Active Automata Learning: from L^* to $L^\#$

Frits Vaandrager

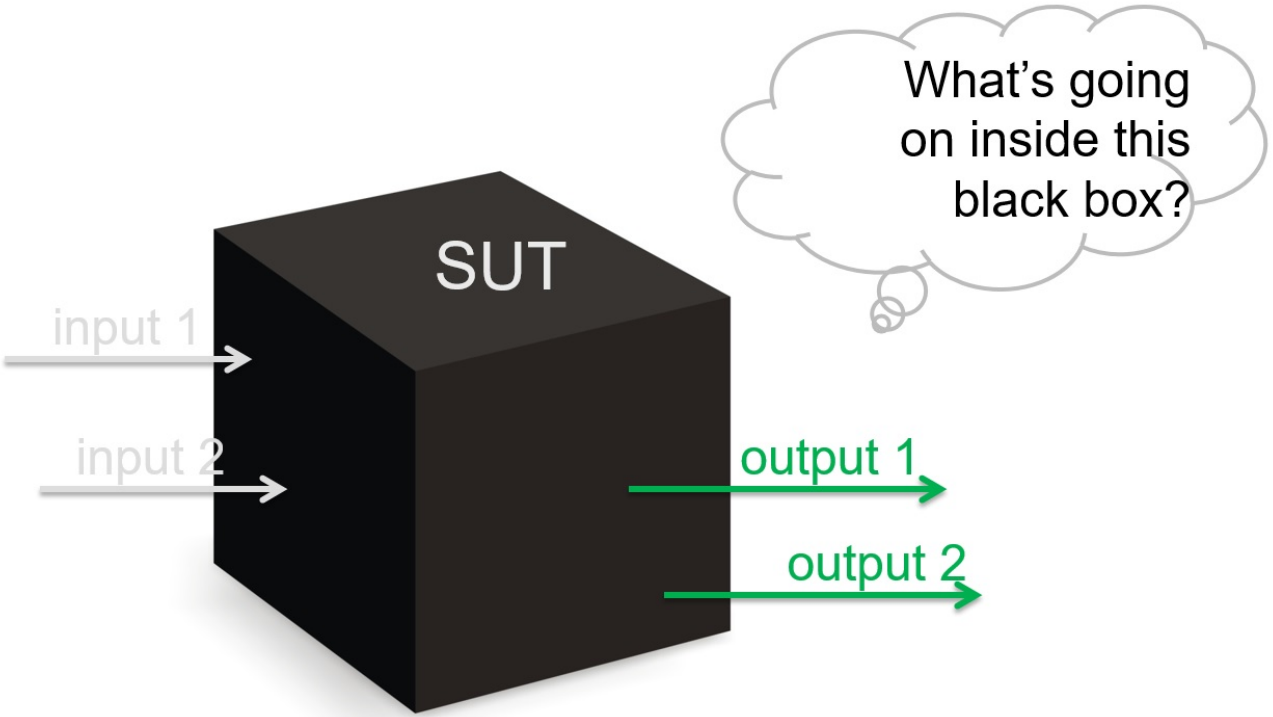
Radboud University Nijmegen

VSTTE, October 19, 2021

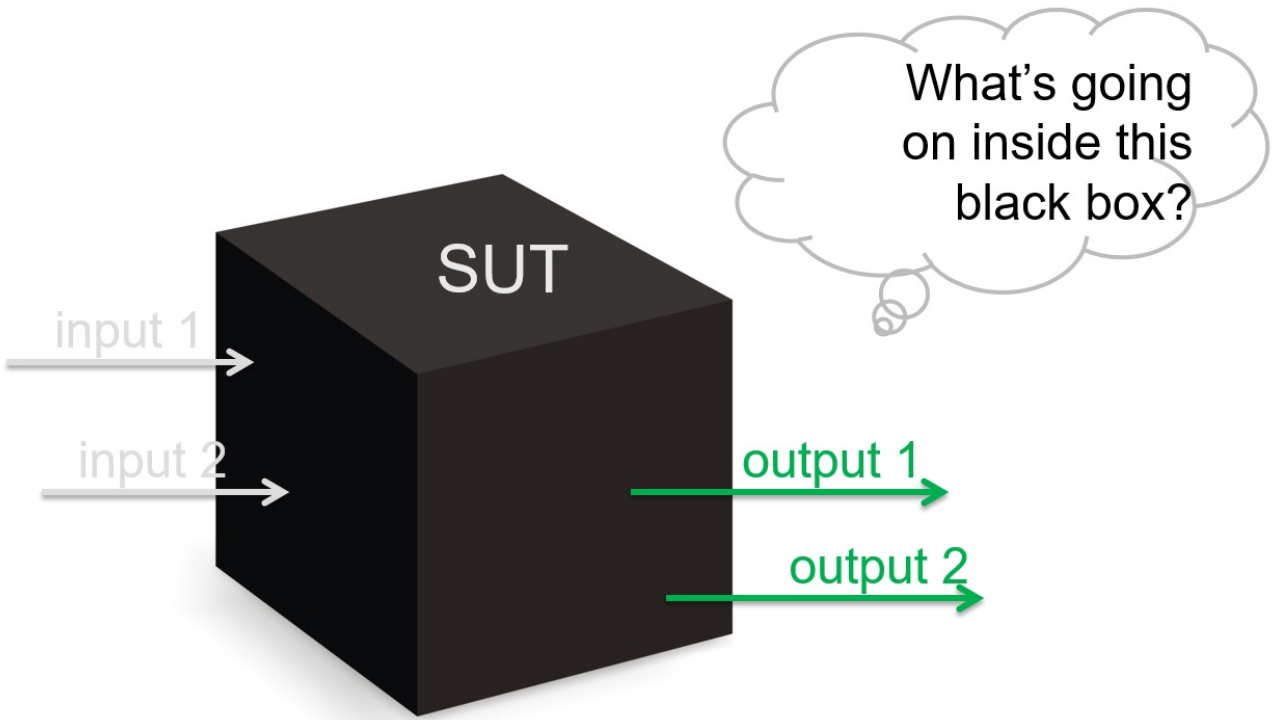
Outline

1. Introduction
2. L^* and its Descendants
3. Applications
4. $L^\#$: A New Perspective
5. Conclusions and Future Work

Research Question



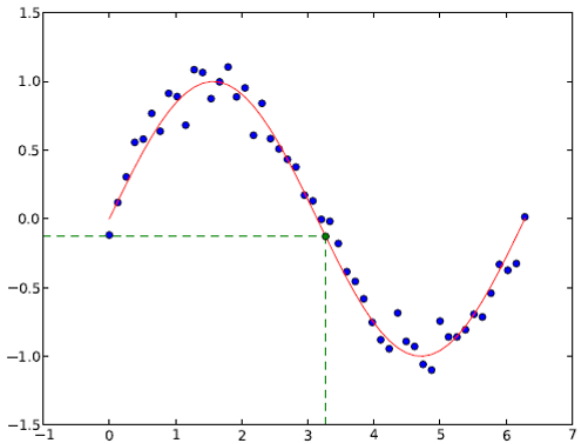
Research Question

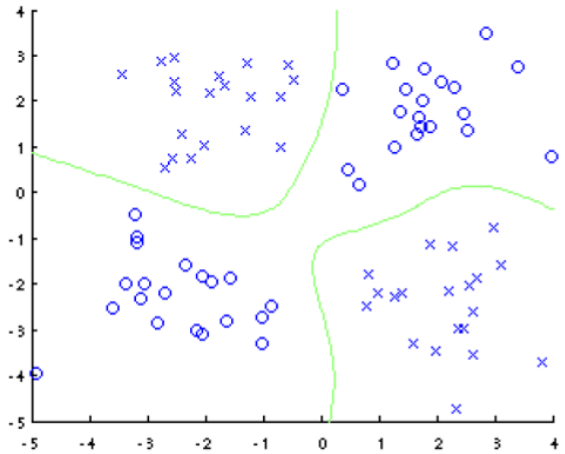


We assume SUT behaves deterministically and can be reset.

Machine Learning in General

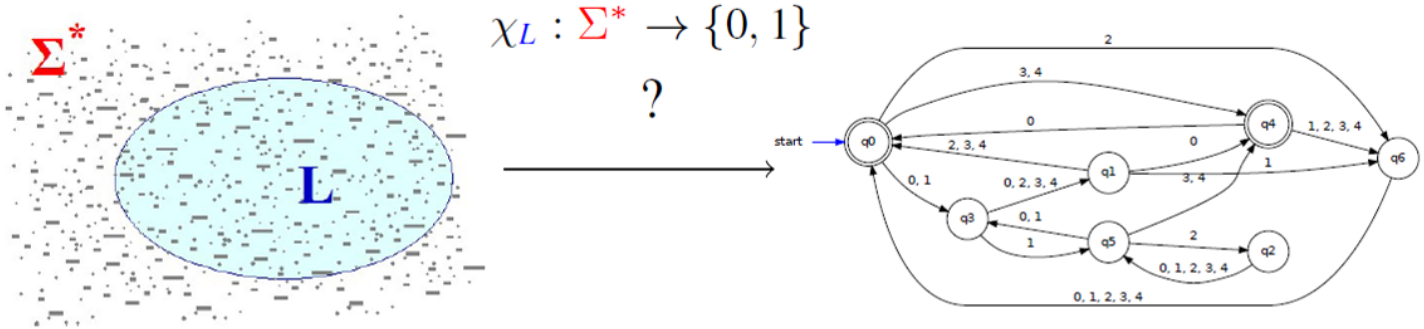
- Given a sample $M = \{(x, y) \mid x \in X, y \in Y\}$
- Find $f : X \to Y$ such that $f(x) = y, \forall (x, y) \in M$
- Predict $f(x)$ for all $x \in X$





Learning Regular Languages

Let Σ be an alphabet and let $L \subseteq \Sigma^*$ be a regular language (*the target language*)



- Edward F Moore, *Gedanken-experiments on sequential machines*, 1956
- E. Mark Gold, *System Identification via State Characterization*, 1972
- Dana Angluin, *Learning regular sets from queries and counterexamples*, 1987

Regular Languages and Congruences

Definition

The equivalence relation $\sim_L$ on Σ^* induced by a language $L \subseteq \Sigma^*$:

$$u \sim_L v \quad \textit{iff} \quad \forall w \in \Sigma^* : u \cdot w \in L \Leftrightarrow v \cdot w \in L$$

This relation is a right-congruence with respect to concatenation:

$$\forall u, v, w \in \Sigma^* : u \sim_L v \Rightarrow u \cdot w \sim_L v \cdot w$$

Theorem (Myhill-Nerode, 1958)

Language L is regular iff $\sim_L$ has finitely equivalence classes.

Visualisation

Consider the regular language $a(a \mid b)^* b$

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

Visualisation

Consider the regular language $a(a \mid b)^* b$

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

Hankel Matrix

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

Hankel Matrix

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

Hankel Matrix

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

Hankel Matrix

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

Hankel Matrix

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

Hankel Matrix

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

Hankel Matrix

- $u \sim_L v$ iff rows of u and v in Hankel matrix for L have the same color

Hankel Matrix

- $u \sim_L v$ iff rows of u and v in Hankel matrix for L have the same color
- Language L is regular iff its Hankel matrix contains a finite number of distinct rows, i.e., colors

Hankel Matrix

- $u \sim_L v$ iff rows of u and v in Hankel matrix for L have the same color
- Language L is regular iff its Hankel matrix contains a finite number of distinct rows, i.e., colors
- The number of states in the smallest DFA for L equals the number of colors in the Hankel matrix

What is the FSM for this Hankel Matrix?

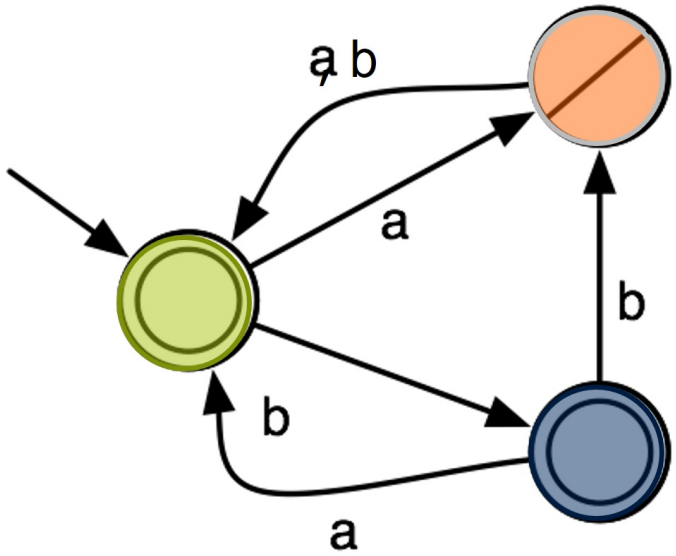
	_	a	b	aa	ab	ba	bb	aaa	aab	aba
_	1	0	1	1	1	1	0	0	1	0
a	0	1	1	0	1	0	1	1	1	1
b	1	1	0	0	1	1	1	1	1	1
aa	1	0	1	1	1	1	0	0	1	0
ab	1	0	1	1	1	1	0	0	1	0
ba	1	0	1	1	1	1	0	0	1	0
bb	0	1	1	0	1	0	1	0	1	1
aaa	0	1	1	0	1	0	1	1	1	1
aab	1	1	0	0	1	1	1	1	1	1
aba	0	1	1	0	1	0	1	1	1	1

What is the FSM for this Hankel Matrix?

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

Solution

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



Colors of rows in Hankel matrix give us the states.
Access strings and one-letter extensions allow us to determine transitions.
Column for empty suffix gives us the accepting states.

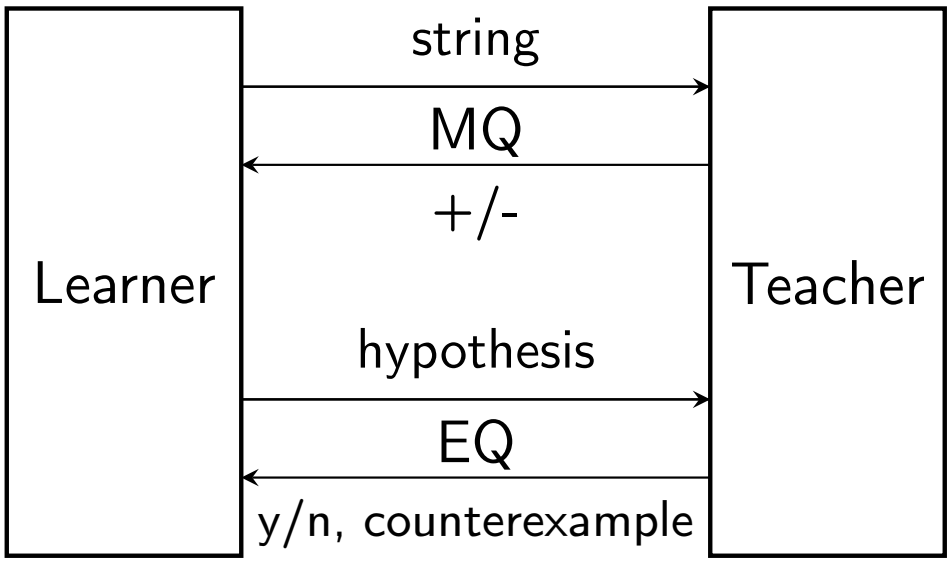
What if Hankel Matrix is Incomplete?

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

Problem to color such a matrix is NP-hard! Gold (1972)

Minimally Adequate Teacher

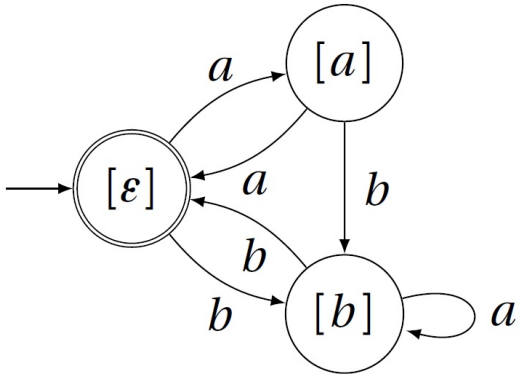
Learning framework proposed by Angluin (1987):



Learner asks membership queries and equivalence queries

Angluin's L^* Algorithm

		ε	a
$\mathcal{U}$	ε	1	0
	a	0	1
	b	0	0
	bb	1	0
$\mathcal{U}\Sigma\setminus\mathcal{U}$	aa	1	0
	ab	0	0
	ba	0	0
	bba	0	1
	bbb	0	0



- Observation table is consistent if whenever rows $u, v \in \mathcal{U}$ are the same, rows ui and vi are also the same, for all $i \in \Sigma$
- Table is closed when every row from $\mathcal{U}\Sigma$ is also a row of $\mathcal{U}$
- If table is not closed extend $\mathcal{U}$; if it is not consistent extend $\mathcal{V}$
- Construct hypothesis when table is closed and consistent
- All prefixes of a counterexample are added to $\mathcal{U}$

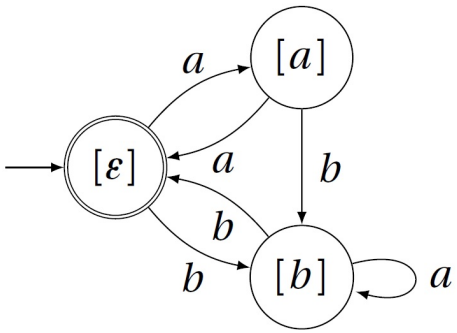
Rivest & Schapire

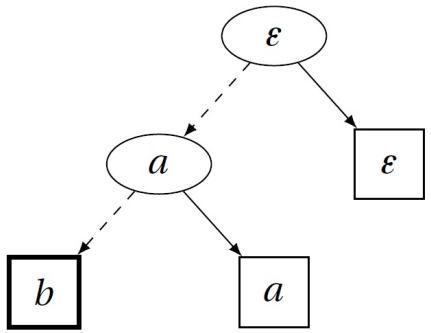
- Optimization of L^* proposed by Rivest & Schapire (1993)
- Rather than adding all prefixes of a counterexample to S, just a single suffix is added to E
- This suffix is constructed using a binary search that requires $\mathcal{O}(\log m)$ additional membership queries, where m is the length of the maximal counterexample

Kearns & Vazirani

- Optimization of L^* proposed by Kearns & Vazirani (1994)
- Observation table contains inherent redundancy: usually not all suffixes needed to distinguish equivalence classes
- Uses discrimination trees instead

		ε	a
$\mathcal{U}$	ε	1	0
	a	0	1
	b	0	0
	bb	1	0
$\mathcal{U}\Sigma\setminus\mathcal{U}$	aa	1	0
	ab	0	0
	ba	0	0
	bba	0	1
	bbb	0	0





TTT Algorithm

- Proposed by Isberner, Howar & Steffen (2014)
- Maintains a spanning Tree, a discrimination Tree, and a discriminator Trie
- space-optimal: there can be no algorithm with an asymptotically lower space complexity correctly accomplishing the same task
- "our evaluations show that TTT is superior to virtually every other learning algorithm"

ADT Algorithm

- Proposed by Markus Frohme (2015) in MSc thesis
- Integrates discrimination trees with adaptive distinguishing sequences, queries in which the choice of input symbols may depend on the output received in response to previous inputs
- Outperforms other algorithms on many benchmarks

Overview of L^* and its descendants

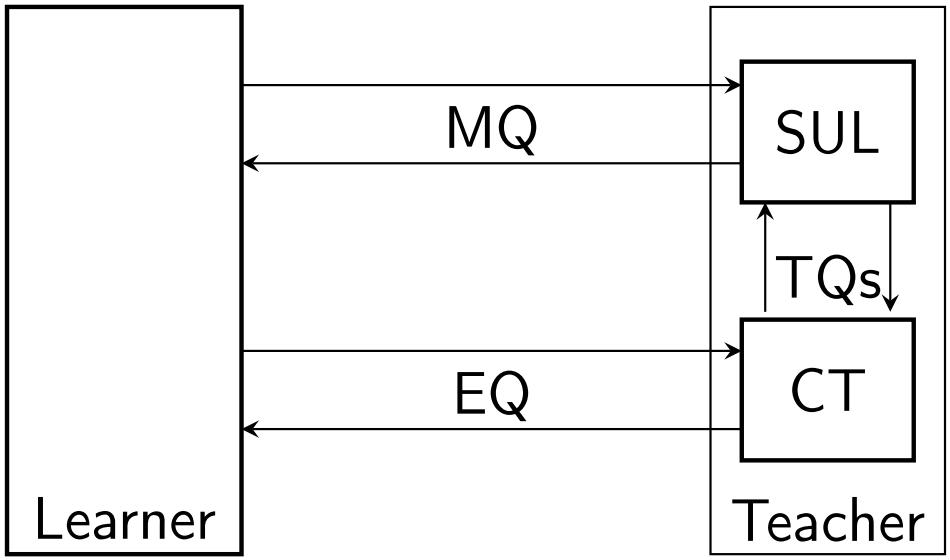
Algorithm	Query Complexity	Symbol Complexity
L^*	$\mathcal{O}(kn^2m)$	$\mathcal{O}(kn^2m)$
Maler/Pnueli	$\mathcal{O}(kn^2m)$	$\mathcal{O}(kn^2m)$
Shabhaz	$\mathcal{O}(kn^2m)$	$\mathcal{O}(kn^2m)$
Suffix1by1	$\mathcal{O}(kn^2m)$	$\mathcal{O}(kn^2m)$
Rivest/Shapire	$\mathcal{O}(kn^2 + n\log m)$	$\mathcal{O}(kn^2m + nm\log m)$
Kearns/Vazirani	$\mathcal{O}(kn^2 + n^2m)$	$\mathcal{O}(kn^2m + n^2m^2)$
Observation Pack	$\mathcal{O}(kn^2 + n\log m)$	$\mathcal{O}(kn^2m + nm\log m)$
TTT	$\mathcal{O}(kn^2 + n\log m)$	$\mathcal{O}(kn^2m + nm\log m)$
ADT	$\mathcal{O}(kn^2 + n\log m)$	$\mathcal{O}(kn^2m + nm\log m)$

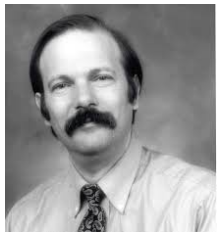
Here n is the number of states of the DFA, k is the number of input symbols, and m is the length of the maximal counterexample; table adapted from Isberner (2015).

LearnLib

Implements active and passive learning algorithms for DFAs and Mealy machines

Black Box Checking (Peled, Vardi & Yannakakis)

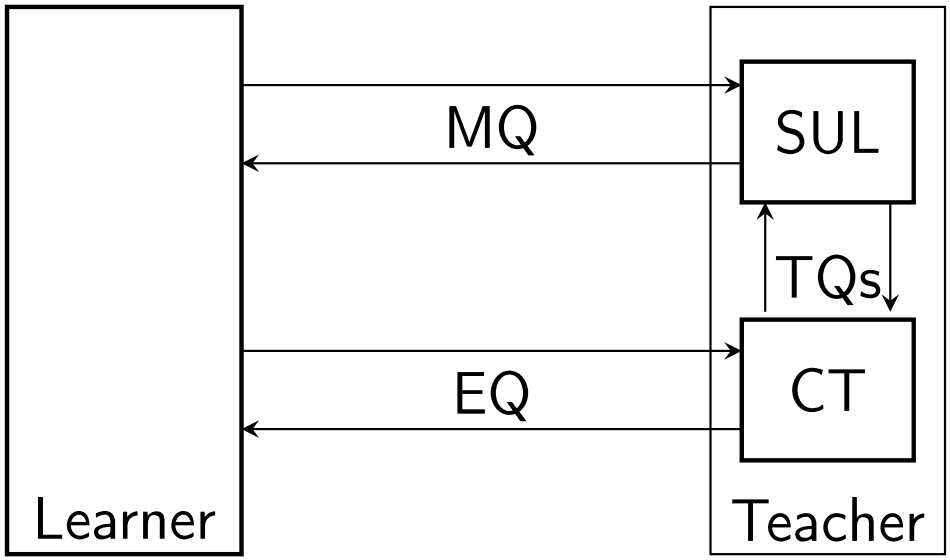


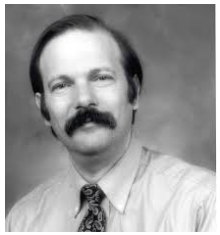


Learner: Formulate hypotheses

Conformance Tester (CT): Test correctness hypotheses

Black Box Checking (Peled, Vardi & Yannakakis)



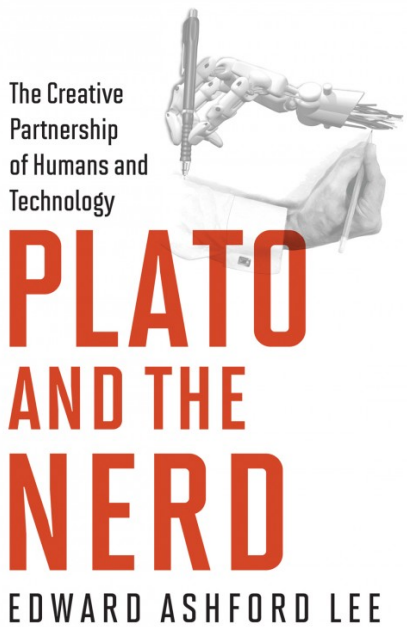


Learner: Formulate hypotheses

Conformance Tester (CT): Test correctness hypotheses

Model learning and conformance testing two sides of same coin!

Two Perspectives on Models

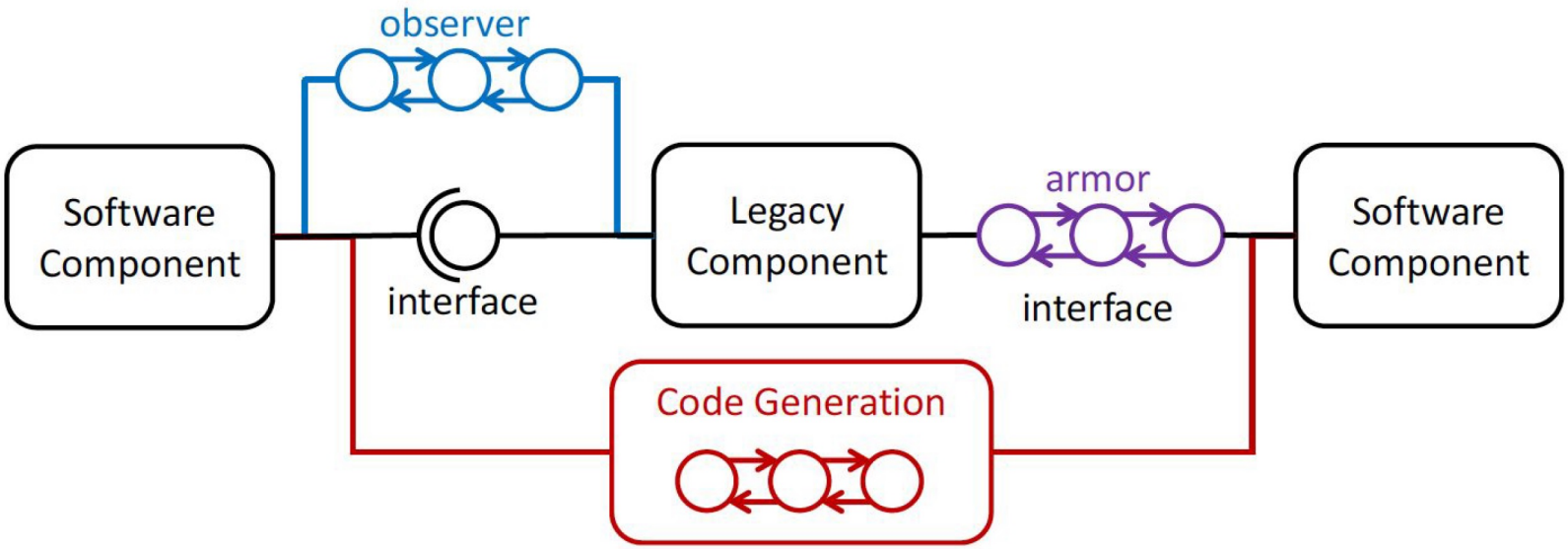


- Model: any description of a system that is not the thing-in-itself.
- Engineering perspective:
  "Can we build a system whose behavior matches that of a given model?"
- Science perspective:
  "Can we build a model whose behavior matches that of a given system?"
- Our vision:
  By properly combining both perspectives we can build better systems.

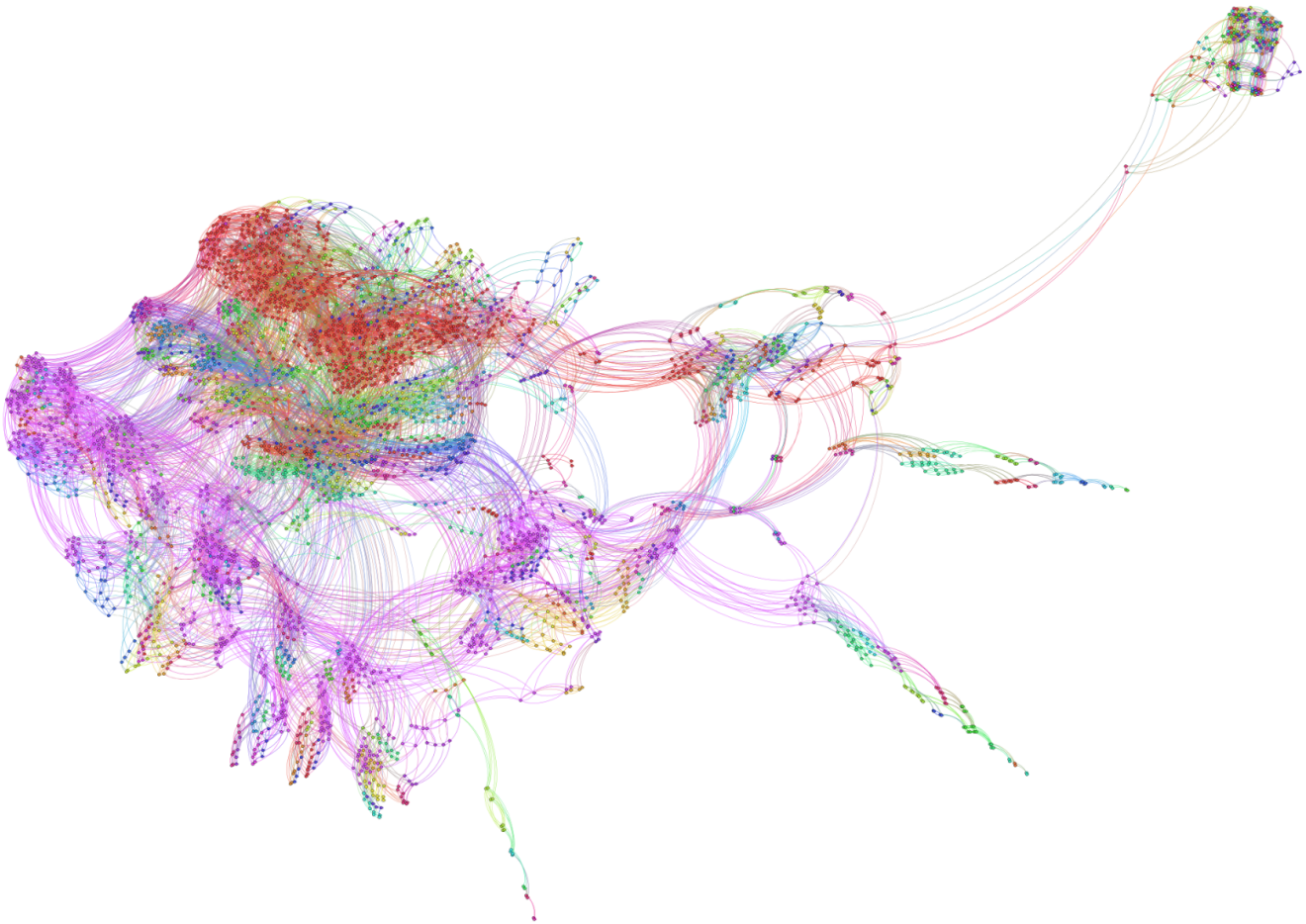
Engine Status Manager in Océ Printer (ICFEM'15)

Can we learn interface models of realistic printer controllers?

Potential Applications Interface Models



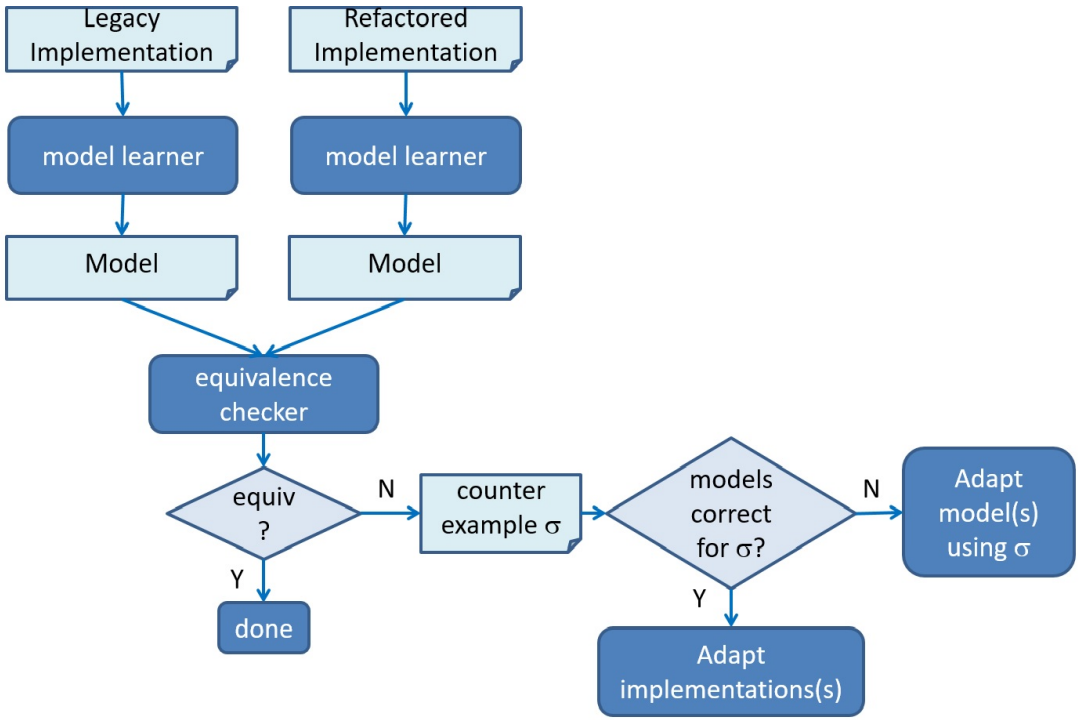
Mealy Machine for Engine Status Manager



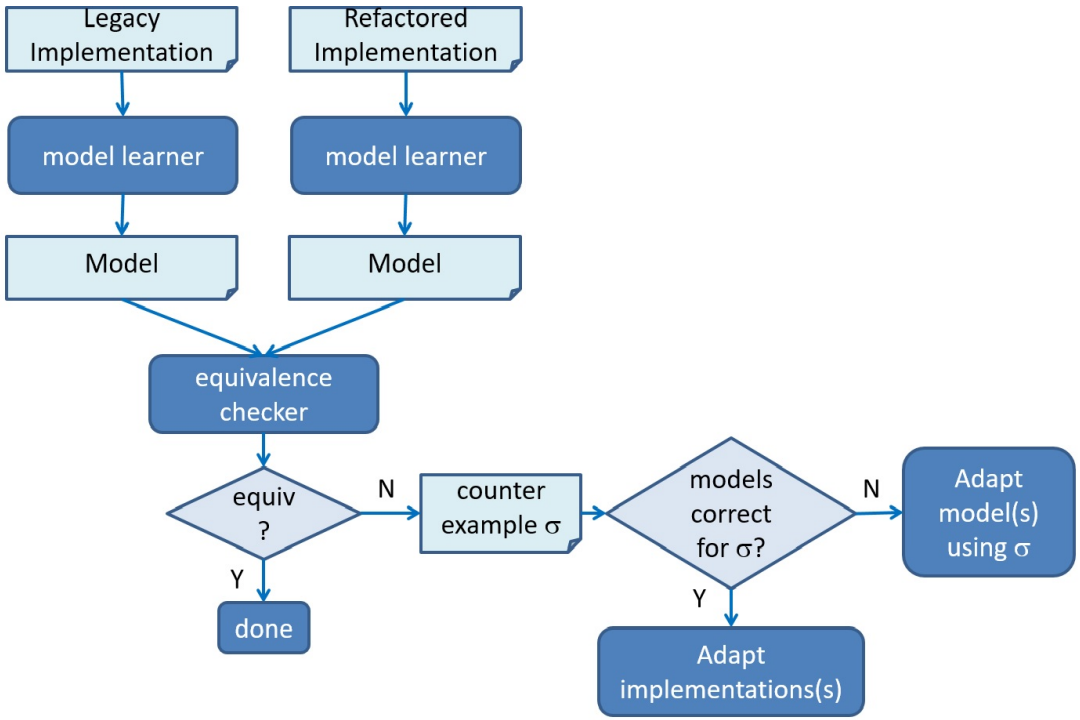
Power Control Service from Philips Healthcare (iFM'16)

Are legacy component and refactored implementation equivalent?

Refactoring Legacy Implementations



Refactoring Legacy Implementations



This approach allowed us to find several bugs in refactored implementations of power control service.

ASML Twinscan

ASML Challenge

Can active automata learning be used to support refactoring of legacy software at ASML?

ASML machines run on legacy software. Recent components have been designed using model-based techniques. Can we learn those?

Can we learn the hundreds of design and interface models used for high level control of the wafer flow during lot operation?

ASML Challenge

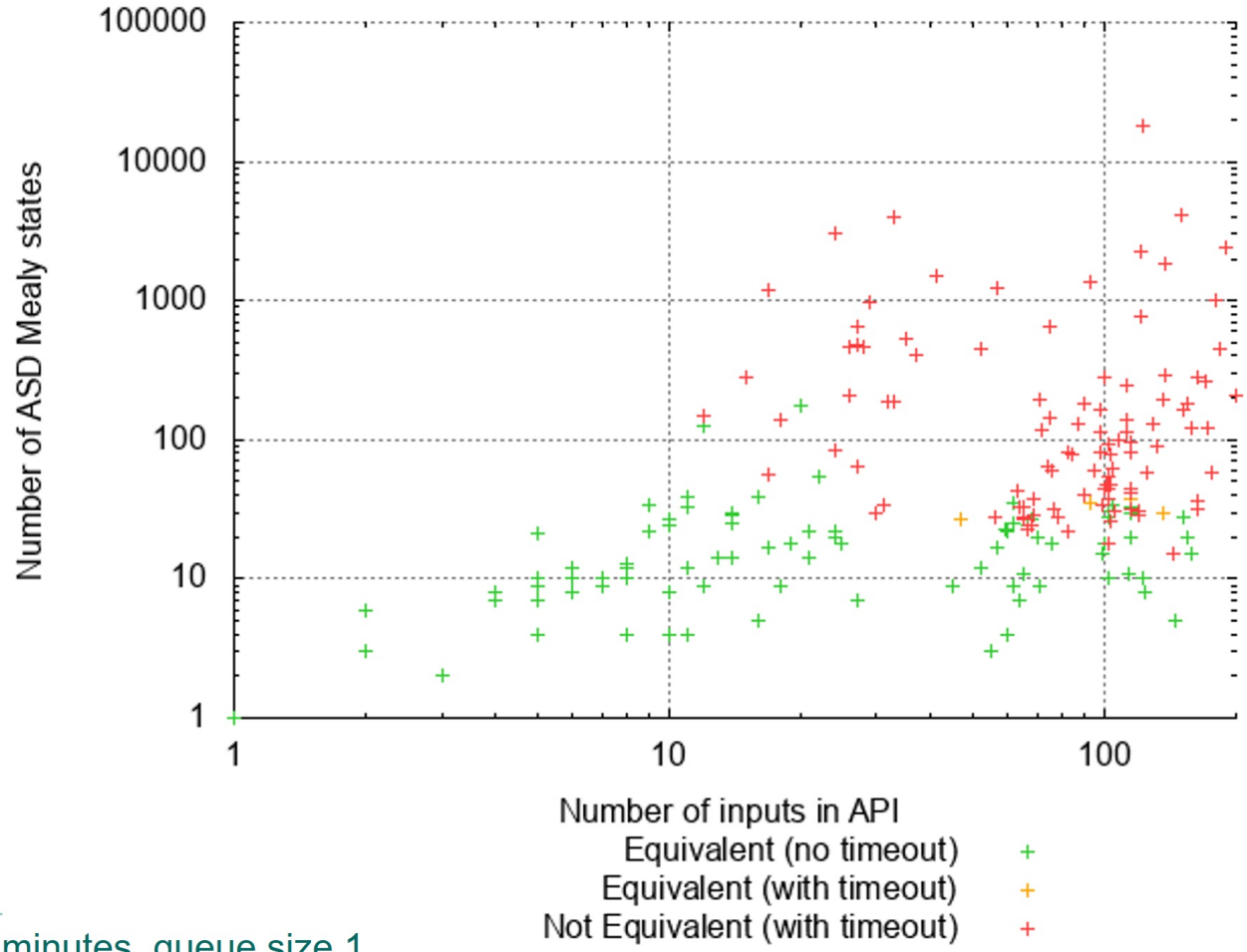
Can active automata learning be used to support refactoring of legacy software at ASML?

ASML machines run on legacy software. Recent components have been designed using model-based techniques. Can we learn those?

Can we learn the hundreds of design and interface models used for high level control of the wafer flow during lot operation?

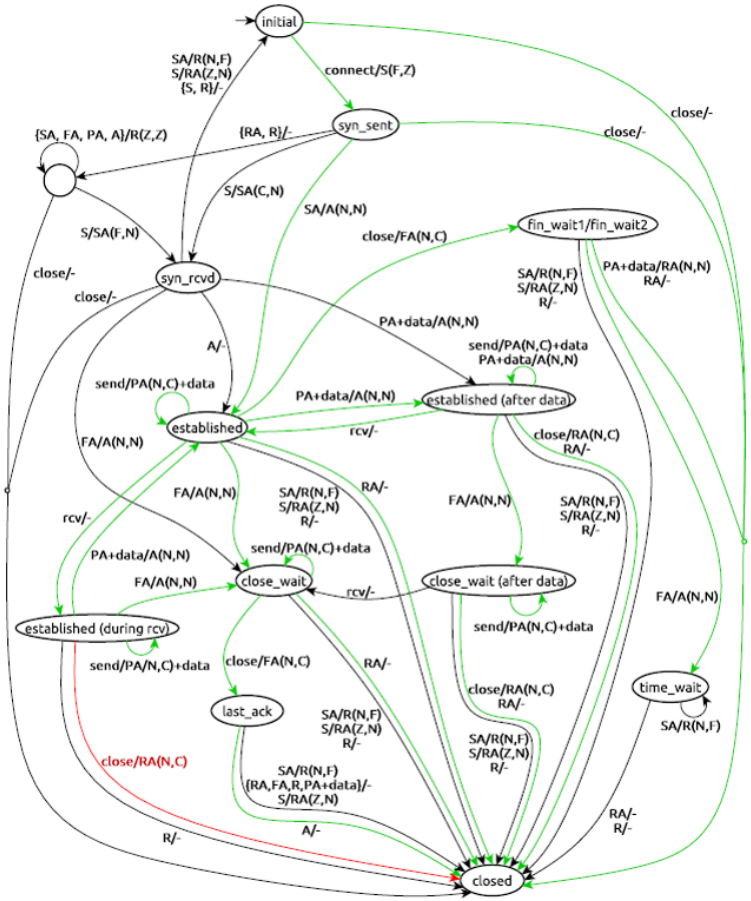
$\Rightarrow$ RERS @ TOOLympics'19

Results LearnLib on ASML Benchmarks



60 minutes, queue size 1

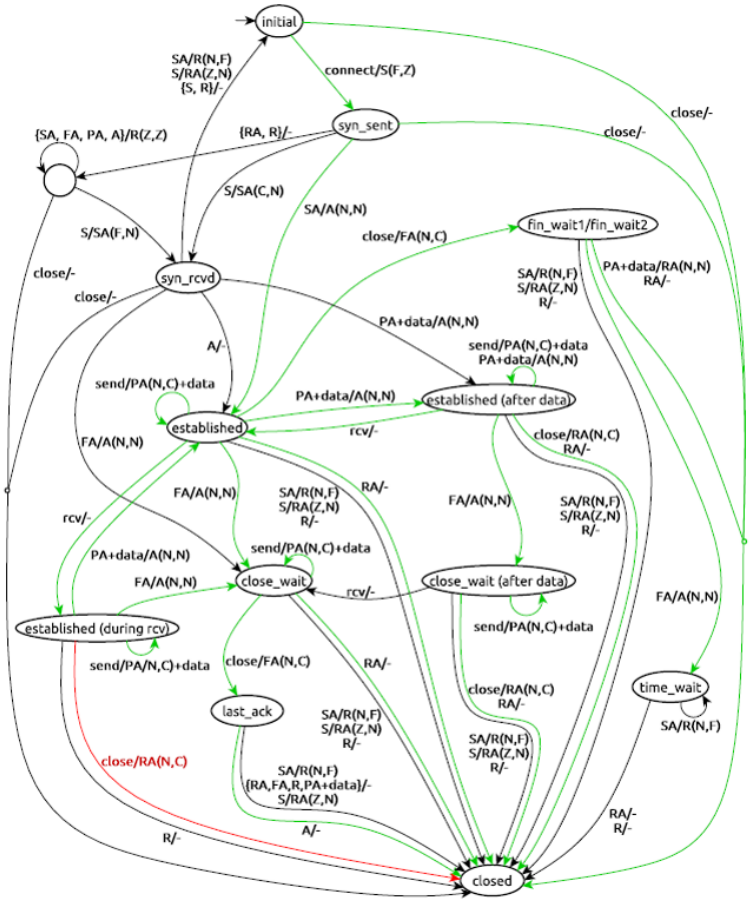
Bugs in Protocol Implementations



Standard violations found in implementations of major protocols:

- TLS (Usenix Security'15)
- TCP (CAV'16)
- SSH (Spin'17)

Bugs in Protocol Implementations



Standard violations found in implementations of major protocols:

- TLS (Usenix Security'15)
- TCP (CAV'16)
- SSH (Spin'17)

These findings led to bug fixes in implementations.

Other Protocol Case Studies

- DTLS
- Session Initiation Protocol (SIP)
- Message Queuing Telemetry Transport (MQTT) protocol
- Quick UDP Internet Connections (QUIC) protocol
- WiFi
- IEC 60870-5-104 protocol
- ...

Conclusions from Case Studies

1. Automata learning is emerging as a highly effective bug-finding technique, and slowly becoming a standard tool in the toolbox of the software engineer.
2. In applications, I/O interactions often require much time; we need algorithms that require fewer input symbols/resets to learn a model.
3. For the larger ASML benchmark, learning time is completely dominated by the equivalence oracle (model based testing); we need to integrate learning and testing.
4. We need extensions that can handle data values and timing.

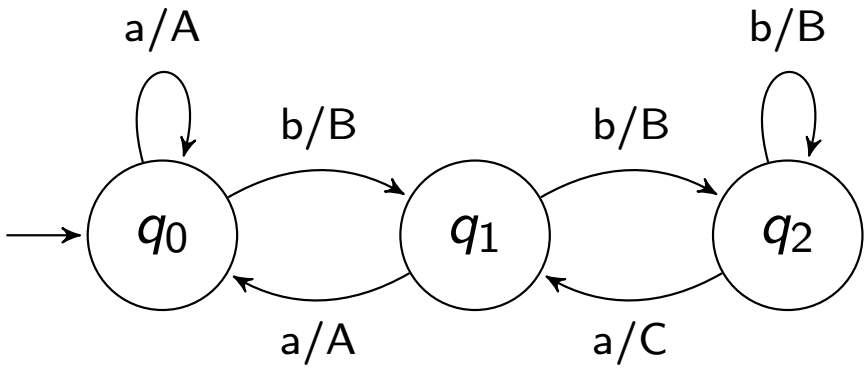
$L^\#$: A New Perspective on Active Automata Learning

Inspired by Brouwer's notion of apartness and Dijkstra's shortest path algorithm

Mealy machines

We fix a finite set I of inputs and a set O of outputs.
The diagram below shows a simple Mealy machine with

- inputs $a, b \in I$,
- outputs $A, B, C \in O$,
- states $Q = \{q_0, q_1, q_2\}$ and initial state q_0,
- transition function $\delta : Q \times I \to Q$, and
- output function $\lambda : Q \times I \to O$.



Equivalene of Mealy Machines

Definition

Two Mealy machines $\mathcal{M}$ and $\mathcal{N}$ are equivalent, $\mathcal{M} \approx \mathcal{N}$, when they produce, starting from their initial states and for each sequence of inputs, the same sequence of outputs.

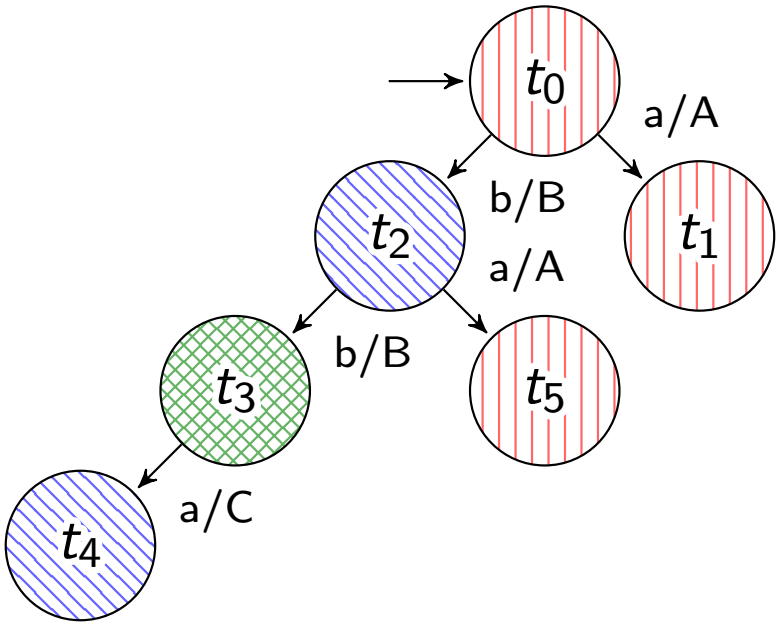
Learning Game

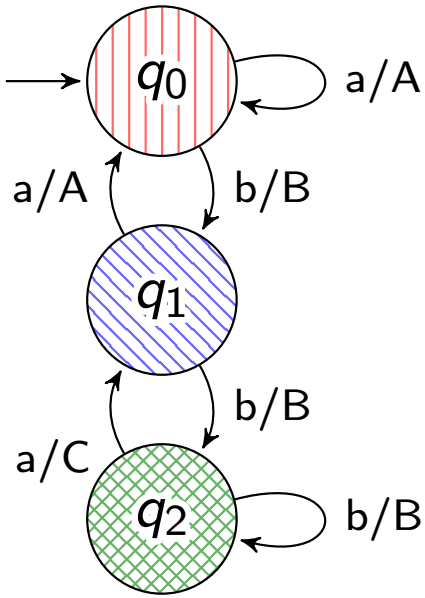
There are two players. The teacher has a hidden Mealy machine $\mathcal{M}$ and the learner has to infer $\mathcal{M}$ by asking two types of queries:

1. Output Query: for an input sequence $\sigma \in I^*$ provided by the learner, the teacher replies with the corresponding output sequence (of type O^*) produced by $\mathcal{M}$.
2. Equivalence Query: for a Mealy machine $\mathcal{H}$ (hypothesis) provided by the learner the teacher replies whether $\mathcal{H}$ is equivalent to $\mathcal{M}$, and if not, it provides a counterexample: a sequence $\sigma \in I^*$ for which $\mathcal{H}$ and $\mathcal{M}$ produce different outputs.

Observation Trees

$L^\#$ stores information from queries in an observation tree.





Definition (Observation Tree)

An observation tree for Mealy machine $\mathcal{M}$ is a tree-shaped, partial Mealy machine $\mathcal{T}$ s.t. there is a functional simulation $f : \mathcal{T} \to \mathcal{M}$.

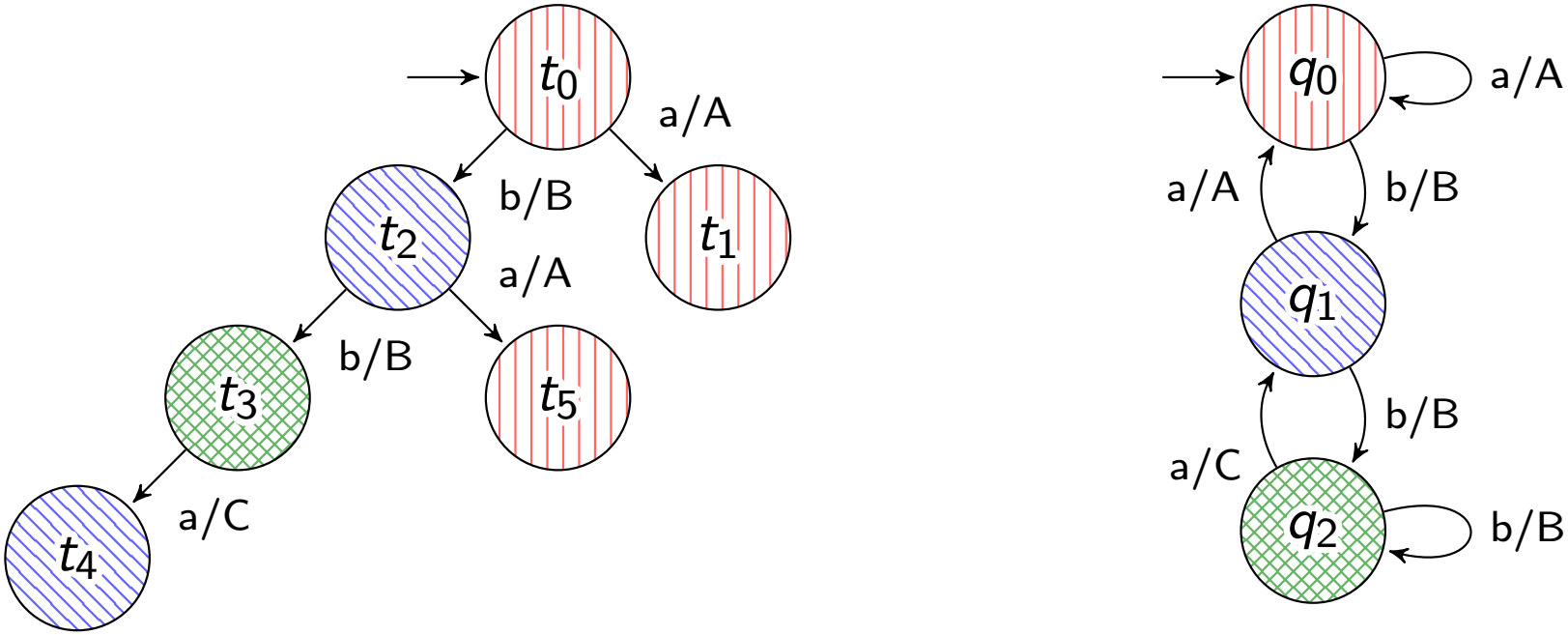
Apartness

In constructive mathematics, an apartness relation is a constructive form of inequality, often taken to be more basic than equality. Two real numbers are said to be apart if there exists (one can construct) a rational number between them.
Notion proposed by Brouwer (1919)(örtlich verschieden, Entfernung).
Heyting (1927) gave an axiomatization:

$$\begin{aligned} &\neg(x\#x) \\ x\#y &\Rightarrow y\#x \\ x\#y &\Rightarrow (x\#z)\vee(y\#z) \end{aligned}$$

Geuvers & Jacobs (2021) study apartness in the context of labeled transition systems and bisimulations.

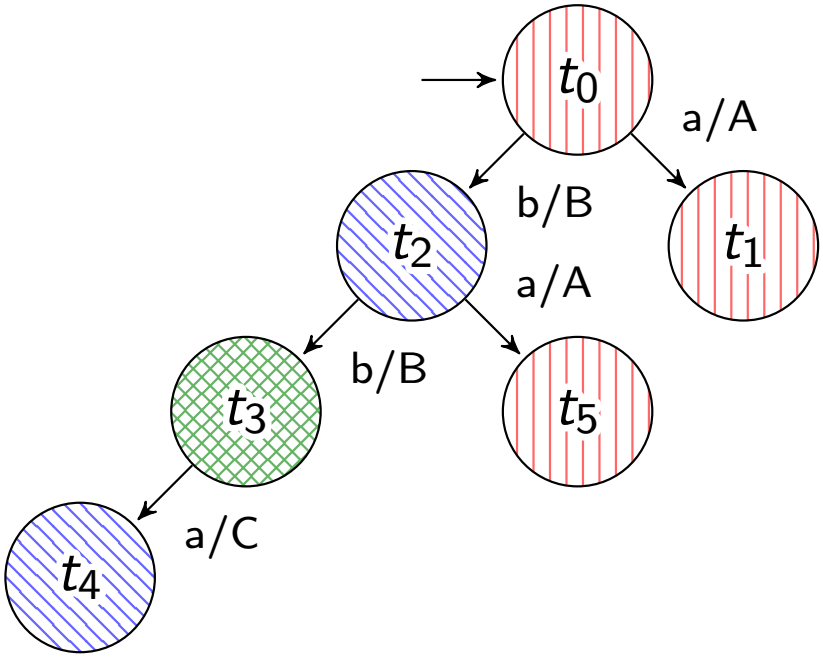
Apartness

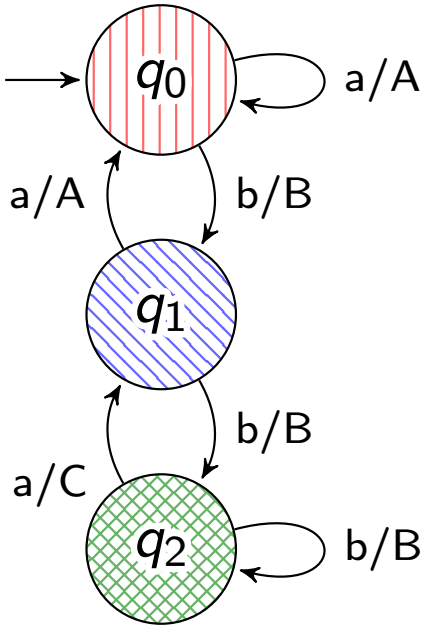


Definition (Apartness)

Two states t, u in an observation tree are apart, $t\#u$, if they both enable an input sequence $\sigma \in I^*$ for which the outputs are different.

Apartness



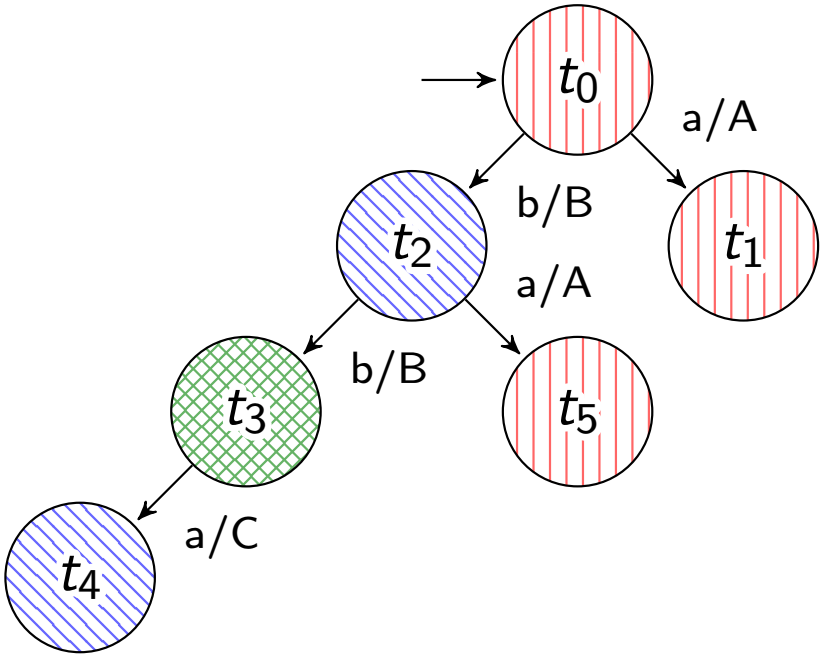


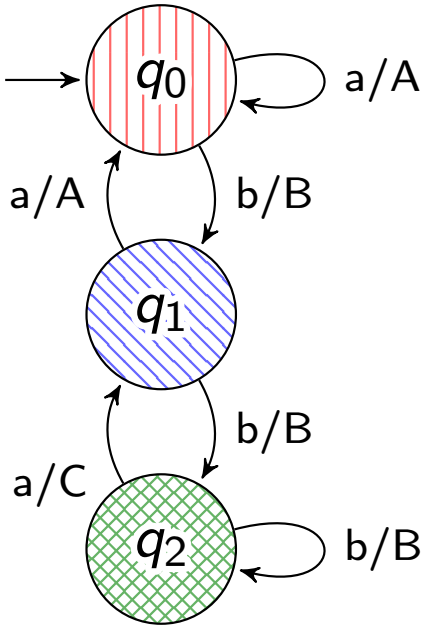
Definition (Apartness)

Two states t, u in an observation tree are apart, $t\#u$, if they both enable an input sequence $\sigma \in I^*$ for which the outputs are different.

$a \vdash t_3\#t_0$

Apartness



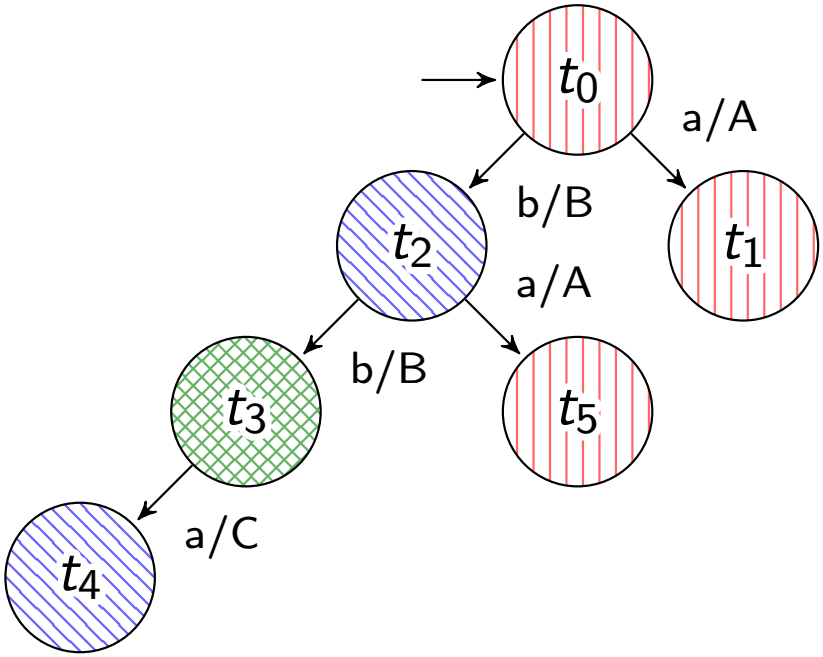


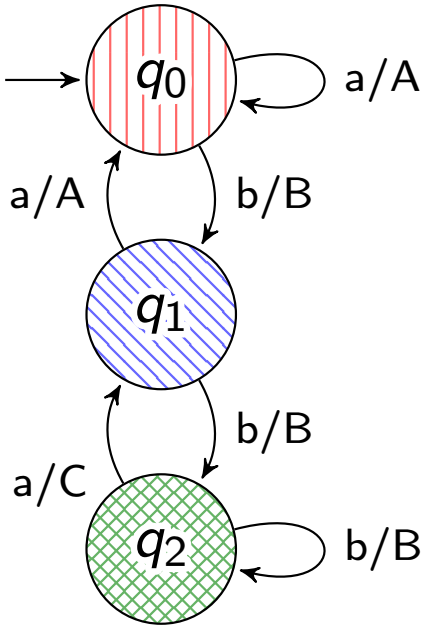
Definition (Apartness)

Two states t, u in an observation tree are apart, $t\#u$, if they both enable an input sequence $\sigma \in I^*$ for which the outputs are different.

$a \vdash t_3\#t_0$ $\quad$ $a \vdash t_3\#t_2$

Apartness





Definition (Apartness)

Two states t, u in an observation tree are apart, $t\#u$, if they both enable an input sequence $\sigma \in I^*$ for which the outputs are different.

$a \vdash t_3\#t_0$ $\quad a \vdash t_3\#t_2$ $\quad ba \vdash t_2\#t_0$

Apartness

Whenever states are apart in observation tree $\mathcal{T}$, the learner knows these are distinct states in the hidden Mealy machine $\mathcal{M}$:

Lemma

For a functional simulation $f : \mathcal{T} \to \mathcal{M}$,

$$q \# p \text{ in } \mathcal{T} \quad \Longrightarrow \quad f(q) \not\approx f(p) \text{ in } \mathcal{M}$$

Apartness

Whenever states are apart in observation tree $\mathcal{T}$, the learner knows these are distinct states in the hidden Mealy machine $\mathcal{M}$:

Lemma

For a functional simulation $f : \mathcal{T} \to \mathcal{M}$,

$$q \# p \text{ in } \mathcal{T} \quad \Longrightarrow \quad f(q) \not\approx f(p) \text{ in } \mathcal{M}$$

Relation $\#$ satisfies the following properties:

Lemma

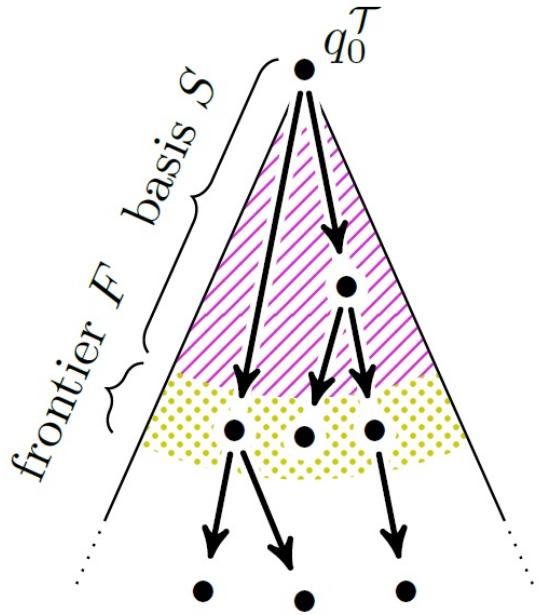
Relation $\#$ is irreflexive, symmetric and weakly co-transitive:

$$\sigma \vdash r \# r' \ \wedge \ q \text{ enables inputs } \sigma \quad \Longrightarrow \quad r \# q \ \vee \ r' \# q$$

Basis and Frontier

Analogous to the shortest path algorithm of Dijkstra (1959), $L^\#$ subdivides the states of the observation tree into three sets:

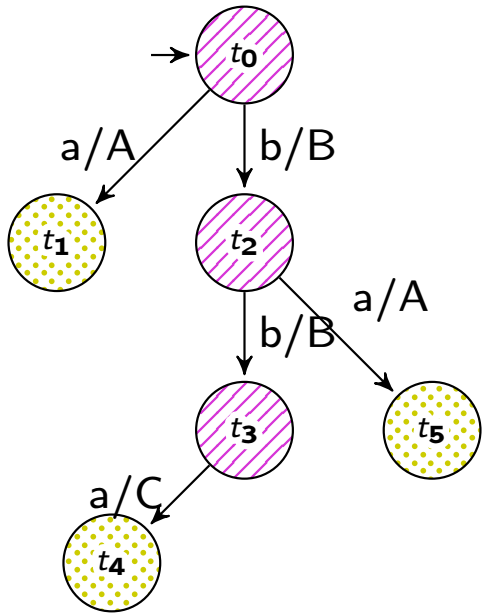
1. basis S: these states form a subtree of $\mathcal{T}$, and are pairwise apart
2. frontier F: the immediate non-basis successors of basis states; the next node to be added to S will be selected from F
3. the remaining states



Definition

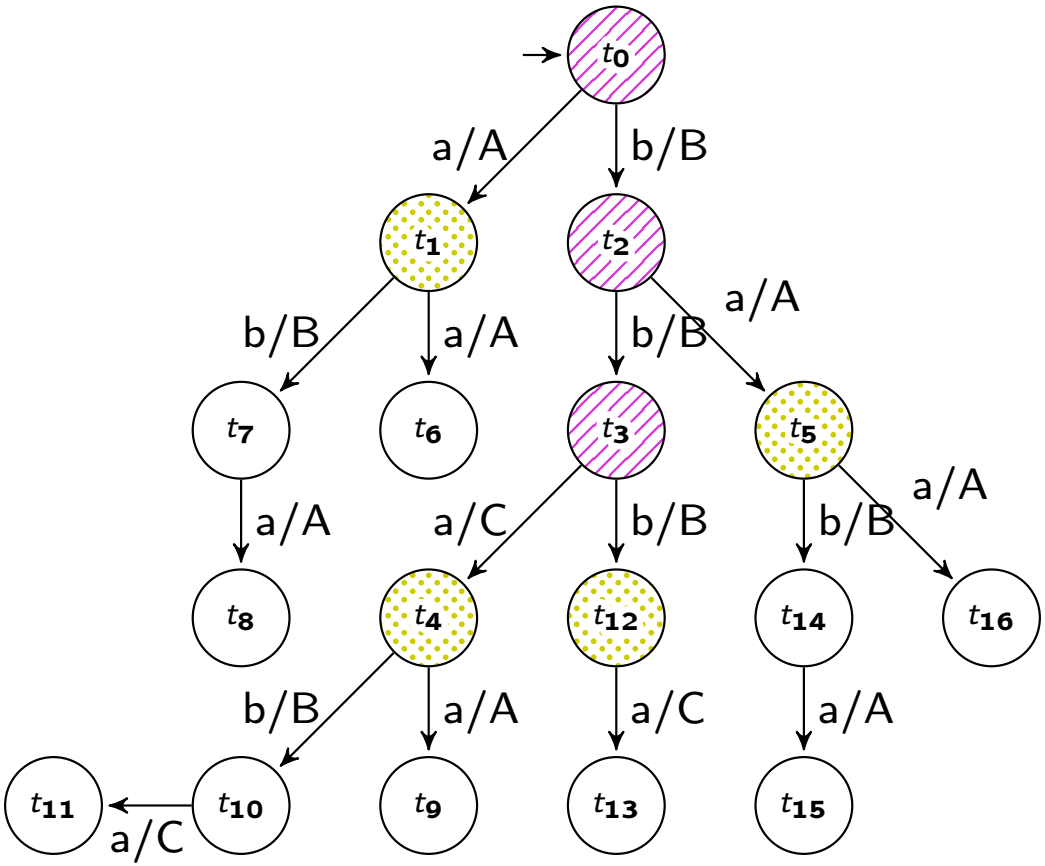
A frontier state in F is isolated if it is apart from all states in S and it is identified if it is apart from all states in S except one.

We use weak co-transitivity to identify frontier states!



Definition

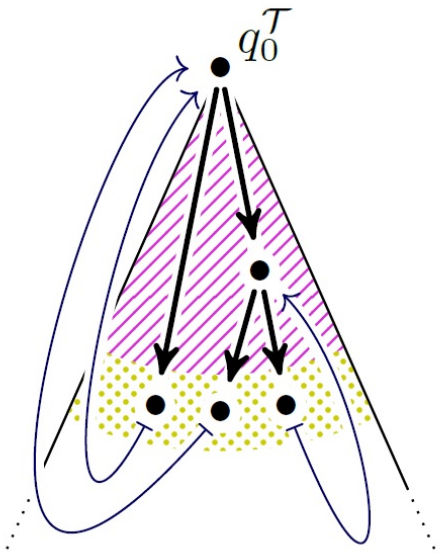
A frontier state in F is isolated if it is apart from all states in S and it is identified if it is apart from all states in S except one.

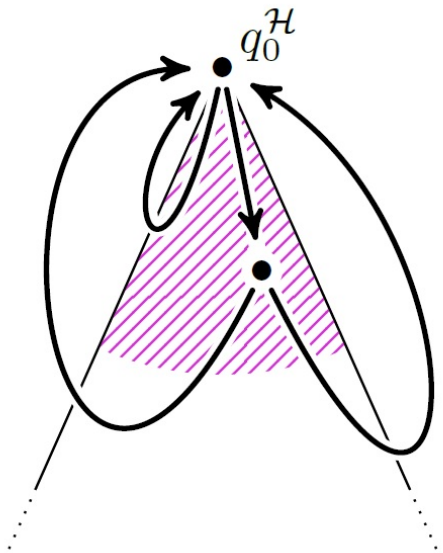


Hypothesis Construction

Definition

Let $\mathcal{T}$ be an observation tree with basis S and frontier F.
A hypothesis is a complete Mealy machine $\mathcal{H}$ with S as its set of states such that $q \xrightarrow{i/o'} p'$ in $\mathcal{H}$ $(q \in S)$ and $q \xrightarrow{i/o} p$ in $\mathcal{T}$ imply $o = o'$ and $\neg(p \# p')$ (in $\mathcal{T}$).





Basic $L^\#$ Algorithm

Algorithm 1 Overall $L^\#$ algorithm

procedure LSharp

do q isolated, for some $q \in F \rightarrow$ ▷ Rule (R1)
  $S \leftarrow S \cup \{q\}$

[] $\delta^{\mathcal{T}}(q, i)\uparrow$, for some $q \in S, i \in I \rightarrow$ ▷ Rule (R2)
  OutputQuery(access(q) i)

[] $\neg(q \# r)$, $\neg(q \# r')$, for some $q \in F$, $r, r' \in S$, $r \neq r' \rightarrow$ ▷ Rule (R3)
  $\sigma \leftarrow$ witness of $r \# r'$
  OutputQuery(access(q) σ)

[] F has no isolated states and basis S is complete $\rightarrow$ ▷ Rule (R4)
  $\mathcal{H} \leftarrow$ BuildHypothesis
  $(b, \sigma) \leftarrow$ CheckConsistency($\mathcal{H}$)
  if $b =$ yes **then**
    $(b, \rho) \leftarrow$ EquivQuery($\mathcal{H}$)
    if $b =$ yes **then: return** $\mathcal{H}$
    else: $\sigma \leftarrow$ shortest prefix of ρ such that $\delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma) \# \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma)$ (in $\mathcal{T}$)
  end if
  ProcCounterEx($\mathcal{H}, \sigma$)

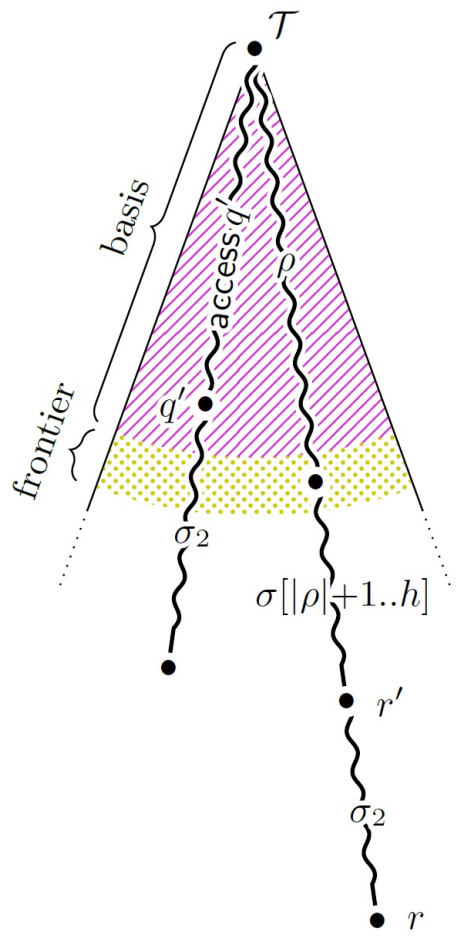
end do

end procedure

Counterexample Processing

Algorithm 3 Processing σ that leads to a conflict, i.e. $\delta^{\mathcal{H}}(q_0, \sigma) \# \delta^{\mathcal{T}}(q_0, \sigma)$

procedure ProcCounterEx($\mathcal{H}$, $\sigma \in I^*$)
  $q \leftarrow \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma)$
  $r \leftarrow \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma)$
  if $r \in S \cup F$ **then**
    return
  else
    $\rho \leftarrow$ unique prefix of σ with $\delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \rho) \in F$
    $h \leftarrow \lfloor \frac{|\rho|+|\sigma|}{2} \rfloor$
    $\sigma_1 \leftarrow \sigma[1..h]$
    $\sigma_2 \leftarrow \sigma[h+1..|\sigma|]$
    $q' \leftarrow \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma_1)$
    $r' \leftarrow \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma_1)$
    $\eta \leftarrow$ witness for $q \# r$
    OutputQuery(access(q') σ_2 η)
    if $q' \# r'$ **then**
      ProcCounterEx ($\mathcal{H}$, σ_1)
    else
      ProcCounterEx ($\mathcal{H}$, access(q') σ_2)
    end if
  end if
end procedure



Correctness of $L^\#$

Whenever the algorithm terminates, the learner has found the correct model. Therefore, correctness amounts to showing termination.

Theorem

Every rule application in $L^\#$ increases the norm $N(\mathcal{T})$ given by:

$$\frac{|S| \cdot (|S|+1)}{2} + |\{(q,i) \in S \times I \mid \delta^{\mathcal{T}}(q,i)\downarrow\}| + |\{(q,q') \in S \times F \mid q \# q'\}|$$

Correctness of $L^\#$

Whenever the algorithm terminates, the learner has found the correct model. Therefore, correctness amounts to showing termination.

Theorem

Every rule application in $L^\#$ increases the norm $N(\mathcal{T})$ given by:

$$\frac{|S| \cdot (|S|+1)}{2} + |\{(q,i) \in S \times I \mid \delta^{\mathcal{T}}(q,i)\downarrow\}| + |\{(q,q') \in S \times F \mid q \# q'\}|$$

Thus the number of rule applications in $L^\#$ is bounded!

Theorem

If $\mathcal{T}$ is an observation tree for $\mathcal{M}$ with n equivalence classes of states and $|I| = k$, then $N(\mathcal{T}) \leq \frac{1}{2} \cdot n \cdot (n+1) + kn + (n-1)(kn+1)$.

Complexity of $L^\#$

Definition

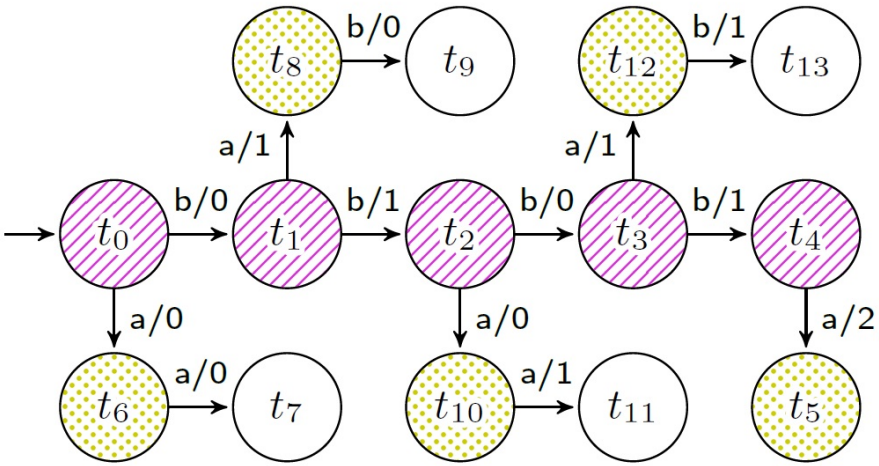
Strategic $L^\#$ is the special case of $L^\#$ where rule (R4) is only applied if none of the other rules is applicable.

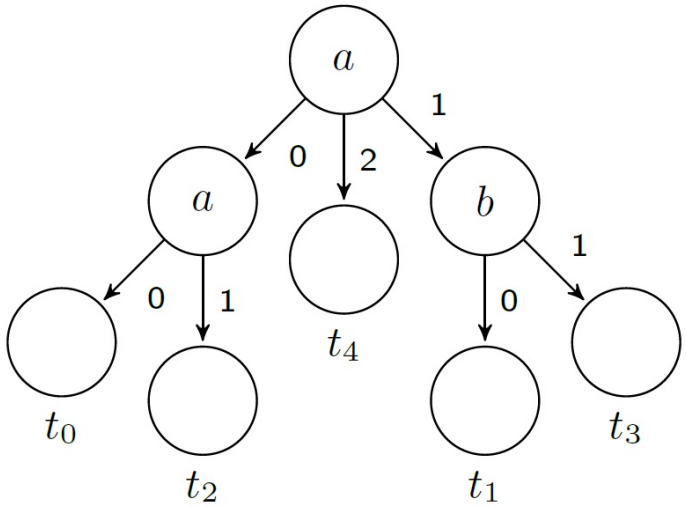
Theorem

Suppose there are k inputs, Mealy machine $\mathcal{M}$ has n states, and the length of the maximal counterexample provided by the teacher is m. Then strategic $L^\#$ learns $\mathcal{M}$ within $\mathcal{O}(kn^2 + n \log m)$ output queries and at most $n - 1$ equivalence queries. At most $\mathcal{O}(kn^2 m + nm \log m)$ input symbols are required to learn $\mathcal{M}$.

Adaptive Distinguishing Sequences

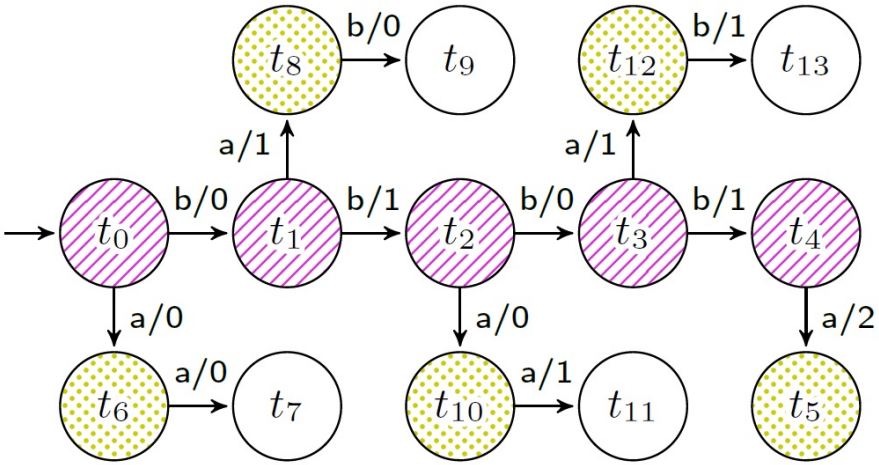
We may speed up identification of frontier states in practice using adaptive distinguishing sequences (ADSs):

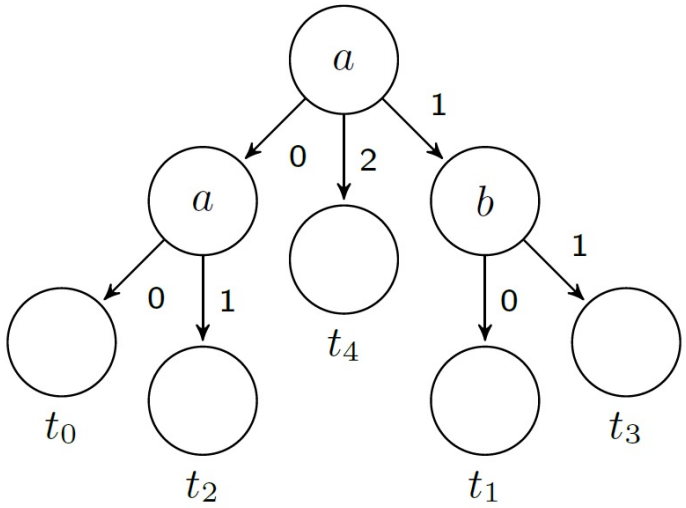




Adaptive Distinguishing Sequences

We may speed up identification of frontier states in practice using adaptive distinguishing sequences (ADSs):





An ADS that maximizes the expected number of new apartness pairs can be efficiently computed directly from the observation tree!

$L^\#$ algorithm with ADS

Algorithm 4 $L^\#$ algorithm with adaptive distinguishing sequences

procedure LSharp ADS

do q isolated, for some $q \in F \rightarrow$ ▷ Rule (R1)
  $S \leftarrow S \cup \{q\}$

▯ $\delta^{\mathcal{T}}(q, i)\uparrow$, for some $q \in S, i \in I \rightarrow$ ▷ Rule (R2)
  OutputQuery(access(q) i Ads(S))

▯ $\neg(q \# r)$, $\neg(q \# r')$, for some $q \in F$, $r, r' \in S$, $r \neq r' \rightarrow$ ▷ Rule (R3)
  OutputQuery(access(q) Ads($\{b \in S \mid \neg(b \# q)\}$))

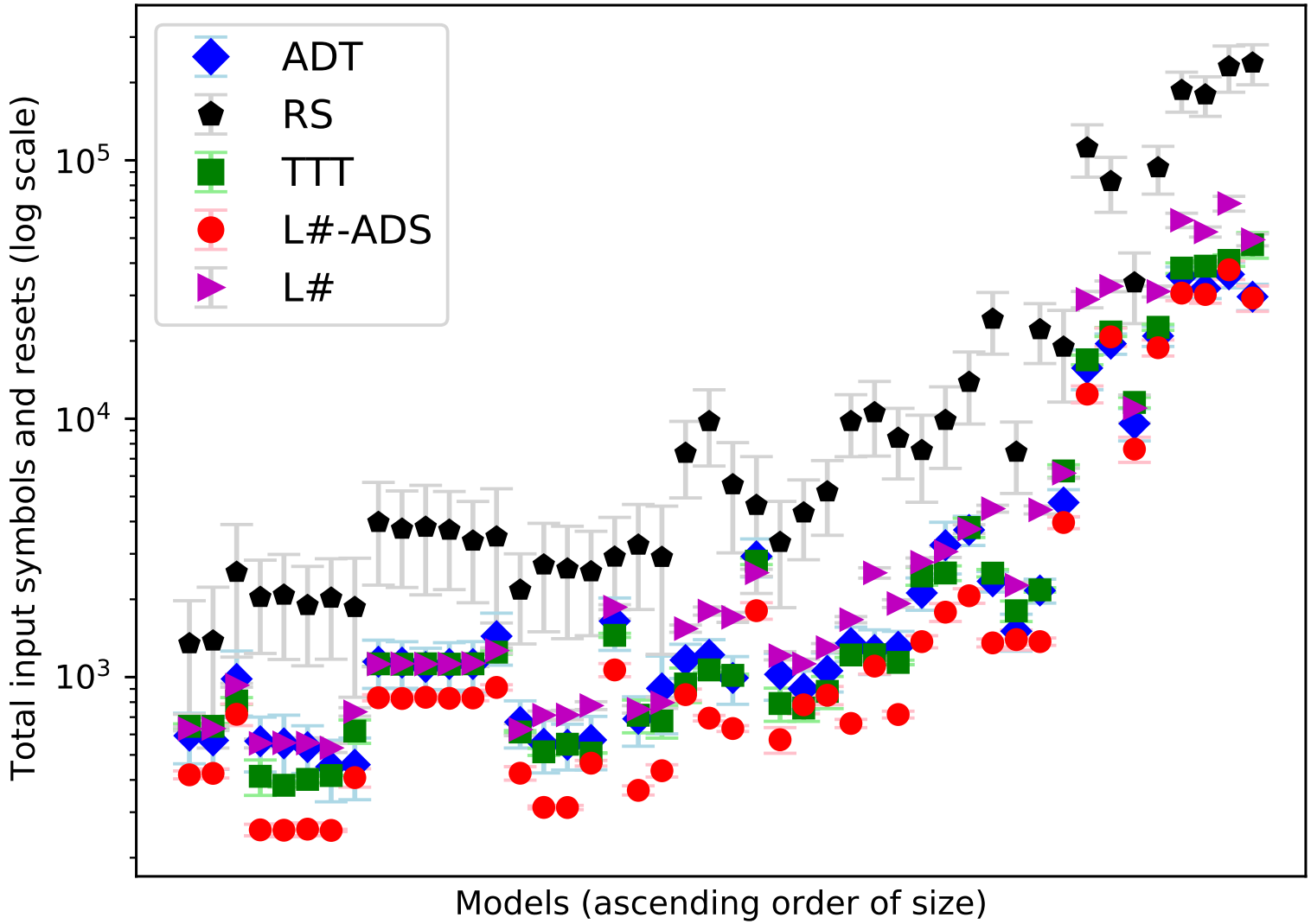
▯ F has no isolated states and basis S is complete $\rightarrow$ ▷ Rule (R4)
  $\mathcal{H} \leftarrow$ BuildHypothesis
  $(b, \sigma) \leftarrow$ CheckConsistency($\mathcal{H}$)
  if $b =$ **yes then**
    $(b, \rho) \leftarrow$ EquivQuery($\mathcal{H}$)
    if $b =$ **yes then: return** $\mathcal{H}$
    else: $\sigma \leftarrow$ shortest prefix of ρ such that $\delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma) \# \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma)$ (in $\mathcal{T}$)
  end if
  ProcCounterEx($\mathcal{H}, \sigma$)

end do
end procedure

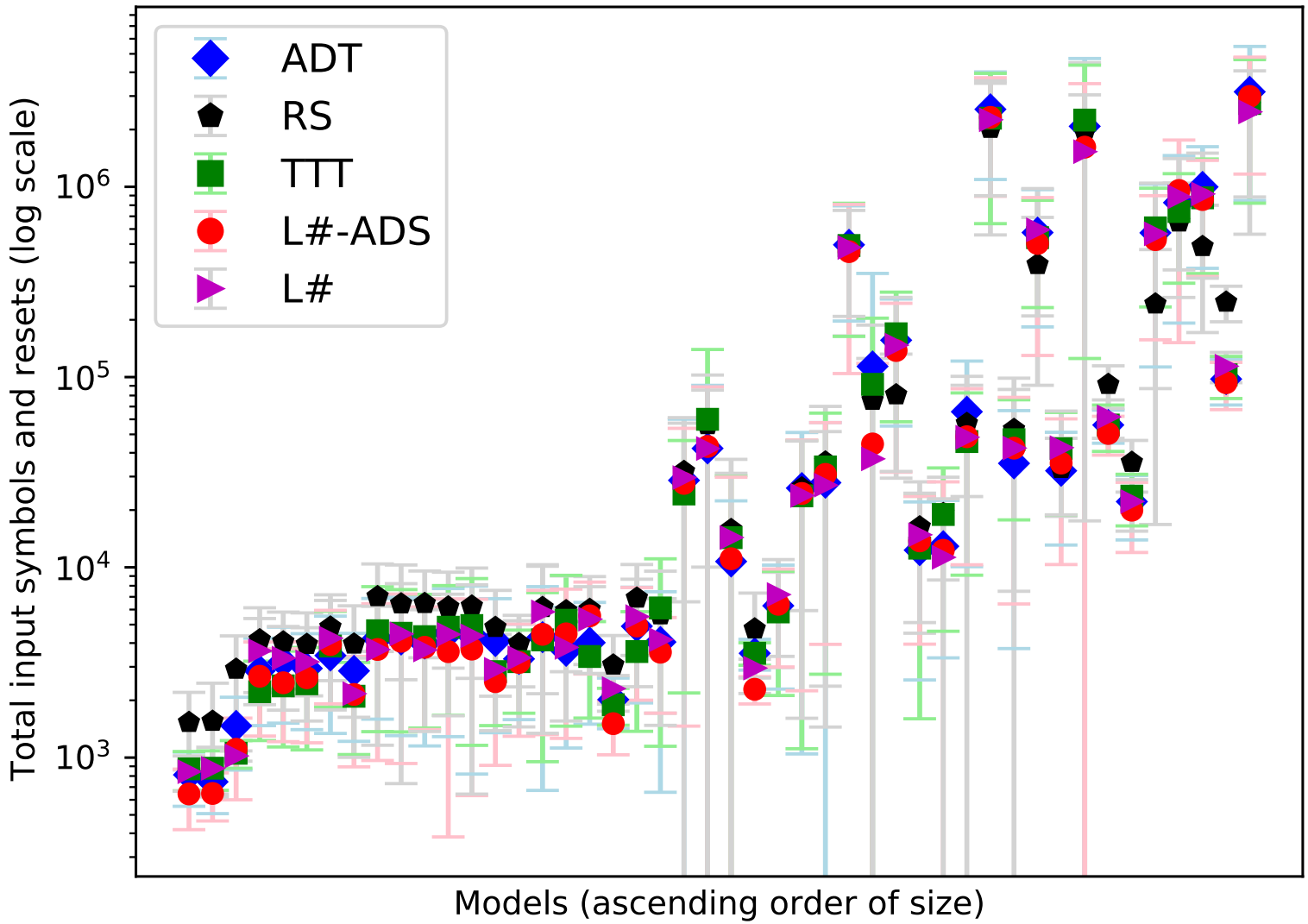
Experimental Evaluation

- We compared a Rust prototype implementation of $L^\#$ with Learnlib implementations of the TTT, ADT and RS (Rivest-Schapire) algorithms
- We implemented equivalence queries using the hybrid ADS conformance testing algorithm
- We used a subset of 46 benchmark models available from the AutomataWiki, which includes models for the SSH, TCP, and TLS protocols, alongside the BankCard models. The largest model in this subset has 66 states and 13 input symbols.

Results


ADT
RS
TTT
L#-ADS
L#
10^5
10^4
10^3
Total input symbols and resets (log scale)
Models (ascending order of size)

Results


ADT
RS
TTT
L#-ADS
L#
Total input symbols and resets (log scale)
10^6
10^5
10^4
10^3
Models (ascending order of size)

Conclusions

- We presented $L^\#$, a new and simple approach to the classical problem of active automata learning, based on four key ideas:
  1. use the observation tree as the main data structure (cf. Soucha & Bogdanov (2020))
  2. focus on apartness rather than equivalence
  3. maintain basis and frontier, as in Dijkstra's algorithm
  4. compute adaptive distinguishing sequences from the tree
- $L^\#$ has the same asymptotic query and symbol complexities as the best existing learning algorithms
- Experiments with a prototype implementation suggest that $L^\#$ is competitive with existing implementations of other algorithms

Future Work

1. Handle big observation trees
2. Optimizations (short separating sequences, better ADSs,..)
3. Integrate active and passive learning
4. Integrate learning and testing
5. Extend $L^\#$ to richer frameworks such as register automata, symbolic automata and weighted automata

Questions?

For details about $L^\#$ see:
Frits Vaandrager, Bharat Garhewal, Jurriaan Rot, and Thorsten Wißmann. A New Approach for Active Automata Learning Based on Apartness. CoRR arXiv:2107.05419, V3, October 2021.

For general intro to active automata learning see:
Frits Vaandrager. Model learning. Communications of the ACM 60(2): 86-95. February 2017.

Consistency Checking

Via a breadth-first search over the product $\mathcal{H} \times \mathcal{T}$ we may check if hypothesis is consistent with observation tree:

Algorithm 2 Check if hypothesis $\mathcal{H}$ is consistent with observation tree $\mathcal{T}$

procedure CHECKCONSISTENCY($\mathcal{H}$)
  $Q \leftarrow$ new *queue* $\subseteq S \times S$
  $enqueue(Q, (q_0^{\mathcal{T}}, q_0^{\mathcal{H}}))$
  while $(q, r) \leftarrow dequeue(Q)$
    if $q \# r$ **then: return** no: access(q)
    for all $q \xrightarrow{i/o} p$ in $\mathcal{T}$ **do**
      $enqueue(Q, (p, \delta^{\mathcal{H}}(r, i)))$
    end for
  end while
  return yes
end procedure

Computing Adaptive Distinguishing Sequences

We recursively define function E, which sends a set $U \subseteq Q^{\mathcal{T}}$ of states to the maximal expected number of apartness pairs (in the absence of unexpected outputs):

$$E(U) = \max_{i \in inp(U)} \left(\sum_{o \in O} \frac{|U \xrightarrow{i/o}| \cdot (|U \xrightarrow{i}| - |U \xrightarrow{i/o}| + E(U \xrightarrow{i/o}))}{|U \xrightarrow{i}|} \right)$$

where

$$\begin{aligned} inp(U) &:= \{ i \in I \mid \exists q \in U : \delta^{\mathcal{T}}(q, i)\downarrow \} \\ U \xrightarrow{i} &:= \{ q \in U \mid \delta^{\mathcal{T}}(q, i)\downarrow \} \\ U \xrightarrow{i/o} &:= \{ q' \in Q^{\mathcal{T}} \mid \exists q \in U : q \xrightarrow{i/o} q' \} \end{aligned}$$

Then $ADS(U)$ is the decision tree constructed recursively by choosing the input i that witnesses the maximum value of $E(U)$.