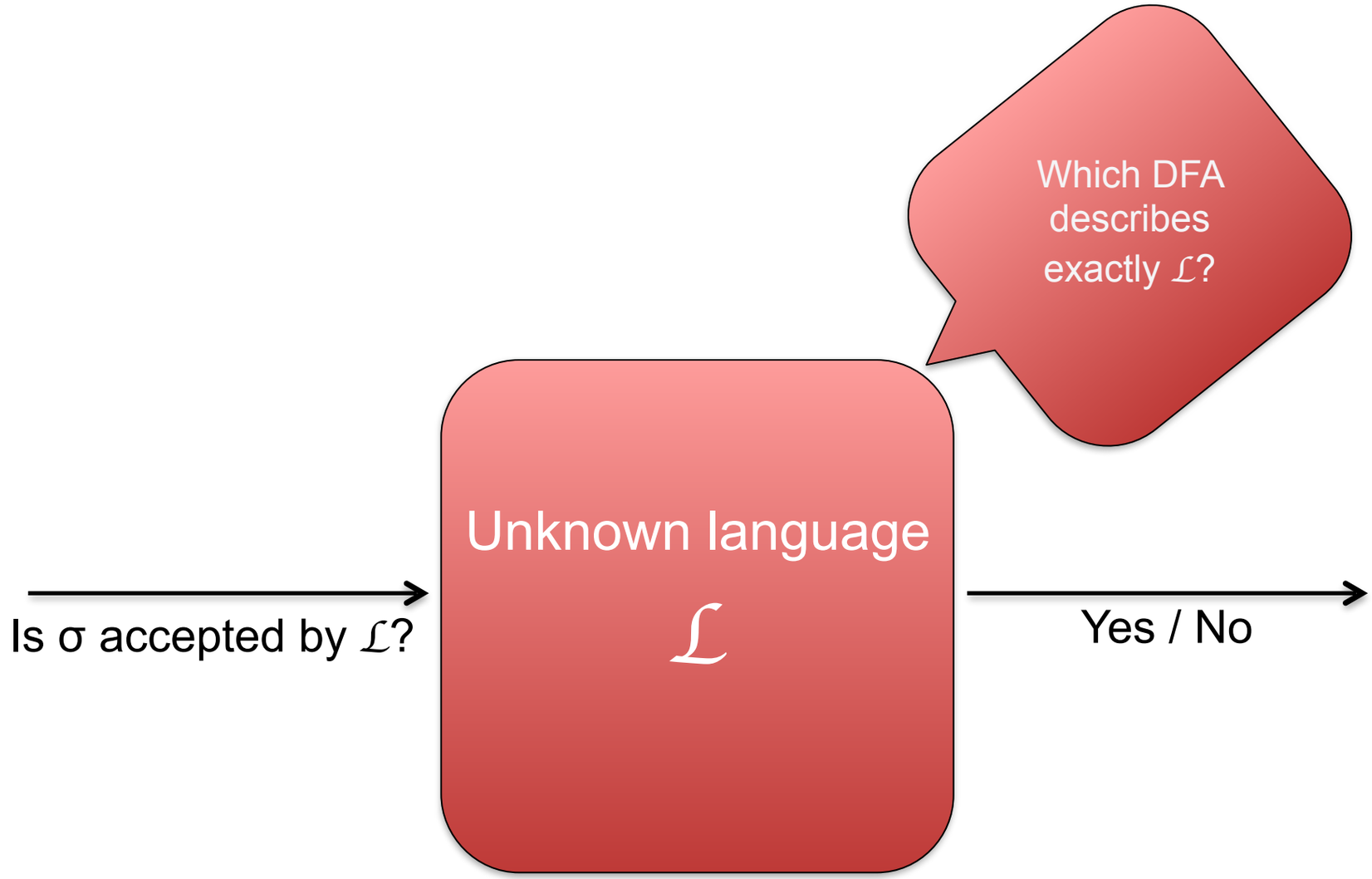
Active Automata Learning
for Model-Based Testing

Michele Volpato

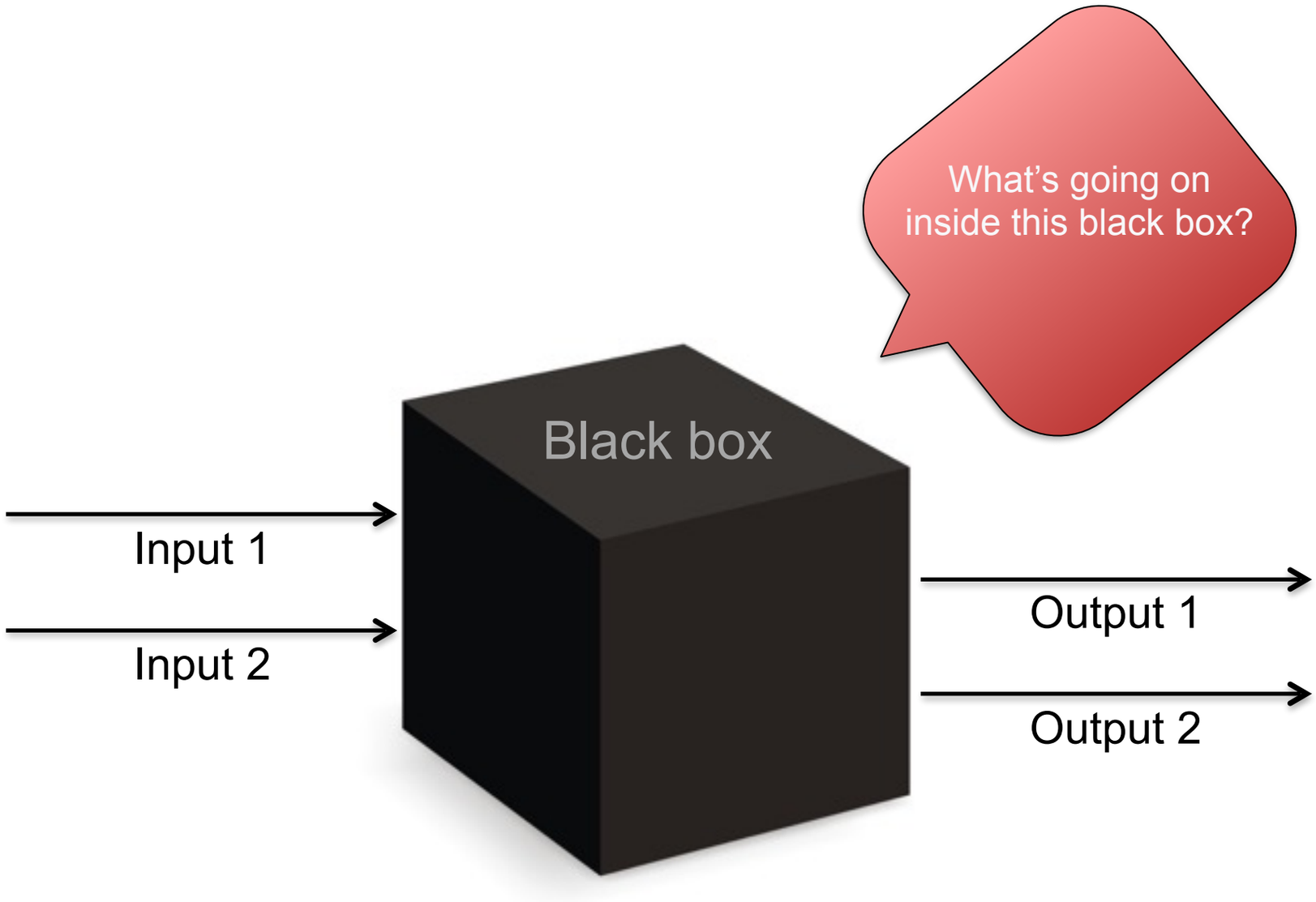
Testing Techniques 2014-2015

Radboud University

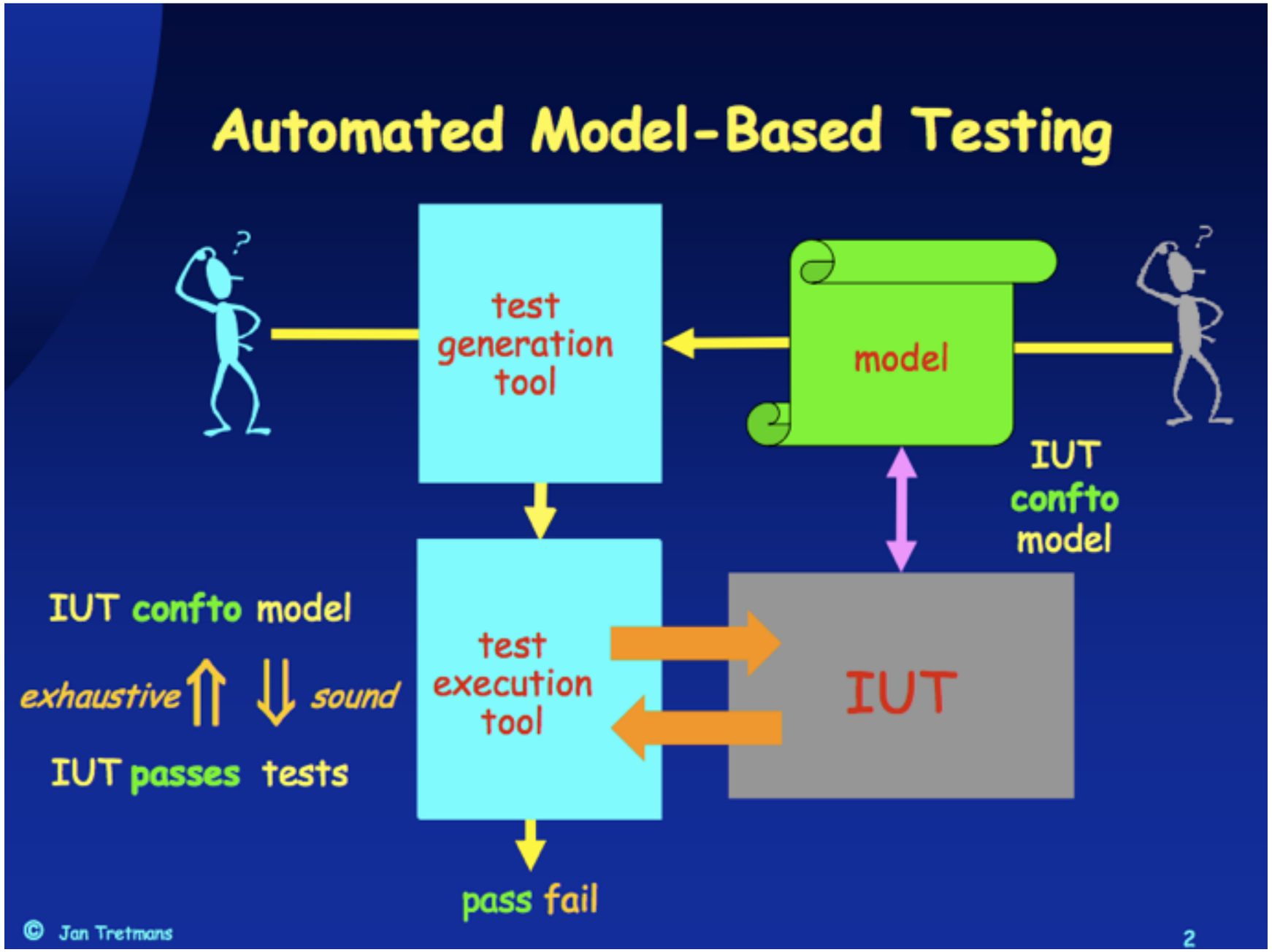
Automata Learning Goal



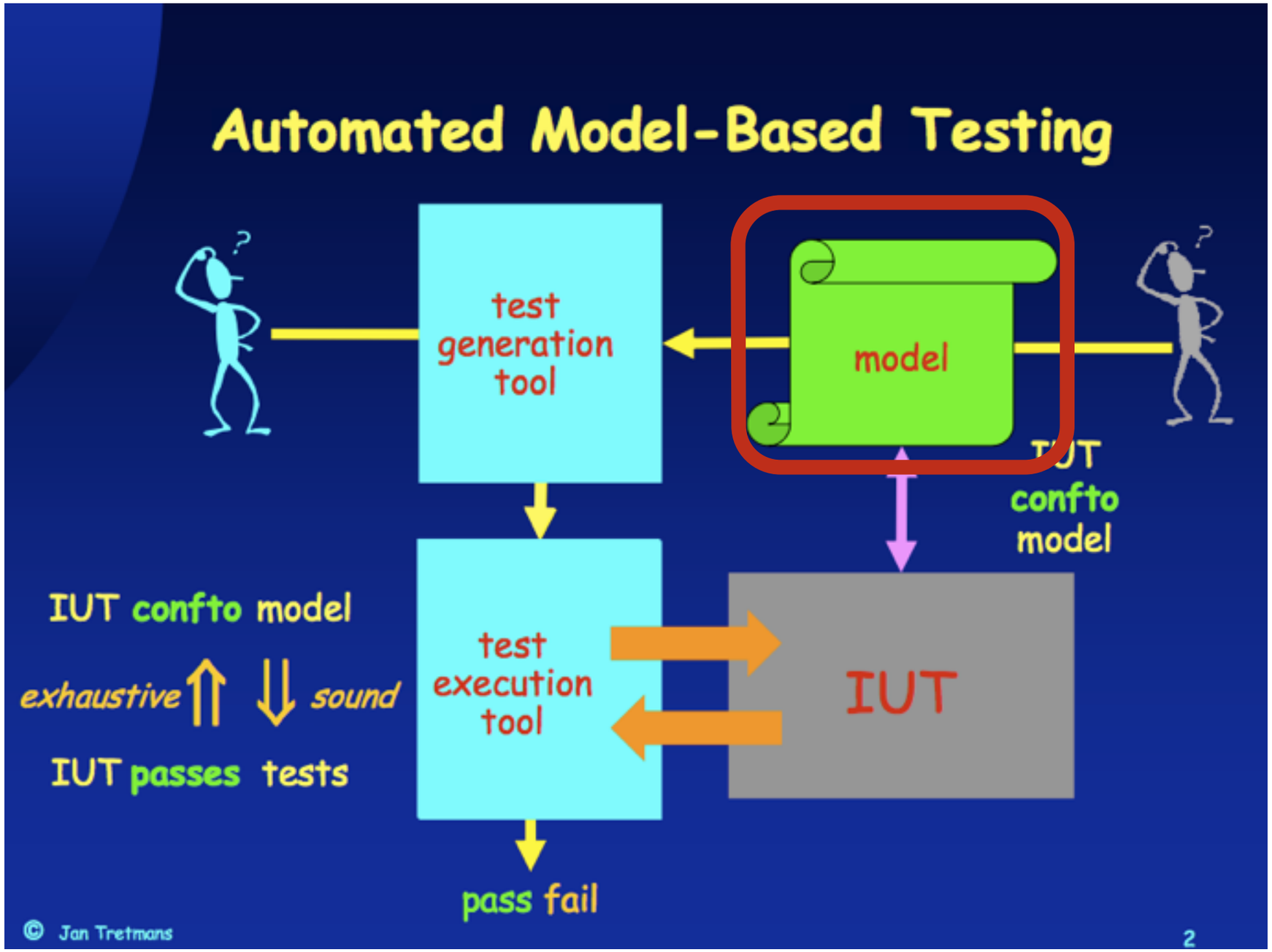
Automata Learning Goal



Motivation


Automated Model-Based Testing
test generation tool
model
IUT confto model
IUT confto model
exhaustive
sound
IUT passes tests
test execution tool
IUT
pass fail
© Jan Tretmans
2

Motivation

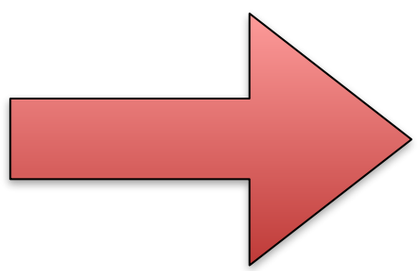

Automated Model-Based Testing
test generation tool
model
IUT confto model
IUT confto model
exhaustive
sound
IUT passes tests
test execution tool
IUT
pass fail
© Jan Tretmans
2

Motivation

- We want to apply Model-Based techniques;
- but often there are no models!
- or existing models are unrelated to implementations.

Motivation

- We want to apply Model-Based techniques;
- but often there are no models!
- or existing models are unrelated to implementations.



Produce models FROM implementations!

Example

River Crossing game

Outline

1. Introduction
2. Active Automata Learning primer
3. Mealy Machines
4. Nerode relation
5. Counterexample handling
6. Other formalisms
7. Towards Nondeterminism

Introduction

Children learn how to use computers.

Introduction

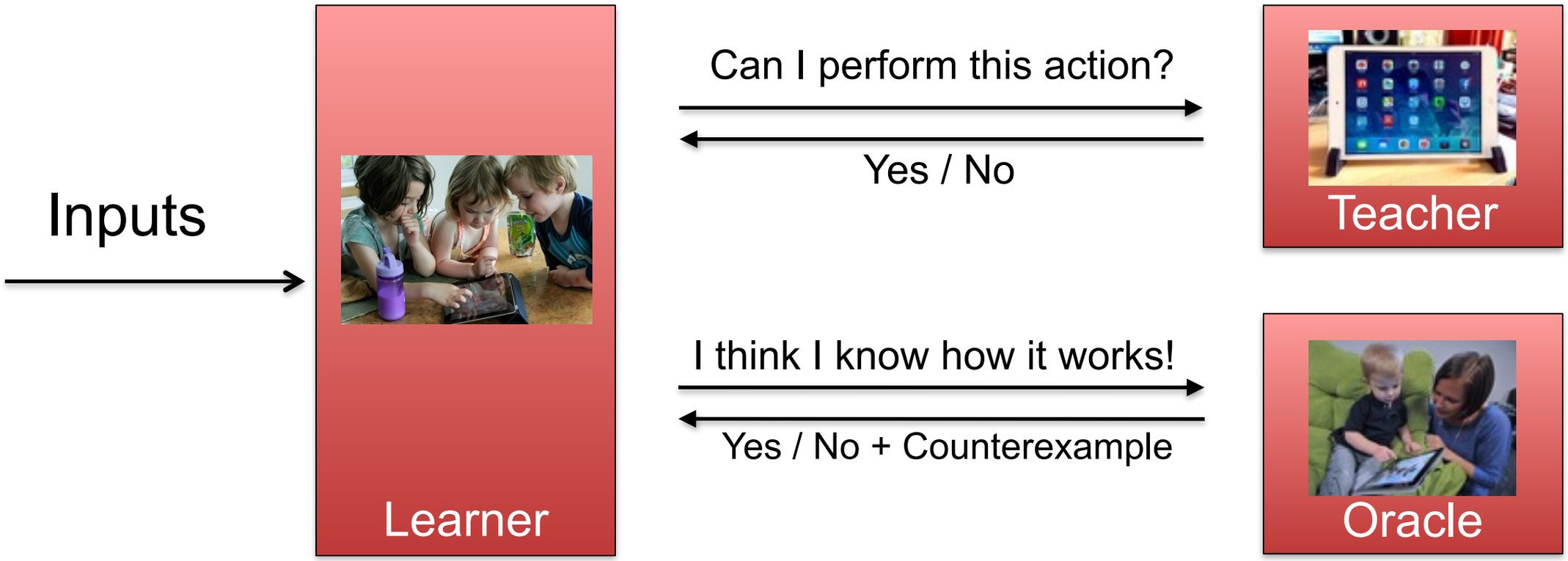
Children learn how to use computers.

Introduction

Children learn how to use computers.

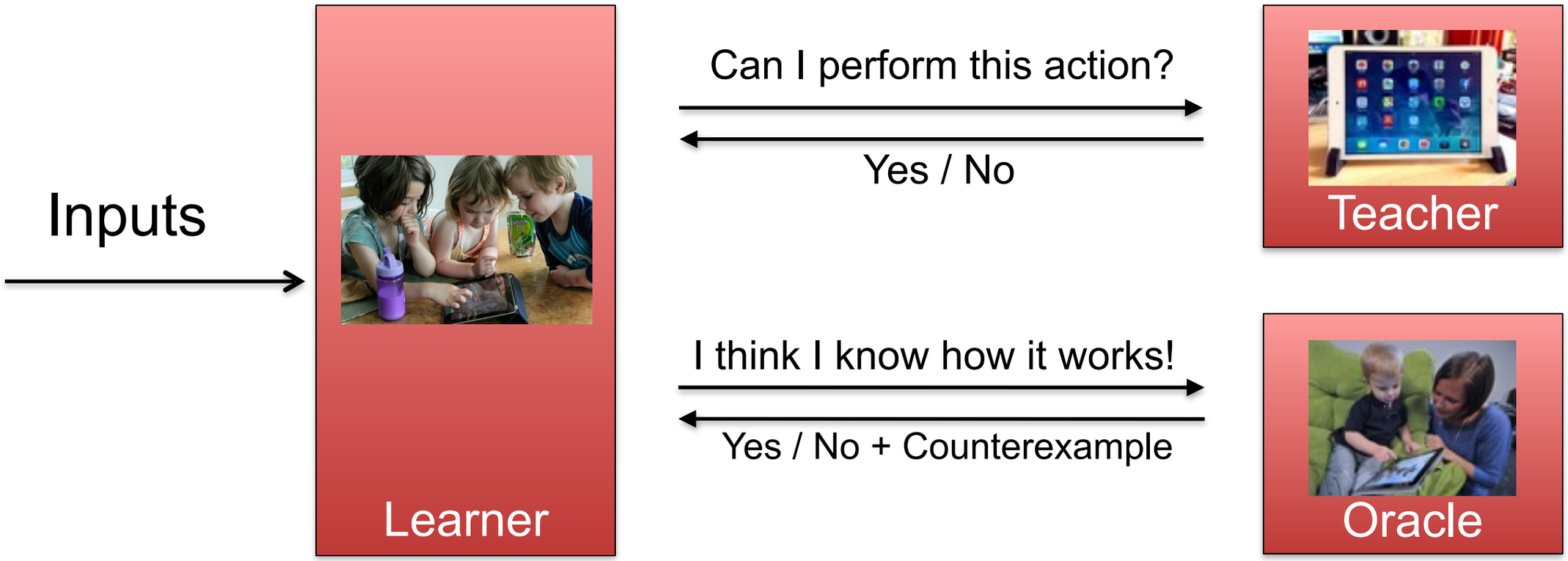
Can computer learn state diagrams as well?

Basic algorithm



Angluin's L* algorithm

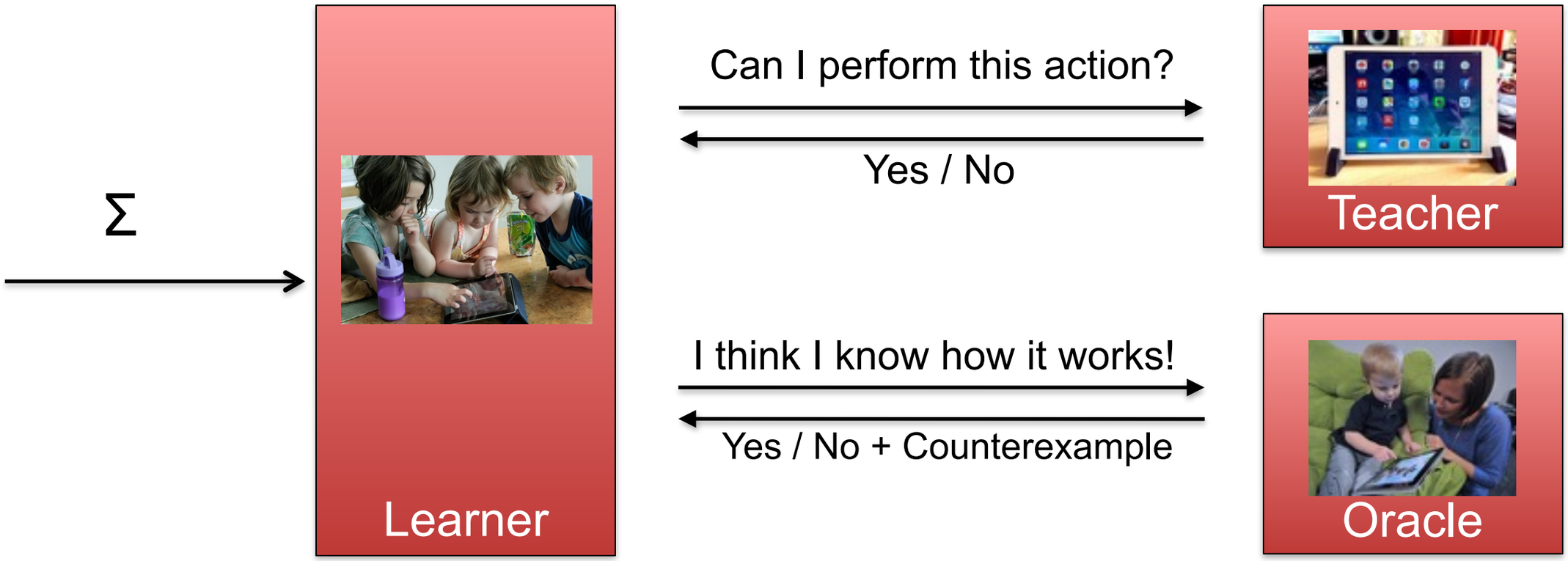
- Learning Finite Automata



1987

Angluin's L* algorithm

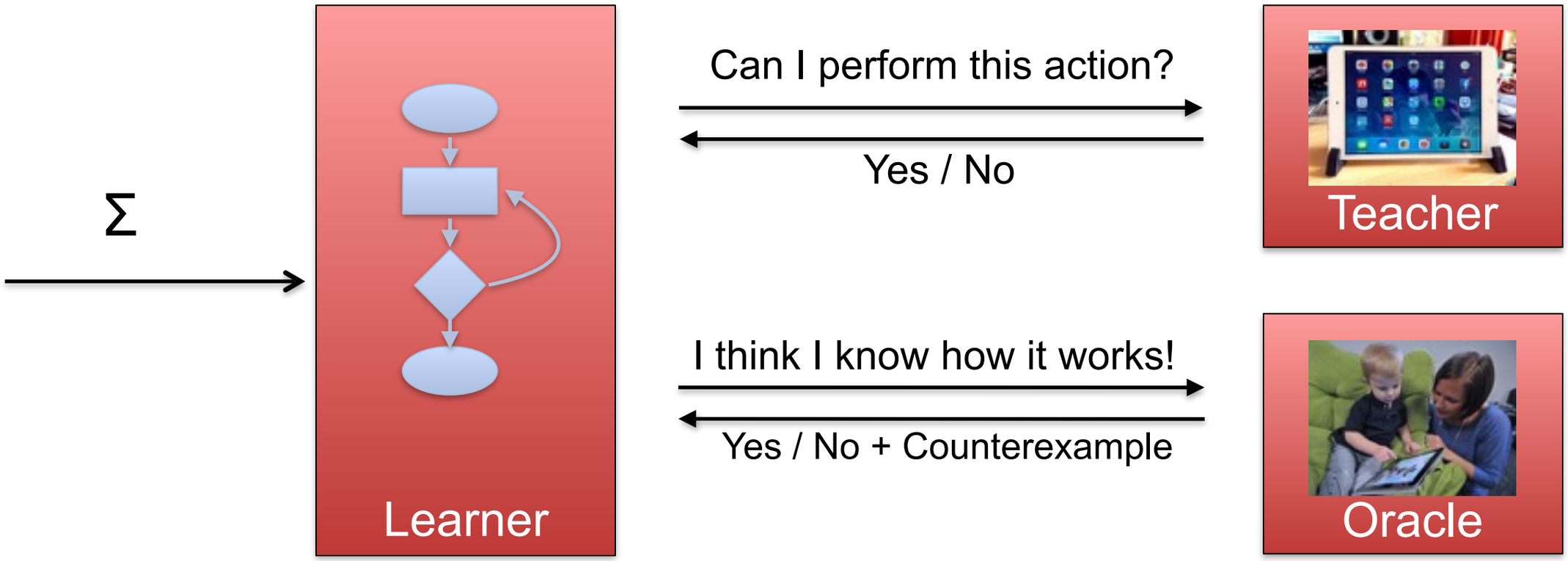
- Learning Finite Automata



1987

Angluin's L* algorithm

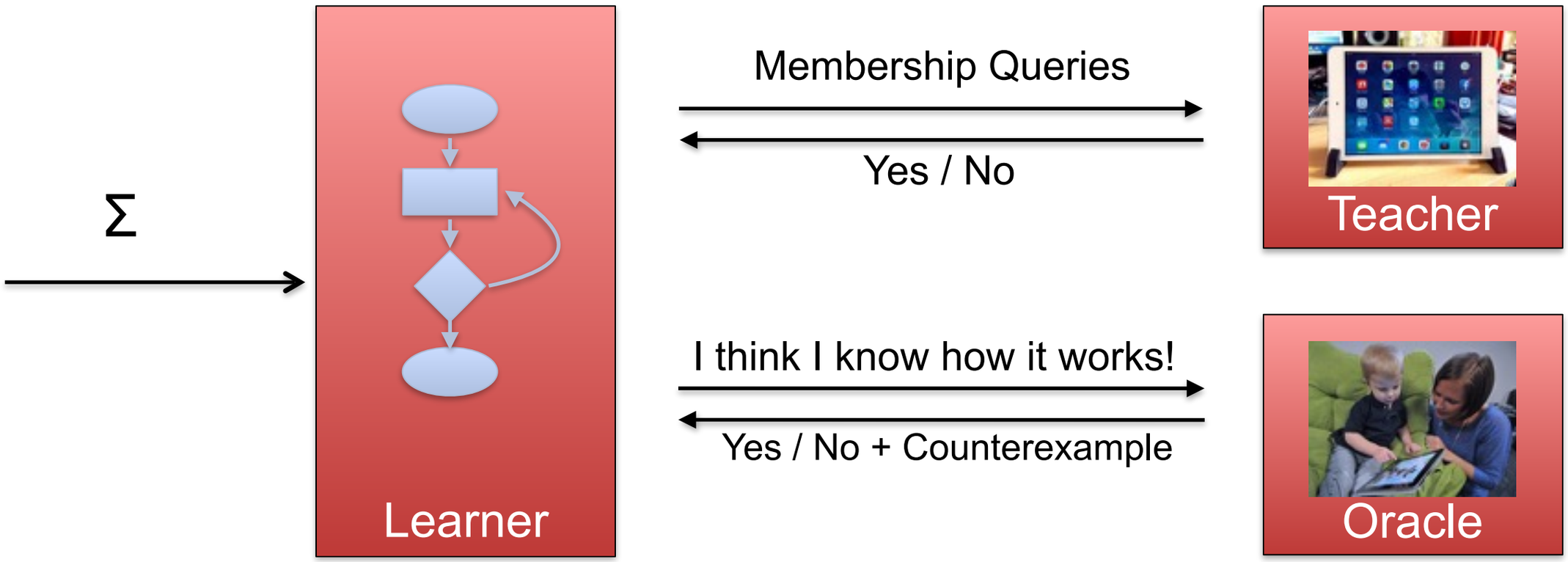
- Learning Finite Automata



1987

Angluin's L* algorithm

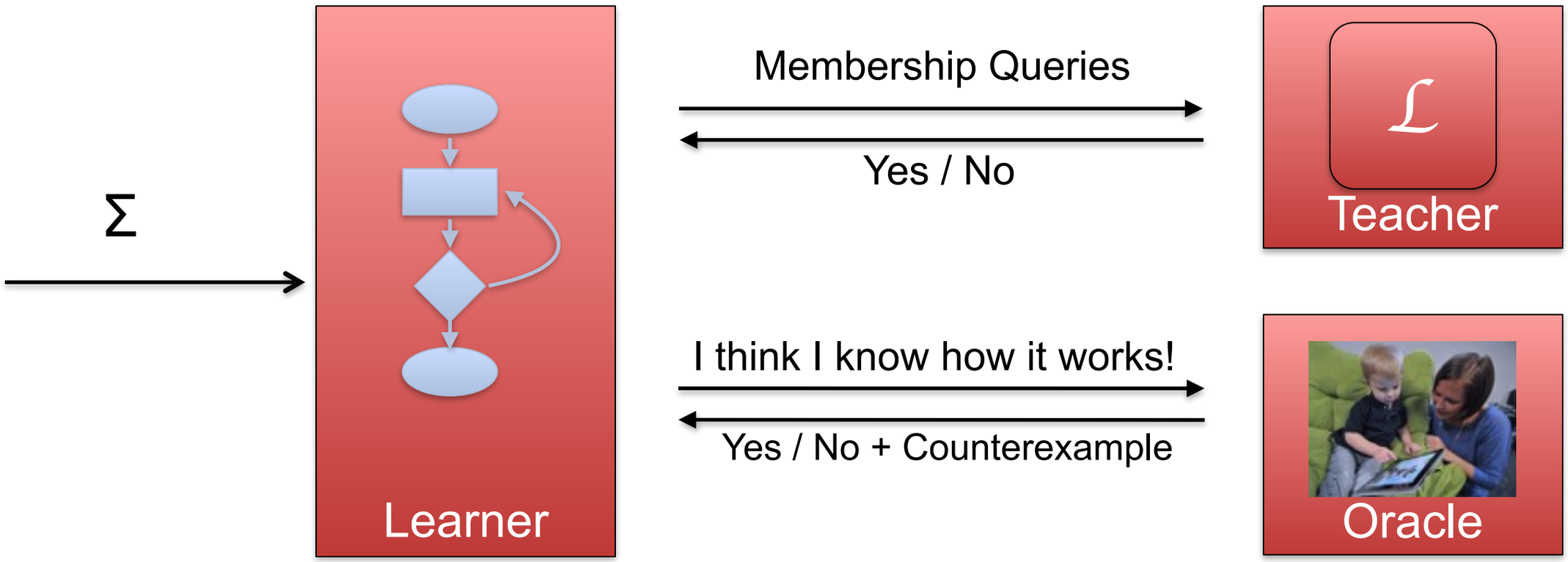
- Learning Finite Automata



1987

Angluin's L* algorithm

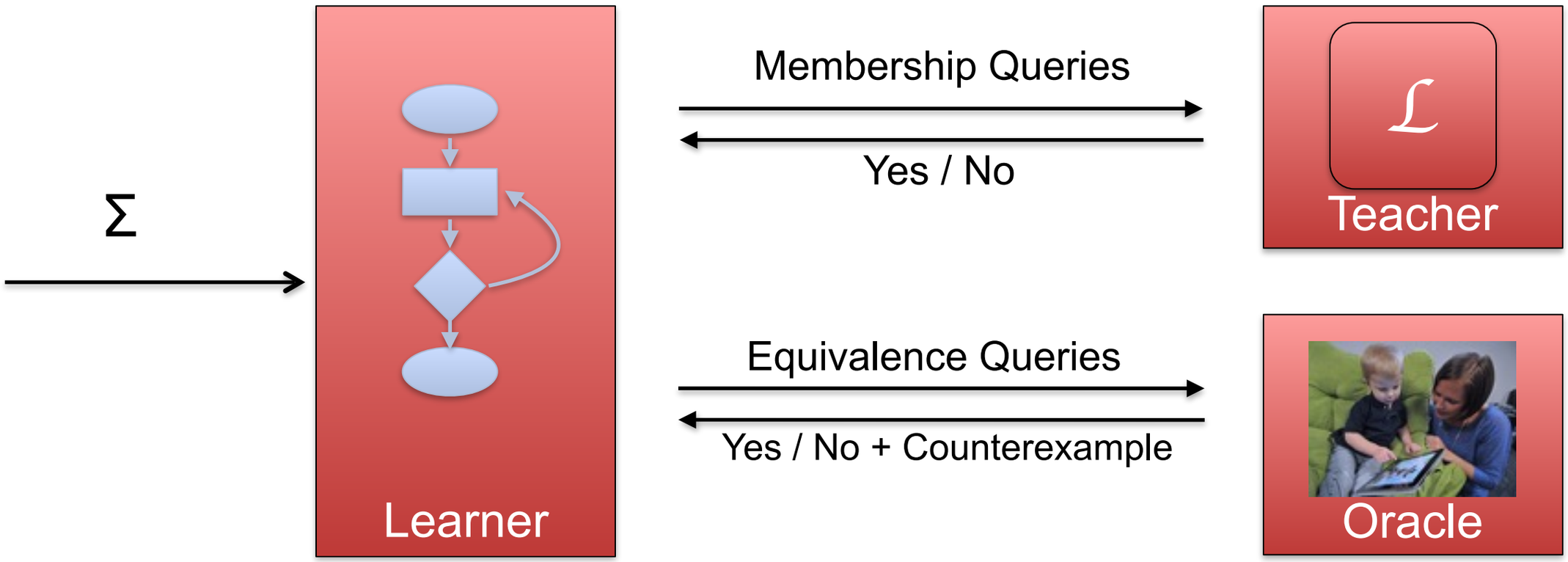
- Learning Finite Automata



1987

Angluin's L* algorithm

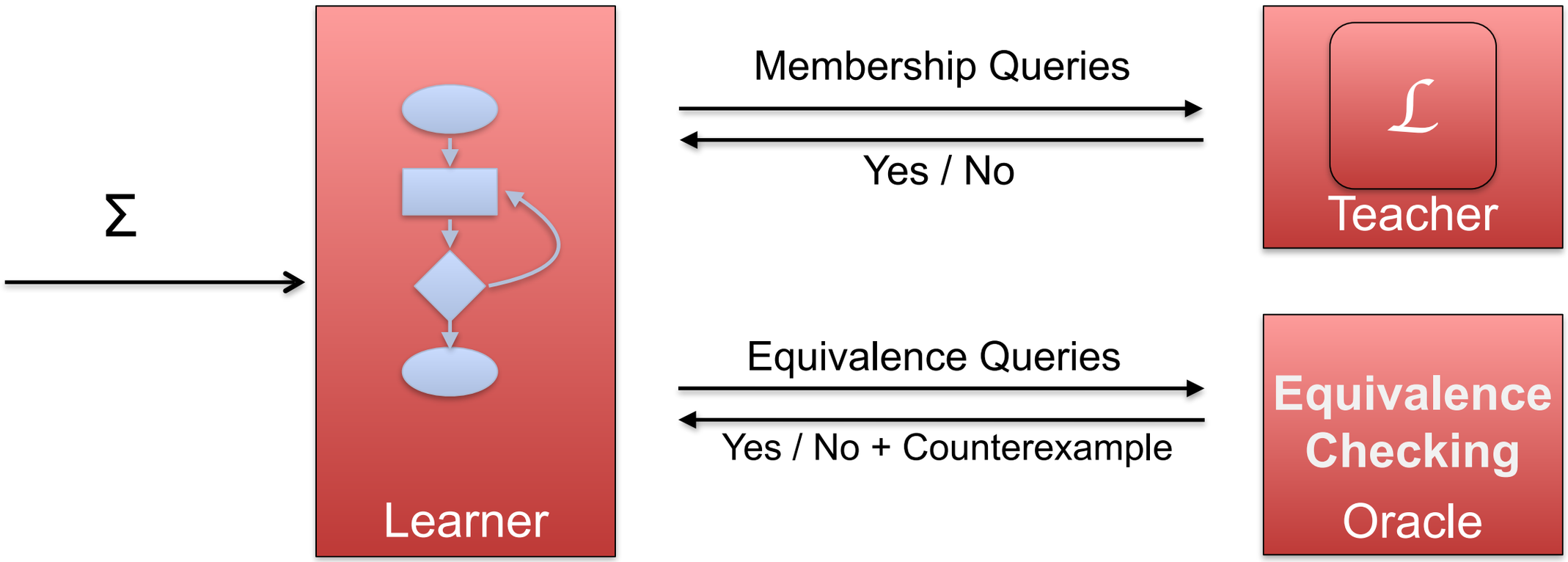
- Learning Finite Automata



1987

Angluin's L* algorithm

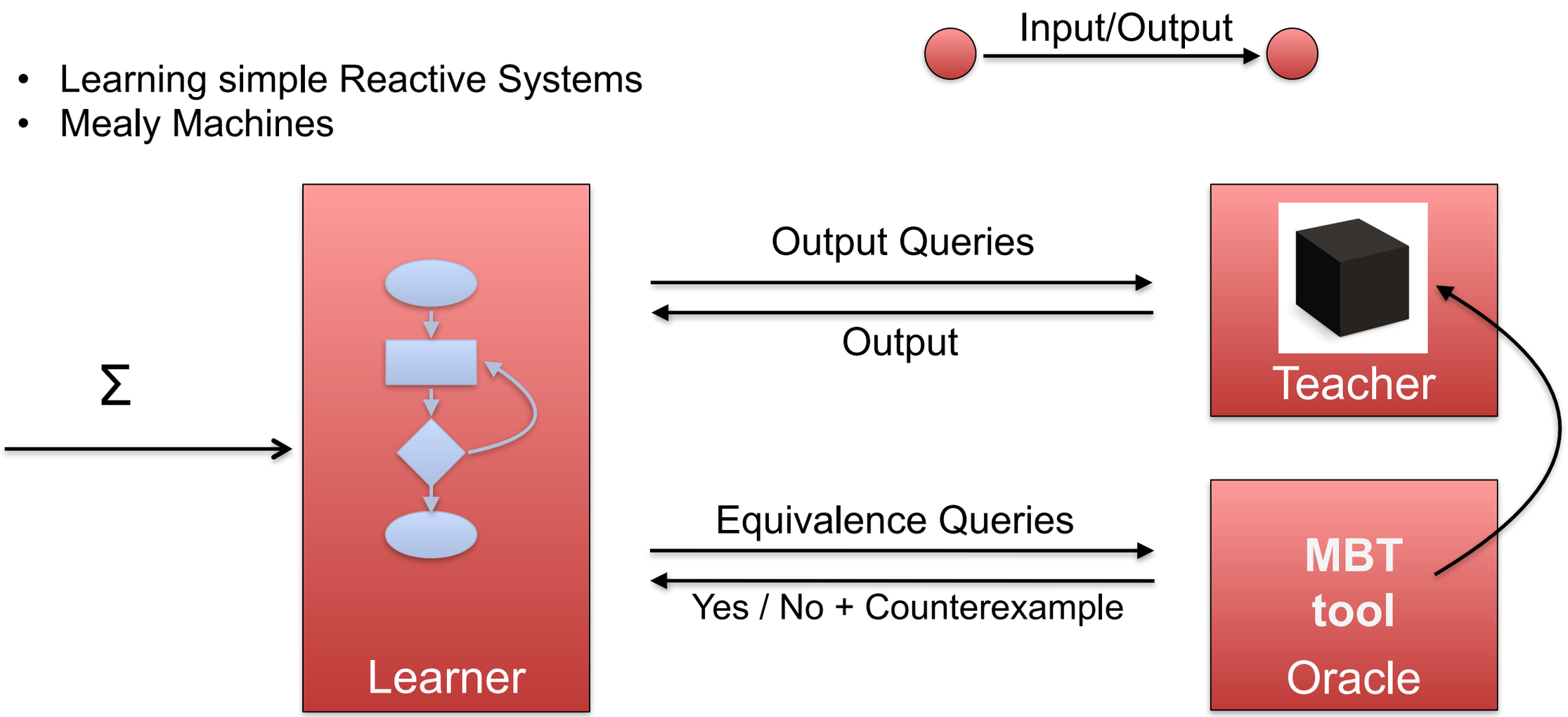
- Learning Finite Automata



1987

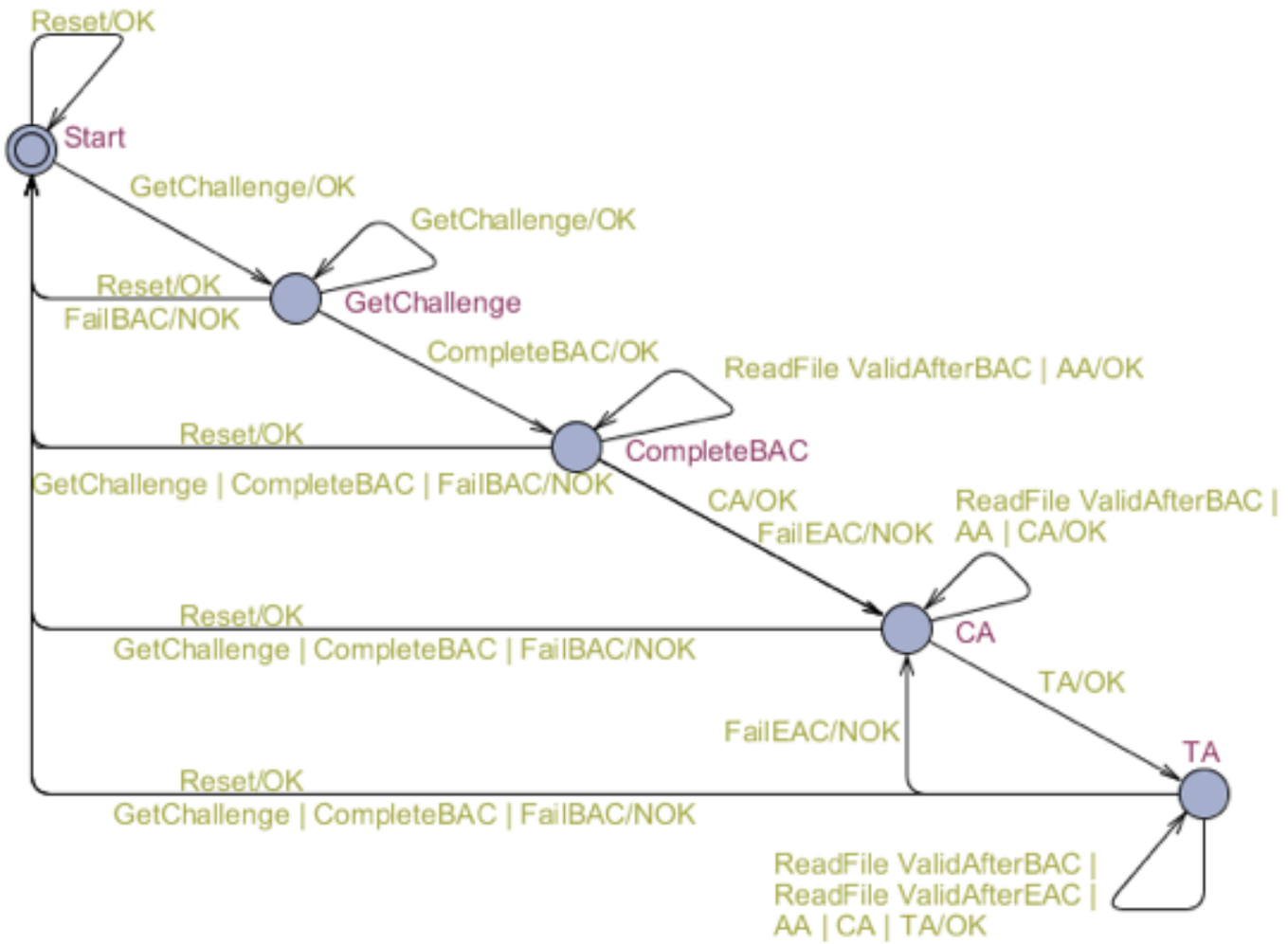
L*-style learning algorithms

- Learning simple Reactive Systems
- Mealy Machines



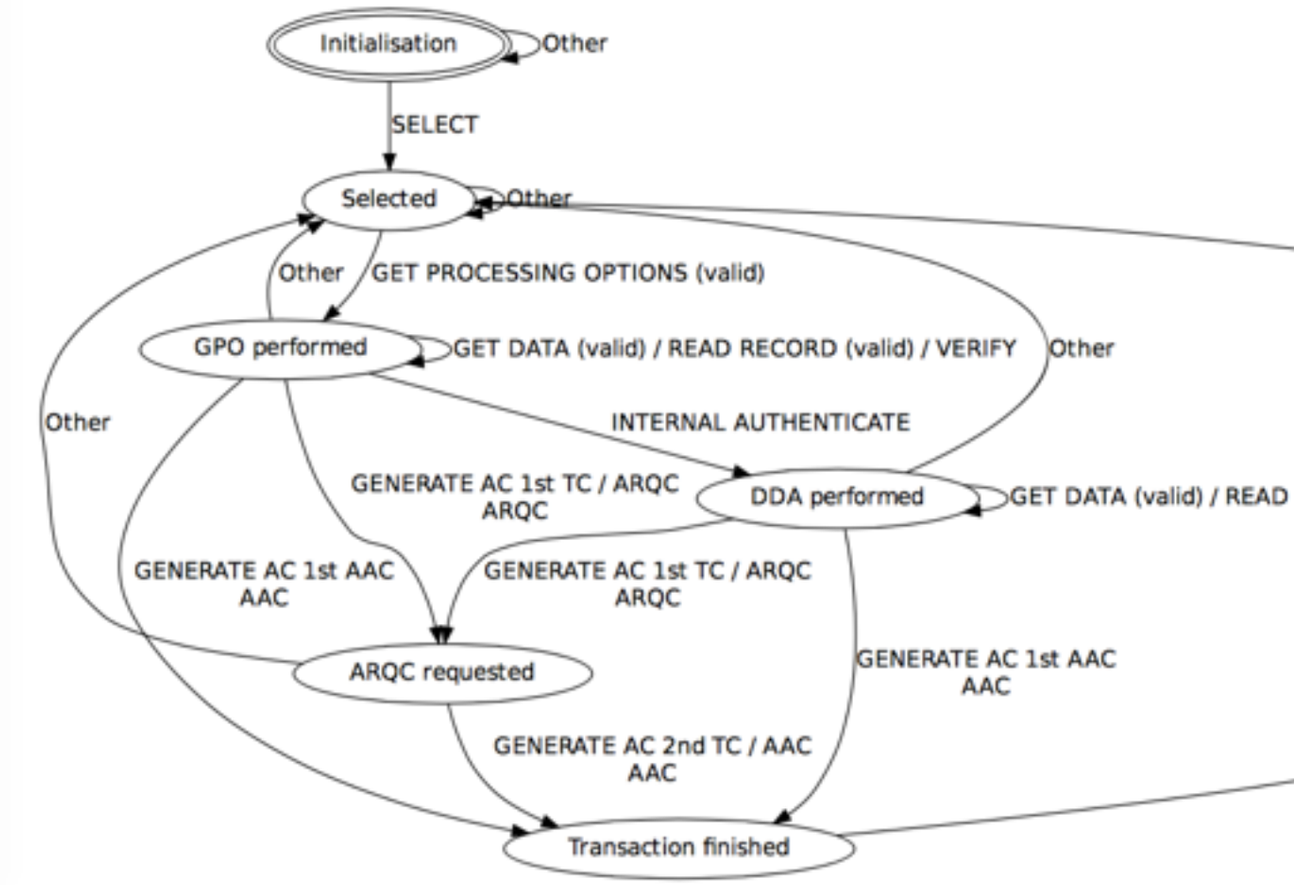
http://learnlib.de/ - 2005

Applications - Biometric Passport



Applications - Banking Cards

- EMV protocol



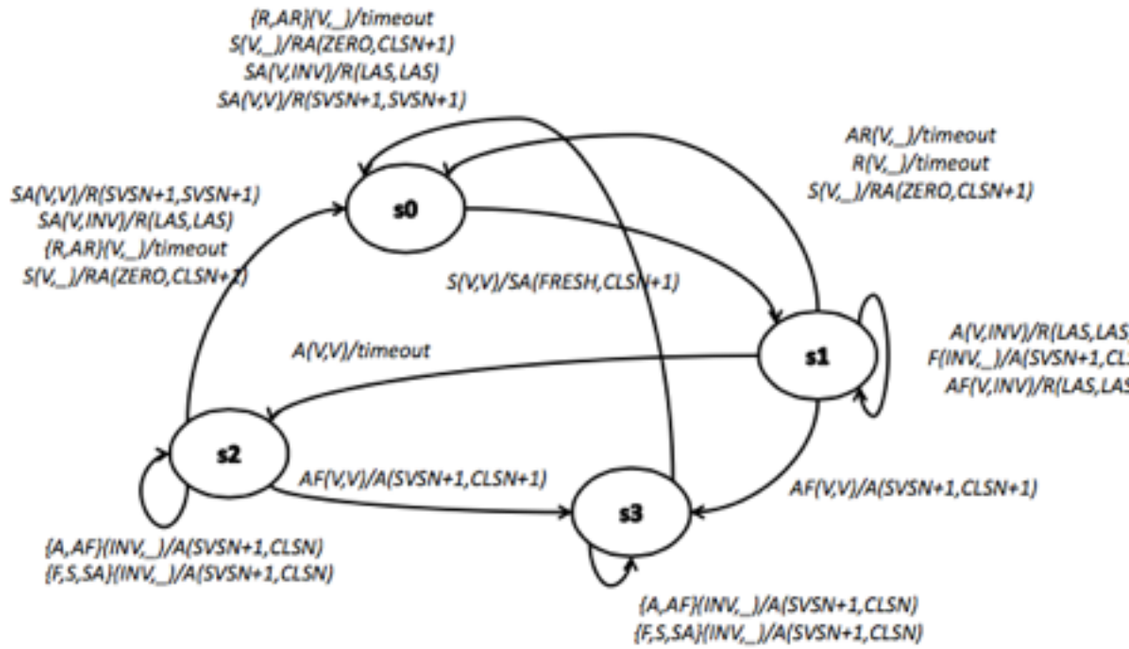
Applications - TCP

Learning Fragments of the TCP Network Protocol

Paul Fiterău-Broştean*, Ramon Janssen, and Frits Vaandrager

Institute for Computing and Information Sciences
Radboud University Nijmegen
P.O. Box 9010, 6500 GL Nijmegen, the Netherlands
{P.FiterauBrostean,f.vaandrager}@cs.ru.nl, ramon.janssen@student.ru.nl

Abstract. We apply automata learning techniques to learn fragments of the TCP network protocol by observing its external behavior. We show that different implementations of TCP in Windows 8 and Ubuntu induce different automata models, thus allowing for fingerprinting of these

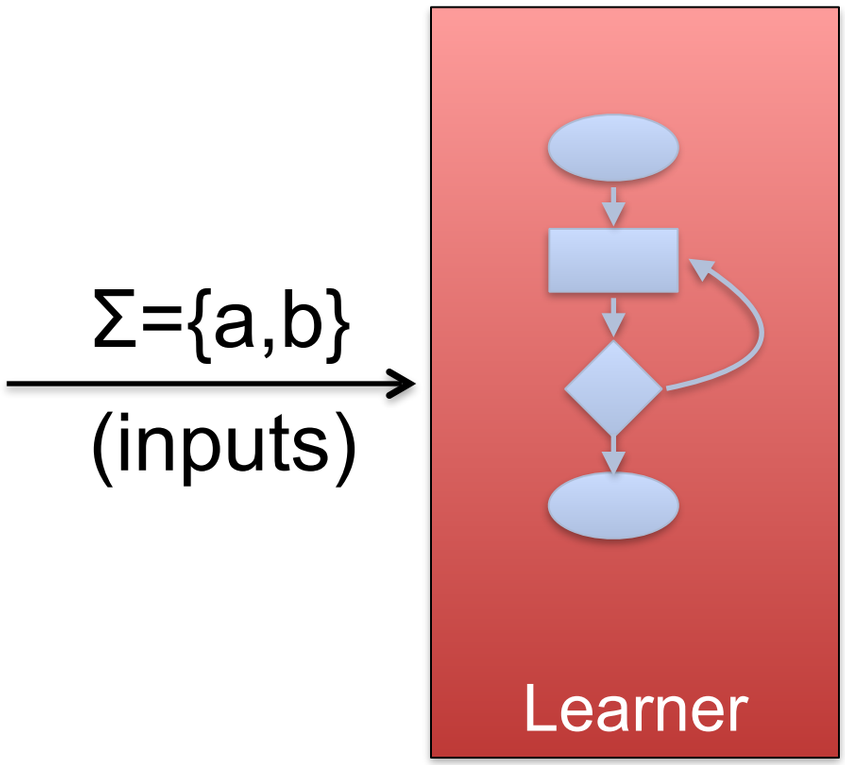


Active Automata Learning primer

Radboud University

Active Automata Learning

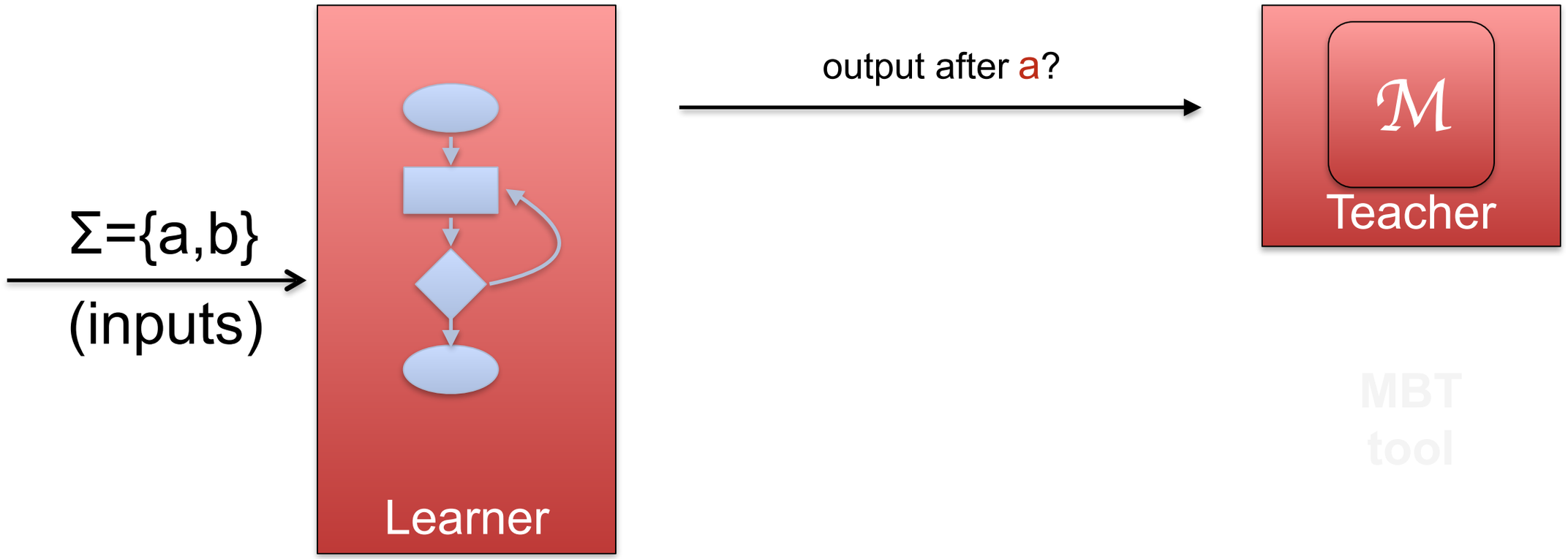
- Objective: Learn unknown Mealy Machine $\mathcal{M}$



MBT tool

Active Automata Learning

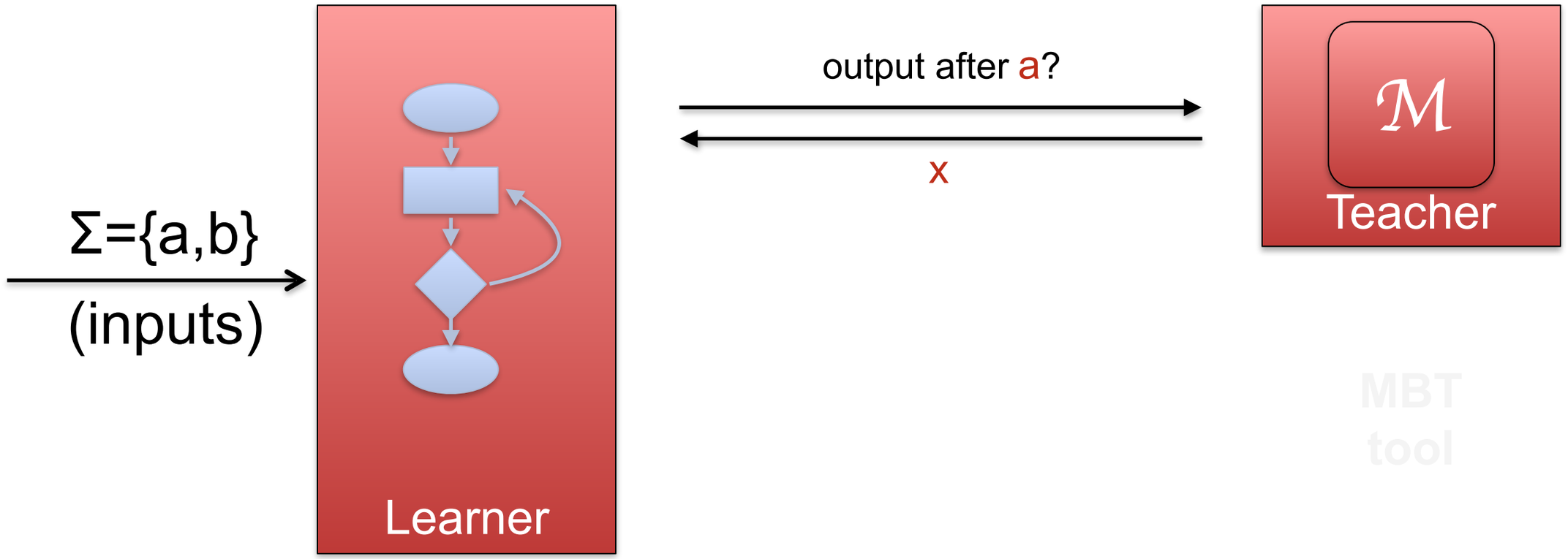
- Objective: Learn unknown Mealy Machine $\mathcal{M}$



MBT tool

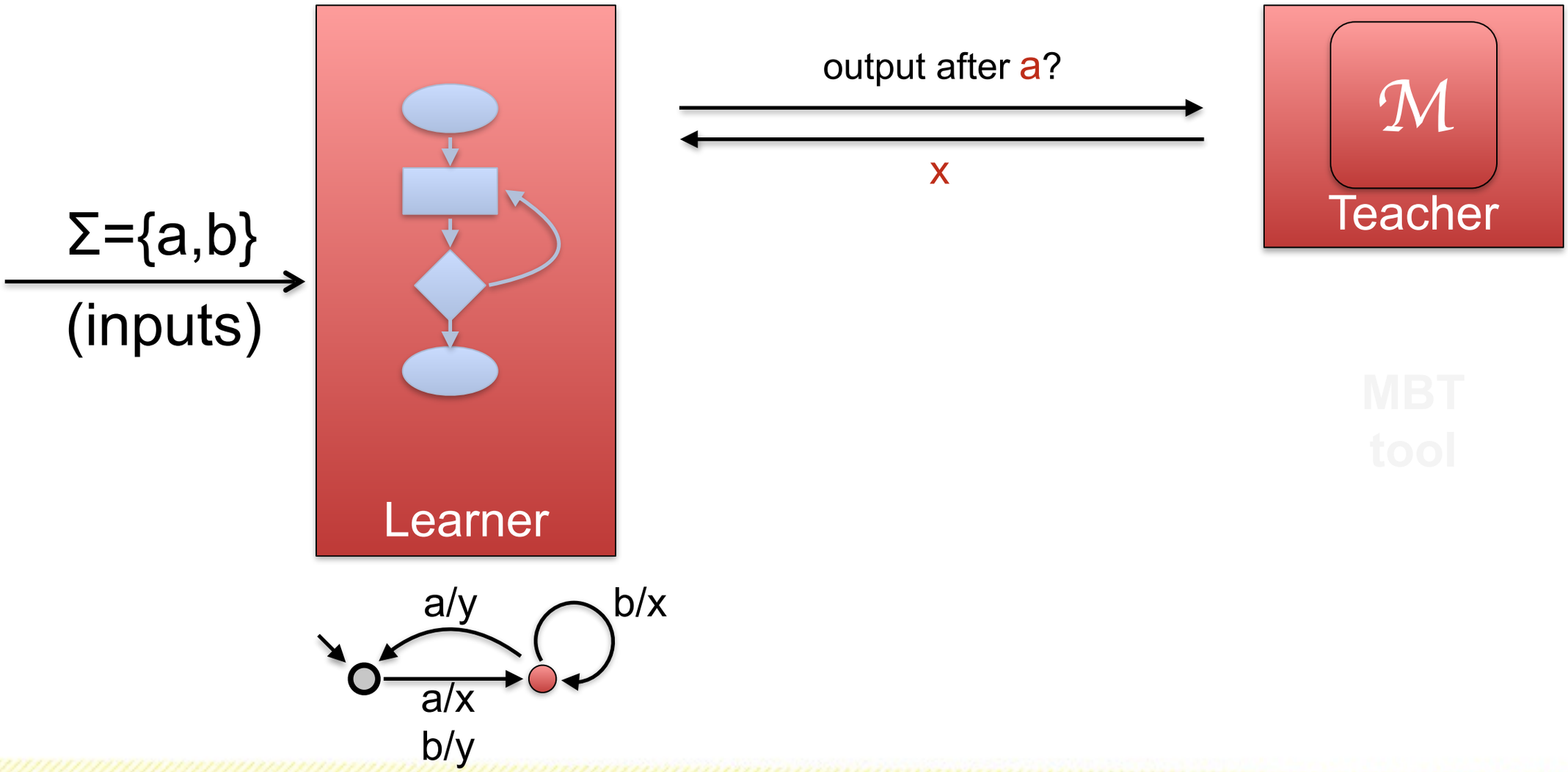
Active Automata Learning

- Objective: Learn unknown Mealy Machine $\mathcal{M}$



Active Automata Learning

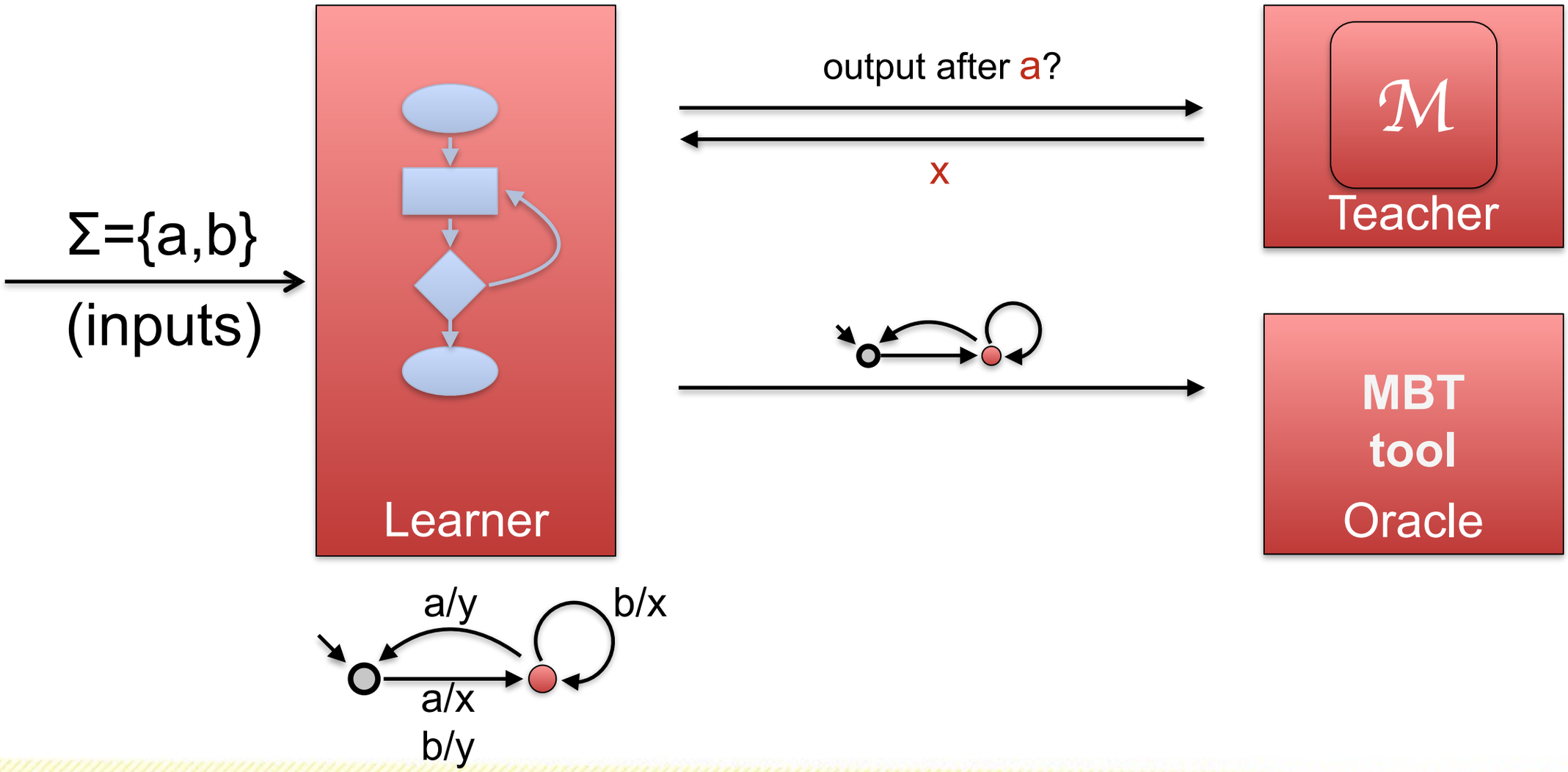
- Objective: Learn unknown Mealy Machine $\mathcal{M}$



MBT tool

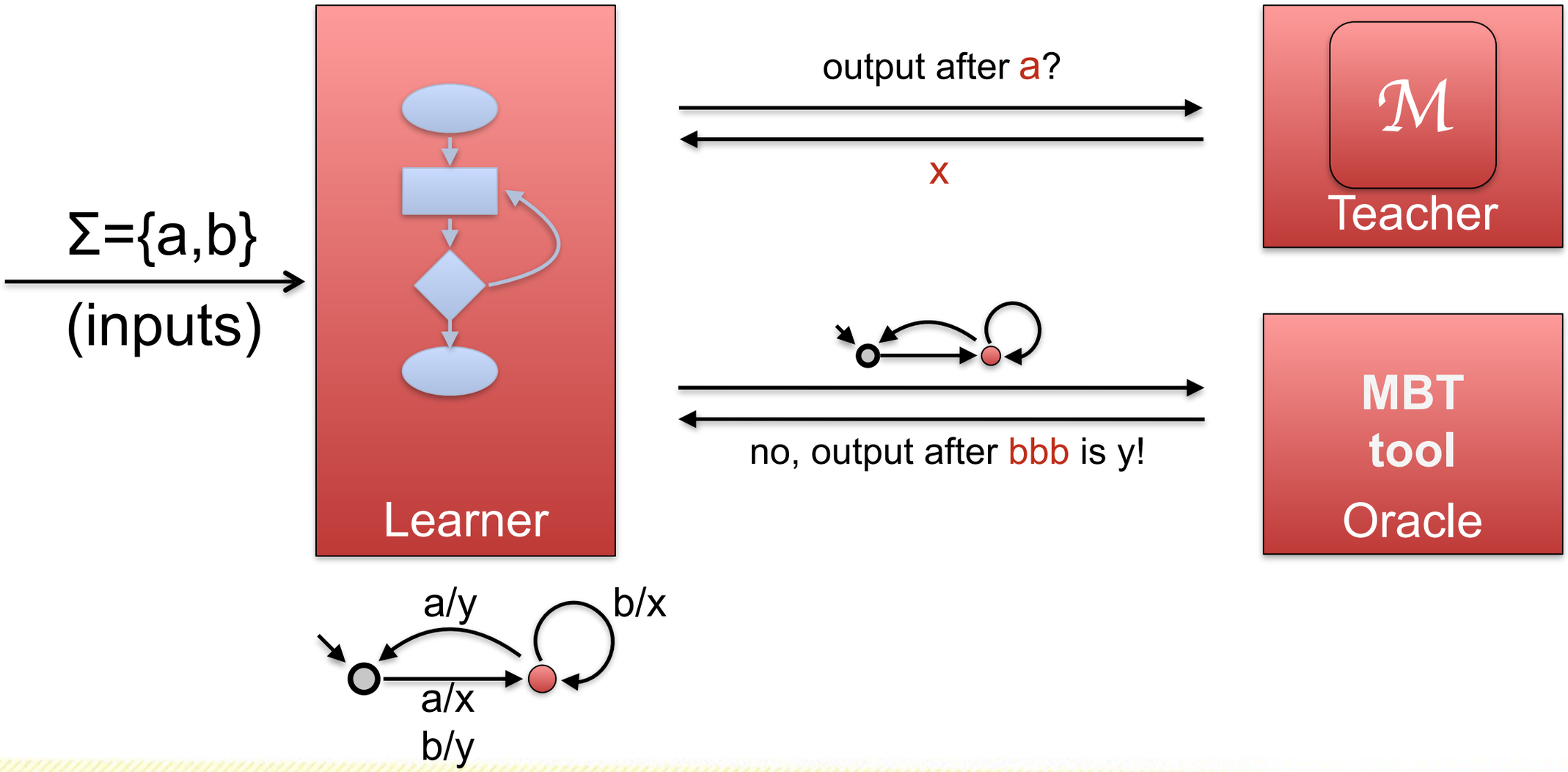
Active Automata Learning

- Objective: Learn unknown Mealy Machine $\mathcal{M}$



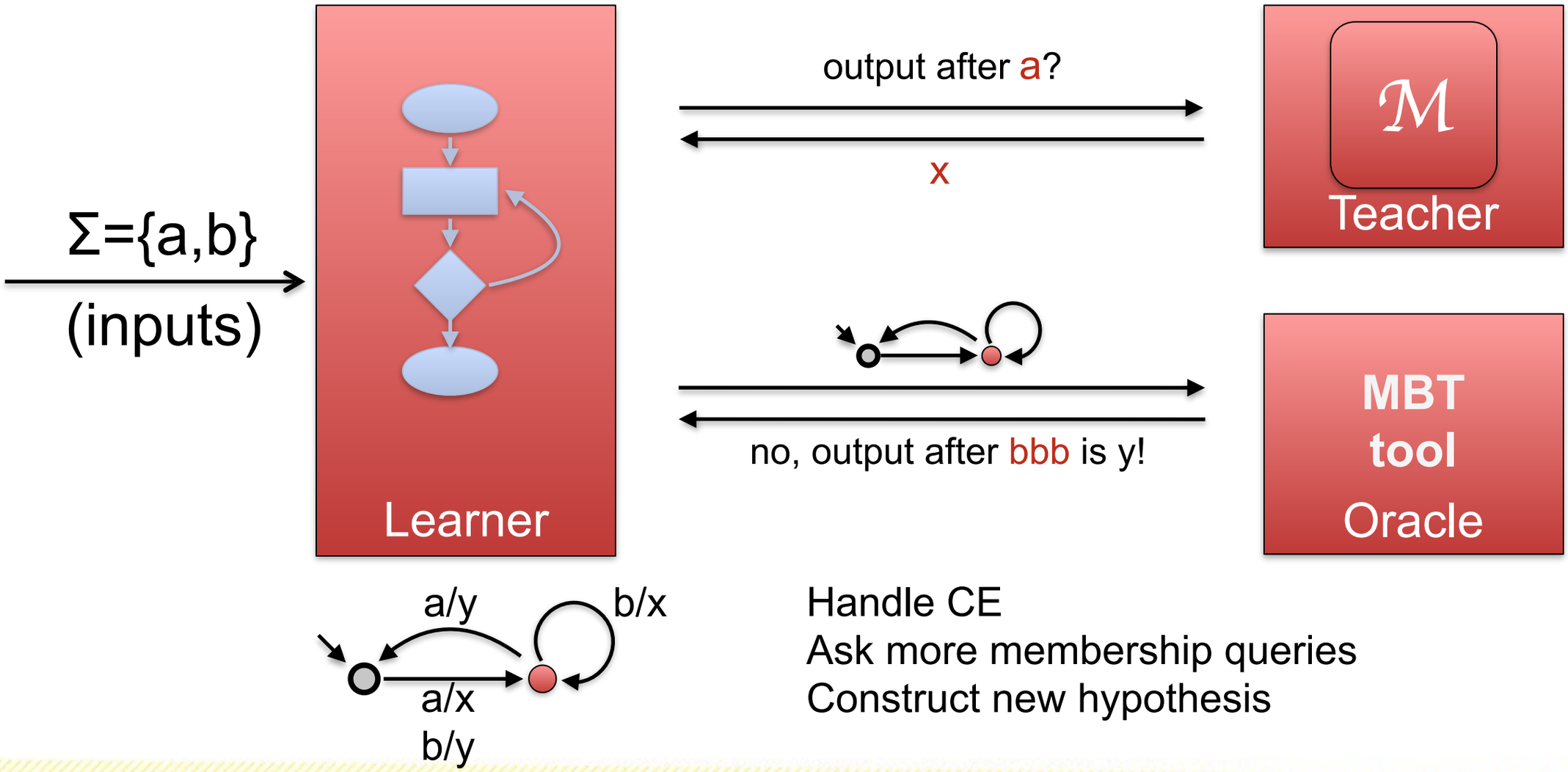
Active Automata Learning

- Objective: Learn unknown Mealy Machine $\mathcal{M}$



Active Automata Learning

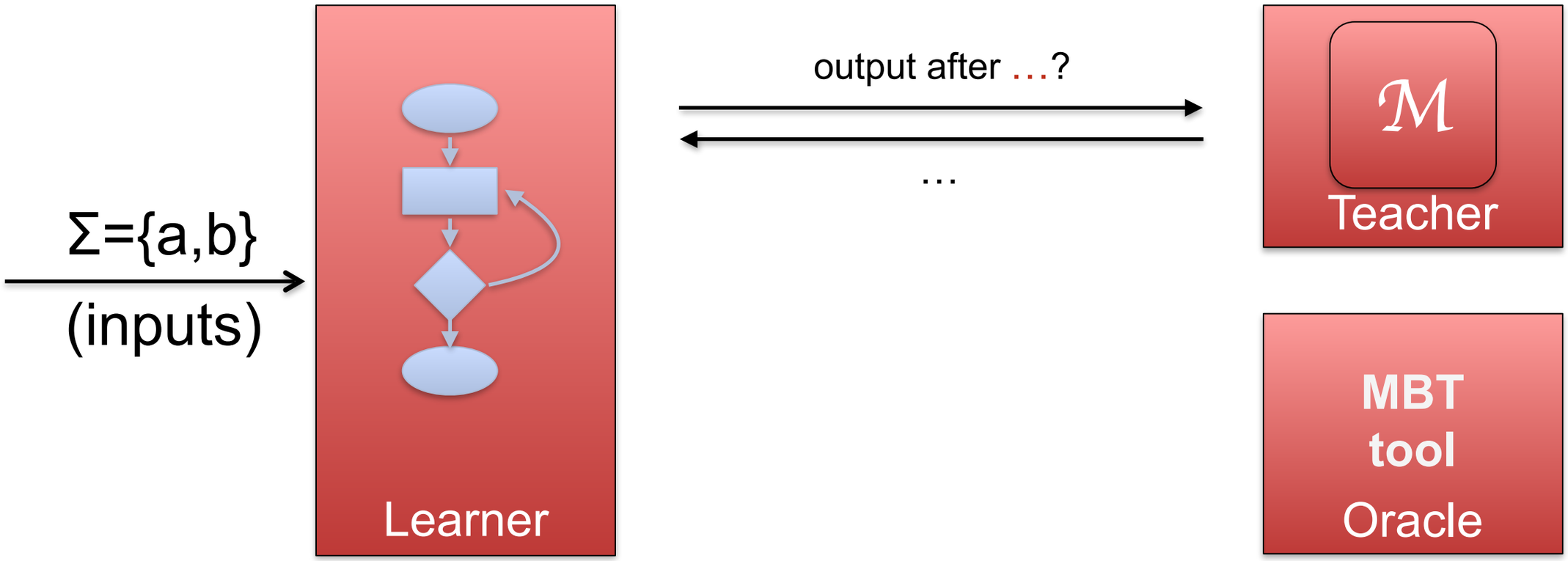
- Objective: Learn unknown Mealy Machine $\mathcal{M}$



Handle CE
Ask more membership queries
Construct new hypothesis

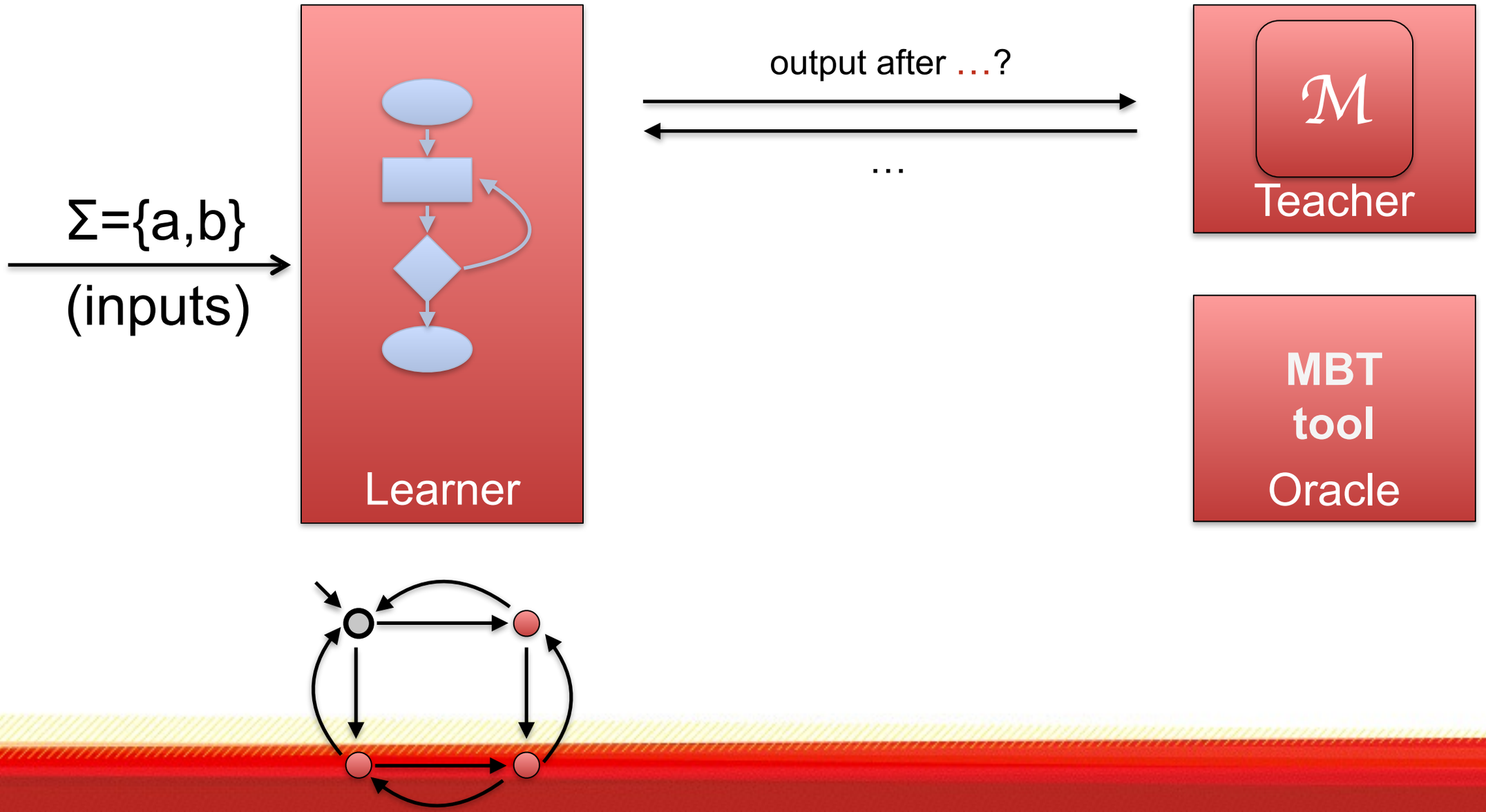
Active Automata Learning

- Objective: Learn unknown regular language $\mathcal{L}$



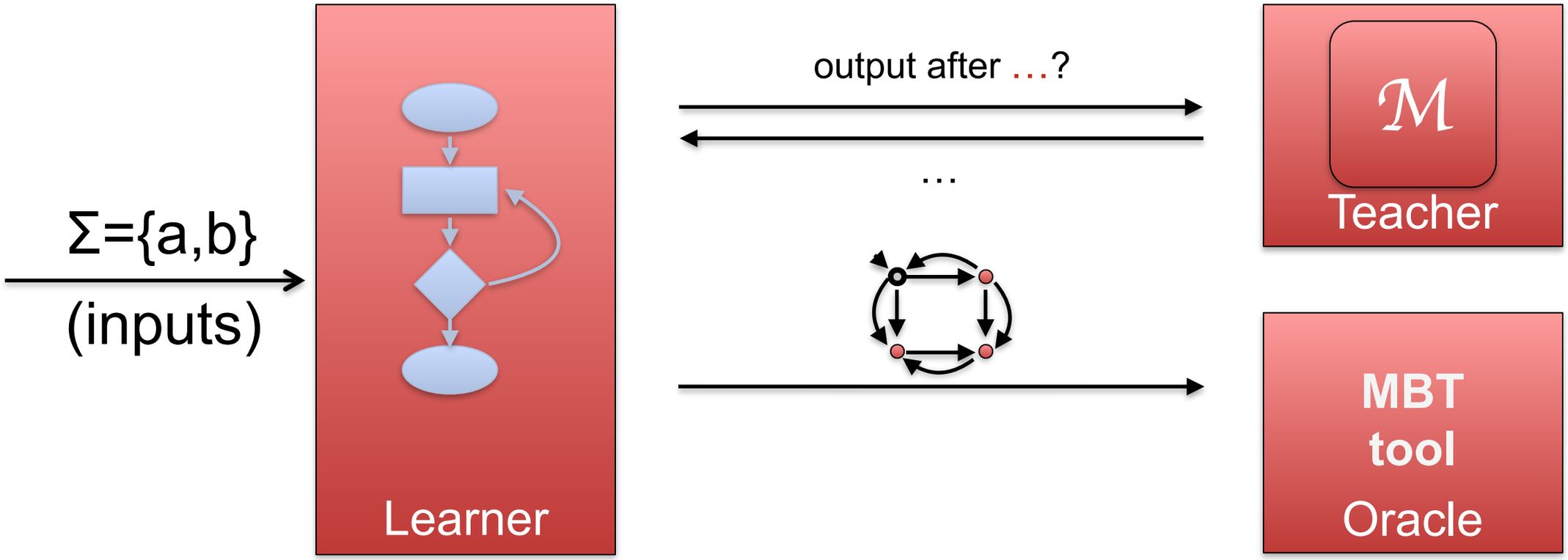
Active Automata Learning

- Objective: Learn unknown regular language $\mathcal{L}$



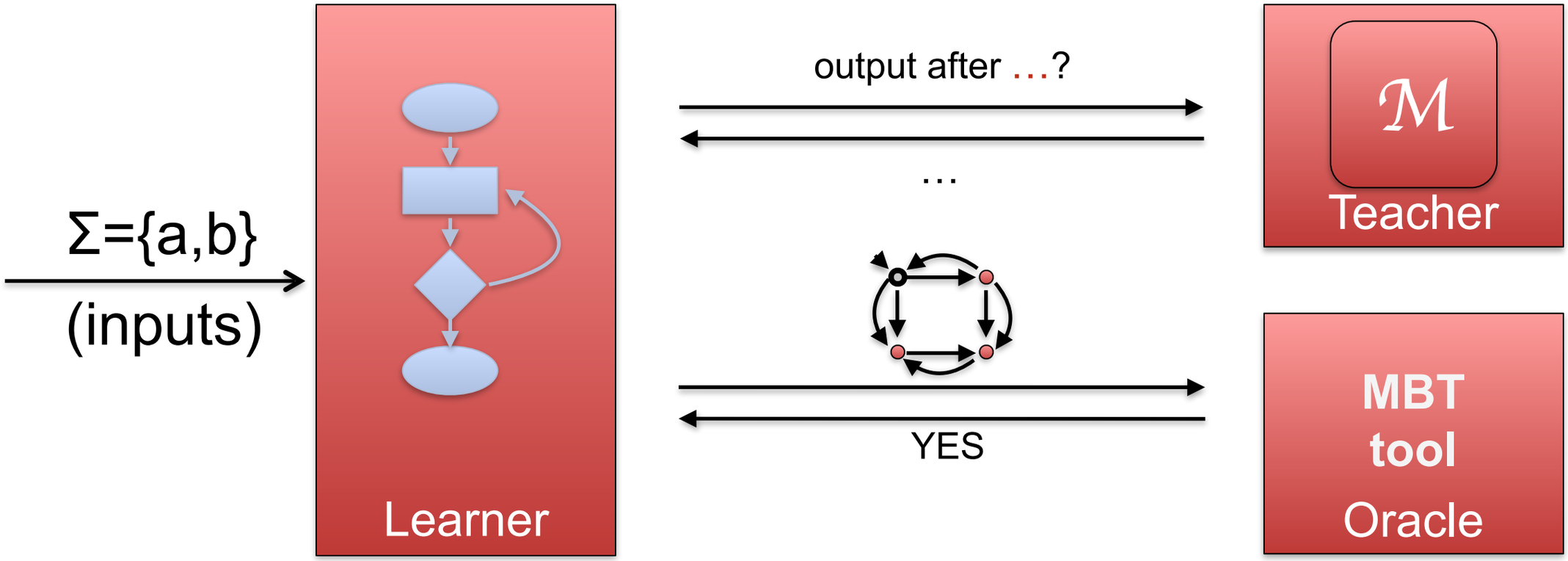
Active Automata Learning

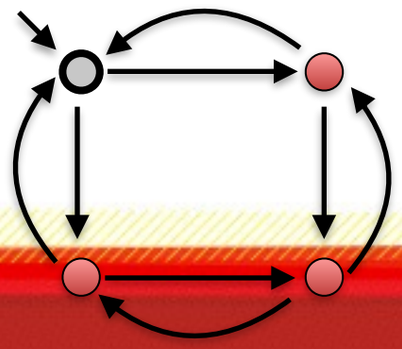
- Objective: Learn unknown regular language $\mathcal{L}$



Active Automata Learning

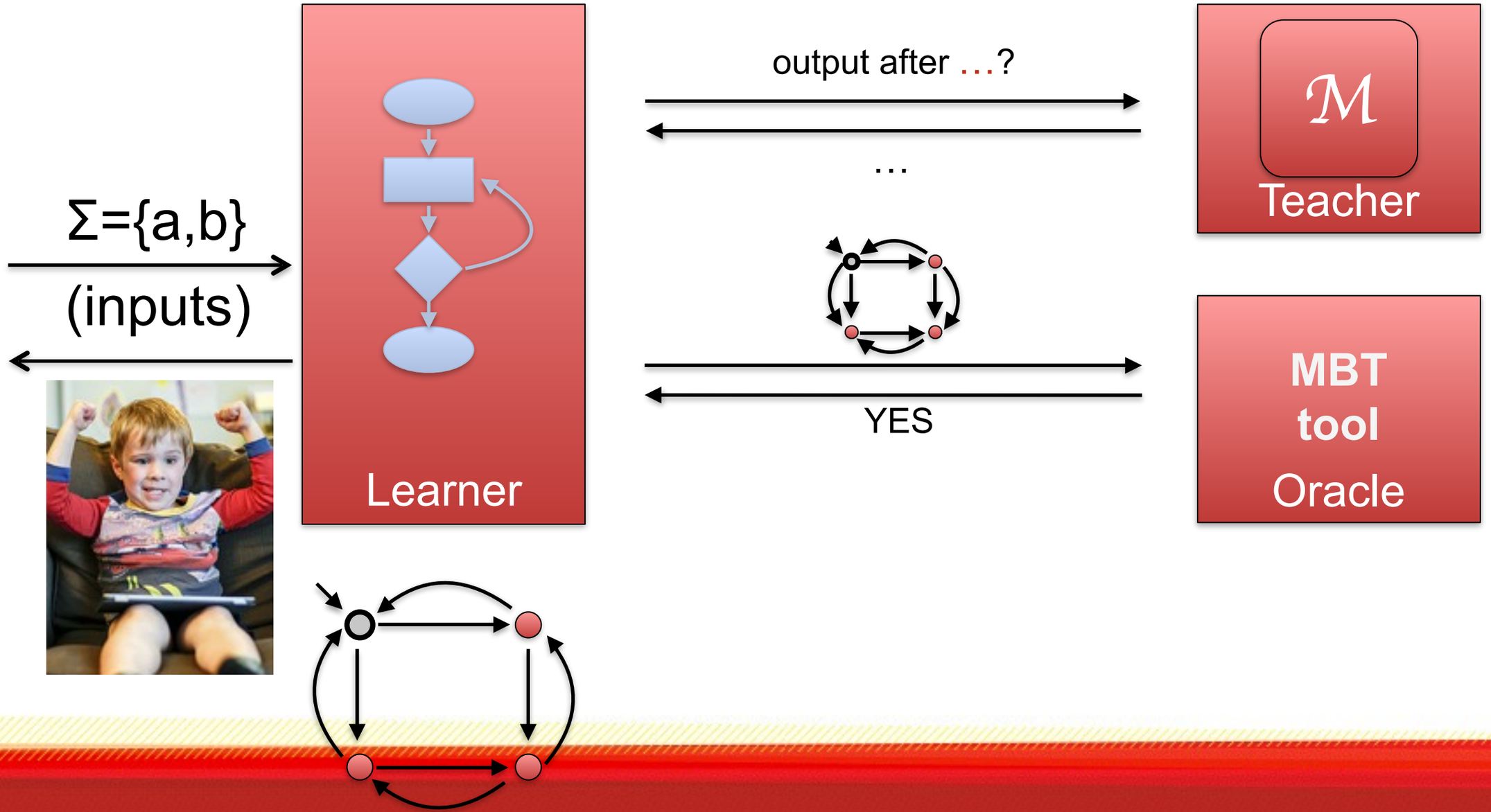
- Objective: Learn unknown regular language $\mathcal{L}$





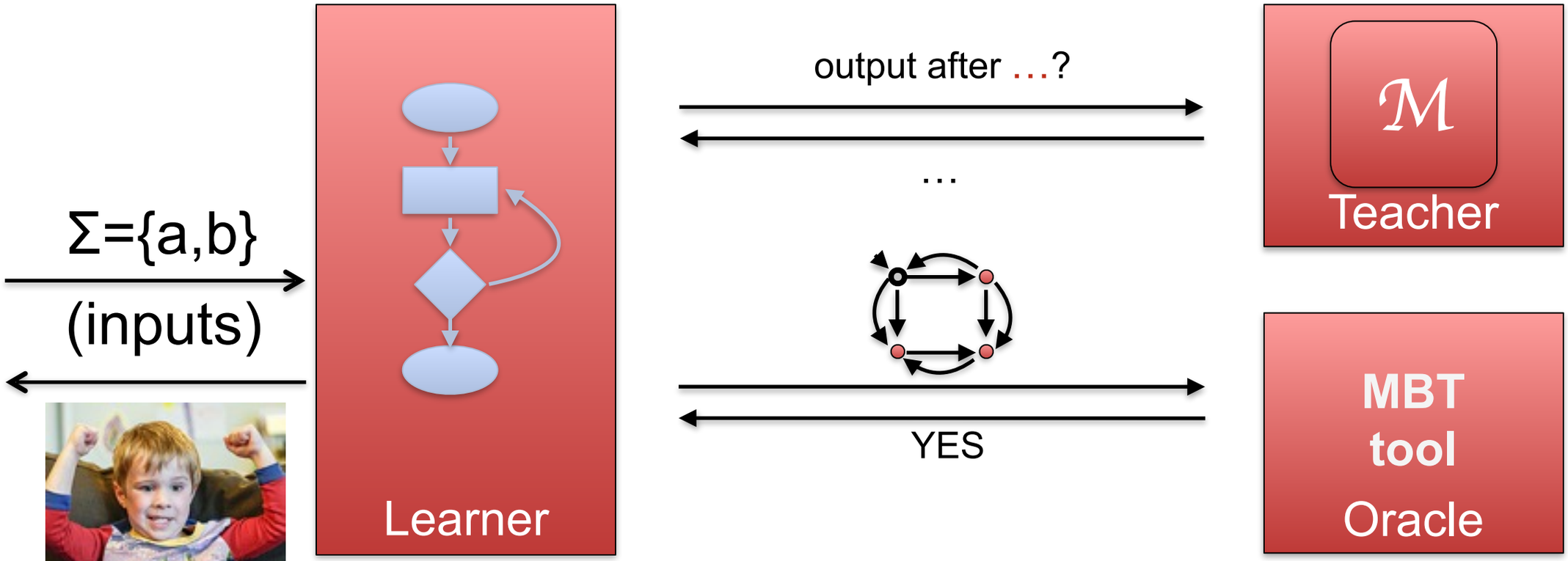
Active Automata Learning

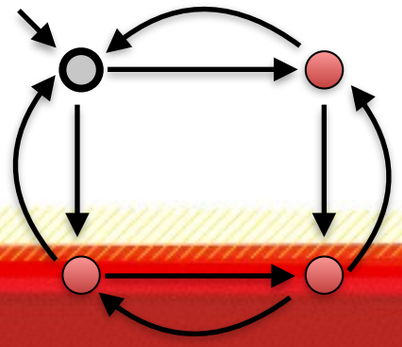
- Objective: Learn unknown regular language $\mathcal{L}$



Active Automata Learning

- Objective: Learn unknown regular language $\mathcal{L}$





A regular language can be learned with polynomially many membership queries and equivalence queries

Angluin 1987

Mealy Machines

Radboud University

Mealy machines

A Mealy machine is defined as a tuple $M = \langle S, s_0, \Sigma, \Omega, \delta, \lambda \rangle$ where

- S is a finite nonempty set of states (be $n = |S|$ the size of the Mealy machine),
- $s_0 \in S$ is the initial state,
- Σ is a finite input alphabet,
- Ω is a finite output alphabet,
- $\delta: S \times \Sigma \to S$ is the transition function, and
- $\lambda: S \times \Sigma \to \Omega$ is the output function.

Extend δ e λ for words

$\delta^* : S \times \Sigma^* \rightarrow S$:

$\delta^*(s,\varepsilon) = s$

$\delta^*(s, \alpha w) = \delta^*(\delta(s, \alpha), w)$

$\lambda^* : S \times \Sigma^* \rightarrow \Omega$:

$\lambda^*(s,\varepsilon) = \varnothing$

$\lambda^*(s, w\alpha) = \lambda(\delta^*(s, w), \alpha)$

A coffe machine

(a) empty

(b) with pod

(c) with water

(d) with pod and water

(e) success

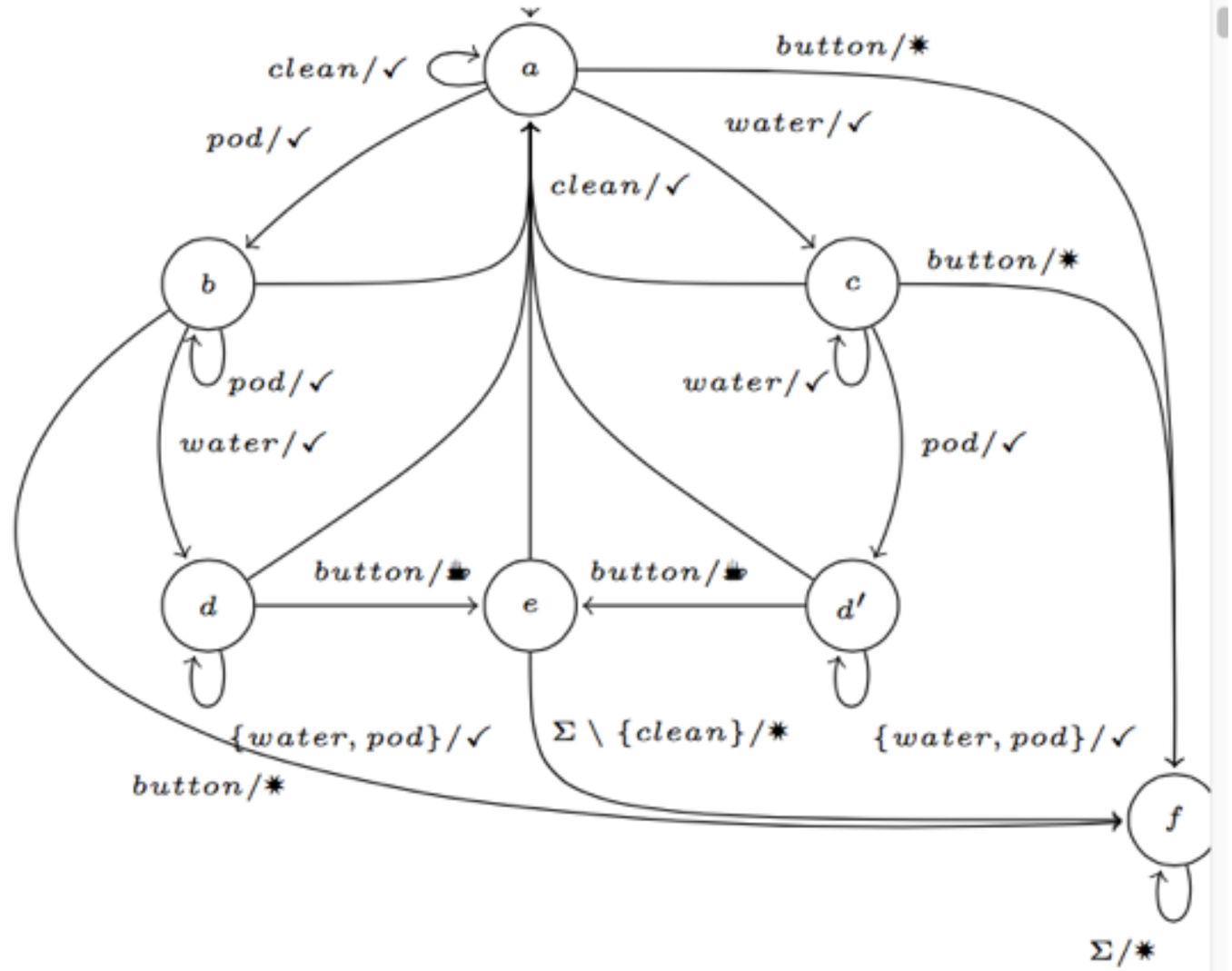
(f) error

A coffe machine

- S = {a,b,c,d,e,f}
- s_0 = a

- Σ = {water, pod, clean, button}

- Ω = {ok, coffee, error}



Nerode Relation

Radboud University

Nerode relation

Let $\mathcal{P}$: $\Sigma^* \rightarrow \Omega$ (let's define a Mealy machine just as a function that, given a word, provides the single output observed after having that word in input)

Definition (Equivalence of words w.r.t. $P\mathcal{L}$)

Two words u,u' $\in \Sigma^*$ are equivalent w.r.t. $\equiv_{\mathcal{P}}$, if and only if for all continuations v $\in \Sigma^*$, the concatenated words uv and u'v are mapped to the same output by $\mathcal{P}$:

$$u \equiv_{\mathcal{P}} u' \Leftrightarrow (\forall v \in \Sigma^* \,.\, \mathcal{P}(uv) = \mathcal{P}(u'v))$$

The idea is that each equivalence class (over $\equiv_{\mathcal{P}}$) represents a state in the hypothesis. Agree?

Finding hypothesis (based on Nerode relation)

- Structure maintained by the learner: Observation Table

Columns are labelled
by suffixes (**suffix closed**, start with Σ)

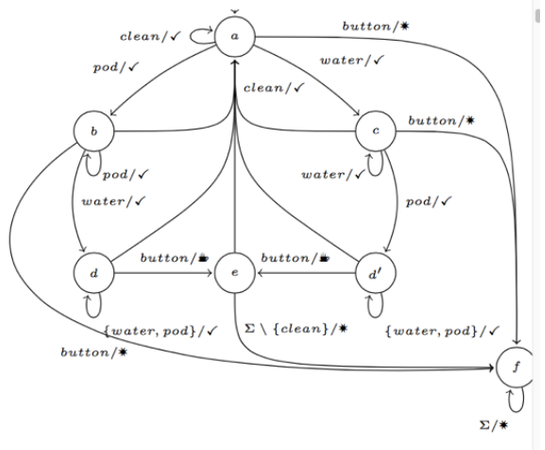
Learner	*w*	*p*	*c*	*b*
ε				
w				
p				
c				
b				

Top part: states

Bottom part: one letter extensions

Cell: output query for
"prefix.suffix"

Rows are labelled
by prefixes
(**prefix closed**)



Teacher

Finding hypothesis (based on Nerode relation)

- Structure maintained by the learner: Observation Table

Columns are labelled
by suffixes (suffix closed, start with Σ)

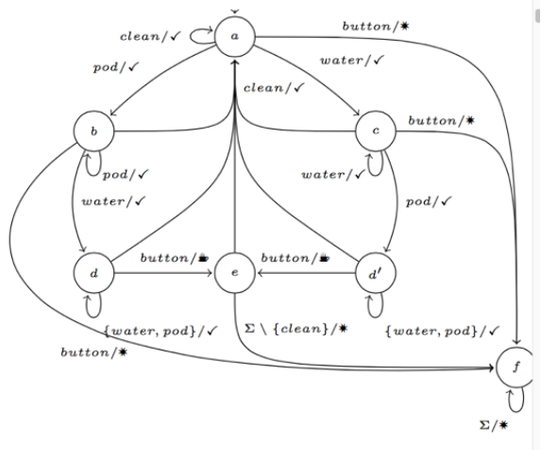
Learner	w	p	c	b
ε	ok	ok	ok	e
w	ok	ok	ok	e
p	ok	ok	ok	e
c	ok	ok	ok	e
b	e	e	e	e

Top part: states

Bottom part: one letter extensions

Cell: output query for
“prefix.suffix”

Rows are labelled
by prefixes
(prefix closed)



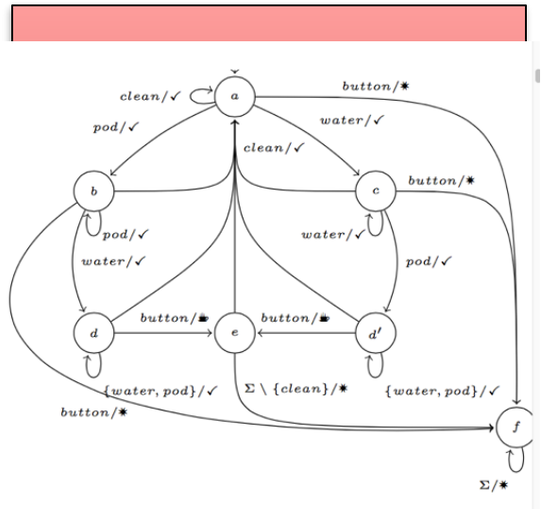
Teacher

Finding hypothesis (based on Nerode relation)

- Structure maintained by the learner: Observation Table

Learner	w	p	c	b
ε	ok	ok	ok	e
w	ok	ok	ok	e
p	ok	ok	ok	e
c	ok	ok	ok	e
b	e	e	e	e

not equivalent (Nerode)



Teacher

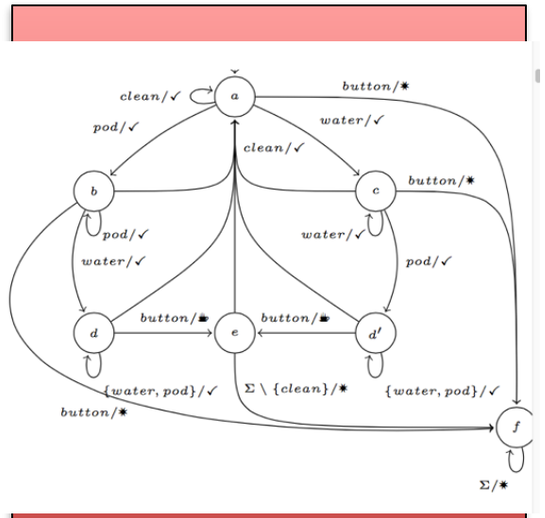
Move row b to the top part of the table!

Finding hypothesis (based on Nerode relation)

- Structure maintained by the learner: Observation Table

	w	*p*	*c*	*b*
ε	***ok***	***ok***	***ok***	***e***
b	***e***	***e***	***e***	***e***
w	***ok***	***ok***	***ok***	***e***
p	***ok***	***ok***	***ok***	***e***
c	***ok***	***ok***	***ok***	***e***

States

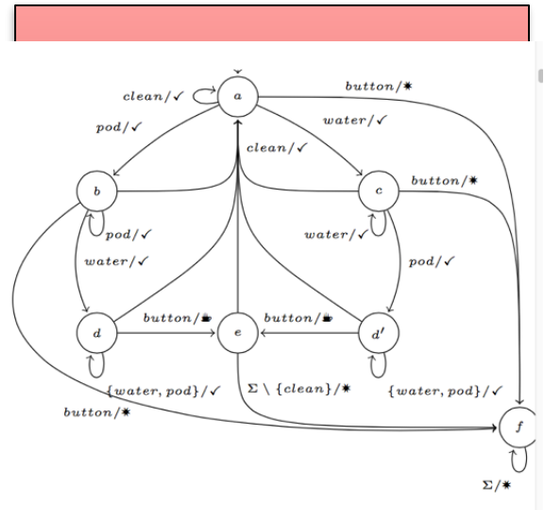


Teacher

Finding hypothesis (based on Nerode relation)

- Structure maintained by the learner: Observation Table

	w	*p*	*c*	*b*
ε	*ok*	*ok*	*ok*	*e*
b	*e*	*e*	*e*	*e*
w	*ok*	*ok*	*ok*	*e*
p	*ok*	*ok*	*ok*	*e*
c	*ok*	*ok*	*ok*	*e*

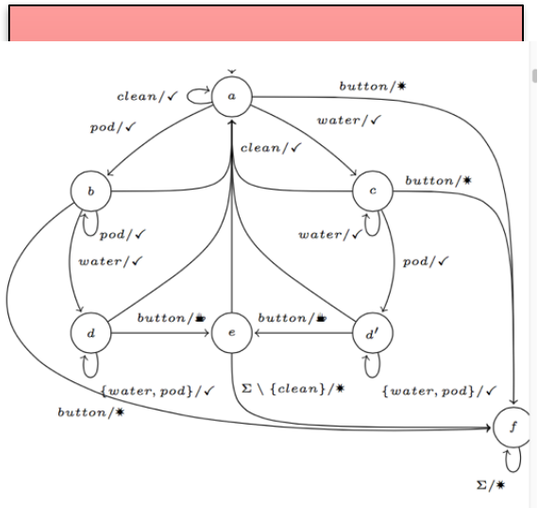


Teacher

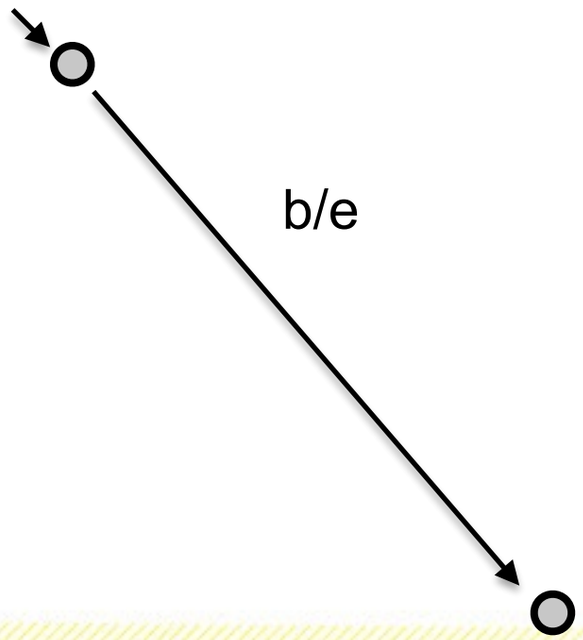
Finding hypothesis (based on Nerode relation)

- Structure maintained by the learner: Observation Table

	w	*p*	*c*	*b*
ε	*ok*	*ok*	*ok*	*e*
b	*e*	*e*	*e*	*e*
w	*ok*	*ok*	*ok*	*e*
p	*ok*	*ok*	*ok*	*e*
c	*ok*	*ok*	*ok*	*e*



Teacher

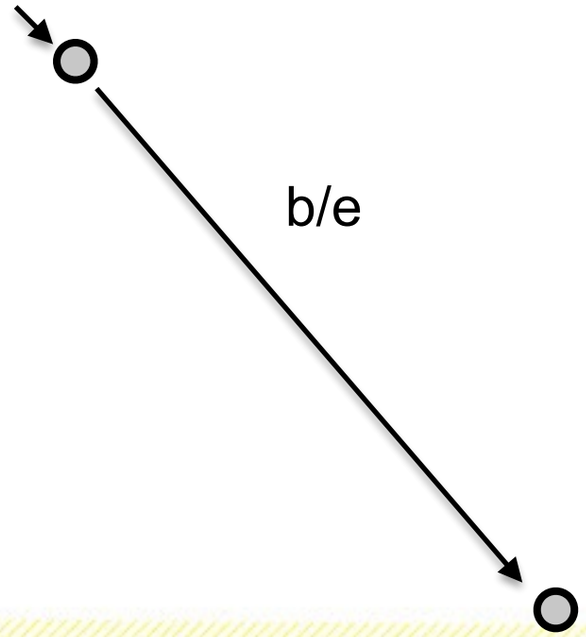


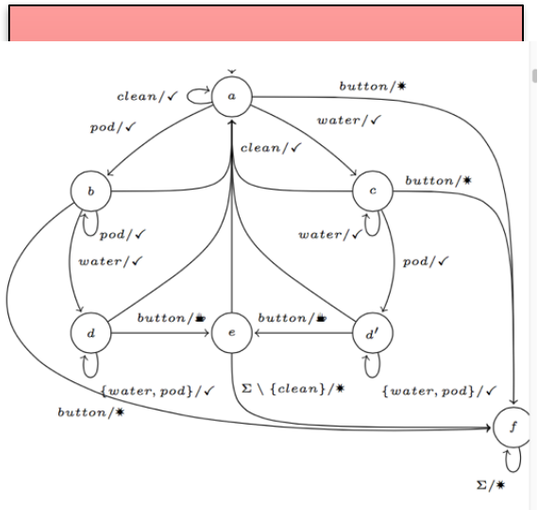
Finding hypothesis (based on Nerode relation)

- Structure maintained by the learner: Observation Table

	w	*p*	*c*	*b*
ε	*ok*	*ok*	*ok*	*e*
b	*e*	*e*	*e*	*e*
w	*ok*	*ok*	*ok*	*e*
p	*ok*	*ok*	*ok*	*e*
c	*ok*	*ok*	*ok*	*e*

Transitions
(target state in according to Nerode relation)





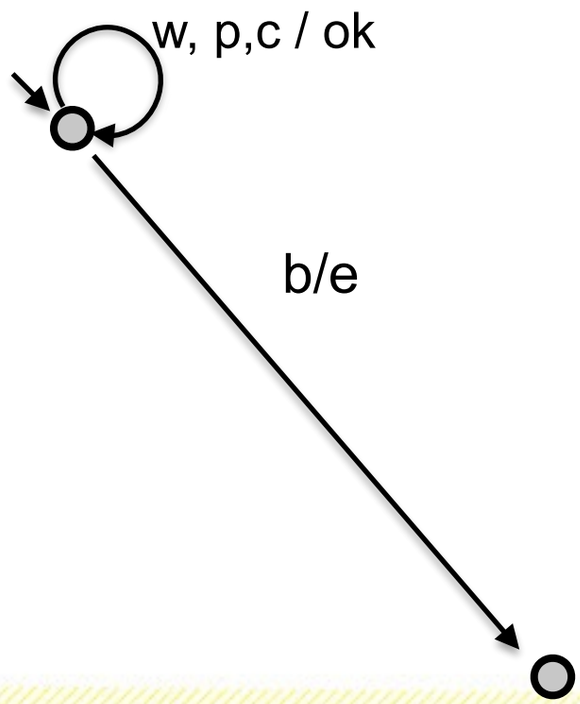
Teacher

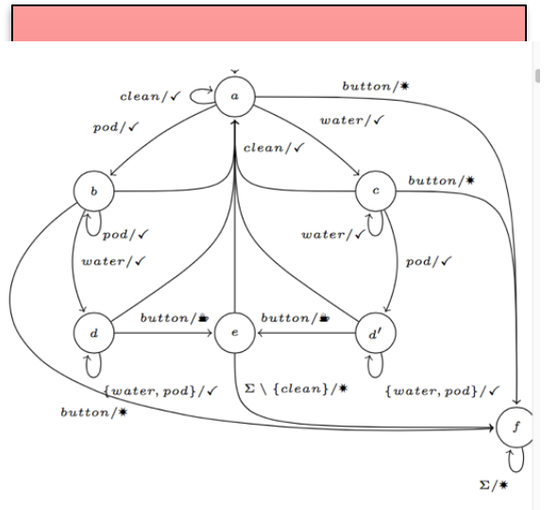
Finding hypothesis (based on Nerode relation)

- Structure maintained by the learner: Observation Table

	w	*p*	*c*	*b*
ε	*ok*	*ok*	*ok*	*e*
b	*e*	*e*	*e*	*e*
w	*ok*	*ok*	*ok*	*e*
p	*ok*	*ok*	*ok*	*e*
c	*ok*	*ok*	*ok*	*e*

Transitions
(target state in according to Nerode relation)





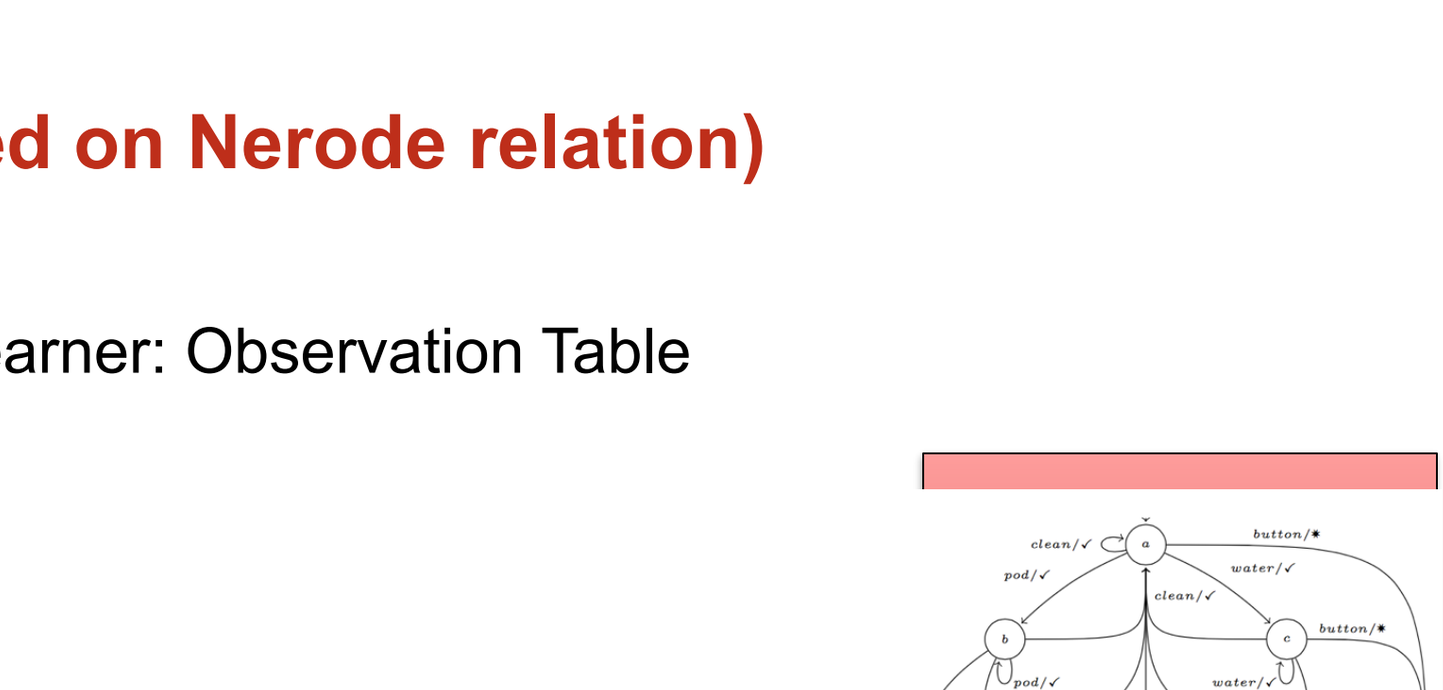
Teacher

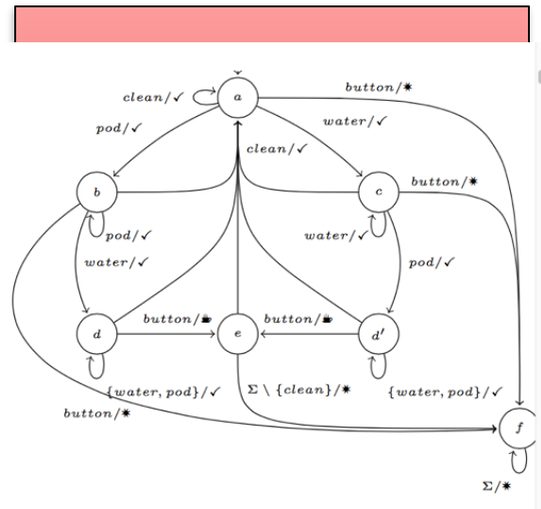
Finding hypothesis (based on Nerode relation)

- Structure maintained by the learner: Observation Table

	w	*p*	*c*	*b*
ε	*ok*	*ok*	*ok*	*e*
b	*e*	*e*	*e*	*e*
w	*ok*	*ok*	*ok*	*e*
p	*ok*	*ok*	*ok*	*e*
c	*ok*	*ok*	*ok*	*e*

Transitions
(target state in according to Nerode relation)





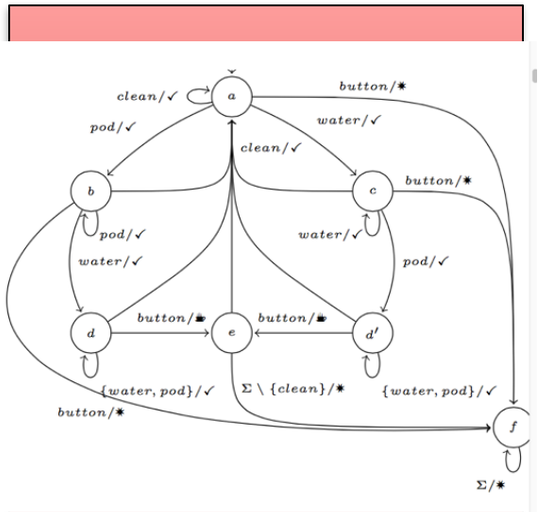
Teacher

Finding hypothesis (based on Nerode relation)

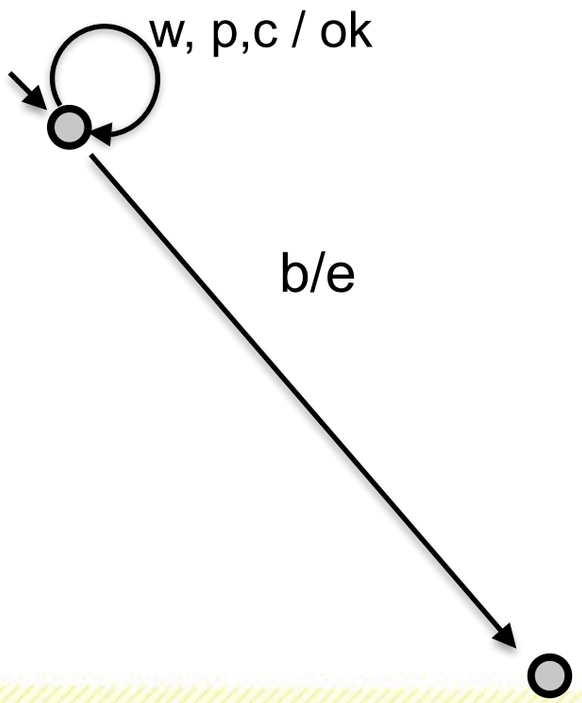
- Structure maintained by the learner: Observation Table

	w	*p*	*c*	*b*
ε	*ok*	*ok*	*ok*	*e*
b	*e*	*e*	*e*	*e*
w	*ok*	*ok*	*ok*	*e*
p	*ok*	*ok*	*ok*	*e*
c	*ok*	*ok*	*ok*	*e*
bw				
bp				
bc				
bb				

New rows



Teacher

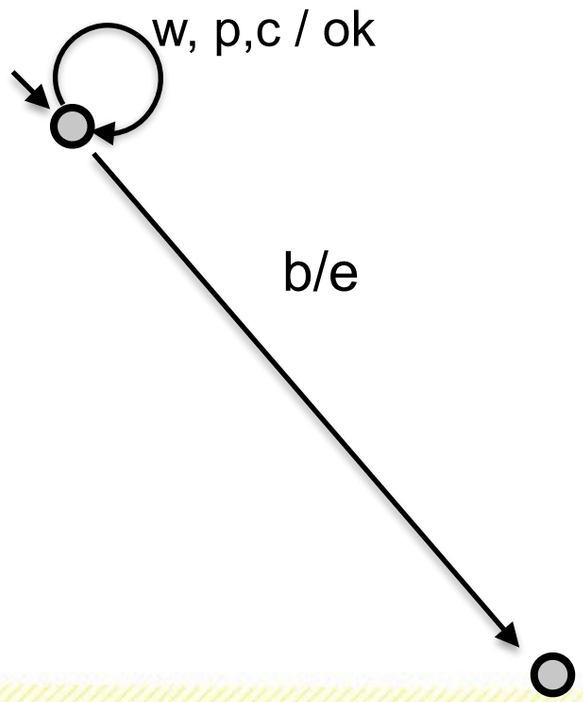


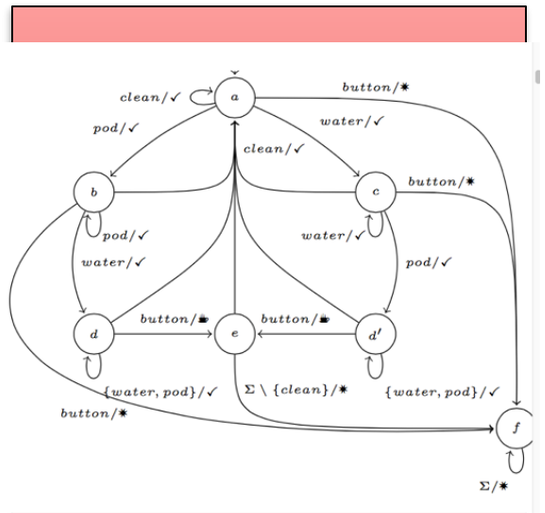
Finding hypothesis (based on Nerode relation)

- Structure maintained by the learner: Observation Table

	w	*p*	*c*	*b*
ε	*ok*	*ok*	*ok*	*e*
b	*e*	*e*	*e*	*e*
w	*ok*	*ok*	*ok*	*e*
p	*ok*	*ok*	*ok*	*e*
c	*ok*	*ok*	*ok*	*e*
bw	*e*	*e*	*e*	*e*
bp	*e*	*e*	*e*	*e*
bc	*e*	*e*	*e*	*e*
bb	*e*	*e*	*e*	*e*

New rows





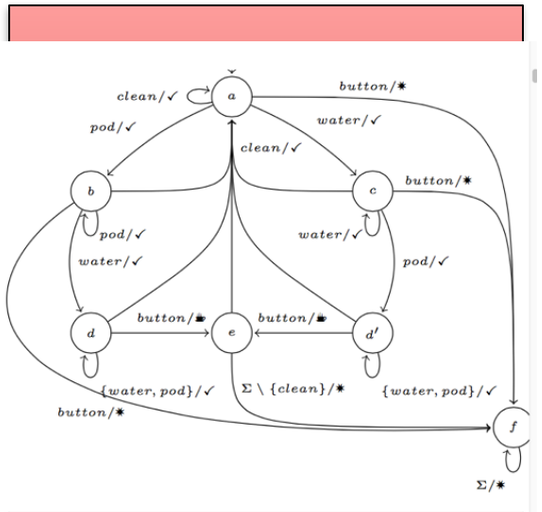
Teacher

Finding hypothesis (based on Nerode relation)

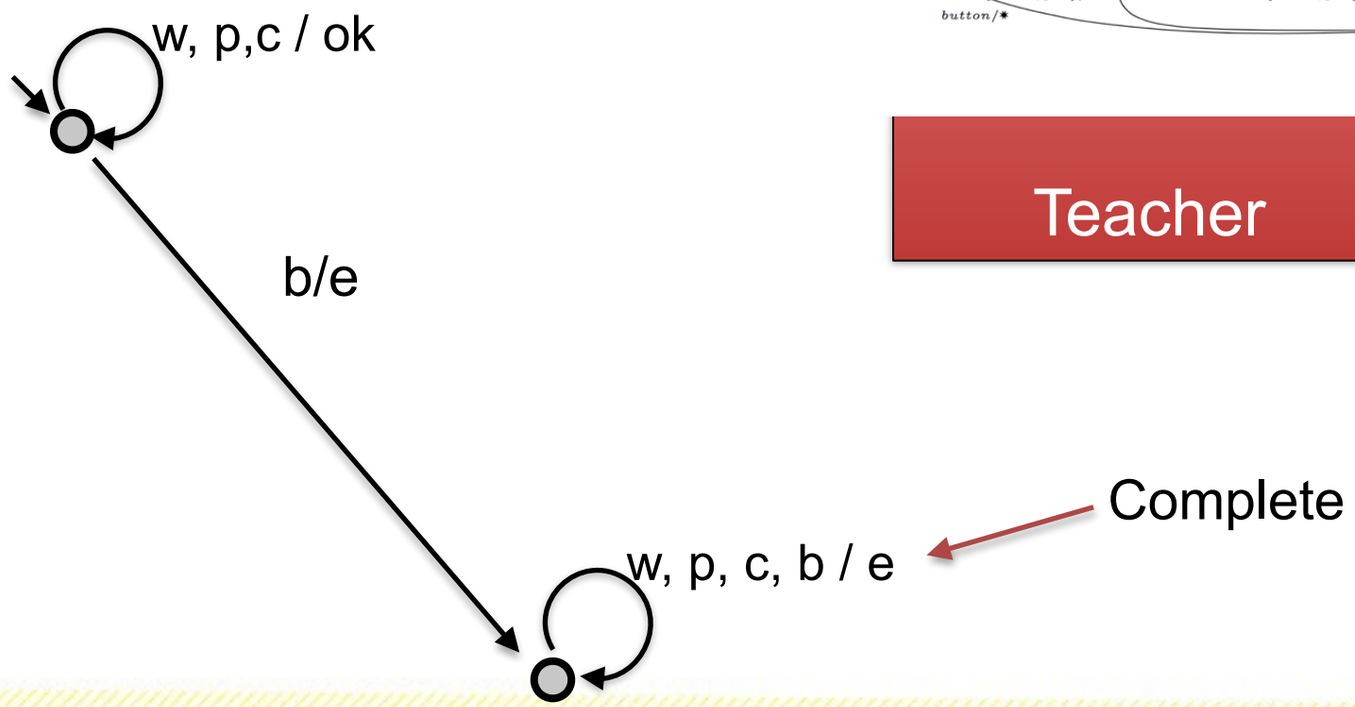
- Structure maintained by the learner: Observation Table

	w	*p*	*c*	*b*
ε	*ok*	*ok*	*ok*	*e*
b	*e*	*e*	*e*	*e*
w	*ok*	*ok*	*ok*	*e*
p	*ok*	*ok*	*ok*	*e*
c	*ok*	*ok*	*ok*	*e*
bw	*e*	*e*	*e*	*e*
bp	*e*	*e*	*e*	*e*
bc	*e*	*e*	*e*	*e*
bb	*e*	*e*	*e*	*e*

New rows



Teacher

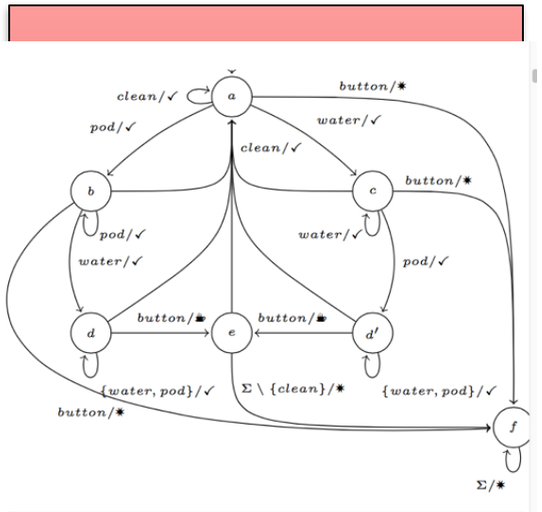


Finding hypothesis (based on Nerode relation)

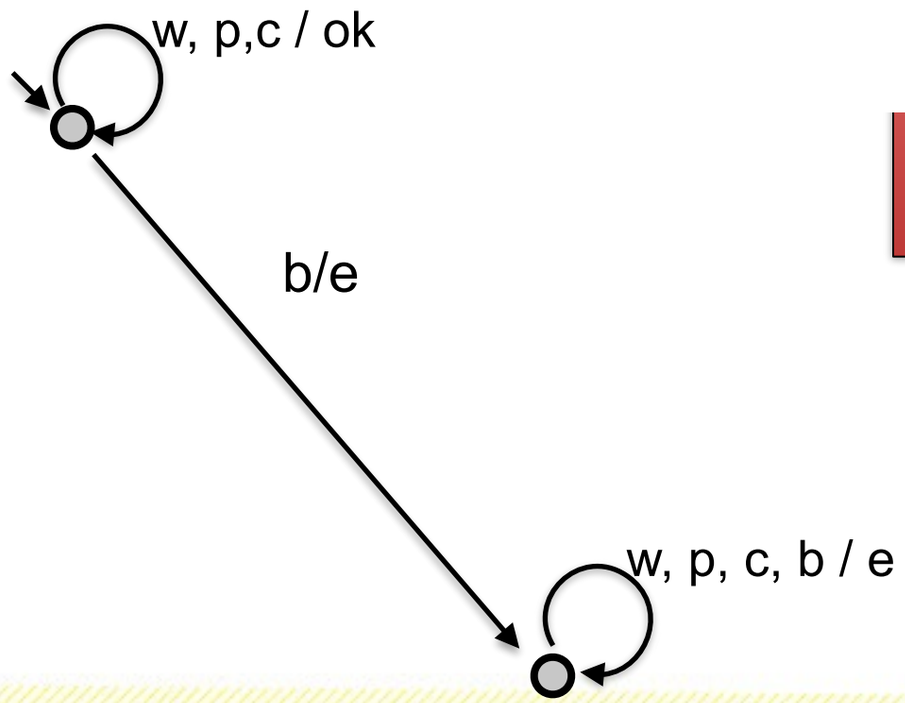
- Structure maintained by the learner: Observation Table

	w	*p*	*c*	*b*
ε	*ok*	*ok*	*ok*	*e*
b	*e*	*e*	*e*	*e*
w	*ok*	*ok*	*ok*	*e*
p	*ok*	*ok*	*ok*	*e*
c	*ok*	*ok*	*ok*	*e*
bw	*e*	*e*	*e*	*e*
bp	*e*	*e*	*e*	*e*
bc	*e*	*e*	*e*	*e*
bb	*e*	*e*	*e*	*e*

No new row should be moved to the top, we say that the table is *Closed*.



Teacher



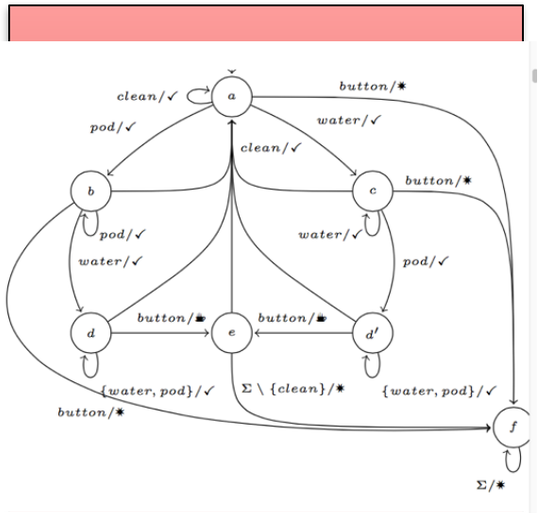
Finding hypothesis (based on Nerode relation)

- Structure maintained by the learner: Observation Table

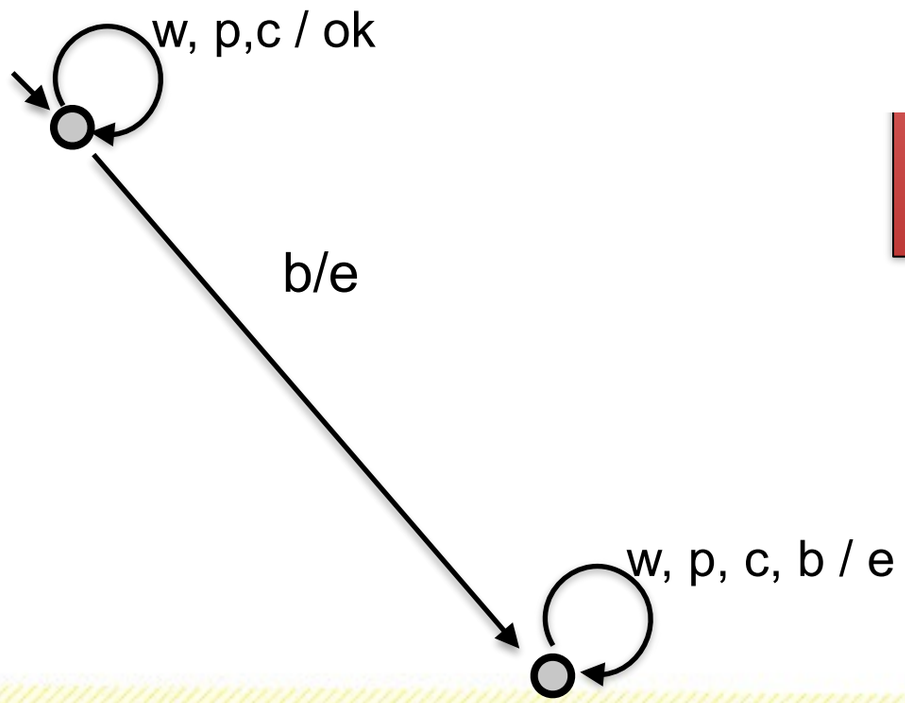
	w	*p*	*c*	*b*
ε	*ok*	*ok*	*ok*	*e*
b	*e*	*e*	*e*	*e*
w	*ok*	*ok*	*ok*	*e*
p	*ok*	*ok*	*ok*	*e*
c	*ok*	*ok*	*ok*	*e*
bw	*e*	*e*	*e*	*e*
bp	*e*	*e*	*e*	*e*
bc	*e*	*e*	*e*	*e*
bb	*e*	*e*	*e*	*e*

No new row should be moved to the top, we say that the table is *Closed*.

Equivalence query



Teacher

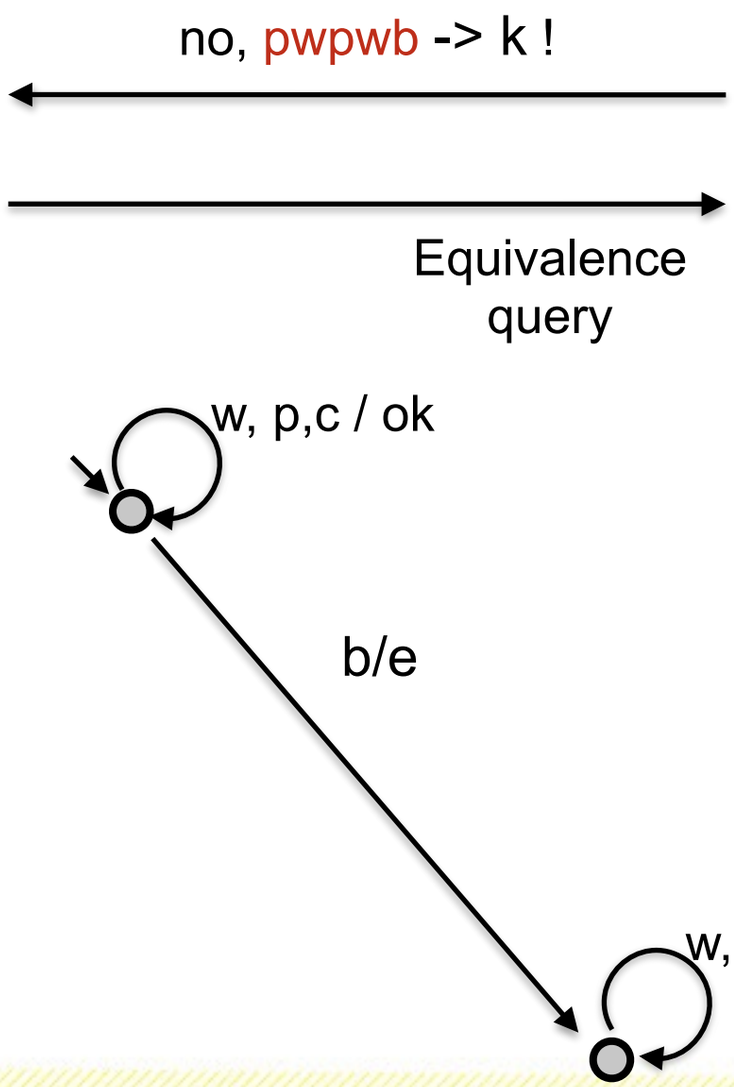


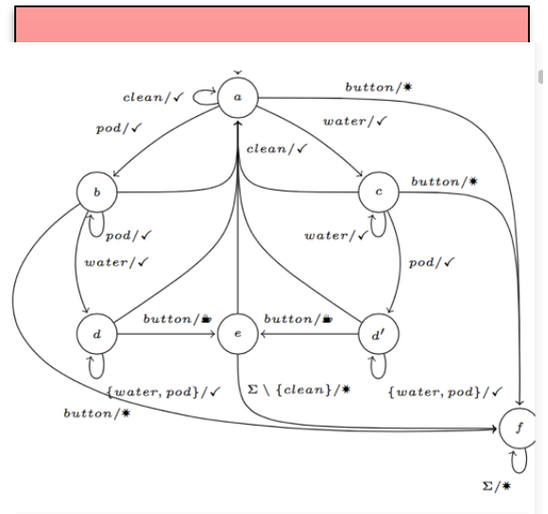
Finding hypothesis (based on Nerode relation)

- Structure maintained by the learner: Observation Table

	w	*p*	*c*	*b*
ε	*ok*	*ok*	*ok*	*e*
b	*e*	*e*	*e*	*e*
w	*ok*	*ok*	*ok*	*e*
p	*ok*	*ok*	*ok*	*e*
c	*ok*	*ok*	*ok*	*e*
bw	*e*	*e*	*e*	*e*
bp	*e*	*e*	*e*	*e*
bc	*e*	*e*	*e*	*e*
bb	*e*	*e*	*e*	*e*

No new row should be moved to the top, we say that the table is *Closed*.





Teacher

On counterexample handling

- “A counterexample provides a suffix such that Nerode relation is proven not valid in our hypothesis”.
- There are several ways to handle a counterexample.
- For the moment: a counterexample contains a new column label (a suffix) that will lead to a new state.
- i.e. a label that will disprove Nerode equivalence between (at least) two rows. (more on this later)
- In this case: “wb”.

On counterexampl

FSM Transition Testing

- State verification :
- UIO sequences
  - sequence x that distinguishes state s from all other states : for all $t \neq s$: $\lambda(s,x) \neq \lambda(t,x)$
  - each state has its own UIO sequence
  - UIO sequences may not exist
- Distinguishing sequence
  - sequence x that produces different output for each state : for all pairs t,s with $t \neq s$: $\lambda(s,x) \neq \lambda(t,x)$
  - a distinguishing sequence may not exist
- W - set of sequences
  - set of sequences W which can distinguish any pair of states : for all pairs $t \neq s$ there is $x \in W$: $\lambda(s,x) \neq \lambda(t,x)$
  - W - set always exists for reduced FSM

© Jan Tretmans Radboud University Nijmegen 20

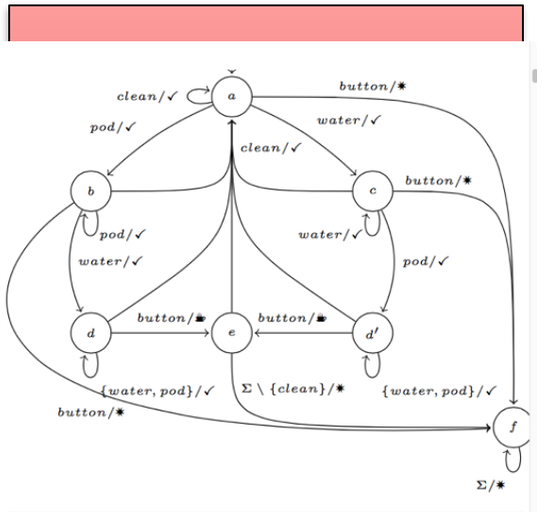
- For the moment: a counterexample contains a new column label (a suffix) that will lead to a new state.

- I.e. a label that will disprove Nerode equivalence between (at least) two rows. (more on this later)

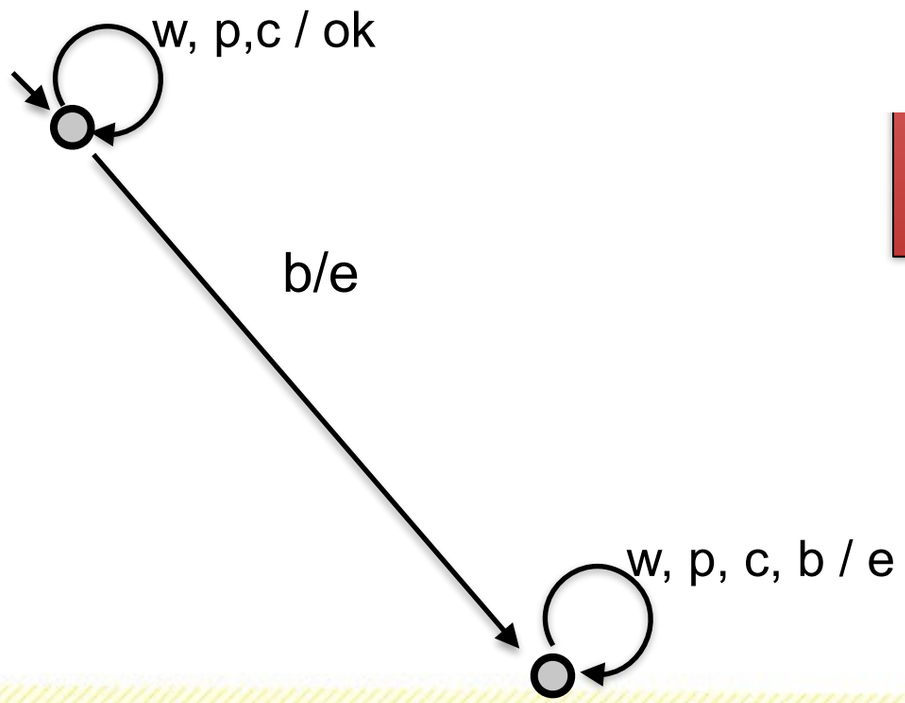
Refining hypothesis (based on Nerode relation)

	w	*p*	*c*	*b*	*wb*
ε	*ok*	*ok*	*ok*	*e*	
b	*e*	*e*	*e*	*e*	
w	*ok*	*ok*	*ok*	*e*	
p	*ok*	*ok*	*ok*	*e*	
c	*ok*	*ok*	*ok*	*e*	
bw	*e*	*e*	*e*	*e*	
bp	*e*	*e*	*e*	*e*	
bc	*e*	*e*	*e*	*e*	
bb	*e*	*e*	*e*	*e*	

New suffix



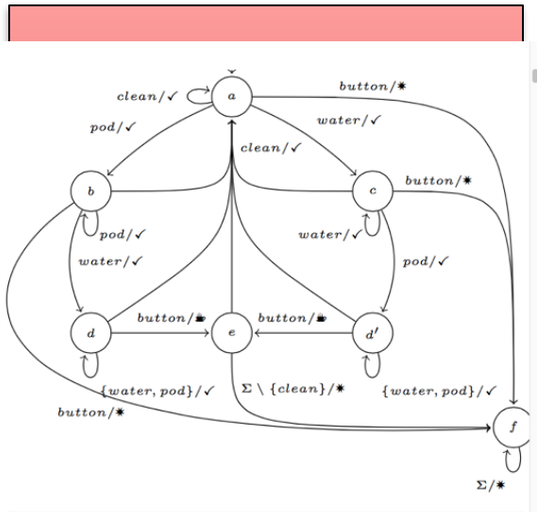
Teacher



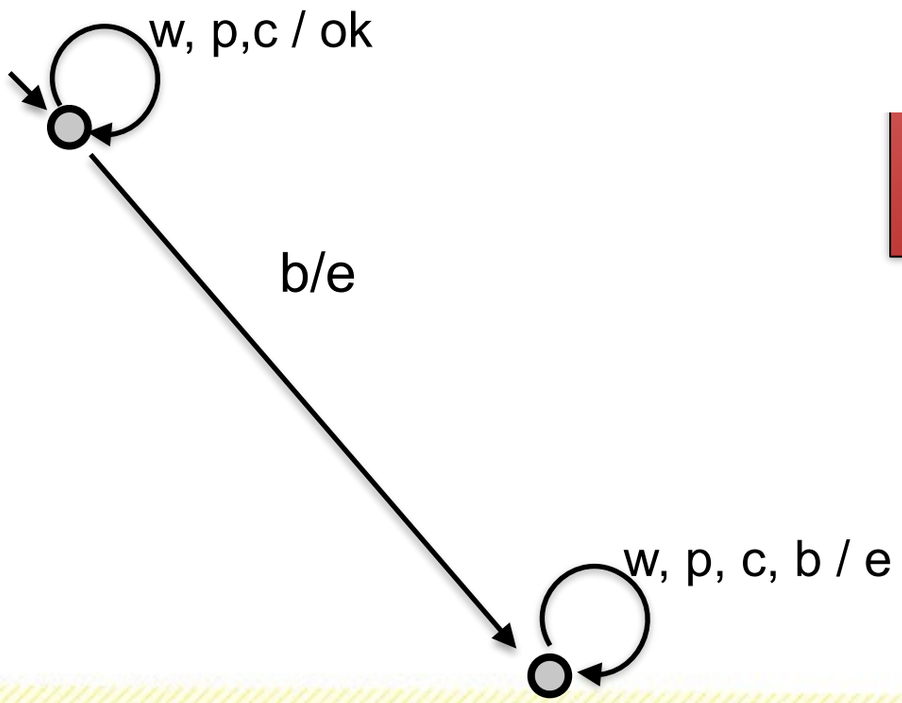
Refining hypothesis (based on Nerode relation)

	w	*p*	*c*	*b*	*wb*
ε	*ok*	*ok*	*ok*	*e*	*e*
b	*e*	*e*	*e*	*e*	*e*
w	*ok*	*ok*	*ok*	*e*	*e*
p	*ok*	*ok*	*ok*	*e*	*k*
c	*ok*	*ok*	*ok*	*e*	*e*
bw	*e*	*e*	*e*	*e*	*e*
bp	*e*	*e*	*e*	*e*	*e*
bc	*e*	*e*	*e*	*e*	*e*
bb	*e*	*e*	*e*	*e*	*e*

New suffix



Teacher

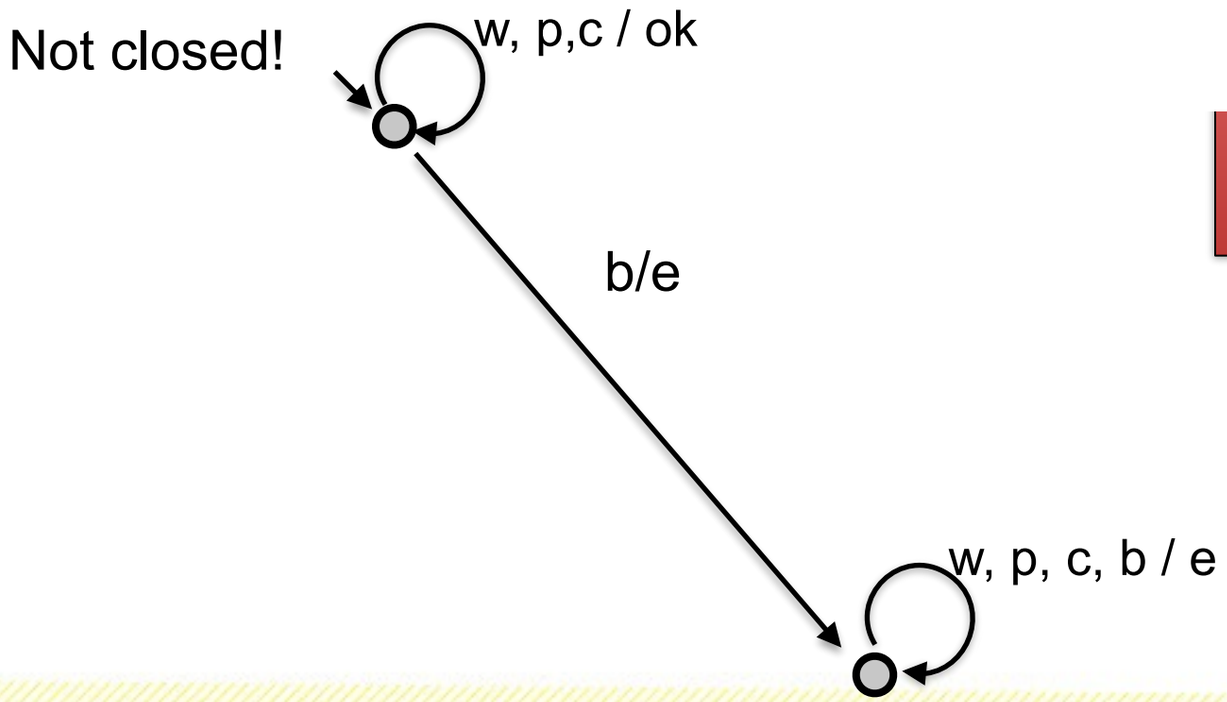


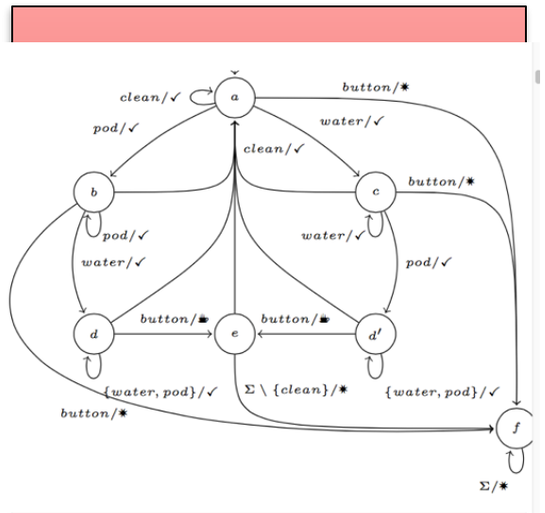
Refining hypothesis (based on Nerode relation)

	w	*p*	*c*	*b*	*wb*
ε	*ok*	*ok*	*ok*	*e*	*e*
b	*e*	*e*	*e*	*e*	*e*
w	*ok*	*ok*	*ok*	*e*	*e*
p	*ok*	*ok*	*ok*	*e*	*k*
c	*ok*	*ok*	*ok*	*e*	*e*
bw	*e*	*e*	*e*	*e*	*e*
bp	*e*	*e*	*e*	*e*	*e*
bc	*e*	*e*	*e*	*e*	*e*
bb	*e*	*e*	*e*	*e*	*e*

New suffix

Not closed!



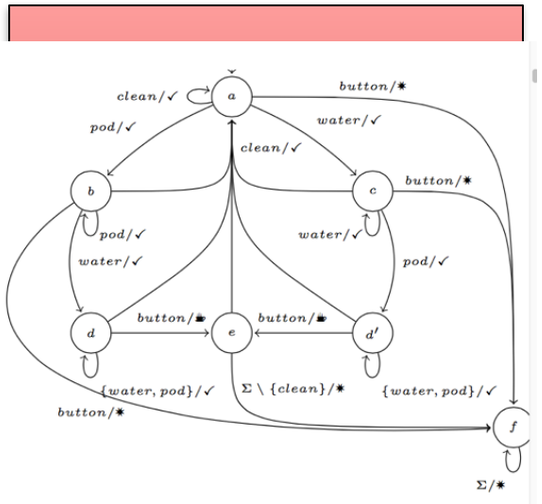


Teacher

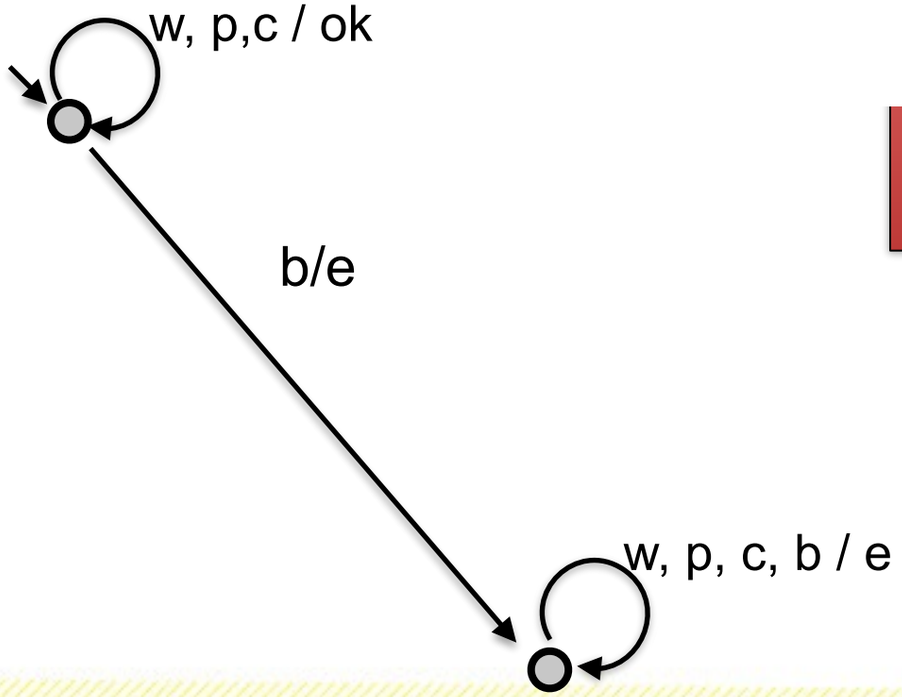
Refining hypothesis (based on Nerode relation)

	w	*p*	*c*	*b*	*wb*
ε	*ok*	*ok*	*ok*	*e*	*e*
b	*e*	*e*	*e*	*e*	*e*
p	*ok*	*ok*	*ok*	*e*	*k*
w	*ok*	*ok*	*ok*	*e*	*e*
c	*ok*	*ok*	*ok*	*e*	*e*
bw	*e*	*e*	*e*	*e*	*e*
bp	*e*	*e*	*e*	*e*	*e*
bc	*e*	*e*	*e*	*e*	*e*
bb	*e*	*e*	*e*	*e*	*e*

New suffix



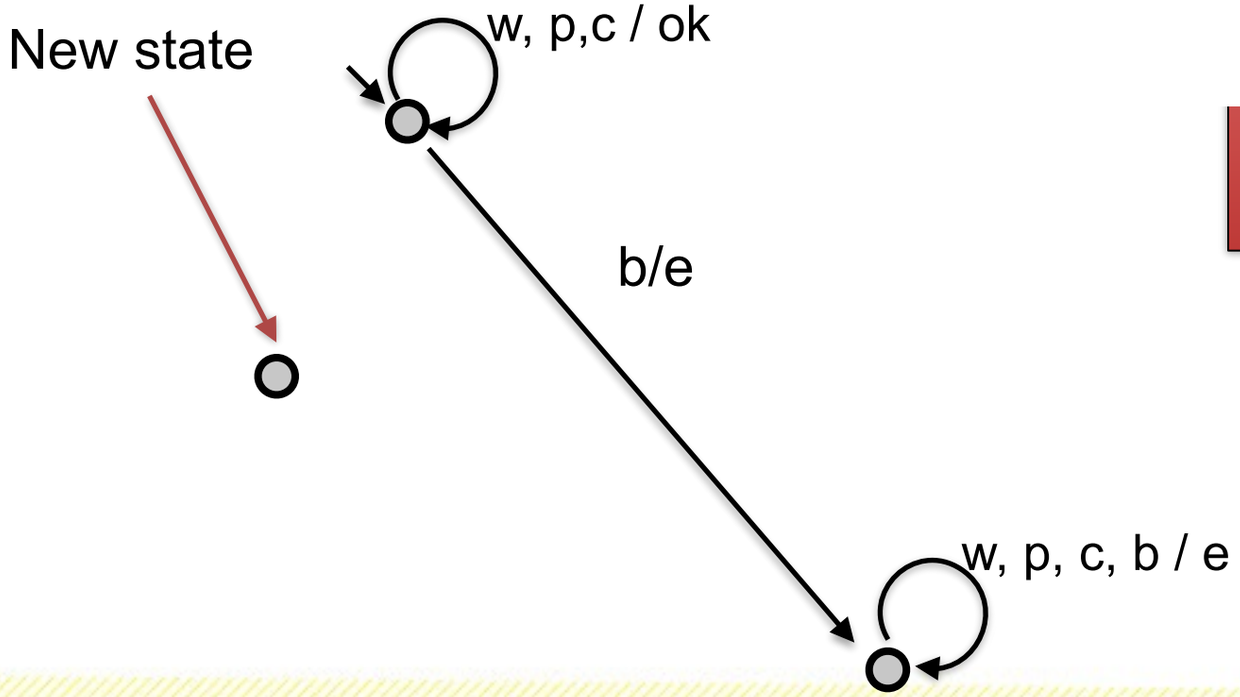
Teacher

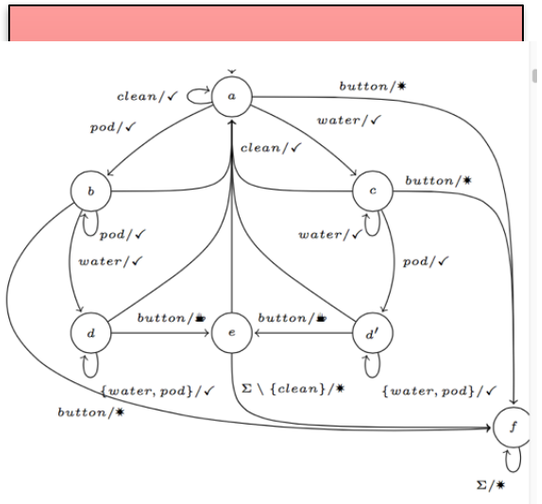


Refining hypothesis (based on Nerode relation)

	w	*p*	*c*	*b*	*wb*
ε	*ok*	*ok*	*ok*	*e*	*e*
b	*e*	*e*	*e*	*e*	*e*
p	*ok*	*ok*	*ok*	*e*	*k*
w	*ok*	*ok*	*ok*	*e*	*e*
c	*ok*	*ok*	*ok*	*e*	*e*
bw	*e*	*e*	*e*	*e*	*e*
bp	*e*	*e*	*e*	*e*	*e*
bc	*e*	*e*	*e*	*e*	*e*
bb	*e*	*e*	*e*	*e*	*e*

New suffix



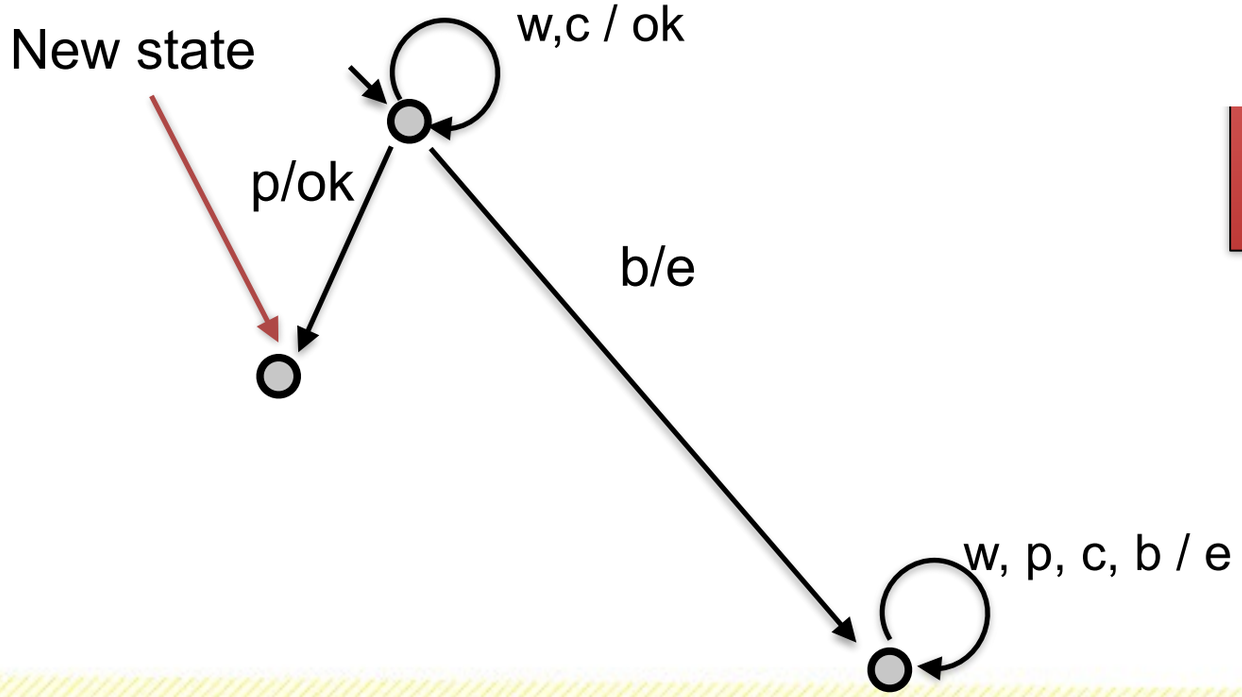


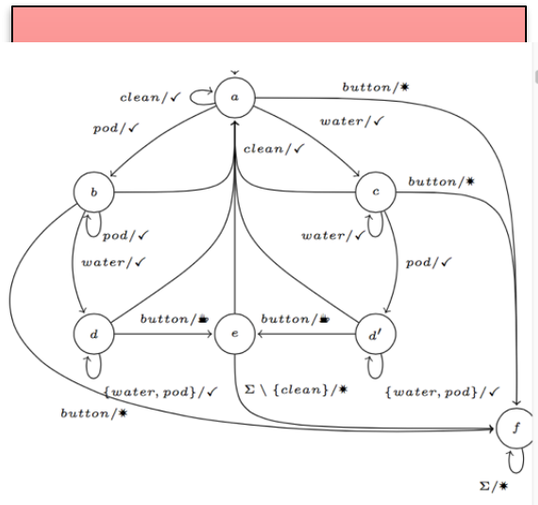
Teacher

Refining hypothesis (based on Nerode relation)

	w	*p*	*c*	*b*	*wb*
ε	*ok*	*ok*	*ok*	*e*	*e*
b	*e*	*e*	*e*	*e*	*e*
p	*ok*	*ok*	*ok*	*e*	*k*
w	*ok*	*ok*	*ok*	*e*	*e*
c	*ok*	*ok*	*ok*	*e*	*e*
bw	*e*	*e*	*e*	*e*	*e*
bp	*e*	*e*	*e*	*e*	*e*
bc	*e*	*e*	*e*	*e*	*e*
bb	*e*	*e*	*e*	*e*	*e*

New suffix



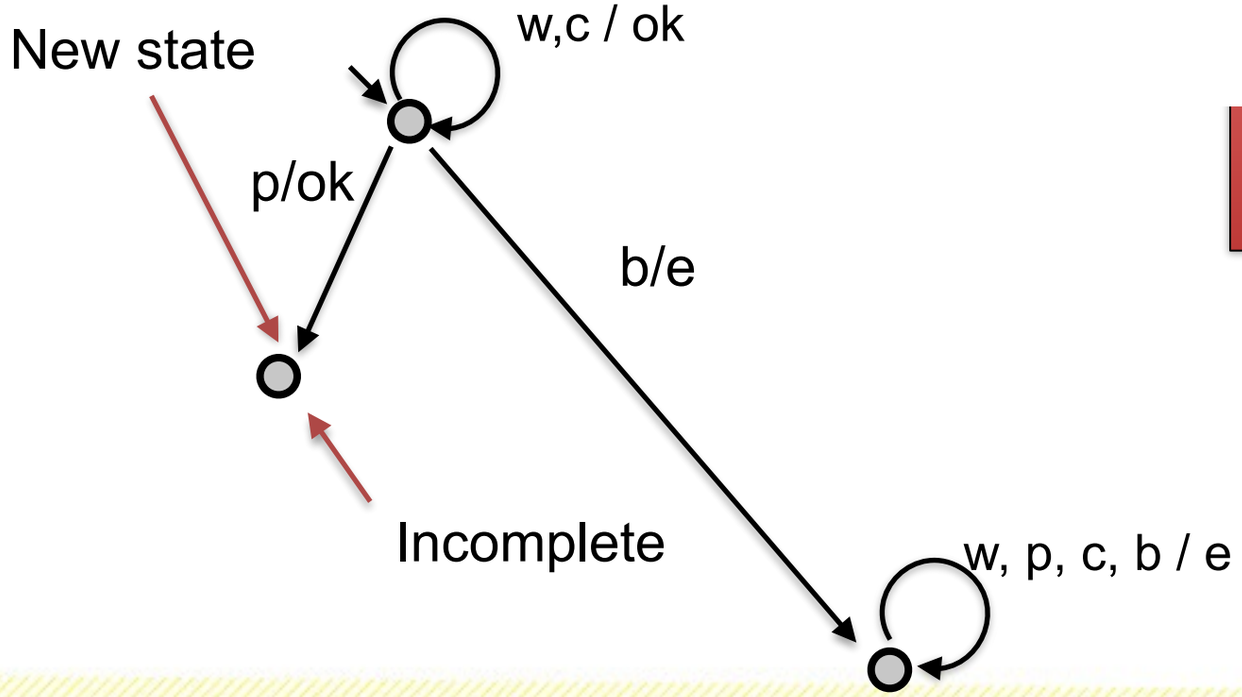


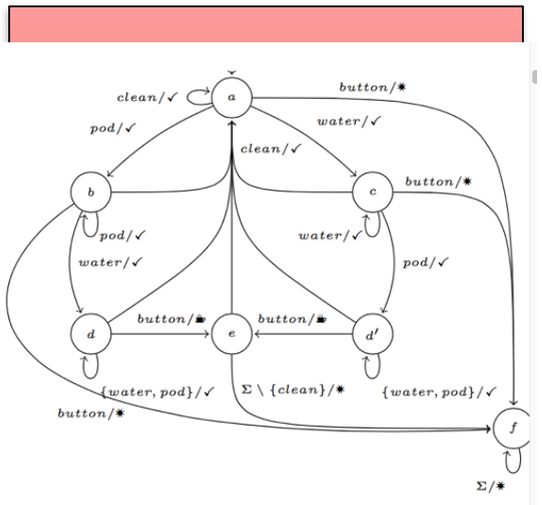
Teacher

Refining hypothesis (based on Nerode relation)

	w	*p*	*c*	*b*	*wb*
ε	*ok*	*ok*	*ok*	*e*	*e*
b	*e*	*e*	*e*	*e*	*e*
p	*ok*	*ok*	*ok*	*e*	*k*
w	*ok*	*ok*	*ok*	*e*	*e*
c	*ok*	*ok*	*ok*	*e*	*e*
bw	*e*	*e*	*e*	*e*	*e*
bp	*e*	*e*	*e*	*e*	*e*
bc	*e*	*e*	*e*	*e*	*e*
bb	*e*	*e*	*e*	*e*	*e*

New suffix



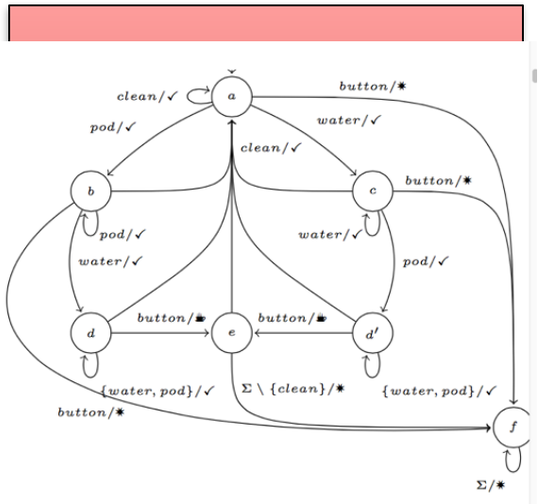


Teacher

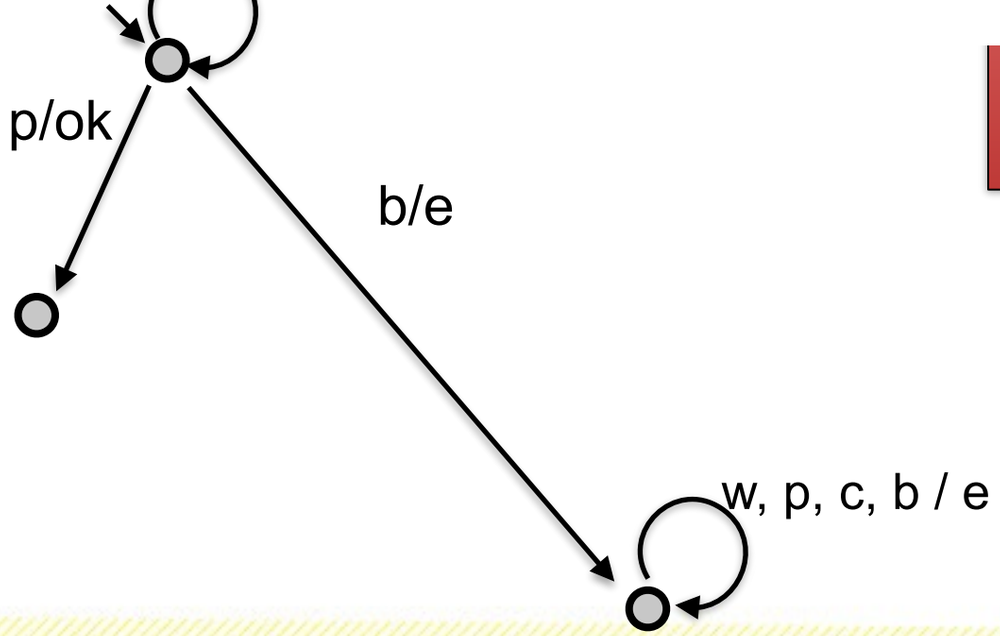
Refining hypothesis (based on Nerode relation)

	w	*p*	*c*	*b*	*wb*
ε	*ok*	*ok*	*ok*	*e*	*e*
b	*e*	*e*	*e*	*e*	*e*
p	*ok*	*ok*	*ok*	*e*	*k*
w	*ok*	*ok*	*ok*	*e*	*e*
c	*ok*	*ok*	*ok*	*e*	*e*
pw					
pp					
pc					
pb					

New suffix



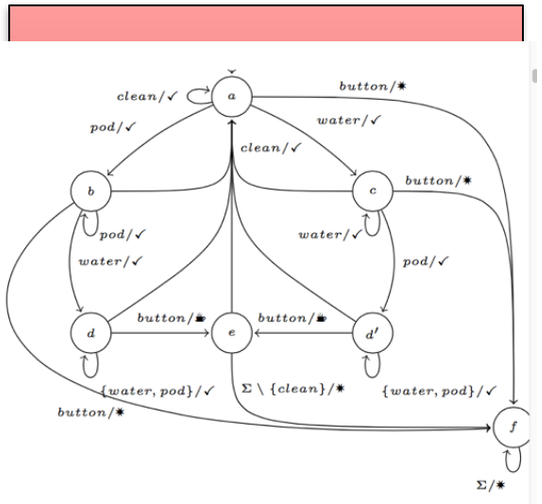
Teacher



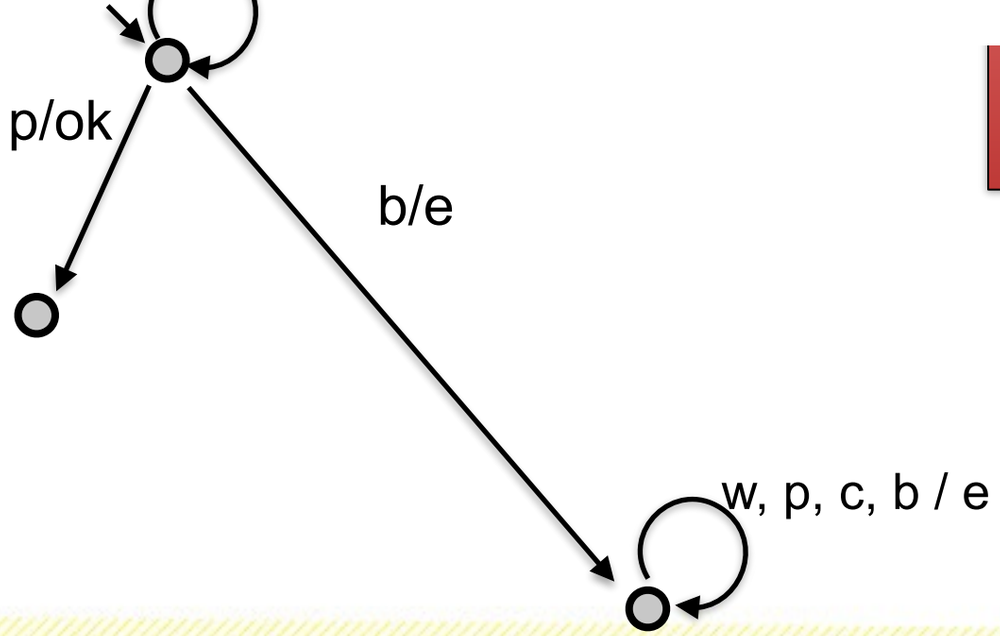
Refining hypothesis (based on Nerode relation)

	w	*p*	*c*	*b*	*wb*
ε	*ok*	*ok*	*ok*	*e*	*e*
b	*e*	*e*	*e*	*e*	*e*
p	*ok*	*ok*	*ok*	*e*	*k*
w	*ok*	*ok*	*ok*	*e*	*e*
c	*ok*	*ok*	*ok*	*e*	*e*
pw	*ok*	*ok*	*ok*	*k*	*k*
pp	*ok*	*ok*	*ok*	*e*	*k*
pc	*ok*	*ok*	*ok*	*e*	*e*
pb	*e*	*e*	*e*	*e*	*e*

New suffix



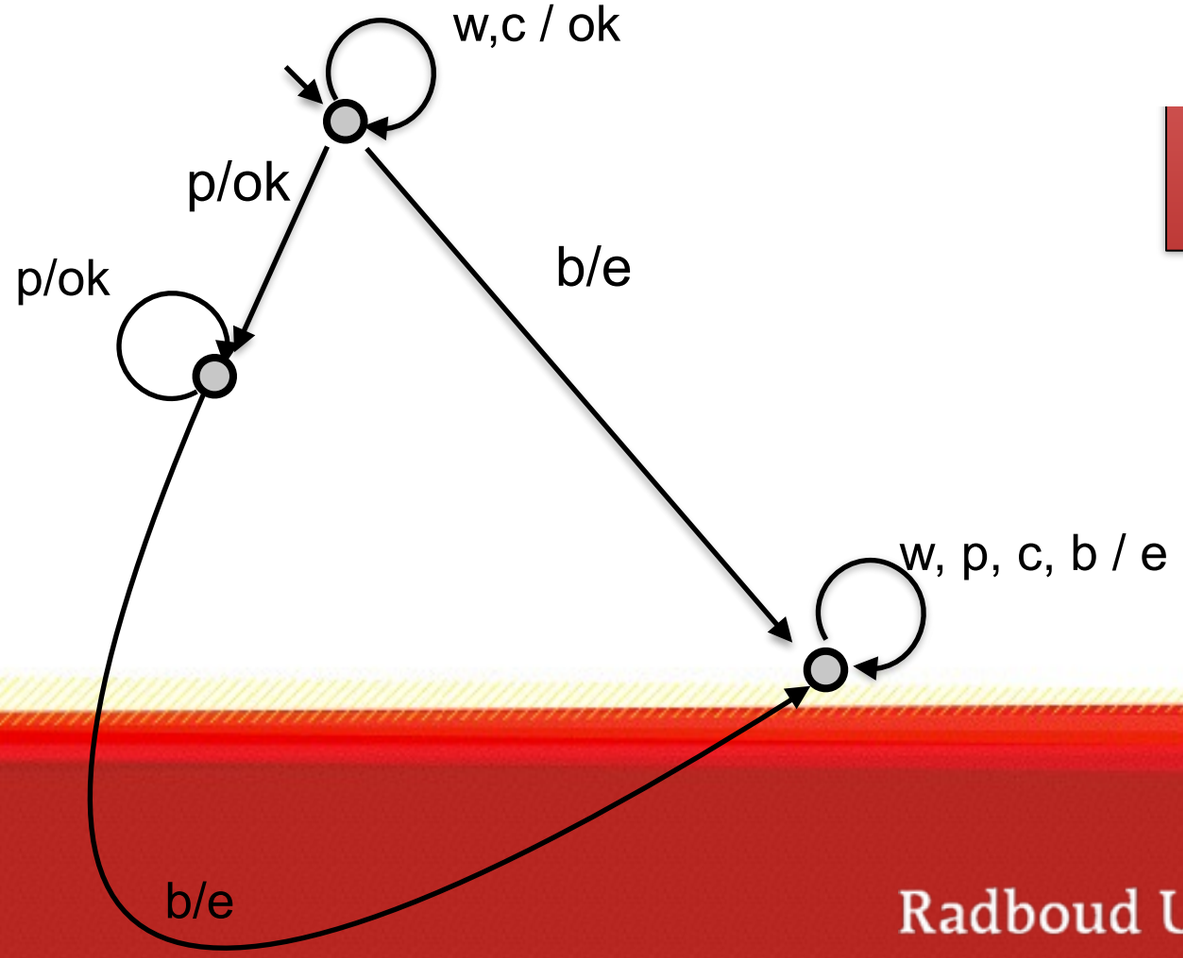
Teacher

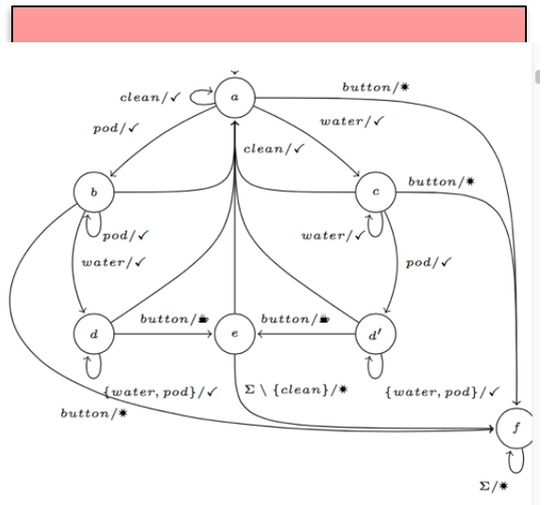


Refining hypothesis (based on Nerode relation)

	w	*p*	*c*	*b*	*wb*
ε	*ok*	*ok*	*ok*	*e*	*e*
b	*e*	*e*	*e*	*e*	*e*
p	*ok*	*ok*	*ok*	*e*	*k*
w	*ok*	*ok*	*ok*	*e*	*e*
c	*ok*	*ok*	*ok*	*e*	*e*
pw	*ok*	*ok*	*ok*	*k*	*k*
pp	*ok*	*ok*	*ok*	*e*	*k*
pc	*ok*	*ok*	*ok*	*e*	*e*
pb	*e*	*e*	*e*	*e*	*e*

New suffix



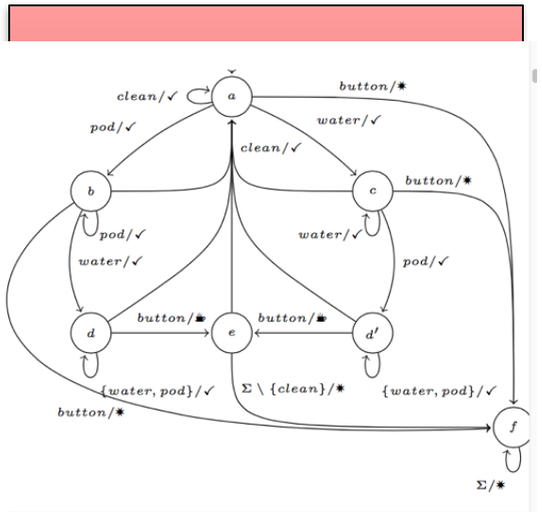


Teacher

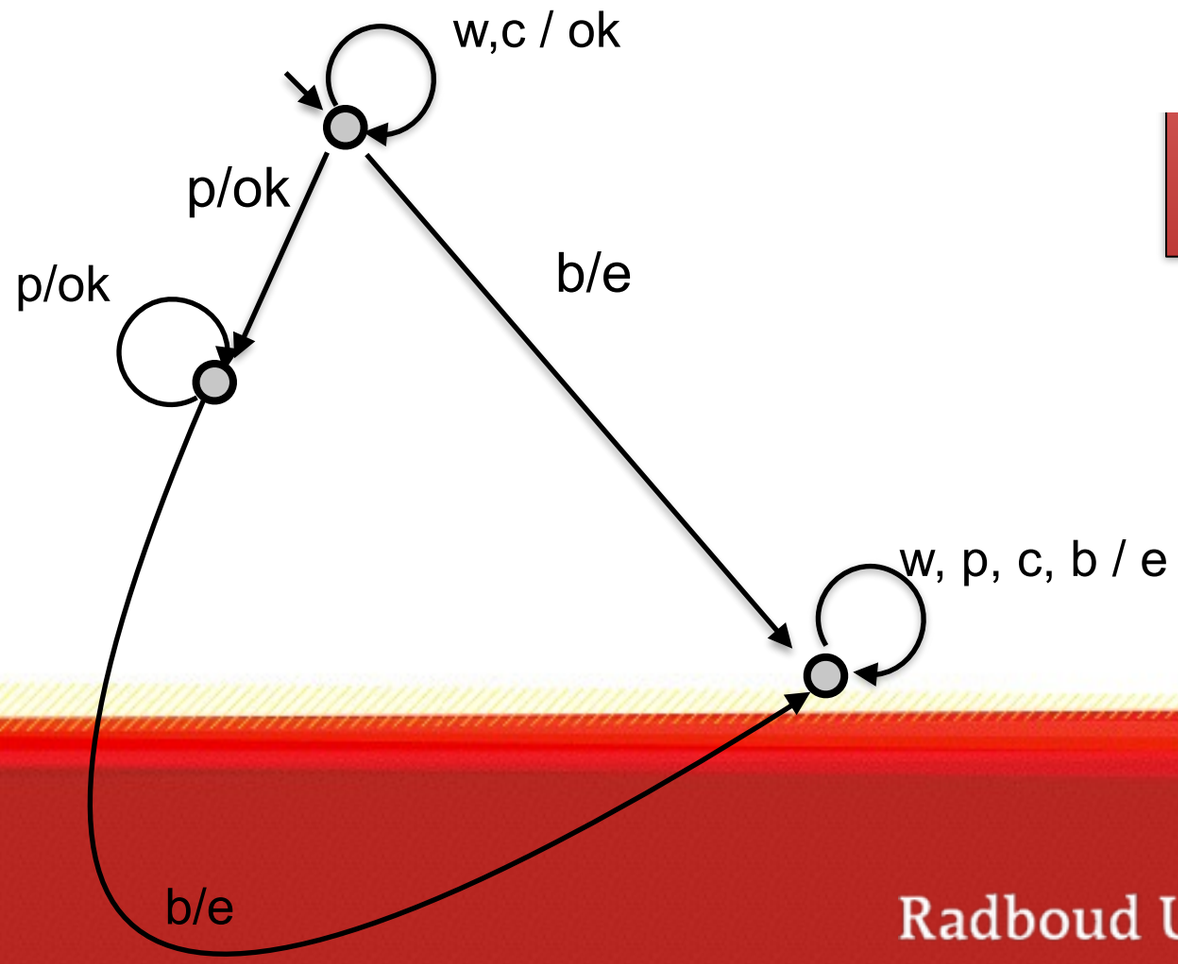
Refining hypothesis (based on Nerode relation)

	w	*p*	*c*	*b*	*wb*
ε	*ok*	*ok*	*ok*	*e*	*e*
b	*e*	*e*	*e*	*e*	*e*
p	*ok*	*ok*	*ok*	*e*	*k*
pw	*ok*	*ok*	*ok*	*k*	*k*
c	*ok*	*ok*	*ok*	*e*	*e*
w	*ok*	*ok*	*ok*	*e*	*e*
pp	*ok*	*ok*	*ok*	*e*	*k*
pc	*ok*	*ok*	*ok*	*e*	*e*
pb	*e*	*e*	*e*	*e*	*e*

New suffix



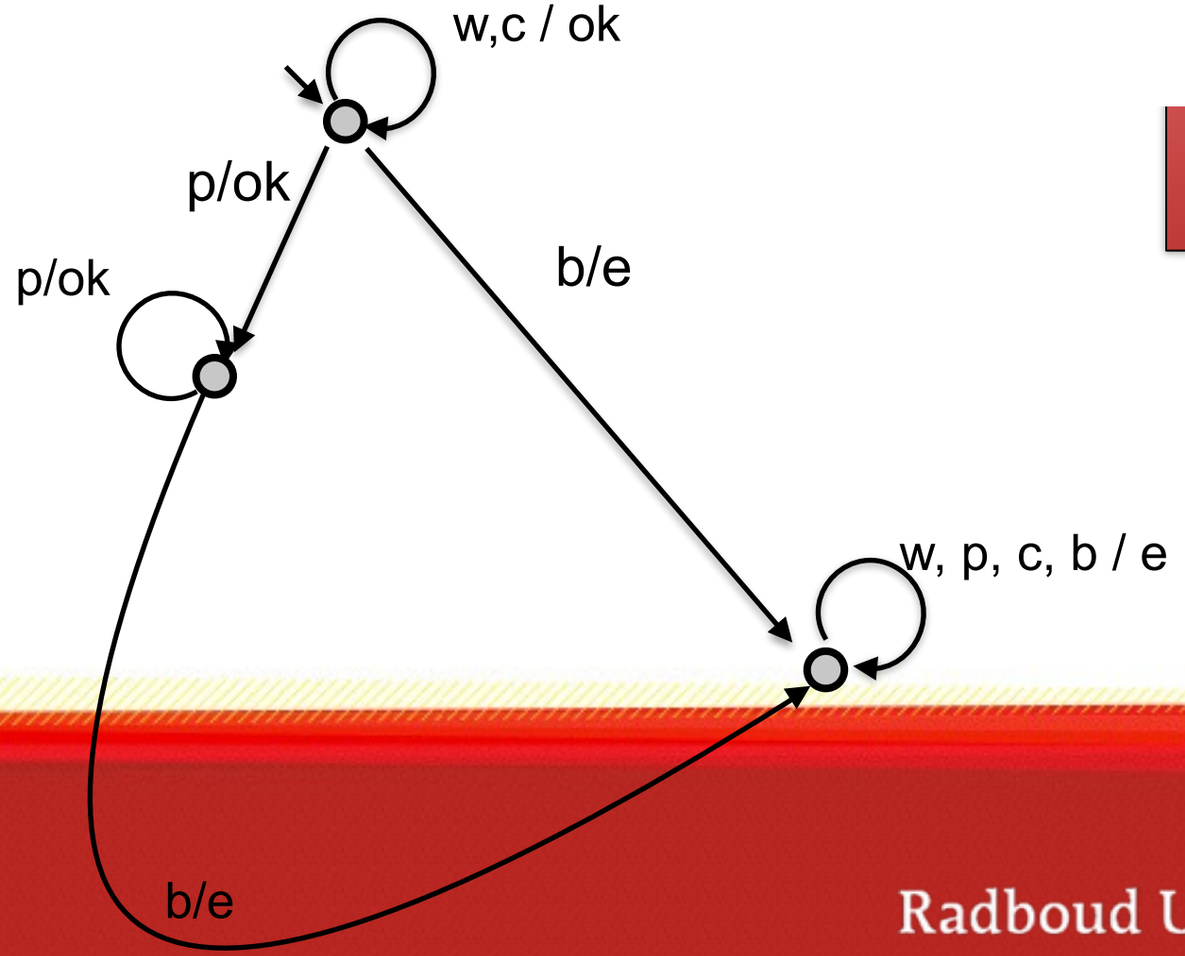
Teacher

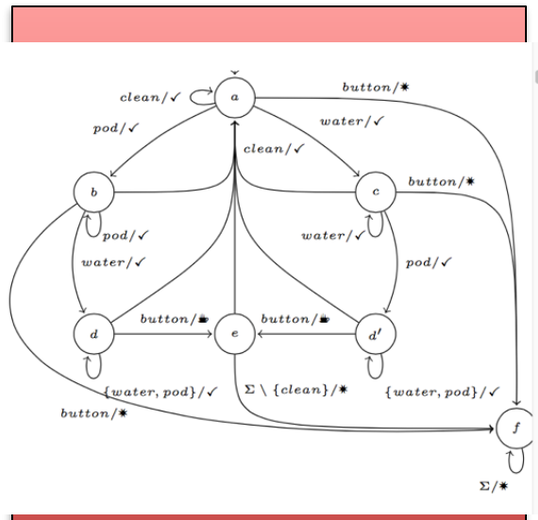


Refining hypothesis (based on Nerode relation)

	w	*p*	*c*	*b*	*wb*
ε	*ok*	*ok*	*ok*	*e*	*e*
b	*e*	*e*	*e*	*e*	*e*
p	*ok*	*ok*	*ok*	*e*	*k*
pw	*ok*	*ok*	*ok*	*k*	*k*
c	*ok*	*ok*	*ok*	*e*	*e*
w	*ok*	*ok*	*ok*	*e*	*e*
pp	*ok*	*ok*	*ok*	*e*	*k*
pc	*ok*	*ok*	*ok*	*e*	*e*
pb	*e*	*e*	*e*	*e*	*e*
pww	*ok*	*ok*	*ok*	*k*	*k*
pwp	*ok*	*ok*	*ok*	*k*	*k*
pwc	*ok*	*ok*	*ok*	*e*	*e*
pwb	*k*	*e*	*ok*	*e*	*e*

New suffix



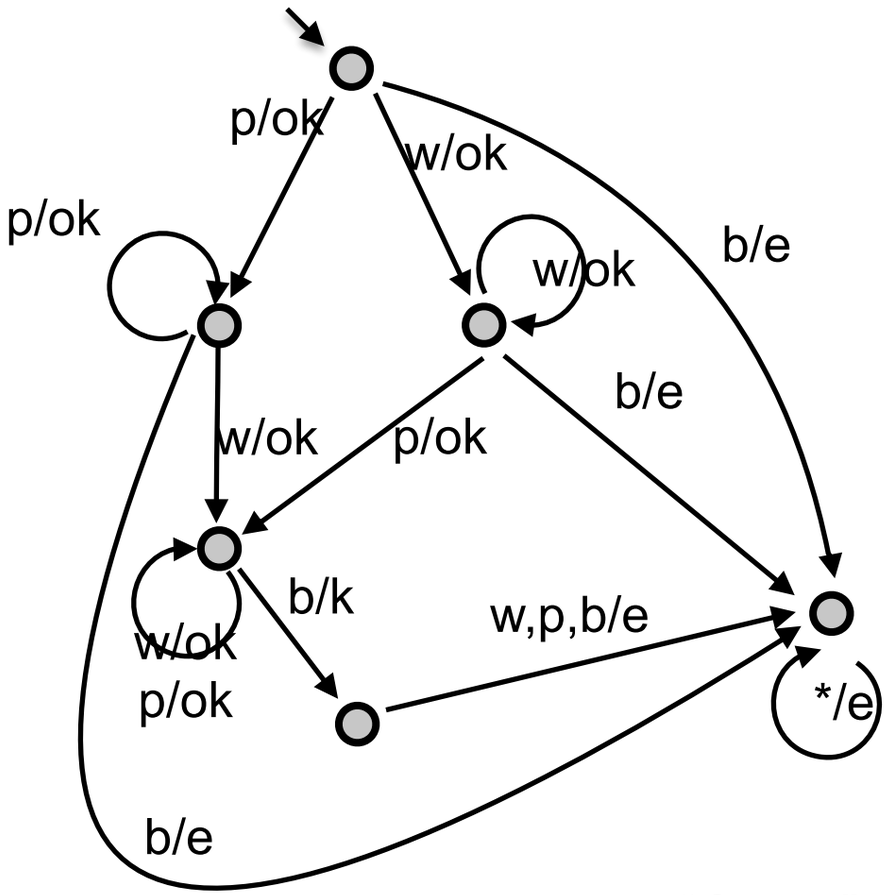


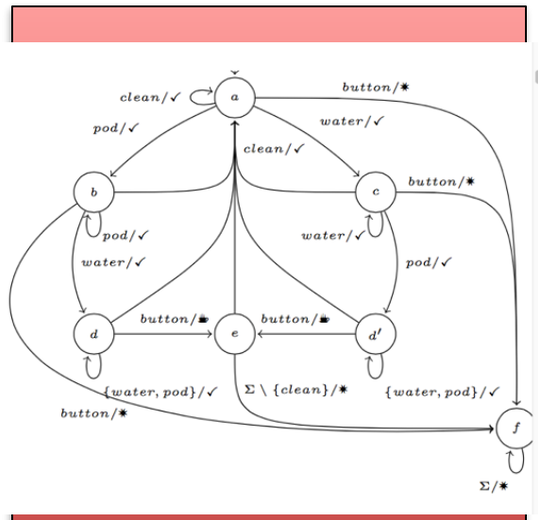
Teacher

A coffe machine

Eventually….

Refining hypothesis (based on Nerode relation)

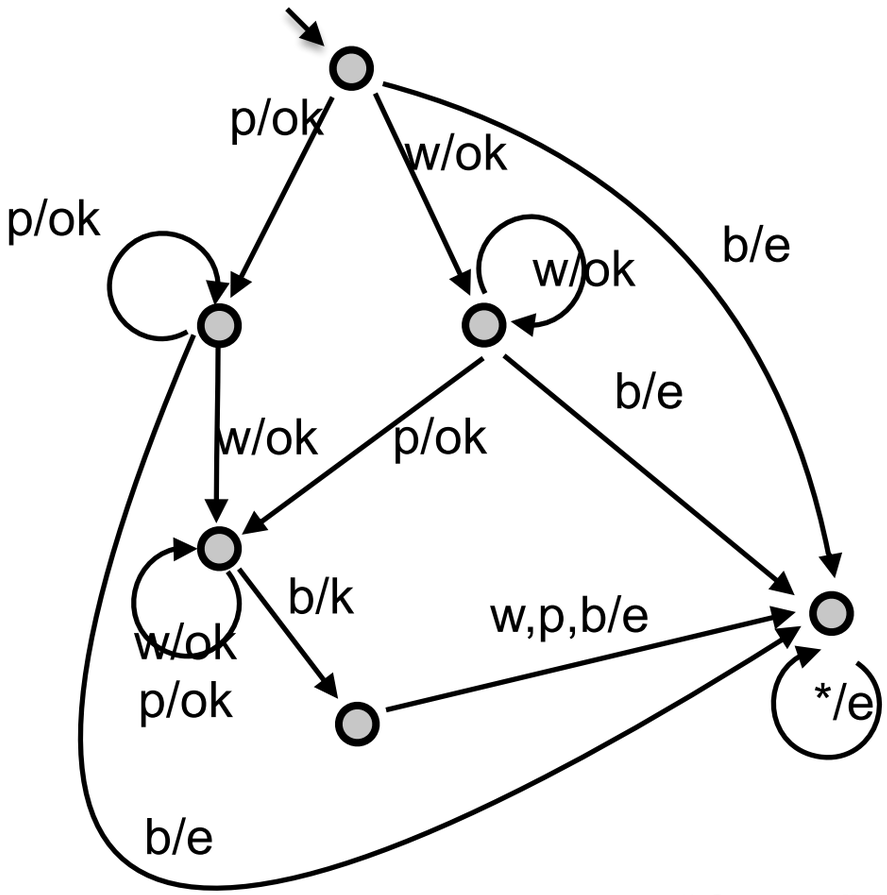




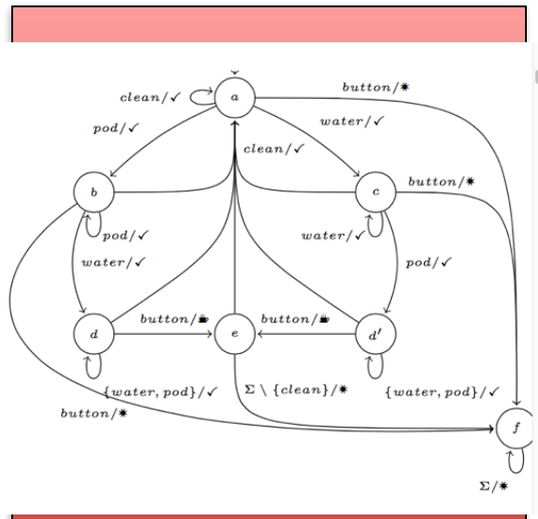
Teacher

"Clean" transitions are not shown!

Refining hypothesis (based on Nerode relation)



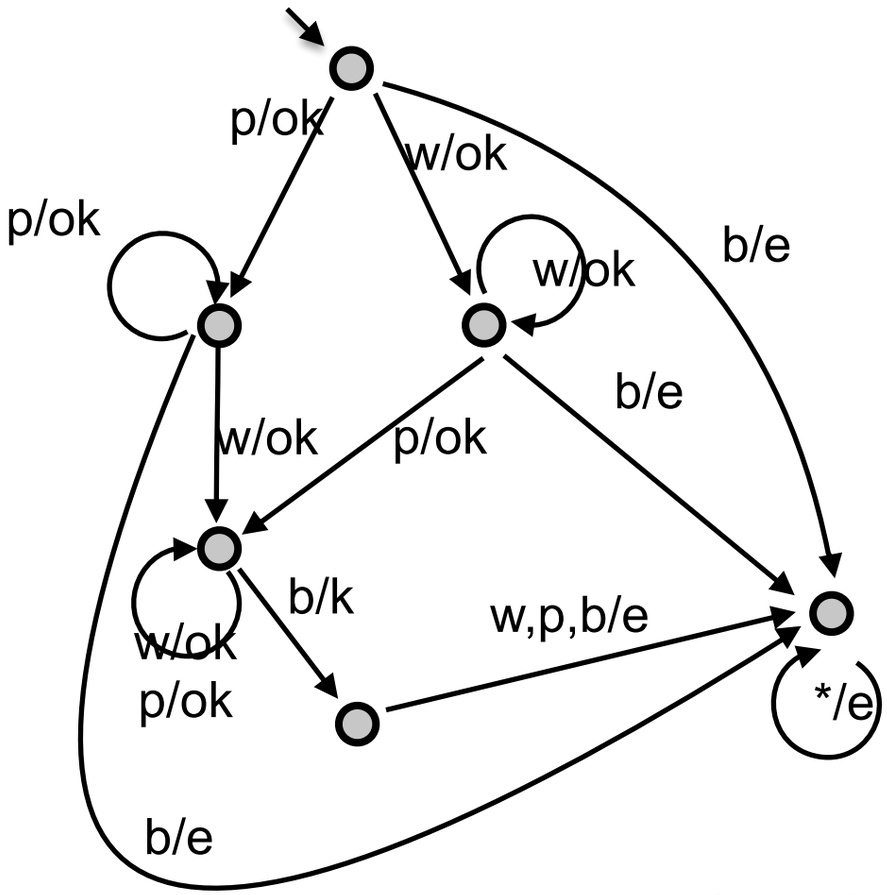
Equivalence query



Teacher

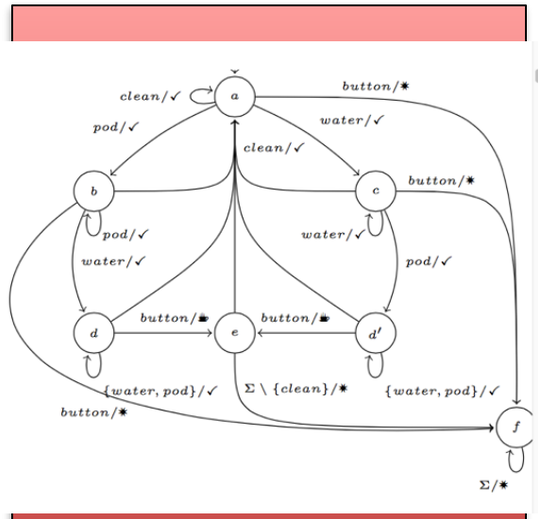
“Clean” transitions are not shown!

Refining hypothesis (based on Nerode relation)



YES!

Equivalence query



Teacher

“Clean” transitions are not shown!

Properties

- Terminates with canonical Mealy machine for $\mathcal{M}$.
- Number of states grows monotonically.
- Number of equivalence queries is bounded by number of states.
- NO information on quality of successive hypotheses!

Counterexample handling

Counterexample and new states

Theorem (Counterexample Decomposition) $\lambda \in \Sigma$

Every counterexample has a suffix λv such that, for two prefixes from the upper part of the table u and u'

$$\mathcal{P}(u\lambda v) \neq \mathcal{P}(u'v)$$

with $u\lambda$ and u' leading to the same state in the current hypothesis.

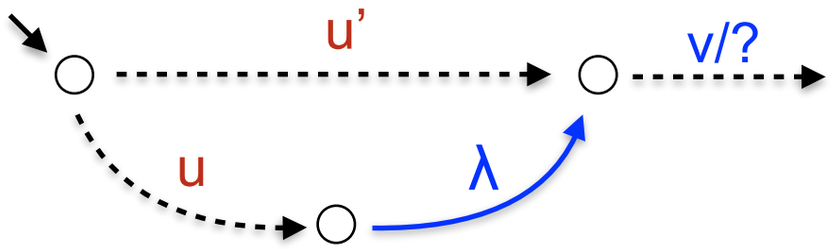
Counterexample and new states

Theorem (Counterexample Decomposition) $\lambda \in \Sigma$

Every counterexample has a suffix λv such that, for two prefixes from the upper part of the table u and u’

$$\mathcal{P}(u\lambda v) \neq \mathcal{P}(u'v)$$

with uλ and u’ leading to the same state in the current hypothesis.



But there is a difference
in behaviour, so
uλ must lead to a another state
(different from all other states in the hypothesis - why?)

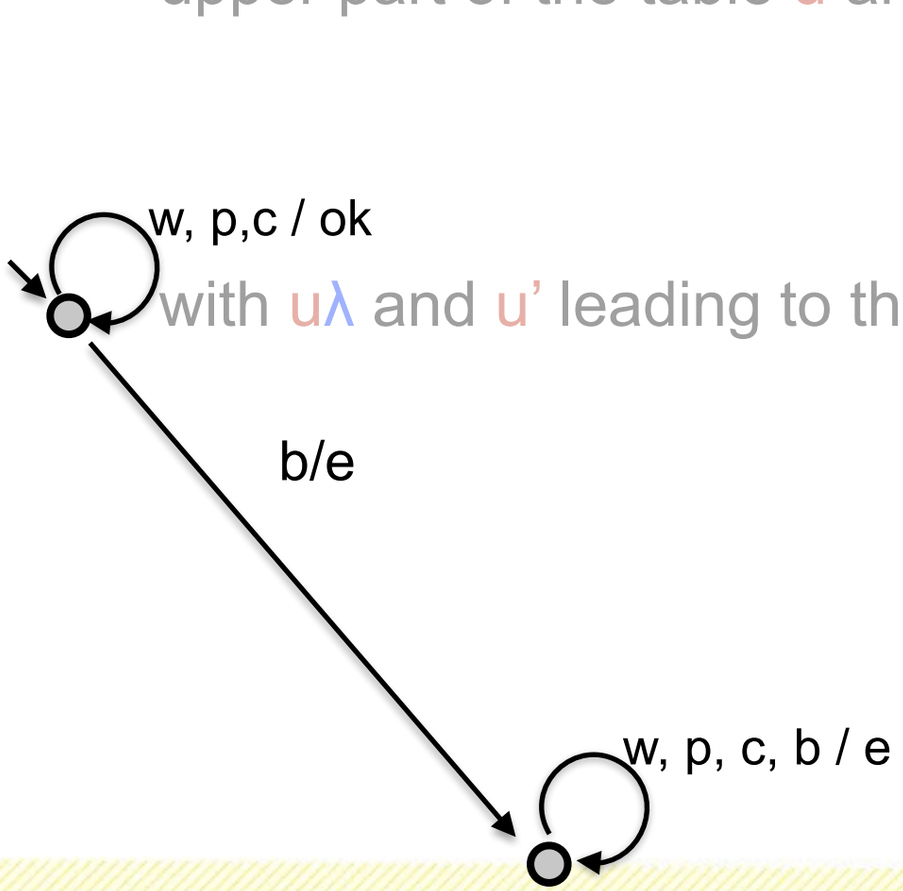
Counterexample and new states

Theorem (Counterexample Decomposition) $\lambda \in \Sigma$

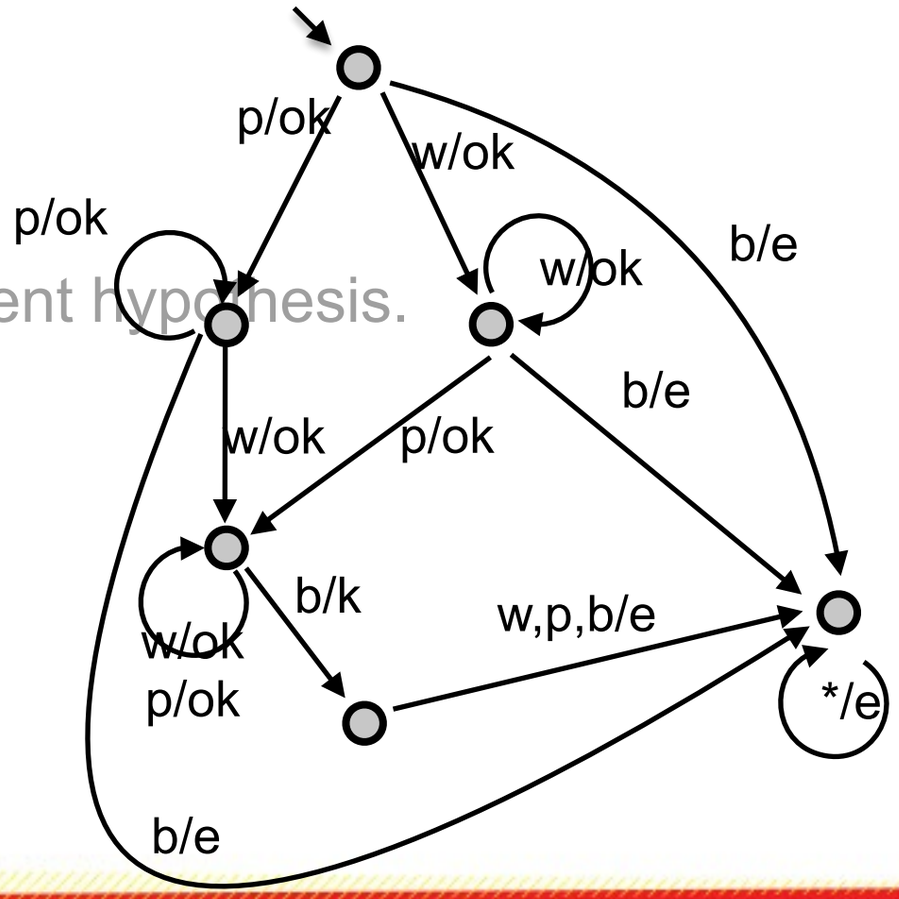
Every counterexample has a suffix λv such that, for two prefixes from the upper part of the table u and u’

$\mathcal{P}(u\lambda v) \neq \mathcal{P}(u'v)$

with $u\lambda$ and u’ leading to the same state in the current hypothesis.



pwpwb -> k



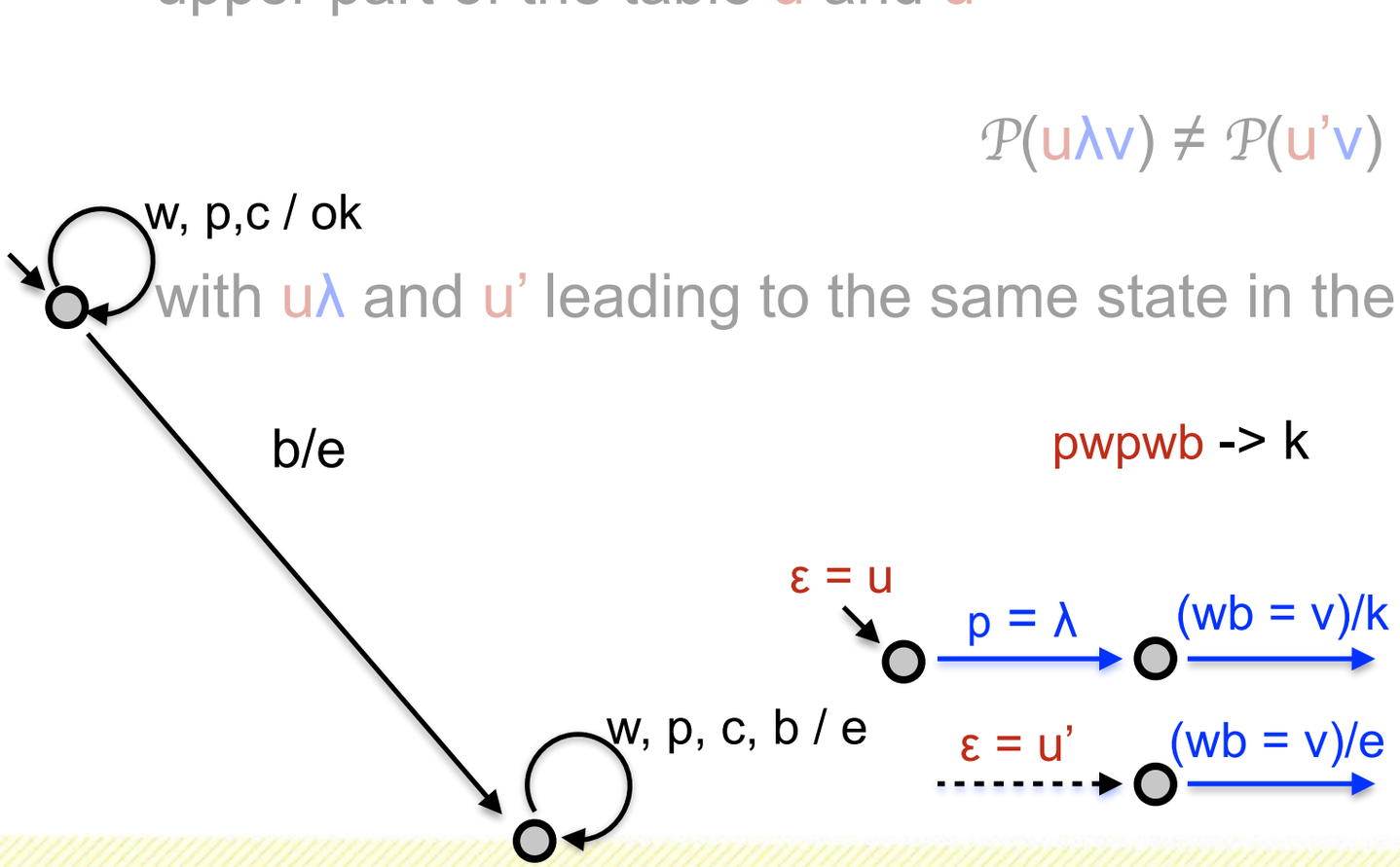
Counterexample and new states

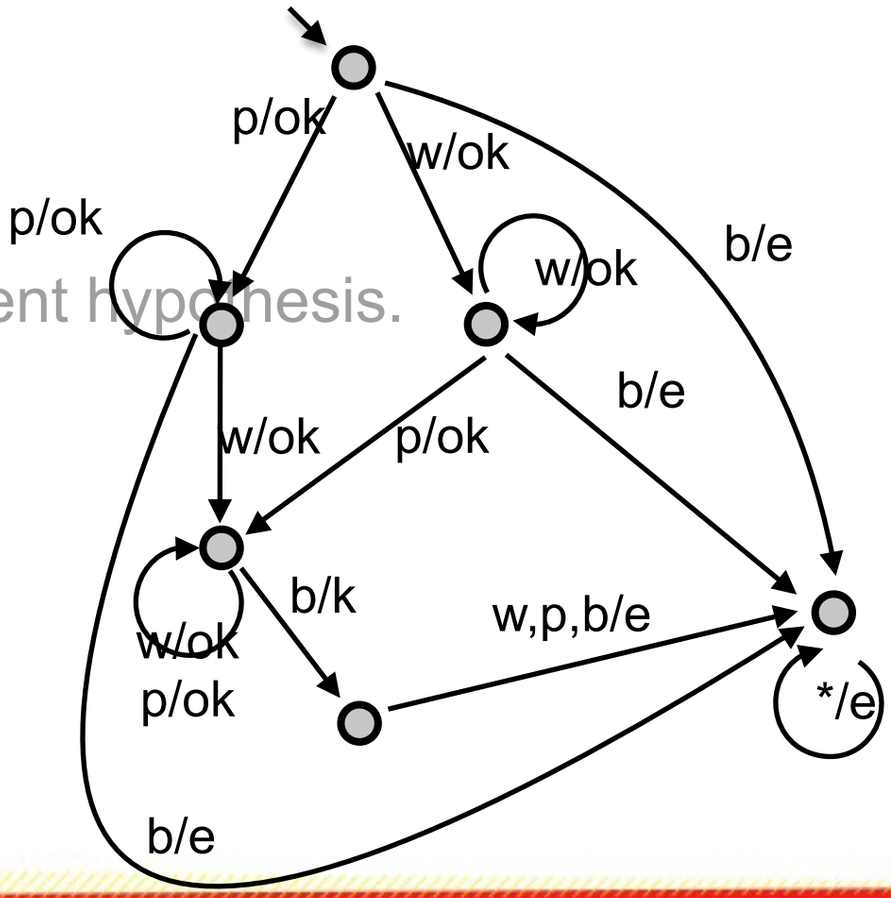
Theorem (Counterexample Decomposition) $\lambda \in \Sigma$

Every counterexample has a suffix λv such that, for two prefixes from the upper part of the table u and u'

$\mathcal{P}(u\lambda v) \neq \mathcal{P}(u'v)$

with uλ and u' leading to the same state in the current hypothesis.





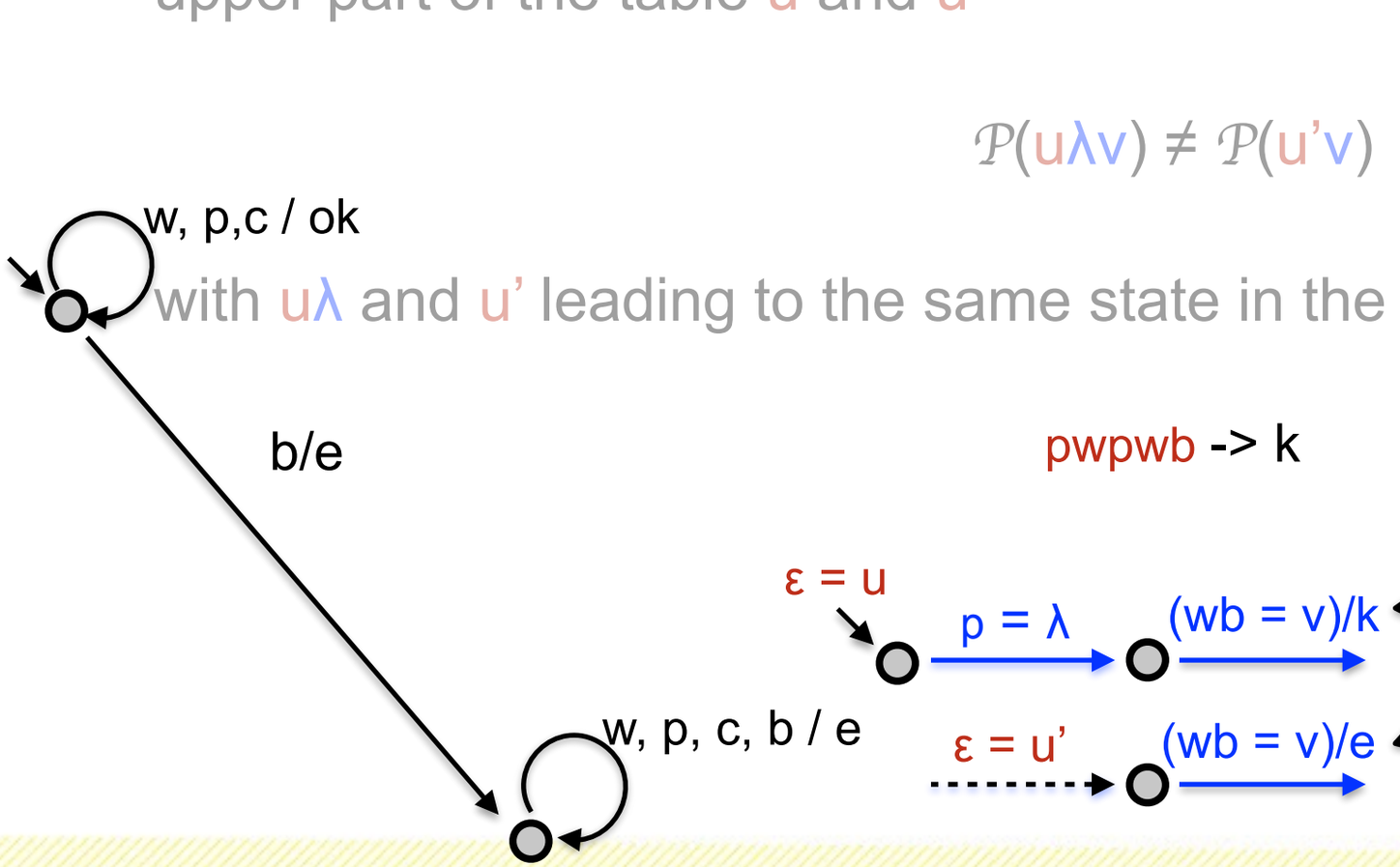
Counterexample and new states

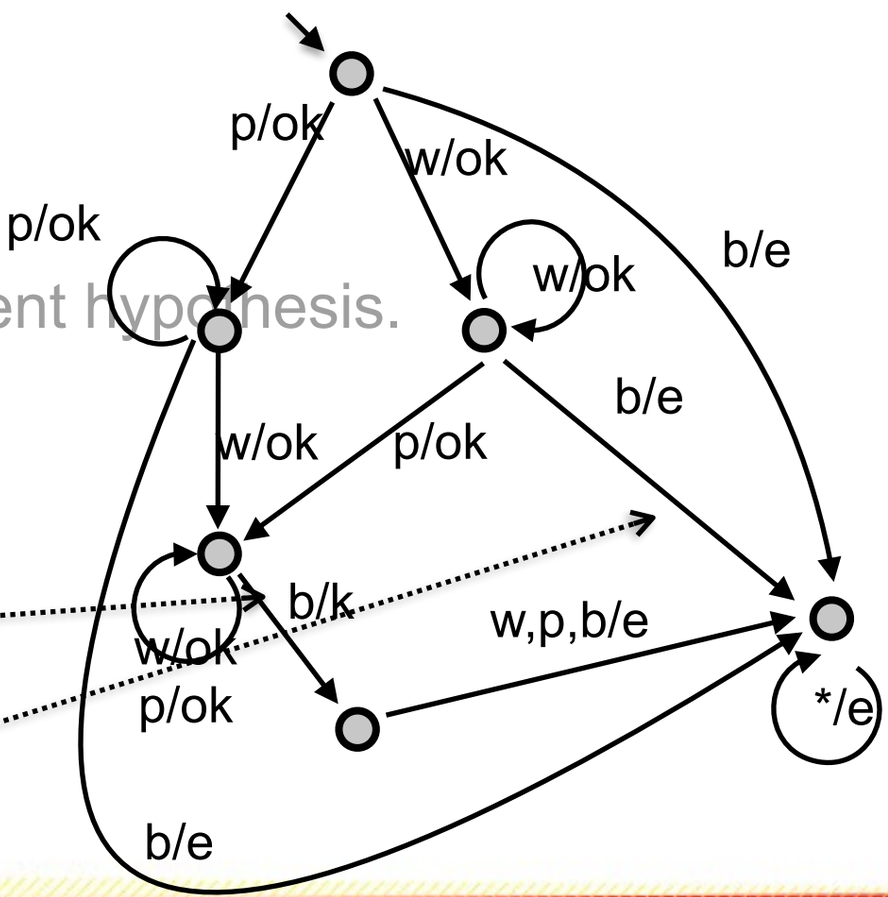
Theorem (Counterexample Decomposition) $\lambda \in \Sigma$

Every counterexample has a suffix λv such that, for two prefixes from the upper part of the table u and u’

$\mathcal{P}(u\lambda v) \neq \mathcal{P}(u'v)$

with uλ and u’ leading to the same state in the current hypothesis.





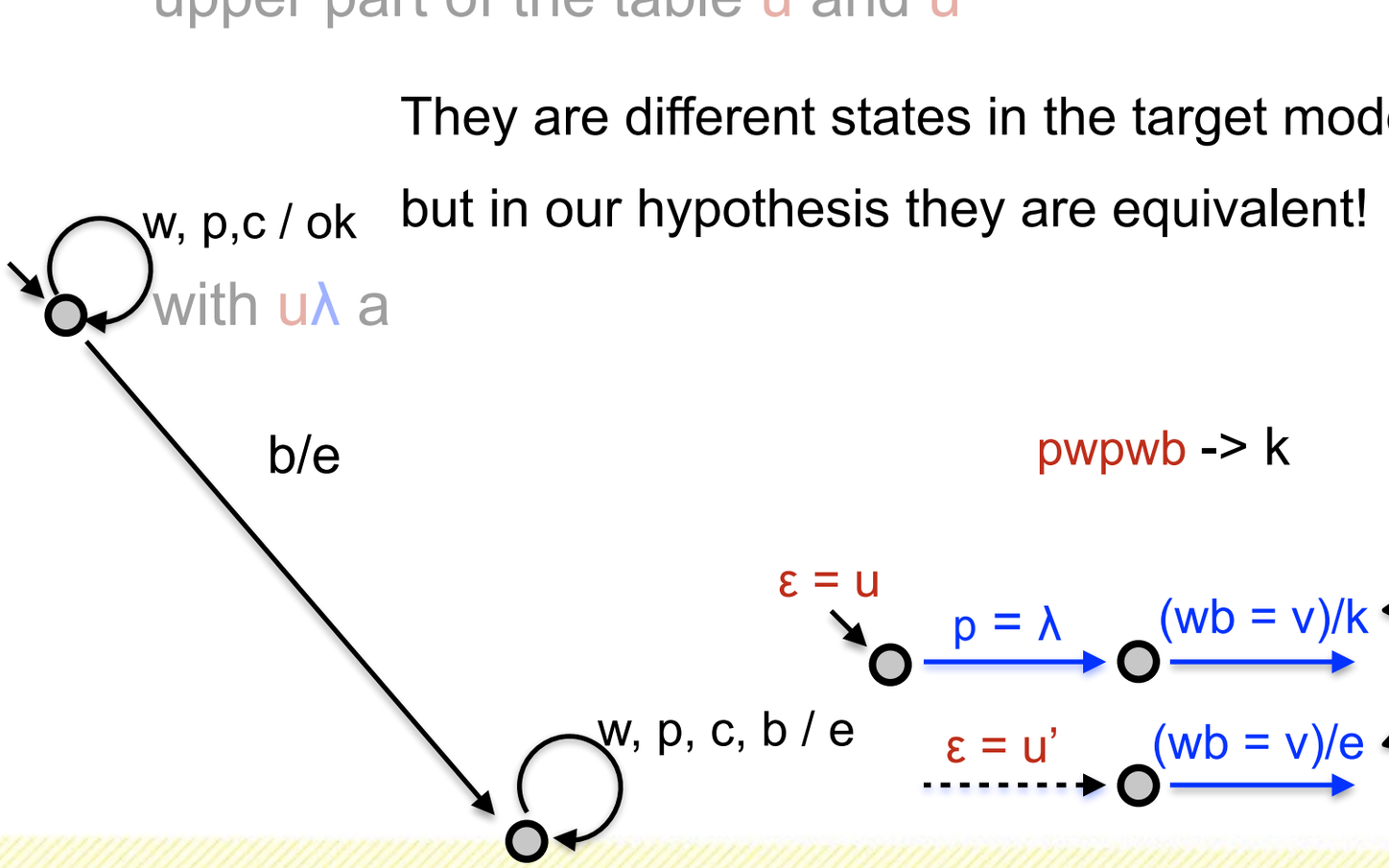
Counterexample and new states

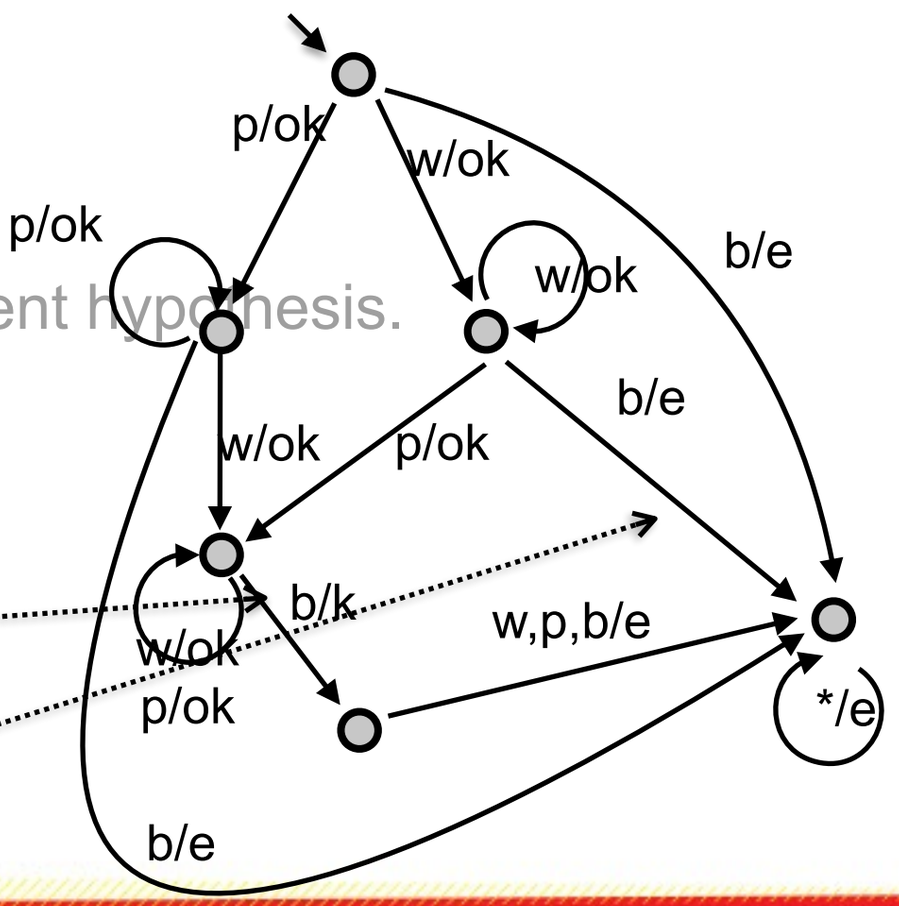
Theorem (Counterexample Decomposition) $\lambda \in \Sigma$

Every counterexample has a suffix λv such that, for two prefixes from the upper part of the table u and u'

They are different states in the target model,

but in our hypothesis they are equivalent!



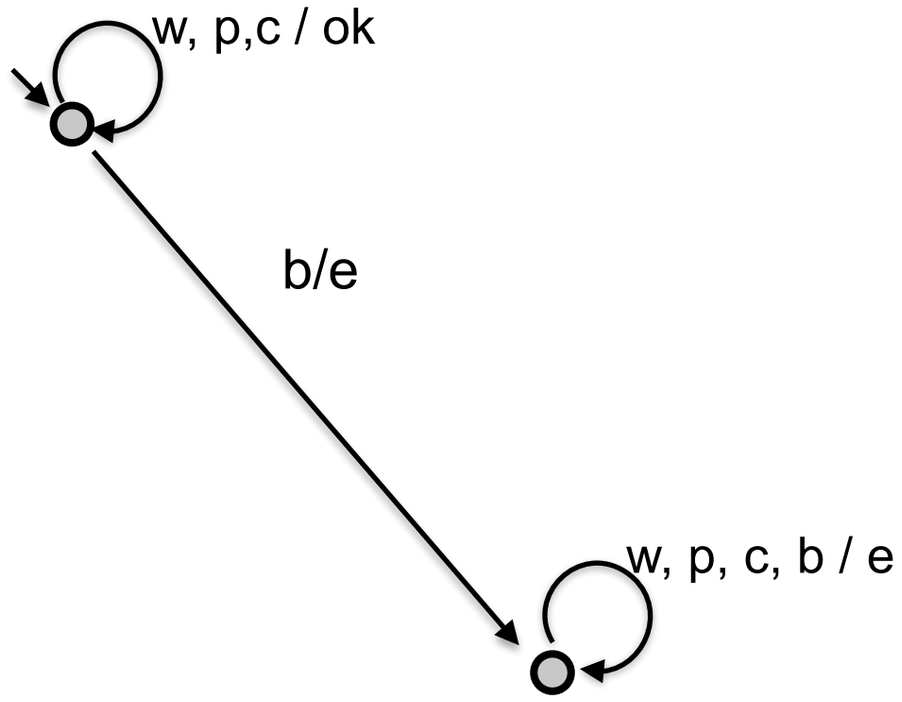


Counterexample and new states

Adding v to the table as a suffix allows us to distinguish state u' from state uλ.

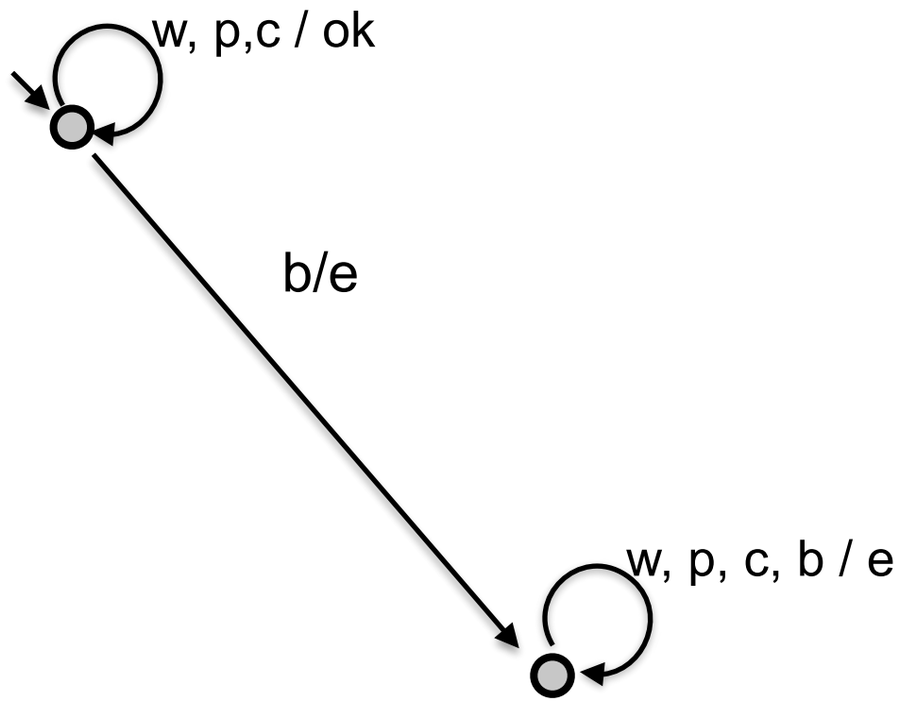
Definitions:
ce is our counterexample;
$\mathcal{P}(\sigma)$ output after σ in target language;
H(σ) output after σ in our hypothesis;
[σ] prefix in the upper part of the table representing the state reached by σ.
$\sigma = \varepsilon\sigma\varepsilon = \sigma_0\sigma_1\sigma_2\ldots\sigma_{m+1}$
$\sigma = [\sigma_0\ldots\sigma_i]\,\sigma_{i+1}\ldots\sigma_{m+1}$
is the word obtained by replacing the first i inputs of σ with the prefix representing the state reached by those i inputs.

How can we find such λ and v?



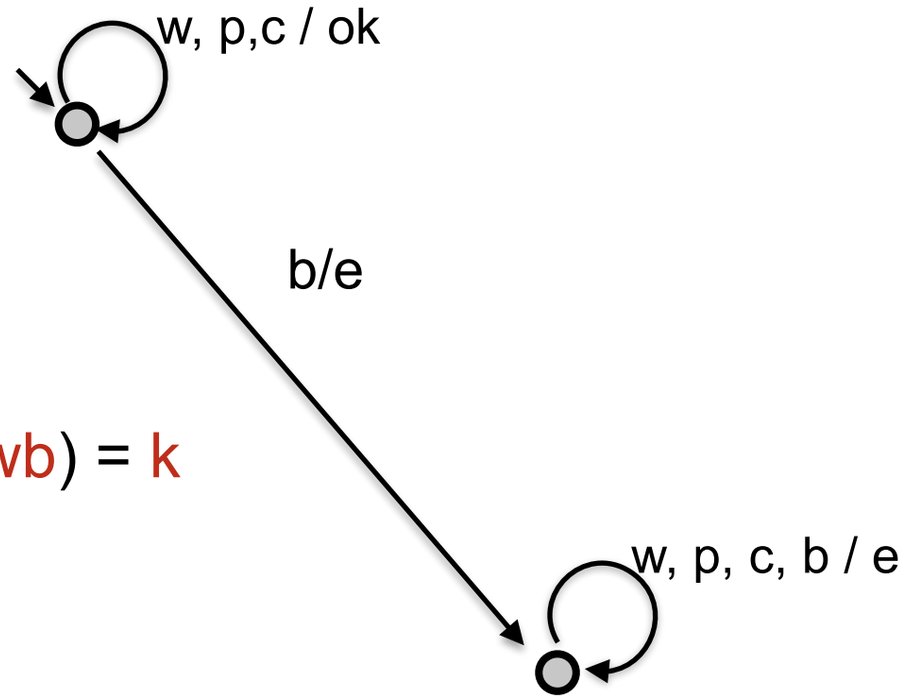
How can we find such λ and v?

- One approach: Binary search!



How can we find such λ and v?

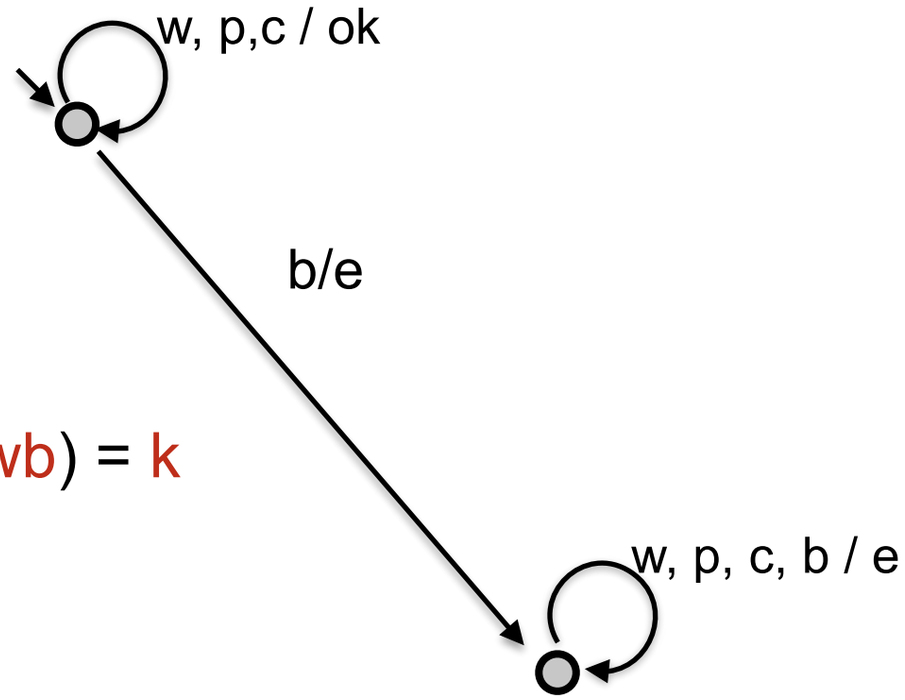
- One approach: Binary search!



- We know that $\mathcal{P}$([pwpwb]) = $\mathcal{P}$(b) = e and $\mathcal{P}$(pwpwb) = k

How can we find such λ and v?

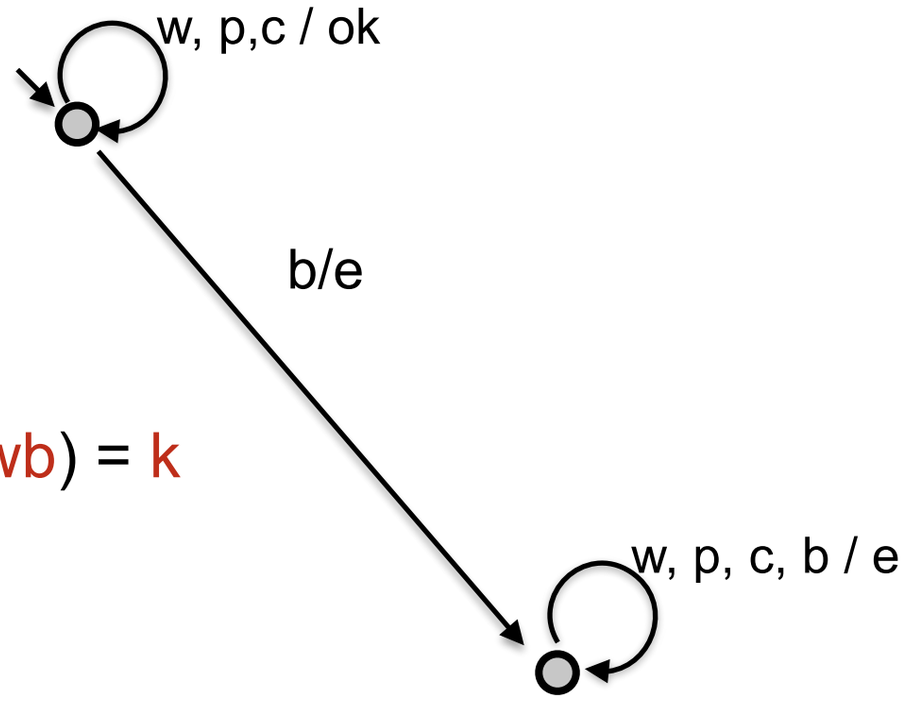
- One approach: Binary search!



- We know that $\mathcal{P}$([pwpwb]) = $\mathcal{P}$(b) = e and $\mathcal{P}$(pwpwb) = k
- What about $\mathcal{P}$([pw]pwb) ?

How can we find such λ and v?

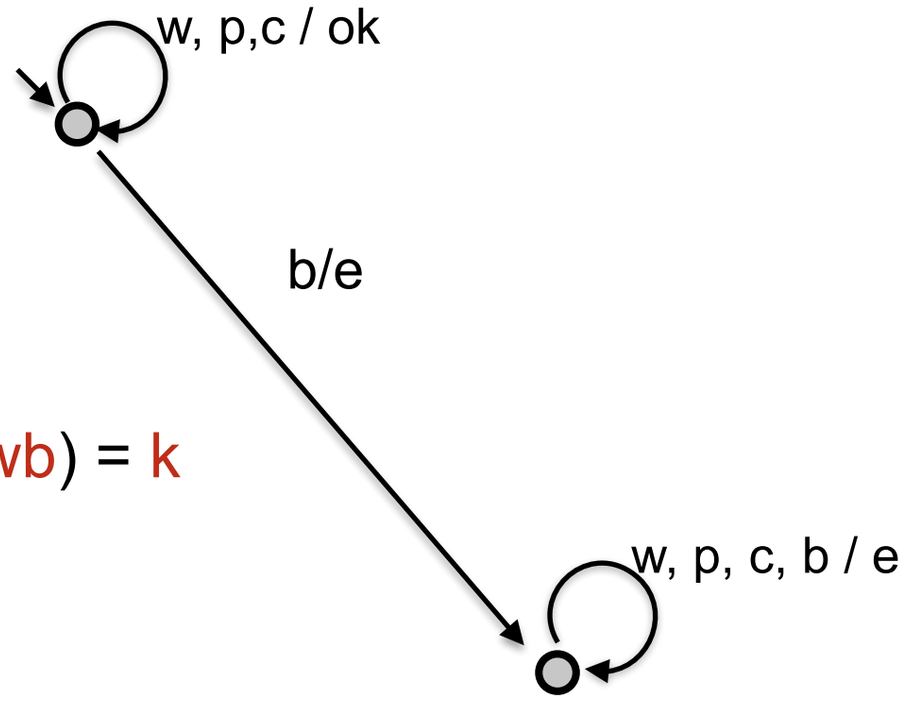
- One approach: Binary search!



- We know that $\mathcal{P}$([pwpwb]) = $\mathcal{P}$(b) = e and $\mathcal{P}$(pwpwb) = k
- What about $\mathcal{P}$([pw]pwb) ?
- $\mathcal{P}$([pw]pwb) = $\mathcal{P}$(εpwb) = k

How can we find such λ and v?

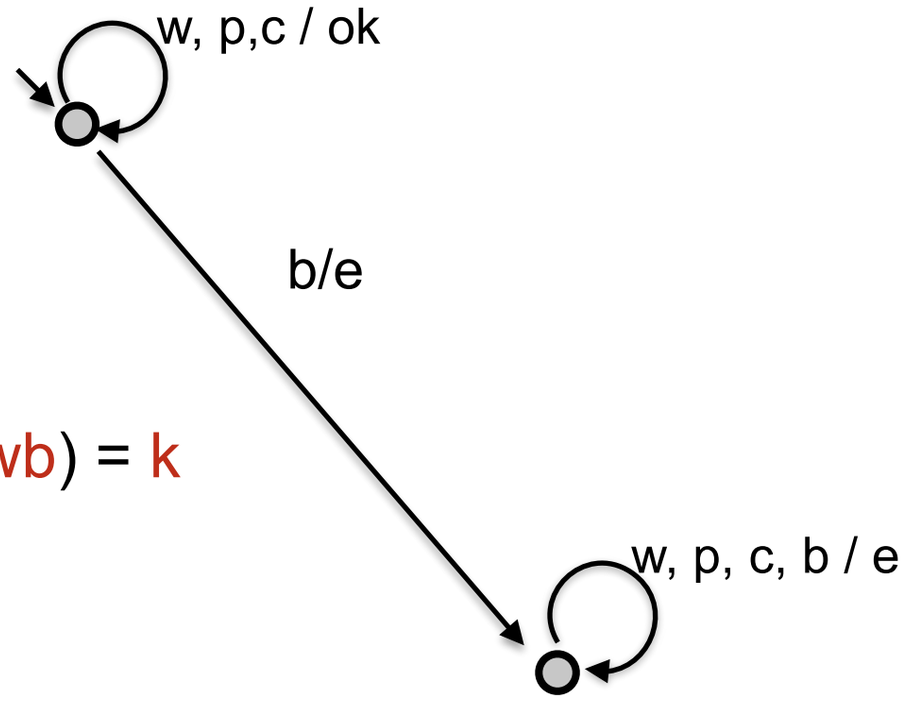
- One approach: Binary search!



- We know that $\mathcal{P}$([pwpwb]) = $\mathcal{P}$(b) = e and $\mathcal{P}$(pwpwb) = k
- What about $\mathcal{P}$([pw]pwb) ?
- $\mathcal{P}$([pw]pwb) = $\mathcal{P}$(εpwb) = k
- We need to extend the brackets:

How can we find such λ and v?

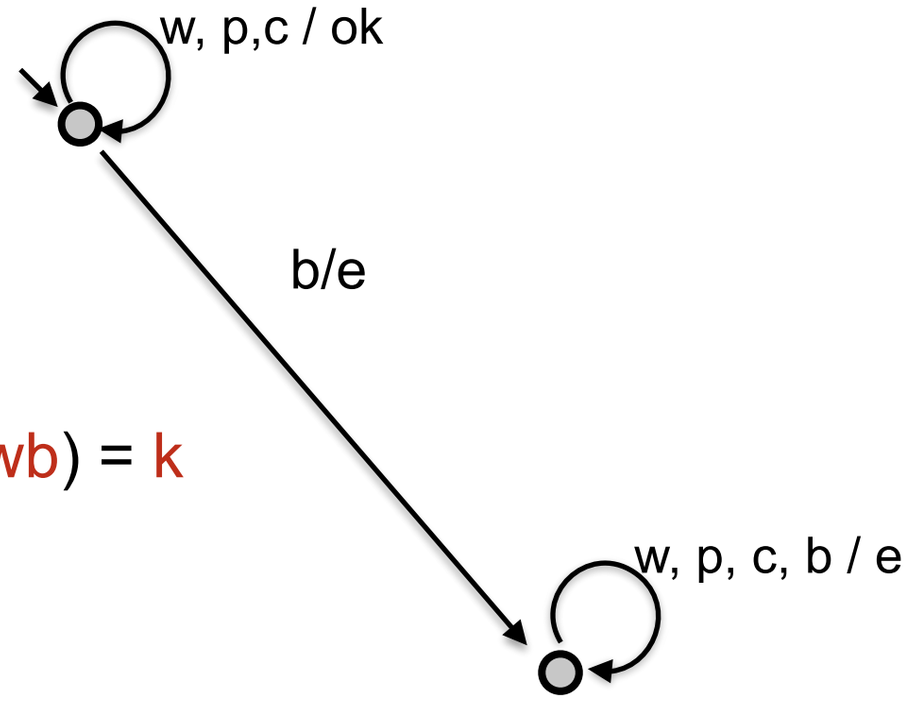
- One approach: Binary search!



- We know that $\mathcal{P}$([pwpwb]) = $\mathcal{P}$(b) = e and $\mathcal{P}$(pwpwb) = k
- What about $\mathcal{P}$([pw]pwb) ?
- $\mathcal{P}$([pw]pwb) = $\mathcal{P}$(εpwb) = k
- We need to extend the brackets:
- $\mathcal{P}$([pwp]wb) = $\mathcal{P}$(εwb) = e

How can we find such λ and v?

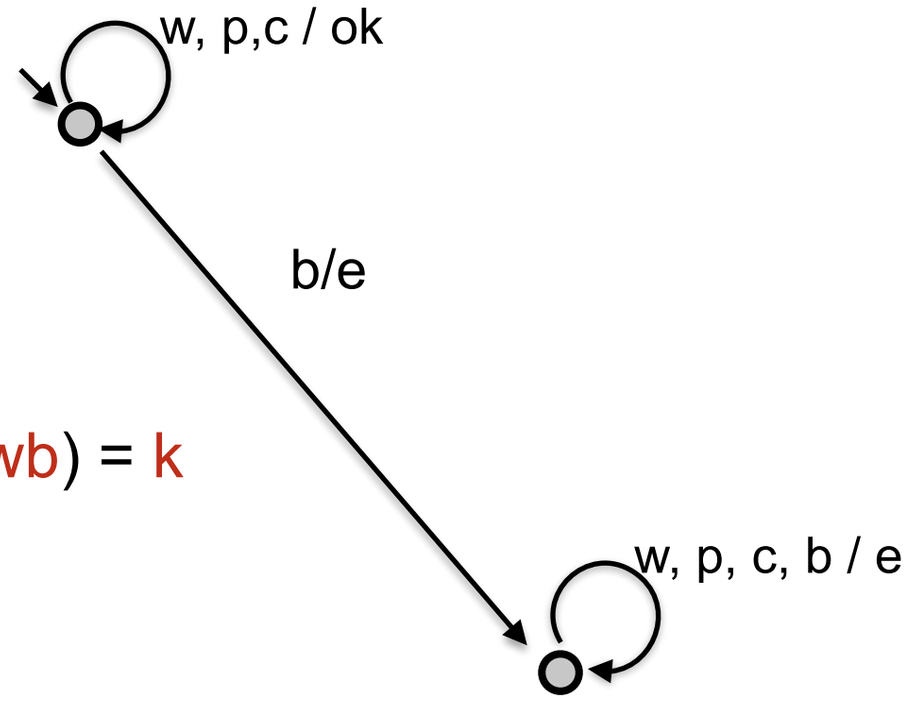
- One approach: Binary search!



- We know that $\mathcal{P}$([pwpwb]) = $\mathcal{P}$(b) = e and $\mathcal{P}$(pwpwb) = k
- What about $\mathcal{P}$([pw]pwb) ?
- $\mathcal{P}$([pw]pwb) = $\mathcal{P}$(εpwb) = k
- We need to extend the brackets:
- $\mathcal{P}$([pwp]wb) = $\mathcal{P}$(εwb) = e
- Thus λ = p and v = wb

How can we find such λ and v?

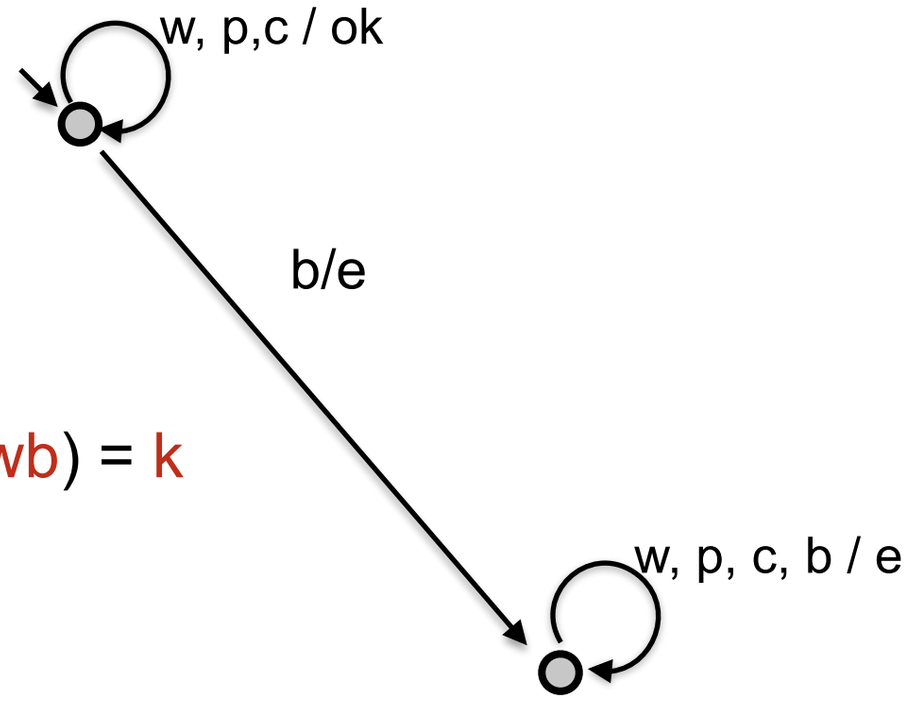
- One approach: Binary search!



- We know that $\mathcal{P}$([pwpwb]) = $\mathcal{P}$(b) = e and $\mathcal{P}$(pwpwb) = k
- What about $\mathcal{P}$([pw]pwb) ?
- $\mathcal{P}$([pw]pwb) = $\mathcal{P}$(εpwb) = k
- We need to extend the brackets:
- $\mathcal{P}$([pwp]wb) = $\mathcal{P}$(εwb) = e
- Thus λ = p and v = wb

How can we find such λ and v?

- One approach: Binary search!



- We know that $\mathcal{P}$([pwpwb]) = $\mathcal{P}$(b) = e and $\mathcal{P}$(pwpwb) = k
- What about $\mathcal{P}$([pw]pwb) ?
- $\mathcal{P}$([pw]pwb) = $\mathcal{P}$(εpwb) = k
- We need to extend the brackets:
- $\mathcal{P}$([pwp]wb) = $\mathcal{P}$(εwb) = e
- Thus λ = p and v = wb

Add wb to the table as a suffix! (and all its suffixes - suffix-closed).

How can we find such λ and v?

- Another approach:

How can we find such λ and v?

- Another approach:

  - Remove the longest prefix of ce which is also in the table as a prefix.

How can we find such λ and v?

- Another approach:

  - Remove the longest prefix of ce which is also in the table as a prefix.
  - Add the rest to the table as a suffix (and all its suffixes - suffix-closed).

How can we find such λ and v?

- Another approach:

- Remove the longest prefix of ce which is also in the table as a prefix.
- Add the rest to the table as a suffix (and all its suffixes - suffix-closed).
- With this approach we are sure to add such v to the suffixes.

How can we find such λ and v?

- Another approach:

- Remove the longest prefix of ce which is also in the table as a prefix.
- Add the rest to the table as a suffix (and all its suffixes - suffix-closed).
- With this approach we are sure to add such v to the suffixes.
- We add (in general) more suffixes than the previous approach.

How can we find such λ and v?

- Another approach:

- Remove the longest prefix of ce which is also in the table as a prefix.
- Add the rest to the table as a suffix (and all its suffixes - suffix-closed).
- With this approach we are sure to add such v to the suffixes.
- We add (in general) more suffixes than the previous approach.

- In our example: ce = pwpwb

How can we find such λ and v?

- Another approach:

- Remove the longest prefix of ce which is also in the table as a prefix.
- Add the rest to the table as a suffix (and all its suffixes - suffix-closed).
- With this approach we are sure to add such v to the suffixes.
- We add (in general) more suffixes than the previous approach.

- In our example: ce = pwpwb
- Shortest prefix in the table was p

How can we find such λ and v?

- Another approach:

- Remove the longest prefix of ce which is also in the table as a prefix.
- Add the rest to the table as a suffix (and all its suffixes - suffix-closed).
- With this approach we are sure to add such v to the suffixes.
- We add (in general) more suffixes than the previous approach.

- In our example: ce = pwpwb
- Shortest prefix in the table was p
- Add {wpwb,pwb,wb} as suffixes to the table

Another way to handle counterexample

- The original counterexample handling:

	w	*p*	*c*	*b*
ε	*ok*	*ok*	*ok*	*e*
b	*e*	*e*	*e*	*e*
w	*ok*	*ok*	*ok*	*e*
p	*ok*	*ok*	*ok*	*e*
c	*ok*	*ok*	*ok*	*e*
bw	*e*	*e*	*e*	*e*
bp	*e*	*e*	*e*	*e*
bc	*e*	*e*	*e*	*e*
bb	*e*	*e*	*e*	*e*

Another way to handle counterexample

- The original counterexample handling:

	w	*p*	*c*	*b*
ε	*ok*	*ok*	*ok*	*e*
b	*e*	*e*	*e*	*e*
w	*ok*	*ok*	*ok*	*e*
p	*ok*	*ok*	*ok*	*e*
c	*ok*	*ok*	*ok*	*e*
bw	*e*	*e*	*e*	*e*
bp	*e*	*e*	*e*	*e*
bc	*e*	*e*	*e*	*e*
bb	*e*	*e*	*e*	*e*

- Add pwpwb (and all its prefixes) to the table in the top part.

Another way to handle counterexample

- The original counterexample handling:

	w	*p*	*c*	*b*
ε	*ok*	*ok*	*ok*	*e*
b	*e*	*e*	*e*	*e*
pwpwb	*e*	*e*	*ok*	*e*
pwpw	*ok*	*ok*	*ok*	*k*
pwp	*ok*	*ok*	*ok*	*k*
pw	*ok*	*ok*	*ok*	*k*
p	*ok*	*ok*	*ok*	*e*
…	*…*	*…*	*…*	*…*
w	*ok*	*ok*	*ok*	*e*

- Add pwpwb (and all its prefixes) to the table in the top part.
- Now the table is **inconsistent.**

Another way to handle counterexample

- The original counterexample handling:

	w	*p*	*c*	*b*
ε	*ok*	*ok*	*ok*	*e*
b	*e*	*e*	*e*	*e*
pwpwb	*e*	*e*	*ok*	*e*
pwpw	*ok*	*ok*	*ok*	*k*
pwp	*ok*	*ok*	*ok*	*k*
pw	*ok*	*ok*	*ok*	*k*
p	*ok*	*ok*	*ok*	*e*
…	*…*	*…*	*…*	*…*
w	*ok*	*ok*	*ok*	*e*

- Add pwpwb (and all its prefixes) to the table in the top part.
- Now the table is **inconsistent.**
- Represent the same state,

Another way to handle counterexample

- The original counterexample handling:

	w	*p*	*c*	*b*
ε	*ok*	*ok*	*ok*	*e*
b	*e*	*e*	*e*	*e*
pwpwb	*e*	*e*	*ok*	*e*
pwpw	*ok*	*ok*	*ok*	*k*
pwp	*ok*	*ok*	*ok*	*k*
pw	*ok*	*ok*	*ok*	*k*
p	*ok*	*ok*	*ok*	*e*
…	*…*	*…*	*…*	*…*
w	*ok*	*ok*	*ok*	*e*

- Add pwpwb (and all its prefixes) to the table in the top part.
- Now the table is **inconsistent.**
- Represent the same state,
- but if we take the same transition (w) we end in different states.

Another way to handle counterexample

- The original counterexample handling:

	w	*p*	*c*	*b*
ε	*ok*	*ok*	*ok*	*e*
b	*e*	*e*	*e*	*e*
pwpwb	*e*	*e*	*ok*	*e*
pwpw	*ok*	*ok*	*ok*	*k*
pwp	*ok*	*ok*	*ok*	*k*
pw	*ok*	*ok*	*ok*	*k*
p	*ok*	*ok*	*ok*	*e*
…	*…*	*…*	*…*	*…*
w	*ok*	*ok*	*ok*	*e*

- Add pwpwb (and all its prefixes) to the table in the top part.
- Now the table is **inconsistent.**

- Represent the same state,
- but if we take the same transition (w) we end in different states.

- The inconsistency is solved by adding wb as a suffix! (same as before)

History of counterexample handling

- Original L* added all prefixes of a counterexample to the upper part of the table. This leads to an inconsistency in the table. To solve the inconsistency a suffix (or more) is added. [Angluin 1987]

- Add a suffix closed set to the observation table. [Shahbaz and Groz 2009]

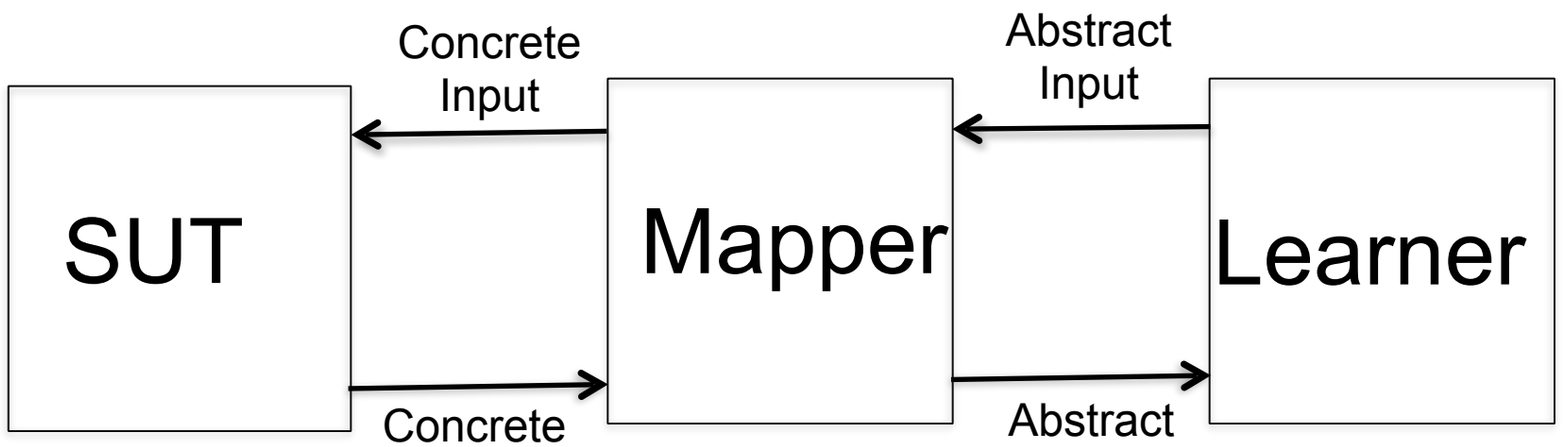
- Add suffixes incrementally. [Irfan et al. 2010]

- Binary search and add one suffix. [Rivest and Schapire 1993]

Other formalisms

Learning Scalarset Mealy Machines

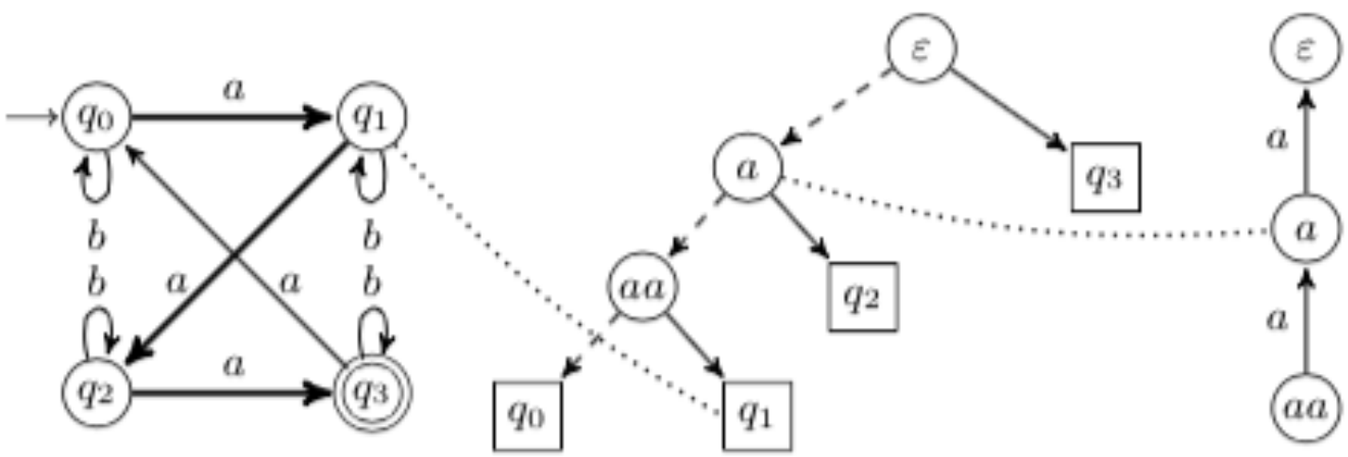
- Using mappers in order to abstract from data (and infinite inputs)



- Aarts et al. Automata Learning Through Counterexample Guided Abstraction Refinement. 2011

Learning Using a different structure

- Using trees instead of observation tables.
- Avoid to ask membership queries in case of redundant information.

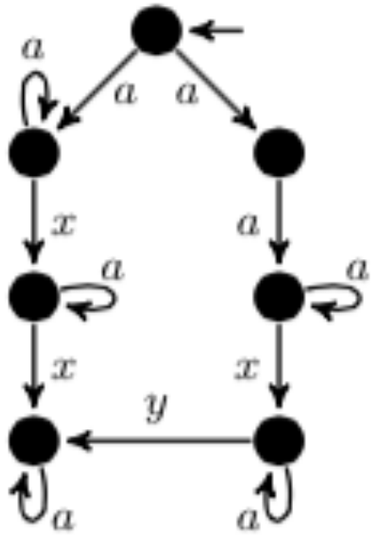


- Isberner et al. The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning. 2014

Towards Nondeterminism

Learning Labelled Transition Systems - with inputs and outputs

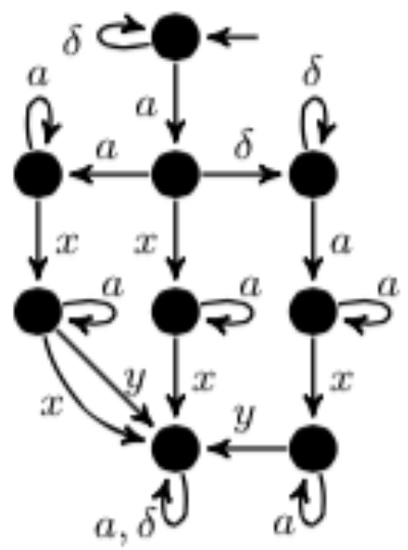
- Using concepts of **ioco** theory.



Labelled Transition System

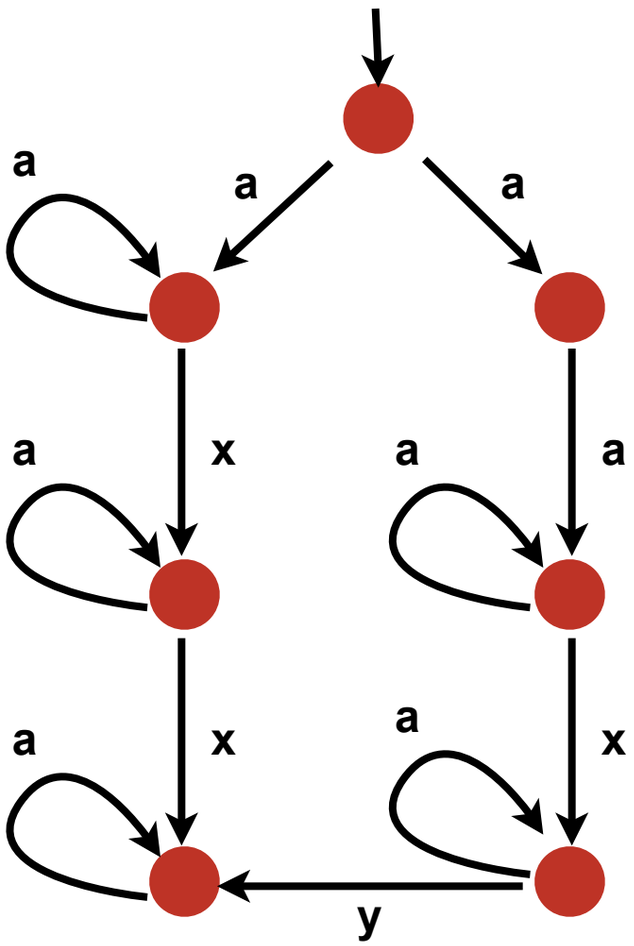
		ϵ
S	ϵ	$\{\delta\}$
	a	$\{x, \delta\}$
	aa	$\{x\}$
	aax	$\{x, y\}$
$S \cdot L_\delta$	δ	$\{\delta\}$
	ax	$\{x\}$
	$a\delta$	$\{\delta\}$
	aaa	$\{x\}$
	$aaxa$	$\{x, y\}$
	$aaxx$	$\{\delta\}$
	$aaxy$	$\{\delta\}$

Observation table



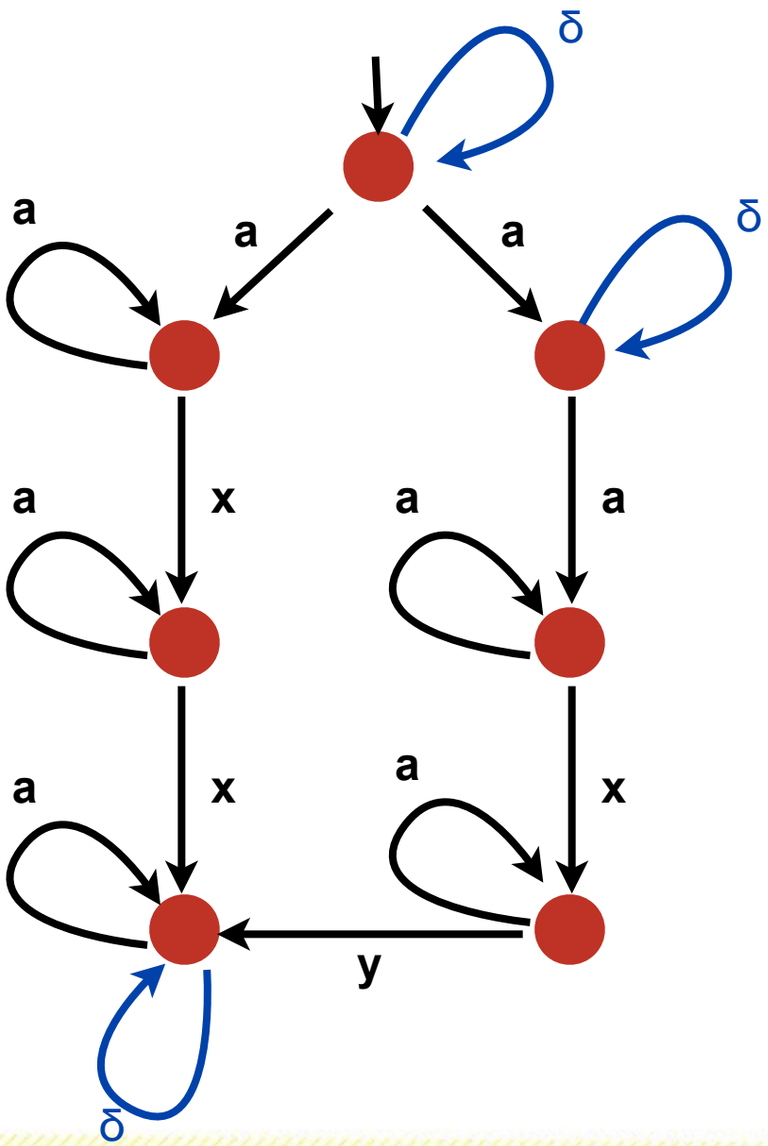
Learned automaton

Learning Labelled Transition Systems - ioco



Nondeterministic labelled
transition systems
with inputs {a} and outputs {x,y}

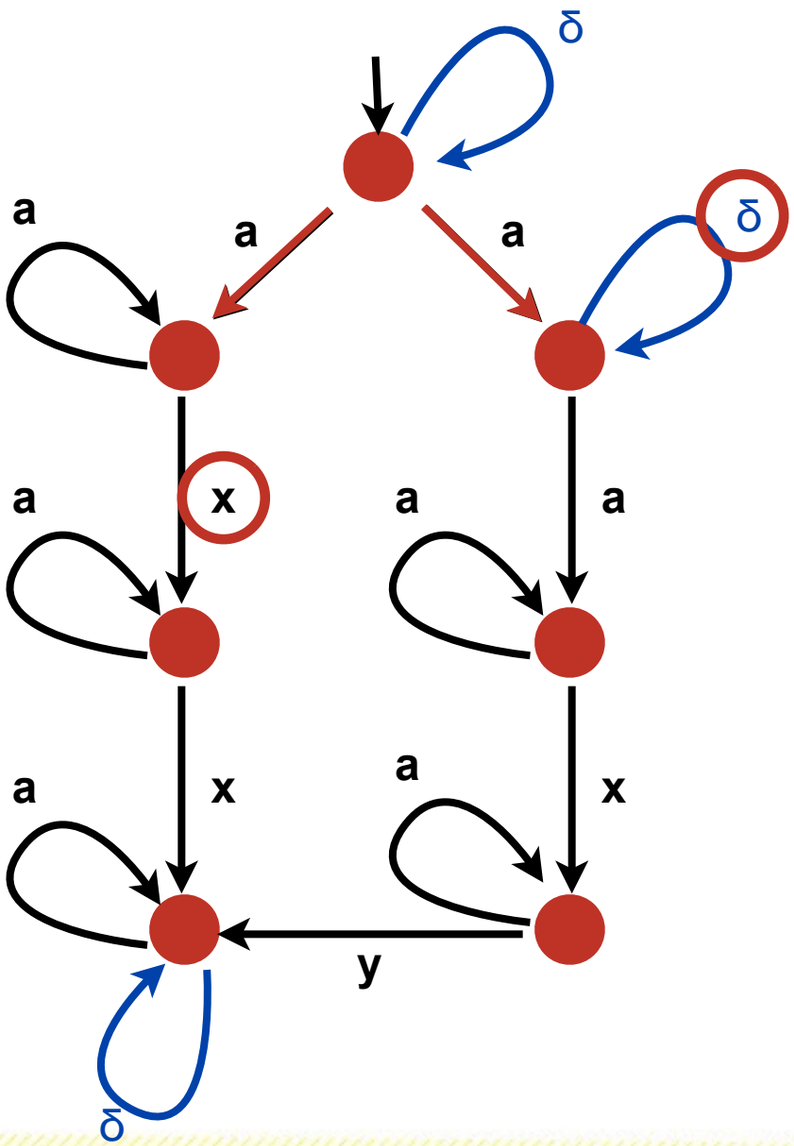
Learning Labelled Transition Systems - ioco



Nondeterministic labelled
transition systems
with inputs {a} and outputs {x,y}

Absence of output
as a special output label δ

Learning Labelled Transition Systems - ioco

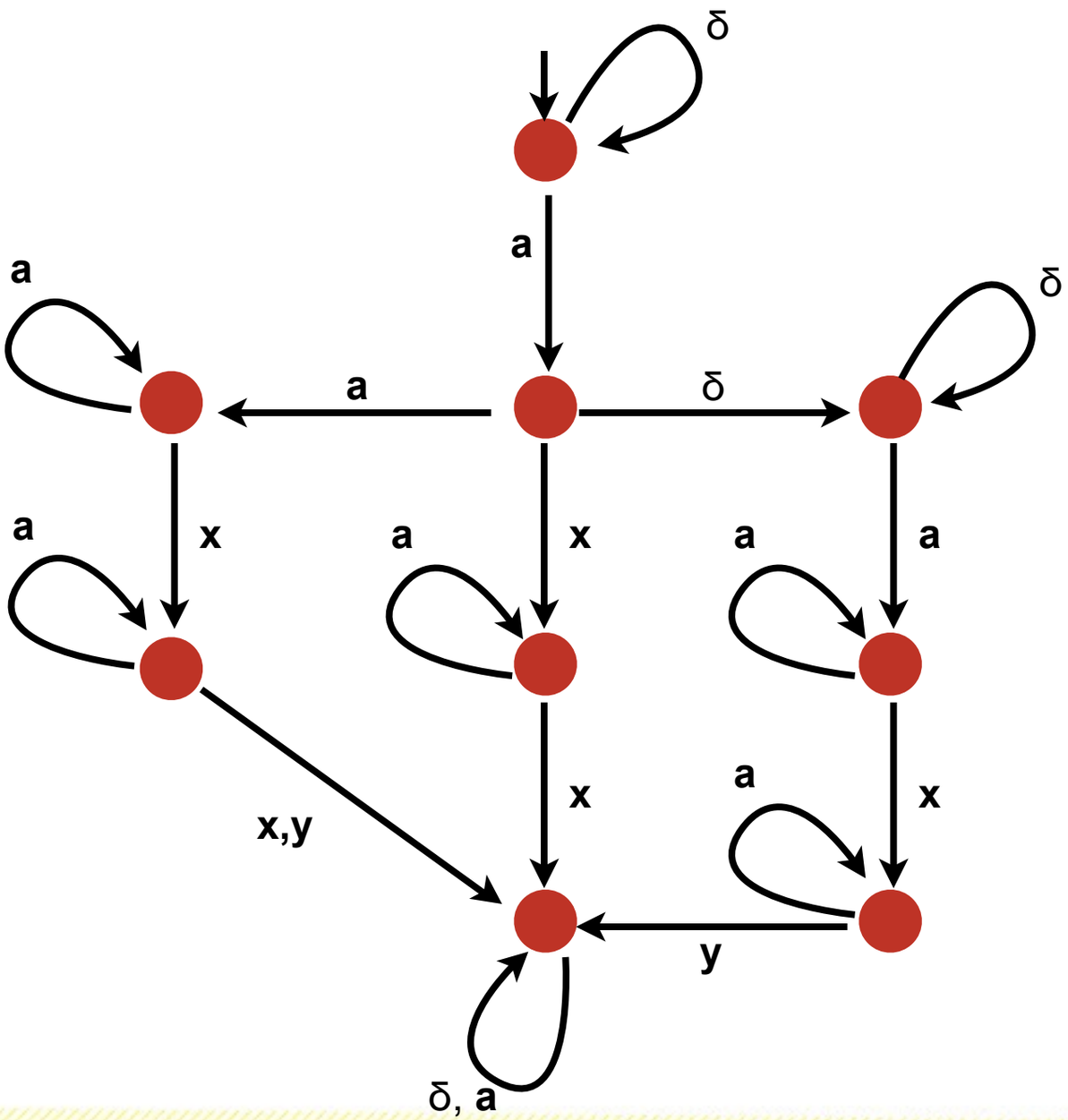


Nondeterministic labelled
transition systems
with inputs {a} and outputs {x,y}

Absence of output
as a special output label δ

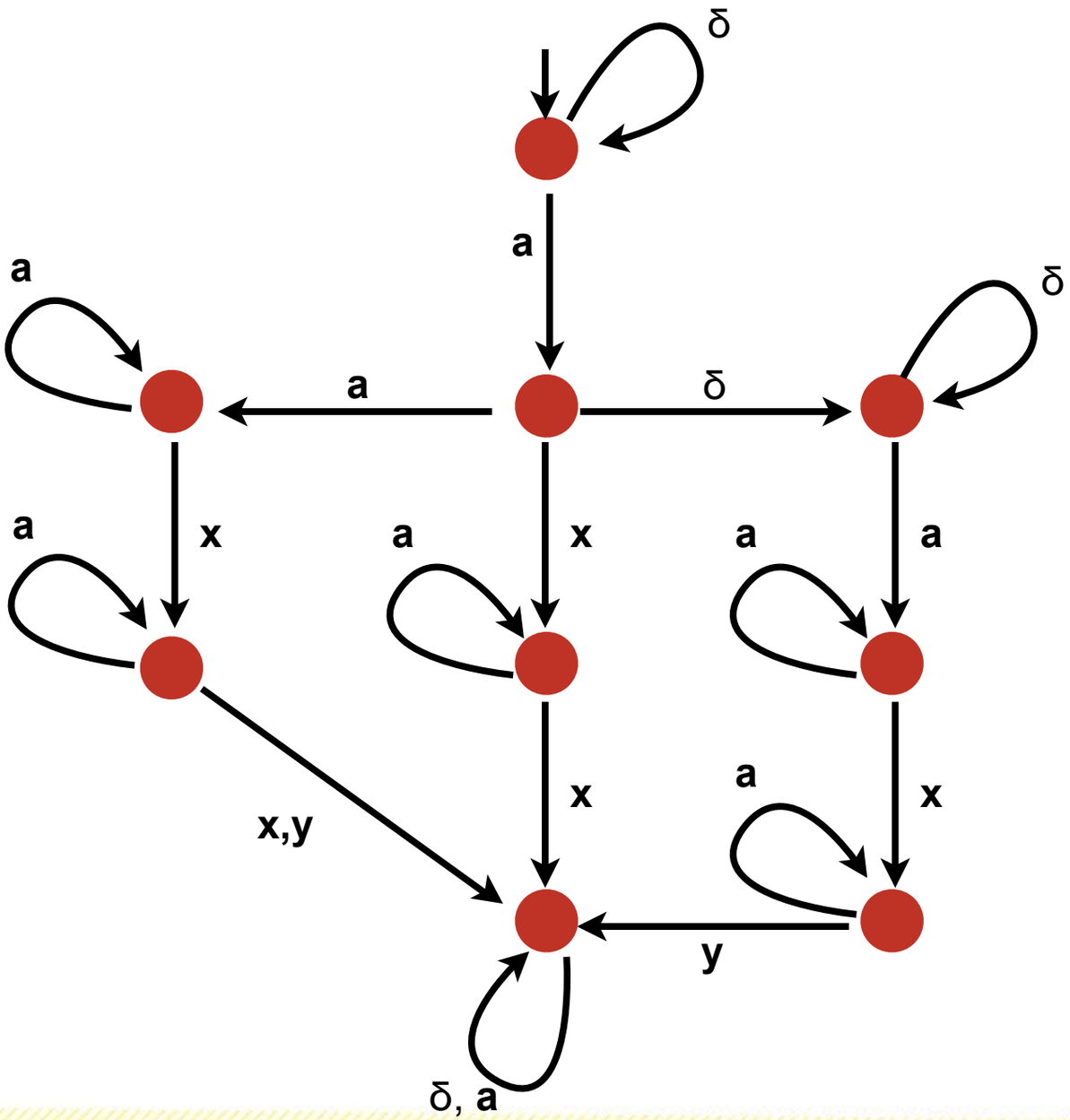
Output set ➔ observed output
after a given trace:
output after ‘a’ = {x, δ}

Learning Labelled Transition Systems - ioco



Suspension automaton: deterministic version of a labelled transition system

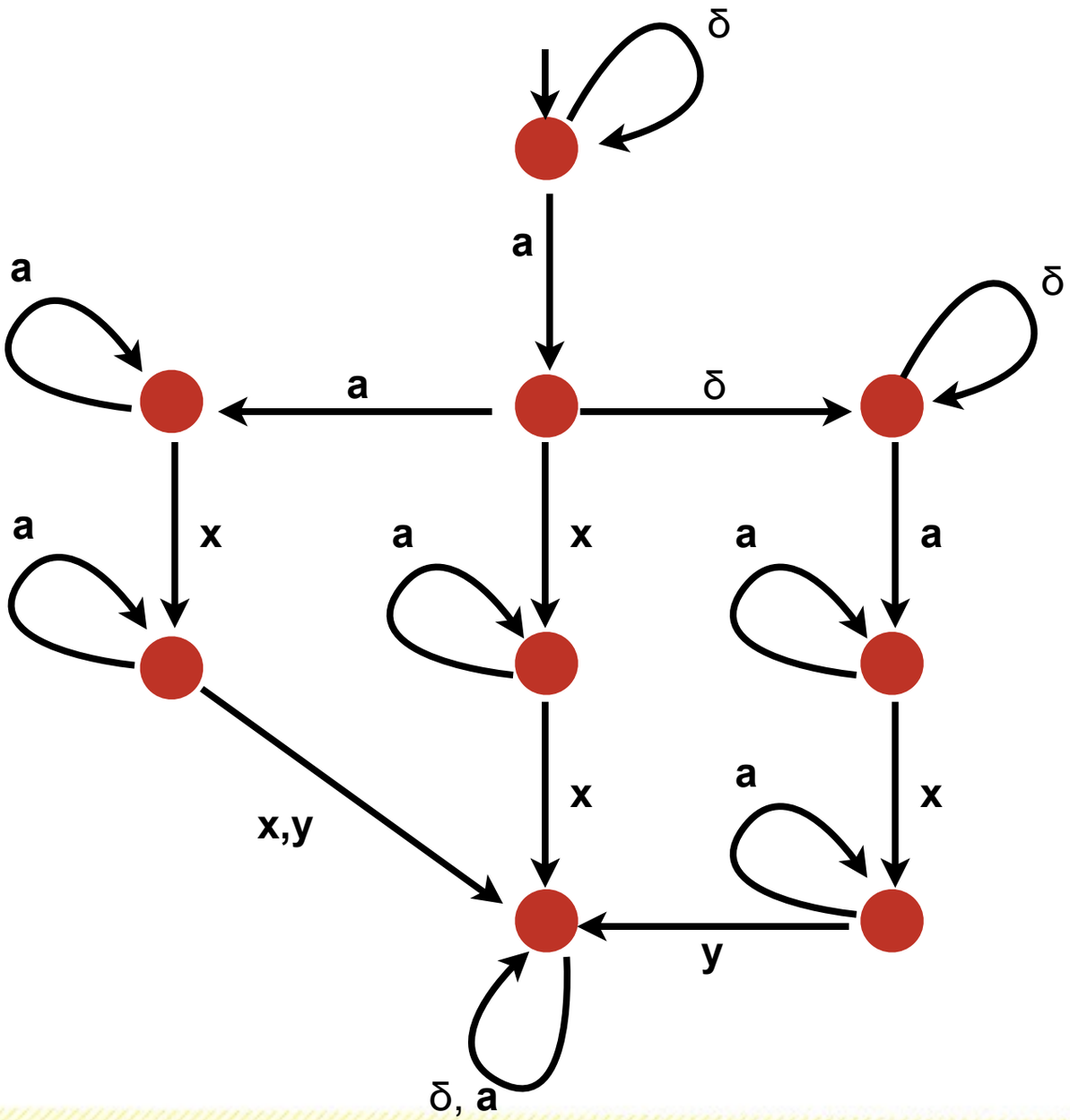
Learning Labelled Transition Systems - ioco



Suspension automaton:
deterministic version of a
labelled transition system

Non-blocking (“output” enabled)

Learning Labelled Transition Systems - ioco

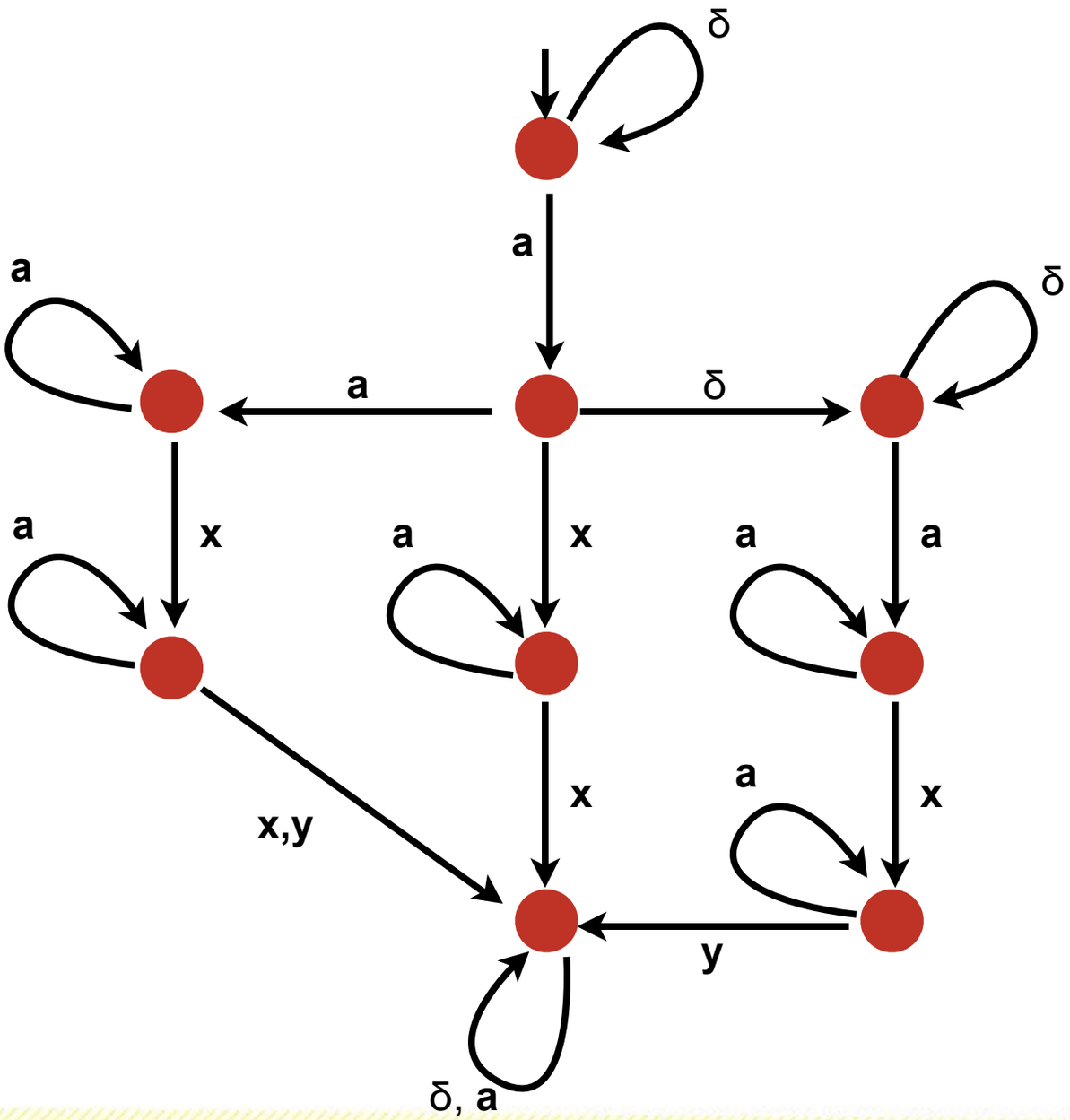


Suspension automaton: deterministic version of a labelled transition system

Non-blocking (“output” enabled)

Anomaly-free: after δ, only inputs (or δ)

Learning Labelled Transition Systems - ioco



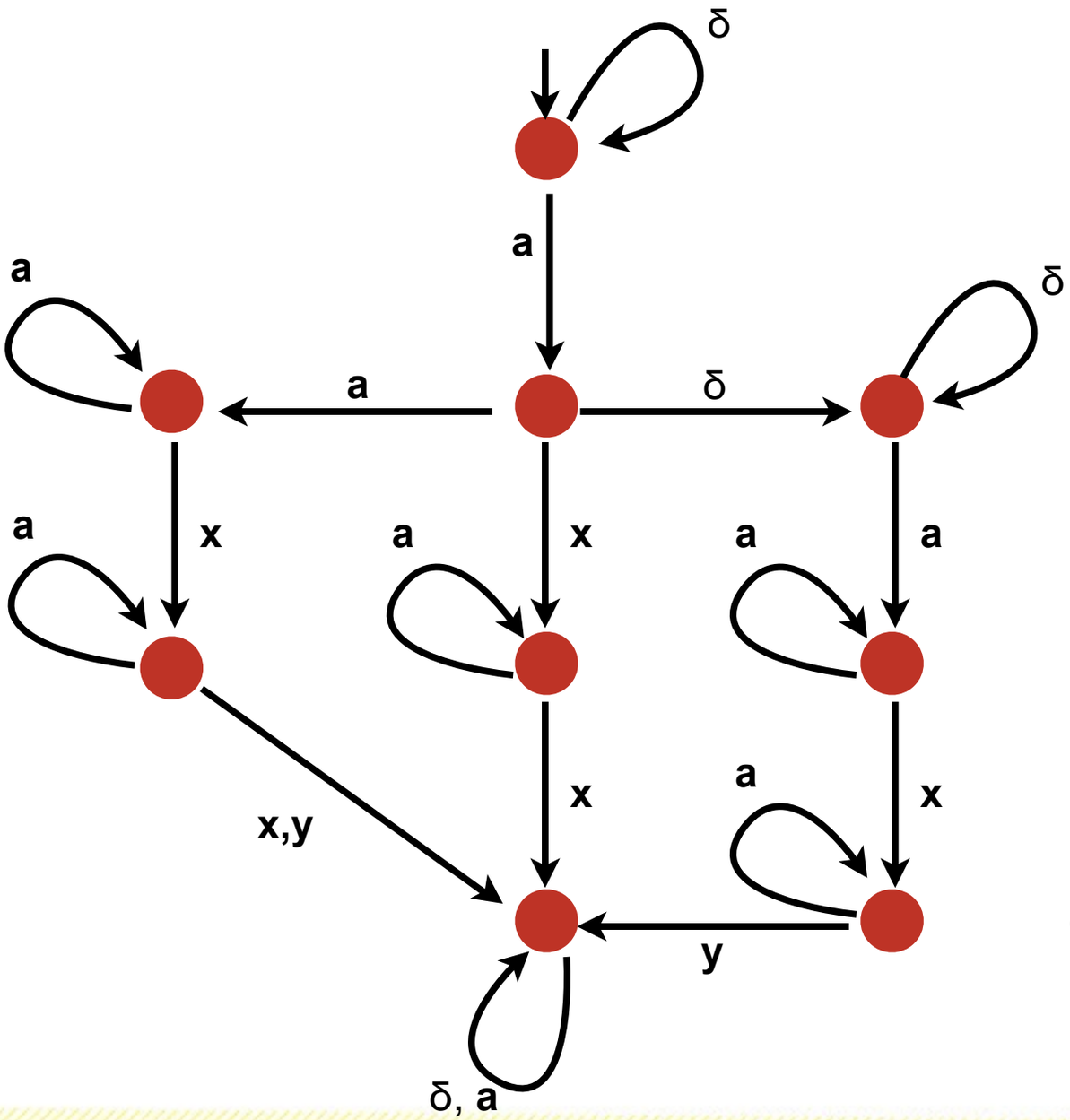
Suspension automaton: deterministic version of a labelled transition system

Non-blocking (“output” enabled)

Anomaly-free: after δ, only inputs (or δ)

Stable: sequences of δ do not change the behavior

Learning Labelled Transition Systems - ioco



Suspension automaton: deterministic version of a labelled transition system

Non-blocking ("output" enabled)

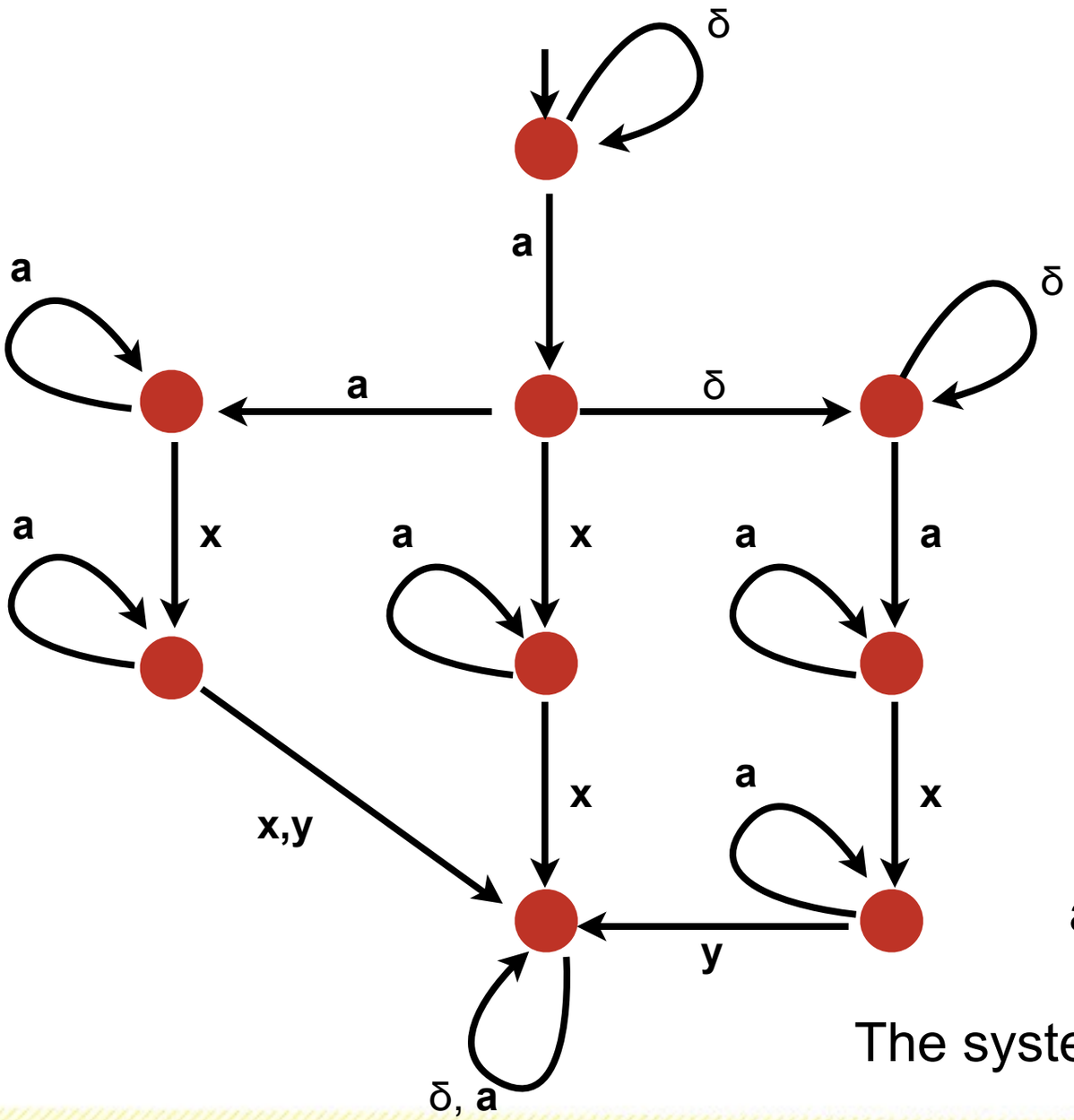
Anomaly-free: after δ, only inputs (or δ)

Stable: sequences of δ do not change the behavior

Quiescence reducible: behavior after δ can be observed without δ

$$\forall q \in Q, \forall \sigma \in L_\delta^* \,.\, \delta{\cdot}\sigma \in traces(q) \Rightarrow \sigma \in traces(q),$$

Learning Labelled Transition Systems - ioco



Suspension automaton: deterministic version of a labelled transition system

Non-blocking ("output" enabled)

Anomaly-free: after δ, only inputs (or δ)

Stable: sequences of δ do not change the behavior

Quiescence reducible: behavior after δ can be observed without δ

The system is still output-nondeterministic!

Learning Labelled Transition Systems - Assumptions

- Testing assumption (unavoidable)
- Perfect equivalence oracle (also in original L*)
- All weather condition (strong)
  - we can retrieve an output set in linear time

Learning Labelled Transition Systems - Observation Table

Outputs are possible in prefixes and suffixes

Each entry is a set of outputs

No alternation input / output

Learner	ε
ε	{δ}
a	{x,δ}
aa	{x}
aax	{x,y}
δ	{δ}
ax	{x}
aδ	{δ}
aaa	{x}
aaxa	{x,y}
aaxx	{δ}

A table should represents a valid suspension automaton.
Equivalence query checks for “trace” equivalence.

Learning Labelled Transition Systems - Advantages

- Prefix-closed + Non-blocking: save membership queries
- Anomaly-free: fill some entries without asking a membership query
- Stable: avoid to add some rows to the table
- Quiescence reducible: **save equivalence queries**

Learning Labelled Transition Systems - Advantages

Learner	ε
ε	{δ}
a	{x,δ}
aa	{x}
aax	{x,y}
δ	{δ}
ax	{x}
aδ	{δ}
aaa	{x}
aaxa	{x,y}
aaxx	{δ}
x	∅

Learning Labelled Transition Systems - Advantages

Learner	ε
ε	{δ}
a	{x,δ}
aa	{x}
aax	{x,y}
δ	{δ}
ax	{x}
aδ	{δ}
aaa	{x}
aaxa	{x,y}
aaxx	{δ}
x	∅

Prefix closedness:
no need
ask x or any
extension

Learning Labelled Transition Systems - Advantages

Anomaly-free: this entry is filled automatically

Learner	ε
ε	{δ}
a	{x,δ}
aa	{x}
aax	{x,y}
δ	{δ}
ax	{x}
aδ	{δ}
aaa	{x}
aaxa	{x,y}
aaxx	{δ}
x	∅

Prefix closedness: no need ask x or any extension

Learning Labelled Transition Systems - Advantages

Anomaly-free: this entry is filled automatically

Learner	ε
ε	{δ}
a	{x,δ}
aa	{x}
aax	{x,y}
δ	{δ}
ax	{x}
aδ	{δ}
aaa	{x}
aaxa	{x,y}
aaxx	{δ}
x	∅

Stable: aδδ will never appear as a row

Prefix closedness: no need ask x or any extension

Learning Labelled Transition Systems - Advantages

Anomaly-free: this entry is filled automatically

Stable: aδδ will never appear as a row

Learner	ε
ε	{δ}
a	{x,δ}
aa	{x}
aax	{x,y}
δ	{δ}
ax	{x}
aδ	{δ}
aaa	{x}
aaxa	{x,y}
aaxx	{δ}
x	∅

Prefix closedness: no need ask x or any extension

Quiescence reducibility saves equivalence queries (not in this lecture)

Dropping the assumptions - Challenges

- Nondeterminism: when do we stop to ask (the same) membership query?
- If we decide to stop asking a query, but we are not sure to have observed all the outputs after that query, can we still construct a valid hypothesis?
- Is there a relation between a (incomplete) hypothesis and the target system?
- Quality of successive hypothesis?

Radboud University

Attributions

Page 6: ”Megan and The Twinge Play Toca Doctor" by Erik Peacock is licensed under CC BY-NC-SA 2.0
Page 6: ”Child and mother with Apple iPad" by Inter Free Press is licensed under CC BY-NC-SA 2.0
Page 7: ”iPad Mini with Retina Display" by Neil Turner is licensed under CC BY-SA 2.0
Page 10: ”Nederlanden paspoort 2011" by Blagomeni is licensed under CC BY-SA 3.0
Page 15: ”I Got Two More Islands!" by Erik Peacock is licensed under CC BY-NC-SA 2.0