Model Learning

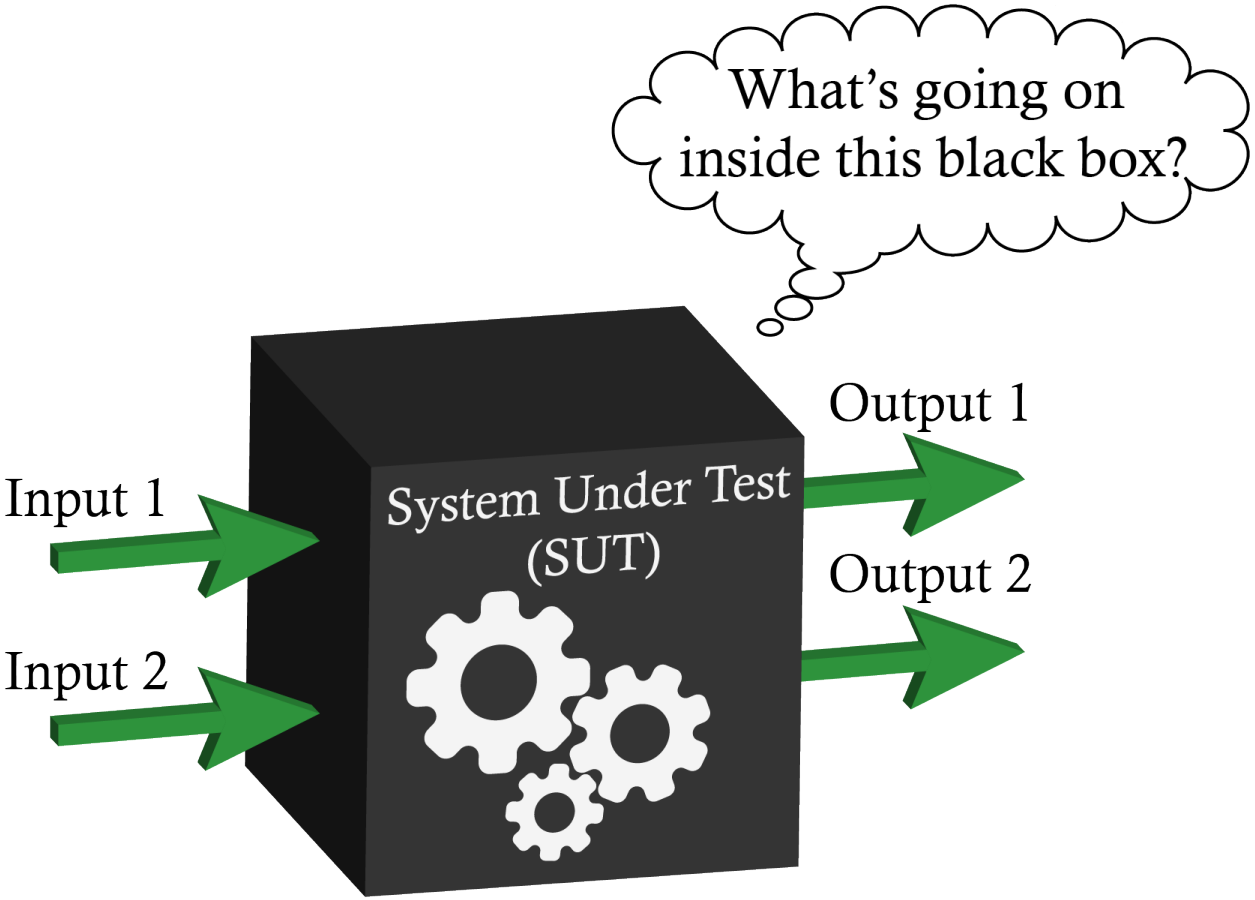
Frits Vaandrager

Radboud University Nijmegen

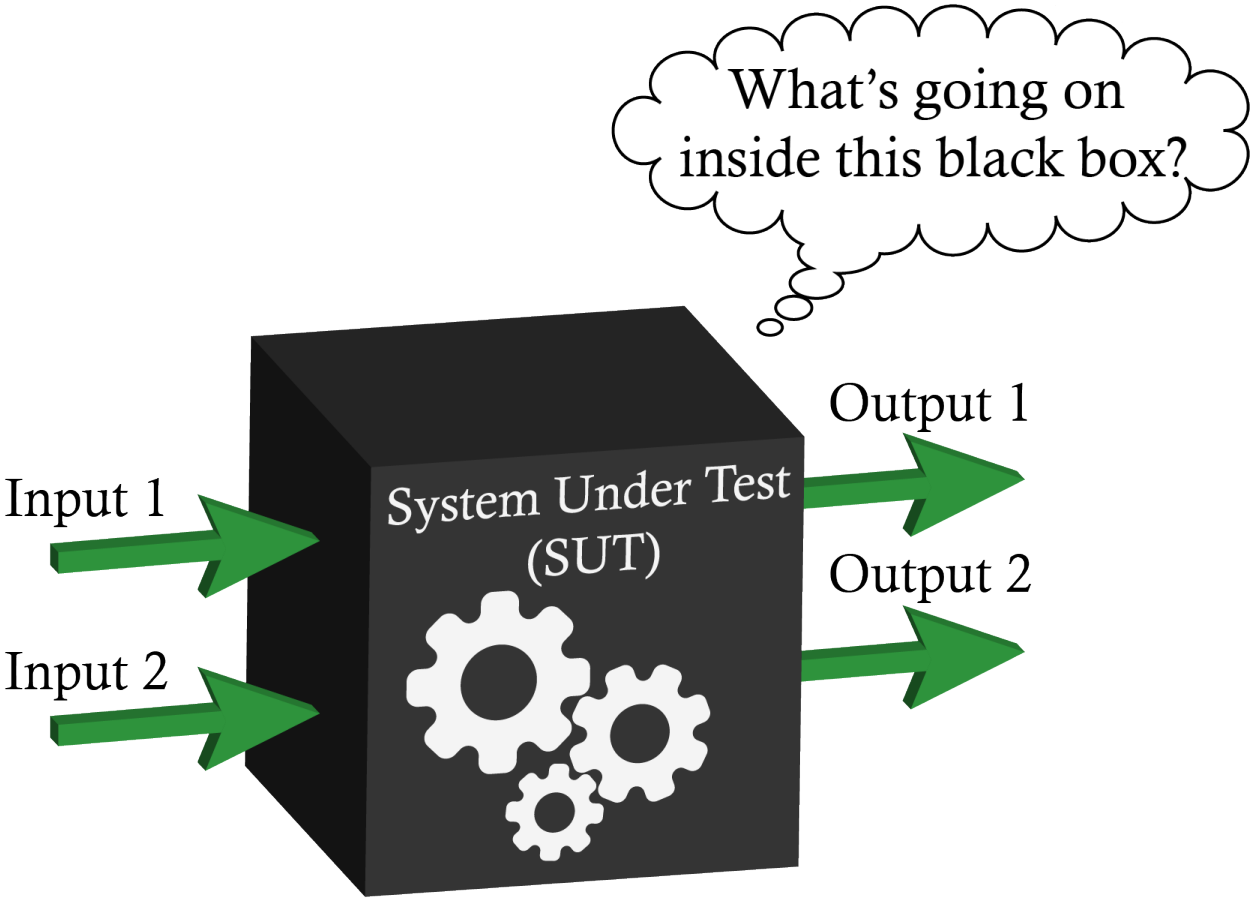
Testing Techniques Course
December 9, 2022

Outline

Research Question



Research Question

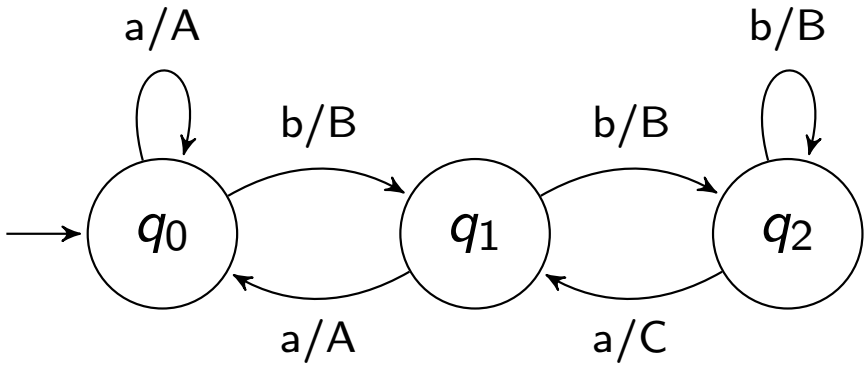


We assume behavior System Under Test (SUT) can be described by a Mealy machine.

Mealy Machines

We fix a finite set I of inputs and a set O of outputs.
The diagram below shows a simple Mealy machine $\mathcal{M}$ with:

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



Equivalence of Mealy Machines

Definition

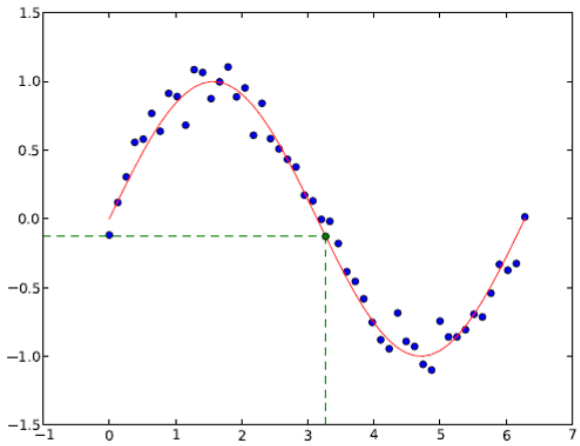
The semantic functional $\lambda^+ : I^+ \to O$ for Mealy machine $\mathcal{M}$ returns the last output of $\mathcal{M}$ in response to any nonempty input sequence.

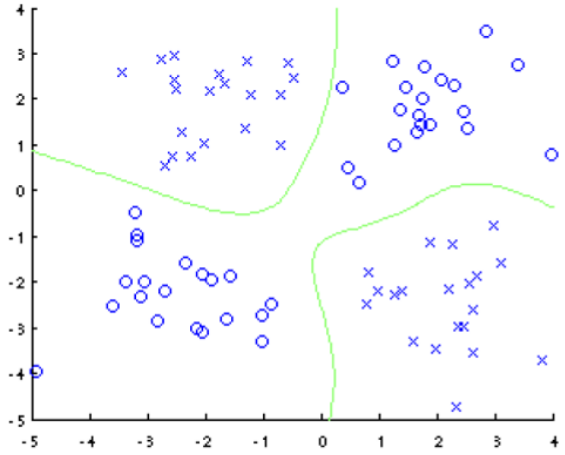
Definition

Two Mealy machines $\mathcal{M}$ and $\mathcal{N}$ are equivalent if and only if they have the same semantic functional.

Machine Learning in General

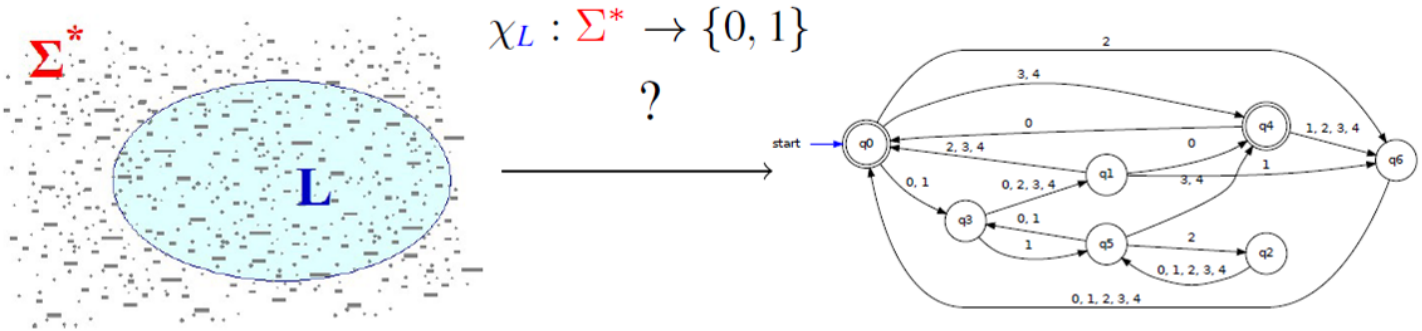
- Given a sample $M = \{(x, y) \mid x \in X, y \in Y\}$
- Find $f : X \to Y$ such that $f(x) = y, \forall (x, y) \in M$
- Predict $f(x)$ for all $x \in X$





Learning Regular Languages

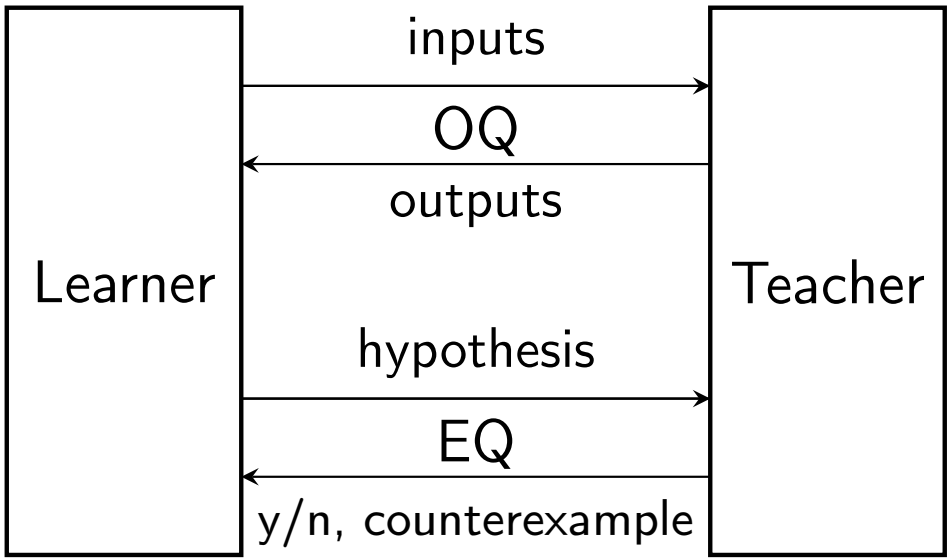
Let Σ be an alphabet and let $L \subseteq \Sigma^*$ be a regular language (*the target language*)



- 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

Minimally Adequate Teacher

Learning framework proposed by Angluin (1987):



Learner asks output queries (OQ) and equivalence queries (EQ)

Learning Game

There are two players: a teacher has a hidden Mealy machine $\mathcal{M}$ and a 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, provides a counterexample: a sequence $\sigma \in I^*$ for which $\mathcal{H}$ and $\mathcal{M}$ produce different outputs.

The Myhill-Nerode Theorem

Definition (Nerode equivalence)

Let $f : I^+ \to O$. Words $u, u' \in I^*$ are equivalent with respect to $\sim_f$ if, for all continuations $v \in I^+$, the concatenated words uv and $u'v$ are mapped to the same output by f:

$$u \sim_f u' \quad \textit{iff} \quad \forall v \in I^+ : f(uv) = f(u'v)$$

Theorem (Myhill-Nerode, 1958)

Function $f : I^+ \to O$ is the semantic functional of some Mealy machine $\mathcal{M}$ iff $\sim_f$ has finitely equivalence classes.

The Myhill-Nerode Theorem

Definition (Nerode equivalence)

Let $f : I^+ \to O$. Words $u, u' \in I^*$ are equivalent with respect to $\sim_f$ if, for all continuations $v \in I^+$, the concatenated words uv and $u'v$ are mapped to the same output by f:

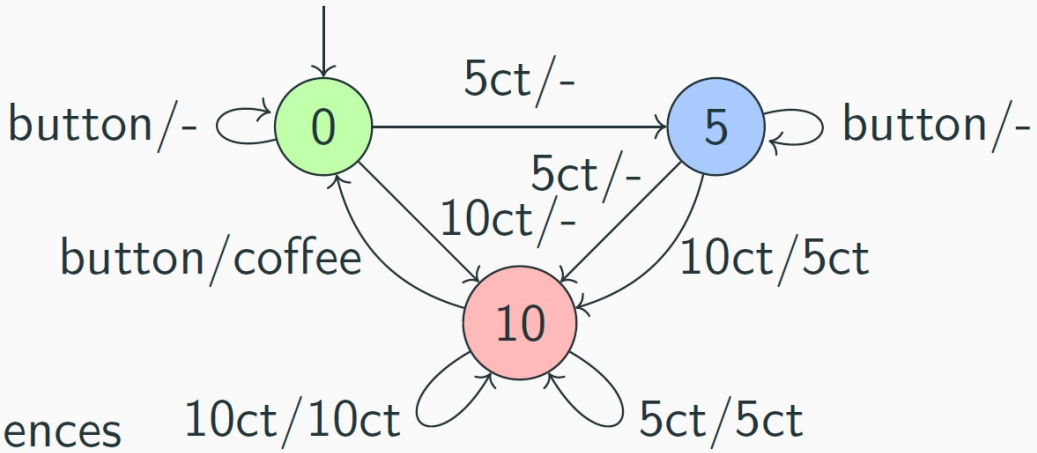
$$u \sim_f u' \quad \textit{iff} \quad \forall v \in I^+ : f(uv) = f(u'v)$$

Theorem (Myhill-Nerode, 1958)

Function $f : I^+ \to O$ is the semantic functional of some Mealy machine $\mathcal{M}$ iff $\sim_f$ has finitely equivalence classes.

Angluin's L^* algorithm approximates the Nerode equivalence: two prefixes are deemed equivalent unless there is a distinguishing suffix

Observation Table



- Rows are states
- Prefixes are access sequences
- Cells are λ^+

P↓/S→	5	10	b	5 5	5 10	5 b	10 5	10 10	10 b	...
ϵ	-	-	-	-	5	-	5	10	c	...
5	-	5	-	5	10	c	5	10	c	...
10	5	10	c	5	10	c	5	10	c	...
b	-	-	-	-	5	-	5	10	c	...
5 5	5	10	c	5	10	c	5	10	c	...
5 10	5	10	c	5	10	c	5	10	c	...
5 b	-	5	-	5	10	c	5	10	c	...
10 5	5	10	c	5	10	c	5	10	c	...
10 10	5	10	c	5	10	c	5	10	c	...
10 b	-	-	-	-	5	-	5	10	c	...

Example

Inputs: $I = \{a, b\}$
SUT $\mathcal{M}$ produces 1 if number of a's and b's are both even, otherwise 0.

For instance, $\lambda^+(\mathcal{M}, abaab) = 0$, $\lambda^+(\mathcal{M}, abaaba) = 1$

Example

Inputs: $I = \{a, b\}$
SUT $\mathcal{M}$ produces 1 if number of a's and b's are both even, otherwise 0.

Start: prefix ϵ and suffixes I

P↓/S→	a	b	"colour"
ϵ			

Example

Inputs: $I = \{a, b\}$
SUT $\mathcal{M}$ produces 1 if number of a's and b's are both even, otherwise 0.

Fill cells by observing output (membership queries) after *prefix·suffix*

P↓/S→	a	b	"colour"
ϵ	0	0	

Example

Inputs: $I = \{a, b\}$
SUT $\mathcal{M}$ produces 1 if number of a's and b's are both even, otherwise 0.

Mark the new state and give it a new colour

P↓/S→	a	b	"colour"
→ ϵ	0	0	

Example

Inputs: $I = \{a, b\}$
SUT $\mathcal{M}$ produces 1 if number of a's and b's are both even, otherwise 0.

Table not closed: marked prefix ϵ needs extensions a and b

P↓/S→	a	b	"colour"
→ ϵ	0	0	
a			
b			

Example

Inputs: $I = \{a, b\}$
SUT $\mathcal{M}$ produces 1 if number of a's and b's are both even, otherwise 0.

Fill cells

P↓/S→	a	b	"colour"
→ ϵ	0	0	
a	1	0	
b	0	1	

Example

Inputs: $I = \{a, b\}$
SUT $\mathcal{M}$ produces 1 if number of a's and b's are both even, otherwise 0.

New states: mark and give them new colours

P↓/S→	a	b	"colour"
→ ϵ	0	0	
→ a	1	0	
→ b	0	1	

Example

Inputs: $I = \{a, b\}$
SUT $\mathcal{M}$ produces 1 if number of a's and b's are both even, otherwise 0.

Table not closed: marked prefixes a and b need extensions

P↓/S→	a	b	"colour"
→ ϵ	0	0	
→ a	1	0	
→ b	0	1	
aa			
ab			
ba			
bb			

Example

Inputs: $I = \{a, b\}$
SUT $\mathcal{M}$ produces 1 if number of a's and b's are both even, otherwise 0.

Fill cells

P↓/S→	a	b	"colour"
$\rightarrow \epsilon$	0	0	
$\rightarrow$ a	1	0	
$\rightarrow$ b	0	1	
aa	0	0	
ab	0	0	
ba	0	0	
bb	0	0	

Example

Inputs: $I = \{a, b\}$
SUT $\mathcal{M}$ produces 1 if number of a's and b's are both even, otherwise 0.

States are already present: do not mark, and reuse colours

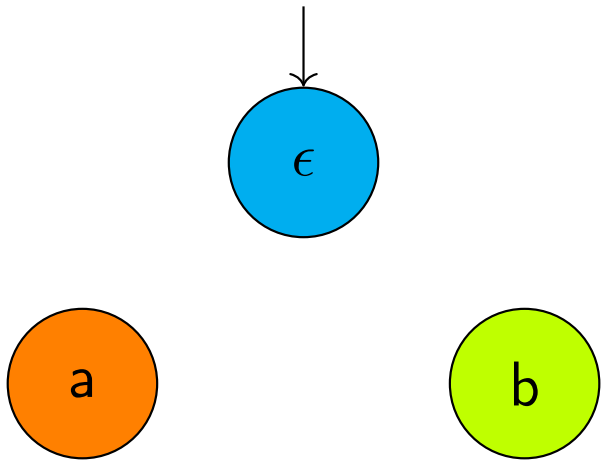
P↓/S→	a	b	"colour"
→ ϵ	0	0	
→ a	1	0	
→ b	0	1	
aa	0	0	
ab	0	0	
ba	0	0	
bb	0	0	

Example

Inputs: $I = \{a, b\}$
SUT $\mathcal{M}$ produces 1 if number of a's and b's are both even, otherwise 0.

Closed table: draw hypothesis!

P↓/S→	a	b	"colour"
→ ϵ	0	0	
→ a	1	0	
→ b	0	1	
aa	0	0	
ab	0	0	
ba	0	0	
bb	0	0	

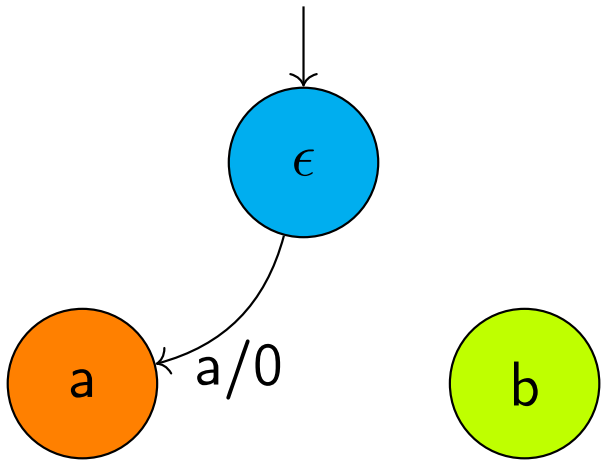


Example

Inputs: $I = \{a, b\}$
SUT $\mathcal{M}$ produces 1 if number of a's and b's are both even, otherwise 0.

Cell (ϵ, a) contains output 0, and row $\epsilon \cdot a$ has colour (state) a

P↓/S→	a	b	"colour"
→ ϵ	0	0	
→ a	1	0	
→ b	0	1	
aa	0	0	
ab	0	0	
ba	0	0	
bb	0	0	

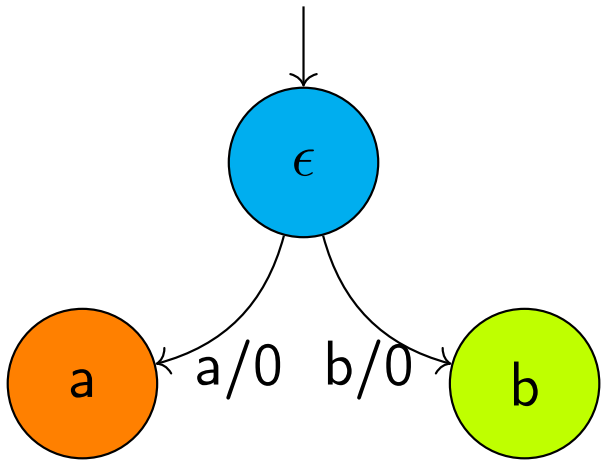


Example

Inputs: $I = \{a, b\}$
SUT $\mathcal{M}$ produces 1 if number of a's and b's are both even, otherwise 0.

Cell (ϵ, b) contains output 0, and row $\epsilon \cdot b$ has colour (state) b

P↓/S→	a	b	"colour"
→ ϵ	0	0	
→ a	1	0	
→ b	0	1	
aa	0	0	
ab	0	0	
ba	0	0	
bb	0	0	

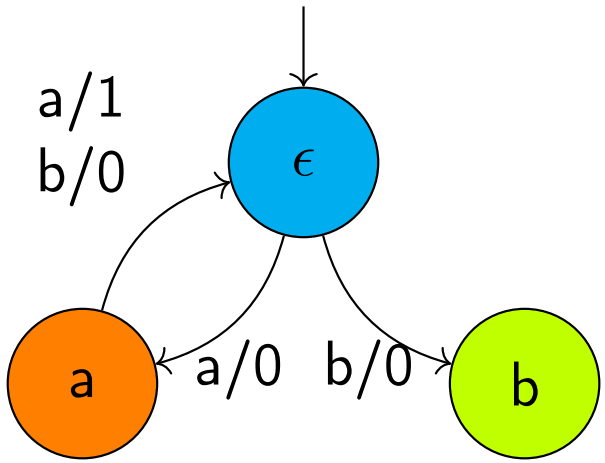


Example

Inputs: $I = \{a, b\}$
SUT $\mathcal{M}$ produces 1 if number of a's and b's are both even, otherwise 0.

Cell (a, a) contains output 1, and row $a \cdot a$ has colour (state) ϵ

P↓/S→	a	b	"colour"
→ ϵ	0	0	
→ a	1	0	
→ b	0	1	
aa	0	0	
ab	0	0	
ba	0	0	
bb	0	0	

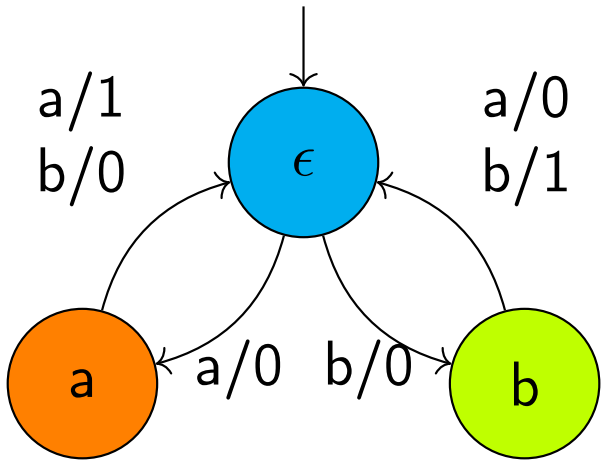


Example

Inputs: $I = \{a, b\}$
SUT $\mathcal{M}$ produces 1 if number of a's and b's are both even, otherwise 0.

Cell (b, a) contains output 0, and row $b \cdot a$ has colour (state) ϵ

P↓/S→	a	b	"colour"
→ ϵ	0	0	
→ a	1	0	
→ b	0	1	
aa	0	0	
ab	0	0	
ba	0	0	
bb	0	0	



Counterexample handling

Assume a counterexample π

Apparently $\pi = \sigma\rho$, where σ goes to an unknown state and ρ shows it to be different from the (wrong) hypothesis state

Add prefixes of π until you find a new state

Counterexample handling

SUT produces a 1 if the number of a's and b's so far are both even, otherwise produces a 0.

Now, consider counterexample abab: add prefixes

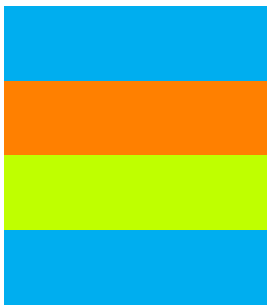
	a	b	"colour"
$\rightarrow \epsilon$	0	0	
$\rightarrow$ a	1	0	
$\rightarrow$ b	0	1	
ab	0	0	
aba			
aa	0	0	
ba	0	0	
bb	0	0	

Counterexample handling

SUT produces a 1 if the number of a's and b's so far are both even, otherwise produces a 0.

Prefixes a and ab are already there, so add prefix aba

	a	b	"colour"
$\rightarrow \epsilon$	0	0	
$\rightarrow$ a	1	0	
$\rightarrow$ b	0	1	
ab	0	0	
aba			
aa	0	0	
ba	0	0	
bb	0	0	

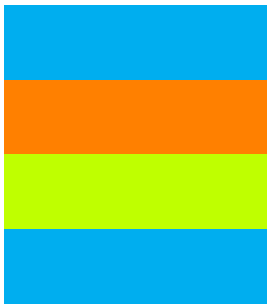


Counterexample handling

SUT produces a 1 if the number of a's and b's so far are both even, otherwise produces a 0.

Fill the table

	a	b	"colour"
$\rightarrow \epsilon$	0	0	
$\rightarrow$ a	1	0	
$\rightarrow$ b	0	1	
ab	0	0	
aba	0		
aa	0	0	
ba	0	0	
bb	0	0	

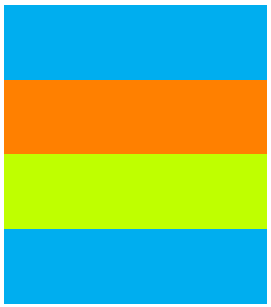


Counterexample handling

SUT produces a 1 if the number of a's and b's so far are both even, otherwise produces a 0.

Fill the table

	a	b	"colour"
$\rightarrow \epsilon$	0	0	
$\rightarrow$ a	1	0	
$\rightarrow$ b	0	1	
ab	0	0	
aba	0	1	
aa	0	0	
ba	0	0	
bb	0	0	



Counterexample handling

SUT produces a 1 if the number of a's and b's so far are both even, otherwise produces a 0.

No new states? What is wrong?

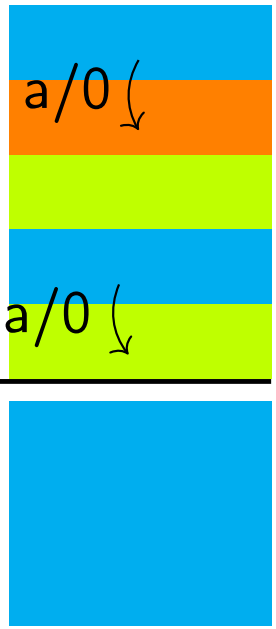
	a	b	"colour"
$\rightarrow \epsilon$	0	0	
$\rightarrow$ a	1	0	
$\rightarrow$ b	0	1	
ab	0	0	
aba	0	1	
aa	0	0	
ba	0	0	
bb	0	0	

Counterexample handling

SUT produces a 1 if the number of a's and b's so far are both even, otherwise produces a 0.

There is a transition $\epsilon \xrightarrow{a/0} a$, but also $\epsilon \xrightarrow{a/0} b$. Inconsistent table!

	a	b
$\rightarrow \epsilon$	0	0
$\rightarrow$ a	1	0
$\rightarrow$ b	0	1
ab	0	0
aba	0	1
aa	0	0
ba	0	0
bb	0	0



Counterexample handling

SUT produces a 1 if the number of a's and b's so far are both even, otherwise produces a 0.

Two a-transition to states which differ on suffix a: add suffix aa

	a	b	aa	"colour"
$\rightarrow \epsilon$	0	0		
$\rightarrow$ a	1	0		a/0
$\rightarrow$ b	0	1		
ab	0	0		
aba	0	1		a/0
aa	0	0		
ba	0	0		
bb	0	0		

Counterexample handling

SUT produces a 1 if the number of a's and b's so far are both even, otherwise produces a 0.

Fill the table

	a	b	aa	"colour"
$\rightarrow \epsilon$	0	0	1	
$\rightarrow$ a	1	0	0	
$\rightarrow$ b	0	1	0	
ab	0	0	0	
aba	0	1	0	
aa	0	0	1	
ba	0	0	0	
bb	0	0	1	

Counterexample handling

SUT produces a 1 if the number of a's and b's so far are both even, otherwise produces a 0.

Mark the newly found state, and give a new color

	a	b	aa	"colour"
$\rightarrow \epsilon$	0	0	1	
$\rightarrow$ a	1	0	0	
$\rightarrow$ b	0	1	0	
$\rightarrow$ ab	0	0	0	
aba	0	1	0	
aa	0	0	1	
ba	0	0	0	
bb	0	0	1	

Counterexample handling

SUT produces a 1 if the number of a's and b's so far are both even, otherwise produces a 0.

Close the table

	a	b	aa	"colour"
$\rightarrow \epsilon$	0	0	1	
$\rightarrow$ a	1	0	0	
$\rightarrow$ b	0	1	0	
$\rightarrow$ ab	0	0	0	
aba	0	1	0	
aa	0	0	1	
ba	0	0	0	
bb	0	0	1	
abb				

Counterexample handling

SUT produces a 1 if the number of a's and b's so far are both even, otherwise produces a 0.

Fill the table

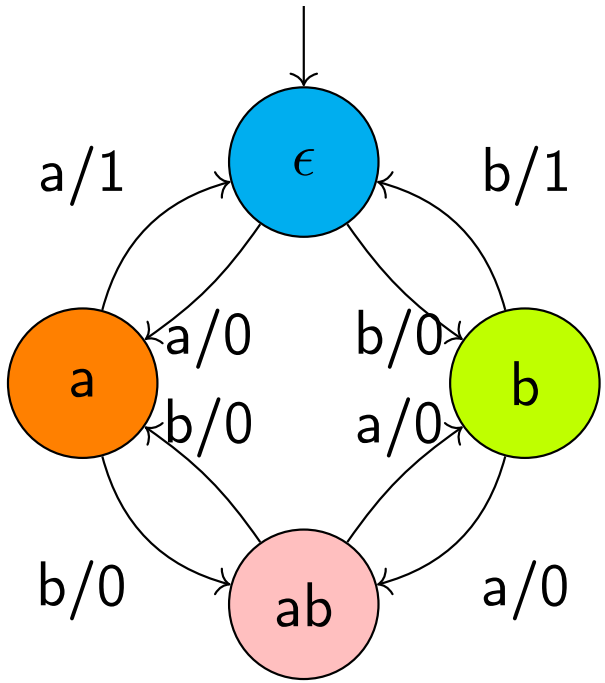
	a	b	aa	"colour"
$\rightarrow \epsilon$	0	0	1	
$\rightarrow$ a	1	0	0	
$\rightarrow$ b	0	1	0	
$\rightarrow$ ab	0	0	0	
aba	0	1	0	
aa	0	0	1	
ba	0	0	0	
bb	0	0	1	
abb	1	0	0	

Counterexample handling

SUT produces a 1 if the number of a's and b's so far are both even, otherwise produces a 0.

Draw a new hypothesis

	a	b	aa	"colour"
$\rightarrow \epsilon$	0	0	1	
$\rightarrow$ a	1	0	0	
$\rightarrow$ b	0	1	0	
$\rightarrow$ ab	0	0	0	
aba	0	1	0	
aa	0	0	1	
ba	0	0	0	
bb	0	0	1	
abb	1	0	0	



Angluin's L^* Algorithm

1. Maintain a set $\mathcal{U}$ of prefixes, initially $\mathcal{U} = \{\epsilon\}$
2. Maintain a set $\mathcal{V}$ of suffixes, initially $\mathcal{V} = I$
3. Maintain an observation table with rows $\mathcal{U} \cup \mathcal{U}I$ and columns $\mathcal{V}$
4. Fill the table using output queries
5. Table is closed when every row from $\mathcal{U}I$ is also a row from $\mathcal{U}$; if table is not closed extend $\mathcal{U}$ and go to step 4
6. Table is consistent if whenever rows $u, v \in \mathcal{U}$ are the same, rows ui and vi are also the same, for all $i \in I$; if table is not consistent extend $\mathcal{V}$ and go to step 4
7. When table is both closed and consistent construct hypothesis and perform equivalence query
8. If reply is "no" add all prefixes of counterexample to $\mathcal{U}$ and go to step 4

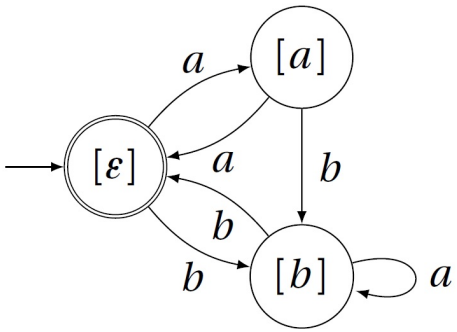
Rivest & Schapire

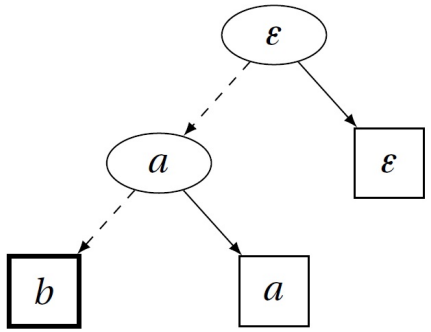
- Optimization of L^* proposed by Rivest & Schapire (1993)
- Rather than adding all prefixes of a counterexample to $\mathcal{U}$, just add a single suffix to $\mathcal{V}$
- Suffix is constructed using a binary search that requires $\mathcal{O}(\log m)$ membership queries, where m is length of maximal counterexample
- Table will always remain consistent!

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

		$\mathcal{V}$: 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
- "our evaluations show that TTT is superior to virtually every other learning algorithm"

ADT Algorithm

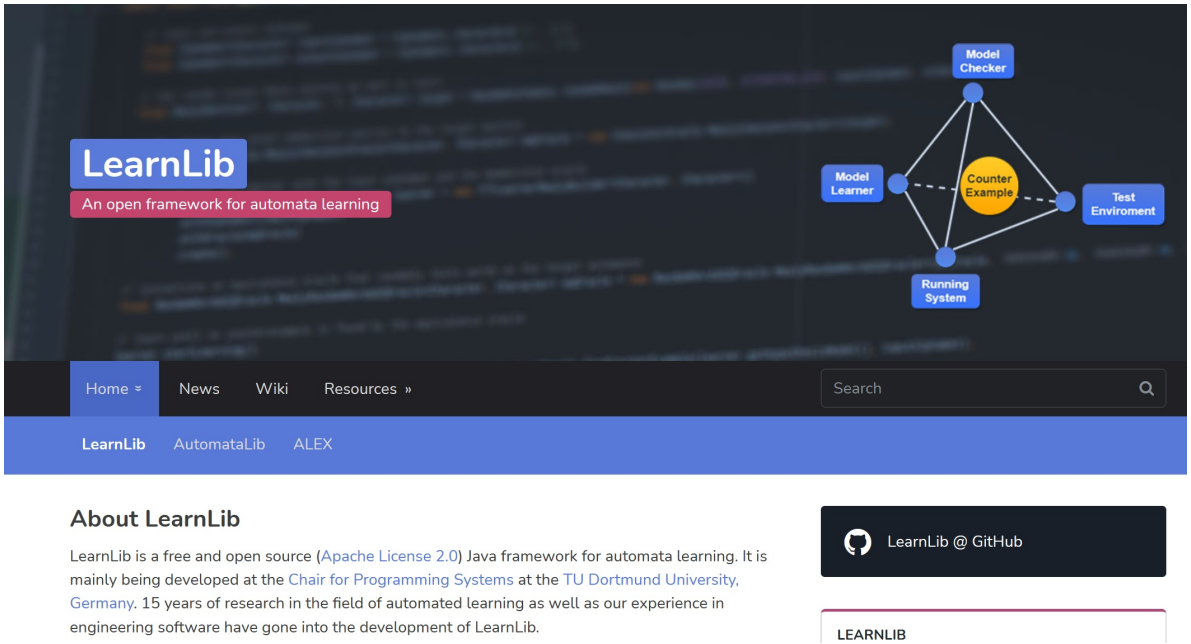
- Proposed by Markus Frohme (2015) in MSc thesis
- Integrates discrimination trees with adaptive distinguishing sequences, queries in which choice of inputs may depend on outputs received in response to previous inputs
- Outperforms other algorithms on many benchmarks

Overview of L^* and its descendants

Algorithm	Query Complexity	Symbol Complexity
L^*	$\mathcal{O}(kn^2m)$	$\mathcal{O}(kn^2m^2)$
Maler/Pnueli	$\mathcal{O}(kn^2m)$	$\mathcal{O}(kn^2m^2)$
Shabhaz	$\mathcal{O}(kn^2m)$	$\mathcal{O}(kn^2m^2)$
Suffix1by1	$\mathcal{O}(kn^2m)$	$\mathcal{O}(kn^2m^2)$
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	$\mathcal{O}(kn^2 + n\log m)$	$\mathcal{O}(kn^2m + nm\log m)$

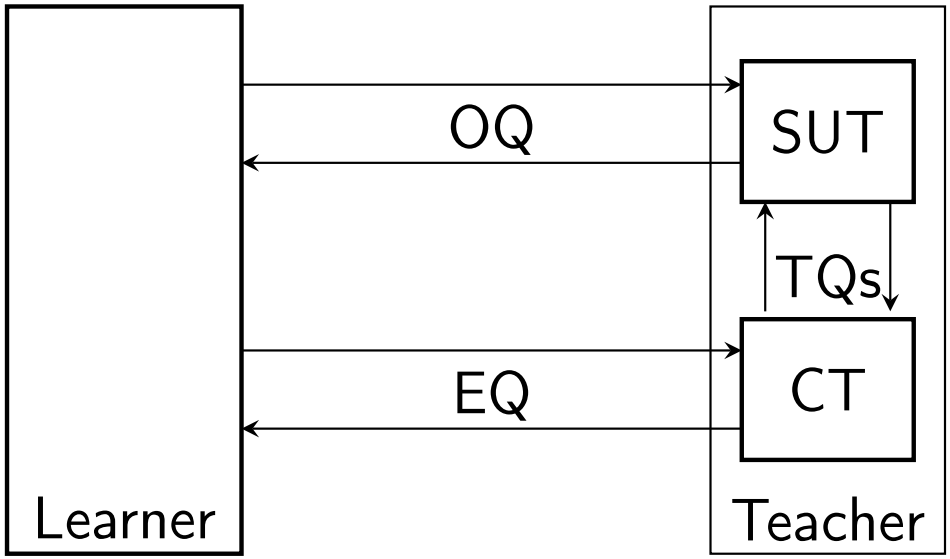
Here n is number of states, k is number of inputs, and m is length of maximal counterexample; table adapted from Isberner (2015).

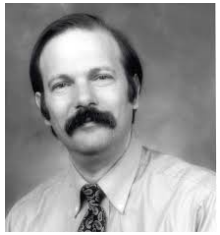
LearnLib



Implements active and passive learning algorithms for DFAs and Mealy machines

Black Box Checking (Peled, Vardi & Yannakakis)

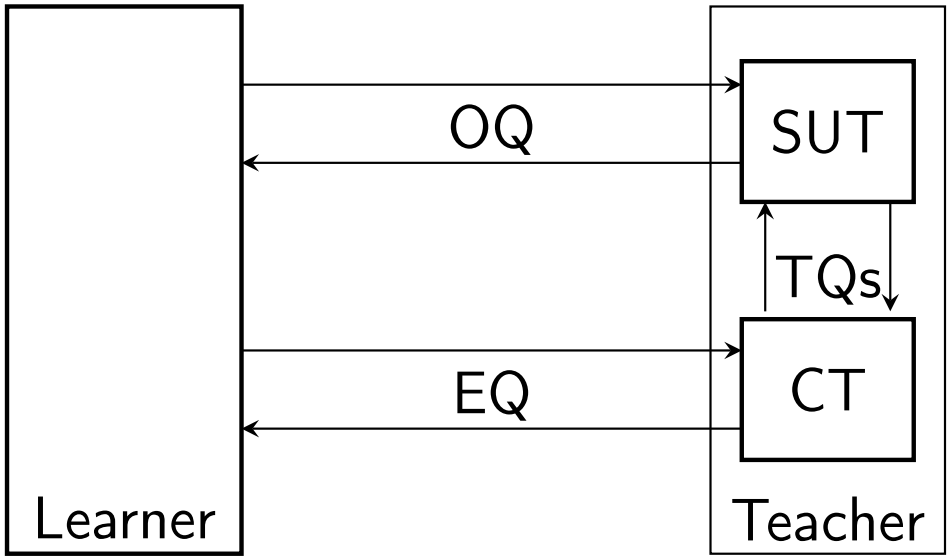


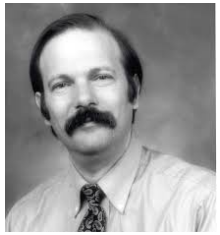




Learner: Formulate hypotheses

Conformance Tester (CT): Test correctness hypotheses

Black Box Checking (Peled, Vardi & Yannakakis)



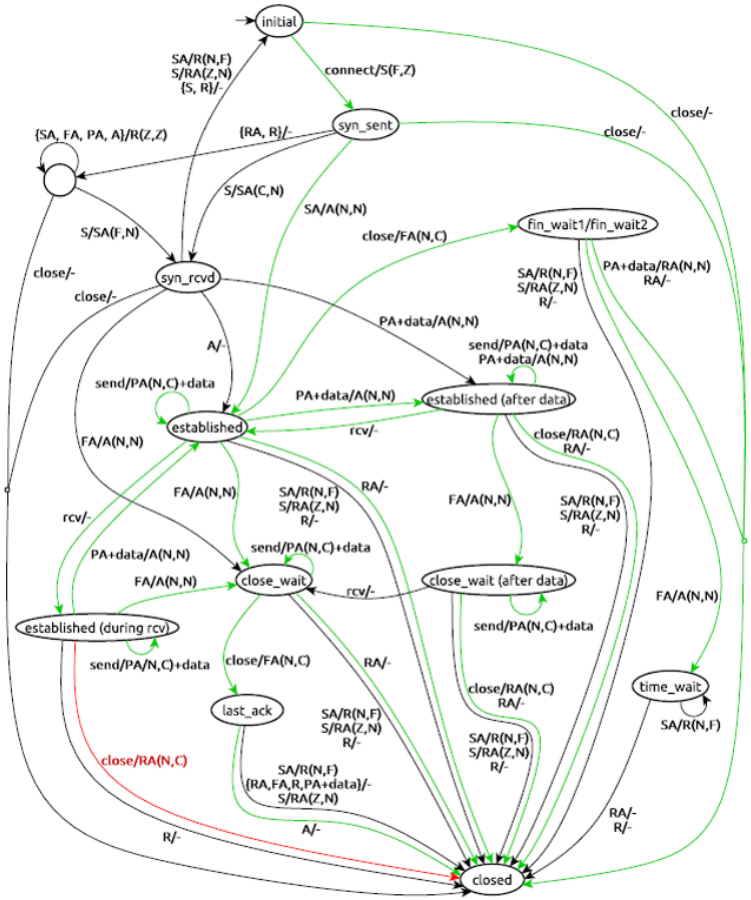




Learner: Formulate hypotheses

Conformance Tester (CT): Test correctness hypotheses

Model learning and conformance testing two sides of same coin!

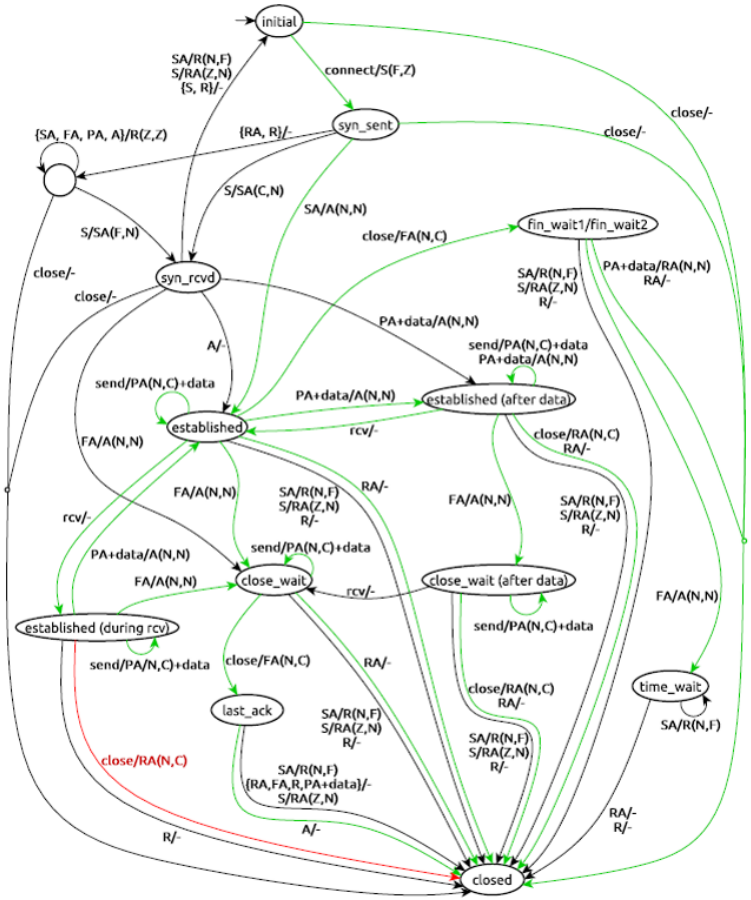
Bugs in Protocol Implementations



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.

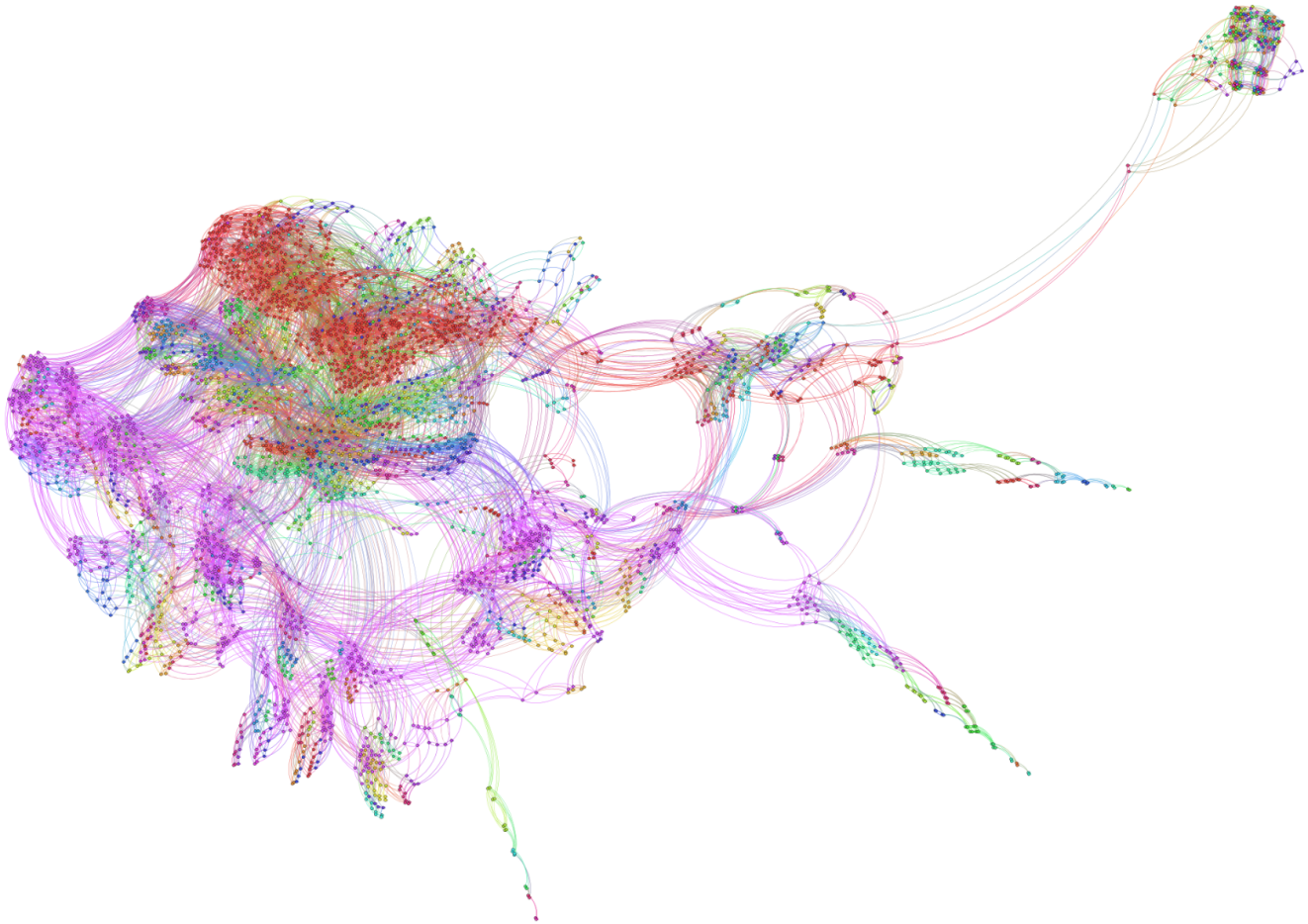
Other Protocol Case Studies

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

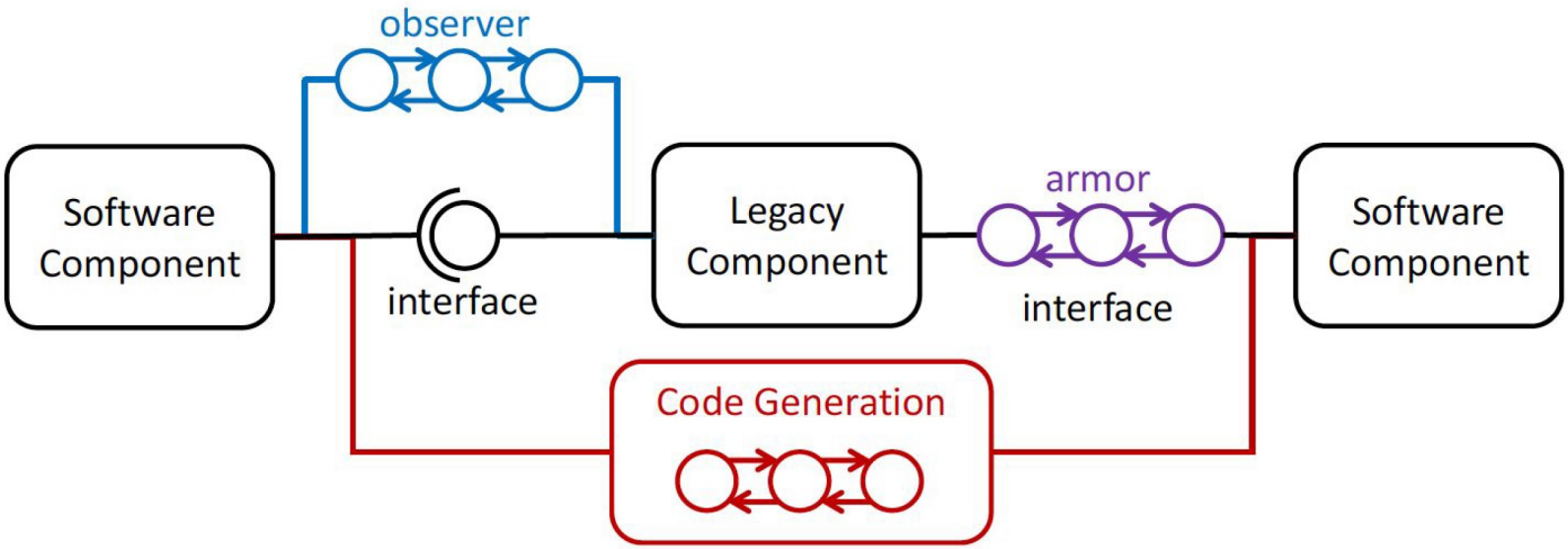
Engine Status Manager in Océ Printer (ICFEM'15)

Can we learn interface models of realistic printer controllers?

Mealy Machine for Engine Status Manager



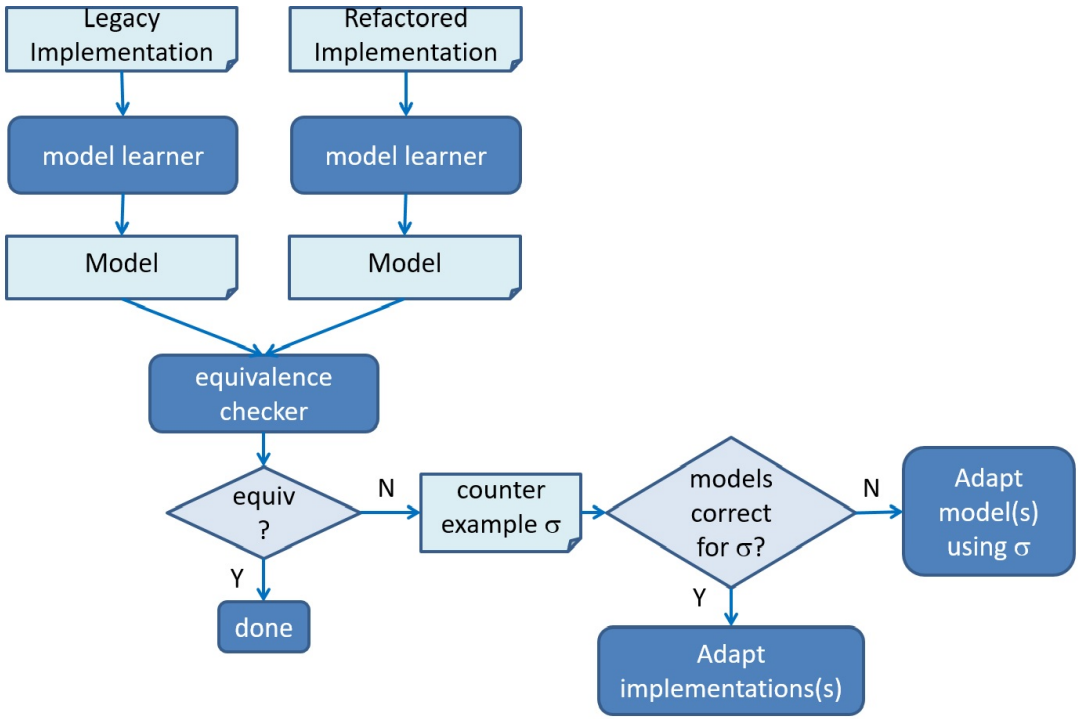
Potential Applications Interface Models



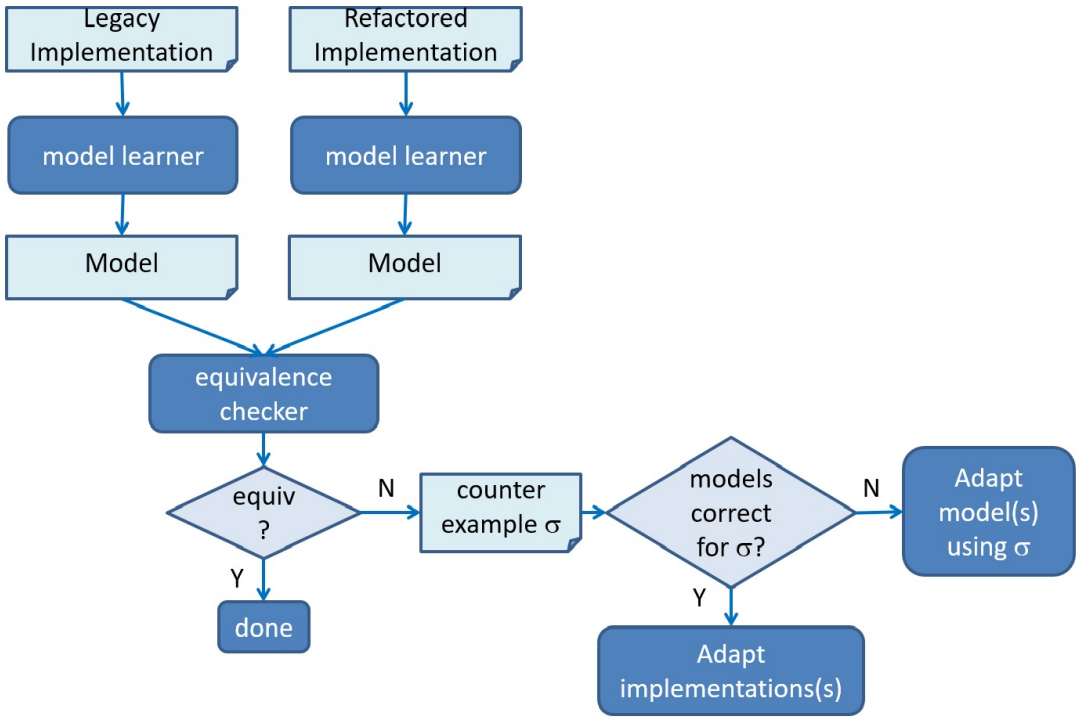
Power Control Service from Philips Healthcare (iFM'16)

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?

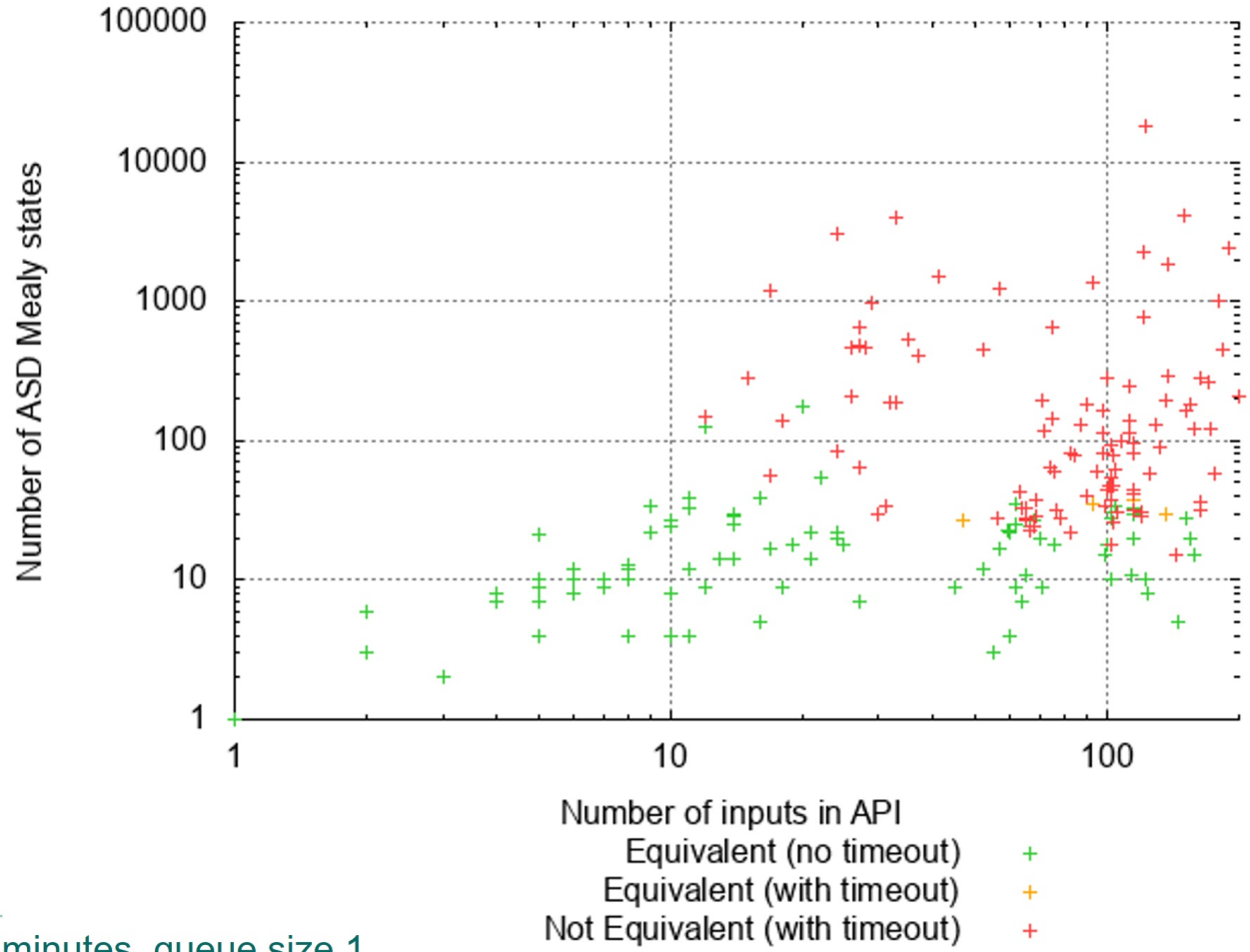
ASML Challenge

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?

$\Rightarrow$ RERS @ TOOLympics'19

Results LearnLib on ASML Benchmarks



60 minutes, queue size 1

Conclusions from Case Studies

Automata learning is emerging as an effective bug-finding technique, slowly becoming a standard tool in the toolbox of software engineers.

However:

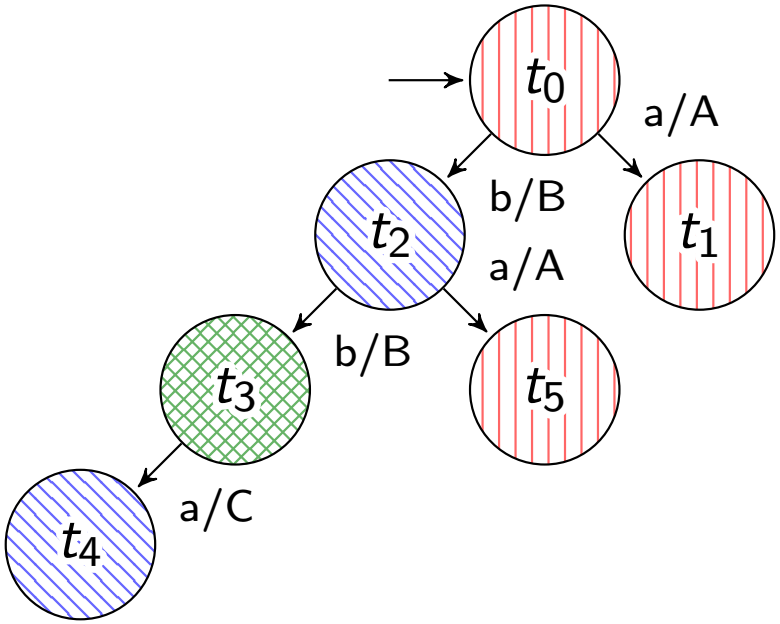
1. I/O interactions often require much time; we need algorithms that require fewer input symbols/resets to learn a model.
2. For larger cases, learning time is completely dominated by equivalence oracle; we need to integrate learning and testing.
3. We need extensions that can handle data values and timing.

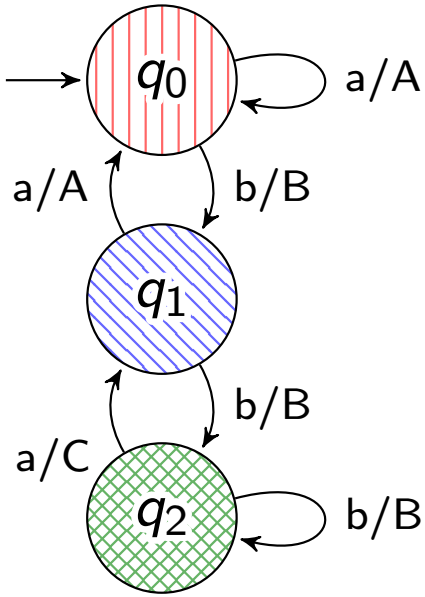
$L^\#$: A New Perspective on Active Automata Learning

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

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 $f : \mathcal{T} \to \mathcal{M}$.

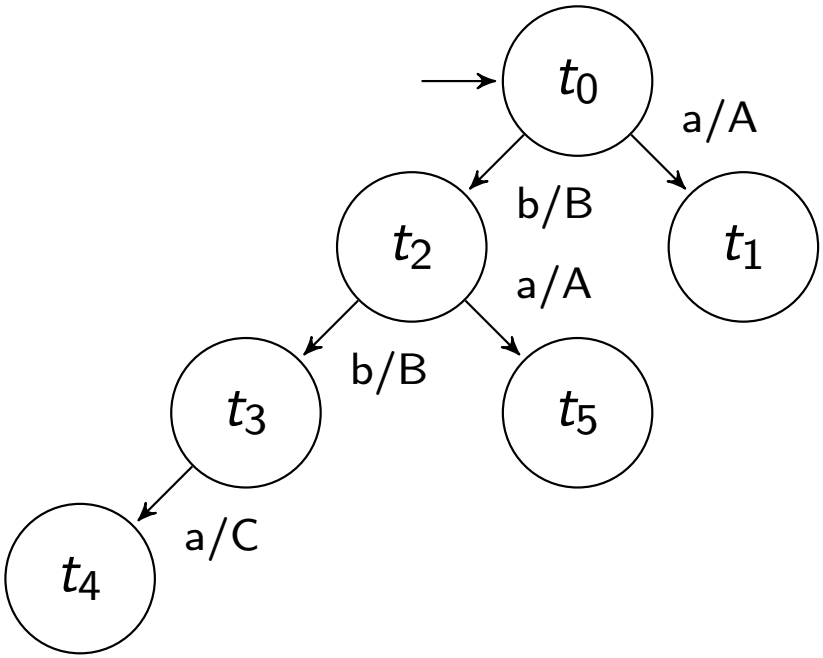
Apartness

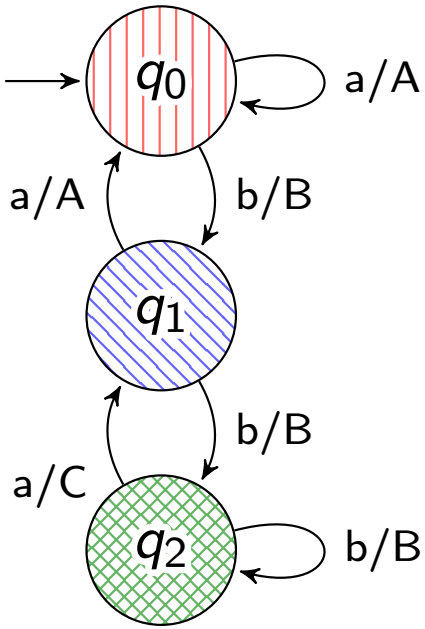
In constructive mathematics, an apartness relation is a constructive form of inequality, often taken to be more basic than equality. Two real numbers are apart if there exists (one can construct) a rational number between them.
Notion proposed by Brouwer (1919)(örtlich verschieden, Entfernung).
Heyting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

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

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

Apartness

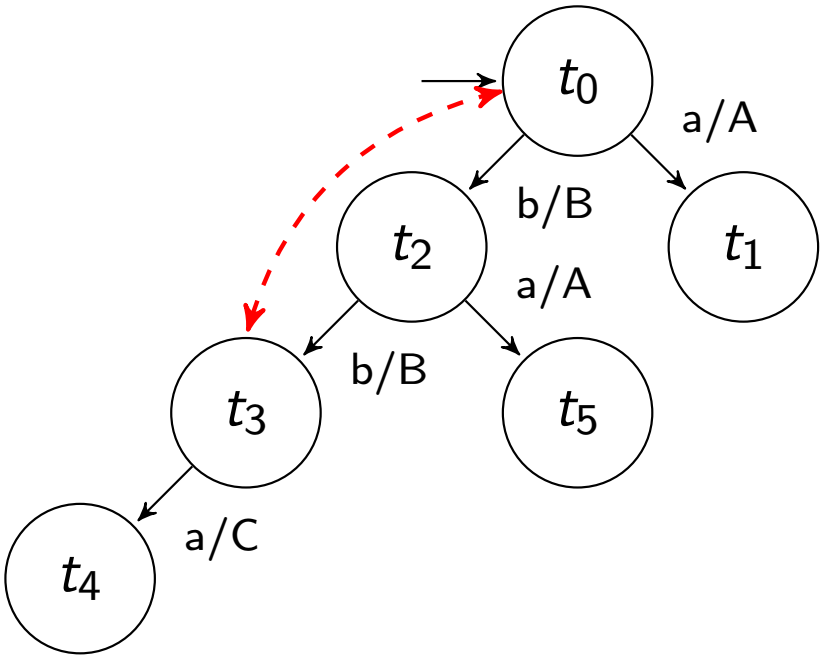




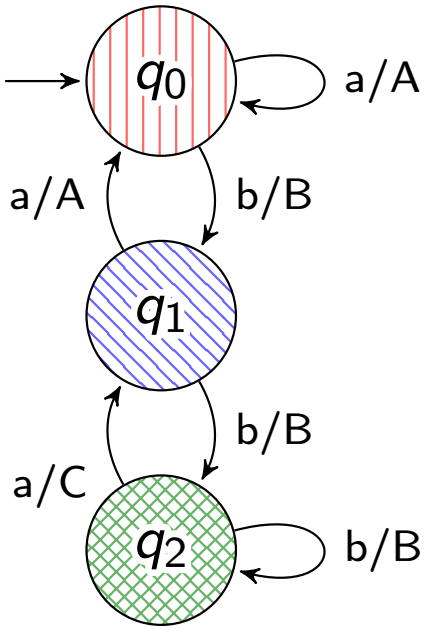
Definition (Apartness)

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



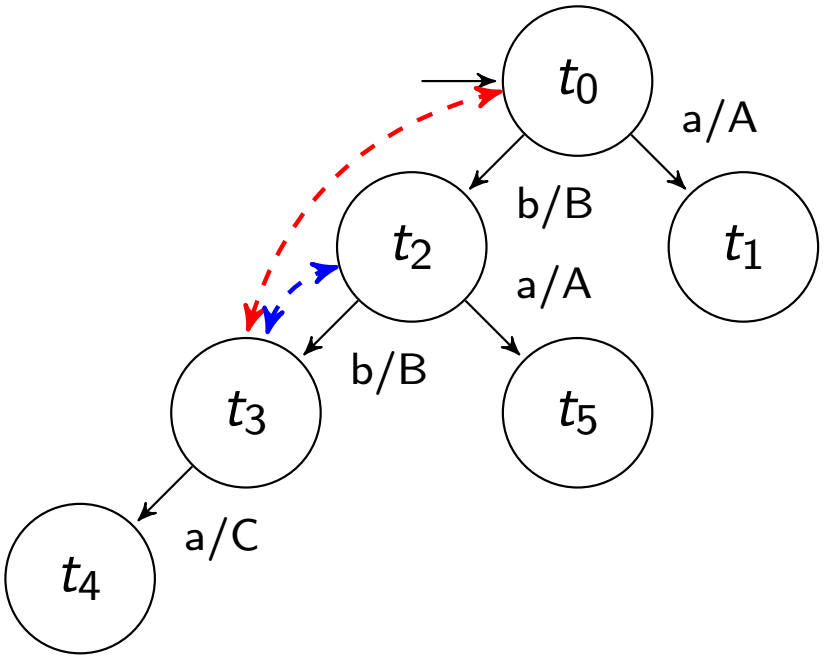
$a \vdash t_3 \# t_0$



Definition (Apartness)

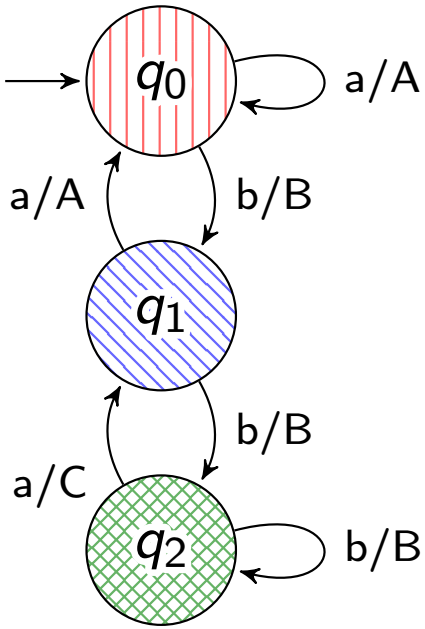
Two states t, u in an observation tree are apart, $t \# u$, if they both enable an input sequence $\sigma \in I^*$ for which the outputs are different.

Apartness



$a \vdash t_3 \# t_0$

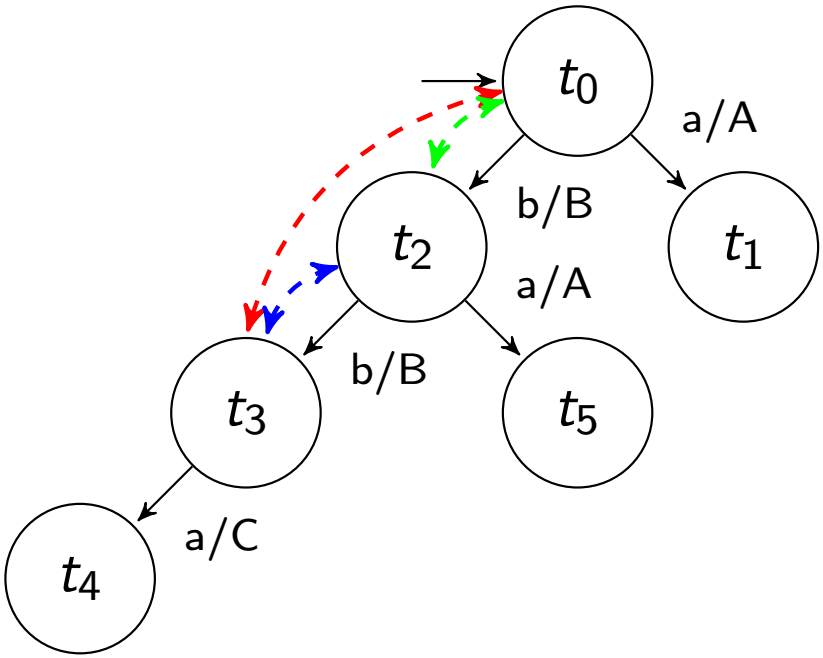
$a \vdash t_3 \# t_2$



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.

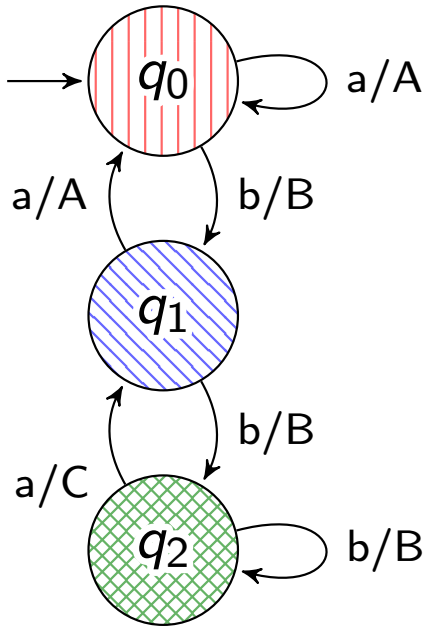
Apartness



$a \vdash t_3 \# t_0$

$a \vdash t_3 \# t_2$

$ba \vdash t_2 \# t_0$



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

Whenever states are apart in observation tree $\mathcal{T}$, the learner knows these correspond to 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 correspond to 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

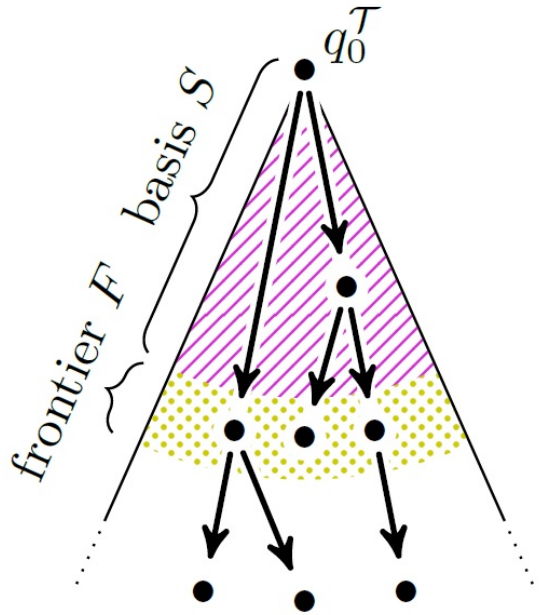
Relation $\#$ is irreflexive, symmetric and weakly co-transitive:

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

Basis and Frontier

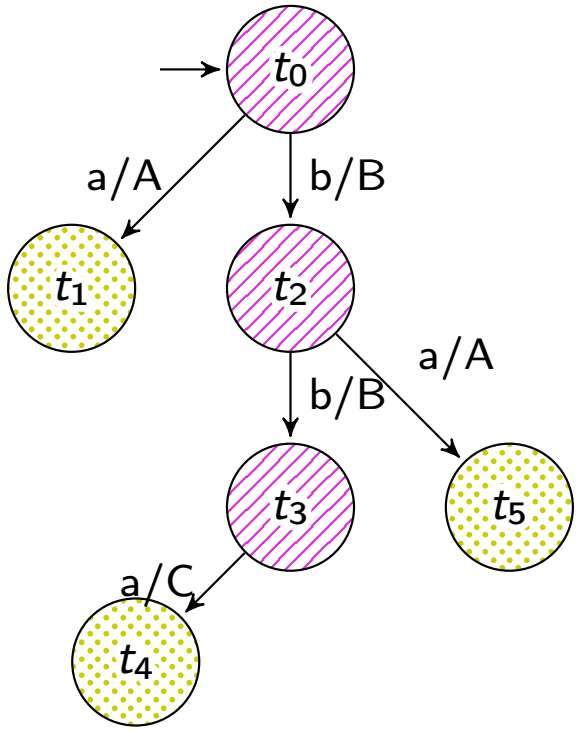
Like the shortest path algorithm of Dijkstra (1959), $L^\#$ subdivides the states of the observation tree into three sets:

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



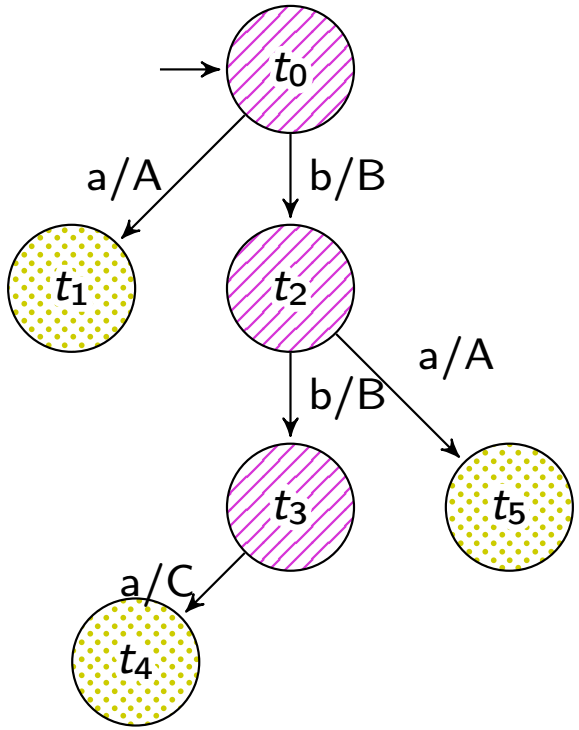
Definition

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



Definition

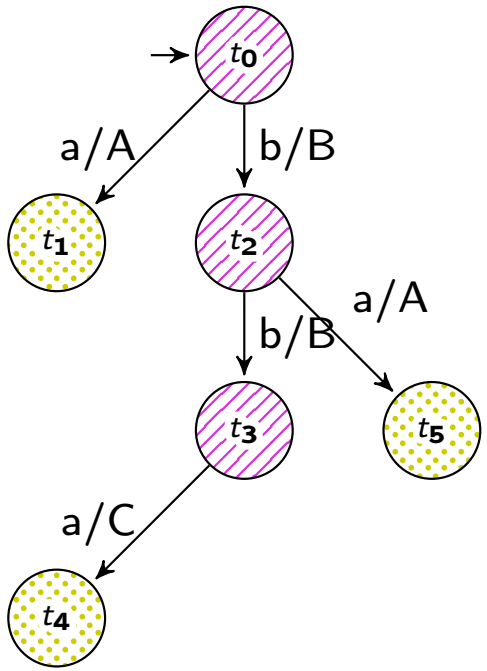
A frontier state in F is isolated if it is apart from all states in S and identified if it is apart from all states in S except one.



We use weak co-transitivity to identify frontier states!

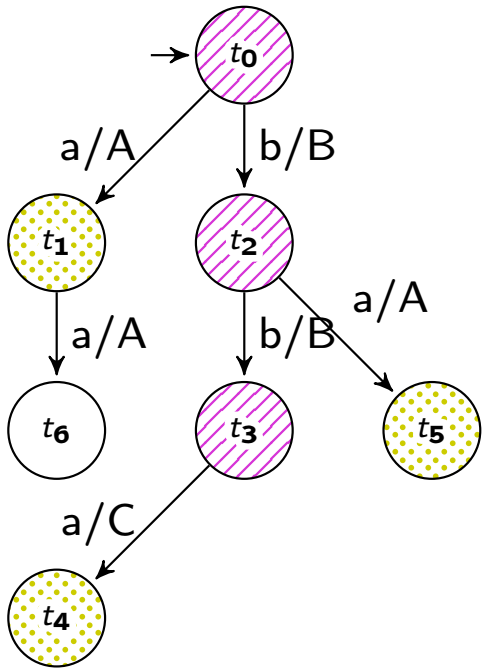
Definition

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.



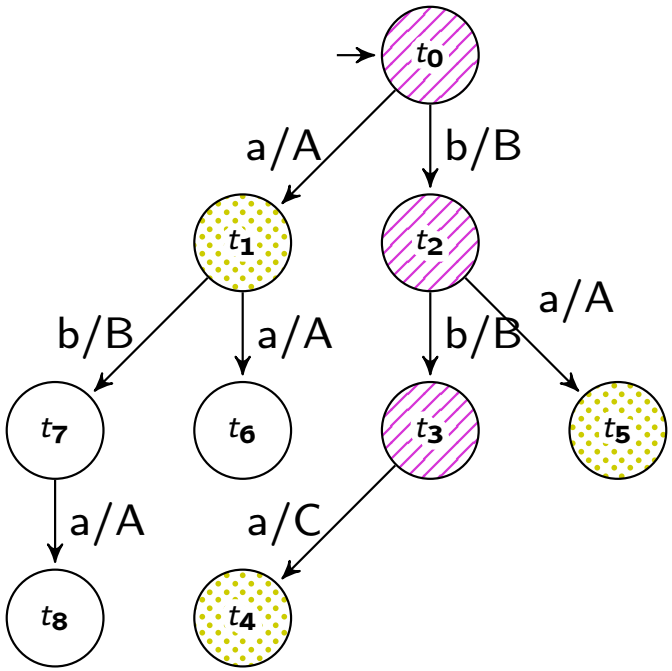
Definition

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.



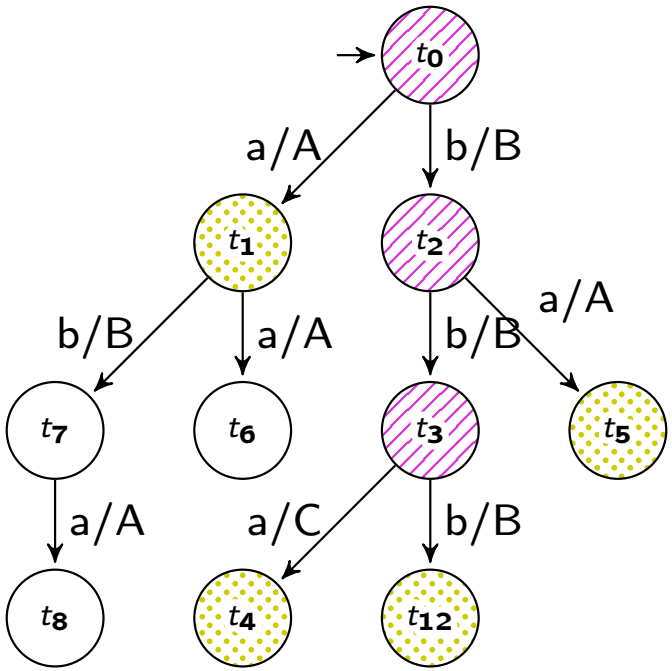
Definition

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.



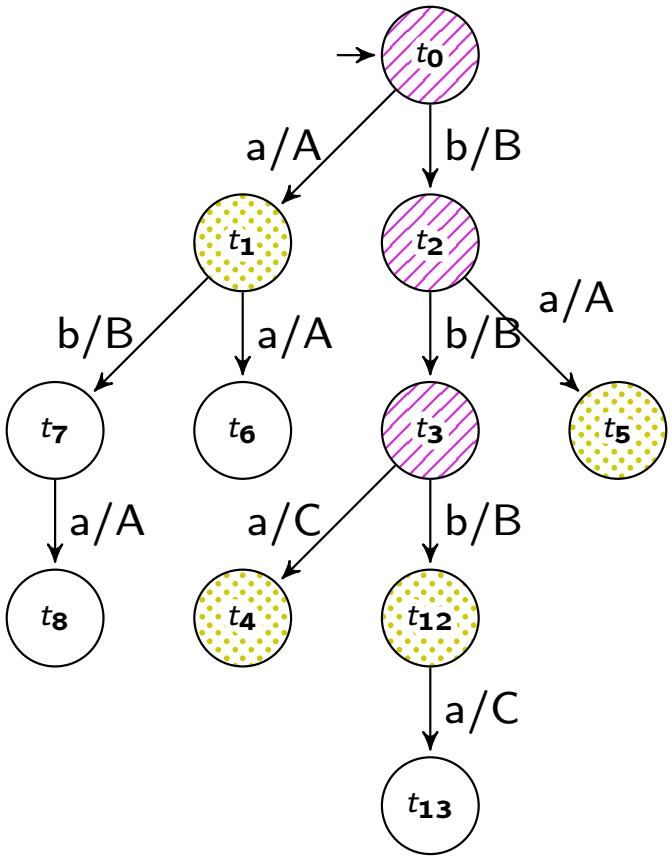
Definition

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.



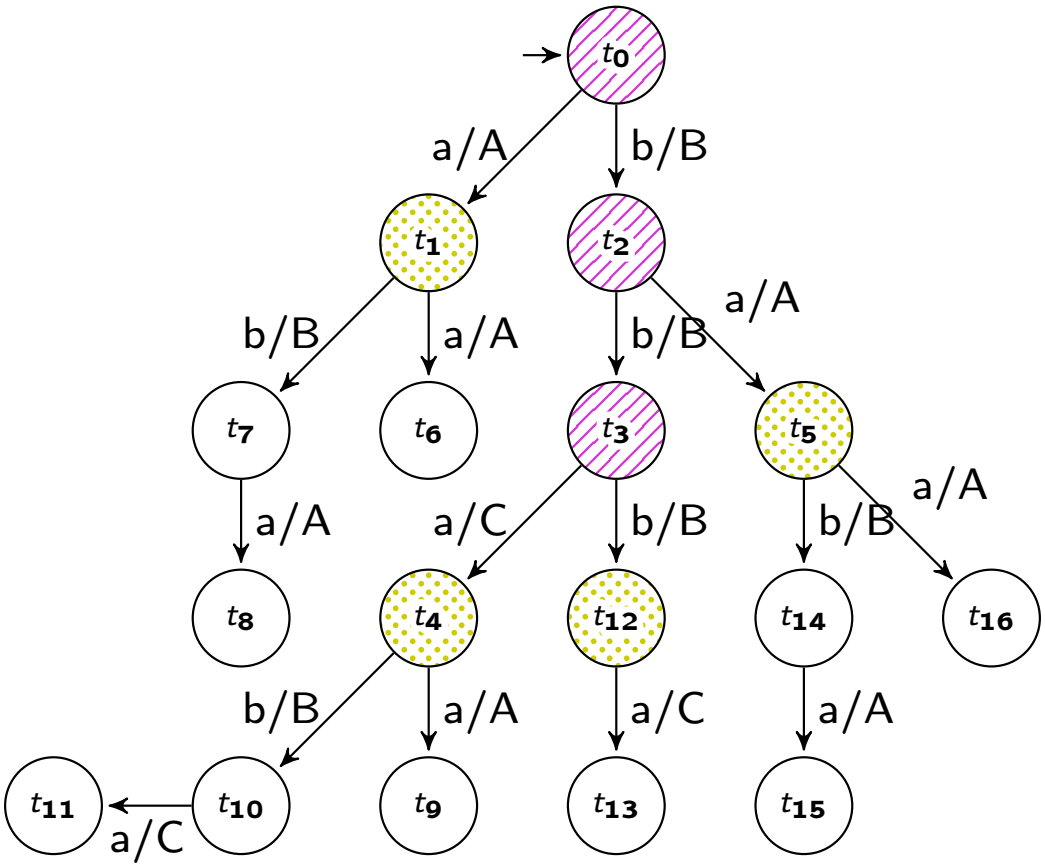
Definition

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.



Definition

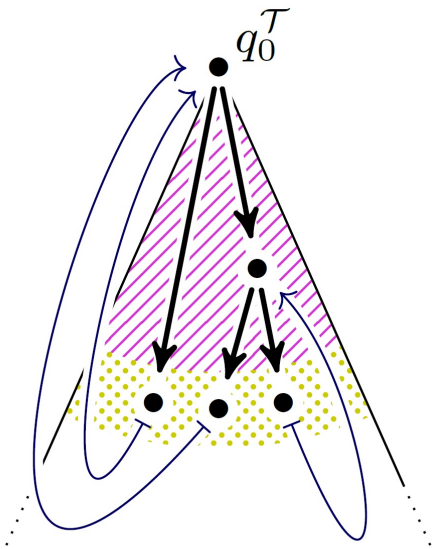
A frontier state in F is isolated if it is apart from all states in S and it is identified if it is apart from all states in S except one.

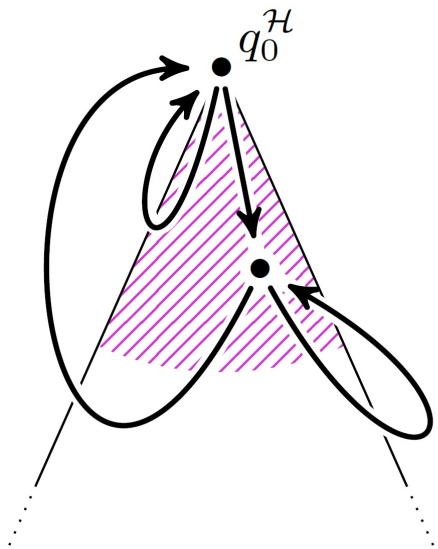


Hypothesis Construction

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)

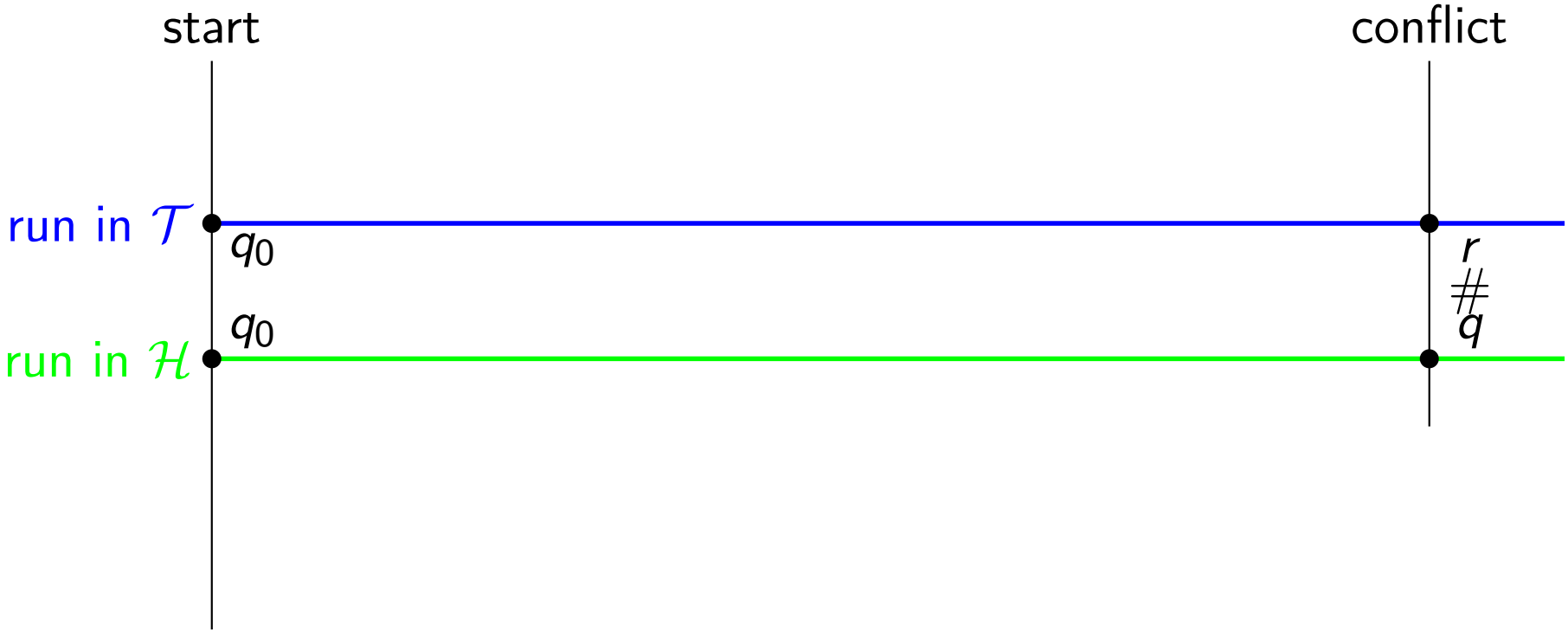
[] $\neg(q \# r)$, $\neg(q \# r')$, for some $q \in F$, $r, r' \in S$, $r \neq r' \rightarrow$ ▷ Rule (R3)
  $\sigma \leftarrow$ witness of $r \# r'$
  OutputQuery(access(q) σ)

[] 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

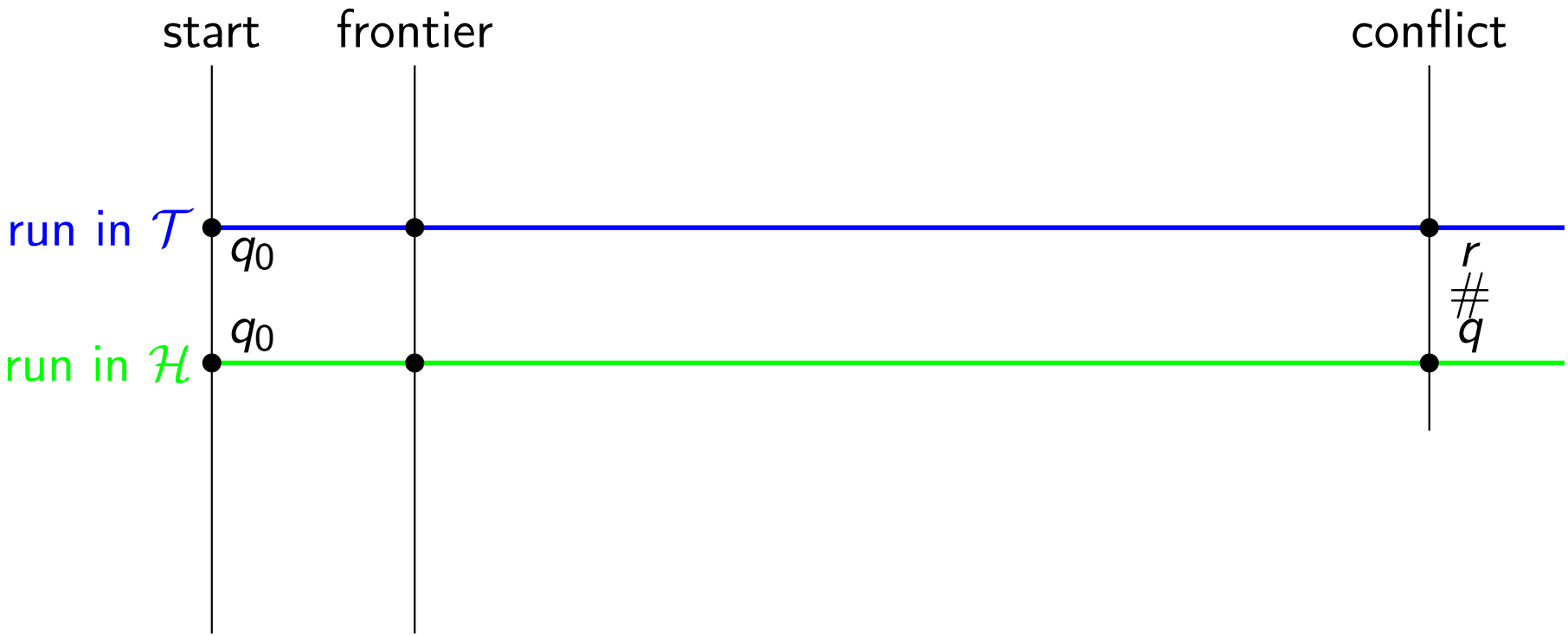
end procedure

Counterexample Processing



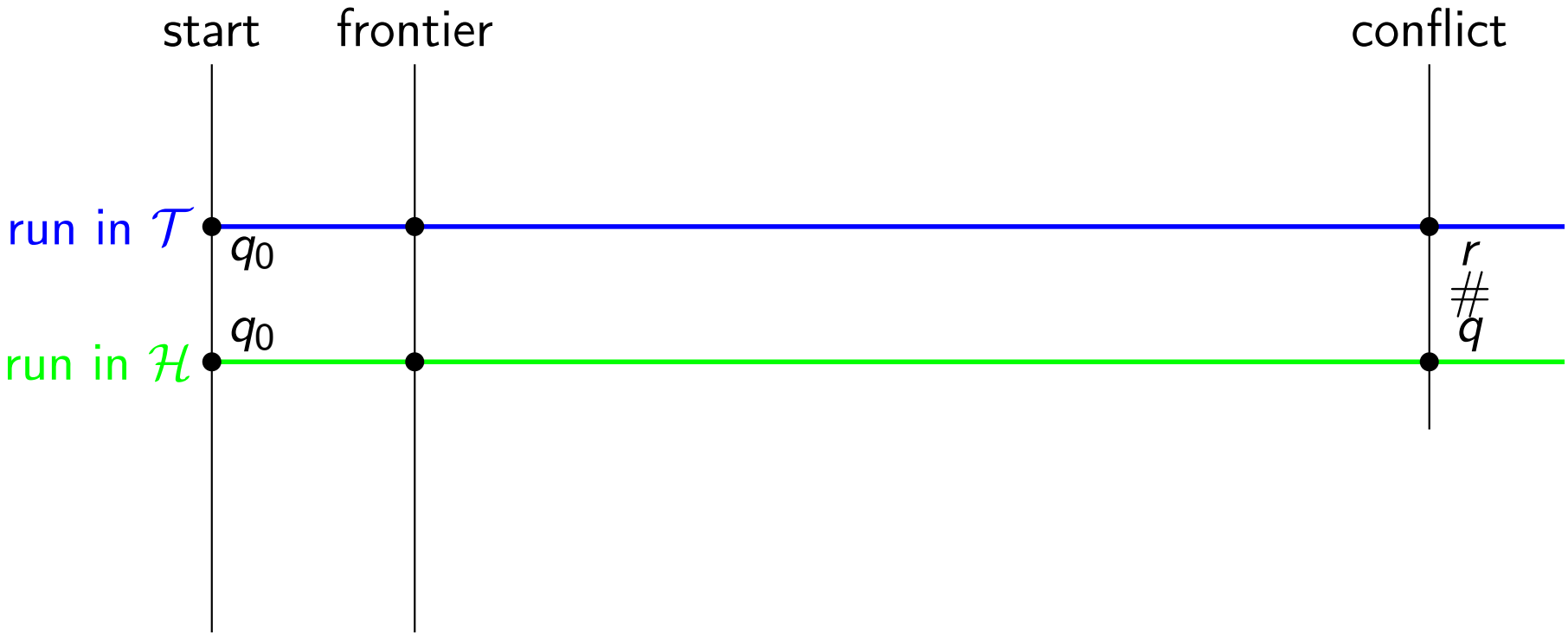
Conflict: state in run $\mathcal{T}$ is apart from corresponding state in run $\mathcal{H}$

Counterexample Processing



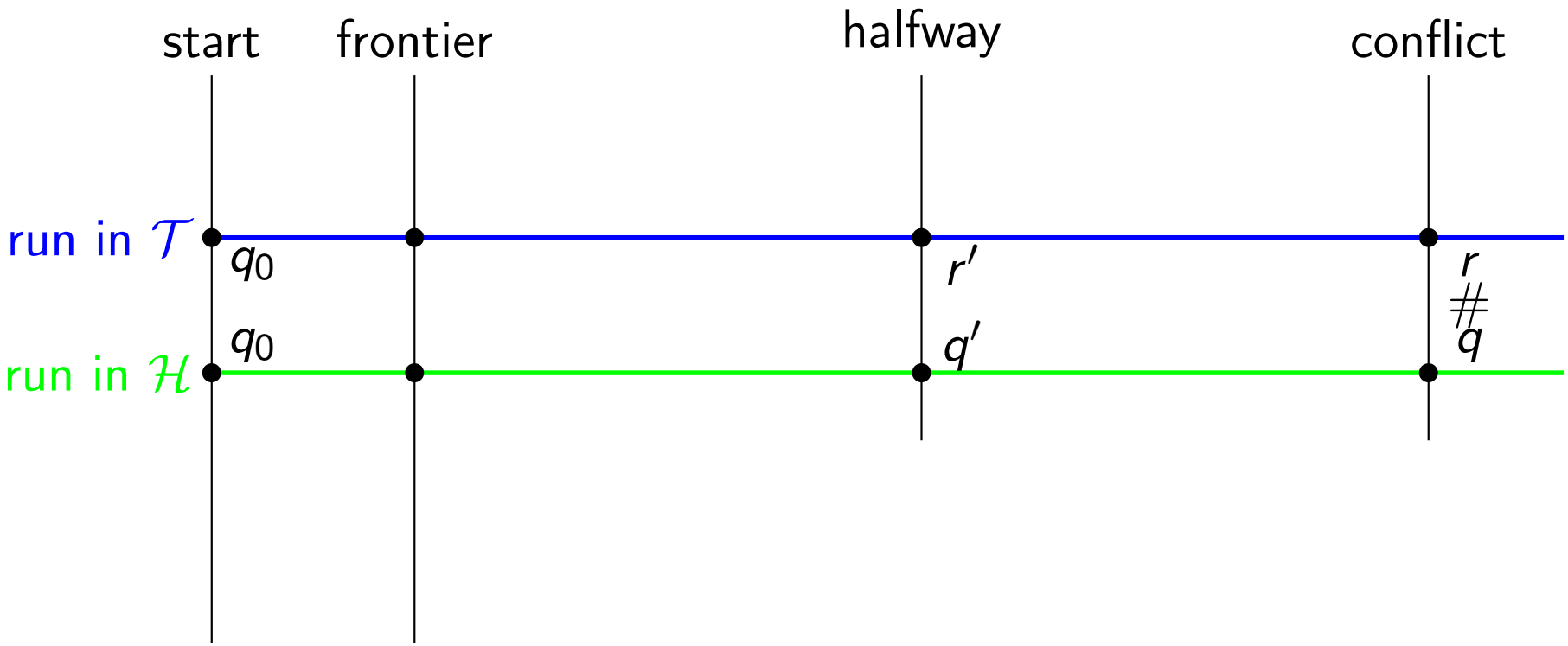
Runs $\mathcal{T}$ and $\mathcal{H}$ agree in basis, so conflict occurs at frontier or later

Counterexample Processing

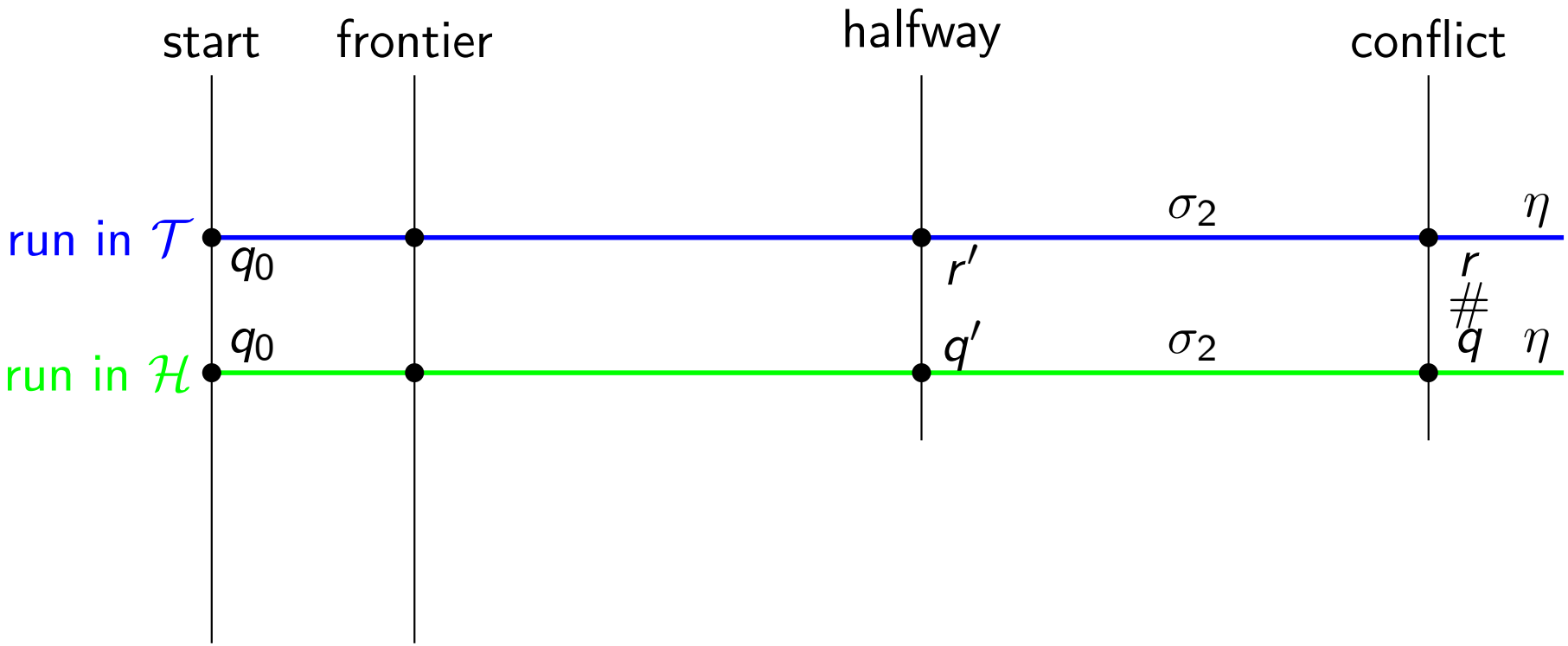


If conflict occurs at frontier we are done, otherwise we construct a conflict whose distance to the frontier is at least twice as small

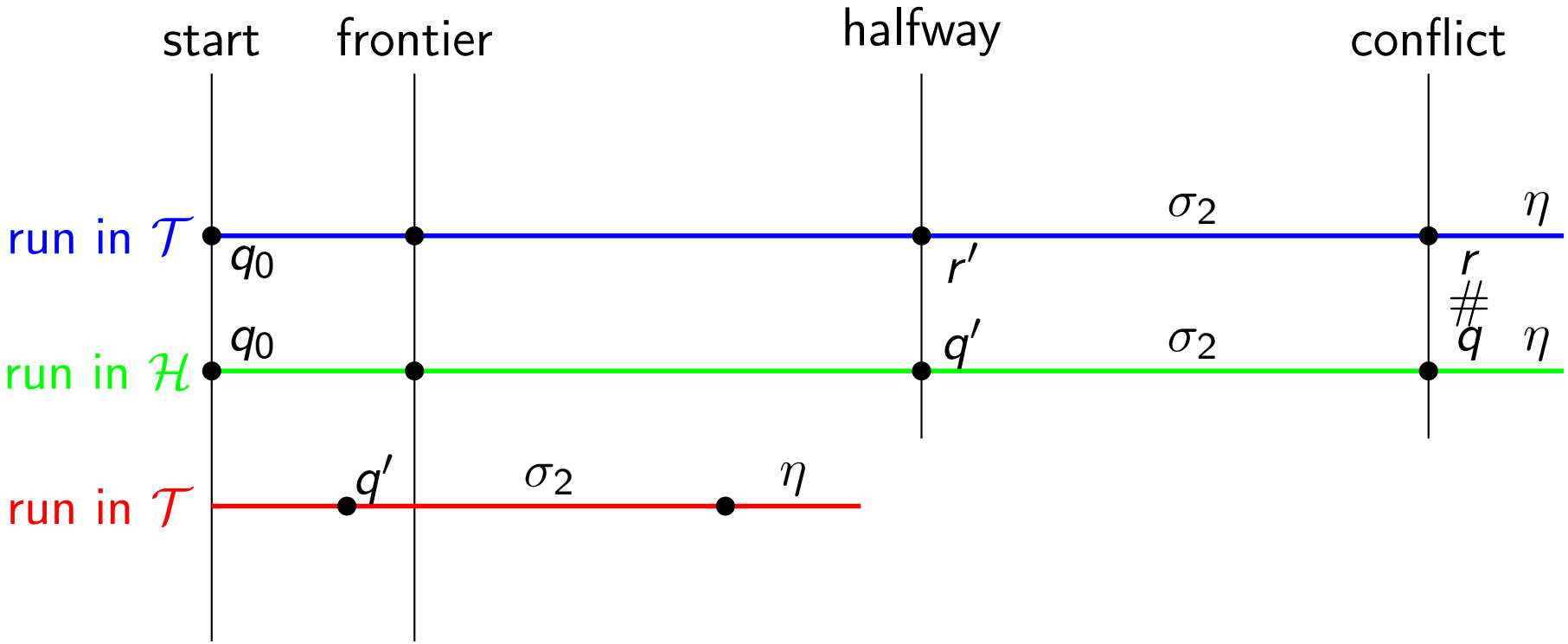
Counterexample Processing



Counterexample Processing

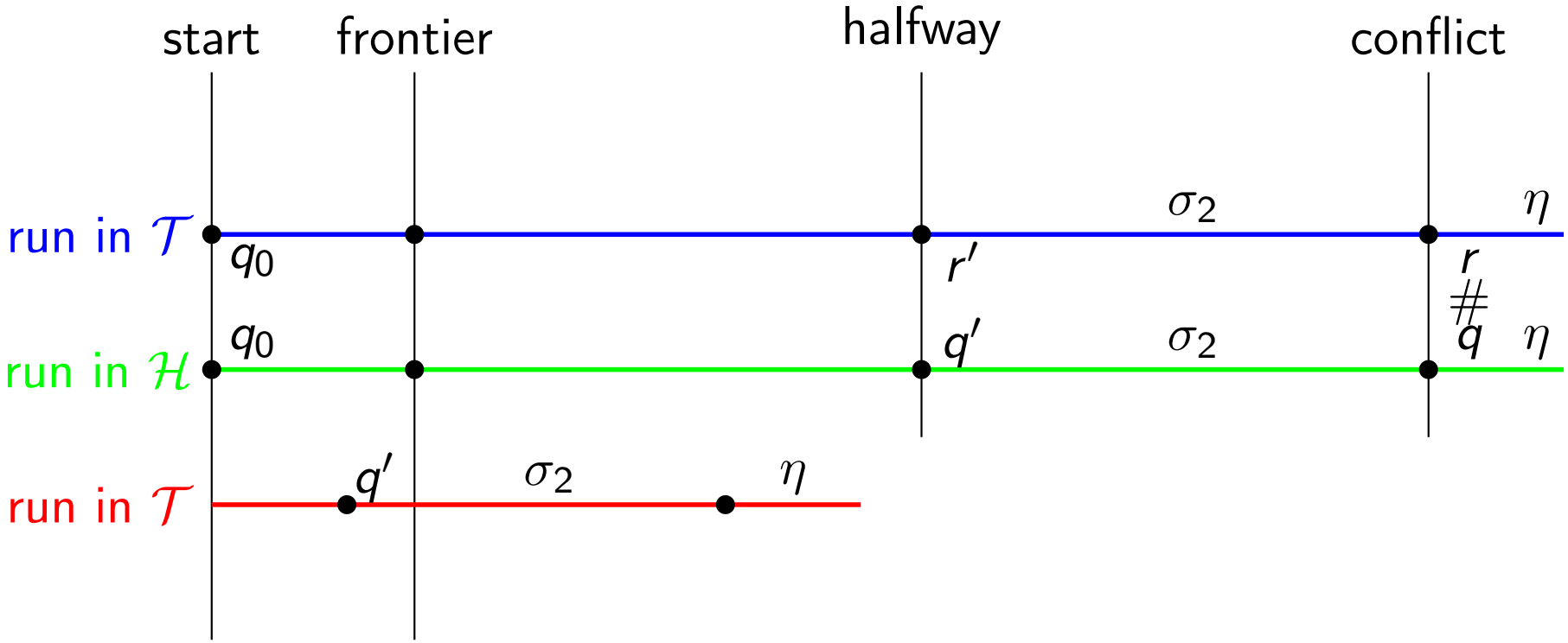


Counterexample Processing



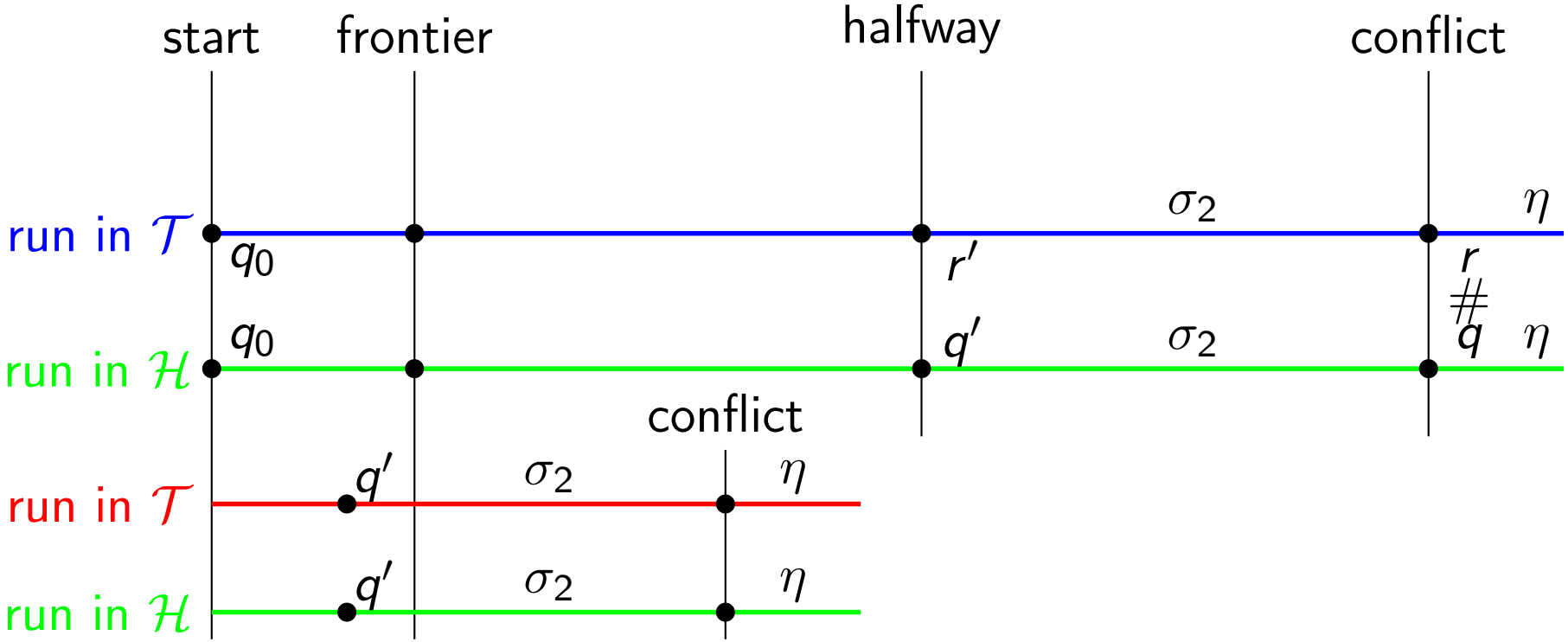
Key idea: perform output query access$(q')\sigma_2\eta$

Counterexample Processing



Key idea: perform output query access$(q')\sigma_2\eta$
If outputs for $\sigma_2\eta$ from r' and q' are different in $\mathcal{T}$ then $r'\#q'$

Counterexample Processing

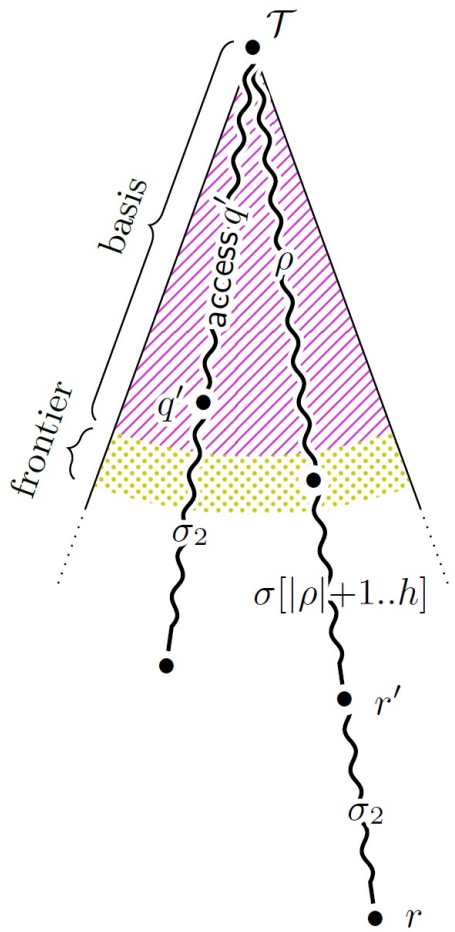


Key idea: perform output query access$(q')\sigma_2\eta$
If outputs for $\sigma_2\eta$ from r' and q' are different in $\mathcal{T}$ then $r'\#q'$
Else access$(q')\sigma_2$ leads to a conflict!

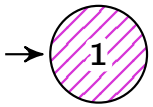
Counterexample Processing Algorithm

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

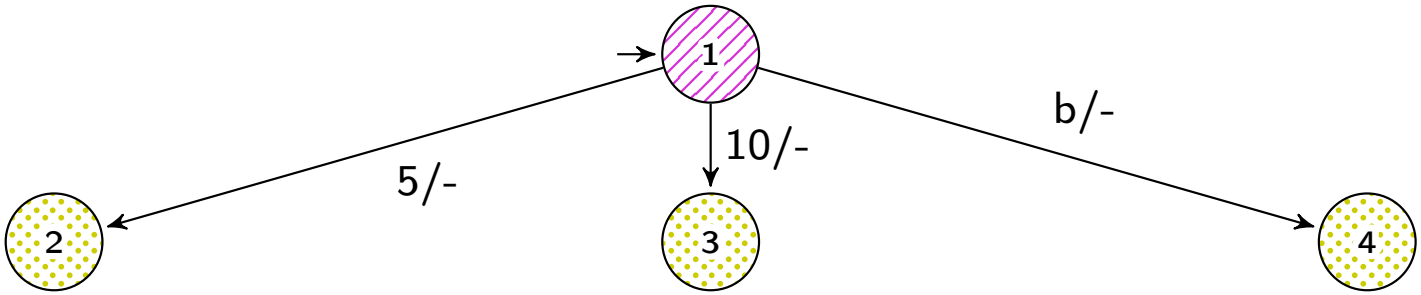
procedure ProcCounterEx($\mathcal{H}$, $\sigma \in I^*$)
  $q \leftarrow \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma)$
  $r \leftarrow \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma)$
  if $r \in S \cup F$ **then**
    return
  else
    $\rho \leftarrow$ unique prefix of σ with $\delta^{\mathcal{T}}(q_0^{\mathcal{T}},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



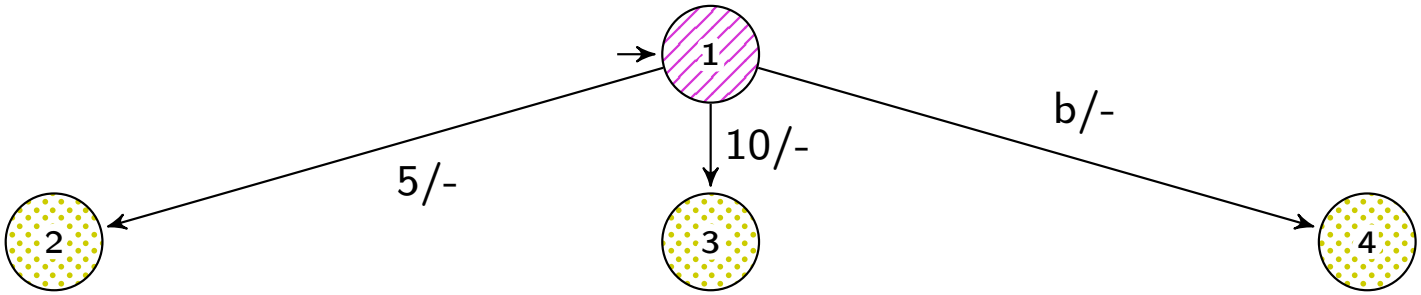
Learning Coffee Machine Model with $L^\#$

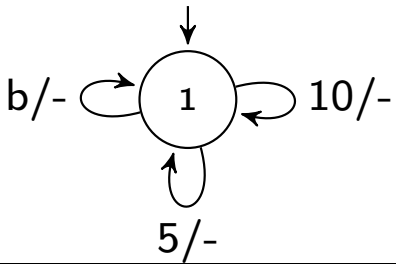


Learning Coffee Machine Model with $L^\#$

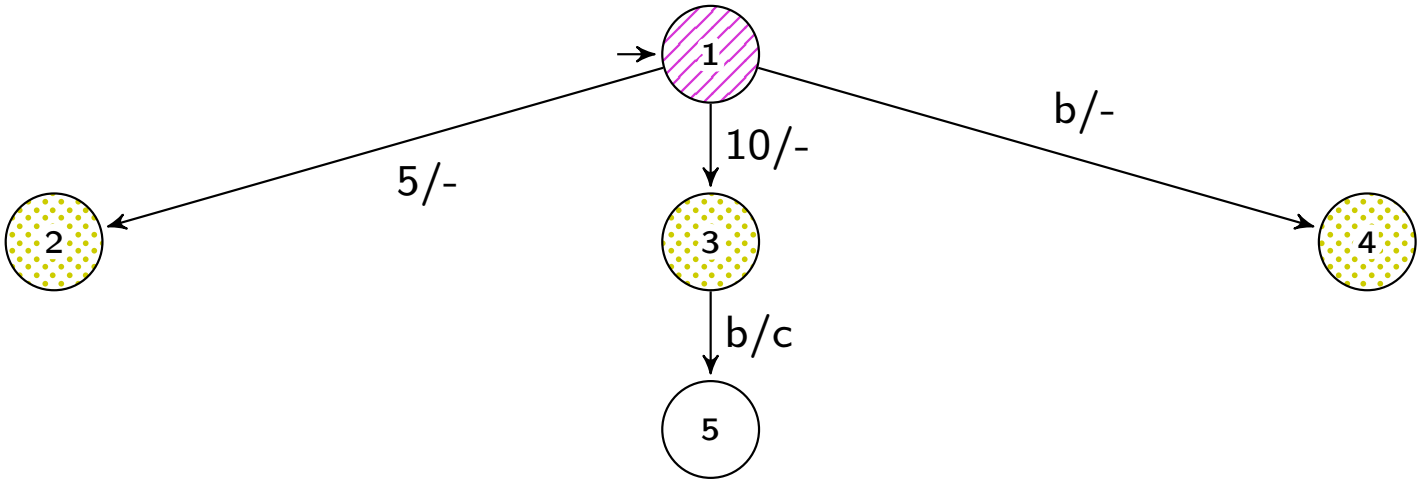


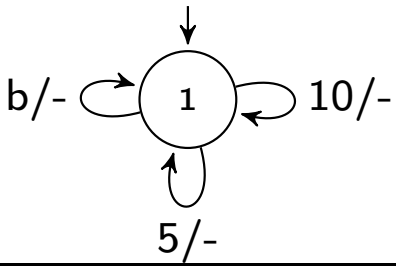
Learning Coffee Machine Model with $L^\#$



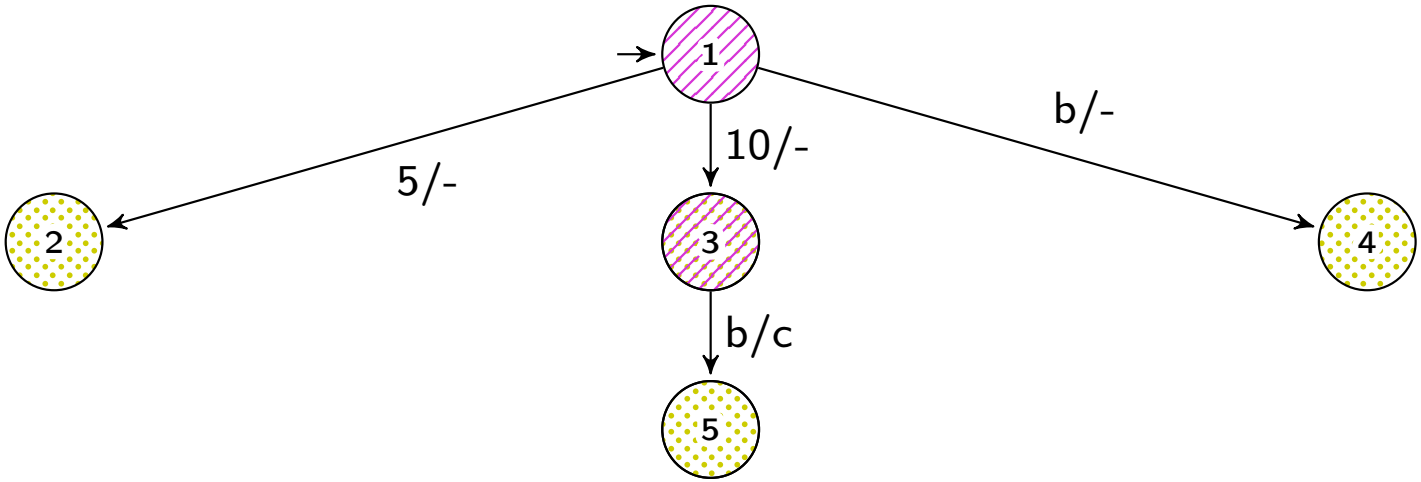


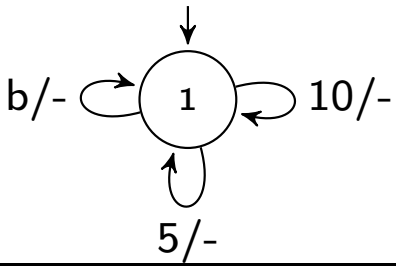
Learning Coffee Machine Model with $L^\#$



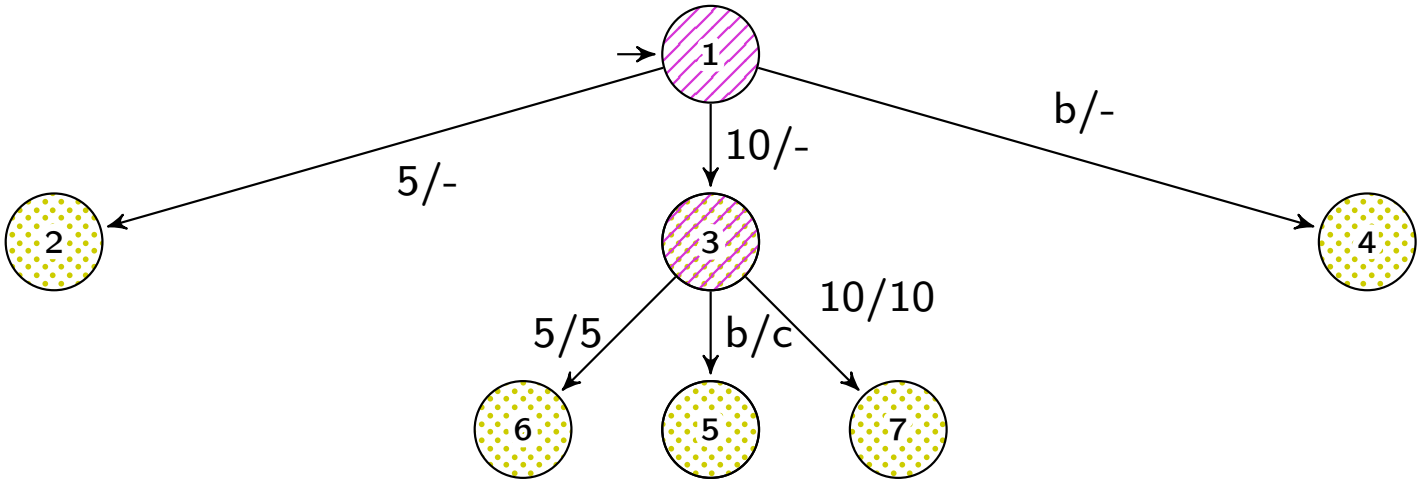


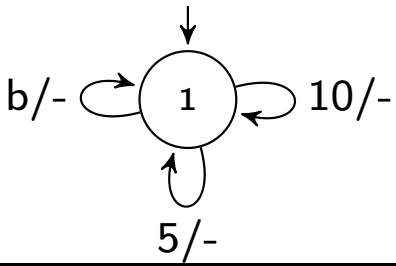
Learning Coffee Machine Model with $L^\#$



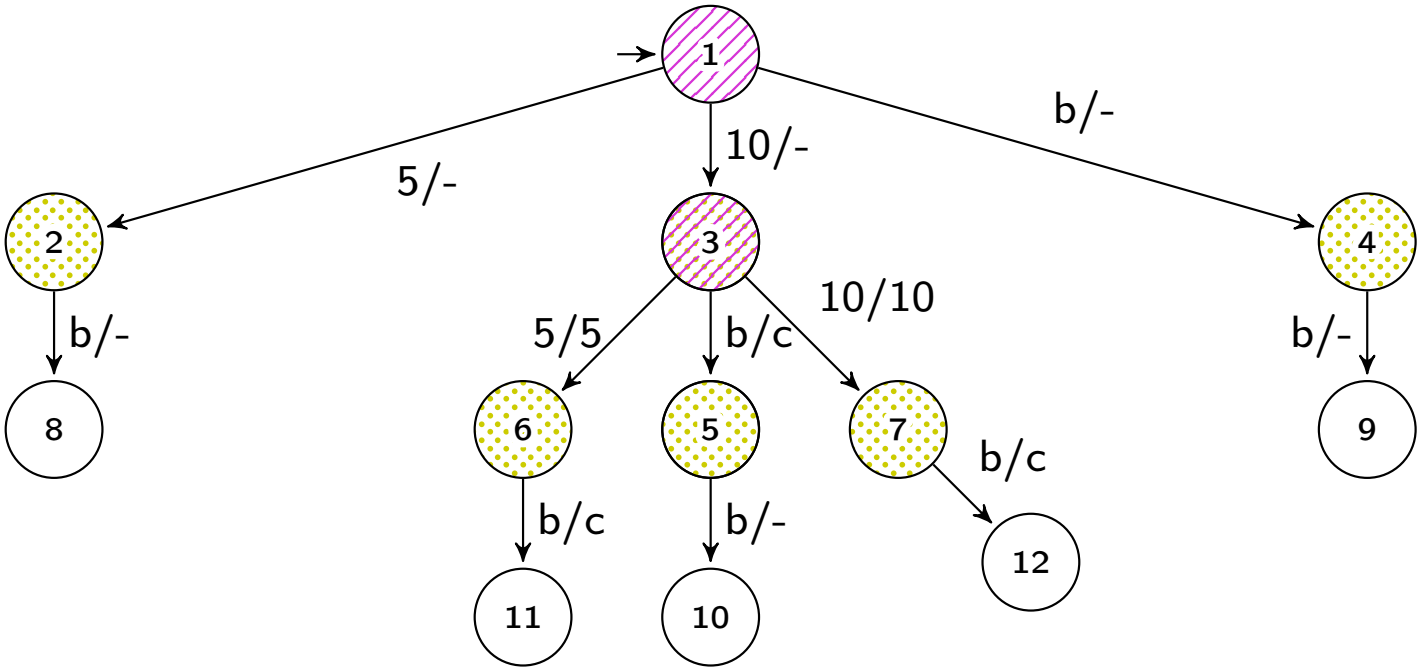


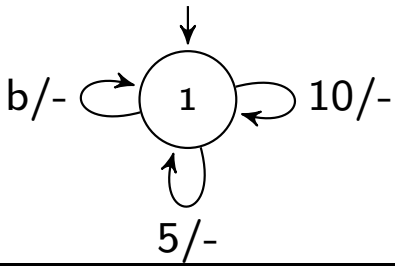
Learning Coffee Machine Model with $L^\#$



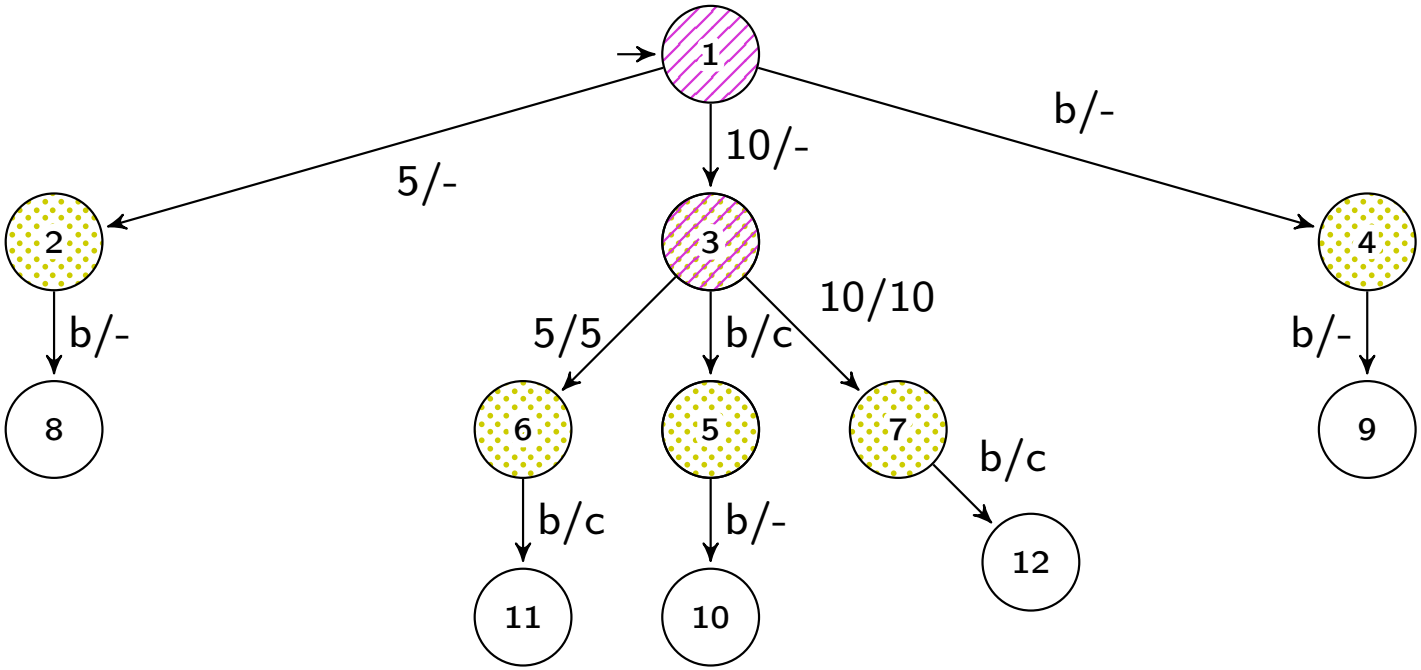


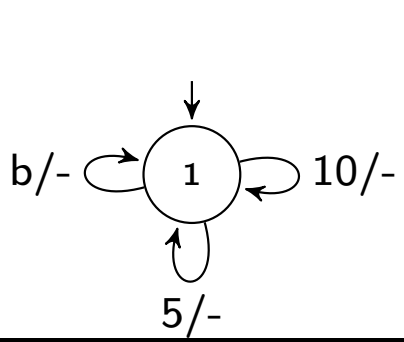
Learning Coffee Machine Model with $L^\#$

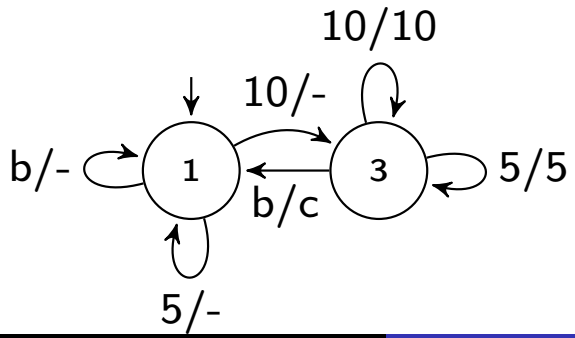




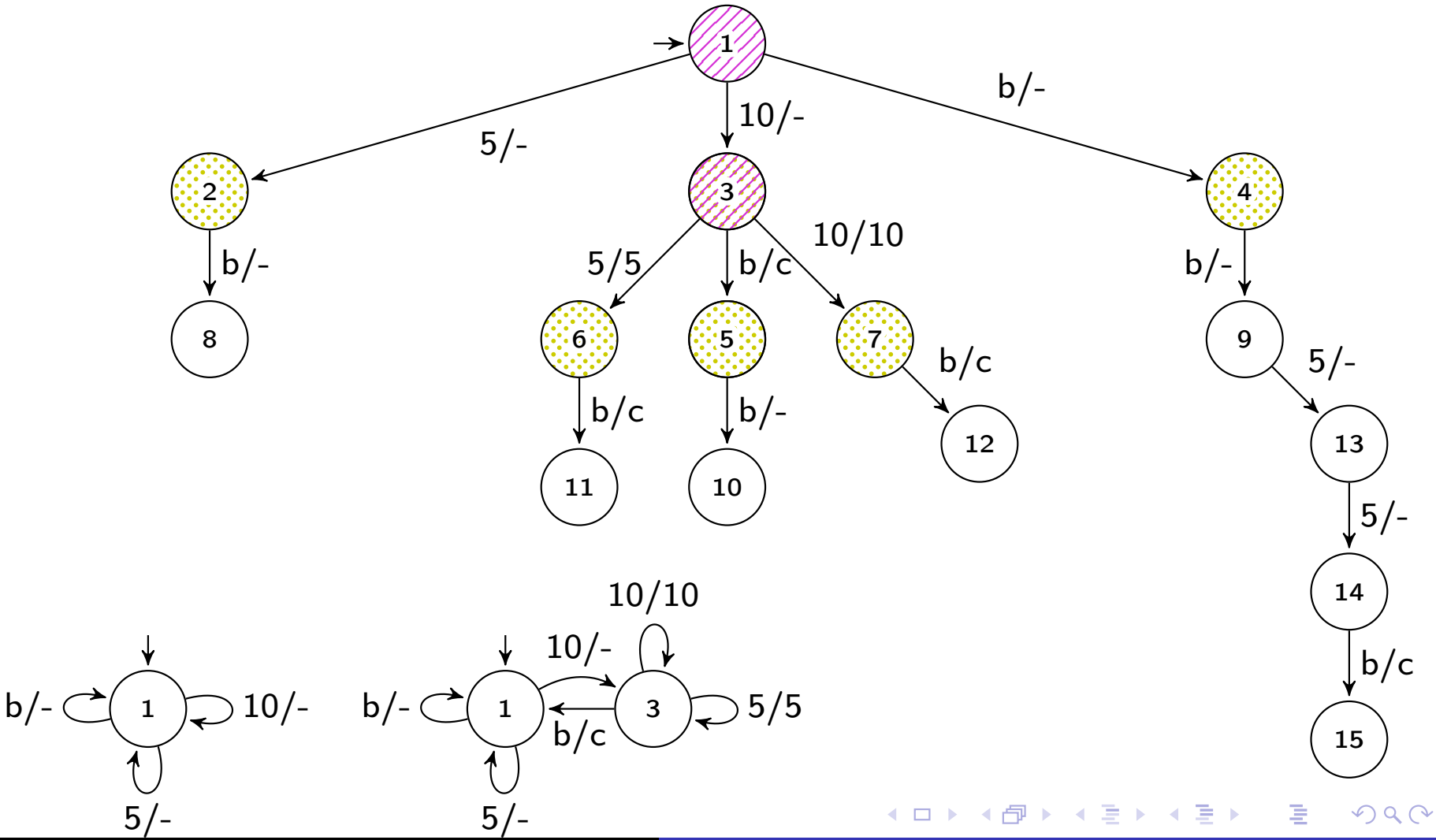
Learning Coffee Machine Model with $L^\#$



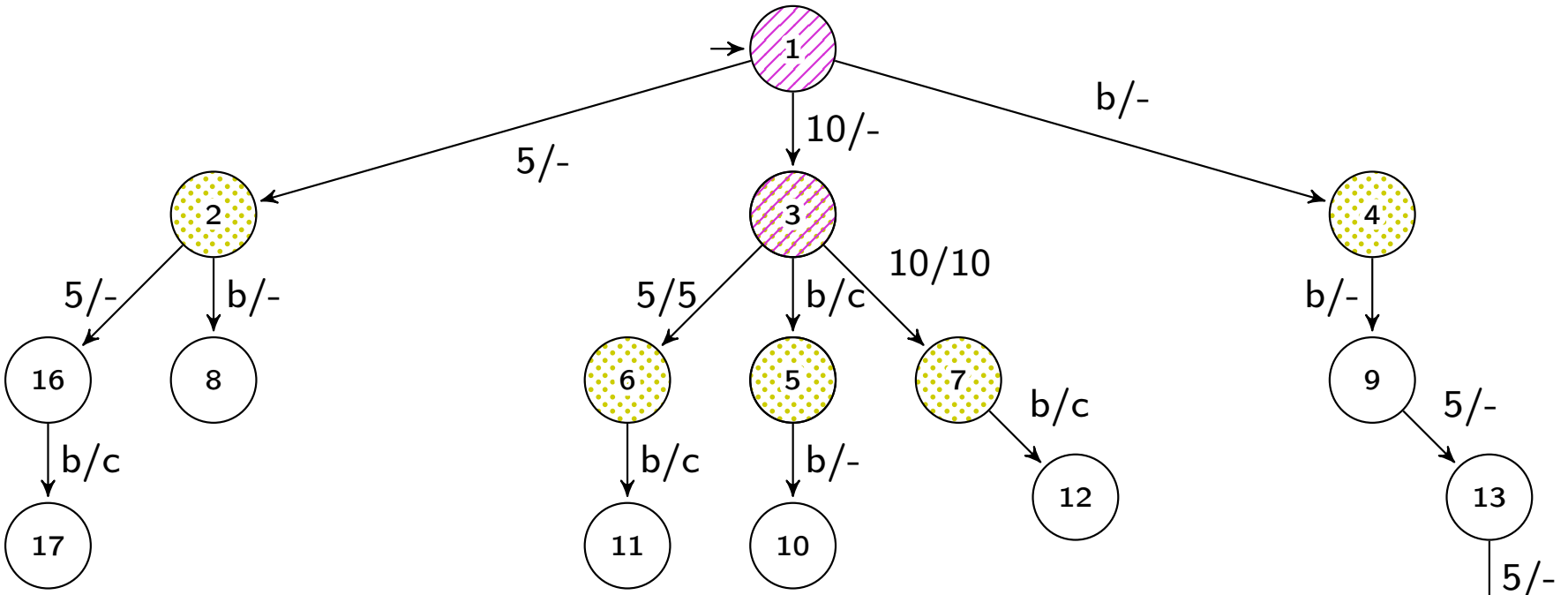


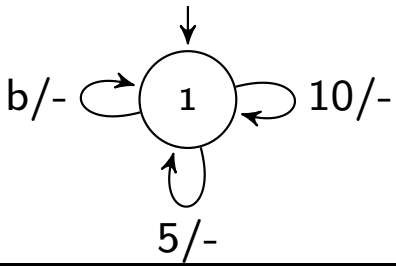


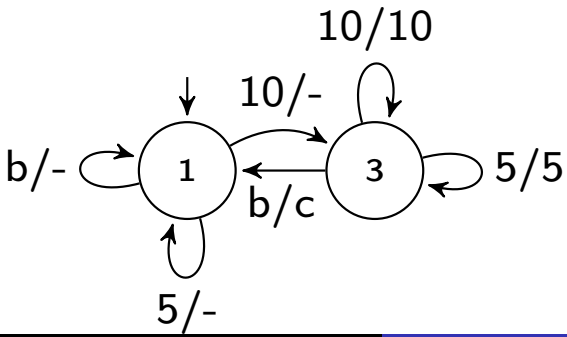
Learning Coffee Machine Model with $L^\#$



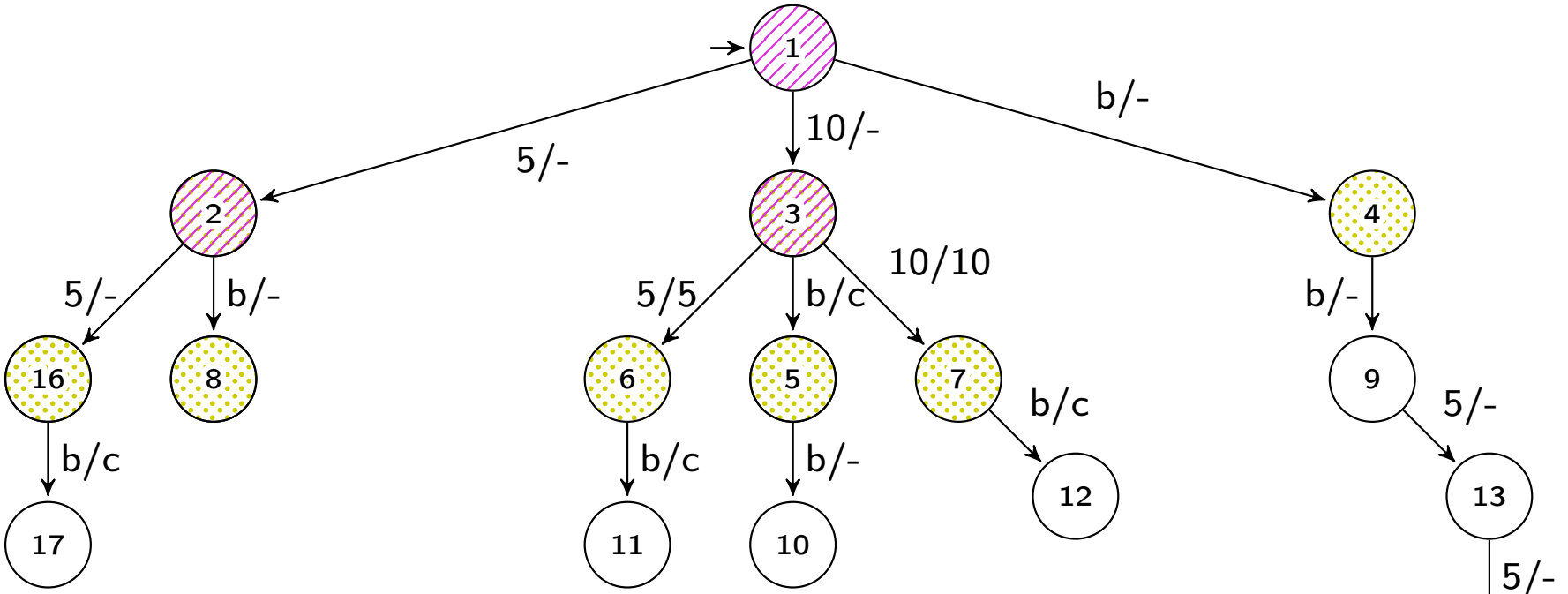
Learning Coffee Machine Model with $L^\#$

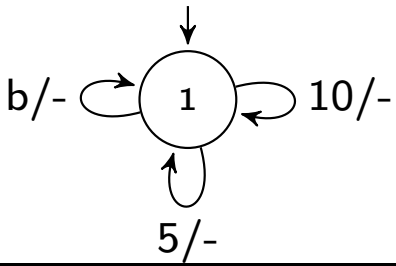


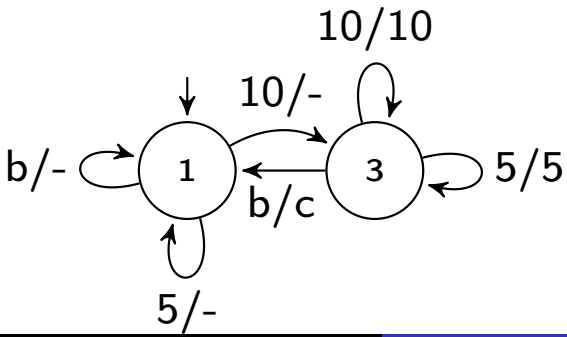




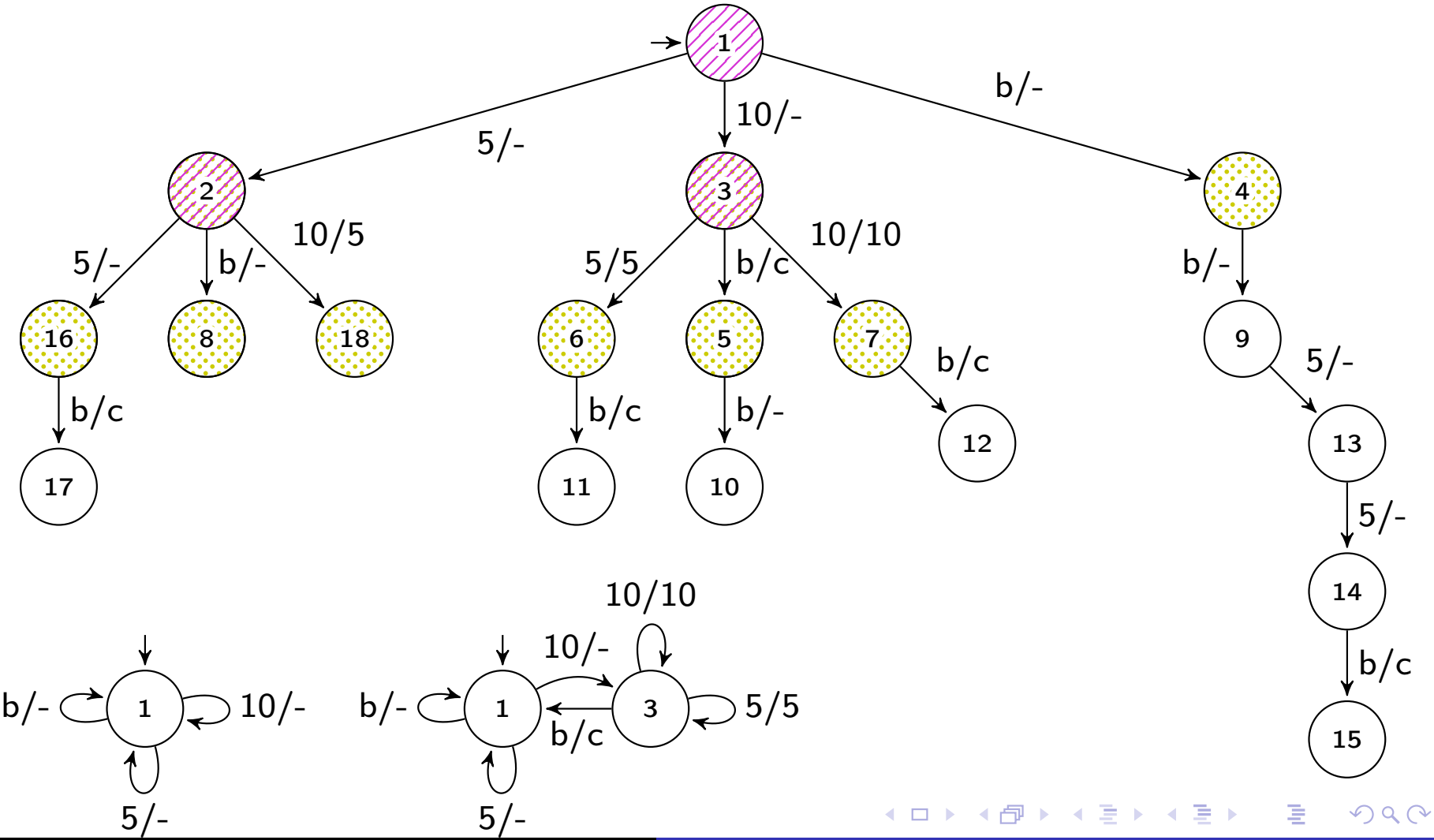
Learning Coffee Machine Model with $L^\#$



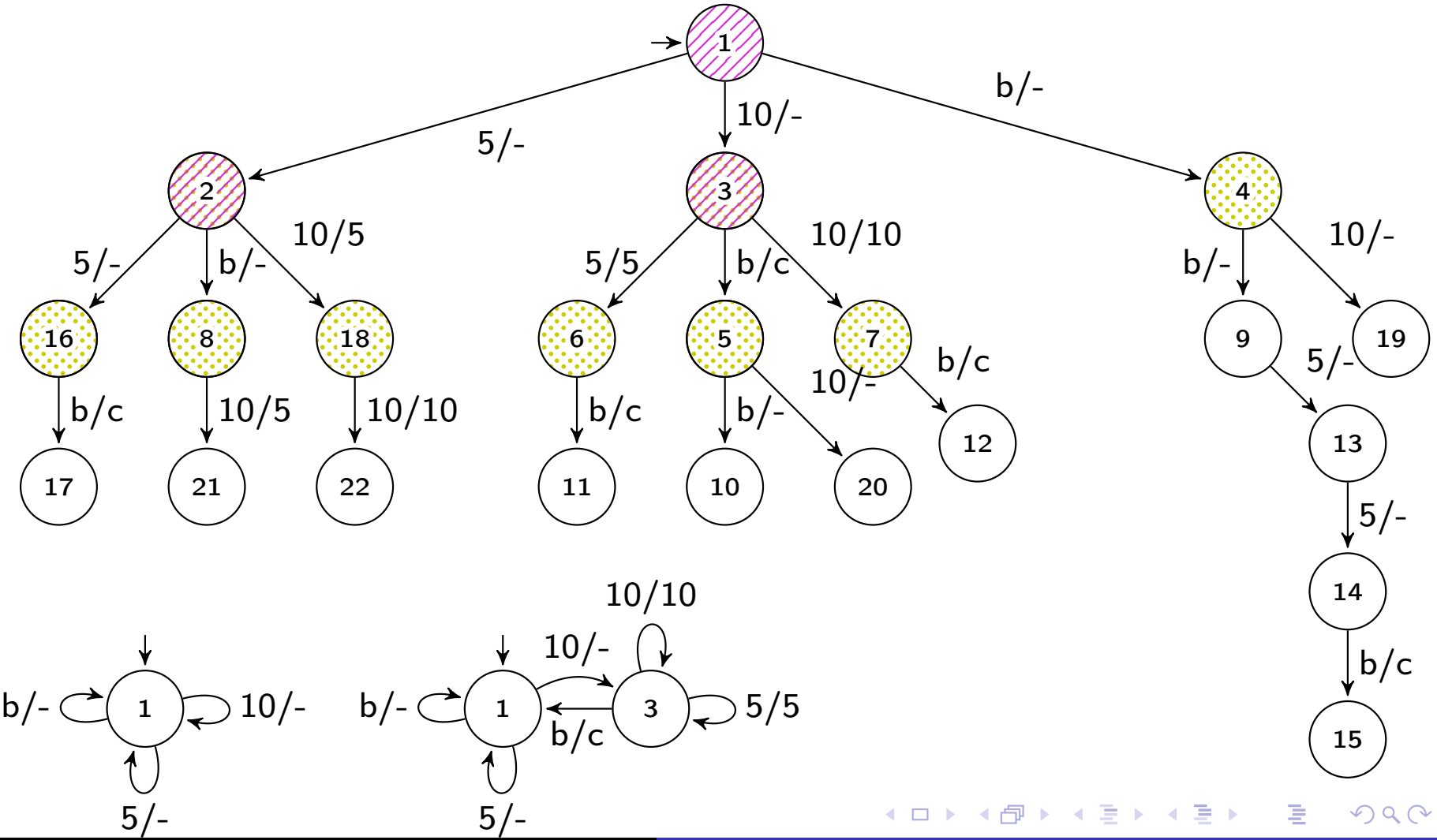




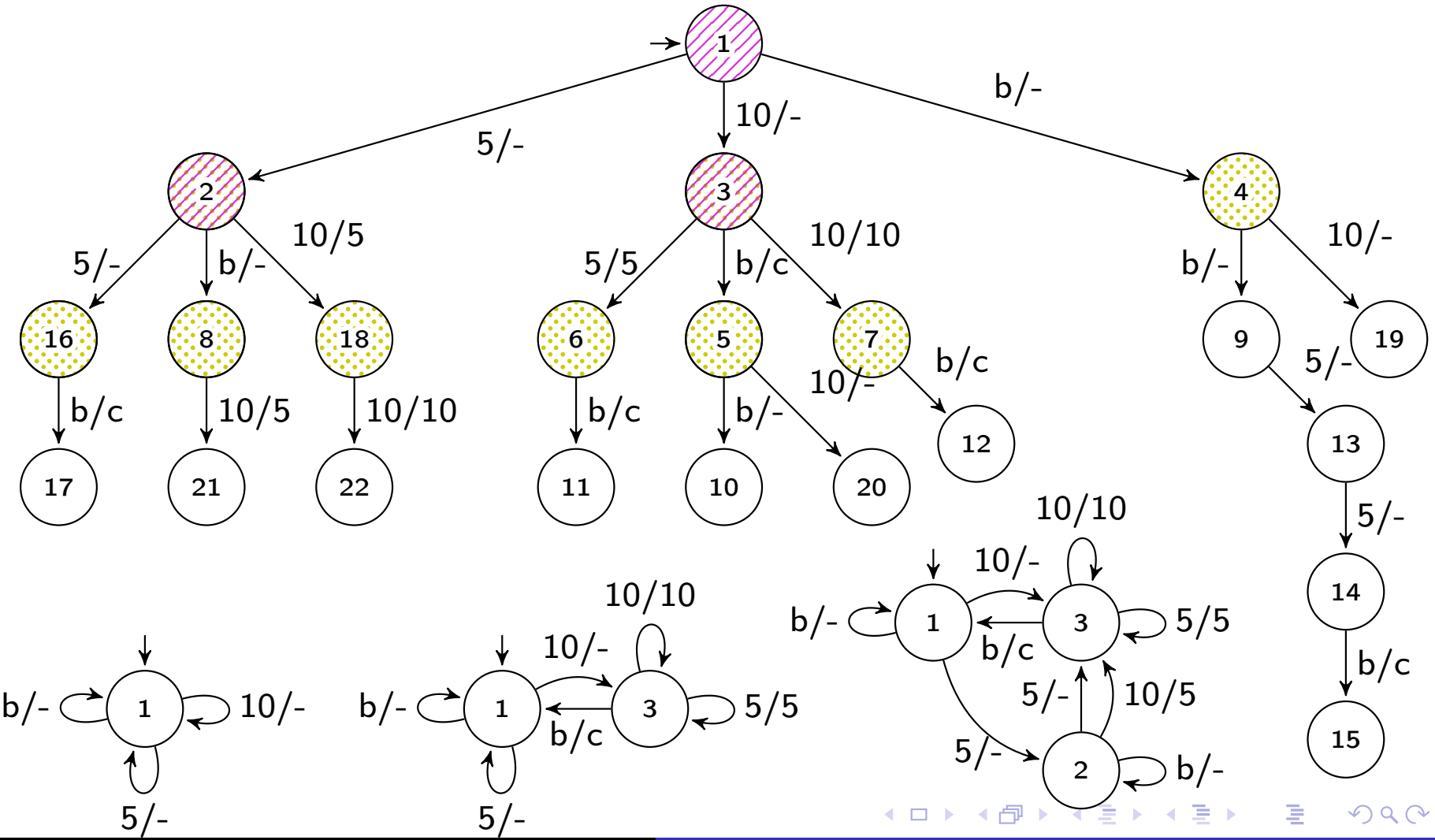
Learning Coffee Machine Model with $L^\#$



Learning Coffee Machine Model with $L^\#$



Learning Coffee Machine Model with $L^\#$



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

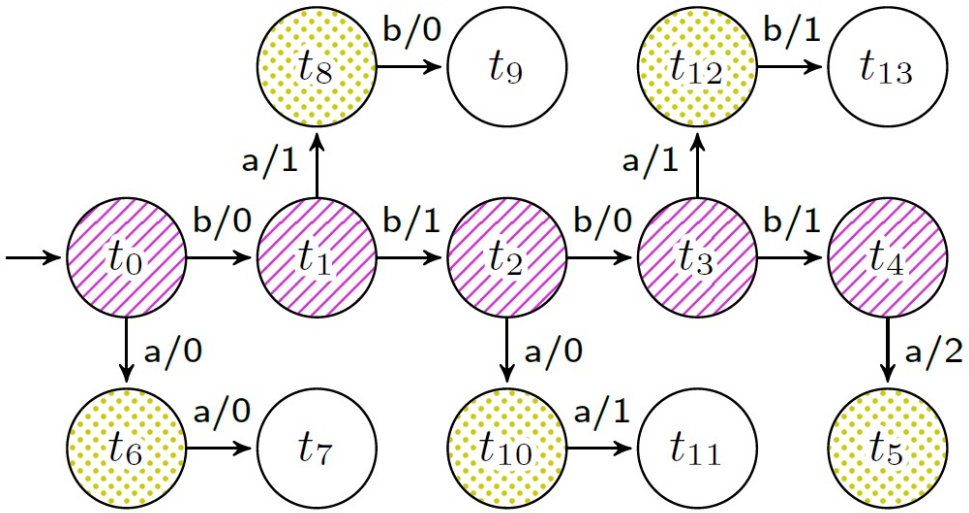
Strategic $L^\#$ is the special case of $L^\#$ where rule (R4) is only applied if none of the other rules is applicable.

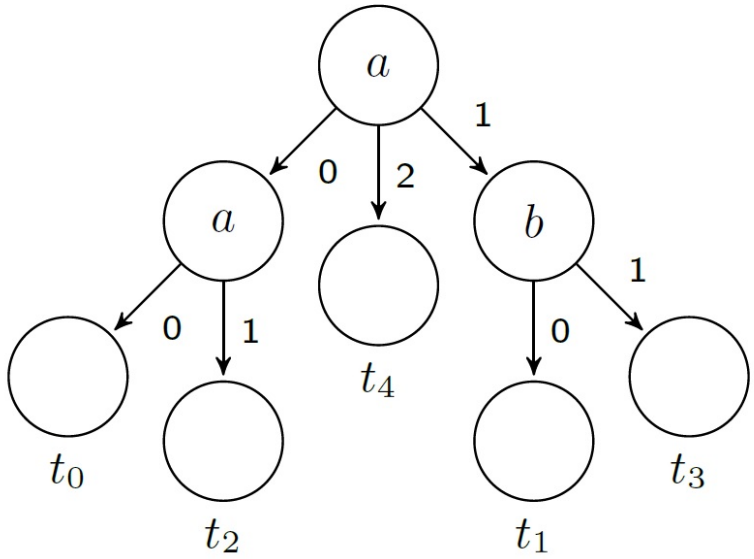
Theorem

Suppose there are k inputs, Mealy machine $\mathcal{M}$ has n states, and the length of the maximal counterexample is m. Then strategic $L^\#$ learns $\mathcal{M}$ with $\mathcal{O}(kn^2 + n \log m)$ output queries and at most $n - 1$ equivalence queries.

Adaptive Distinguishing Sequences

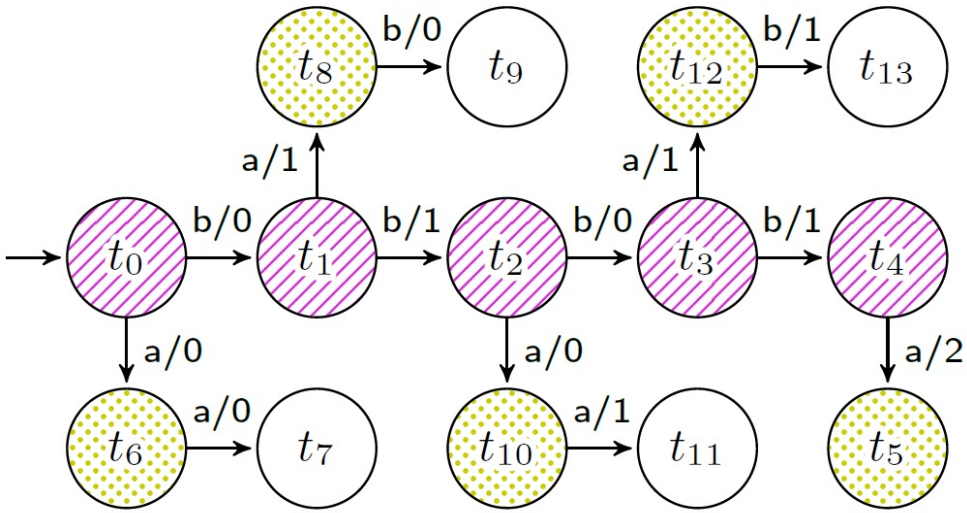
We may speed up identification of frontier states in practice using adaptive distinguishing sequences (ADSs):

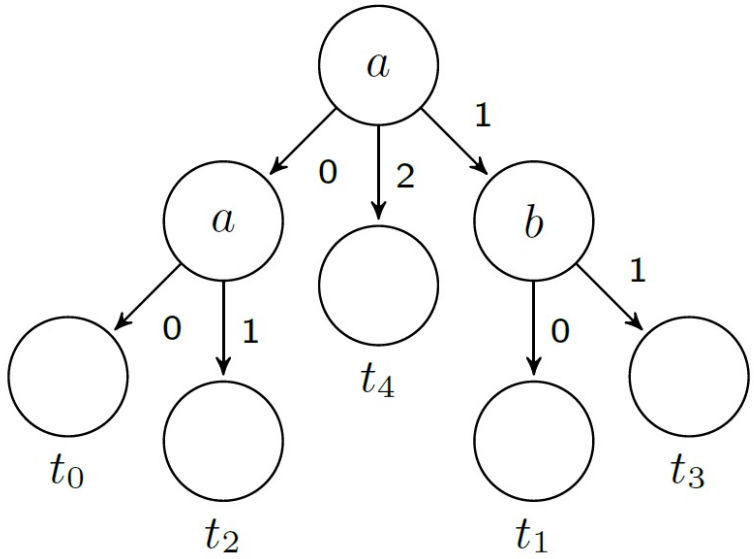




Adaptive Distinguishing Sequences

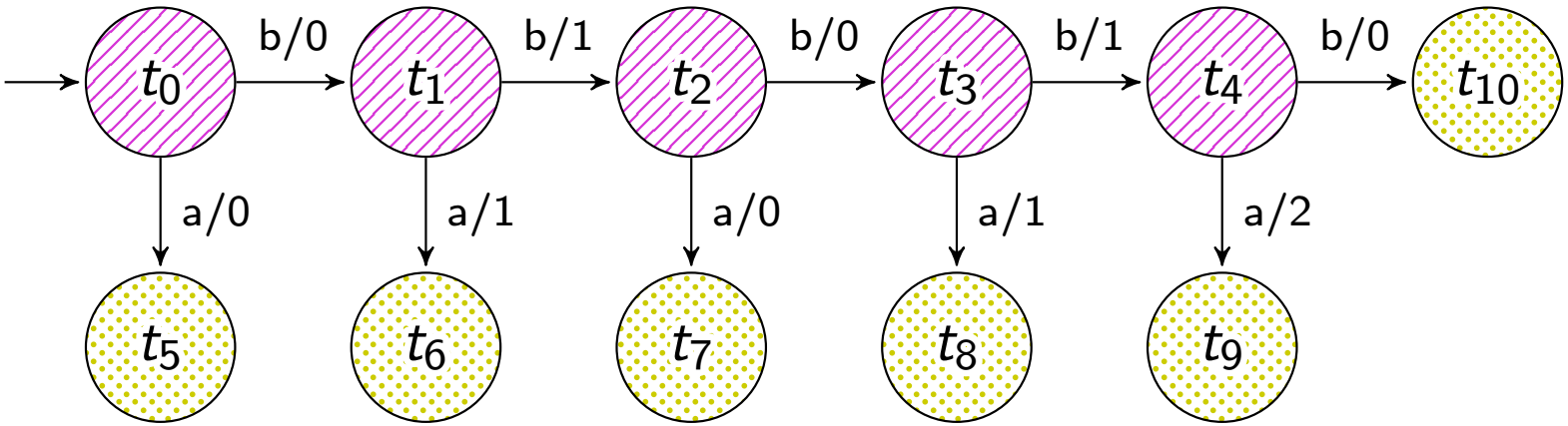
We may speed up identification of frontier states in practice using adaptive distinguishing sequences (ADSs):





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

Adaptive Distinguishing Sequences



Assume a frontier state is equivalent to one of the five basis states, with each basis state equally likely. The we may expect

- input a leads to $\frac{2\times3+2\times3+1\times4}{5} = 3.2$ new apartness pairs
- input b leads to $\frac{3\times2+2\times3}{5} = 2.4$ new apartness pairs

So a is the better choice! We apply this idea recursively.

Computing Adaptive Distinguishing Sequences

Function E 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

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

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

$L^\#$ algorithm with ADS

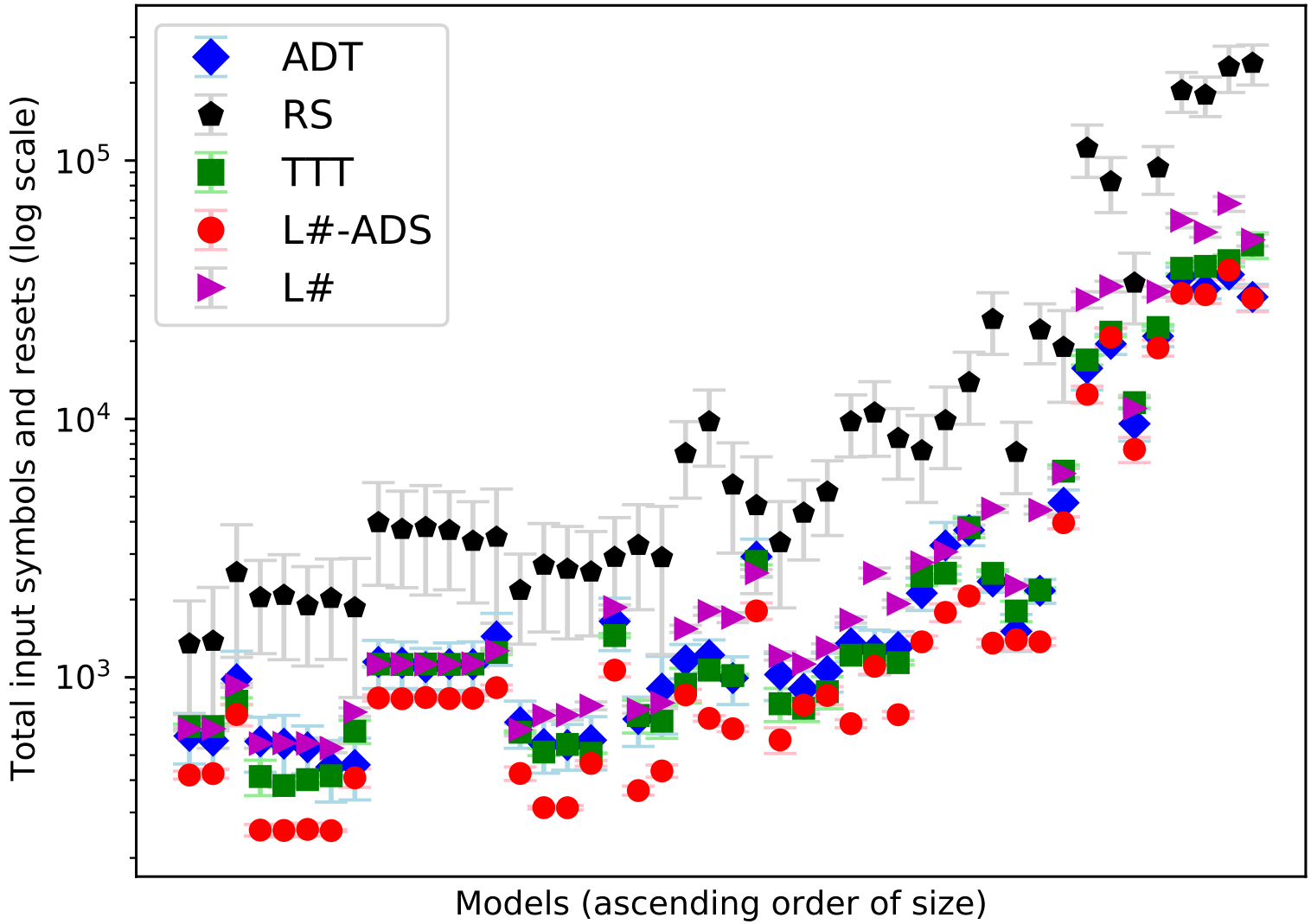
Algorithm 4 $L^\#$ algorithm with adaptive distinguishing sequences

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

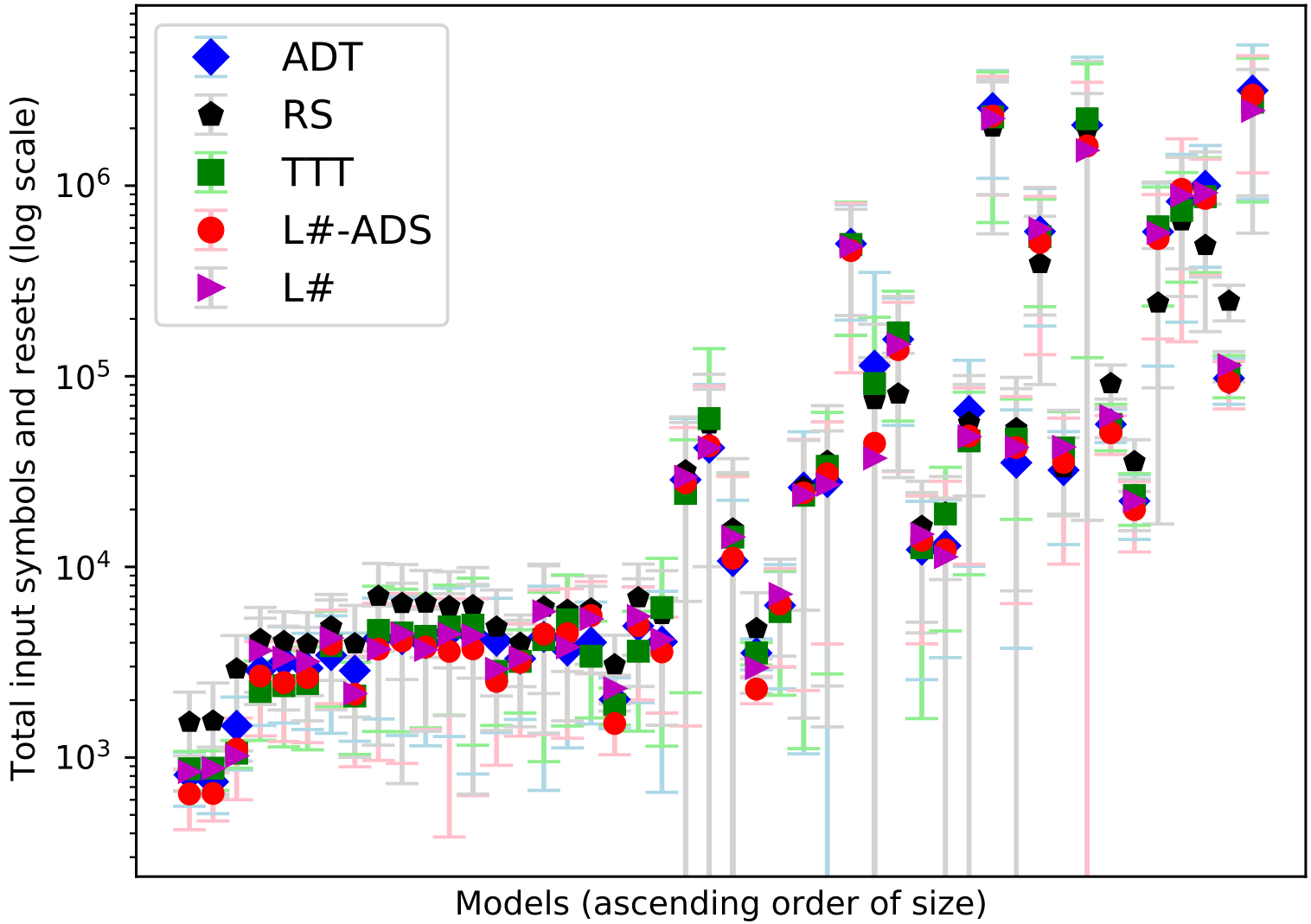
Experimental Evaluation

- 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 BankCard models.
- Largest benchmark has 66 states and 13 input symbols.

Results: Symbols Needed for Learning



Results: Symbols Needed for Learning and Testing



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 learning and testing
4. Integrate active and passive learning
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.