On Automata Learning and Conformance Testing

Bengt Jonsson
Uppsala University

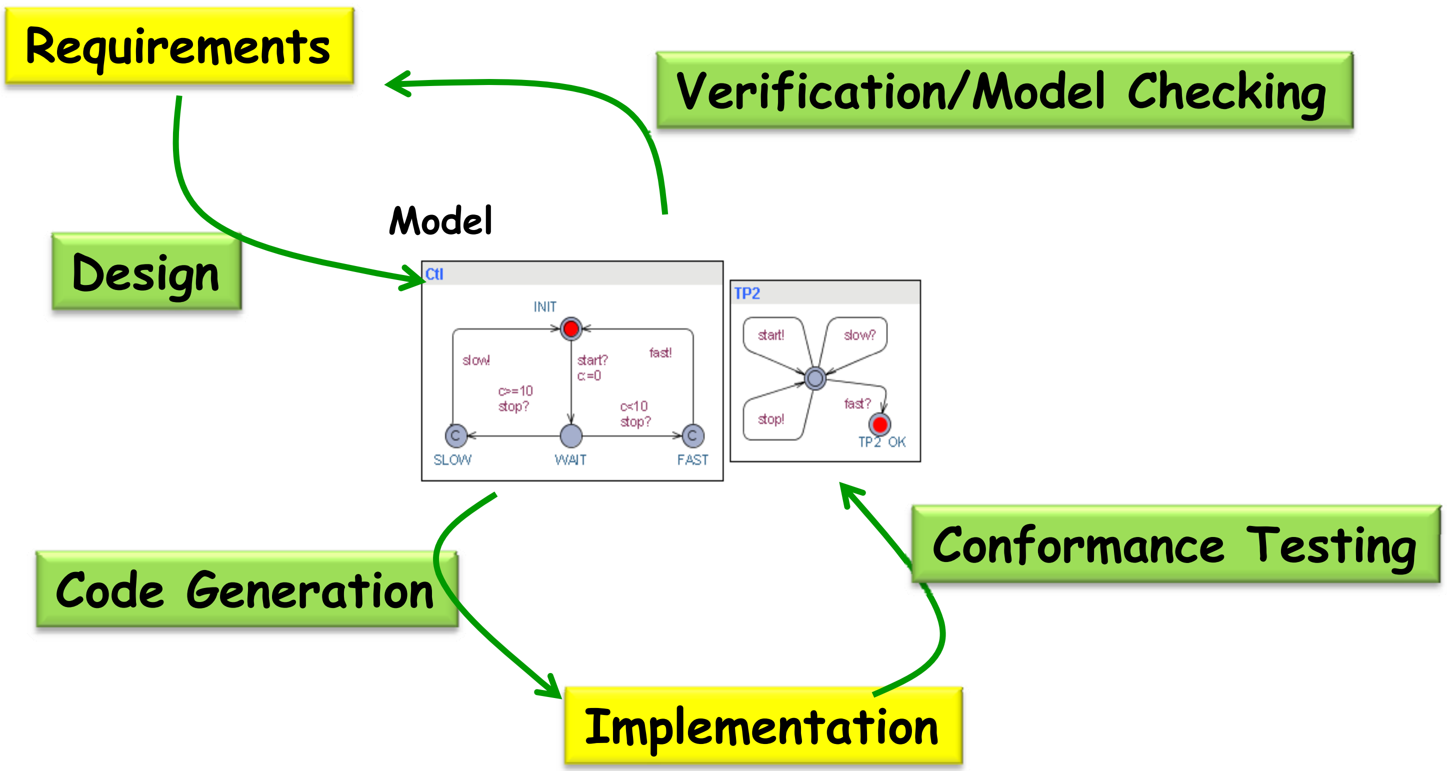
Acknowledgments

Fides Aarts, Therese Berg, Johan Blom, Olga Grinchtein, Anders Hessel, Falk Howar, Martin Leucker, Maik Merten, Paul Pettersson, Harald Raffelt, Bernhard Steffen, Johan Uijen

Outline

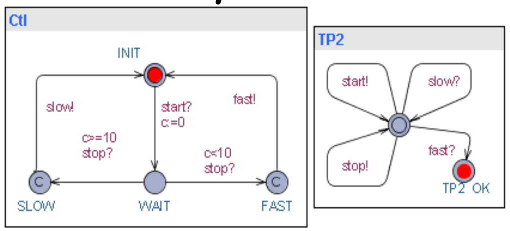
- Motivation
- Automata Learning
- Conformance Testing, Model Checking
- Extensions to richer Automata Models
- Applications in Protocol Model Generation

Modeling in System Development



Model BasedTest Generation

Model:
What the system should do



Test case generator

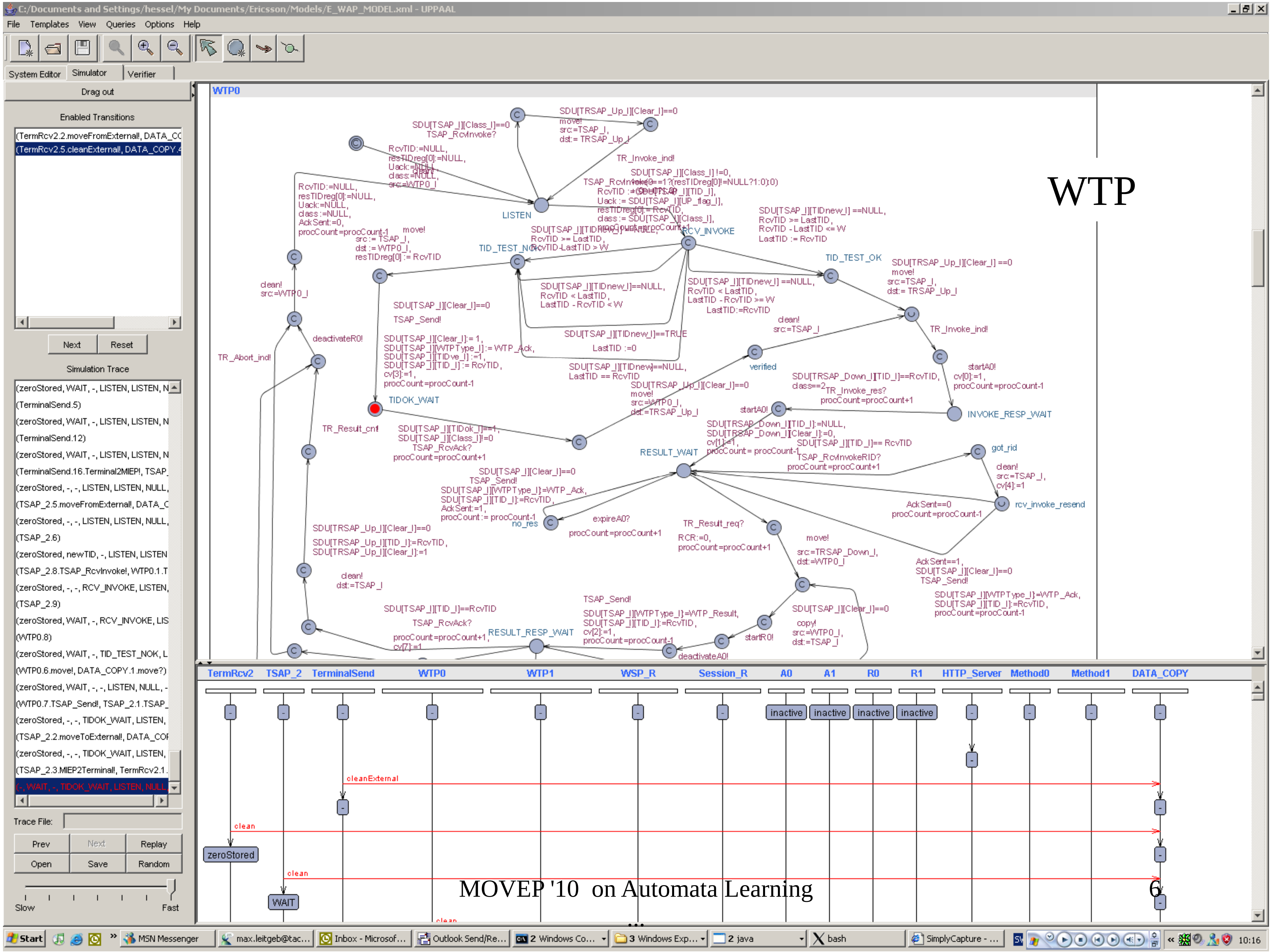
Test Objective:
What should be tested?

Test Suite

Well-Developed Tools:
- TGV, TorX, Gotcha, …
- Conformic Qtronic, …

Implementation Under Test

WTP



C:/Documents and Settings/hessel/My Documents/Ericsson/Models/E_WAP_MODEL.xml - UPPAAL
File Templates View Queries Options Help
System Editor
Simulator
Verifier
Drag out
Enabled Transitions
Next
Reset
Simulation Trace
Trace File:
Prev
Next
Replay
Open
Save
Random
Slow
Fast
WTP0
LISTEN
RCV_INVOKE
TID_TEST_NOK
TID_TEST_OK
TIDOK_WAIT
INVOKE_RESP_WAIT
RESULT_WAIT
RESULT_RESP_WAIT
rcv_invoke_resend
TermRcv2
TSAP_2
TerminalSend
WTP0
WTP1
WSP_R
Session_R
A0
A1
R0
R1
HTTP_Server
Method0
Method1
DATA_COPY
inactive
cleanExternal
clean
zeroStored
WAIT

Modeling Gap

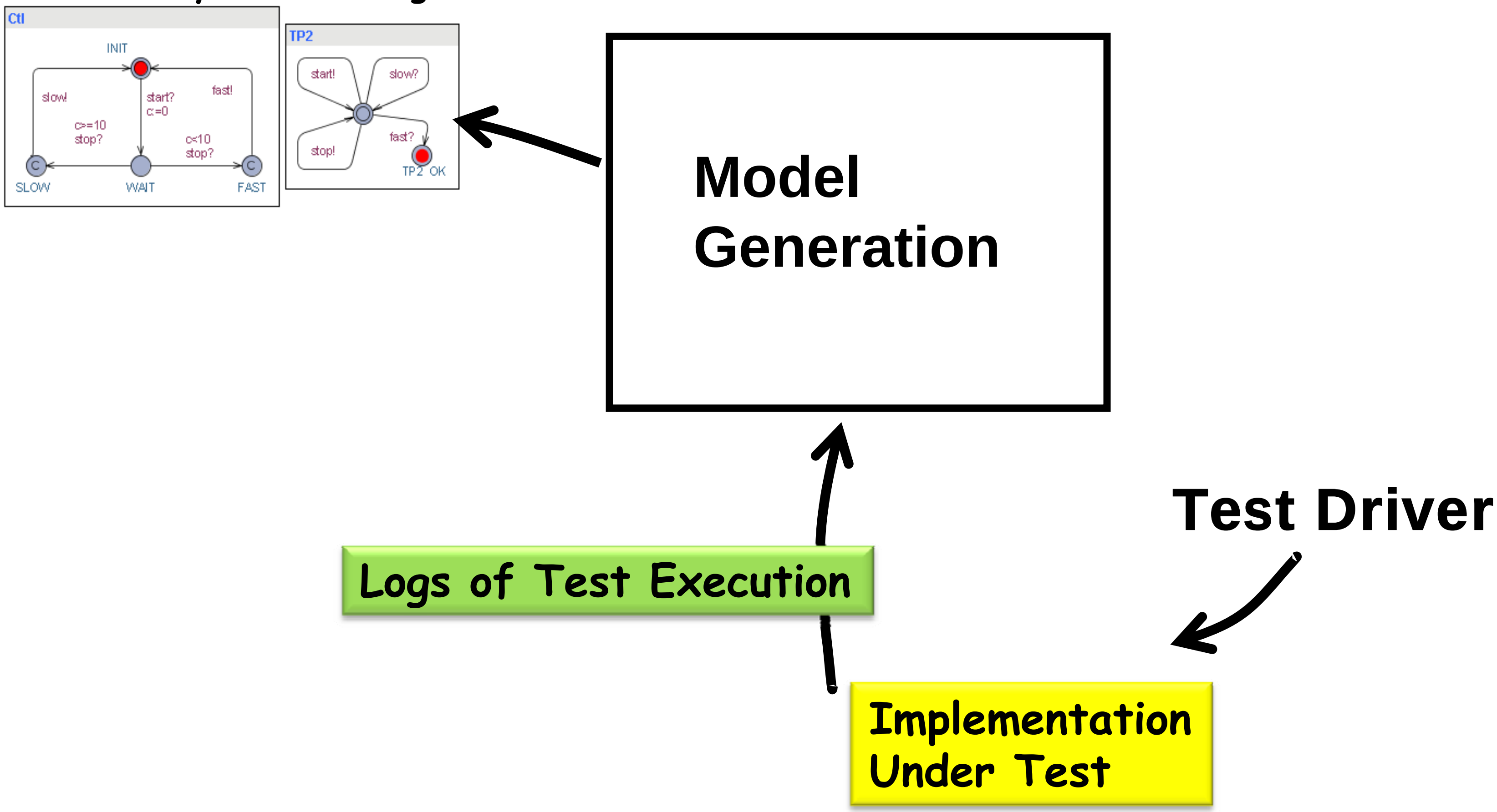
- **Typically, models are not available**

- **Modeling SUT** [system under test] is among **biggest obstacles** in Model Based Testing [A. Hartman]

- What to do if there is no model?
  (the norm in practice)

Supporting Model Generation

Model:
What the system is doing



How to support generation of models?

- Model Behavior of existing implementation
  - By observations gained during extensive testing
- Potential Applications:
  - Regression testing
  - Migrating from manual to model-based testing
  - Modeling environment of SUT, libraries
- Problem: Constructing State Machines from traces/executions/words
  - Has been studied in Automata Learning

Simplest form of Automata Learning

- From sample of words
- find simple(st) state machine that explains them

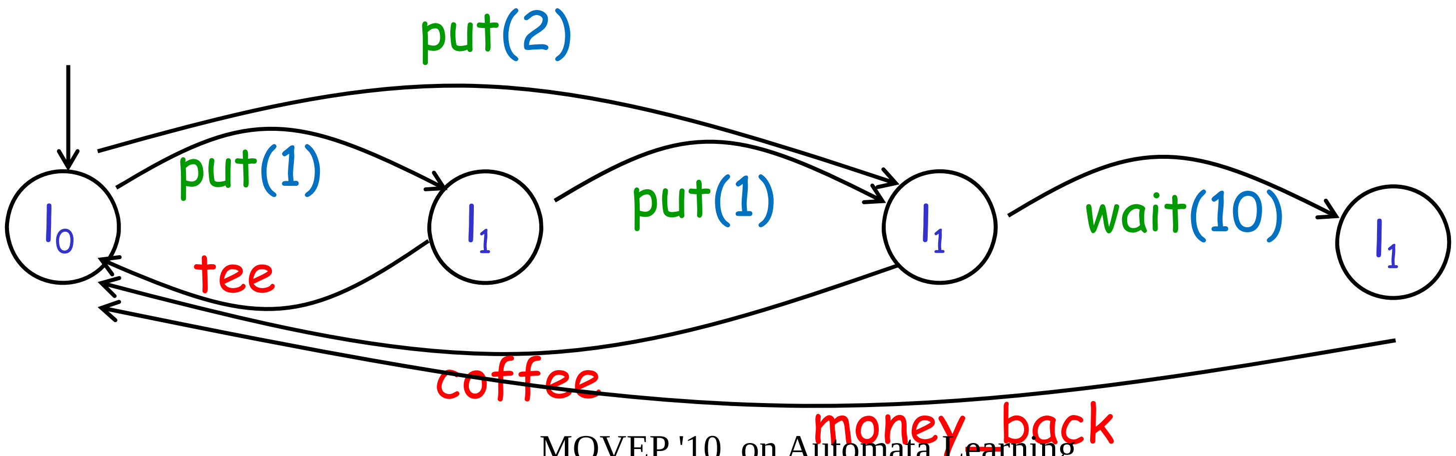
Requirements Capture

- Generate State Machine Specification from set of allowed (and disallowed) scenarios:

put(1) put(1) / coffee
put(1) / tee
put(2) /coffee
put(1) put (1) wait(10) / money_back

Instances:

- Play engine [Harel,Marelly]
- Smyle [Bollig,Katoen,Kern,Leucker]



Compositional Verification [Giannakopoulou,Pasareanu et al]

Complex Model Checking Problem:

$$E \parallel M \models \varphi$$

If

Checking E || M |= φ too complex:

Find abstraction A of E, s.t.:

E refines A

A || M |= φ

E || M |= φ

Building A using Learning

ASSUME:

w || M |= φ

can be checked for single behavior w

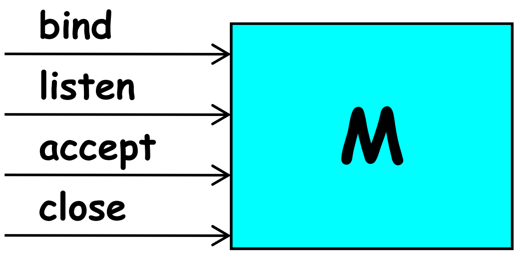
Check w || M |= φ for many w,

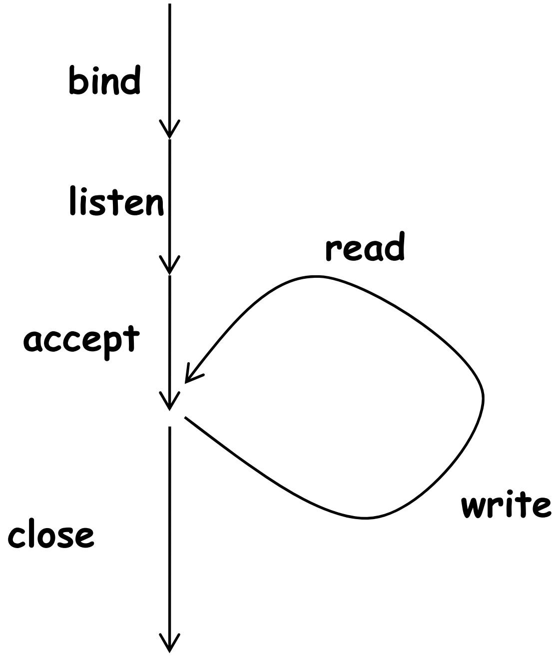
Construct A from these checks

Check whether A satisfies premises

Specification Mining[Ammons,Bodik,Larus]

API:





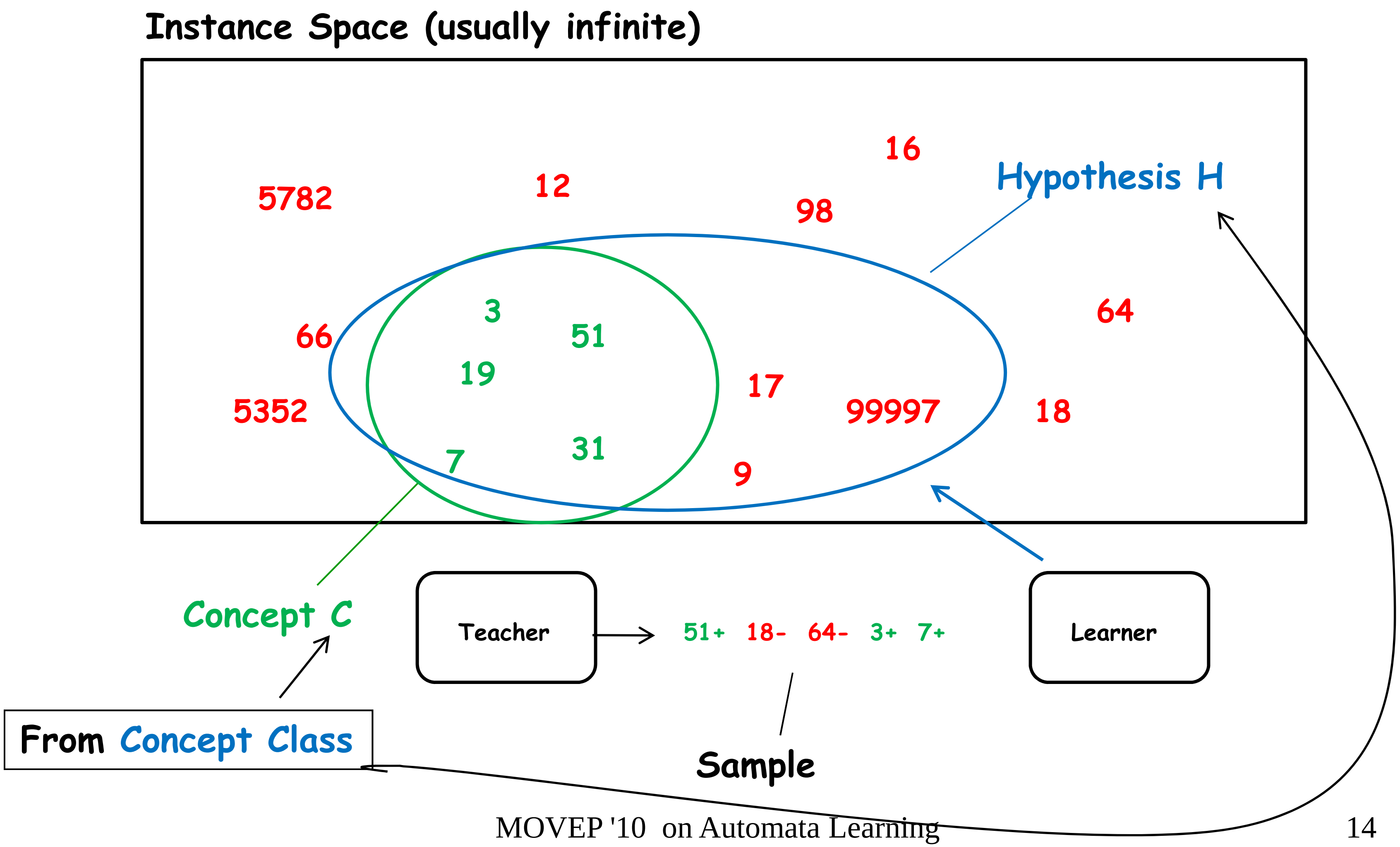
Problem: Find restrictions on how API calls may be ordered

Assume we have well-tested programs that use the API

Analyze executions of such programs.

Form an Automaton that summarize these executions.

Learning



Instance Space (usually infinite)
5782
12
16
98
Hypothesis H
66
3
51
19
64
17
5352
99997
18
7
31
9
Concept C
Teacher
51+ 18- 64- 3+ 7+
Learner
From Concept Class
Sample

Some Terminology

Given an Instance space X

- Concept is a subset of X
- Concept Class is a class of Concepts
- Sample is a (finite) set of labeled examples
  - x+ where $x \in C$
  - x- where $x \notin C$
- Learner produces Hypothesis (in Concept Class) from Sample
- Teacher knows Concept, produces Sample
  - Can also, e.g., answer queries
- Hypothesis H is correct if $H = C$
- Hypothesis H is consistent with sample if
  - if x+ in sample then $x \in H$
  - if x- in sample then $x \notin H$
- Concepts have Representations
  - size of Concept C = size of its Representation

Automata learning

- Assume finite set Σ of symbols
- Instance space: Σ^*
- Concept Class: Regular languages
- Representation of Concept: DFA
- Sample is a (finite) set of labeled words
  - w+ where $w \in L$
  - w- where $w \notin L$

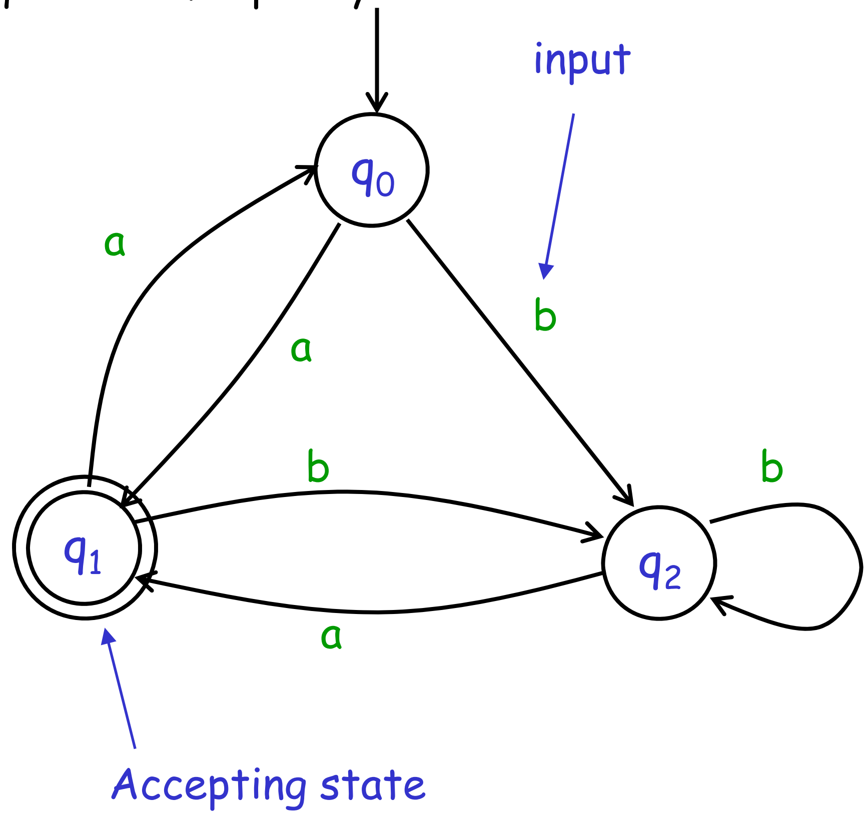
Deterministic Finite Automata (DFA)

Finite State Machines accepting sequences of input symbols

Σ	alphabet of symbols
Q	states
$\delta: Q \times \Sigma \rightarrow Q$	transition function
$F \subseteq Q$	accepting states

Assumptions:

- Deterministic
- Completely specified



Deterministic Finite Automata (DFA)

Finite State Machines accepting sequences of input symbols

Σ symbols

Q states

$\delta: Q \times \Sigma \to Q$ transition function

$F \subseteq Q$ accepting states

Assumptions:

- Deterministic
- Completely specified

Myhill-Nerode:

Given language L

For prefix u, define $L_u = \{v \mid uv \in L\}$

Nerode congruence: $u \approx u'$ iff $L_u = L_{u'}$

Unique Minimal DFA accepts regular L

Q : equivalence classes $[u]_\approx$

$\delta([u]_\approx, a) = [ua]_\approx$ transition function

$F : \{[u]_\approx \mid u \in L\}$ accepting states

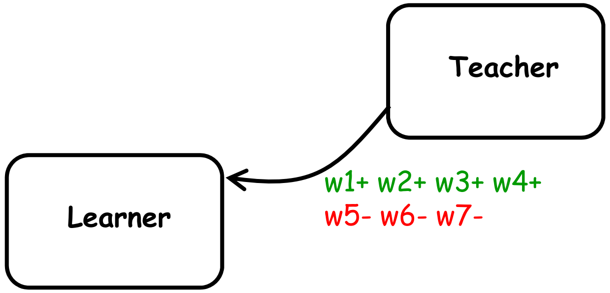
Automata Learning: Frameworks

Construct DFA from sample of accepted and rejected words.

Passive learning: sample given

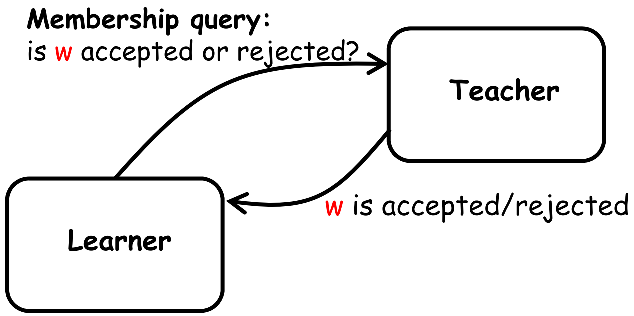
- Only accepted words (positive sample)
- Accepted and rejected words

Observing SUT/test suites



Active learning: Learner chooses words, teacher classifies

Testing SUT



Mealy Machines

Finite State Machines w. input & output

I input symbols

O output symbols

Q states

δ: $Q \times I \rightarrow Q$ transition function

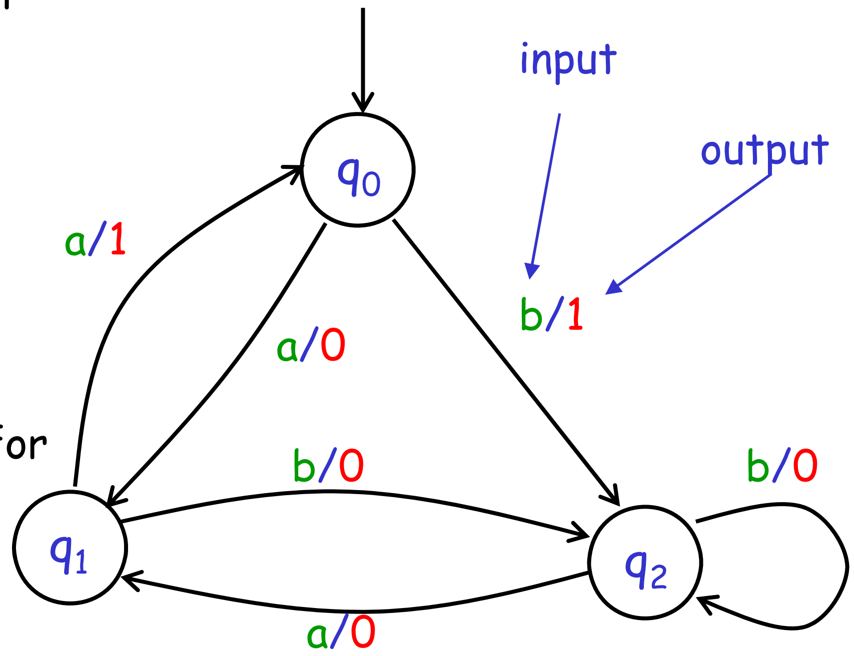
λ: $Q \times I \rightarrow O$ output function

•Often used for protocol modeling, for protocol testing techniques,

Assumptions:

•Deterministic

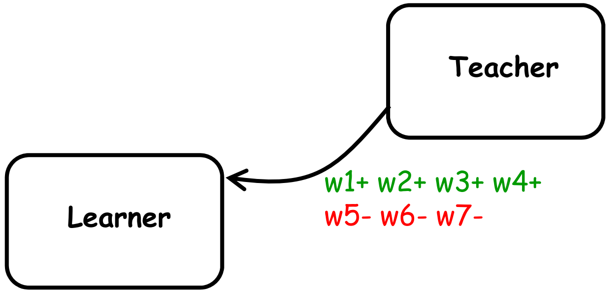
•Completely specified



Passive Learning:

Construct DFA from sample of accepted and rejected words.

- Which DFA?
- The most succinct one!
  - **which conforms to sample,**
  - and has fewest states



- Finding smallest DFA is NP-hard [Gold 78]
- Can be found by constraint solving (Biermann's algorithm)

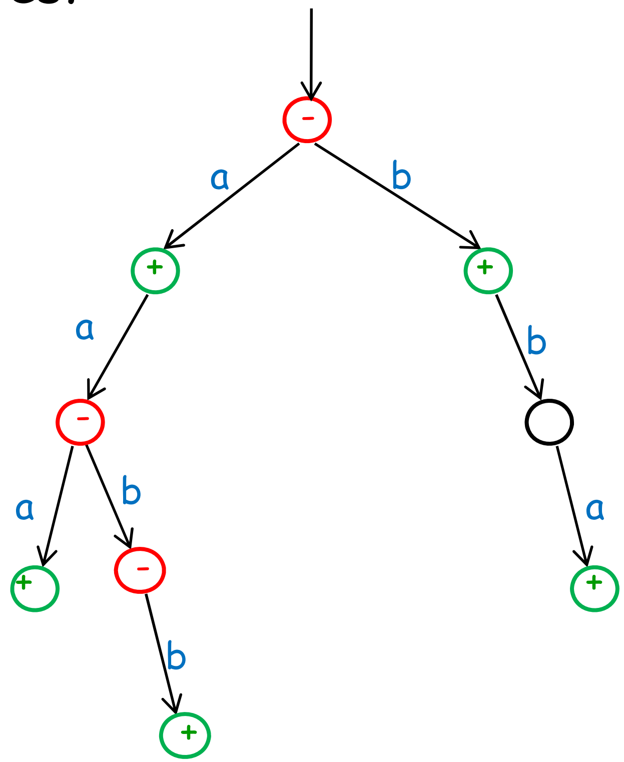
Biermann's Algorithm

Is there a conformant DFA with n states?

Encode this as a CSP problem

- Map each prefix u in tree to some state $q_u \in \{1 .. n\}$
- Subject to constraints:
  - $q_u \neq q_v$ if u accepted, v rejected
  - if ua va are prefixes, then
    $q_u = q_v$ implies $q_{ua} = q_{va}$

Try example for $n = 3$



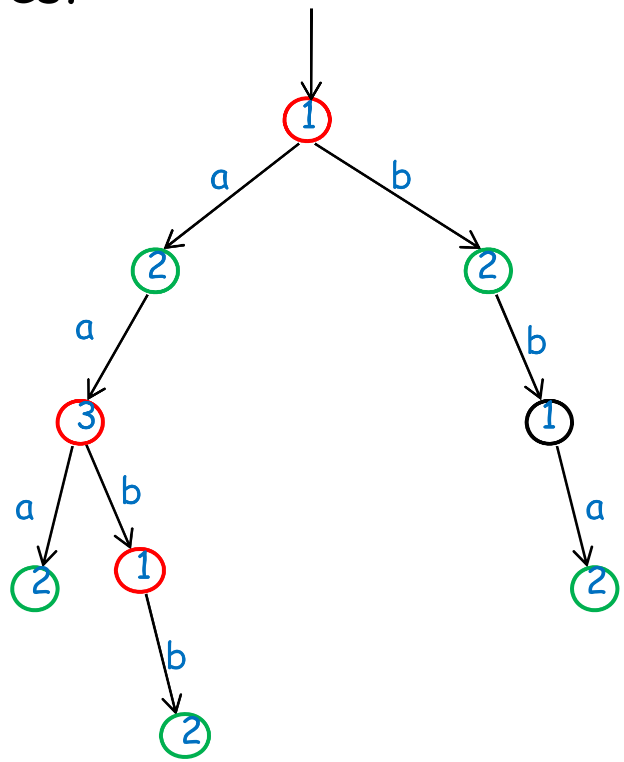
Biermann's Algorithm

Is there a conformant DFA with n states?

Encode this as a CSP problem

- Map each prefix u in tree to some state $q_u \in \{1 .. n\}$
- Subject to constraints:
  - $q_u \neq q_v$ if u accepted, v rejected
  - if ua va are prefixes, then
    $q_u = q_v$ implies $q_{ua} = q_{va}$

Try example for $n = 3$

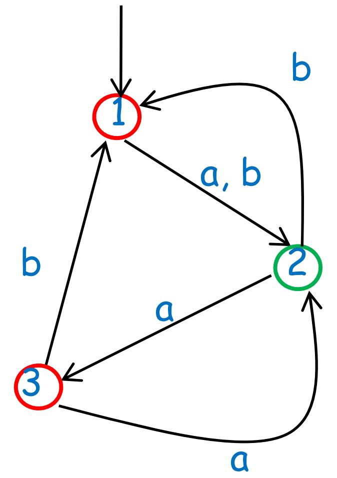


Biermann's Algorithm

Is there a conformant DFA with n states?

Encode this as a CSP problem

- Map each prefix u in tree to some state $q_u \in \{1 .. n\}$
- Subject to constraints:
  - $q_u \neq q_v$ if u accepted, v rejected
  - if ua va are prefixes, then
    $q_u = q_v$ implies $q_{ua} = q_{va}$



Try example for $n = 3$

Check

Accepted: a b aaa aabb bba

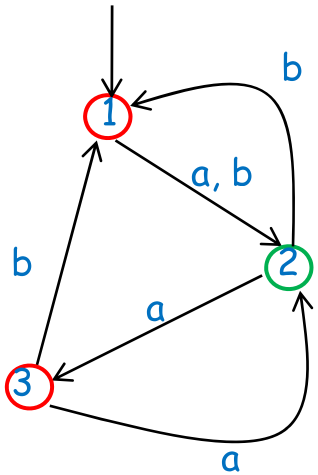
Rejected: λ aa aab

Discussion

- Problem w. Biermanns algorithm: Exponential
- Q: Is there a setting to learn automata polynomially in some way?
- By Gold's result, we cannot hope to learn minimal DFA from arbitrary sample.

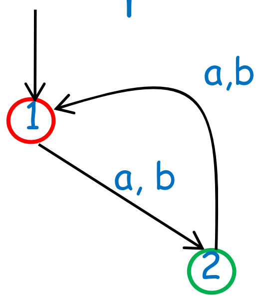
Identification in the Limit

L

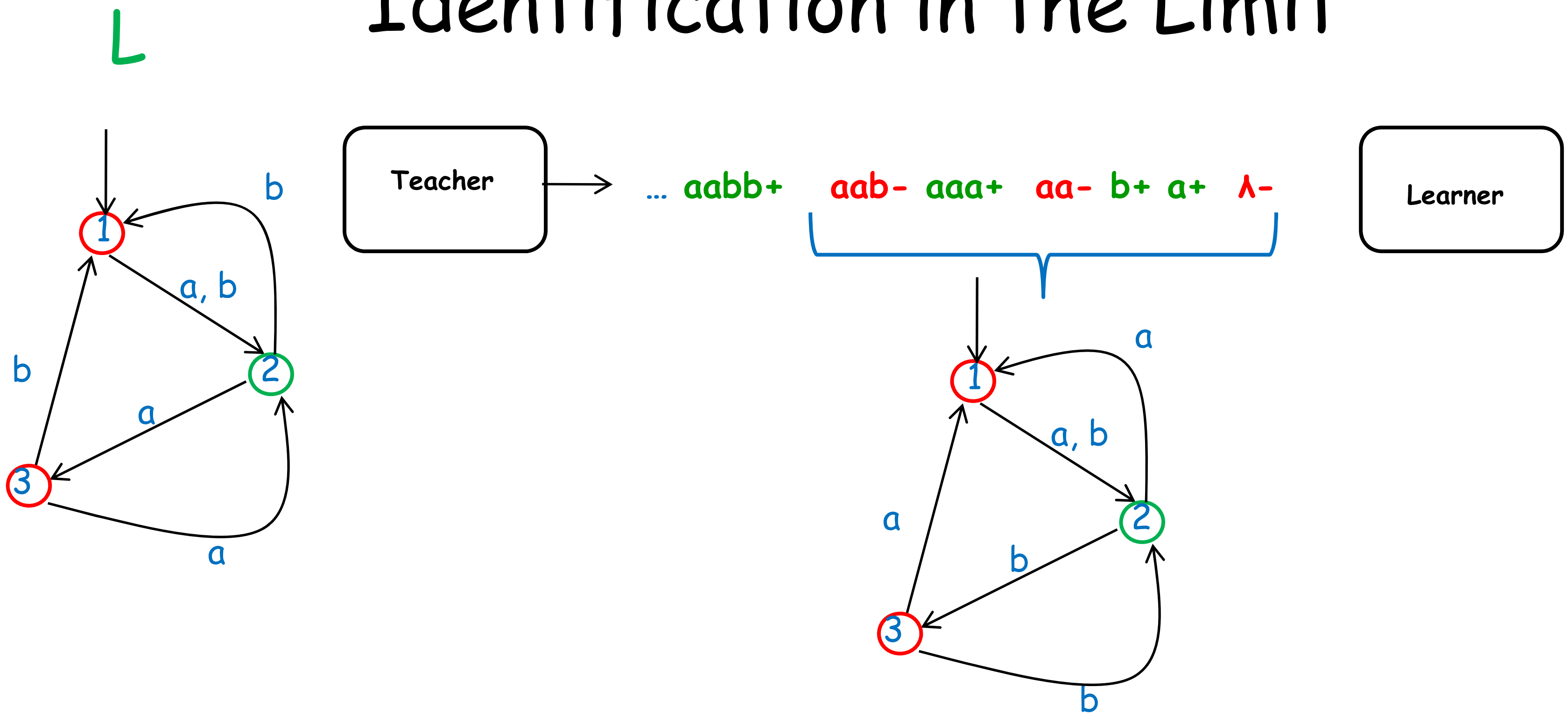


Teacher → … aabb+ aab- aaa+ aa- b+ a+ λ- Learner

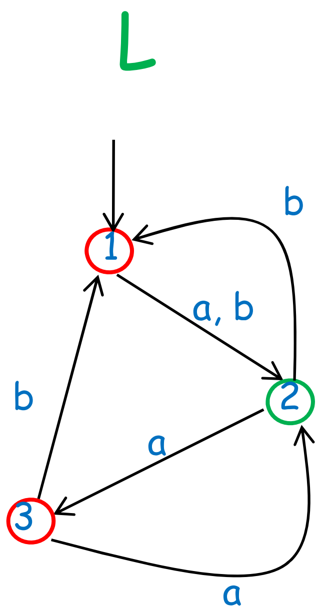
Enumeration of Σ*

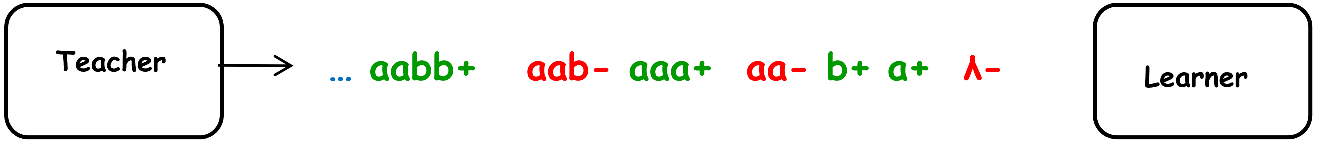


Identification in the Limit



Identification in the Limit





- Assume Teacher incrementally enumerates all words (classified) in Σ^*
- After each word, Learner can use previous words to form hypothesis H

Learner **identifies L in the limit**,

if H converges to correct hypothesis after finitely many words

Still, (exponentially) much data may be needed

Efficient Identification in the Limit

Teacher → … aabb+ aab- aaa+ aa- b+ a+ λ- Learner

Concept Class is **efficiently identifiable in the limit** if
∃polynomials p,q, s.t. for any concept C in concept class

- Learner can produce H in time O(p(|seen sample|))
- Exists sample S of size O(q(|C|)) s.t. Learner produces correct H whenever seen sample contains S

- S called "characteristic sample" for C
- S can depend on Learner

Observations

if Concept class is efficiently identifiable in the limit, then

- Learner needs polynomial time to produce hypothesis
- Concepts characterized by polynomial-size characteristic sets
- With "helpful" Teacher, the Learner needs only polynomially much data to infer *C*
- With "unhelpful" Teacher, the Learner may need a lot of data to infer *C*
- Learner should work well for characteristic sets, should make "reasonable" hypotheses otherwise.

Characteristic Samples

A characteristic sample S for C should uniquely characterize C in the following sense:

Learner should produce hypothesis C from any sample that contains S and is consistent with C

Implies that if

- S is characteristic sample for C and
- S' is characteristic sample for C'

then either

- C is inconsistent with S' or
- C' is inconsistent with S
- (otherwise what to do with $S \cup S'$?)

Characteristic Samples for DFAs

A characteristic sample for L should identify its DFA.
This can be done by

- Demonstrating that there are n states
  - Each state represented by access string u

    u represents $\delta(q_0,u)$
- For each state q and symbol a,

  uniquely identify $\delta(q,a)$

Separating Sequences

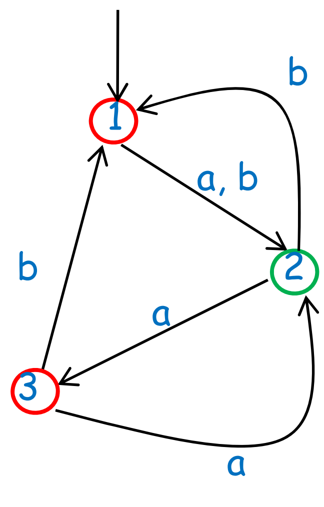
A **separating sequence** for q and q' is a suffix v such that

$\delta(q,v)$ is accepting and $\delta(q',v)$ is rejecting (or vice versa)

1 2 : λ

1 3 : b (not a)

2 3 : λ



...

Separating Sequences

A **separating sequence** for q and q' is a suffix v such that

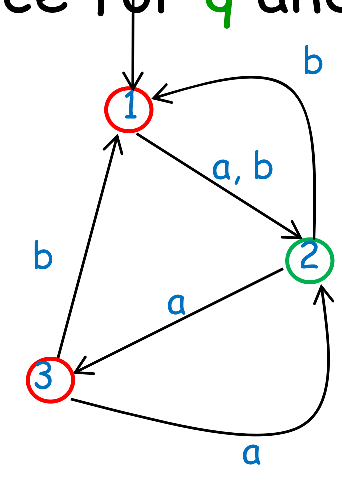
$\delta(q,v)$ is accepting and $\delta(q',v)$ is rejecting (or vice versa)

A **separating family of DFA** is a family of sets

$\{ Z_q \mid q \text{ is a state of DFA}\}$

s.t. $Z_q \cap Z_{q'}$ contains separating sequence for q and q'

1 : λ b
2 : λ
3 : λ b



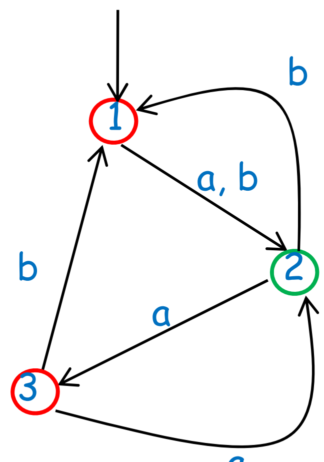
Separating Sequences

A **separating family of DFA** is a family of sets

$\{ Z_q \mid q$ is a state of DFA$\}$

s.t. $Z_q \cap Z_{q'}$ contains separating sequence for q and q'

If all Z_q are equal (to W), then W is a **characterizing set**



1	:	λ	b
2	:	λ	
3	:	λ	b

Separating Sequences

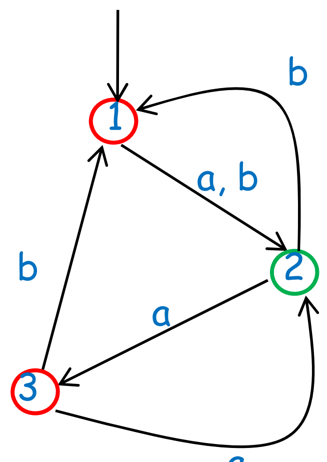
A **separating family of DFA** is a family of sets

$\{ Z_q \mid q$ is a state of DFA$\}$

s.t. $Z_q \cap Z_{q'}$ contains separating sequence for q and q'

If all Z_q are equal (to W), then W is a **characterizing set**

W : λ b



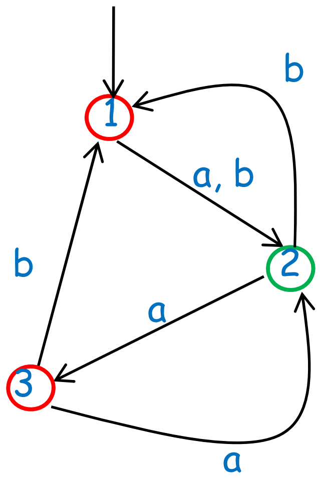
Characteristic Sample

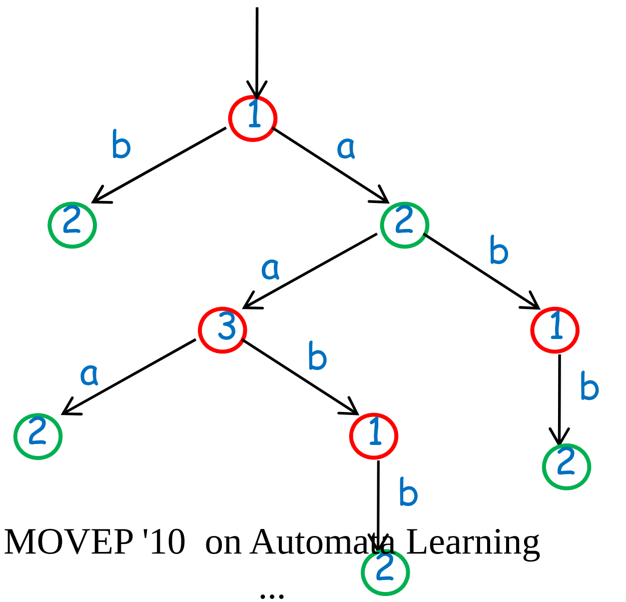
Let Sp(L) be prefixes in minimal spanning tree of DFA(L)

Let K(L) be $\{ ua \mid u \in Sp(L) \quad a \in \Sigma \}$

Let Characteristic Sample be

$$Sp(L) \cup \{ uv \mid u \in Sp(L) \cup K(L) \quad v \in Z_{q_u} \}$$



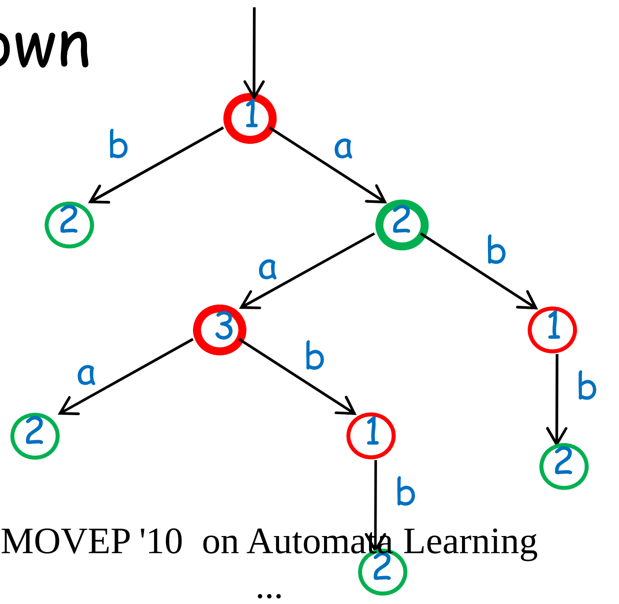


Λ
a
aa
b
ab
aaa
aab
abb
aabb

Why characteristic sample?

When forming DFA from prefix tree:

- The states $\{q_u \mid u \in Sp(L)\}$ cannot be merged
  - since they are separated by suffixes
- Each state in $\{q_u \mid u \in K(L)\}$ can be merged with at most one state in $\{q_u \mid u \in Sp(L)\}$
- Easy to construct minimal DFA from sample
  - if $Sp(L)$ is known

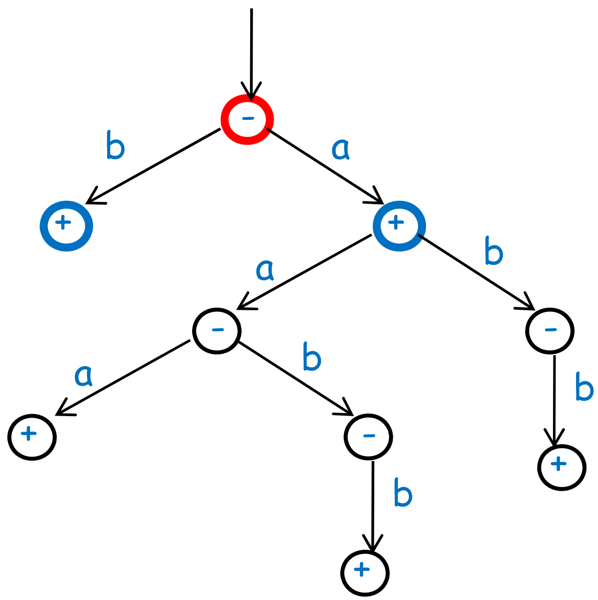


Λ
a
aa
b
ab
aaa
aab
abb
aabb

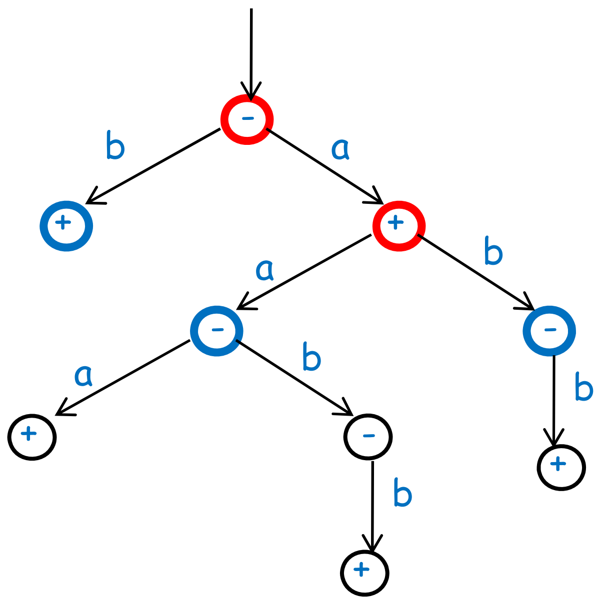
State Merging Algorithms

- Traverse the prefix tree from root
- For each new state
  - if possible, merge it with some seen state
  - Otherwise, promote it to a new state in the resulting DFA
- Red states are determined to become DFA states
- Blue states (frontier) are the successors of red states, waiting to be candidates for merging with red states.
- Repeatedly
- Merge blue with red if no inconsistency results
- "Unmergeable" blue state becomes red

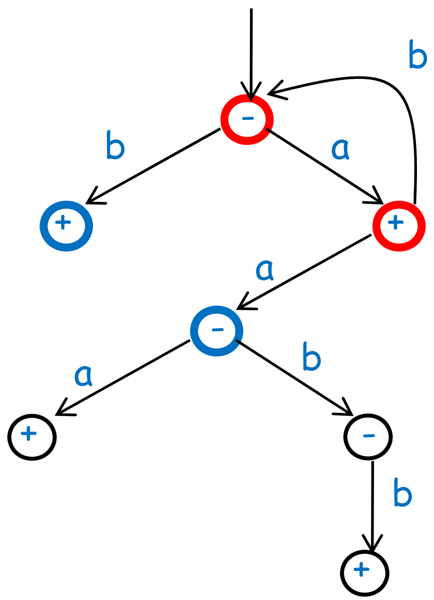
State Merging: Example



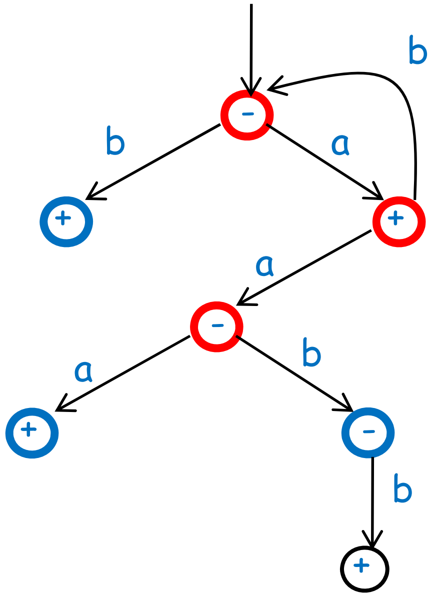
State Merging: Example



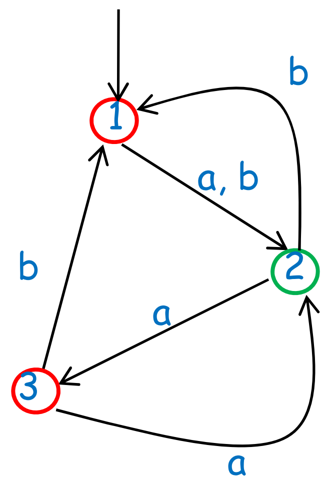
State Merging: Example



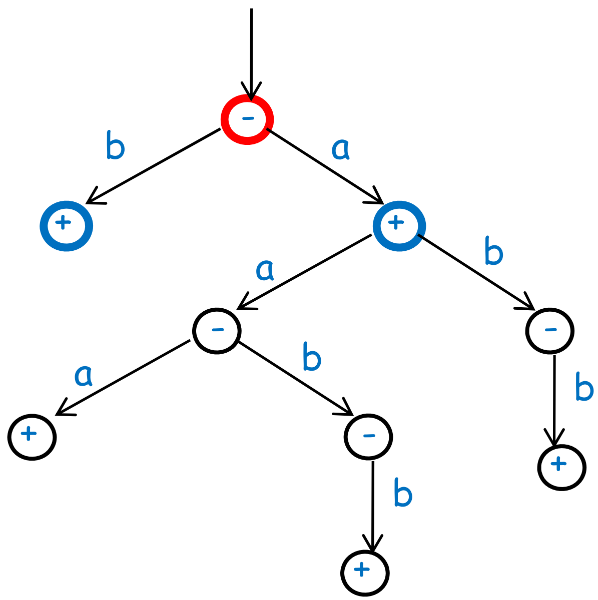
State Merging: Example



State Merging: Example



What if we change order?



About State Merging

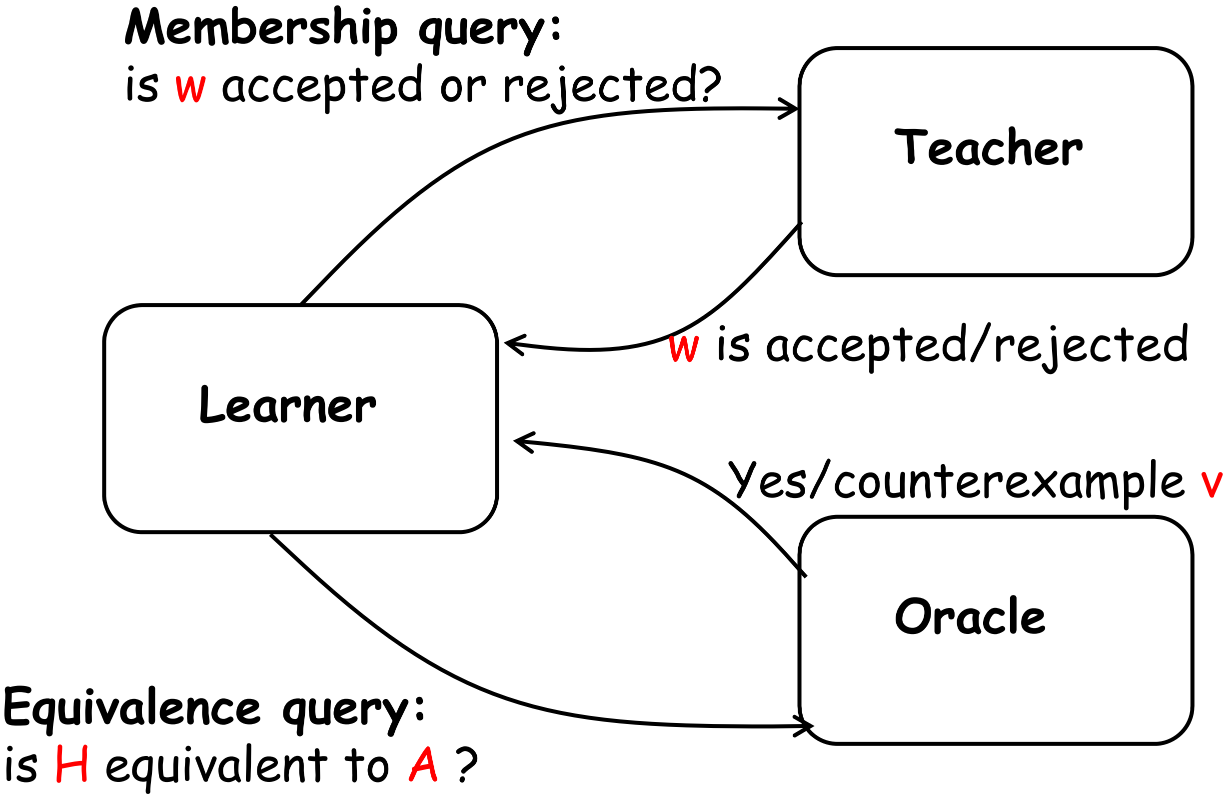
- Order in which blue states are considered matters.
- If considered states stay within $\{q_u \mid u \in K(L)\}$
  a minimal DFA will be constructed
- Otherwise, "suboptimal" merges may result
- Remedy: Teacher and Learner agree on a fixed technique to construct $Sp(L)$
  - e.g., to consider strings in lexicographic order
  - RPNI algorithm. [Oncina, Garcia]
- Otherwise: use heuristics for choosing "best merge",
  - e.g., to select states with "largest" subtrees.

About State Merging

- Time Complexity (in size of sample):
  - At most a quadratic number of candidate merges considered.
  - Each merge takes linear time to check
  - I.e., time complexity is polynomial.

Active Learning

Learner actively constructs the characteristic sample,



Ideas

- Maintain candidates for

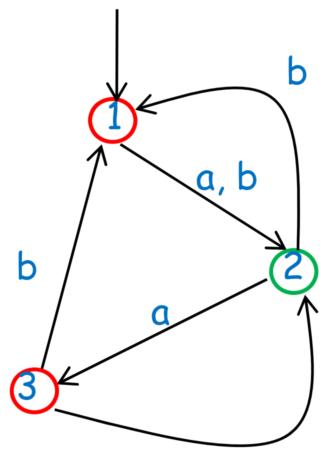
  $Sp(L) \qquad K(L) \qquad W$

  where W is a distinguishing set
- Ask membership queries for

  $$\{ uv \mid u \in Sp(L) \cup K(L) \quad v \in W \}$$

- If u in $K(L)$ is separated from all prefixes in $Sp(L)$ by separating suffix, move u to $Sp(L)$ and extend $K(L)$
- For new u' in $K(L)$ let W be large enough to separate u' from all but (at most) one prefix in $Sp(L)$

L* Algorithm



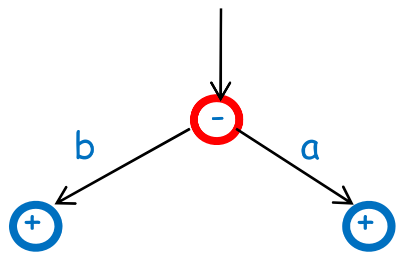
Observation table

W

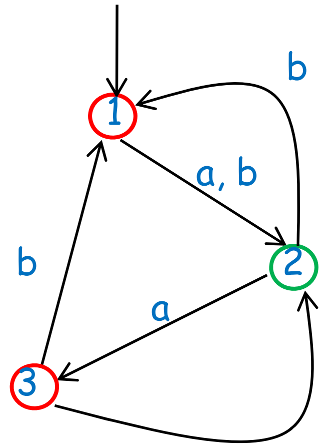
Sp(L)

K(L)

	λ	
λ	-	
a	+	
b	+	



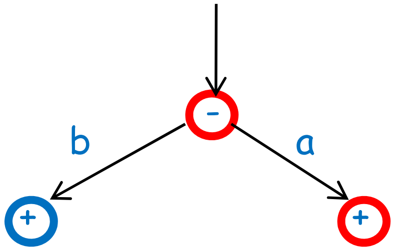
L* Algorithm



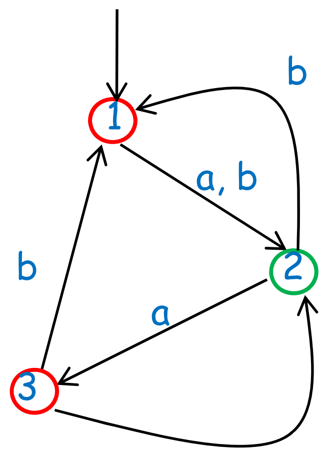
Observation table

W

		λ	
Sp(L)	λ	-	
	a	+	
K(L)			
	b	+	



L* Algorithm



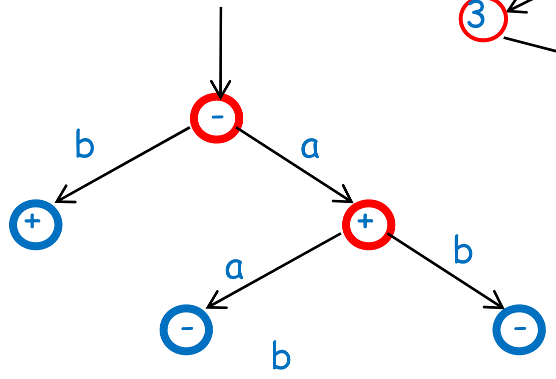
Observation table

W

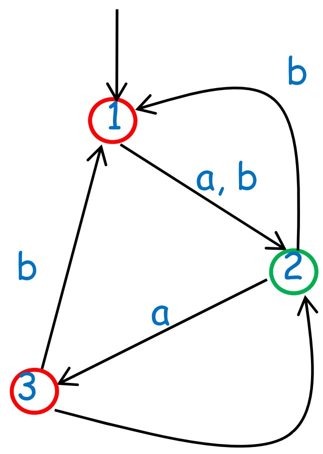
	λ	
λ	-	
a	+	
b	+	
aa	-	
ab	-	

Sp(L)

K(L)



Closed - Form Hypothesis



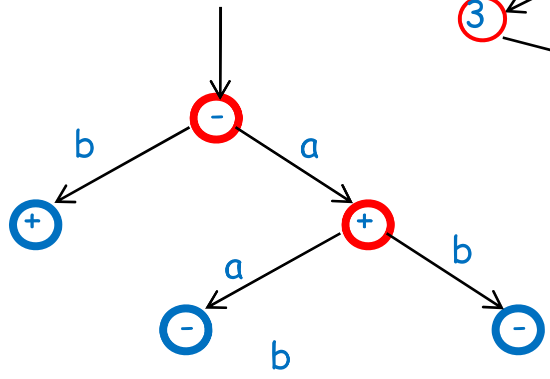
Observation table

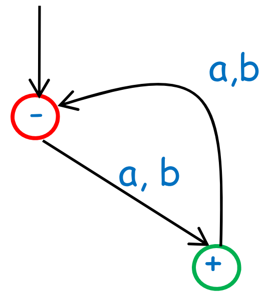
W

	λ	
λ	-	
a	+	
b	+	
aa	-	
ab	-	

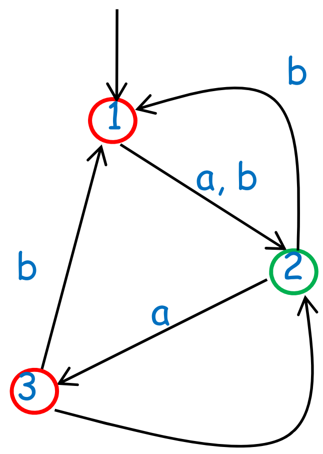
Sp(L): λ, a

K(L): b, aa, ab





Ask Equivalence Query



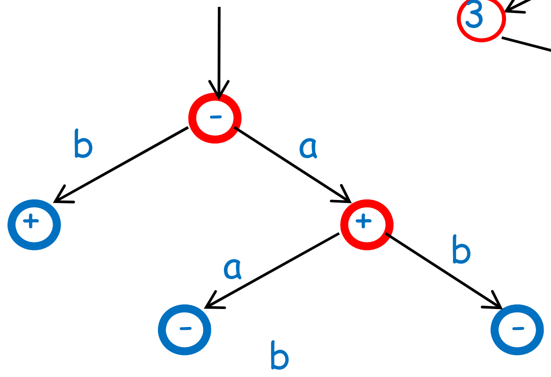
Observation table

W

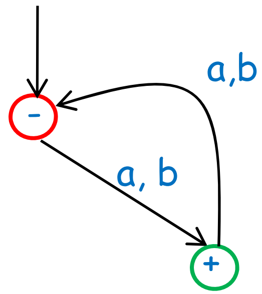
	λ	
λ	-	
a	+	
b	+	
aa	-	
ab	-	

Sp(L)

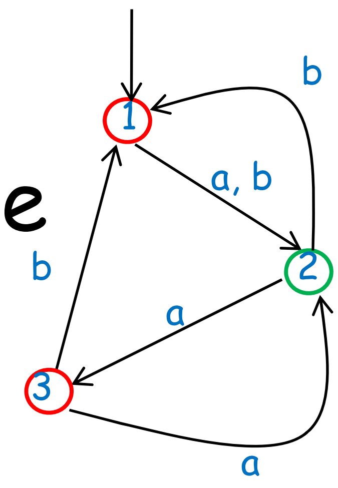
K(L)

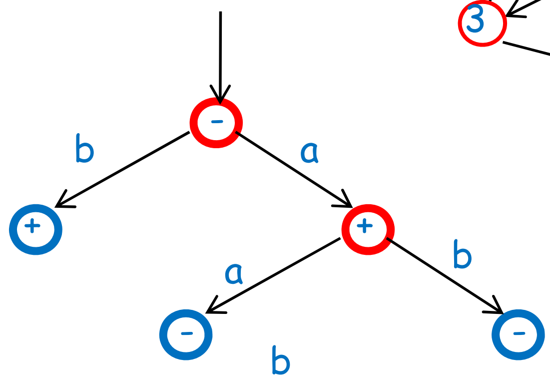


aab-



Decompose counterexample



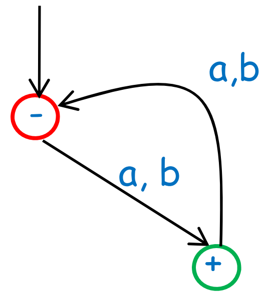


Observation table

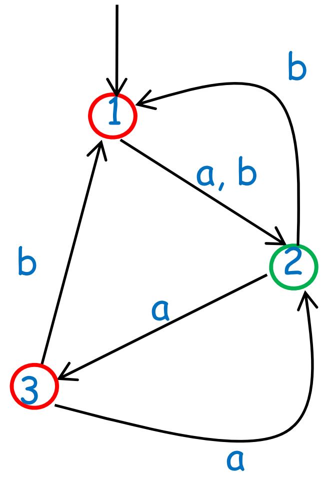
W

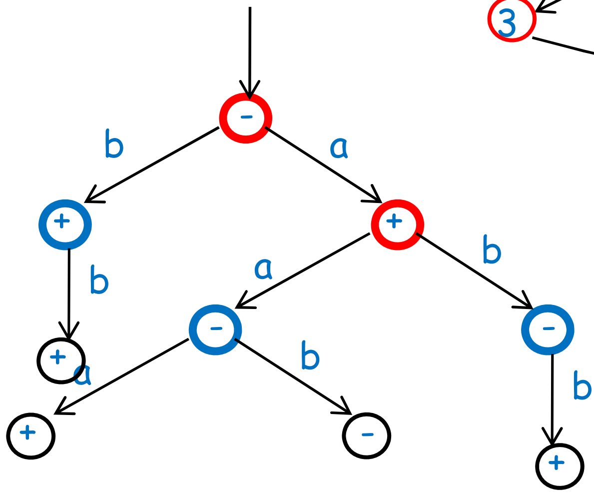
		λ	
Sp(L)	λ	-	
	a	+	
K(L)	b	+	
	aa	-	
	ab	-	

aab-



Add new suffix to W





Observation table

W

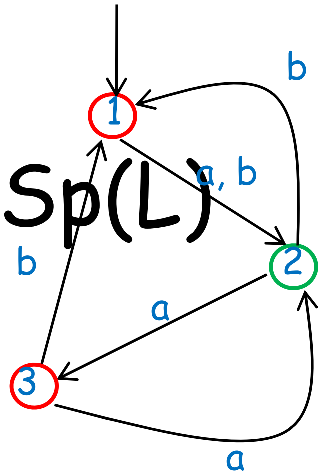
	λ	b
λ	-	+
a	+	-
b	+	-
aa	-	-
ab	-	-

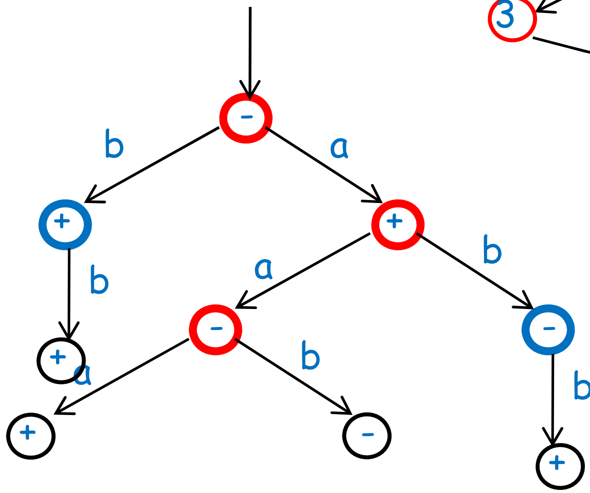
Sp(L)

K(L)

aab-

Not closed- Add new prefix to Sp(L)





Observation table

W

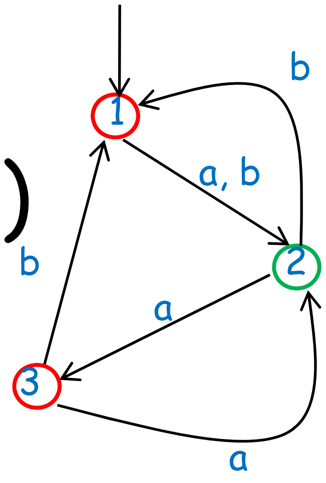
	λ	b
λ	-	+
a	+	-
aa	-	-
b	+	-
ab	-	+

Sp(L)

K(L)

aab-

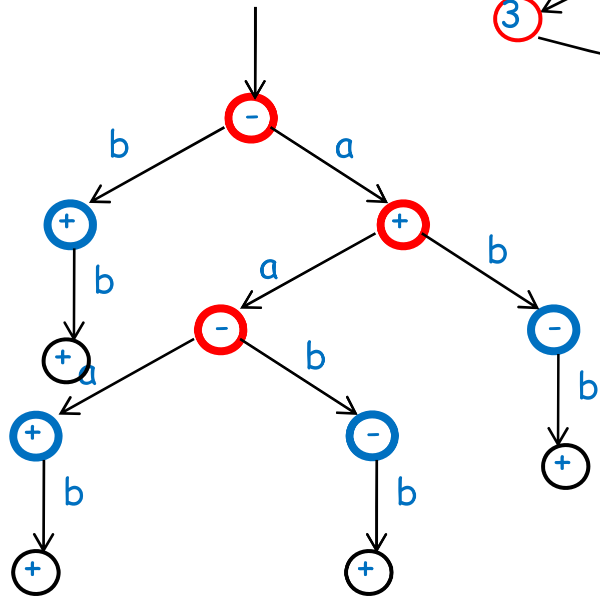
Add new extensions to K(L)



Observation table

W

		λ	b
Sp(L)	λ	-	+
	a	+	-
	aa	-	-
K(L)	b	+	-
	ab	-	+
	aaa	+	-
	aab	-	+



aab-

About L* [Angluin]

- DFA with n states can be learned using
  - $\leq n$ equivalence queries
  - $O(|\Sigma|n^2 + n \log m)$ membership queries
    - m is size of longest counterexample
- Produced hypothesis is always minimal DFA which is consistent with seen membership queries
  - These are a characteristic set for hypothesis
- Equivalence query idealizes (possibly) exponential search for deviations from model
- The setup with Membership and Equivalence queries makes it possible to formulate polymial-complexity algorithm.

Mealy Machines

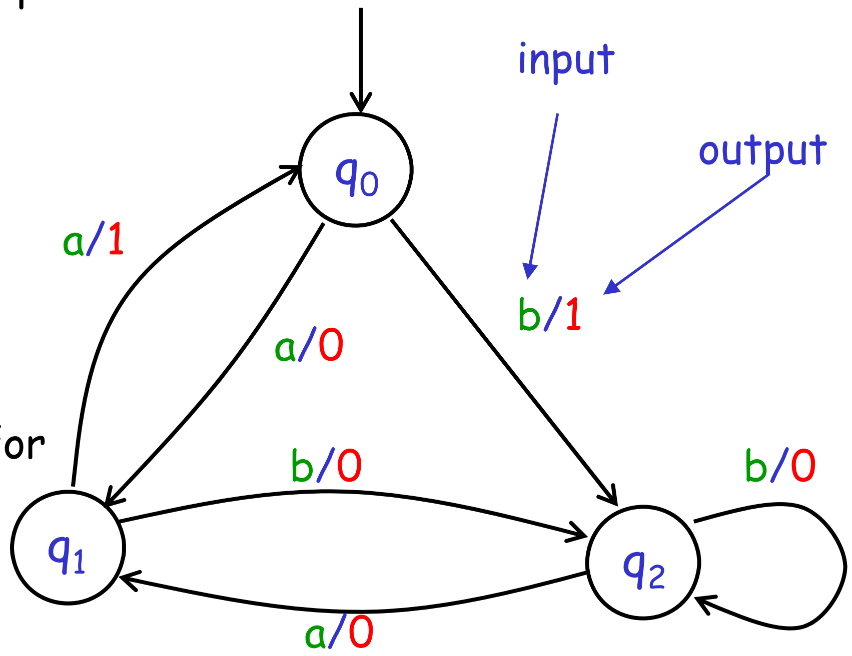
- Finite State Machines w. input & output

I	input symbols
O	output symbols
Q	states
δ: $Q \times I \rightarrow Q$	transition function
λ: $Q \times I \rightarrow O$	output function

- Often used for protocol modeling, for protocol testing techniques,

Assumptions:

- Deterministic
- Completely specified



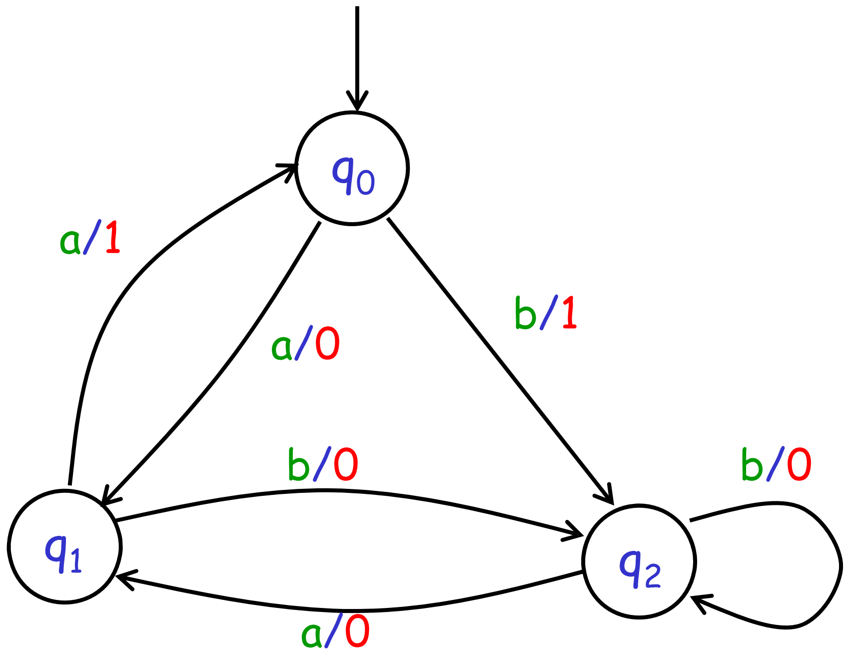
Conformance Testing

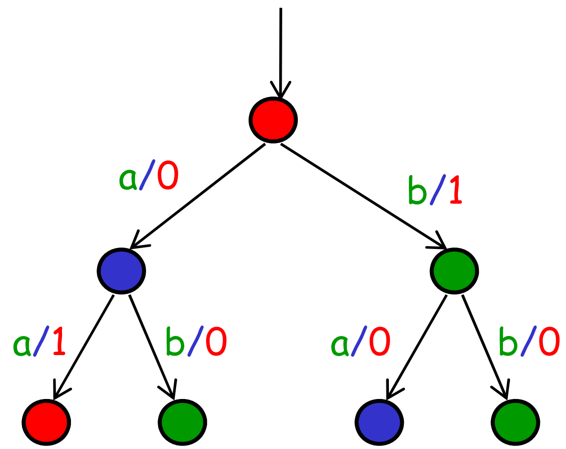
- Given MM A, construct a sample (i.e., a test suite) S such that A is "best fit" to explain S
  - Typically: A is the only MM with $\leq|A|$ states, which is consistent with S

W-method

Let Sp(L) be prefixes in minimal spanning tree of MM

Let K(L) be { ua | u ∈ Sp(L) a ∈ I }



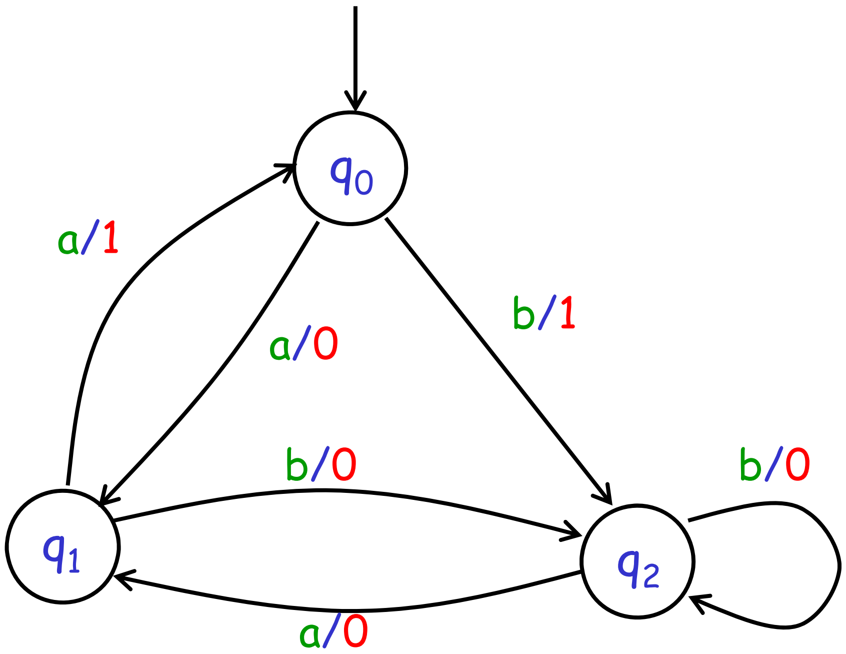


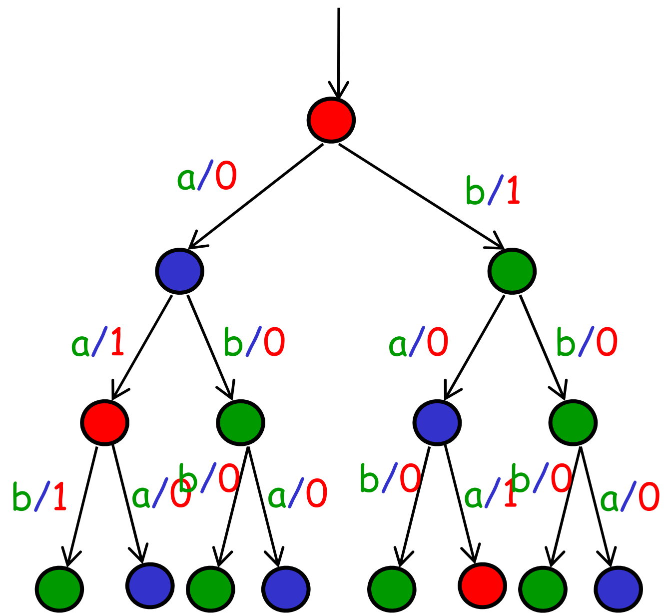
W-method

Let Sp(L) be prefixes in minimal spanning tree of MM

Let K(L) be $\{\ ua \mid u \in Sp(L) \quad a \in I\ \}$

Let Sample be $\{\ uv \mid u \in Sp(L) \cup K(L) \quad v \in W\ \}$
where W is a distinguishing set



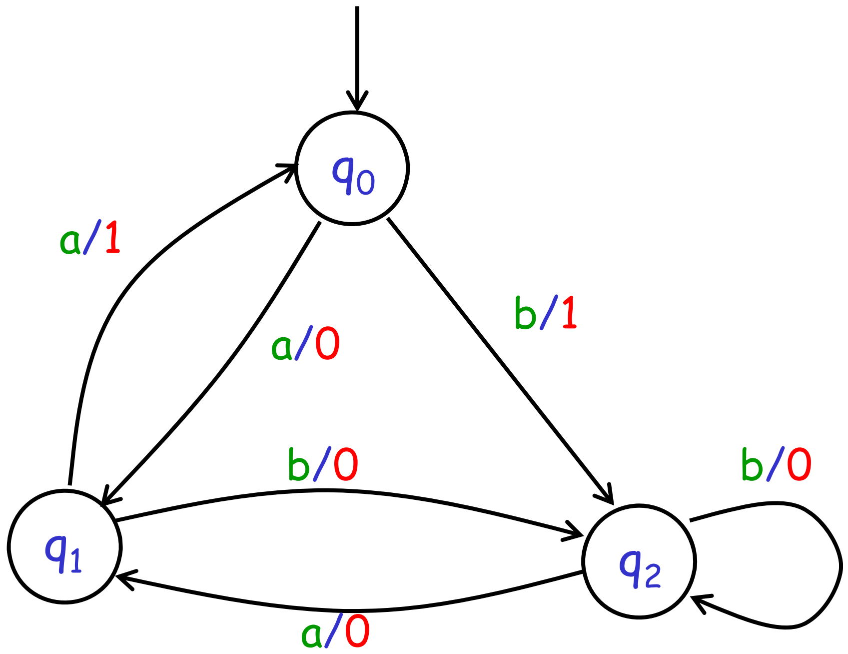


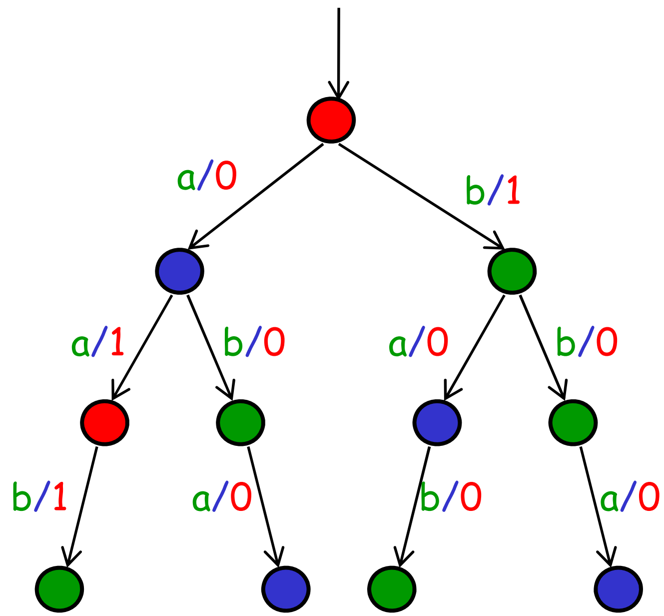
Z-method

Let $Sp(L)$ be prefixes in minimal spanning tree of MM

Let $K(L)$ be $\{ ua \mid u \in Sp(L) \quad a \in I \}$

Let Sample be $\{ uv \mid u \in Sp(L) \cup K(L) \quad v \in Z_{q_u} \}$
where $\{Z_q \mid q \in Sp(L) \}$ is a separating family of MM





Learning vs. Conformance Testing

- Learning: Find Concept *A* which is "best fit" to explain a given sample *S*
- Conformance Testing: Given Concept *A*, construct a sample *S* such that *A* is "best fit" to explain *S*
- For automata learning: A characteristic sample for *A* is also a conformance test suite for *A*

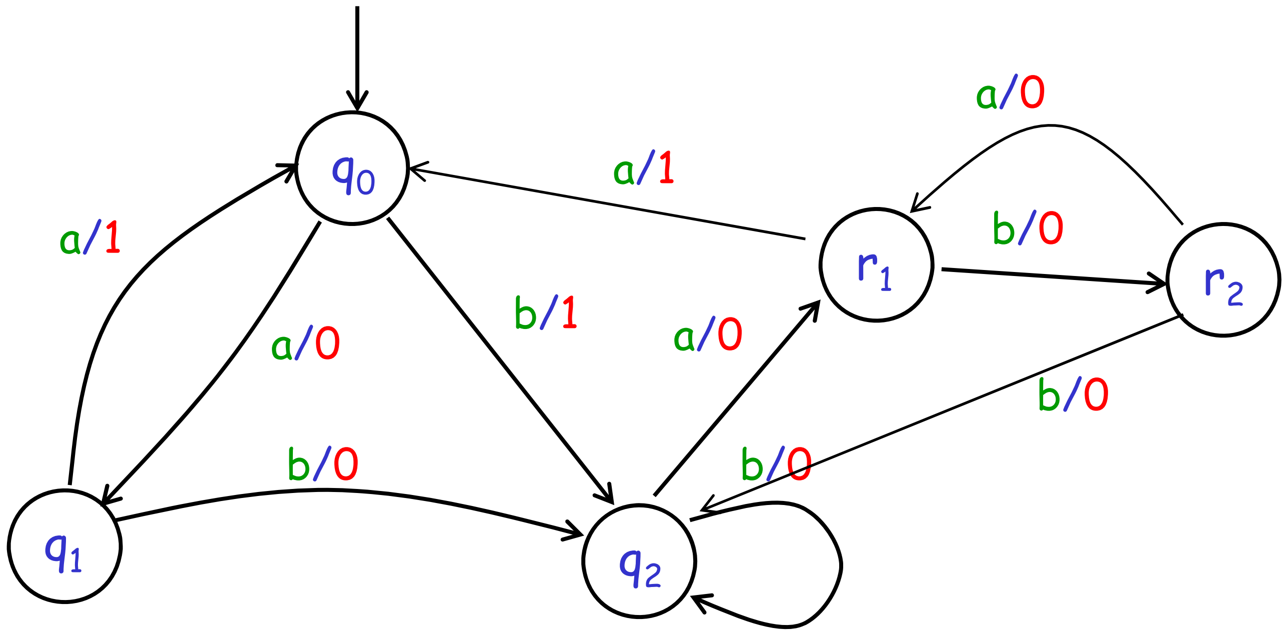
L* vs. W-method

- A sample generated by L* is also a conformance test suite generated by the W-method
- A conformance test suite generated by the W-method is a characteristic sample
- A is the only MM of size $\leq |A|$ which is consistent with S

Q: Can we check whether A is the only automaton of size $\leq |A| + k$ which is consistent with S

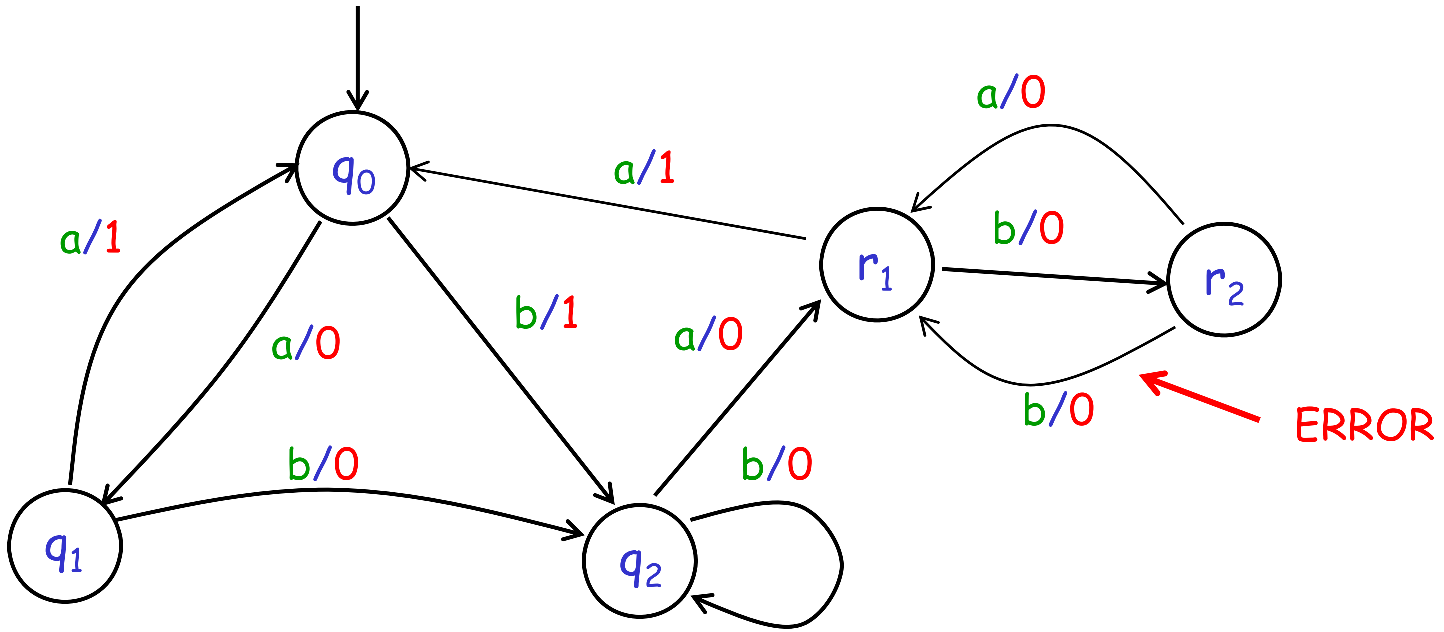
Vasilevski-Chow test suite

- Let k =2
- Test suite should allow non-minimised MM



Vasilevski-Chow test suite

- Let k =2
- Test suite should allow non-minimised MM
- Must cope with anomaly



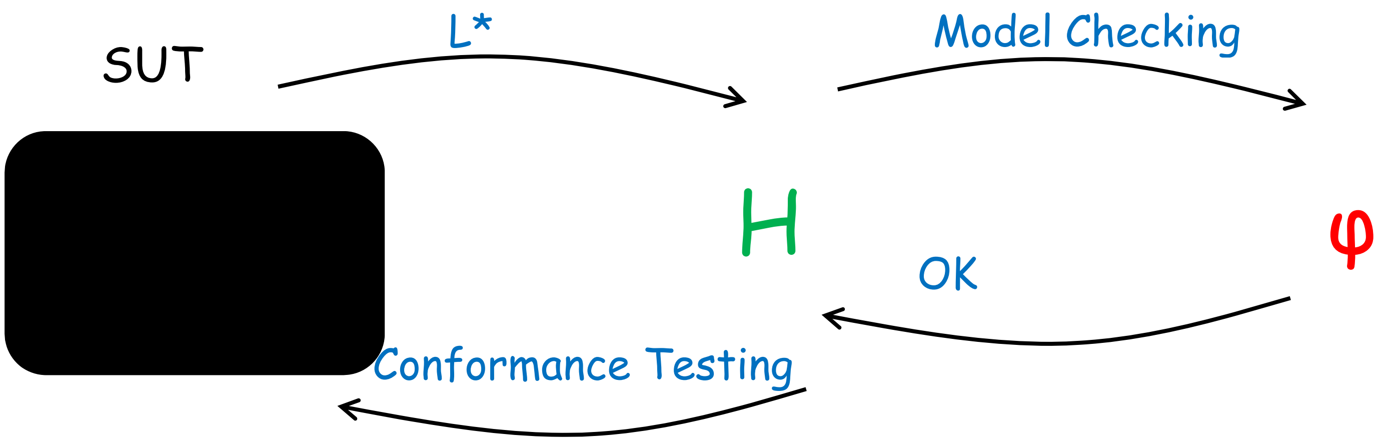
Resulting test suite

- Let W be a characterizing set for A
- VC-test suite has form

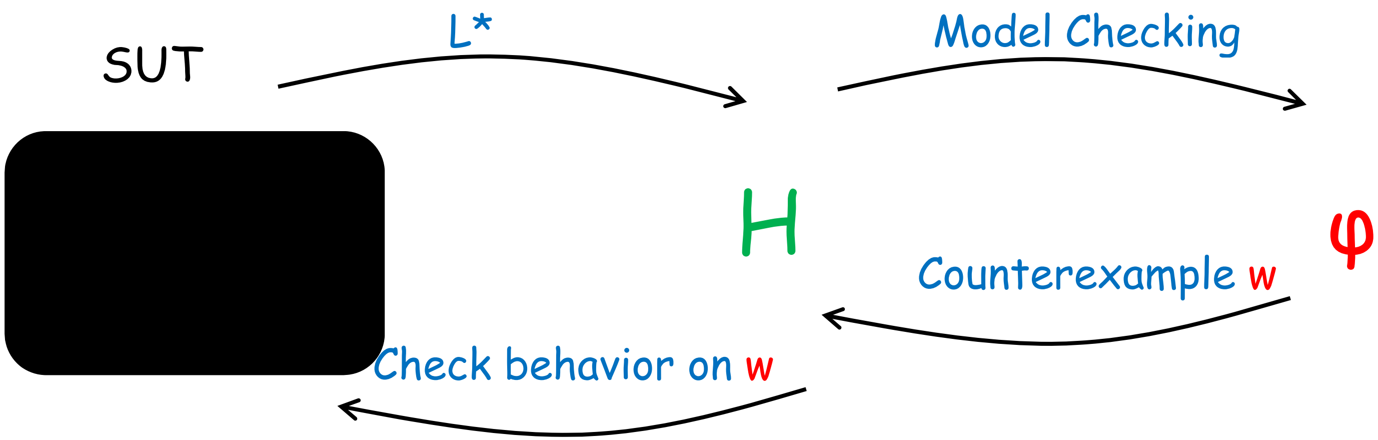
$$S = \{ uxv \mid u \in Sp(L) \cup K(L) \quad x \in I^{\leq k} \quad v \in W \}$$

- A is only MM of size $\leq |A| + k$ which is consistent with S
- Size of sample: $O(|\Sigma|^{k+1}\, n^2)$

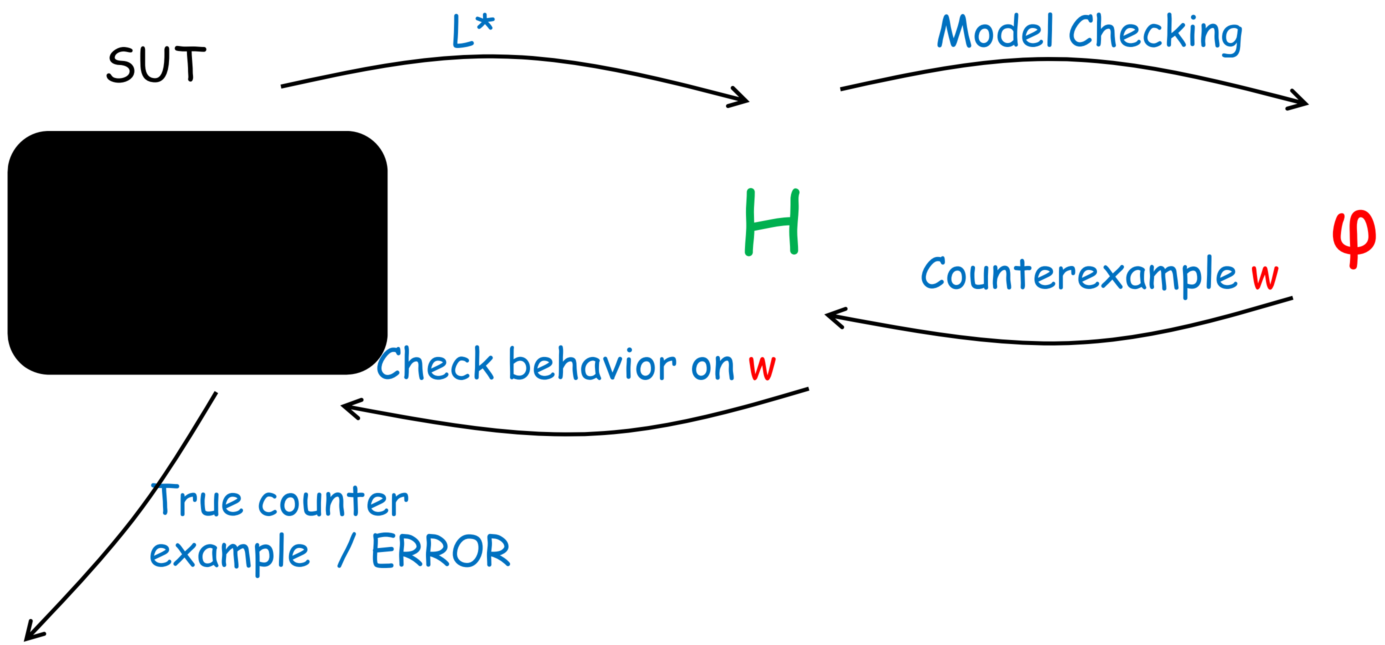
Adaptive Model Checking [Peled Yannakakis 02]



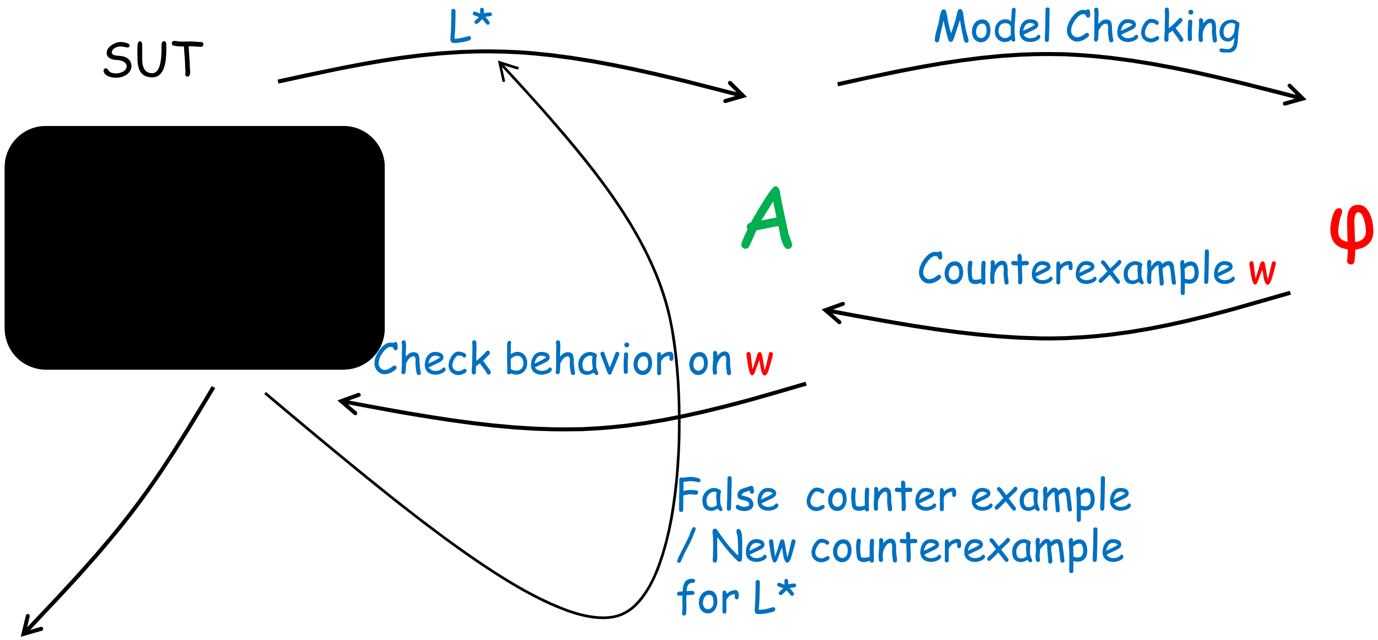
Adaptive Model Checking [Peled Yannakakis 02]



Adaptive Model Checking [Peled Yannakakis 02]



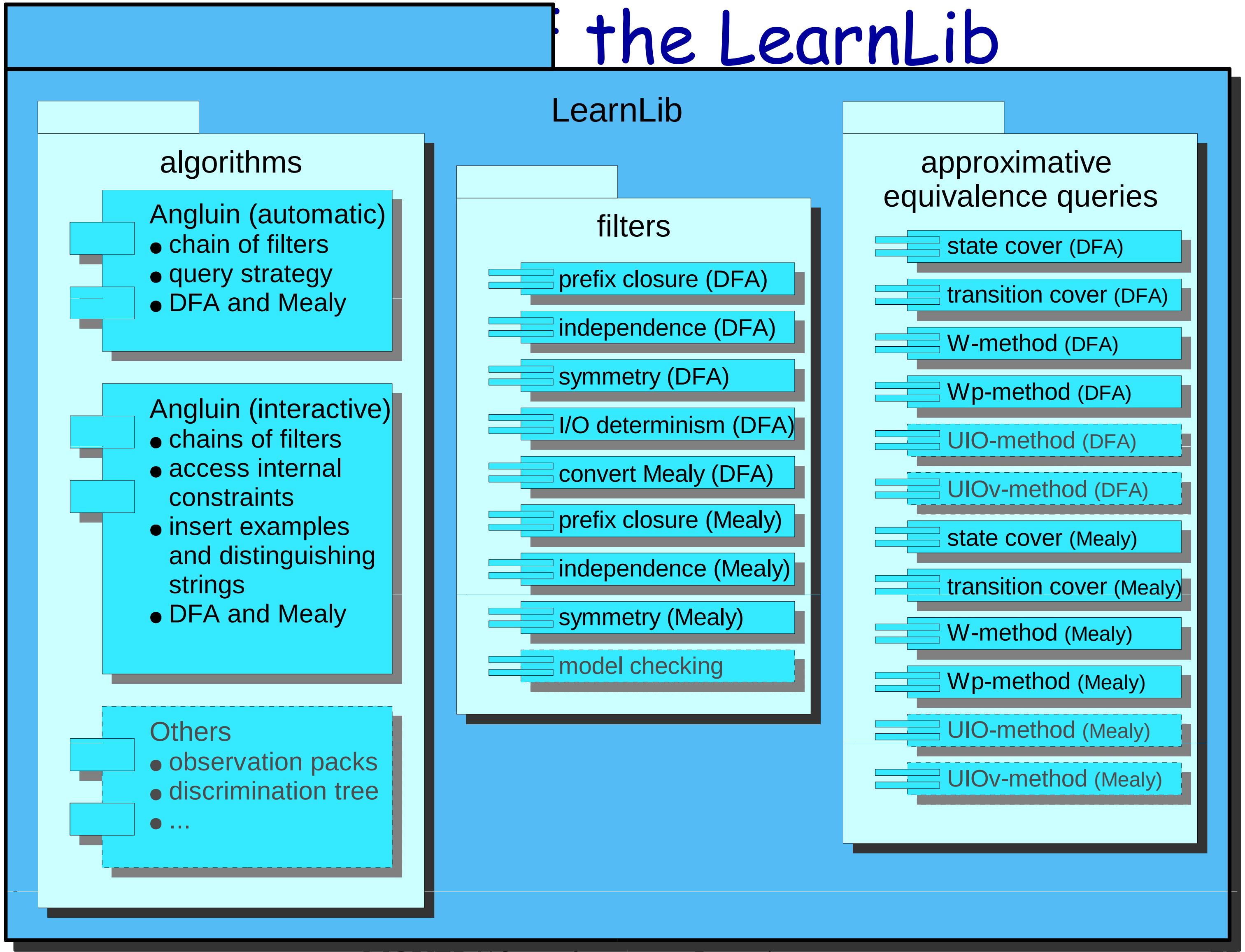
Adaptive Model Checking [Peled Yannakakis 02]



LearnLib: a Tool for Inferring Models

- Developed at Dortmund Univ. [Steffen, Raffelt, Howar, Merten]
- Central Idea: use domain-specific knowledge to reduce the number of queries:
  - Prefix-closure
  - Independence between symbols (e.g., in parallel components)
  - Symmetries
- These properties correspond to “filters” between observation table and SUT

the LearnLib



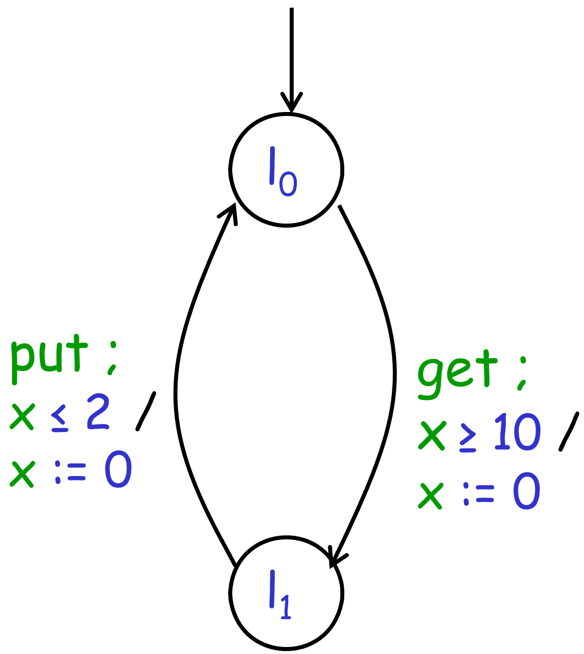
...

Whata about Extensions of Automata?

- Input and output symbols parameterized by data values.
- State variables remember parameters in received input
- Types of parameters could be, .e.,g
  - Identifiers of connections, sessions, users
  - Sequence numbers
  - Time values

Timed Automata

- Based on standard automata
- Clocks give upper and lower bounds on distance in time between occurrences of symbols.
- Temporal properties of Timed Automata (reachability, LTL, …) can be model-checked
- Implemented in tools (UPPAAL, IF/Kronos)



Timed words:

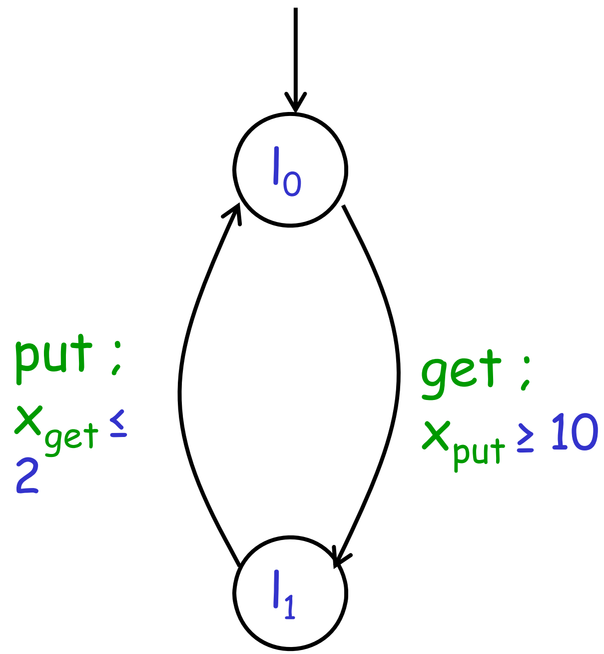
(get, 14.4) (put, 16.4) (get, 29.34) (put, 30.3) …

Event-Recording Automata

- Timed Automata can not be determinized in general
- Event-Recording Automata (ERA): One clock for each symbol, which is reset on that symbol.
- ERA can be determinized

Assumption:

Inference algorithm can precisely control and record timing of symbols.



Timed words:

(get, 14.4) (put, 16.4) (get, 29.34) (put, 30.3) ...

Clocked words:

(get, [14.4,14.4]) (put, [2.0,14.4]) (get, [14.94,12.94]) (get, [0.96,13.9]) ...

Event-Recording Automata

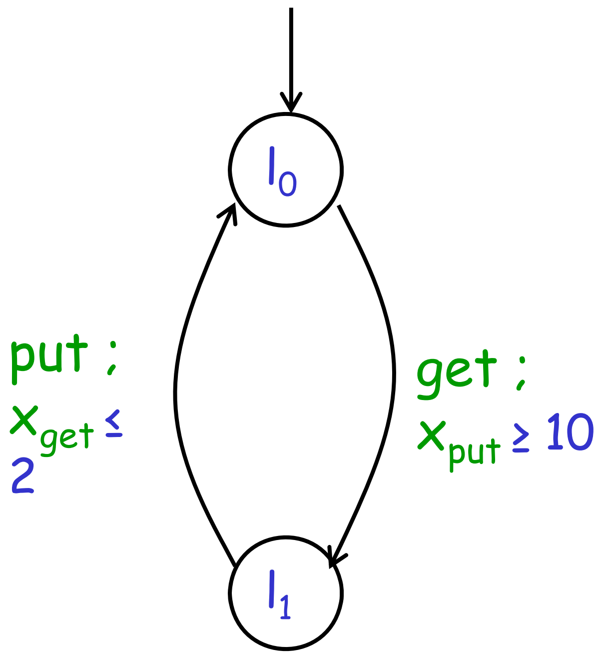
Σ (symbols) {put, get}

L (locations) $\{l_0, l_1\}$

l_0 (initial location)

E (edges) $\subseteq$ L x Σ x Guards x L

F (accepting locations) $\subseteq$ L



Event-Recording Automata

Σ (symbols) {put, get}

L (locations) $\{l_0, l_1\}$

l_0 (initial location)

E (edges) $\subseteq$ $L \times \Sigma \times \text{Guards} \times L$

Conjunctions of interval constraints

F (accepting locations) $\subseteq L$

Semantics

Q (states) $L \times R^{\geq 0} \times R^{\geq 0}$

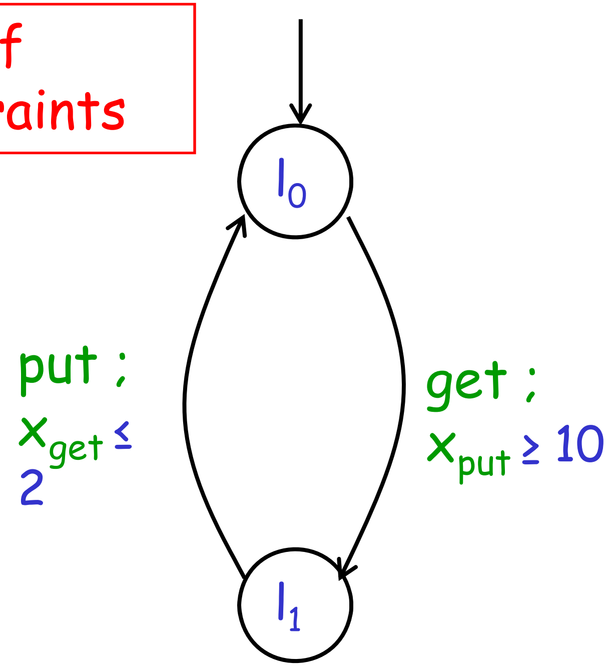
q_0 (initial state) $(l_0, [0,0])$

I $\Sigma \times R^{\geq 0} \times R^{\geq 0}$

δ: $Q \times I \rightarrow Q$

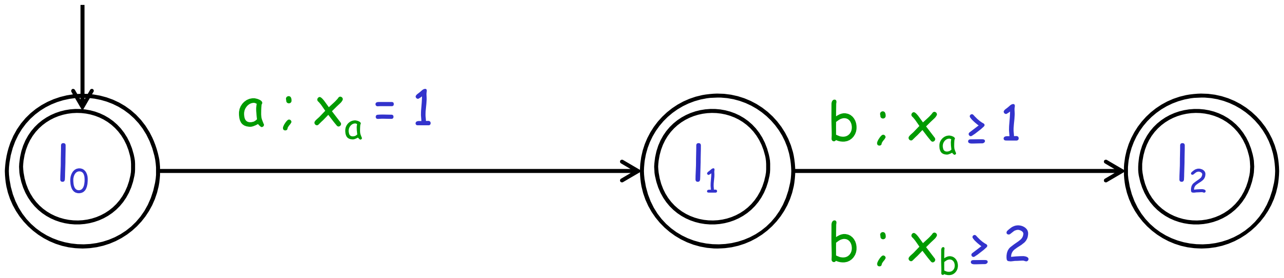
$\delta(\langle l_0, [0,0] \rangle, \langle \text{get}, [14.4, 14.4] \rangle) = \langle l_1, [0, 14.4] \rangle$

$\delta(\langle l_1, [0, 14.4] \rangle, \langle \text{put}, [2.0, 14.4] \rangle) = \langle l_0, [2.0, 0] \rangle$



Non-Unique Representation

- Deterministic ERAs do not have unique representations



Learning DERAs by Quotienting [Grinchtein , Leucker, al.]

- **Find equivalence relation $\approx$ on symbols and states, s.t.**
  - **$\approx$ respects accepting/non-accepting states**
  - $q \approx q'$ $a \approx a'$ **implies** $\delta(q,a) \approx \delta(q',a')$
- **Learn the Quotient DFA**

  $\Sigma / \approx$ $\quad Q / \approx$ $\quad \delta^{\approx}$ ($\delta([q]_{\approx},[a]_{\approx}) = [\delta(q,a)]_{\approx}$) $\quad F / \approx$

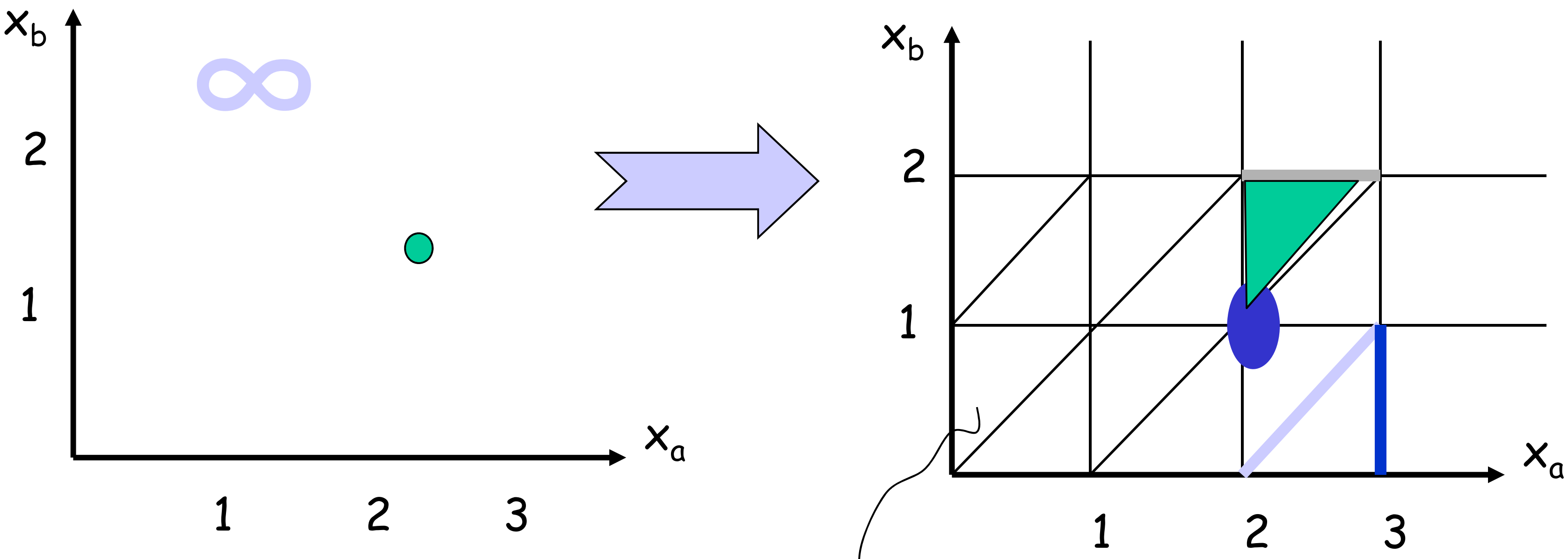
For DERAs

- **Equivalence on states based on region equivalence**
- **Assume largest constant K_a in constraints on x_a**
- $\langle l , [x_a, x_b]\rangle \approx \langle l , [y_a, y_b]\rangle$ **iff**
  - $x_a > K_a$ **and** $y_a > K_a$ **or**
    integer parts of x_a **and** y_a **same** **and** x_a **is integer iff** y_a **is integer**
  - **same for** x_b **and** y_b
  - **If** $x_a \leq K_a$ **and** $x_b \leq K_b$ **then** $x_a \leq x_b$ **iff** $y_a \leq y_b$
- $\langle a , [x_a, x_b]\rangle \approx \langle a , [y_a, y_b]\rangle$ **iff for all** $k \leq K_a$
  - $x_a \leq k$ **iff** $y_a \leq k$ **and** $x_a \geq k$ **iff** $y_a \geq k$

Regions: From infinite to finite

Concrete State
(l, [2.2, 1.5])

Symbolic state (region)
(l, ◥)



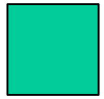
An equivalence class (i.e. a *region*)
... There are only *finite* many such!!

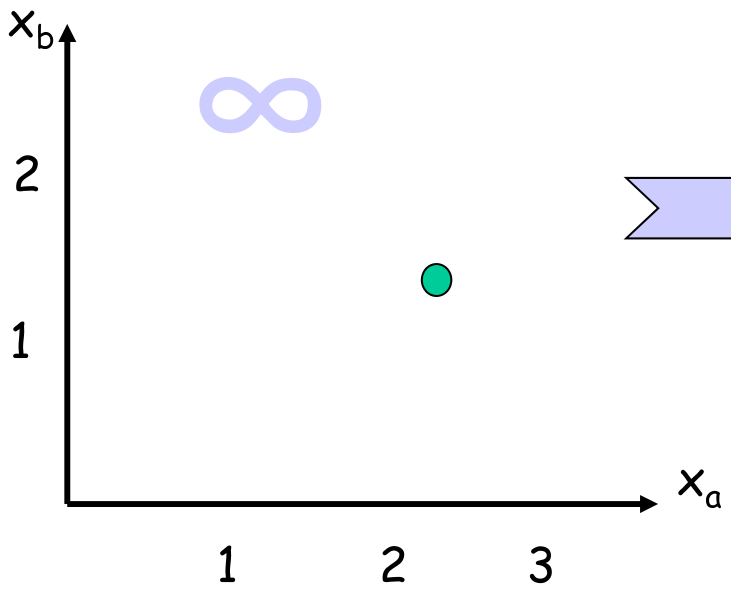
Abstraction of symbols

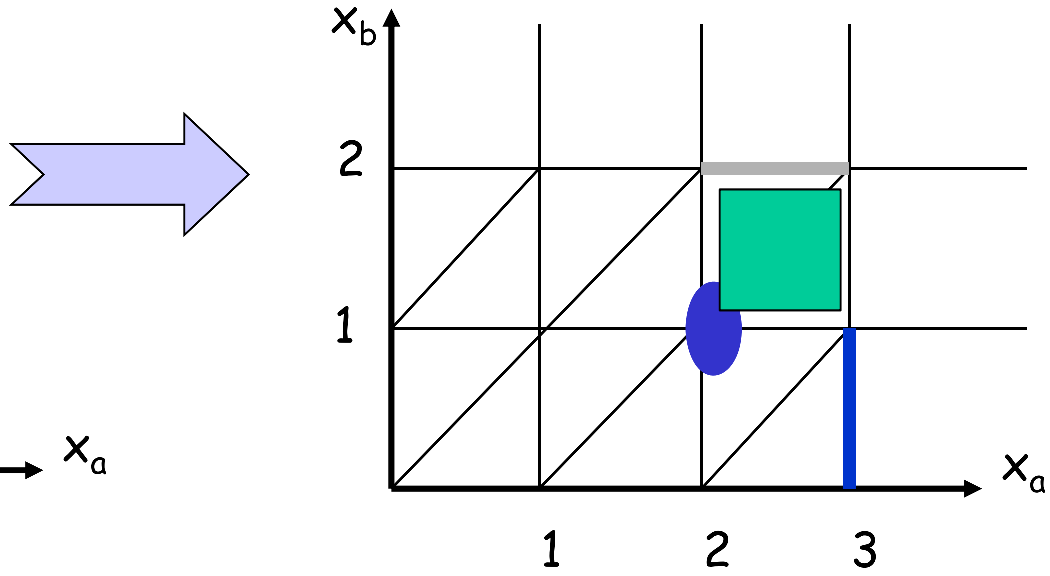
Concrete Symbol

(a, [2.2, 1.5])

Abstract symbol

(a, )

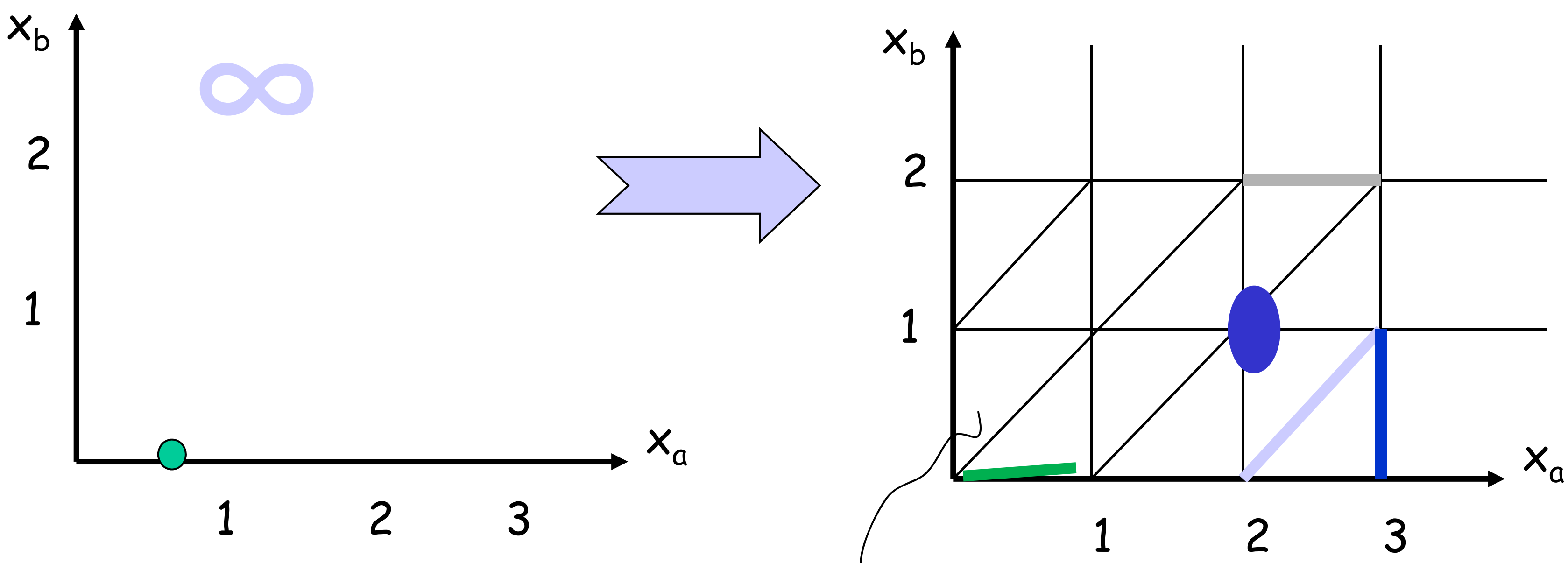




We need only initial regions

Concrete State
(l, [0.7, 0])

Symbolic state (region)
(l,)

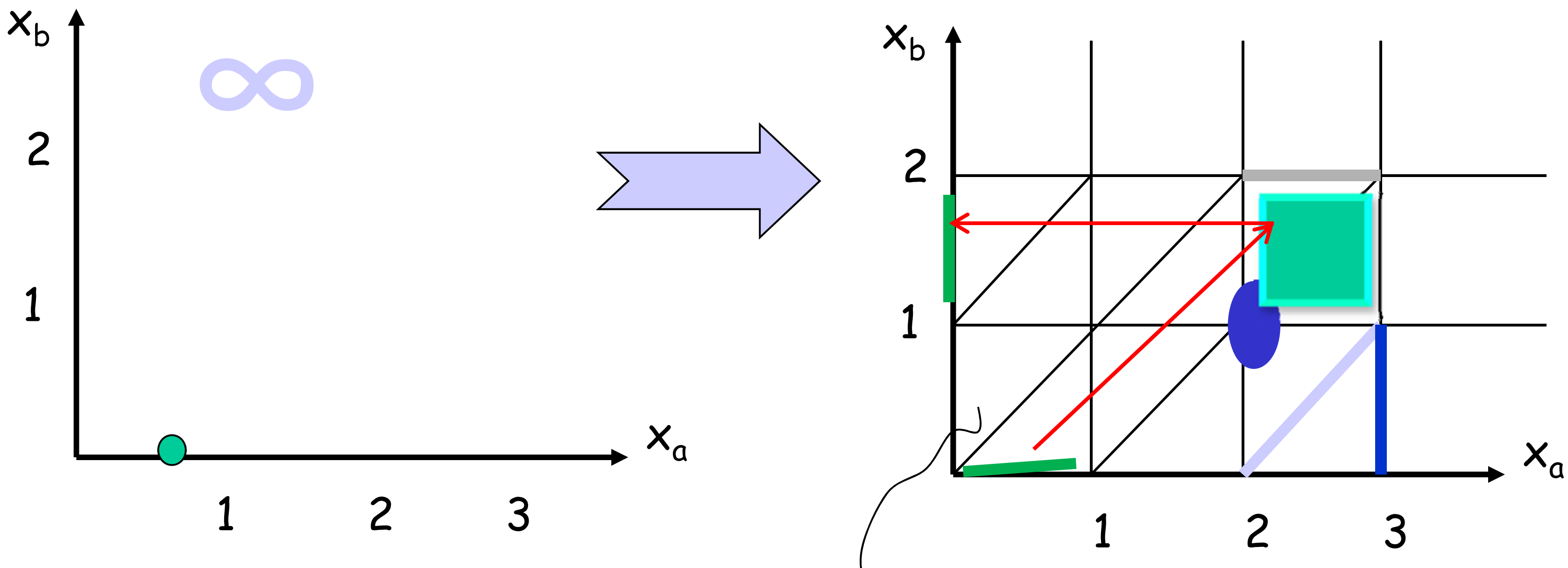


An equivalence class (i.e. a *region*)
… There are only *finite* many such!!

Regions preserved by transitions

Concrete State
(l, [0.7, 0])

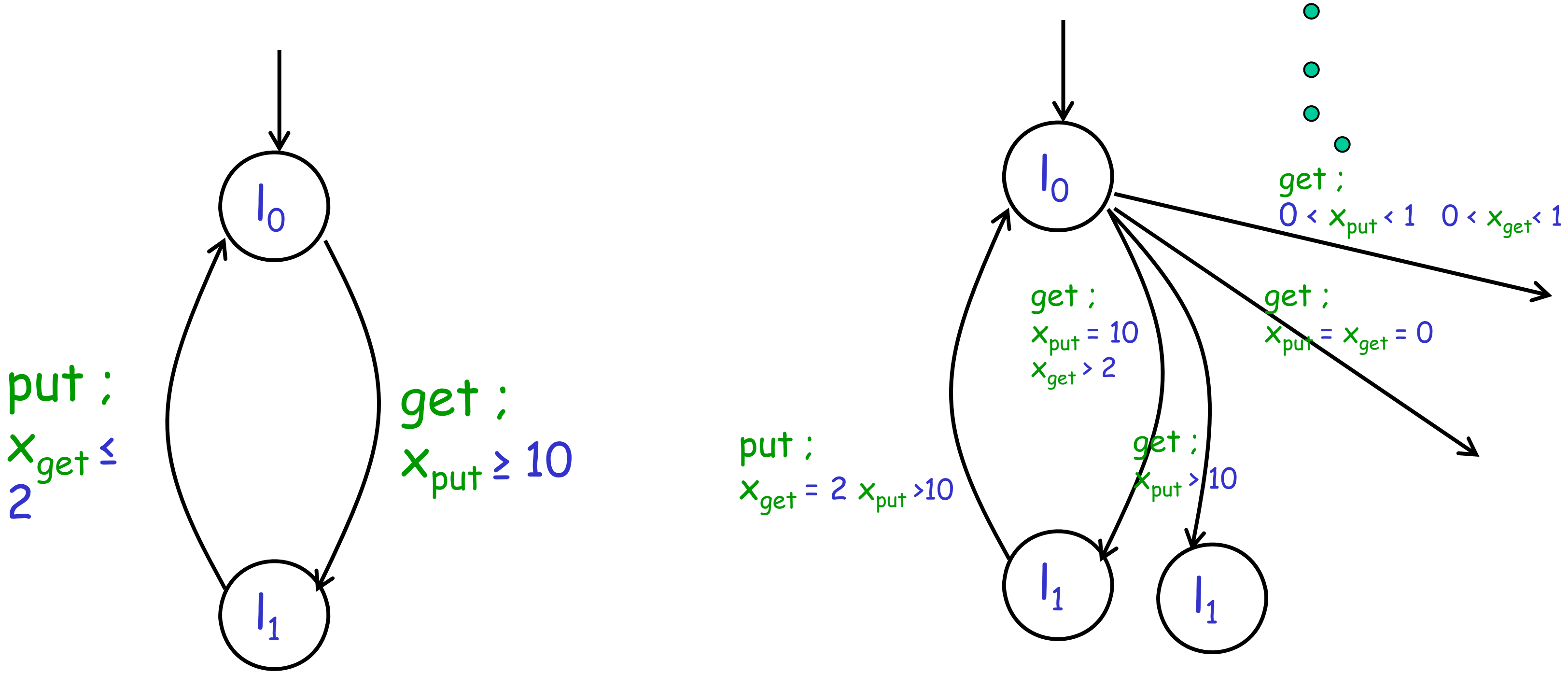
Symbolic state (region)
(l,)



An equivalence class (i.e. a *region*)
… There are only *finite* many such!!

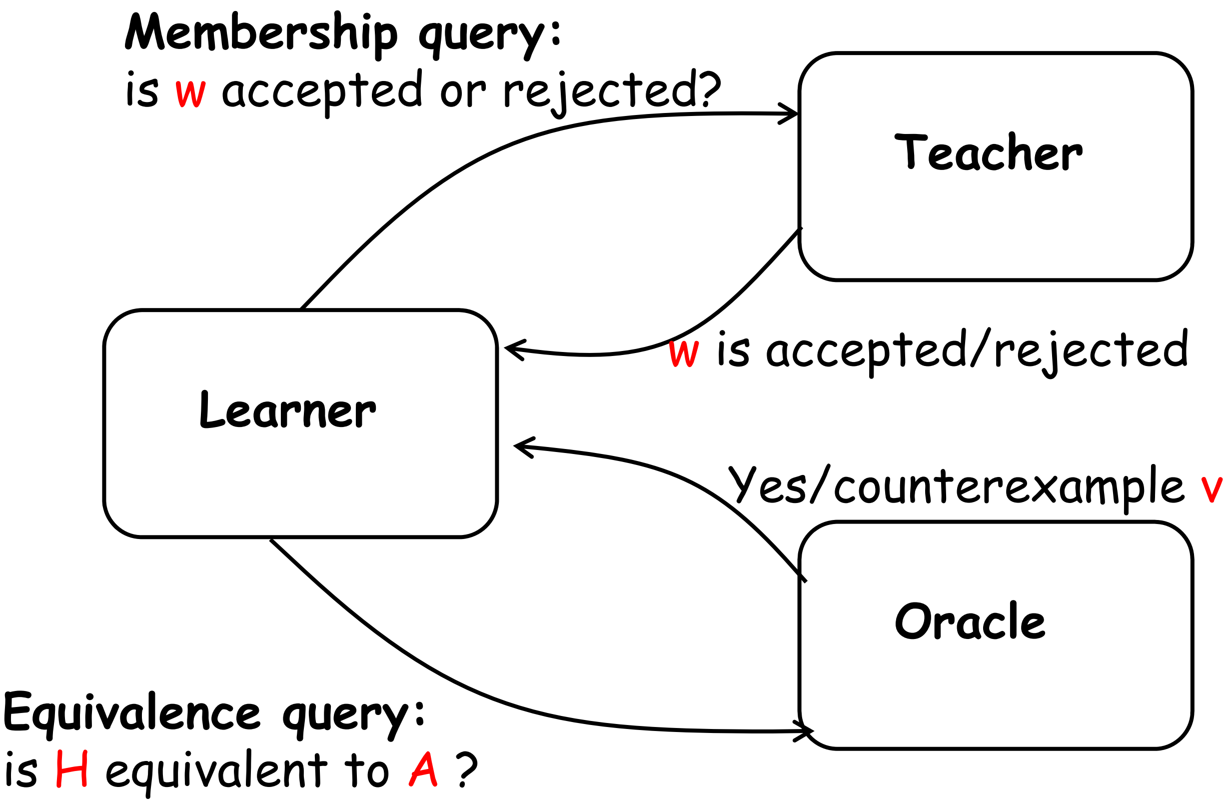
Simple DERAs

- DERA with "small guards"



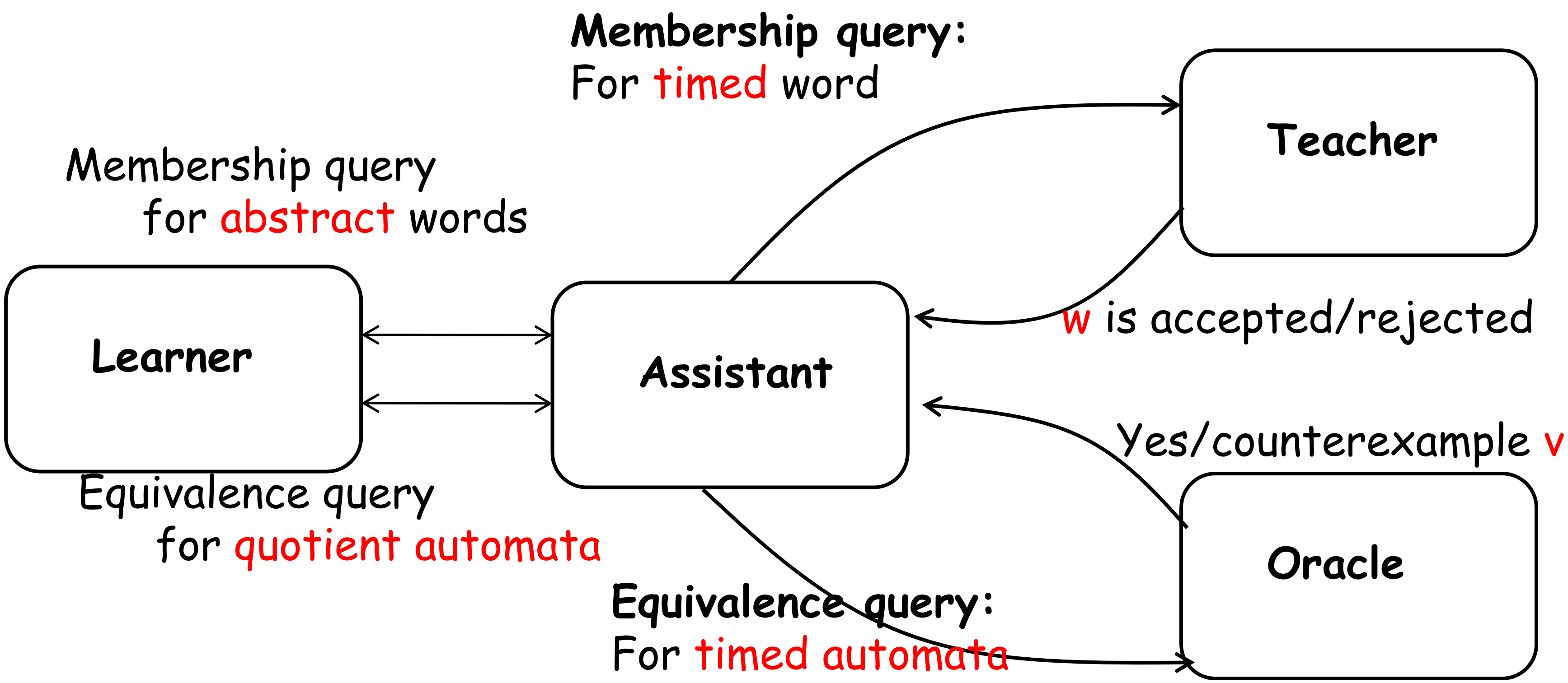
Modifying Setup

The following setup does not work



Adding Assistant

Learner actively constructs the characteristic sample,



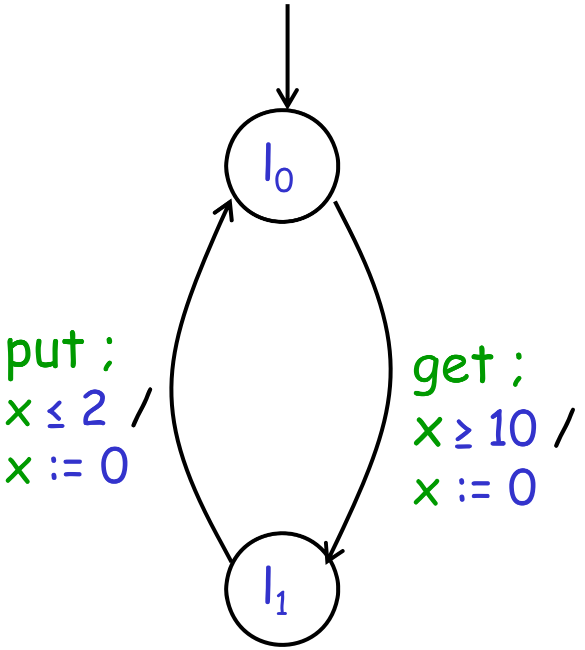
Query Complexity

- **Size of Region graph is roughly**
  $O(|L| \, K^{|\Sigma|})$
- **Number of Membership Queries is about cubic in this number**

Single-Clock Automata [Verwer et al. 09]

Consider Deterministic Timed Automata with one clock

- **Still, no unique minimal representation**
- **But, there is a variant of Nerode Congruence**
  - ***if we know where resets occur***



Timed word:

(get, 14.4) (put, 16.4) (get, 29.34) (put, 30.3) …

Clocked word:

(get, 14.4) (put, 2.0) (get, 12.96)

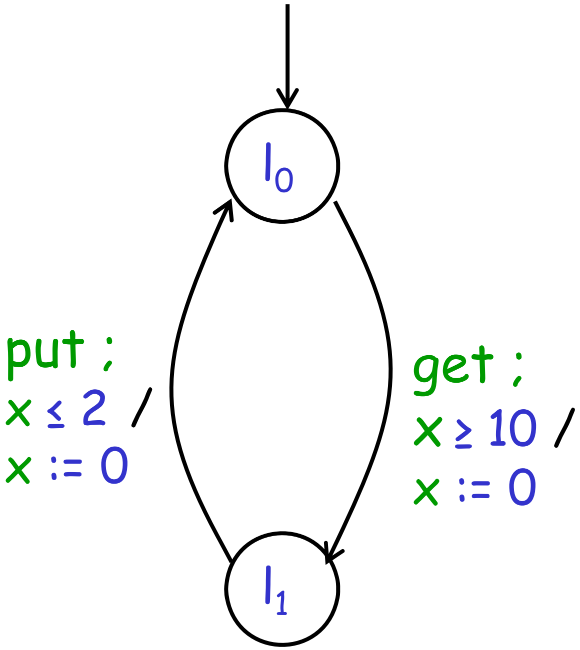
(get, 14.4) reset (put, 2.0) reset (get, 12.96) reset
Is equivalent to
(get, 12.4) reset
but not to
(get, 12.4)

Single-Clock Automata [Verwer et al. 09]

The timed language can be formed from a finite number of Congruence classes

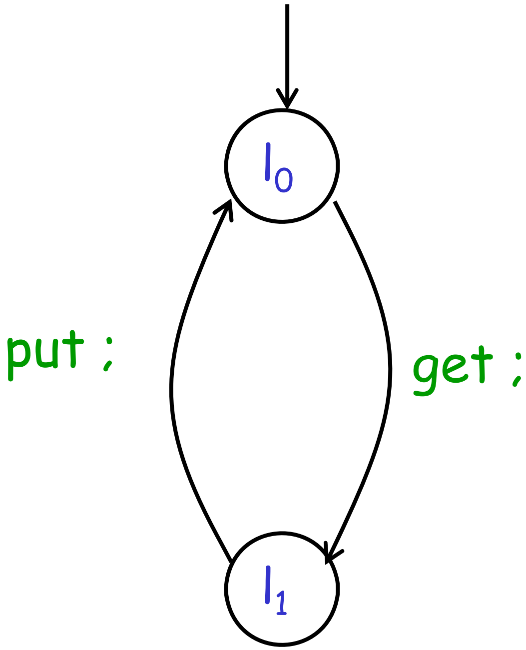
Only, it must be determined when to reset?

Define canonical form by prioritizing conflicts



Refining Guards [Verwer et al. 09]

- **Guards can be refined by counterexamples**



Guards refined from counterexamples

- get @0 put @2 accepted
- get @3 put @7 rejected

Determine the reason for difference by investigating other traces

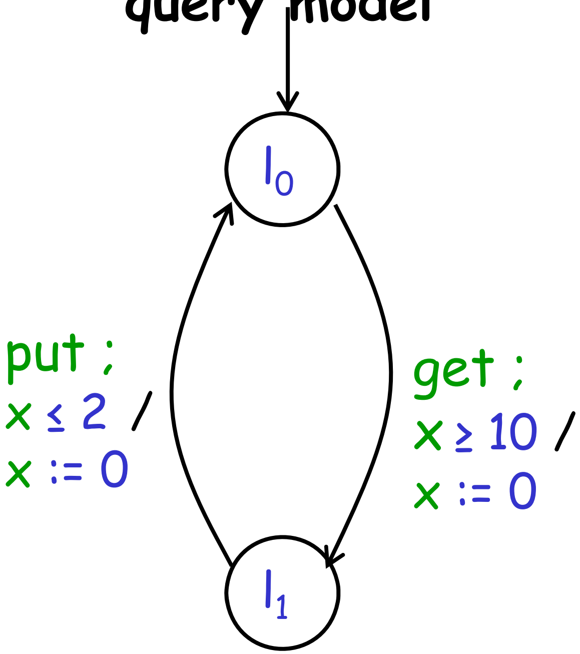
- (binary) search procedure
- Finds "explaining pair", e.g.,
  - get @2.2 put @4.2 accepted
  - get @2.2 put @4.7 rejected
- Suggests reset at get and guard $x \leq 2$ on put transition

Single-Clock Automata [Verwer et al. 09]

Have "reasonable" canonical forms

Exist characteristic samples which are polynomial in size of canonical form (does not depend on largest constant)

Learning can be polynomial in (Membership,Equivalence)-query model



Version for multiple clocks [Grinchtein,Jonsson]

Higher complexity

...

Applications to Realistic Procotols

SIP Protocol [Aarts,Jonsson, Uijen]

From RFC 3261:

- SIP is an application-layer control protocol that can
  - establish, modify, and terminate multimedia sessions (conferences) such as Internet telephony calls.
  - invite participants to already existing sessions, such as multicast conferences.

Structure of SIP packets

Method(From;To; Contact; CallId; CSeq; Via), where

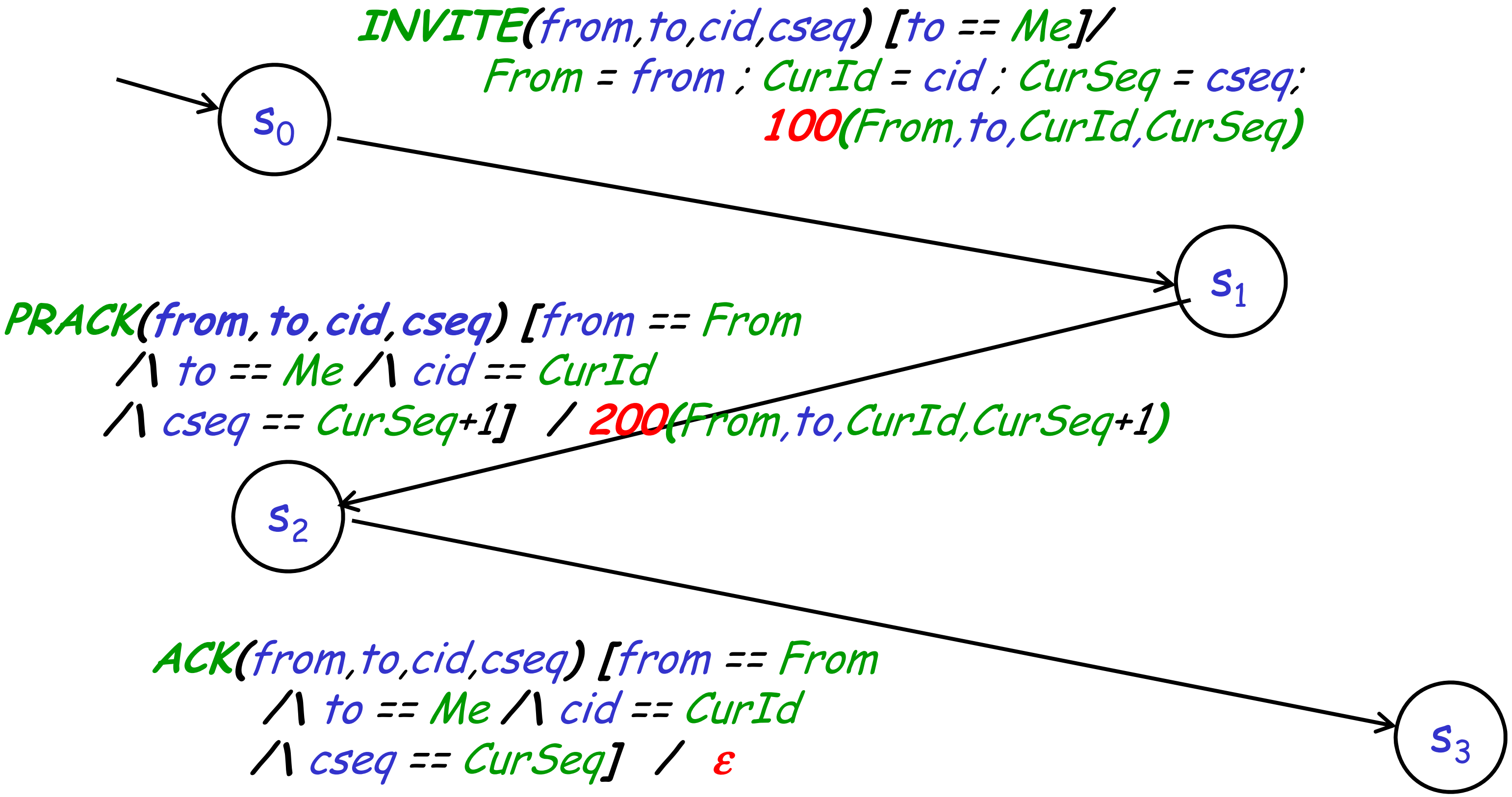
- Method: type of request, either INVITE, PRACK, or ACK.
- From and To: addresses of the originator and receiver
- CallId: unique session identier.
- Cseq: sequence number that orders transactions in a session.

IGNORE THE BELOW

- Contact: address where the Client wants to receive input
- Via: transport path for the transaction.

part of SIP Server

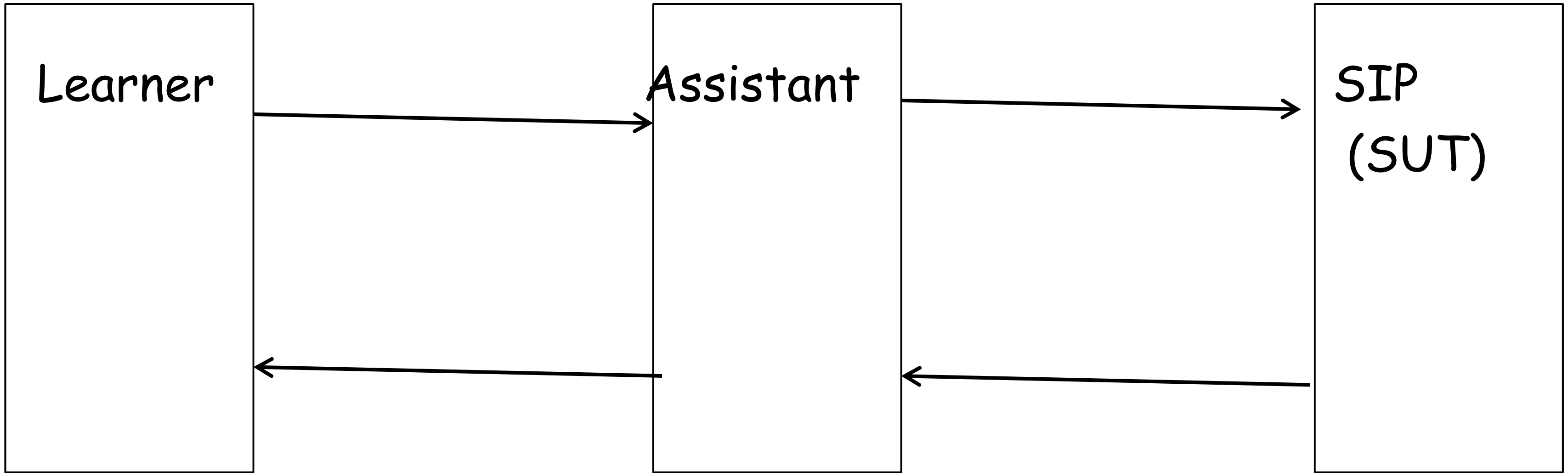
Variables: *From, CurId, CurSeq*
Constants: *Me*



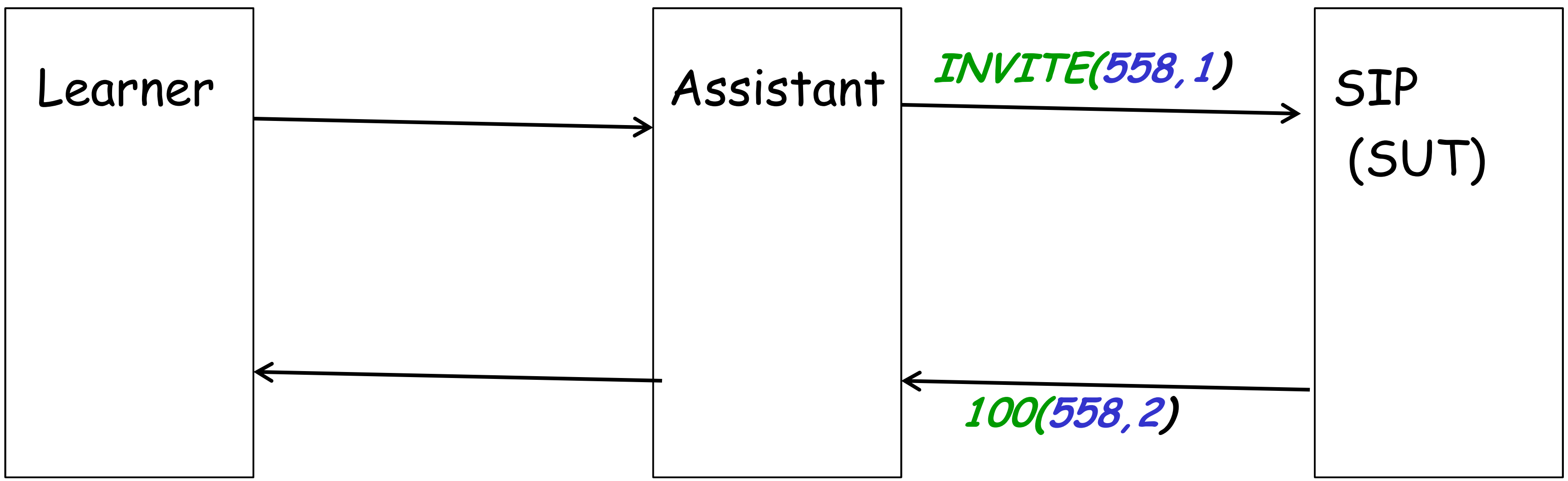
Finding an Abstraction

- **Abstraction of Concrete Message** *PRACK(558,1)*
  depends on internal state of SUT
  previous history
- **Assistant must maintain relevant parts of history:**
  e.g., local copies of *CurId, CurSeq*

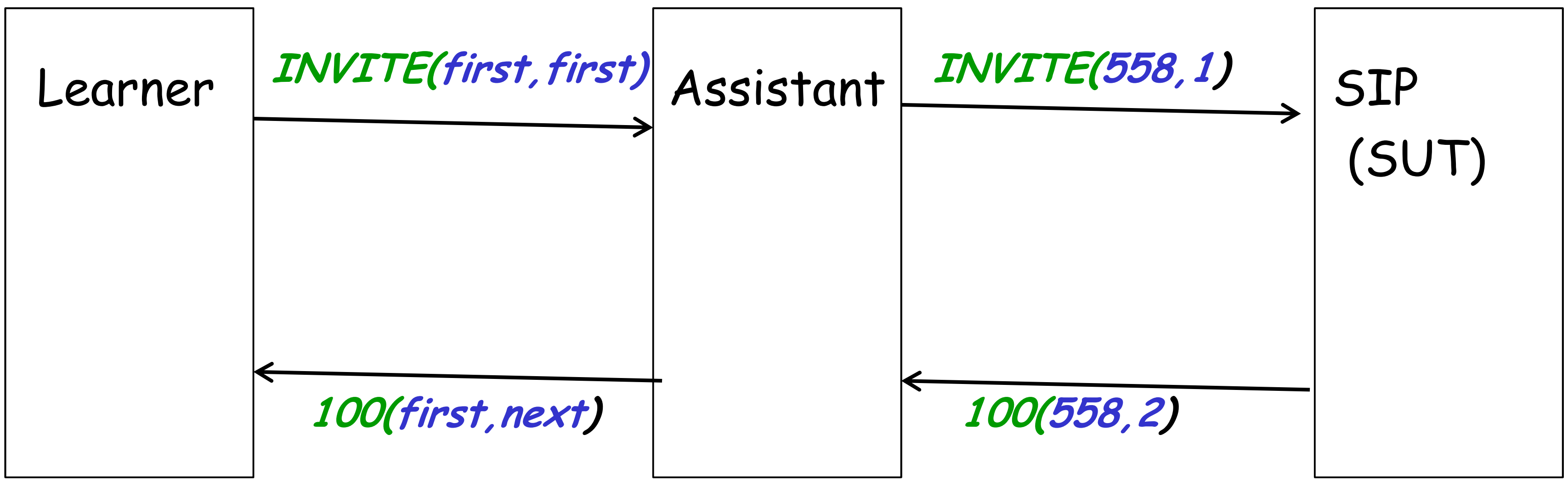
Adapting to Automata Learning



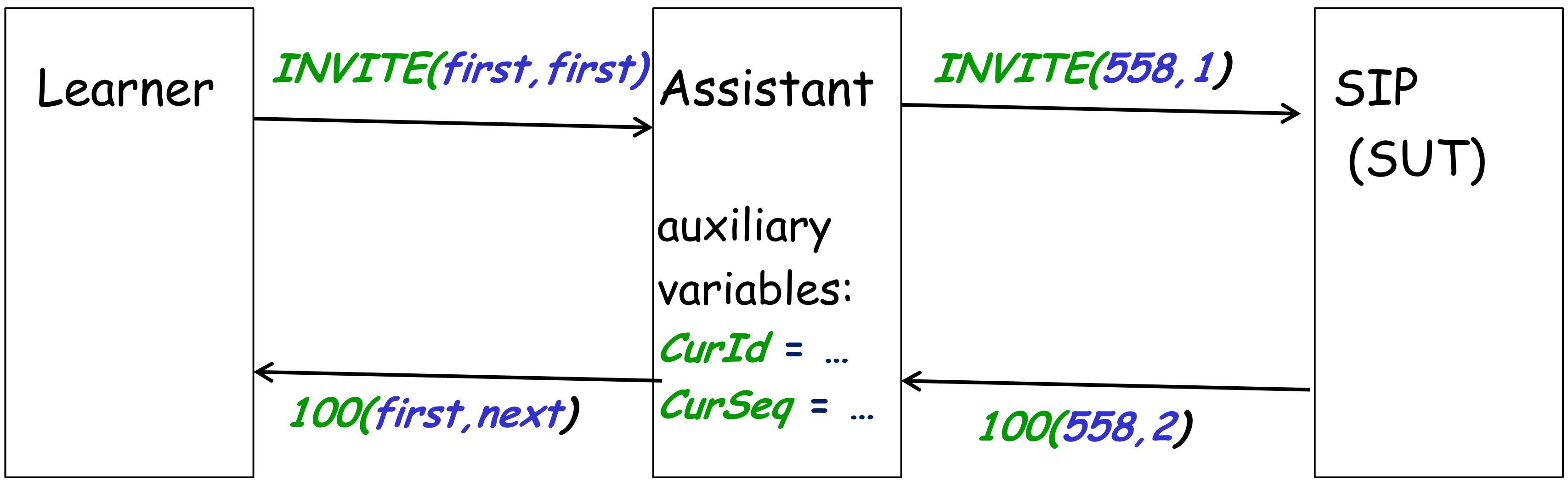
Adapting to Automata Learning



Adapting to Automata Learning



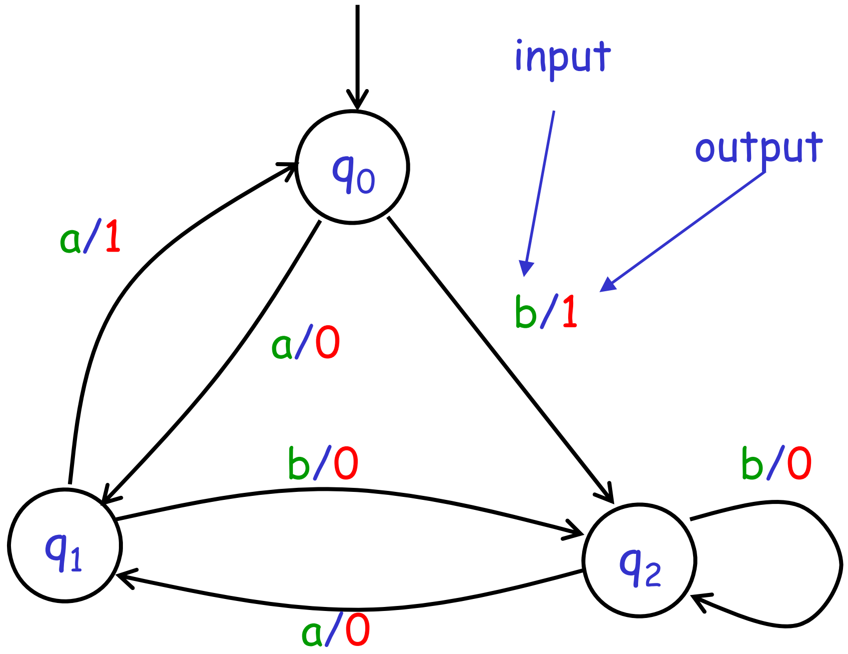
Adapting to Automata Learning



Abstraction: Formal definition

Possibly Infinite State Mealy Machine

I	input symbols
O	output symbols
Q	states
q_0	initial state
$\delta: Q \times I \rightarrow Q$	transition function
$\lambda: Q \times I \rightarrow O$	output function



Abstraction: Formal definition

Possibly Infinite State Mealy Machine		Abstraction	
I	input symbols	I^A	abstract input symbols
O	output symbols	O^A	abstract output symbols
Q	states	R	states
q_0	initial state	r_0	initial state
$\delta: Q \times I \rightarrow Q$	transition function	$\delta^R: R \times (I \cup O) \rightarrow R$	update
$\lambda: Q \times I \rightarrow O$	output function	$\alpha_I: R \times I \rightarrow I^A$	input abstraction
		$\alpha_O: R \times O \rightarrow O^A$	output abstraction

Abstraction: Formal definition

Possibly Infinite State Mealy Machine		Abstraction	
I , O	symbols	I^A , O^A	abstract symbols
Q , q_0	states , initial state	R , r_0	states , initial state
$\delta: Q \times I \to Q$	transition function	$\delta^R: R \times (I \cup O) \to R$	update
$\lambda: Q \times I \to O$	output function	$\alpha_I: R \times I \to I^A$	input abstraction
		$\alpha_O: R \times O \to O^A$	output abstraction

Abstraction: Formal definition

Possibly Infinite State Mealy Machine

I , O	symbols
Q , q_0	states , initial state
δ: $Q \times I \rightarrow Q$	transition function
λ: $Q \times I \rightarrow O$	output function

Abstraction

I^A , O^A	abstract symbols
R , r_0	states , initial state
δ^R: $R \times (I \cup O) \rightarrow R$	update
α_I: $R \times I \rightarrow I^A$	input abstraction
α_O: $R \times O \rightarrow O^A$	output abstraction

Abstracted Mealy Machine

I^A , O^A abstract symbols

$Q \times R$, $\langle q_0, r_0 \rangle$ states , initial state

δ^A: $Q \times R \times I^A \rightarrow Q \times R$ transition function:

$$\delta^A(\langle q,r \rangle , a^A) = \{ \langle \delta (q , a), \delta^R(r , a) \rangle \mid \alpha_I (r , a) = a^A \}$$

λ^A: $Q \times R \times I^A \rightarrow O^A$ output function:

$$\lambda^A(\langle q,r \rangle , a^A) = \{ \alpha_O (\delta^R(r , a) , \lambda (q , a)) \mid \alpha_I (r , a) = a^A \}$$

In general Nondeterministic

Abstraction: Formal definition

Abstracted Mealy Machine

I^A , O^A abstract symbols

$Q \times R$, $\langle q_0, r_0 \rangle$ states , initial state

$\delta^A: Q \times R \times I^A \rightarrow Q \times R$ transition function:

$\delta^A(\langle q,r \rangle , a^A) = \{ \langle \delta(q, a), \delta^R(r, a) \rangle \mid \alpha_I(r, a) = a^A \}$

$\lambda^A: Q \times R \times I^A \rightarrow O^A$ output function:

$\lambda^A(\langle q,r \rangle , a^A) = \{ \alpha_O(\delta^R(r, a), \lambda(q, a)) \mid \alpha_I(r, a) = a^A \}$

Exists equivalence $\approx$ on $Q \times R$ s.t.

- $\langle q,r \rangle \approx \langle q',r' \rangle$ and $\alpha_I(r,a) = \alpha_I(r',a')$ implies
  - $\langle \delta(q, a), \delta^R(r, a) \rangle \approx \langle \delta(q', a'), \delta^R(r', a') \rangle$
  - $\alpha_O(\delta^R(r, a), \lambda(q, a)) = \alpha_O(\delta^R(r', a'), \lambda(q', a'))$

Modified Criterion

Exists equivalence $\approx$ on $Q \times R$ s.t.

- $\langle q,r\rangle \approx \langle q',r'\rangle$ and $\alpha_I(r,a) = \alpha_I(r',a')$ implies
  - $\langle \delta(q,a), \delta^R(r,a)\rangle \approx \langle \delta(q',a'), \delta^R(r',a')\rangle$
  - $\alpha_O(\delta^R(r,a), \lambda(q,a)) = \alpha_O(\delta^R(r',a'), \lambda(q',a'))$

Can happen, e.g., if Q can be written $L \times R$, and

- if $\delta(\langle l,r\rangle, a) = \langle l',r'\rangle$ then
  - $r' = \delta^R(r,a)$
  - l' depends only on $\alpha_I(r,a)$
- if $\lambda(\langle l,r\rangle, a) = b$ then
  - $\alpha_O(\delta^R(r,a), b)$ depends only on $\alpha_I(r,a)$

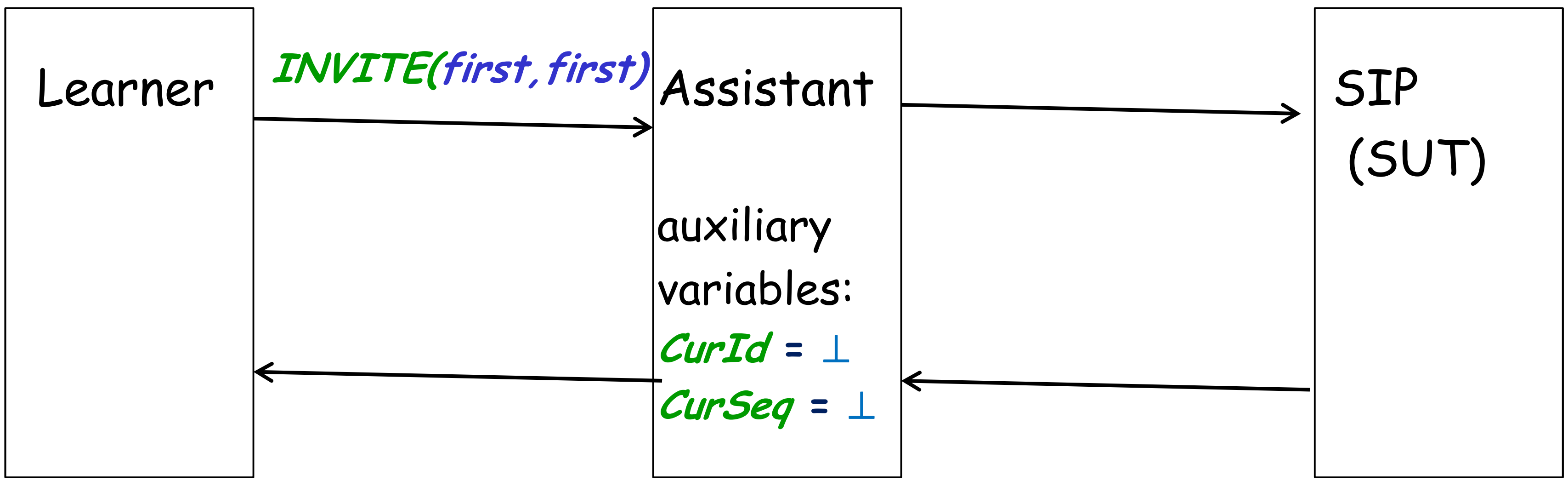
Mapping parameters of input messages

	first	next	last
cid	CurId = ⊥ and **Method = INVITE** or cid = CurId		<otherwise>
cseq	CurSeq = ⊥ and **Method = INVITE** or cseq = CurSeq	cseq = CurSeq+1	<otherwise>

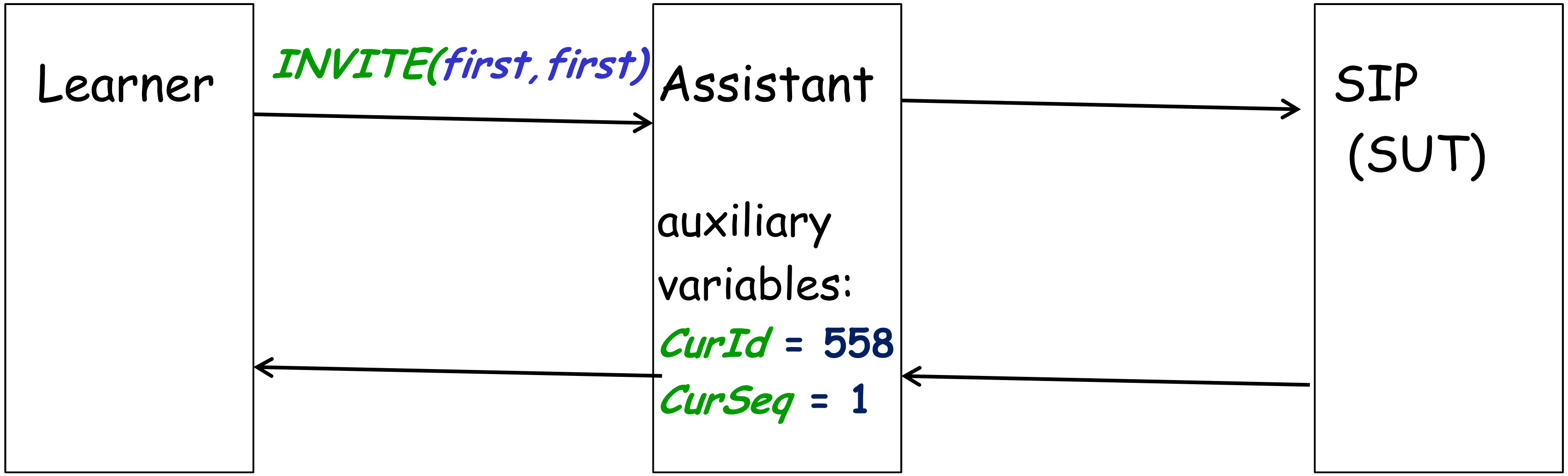
Maintaining auxiliary variables

	first	last	next
CurId	:= cid	<unchanged>	
CurId	:= cseq	<unchanged>	<unchanged>

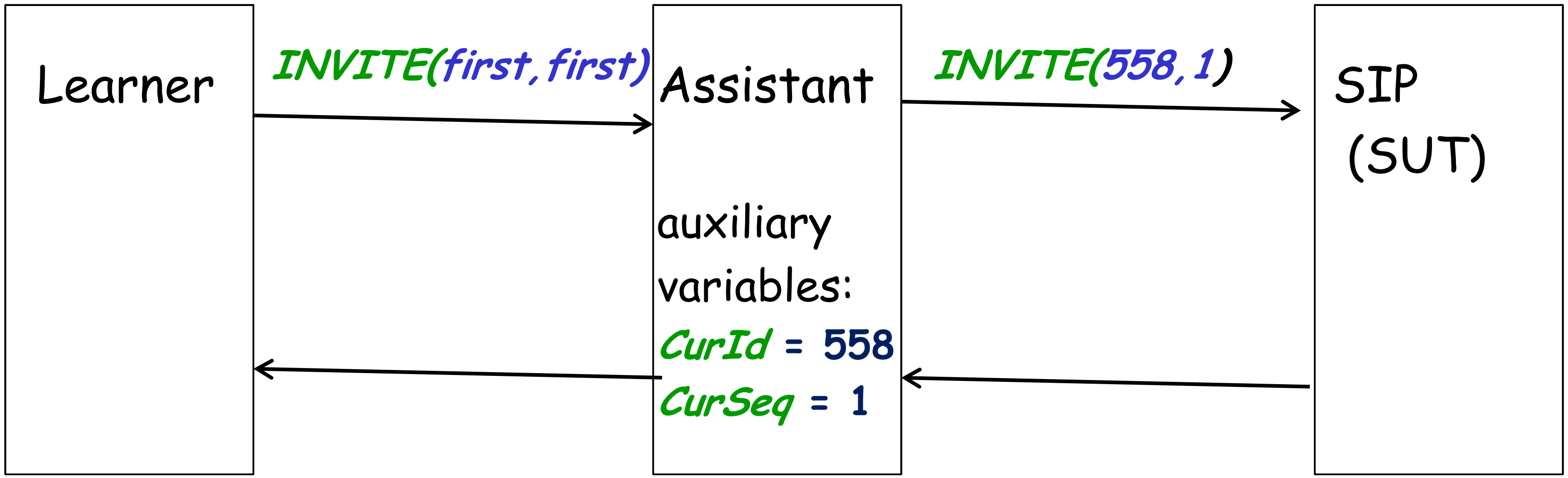
Inference by Abstraction



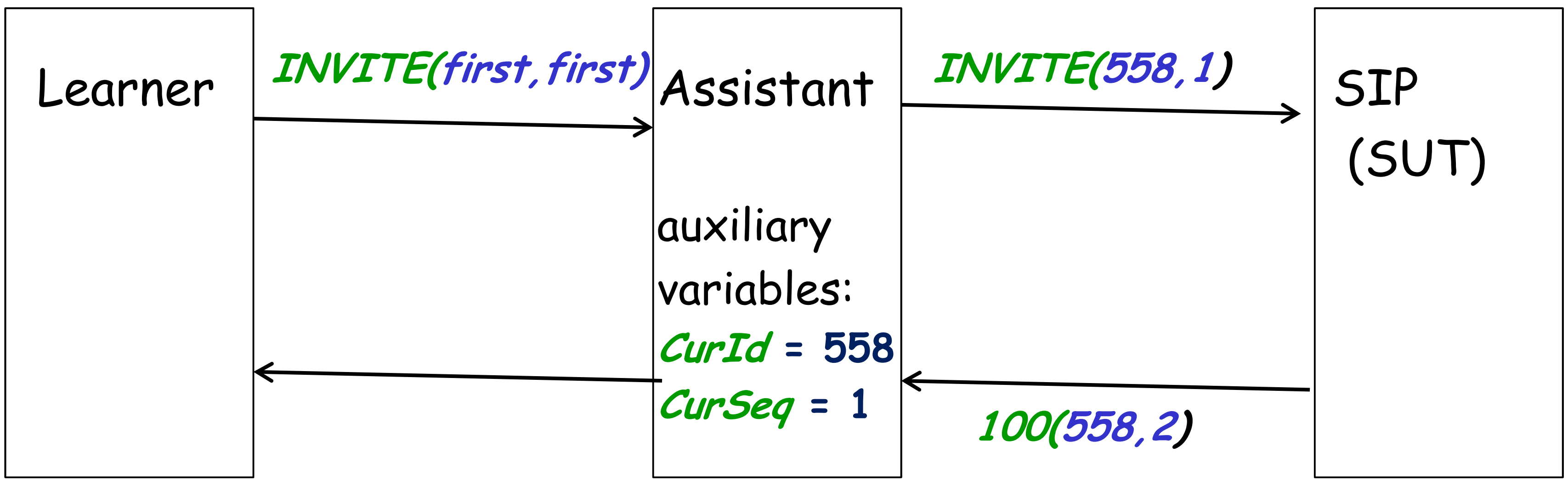
Inference by Abstraction



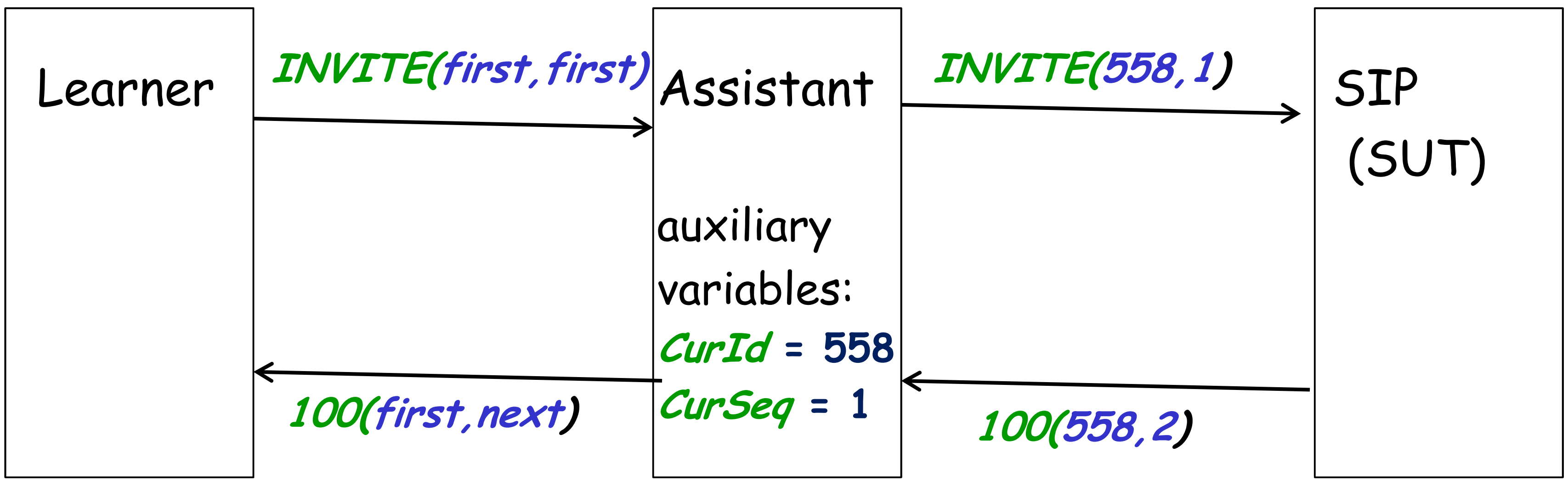
Inference by Abstraction



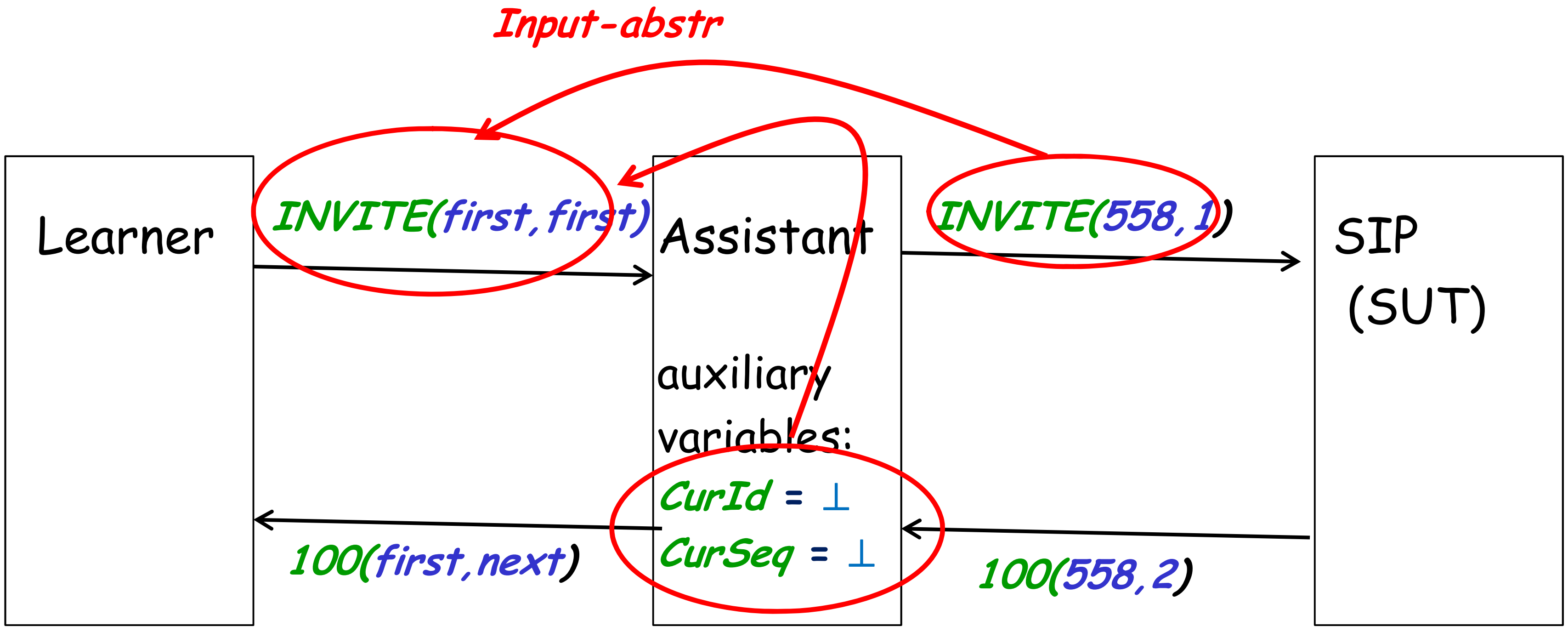
Inference by Abstraction



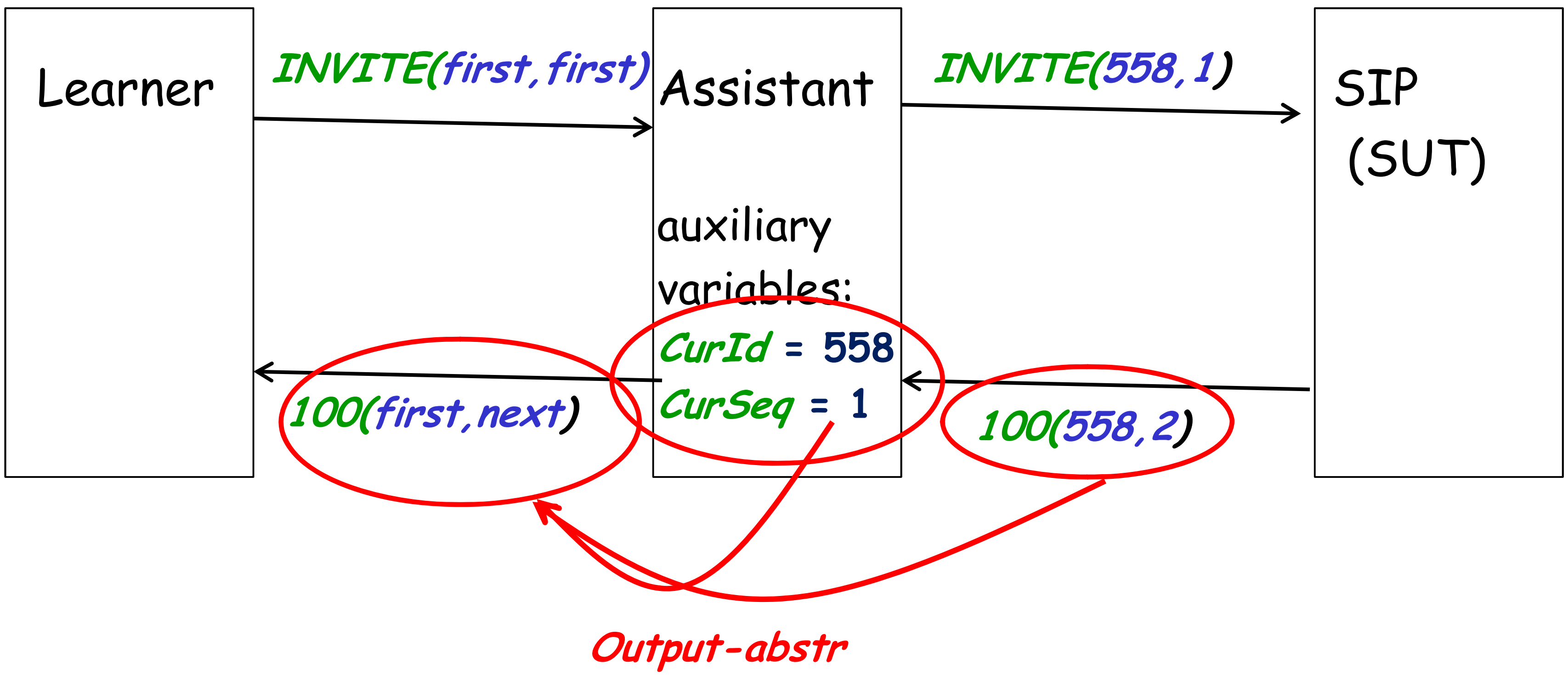
Inference by Abstraction



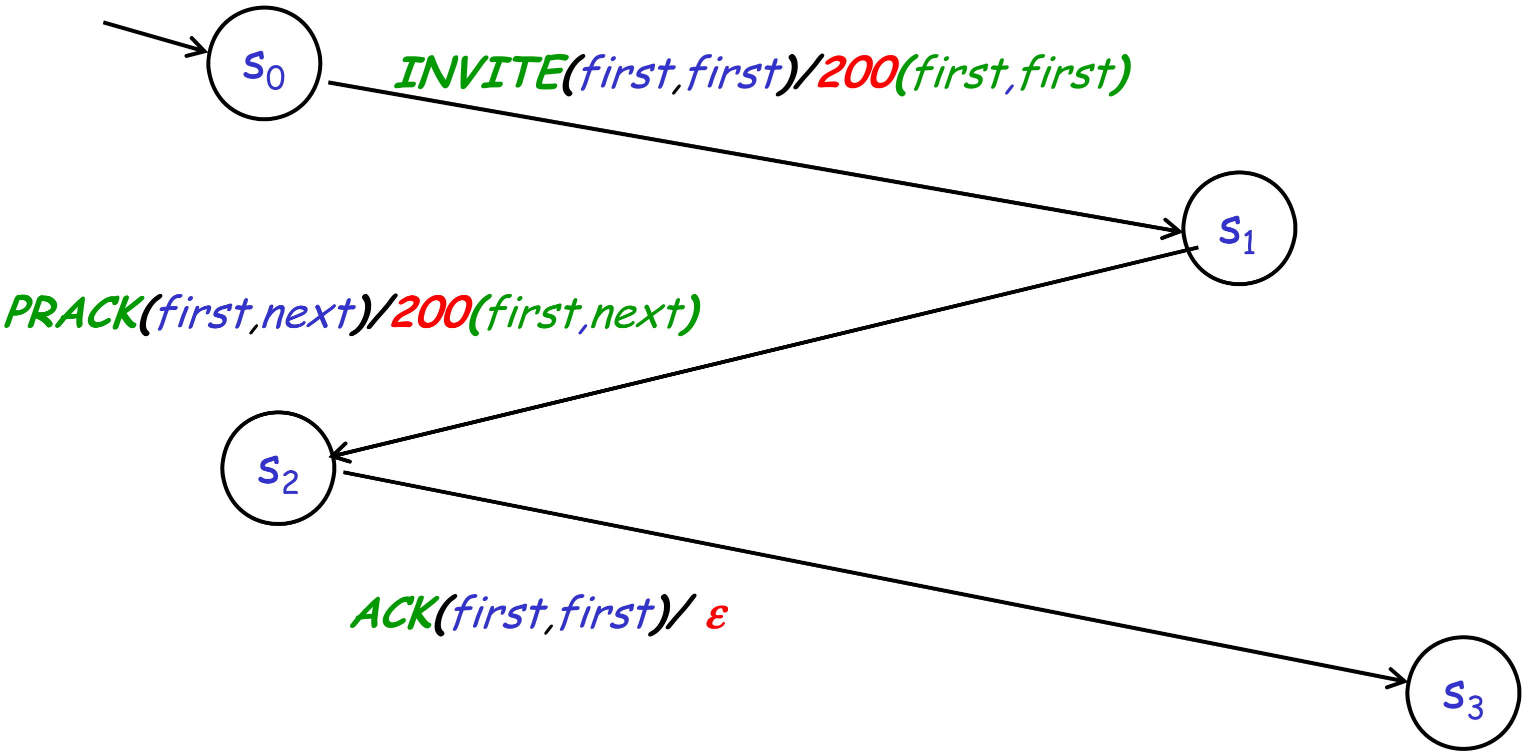
Abstraction Mappings



Abstraction Mappings

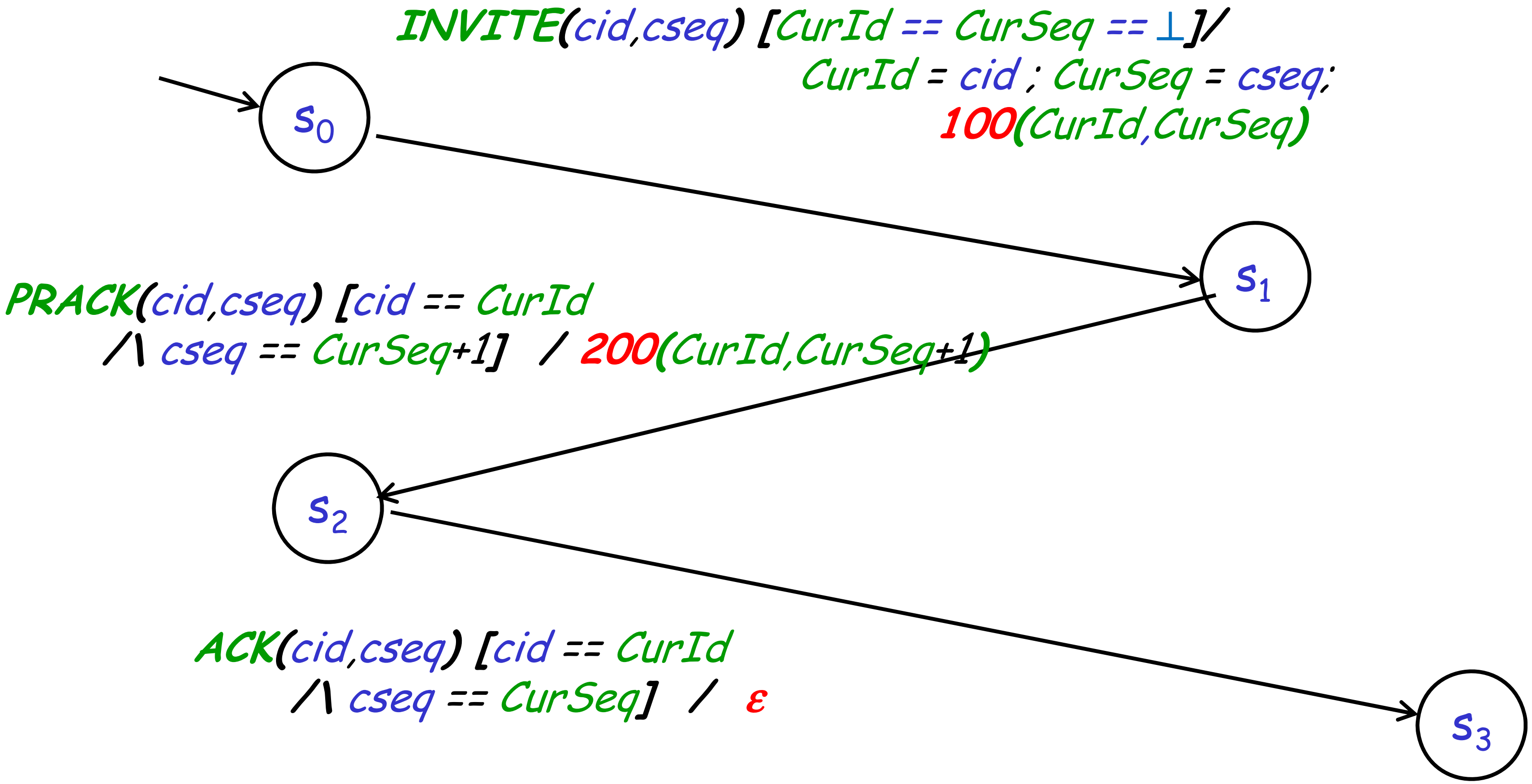


Model inferred by Learner (part)



What the SUT must have done:

Variables: *CurId*, *CurSeq*



Experiments

- Learner: the LearnLib tool (developed at TU Dortmund)
  - Efficient implementation of L*
  - Several equivalence oracles, e.g., controllable-size random test suite.
- SUT: ns-2 protocol simulator
  - Provides implementations of many standard protocols
  - Rather convenient C++ interface (no packet analyzer necessary)
- Assistant
  - Bridges asynchronous interface of LearnLib w. synchronous interface of ns-2
  - Implements instantiation of input symbols, and abstraction of output symbols

Learning SIP in ns-2

- Inference: about 1 thousand membership queries
  one equivalence query
- Model w. 10 locations and 70 transitions
- ns-2 implementation does not check incoming cseq parameter, just returns it.

Resulting Model

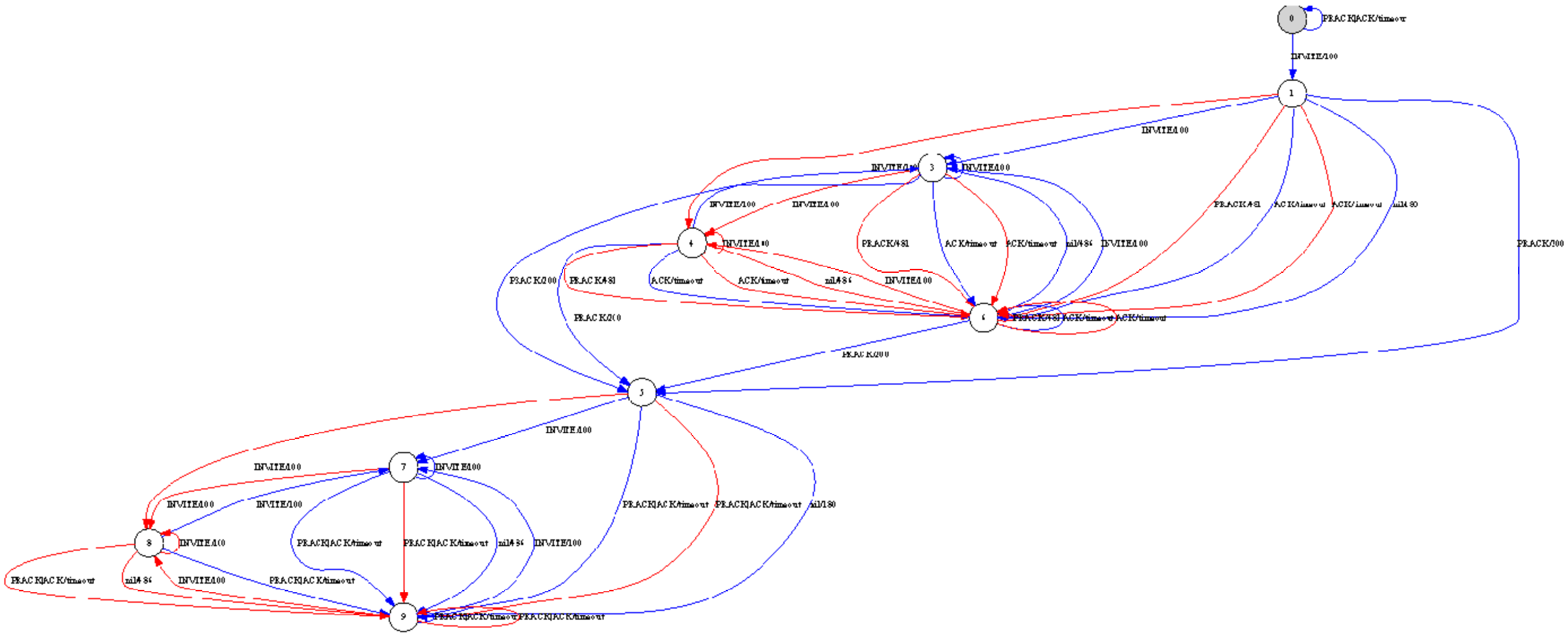


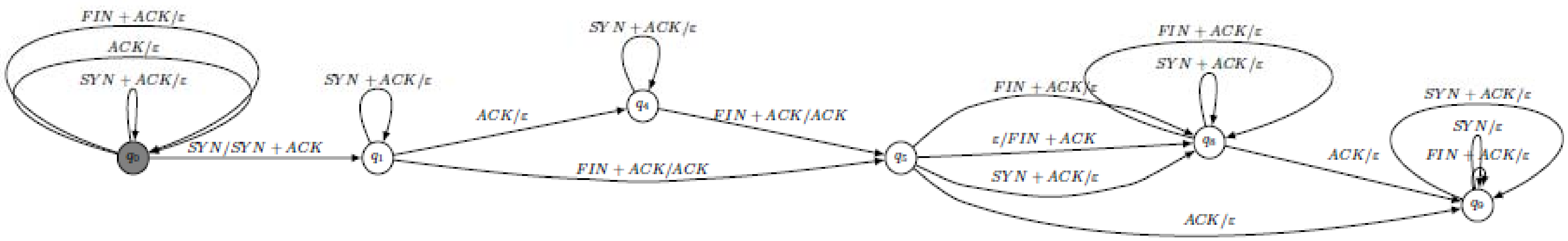
Fig. 3. Full SIP model

Transport Control Protocol (TCP)

- Only connection establishment and termination
- SUT is ns-2 implementation of TCP
- Consider 2 sequence number parameters
- Similar type of abstraction

TCP

- Model of behavior of TCP in ns-2
- Only transitions with “accepted” values of input parameters are shown.
- Values of parameters not displayed



Conclusions

- Basic Principles of Automata Learning for Finite-State systems understood
- Learning and Conformance Testing:
  - Two sides of the same coin.
- Learning for extended automata models largely unexplored

Some Future work

- Techniques for handling common forms of data
- Dynamically refining abstractions
- Learning nondeterministic models
- Learning timed models in practice
- Learning under assumptions on module usage
- Efficient search for counterexamples
- Efficient construction of test harnesses
- Some references can be found at

http://leo.cs.tu-dortmund.de:8100/