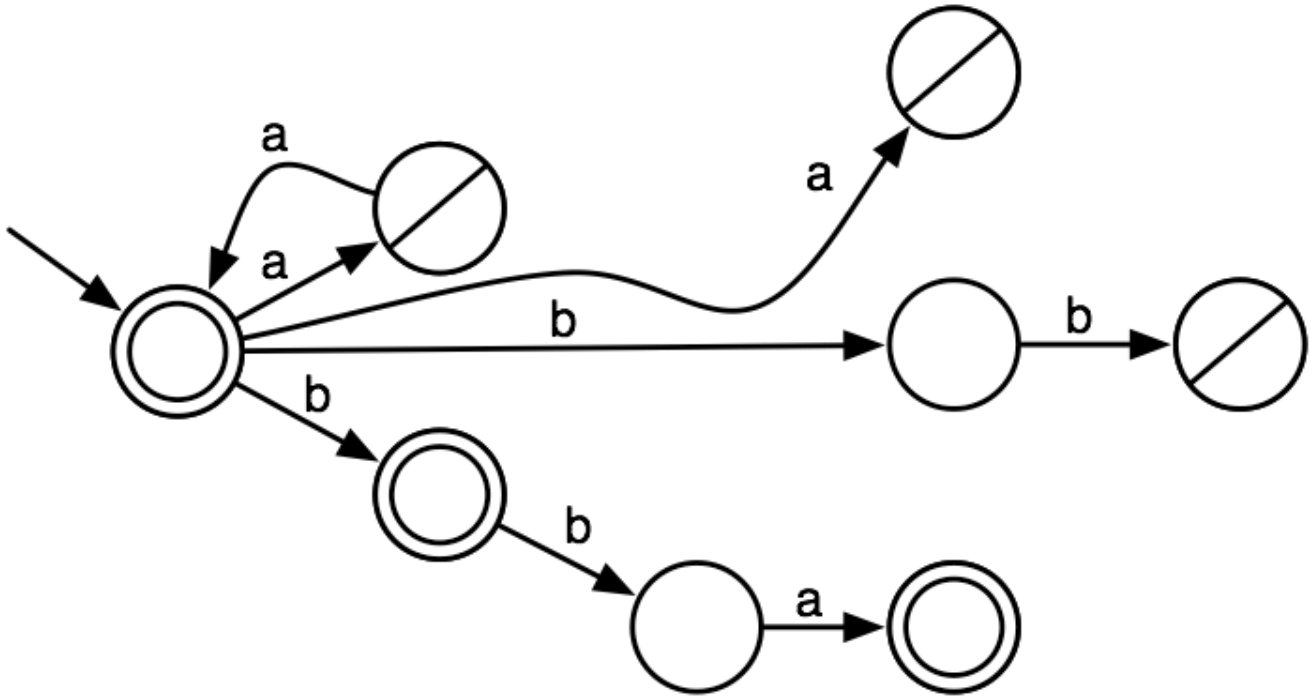
State merging:
move output transitions from one state to the other

State merging



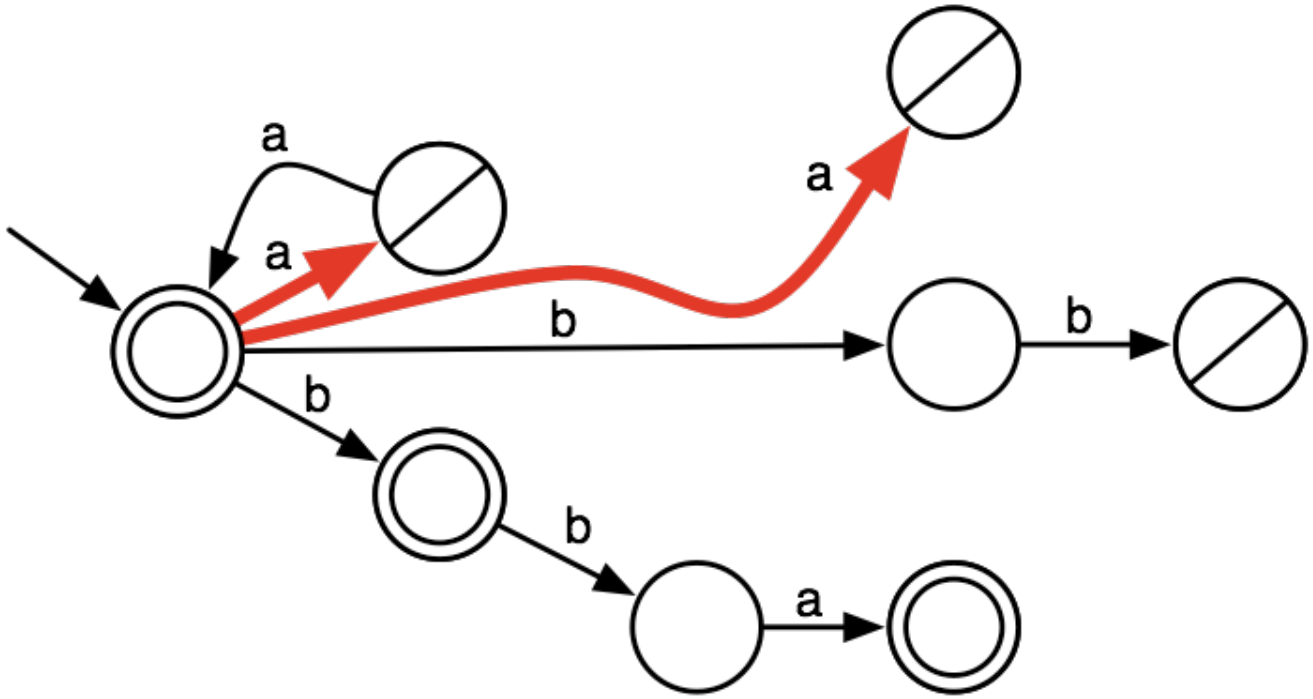
State merging:
move output transitions from one state to the other

State merging

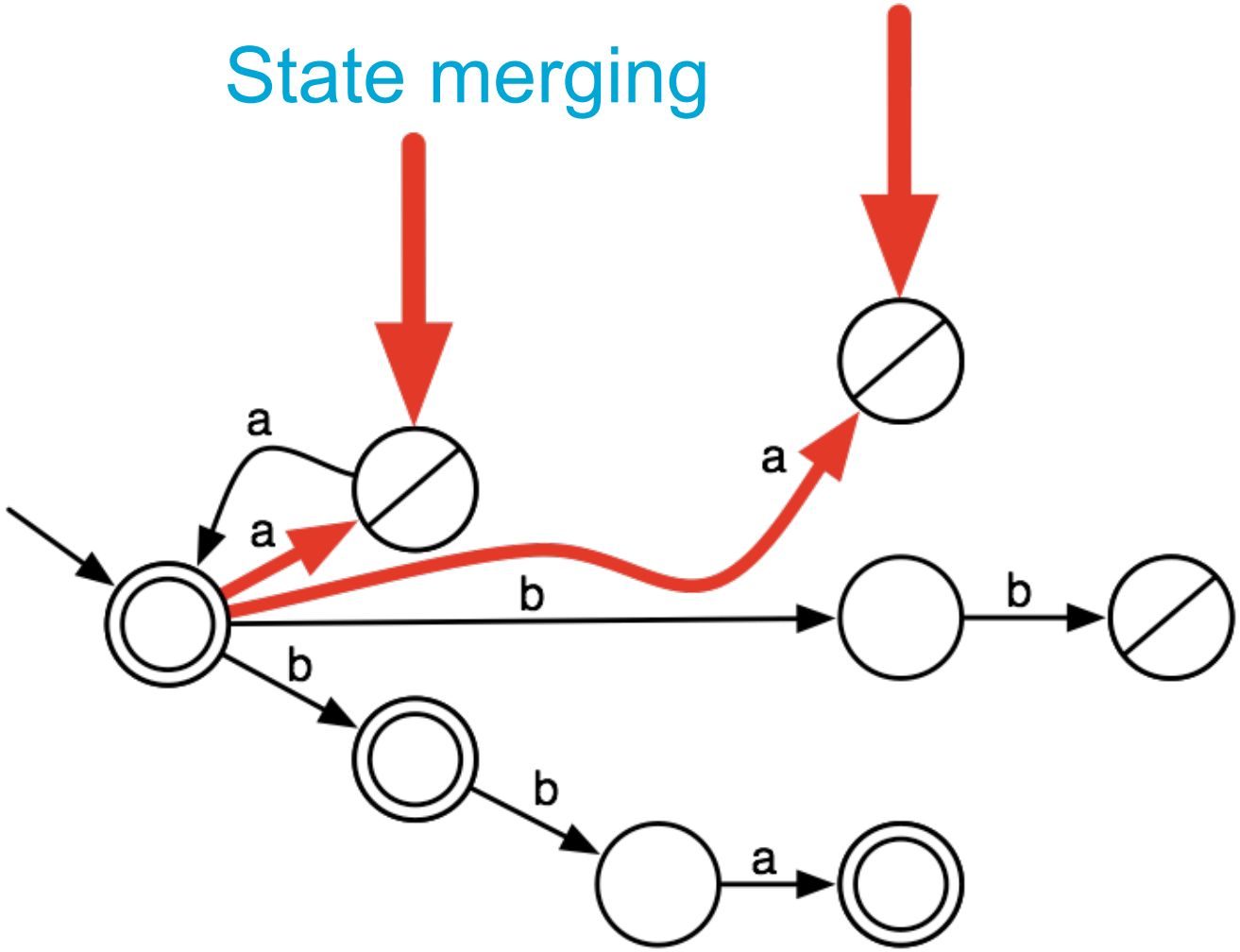


State merging:
delete the obsolete state, maintain positive/negative

State merging

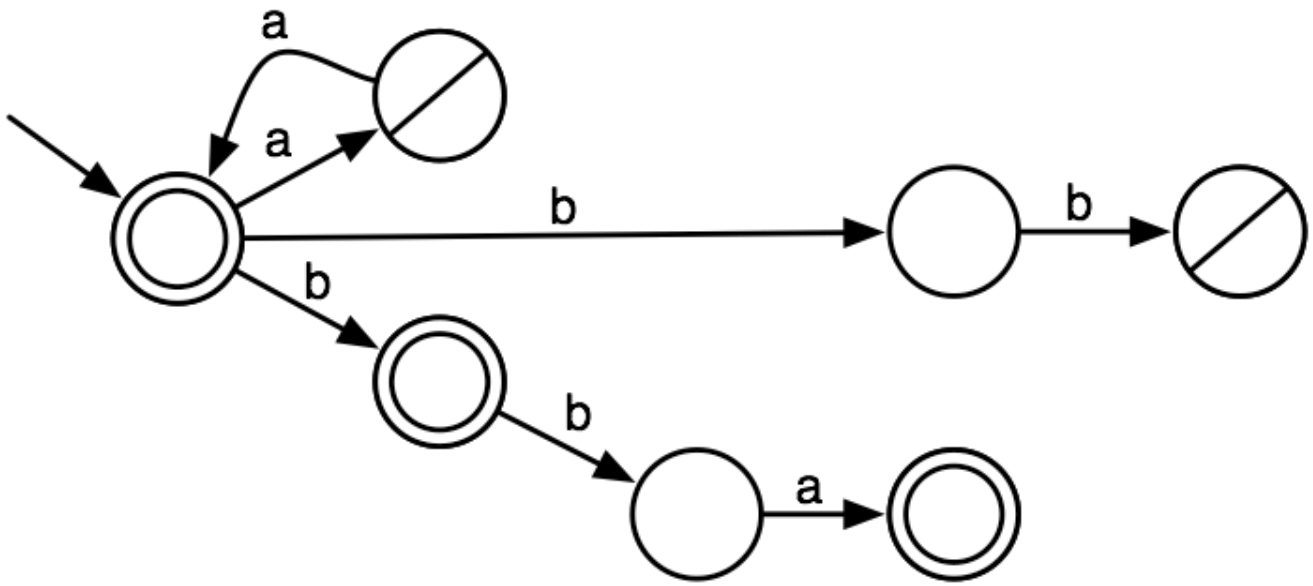


Determinization:
merge the targets of non-deterministic transitions



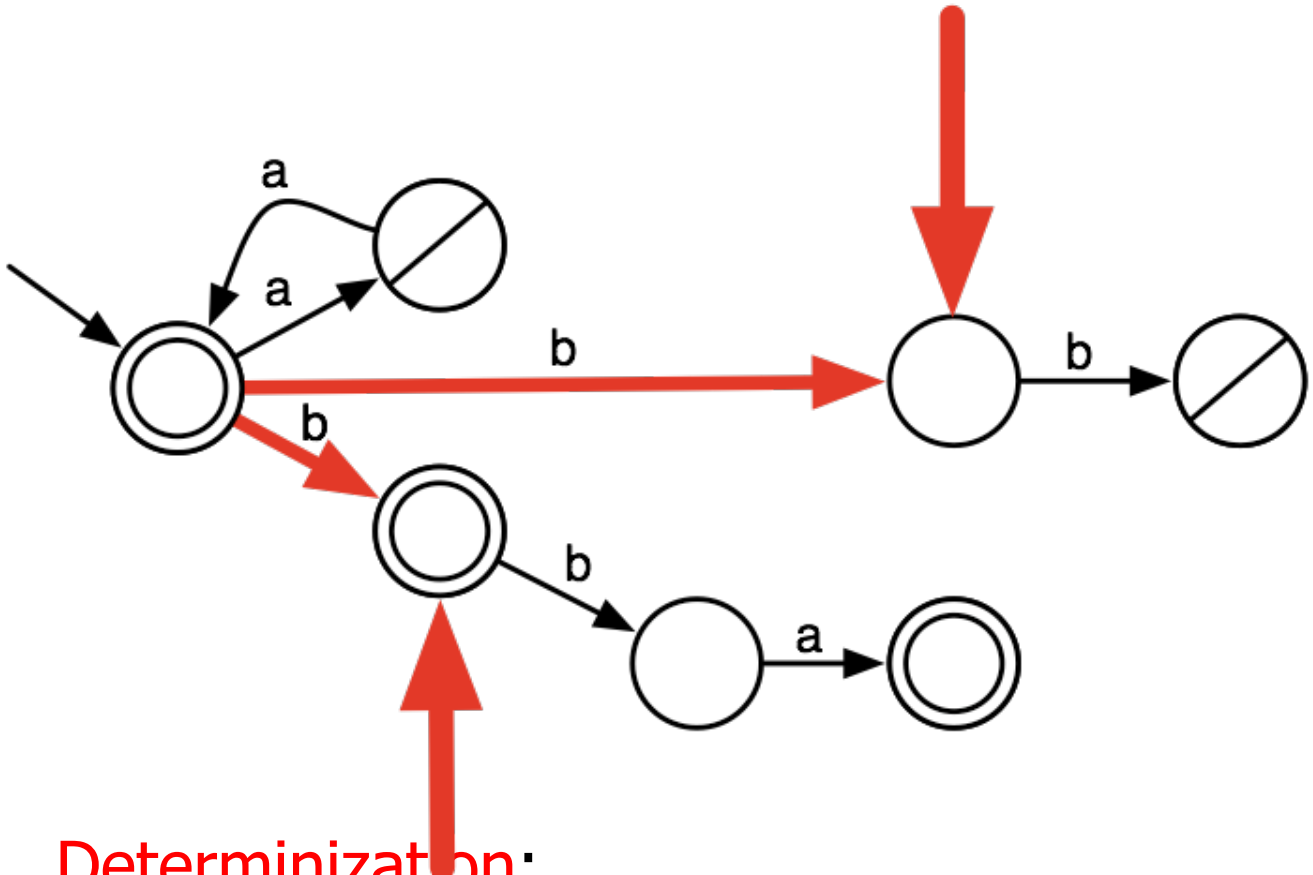
Determinization:
merge the targets of non-deterministic transitions

State merging



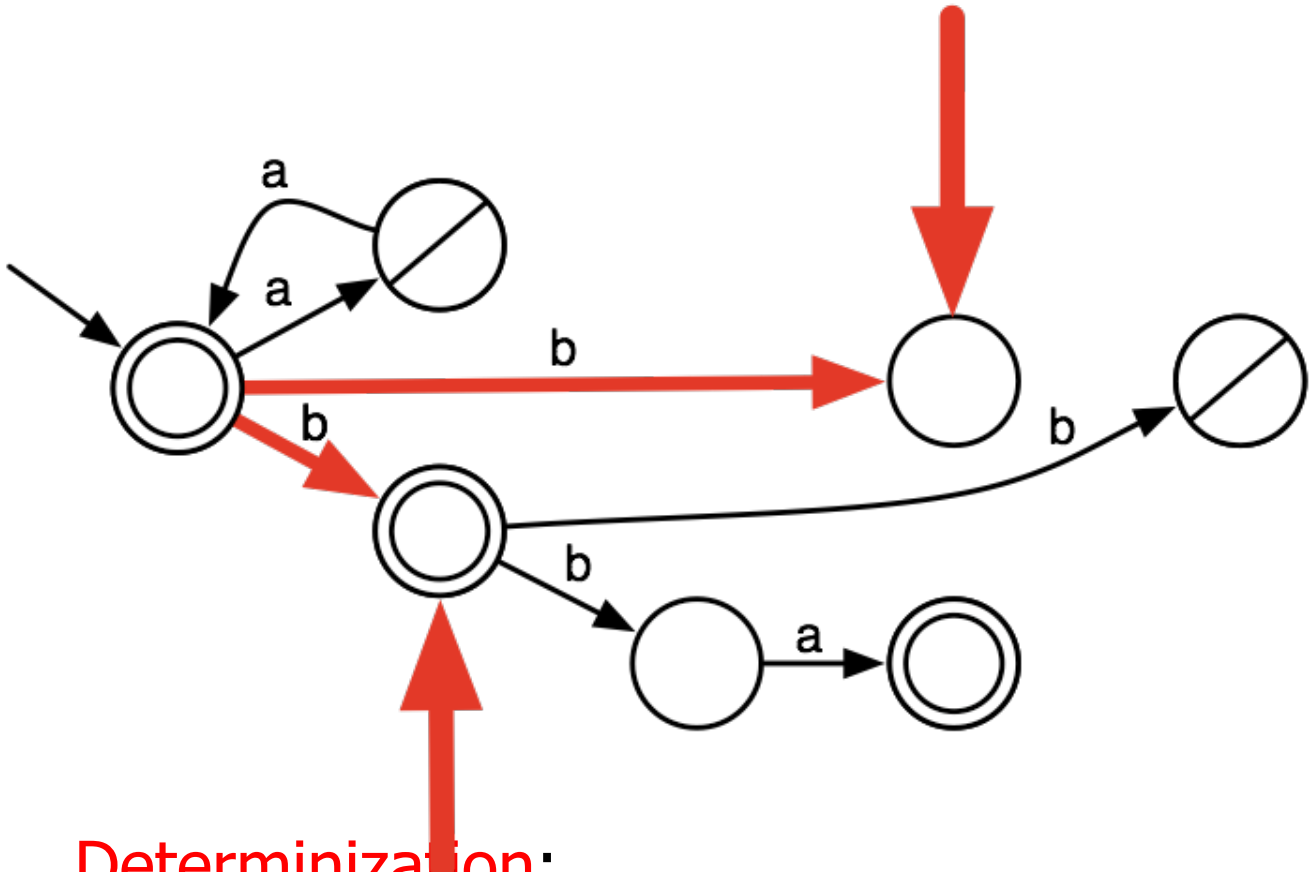
Determinization:
merge the targets of non-deterministic transitions

State merging



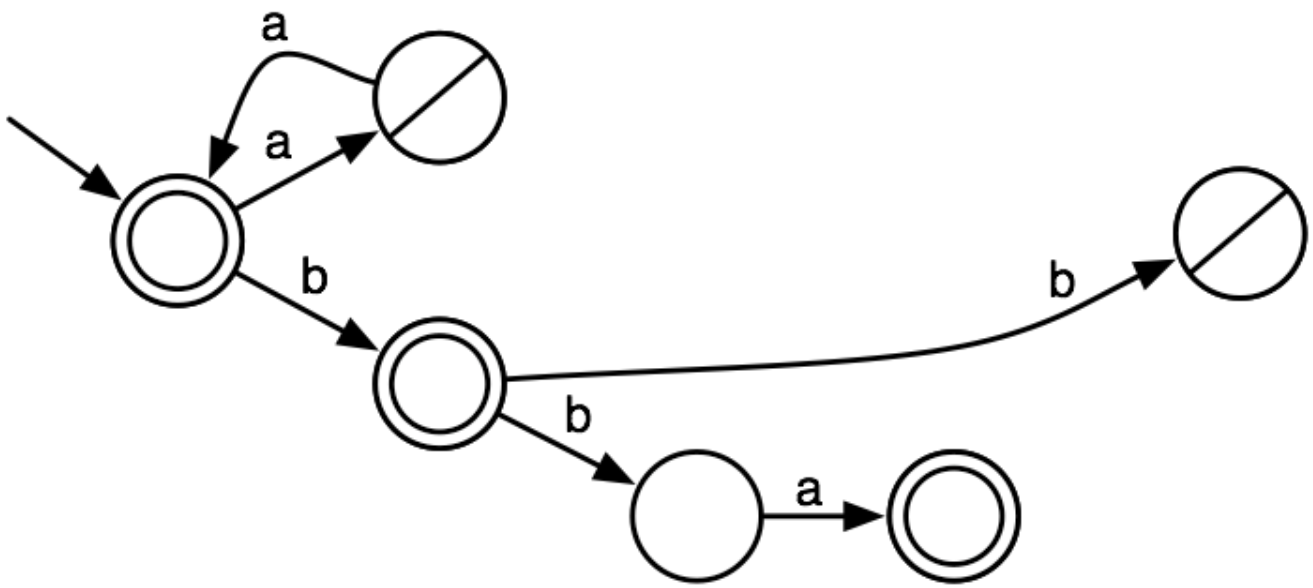
Determinization:
merge the targets of non-deterministic transitions

State merging



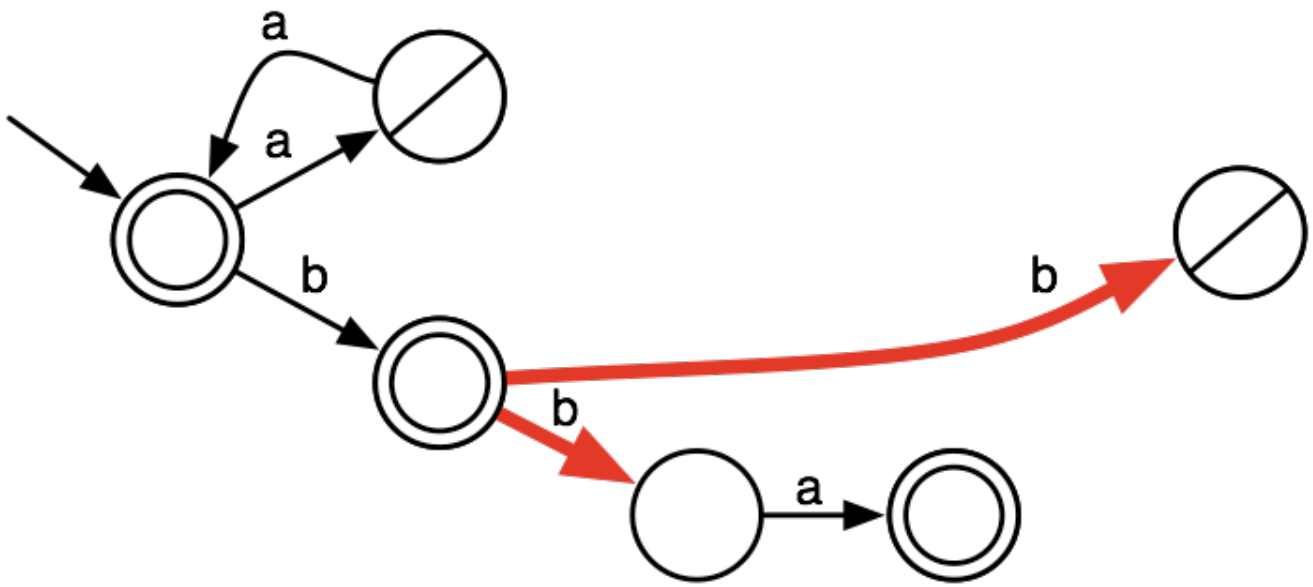
Determinization:
merge the targets of non-deterministic transitions

State merging



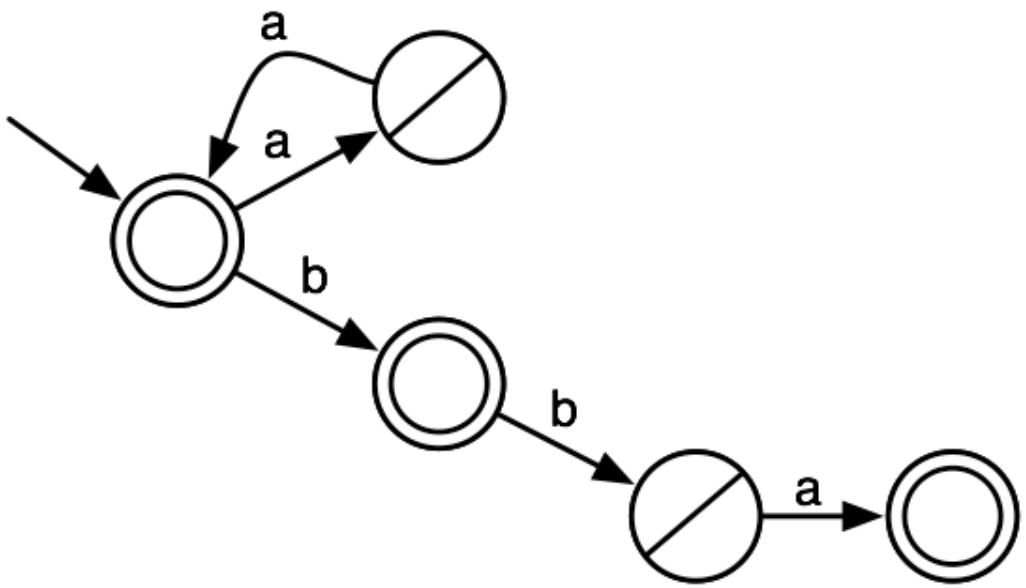
Determinization:
merge the targets of non-deterministic transitions

State merging



Select two new nodes to merge and iterate

State merging



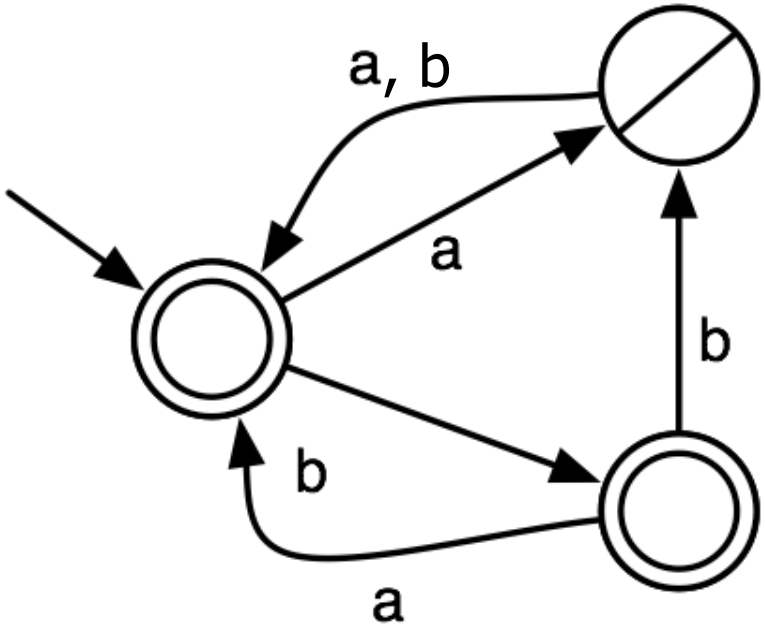
Select two new nodes to merge and iterate

State merging (RPNI)

1. Start from a prefix tree acceptor
2. Color the initial state red, and it's children blue
3. Let B be the first blue state in lexicographical order (a,b,aa,ab,ba,bb,aaa,aab,aba,abb,…)
4. For all red states R in lexicographical order
    1. Merge B with R
    2. Break if consistent, undo the merge otherwise
5. Color B red if it is not merged
6. Color all children of red states blue
7. Goto 3

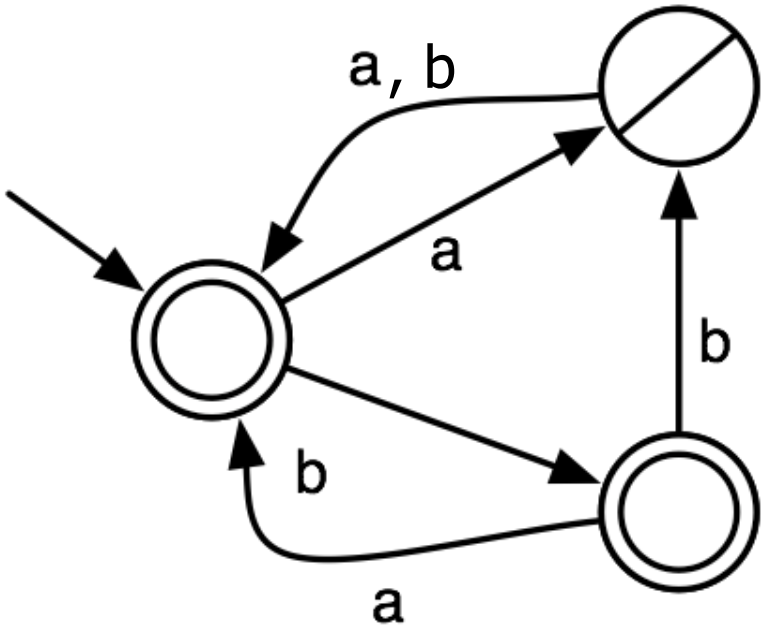
Convergence

- How much data does RPNI need to learn
- ???



Convergence

- How much data does RPNI need to learn
- ???

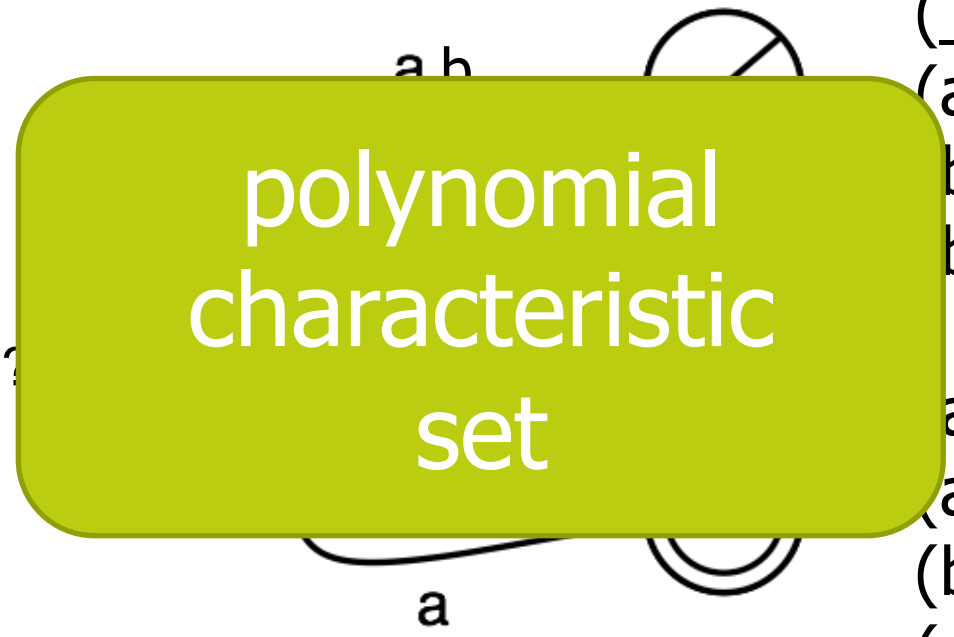


(_,+)
(a,-)
(b,+)
(ba,+)

(aa,+)
(ab,+)
(bb,-)
(aaa,-)
(aba,-)
(baa,-)

Convergence

- How much data does RPNI need to learn
- ???



polynomial characteristic set

(_,+)
(a,-)
(b,+)
(ba,+)
(aa,+)
(ab,+)
(bb,-)
(aaa,-)
(aba,-)
(baa,-)

Results

- DFAs cannot be learned from unlabeled data
- Learning DFAs from labeled data is hard, but
  - In the limit, RPNI requires only a polynomial amount of data to converge
  - If the data source is nice, we only need a polynomial amount of data and a polynomial-time algorithm

State machine learning key ingredients

- When are two states consistent?
  - Positive gets merged with negative
  - Other events are possible
  - Statistics do not match
  - If the Markov property is violated
  - …
- What to use as merge score?
  - Number of positive-positive, negative-negative merges
  - Overlap in sets of possible events
  - Statistical distance
  - …

Passive automaton learning

- Learn a behavioral model from any event sequence
- Use the model as a classifier
- Analyze the model to understand its inner workings

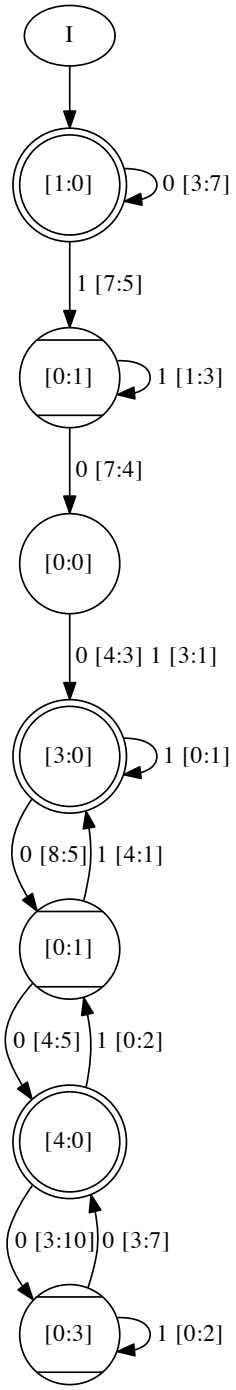
- DFASAT, my own tool to learn these models
  - based on state merging
  - translates part of the learning problem to Satisfiabiliy, and runs SAT-solvers to compute the best DFA
  - winner of the 2010 international state machine learning competition
  - allows many different consistency checks and score functions
    - possible to define your own

  - still needs work…

- Demo…

DFASAT produces

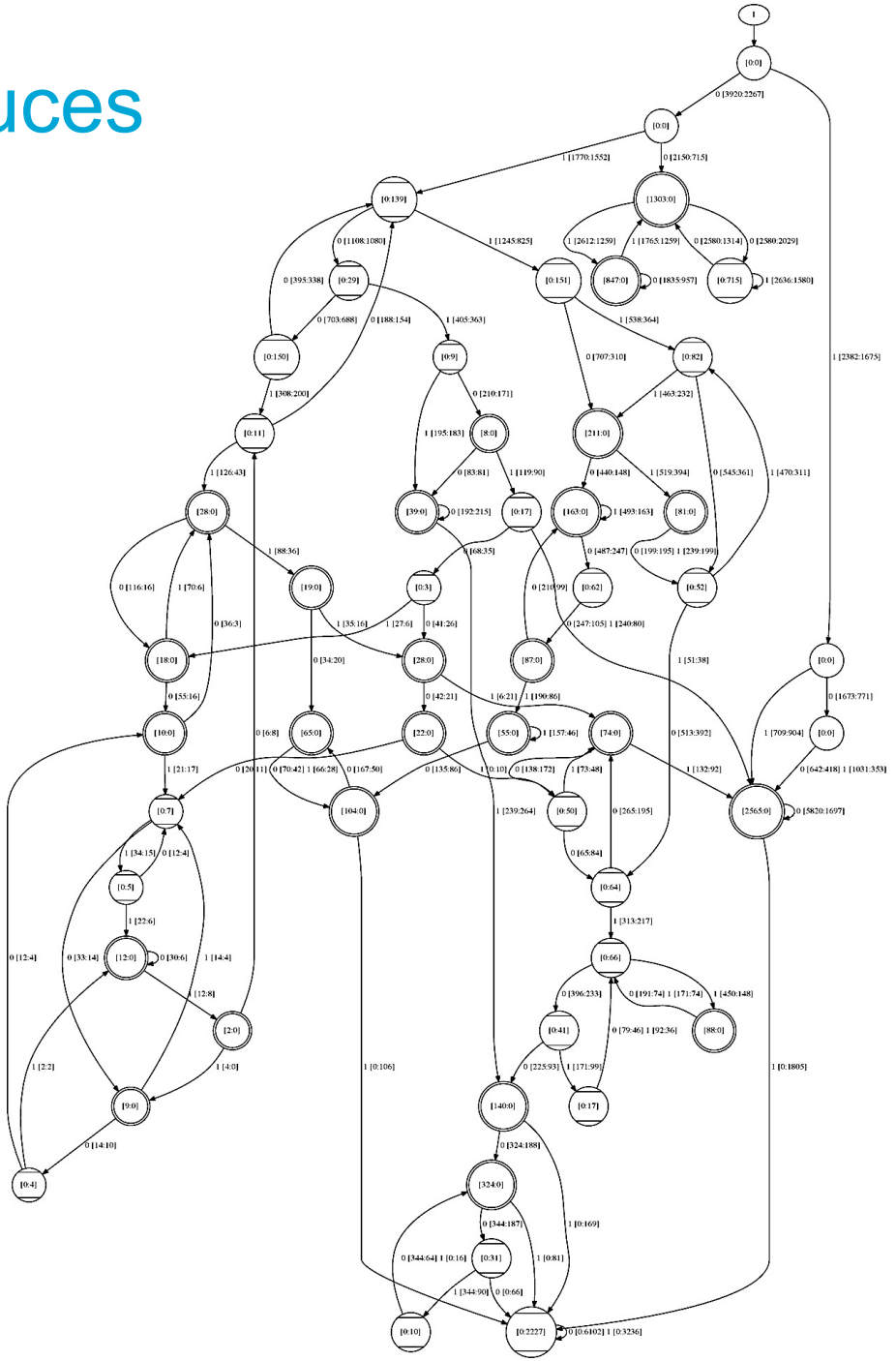
```
1 9 1 0 0 0 0 0 0 0 0
1 15 0 1 1 0 0 0 1 0 1 0 1 0 1 0 0
0 5 0 1 0 0 0
1 2 0 0
0 12 1 1 0 0 0 0 0 0 0 1 0 0
1 5 1 0 1 0 0
1 3 1 0 0
0 6 0 0 0 0 0 0 1
1 3 1 0 1
0 20 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 1 0 0
0
0 13 0 1 1 1 0 1 0 0 0 0 0 0 0
1 3 1 0 1
1 7 1 0 0 0 0 0 0
```



TU Delft

DFASAT produces

1 9 1 0 0 0 0 0 0 0 0
1 15 0 1 1 0 0 0 1 0 1 0 1 0 1 0 0
0 5 0 1 0 0 0
1 2 0 0
0 12 1 1 0 0 0 0 0 0 0 1 0 0
1 5 1 0 1 0 0
1 3 1 0 0
0 6 0 0 0 0 0 0 1
1 3 1 0 1
0 20 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 1 0 0 0
0 13 0 1 1 1 0 1 0 0 0 0 0 0 0
1 3 1 0 1
1 7 1 0 0 0 0 0 0
....
10244 strings in total



TU Delft