FSM Learning

Radek Mařík

Czech Technical University
Faculty of Electrical Engineering
Department of Telecommunication Engineering
Prague CZ

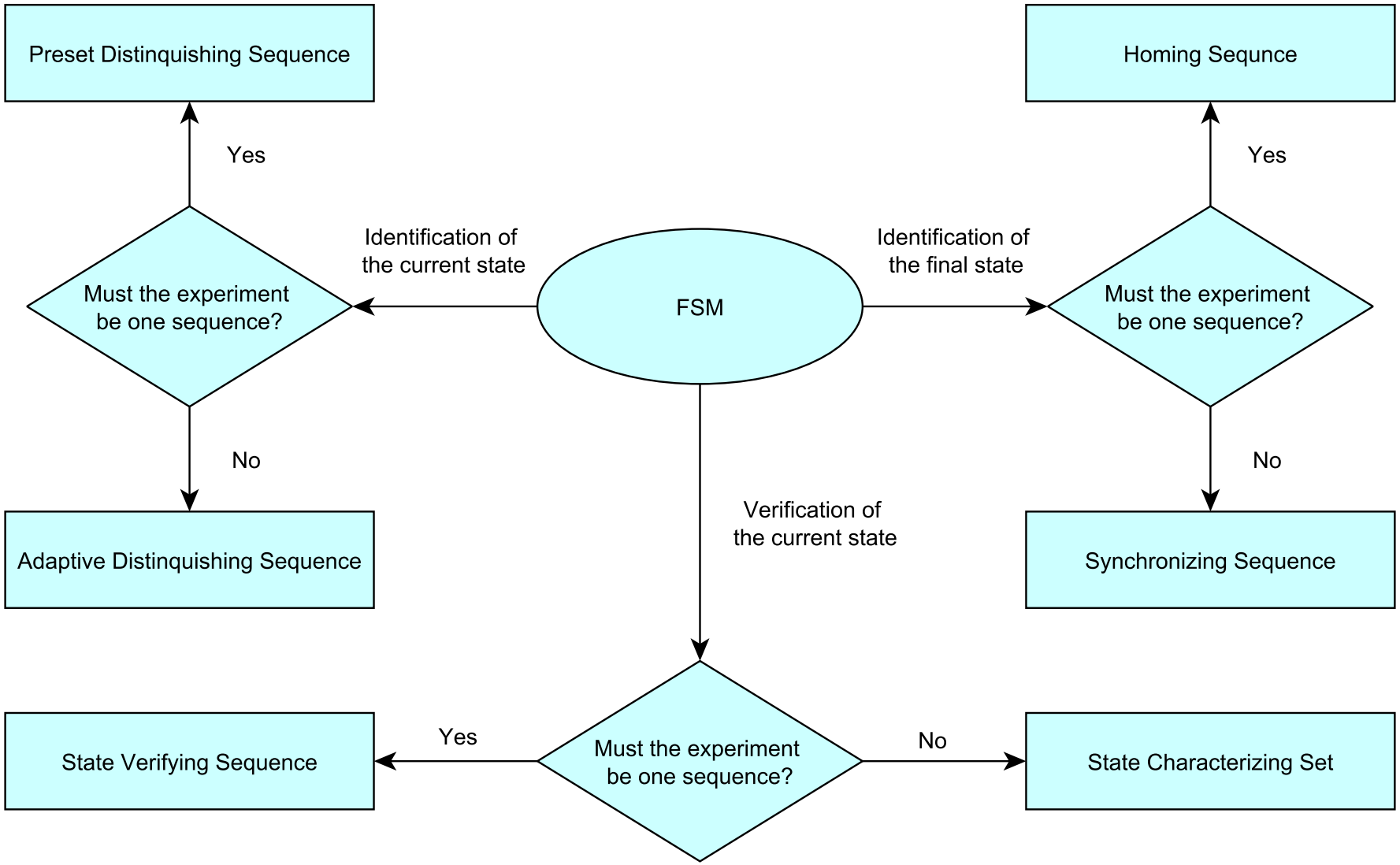
December 19, 2017

Outline

State Identification vs. State Verification

- A **state identification** sequence determines the initial state from which the sequence was applied if a representation of FSM is known.
  - It also finds out the final state.
  - Identification is usually based on a response of the machine, but some sequences are able to determine the final state regardless of the output.
- A **state verification** sequence verifies that the FSM was in a particular initial state which was not known before the experiment is performed.
  - This can be achieved only by observing output and a representation of FSM must be known.

FSM Sequences - Overview [Sou14]



Distinguishing Sequence [Sou14]

Definition 1.1

A **distinguishing sequence** (DS) is an input sequence which distinguishes any two states according to the observed output.

- The application of a DS in each state provides no two identical output sequences.
- The final state is known after applying the DS.
- A distinguishing sequence is one of state identification sequences and also one of state verification sequences.
- If DS is applied in an unknown state, this state and also the final state is easily identified by the output.
- If the FSM is assumed to be in a certain state, the response after applying DS verifies whether the assumption was correct.

Preset Distinguishing Sequence [Sou14]

Definition 1.2

A **preset distinguishing sequence** (PDS) (CZ přednastavená rozlišující sekvence) for a machine is an input sequence x such that the output sequence produced by the machine in response to x is different for each initial state, i.e., $\lambda^*(s_i, x) \neq \lambda^*(s_j, x)$ for every pair of states s_i, s_j , $i \neq j$.

- The distinguishing sequences can be determined from a distinguishing tree.
- A **distinguishing tree** is a successor tree from which all minimal length distinguishing sequences can be derived.

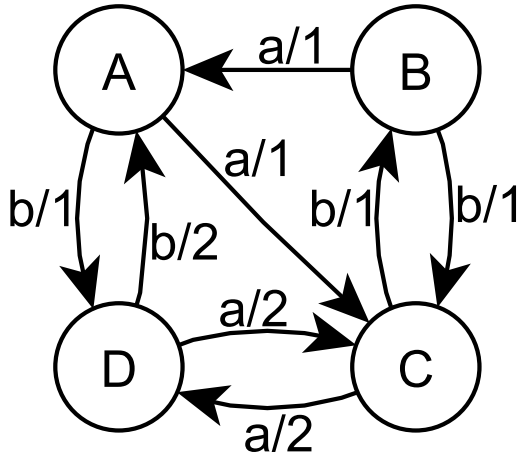
PDS algorithm I [DH94, Sou14]

1. The distinguishing tree has an root node labeled with the set Q of all states of the machine.
2. For each input $a \in \Sigma$, construct a branch from Q to a successor node which represents the set of all next states if the present state is in Q and the input a is applied. Group these states according to the outputs $d \in \Gamma$ associated with the transition to the states.
   Each such group corresponds to the possible next states caused by transitions from Q with input a and output d.
3. Determine terminal nodes of the tree according to the following rules:
   a. A node in which a state appears more than once in a group is a terminal node.
   b. A node which is identical to a node at an earlier level is a terminal node. Note that only groups that are formed by more than a single state should be compared.
   c. A node in which each group consists of a just single state is a terminal node.

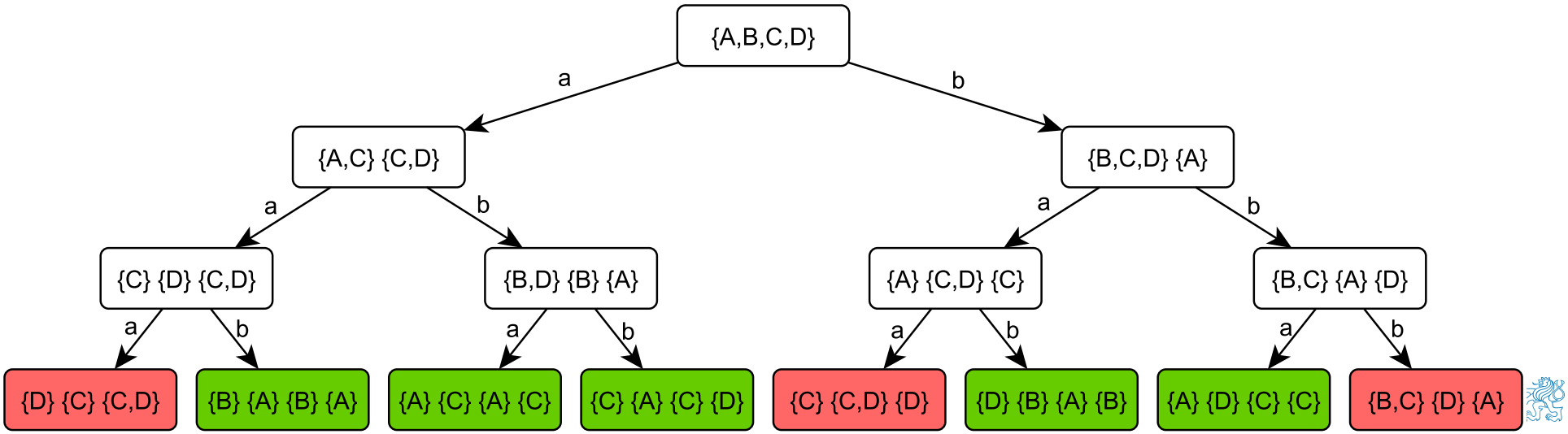
PDS algorithm II [DH94, Sou14]

4. If one or more nodes are terminal nodes defined by rule c) of step 3, the sequence of inputs corresponding to a path from the root node to such a terminal node is a distinguishing sequence for the machine. If all nodes terminate by rule a) or rule b), then the machine has no distinguishing sequence. If there are some nonterminal nodes in the tree, go to step 5.
5. For each nonterminal node Q_i and each input $a \in \Sigma$, construct a branch from Q_i to a successor node representing the next states of Q_i for input a. Group these states according to outputs, as in step 2, but do not group together any states generated by different subgroups of Q_i. Go to step 3.

PDS Example [Sou14]

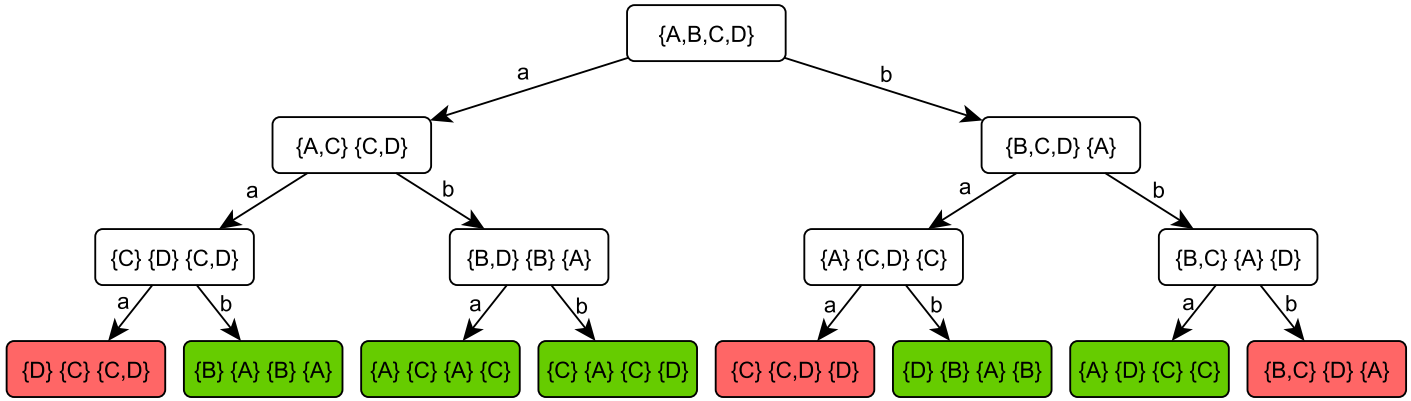


	a	b
A	C / 1	D / 1
B	A / 1	C / 1
C	D / 2	B / 1
D	C / 2	A / 2



PDS Example Sequences [Sou14]

	a	b
A	C / 1	D / 1
B	A / 1	C / 1
C	D / 2	B / 1
D	C / 2	A / 2



	aab	aba	abb	bab	bba
A	122	111	111	121	121
B	111	112	112	122	111
C	221	221	221	111	112
D	222	211	211	211	212

State Verifying Sequence [LY94, Sou14]

- A state verifying sequence (SVS) is also called *simple I/O sequence* [Hsi71] or *Unique Input Output sequence* [LY94].

Definition 1.3

A **state verifying sequence** of a state $s \in S$ is an input sequence $x \in \Sigma^*$, such that the output sequence produced by the machine in response to x from any state other than s is different than that from s, i.e., $\lambda^*(s_i, x) \neq \lambda^*(s, x)$ for any $s_i \neq s$.

- When FSM could be in any particular state, just the SVS of this state is applied. Then the observed output sequence determines whether FSM was in the expected state or not.
- A union of state verifying sequences of all states in FSM is called **states verifying set** (SVSet) of FSM.
- A state does not have to have SVS and so SVSet is not defined.

SVS Algorithm I [Sou14]

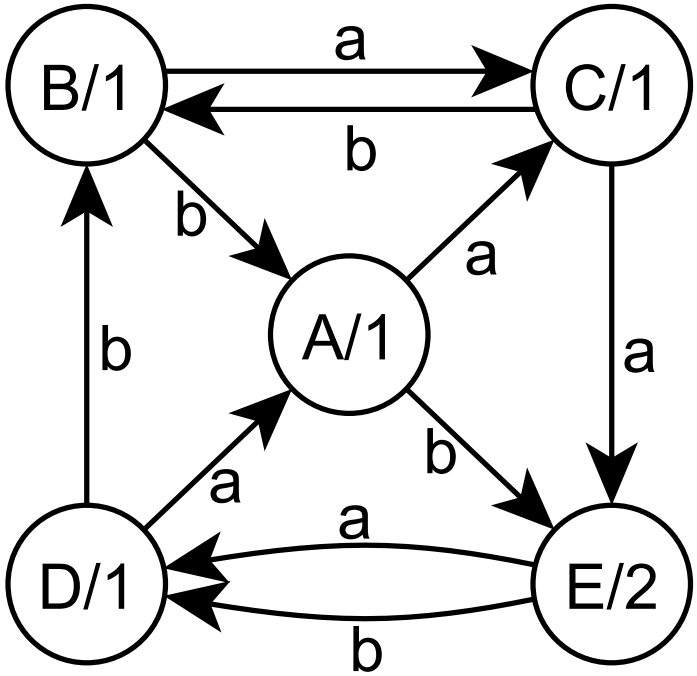
1. The state distinguishing tree of state $s \in S$ has an initial node labeled with the set Q of all states of the machine and state $s \in Q$ is highlighted. For Moore machines the set Q contains only states with the same output symbol on ϵ as state s has.
2. For each input $a \in \Sigma$, construct a branch from Q to a successor node which represents the set of all next states if the state is in Q and the input a is applied and the output is equal to the output of transition from the fixed state $s \in Q$. In each successor node highlight the next state of the fixed state s.
3. Determine terminal nodes of the tree according to the following rules. Note that the rules are listed in order of their testing.
   a. A node in which the highlighted state s appears more than once in the label is a terminal node.
   b. A node which is identical to a node at an earlier level is a terminal node. Note that also highlighted states must be the same.
   c. A node with the fixed state s only is a terminal node.

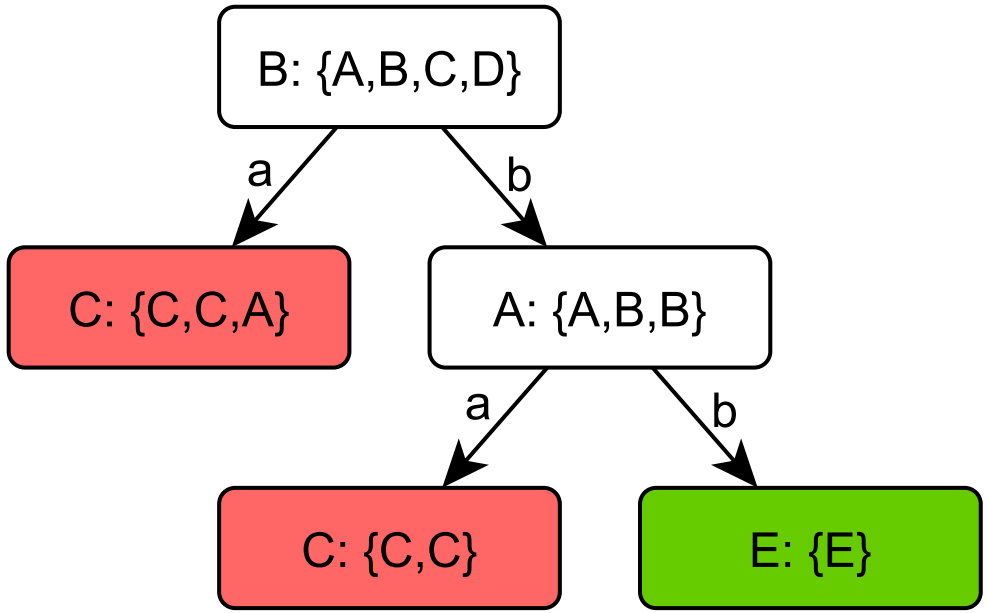
SVS Algorithm II [Sou14]

4. If one or more nodes are terminal nodes defined by rule c) of step 3, the sequence of inputs corresponding to a path from the initial node to such a terminal node is a state verifying sequence for the fixed state s of the initial node.
   If all nodes terminate by rule a) or rule b), then the machine has no state verifying sequence for the state s.
   If there are some nonterminal nodes in the tree, go to step 5.
5. For each nonterminal node Q_i and each input $a \in \Sigma$, construct a branch from Q_i to a successor node representing the next states of Q_i on input a.
   Eliminate states with different outputs than the fixed state $s \in Q_i$ has on input a.
   Highlight a next state of s in each successor node, i.e. if s is fixed then $\delta(s, a)$ is fixed.
   Go to step 3.

SVS Example - Moore Machine [Sou14]

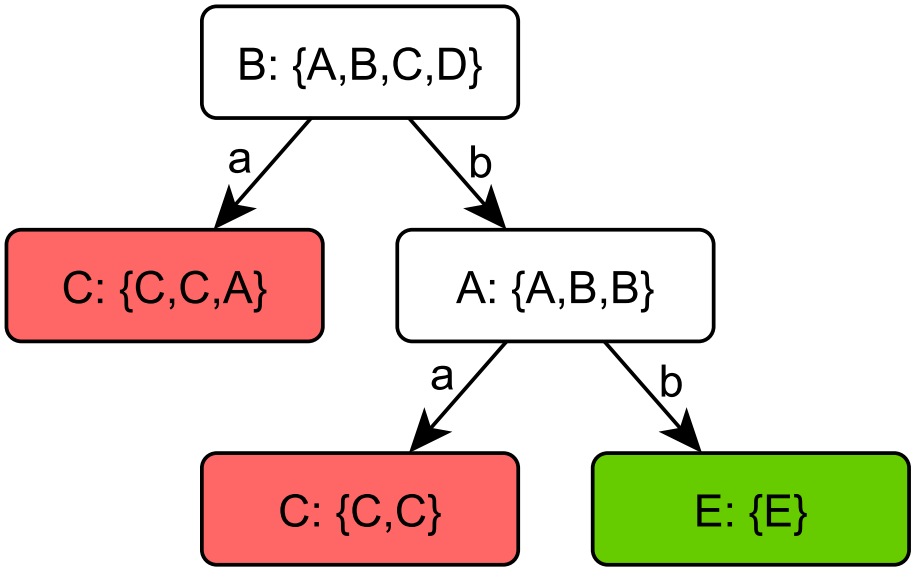
	a	b	ϵ
A	C	E	1
B	C	A	1
C	E	B	1
D	A	B	1
E	D	D	2





SVS Example Sequences [Sou14]

	a	b	ϵ
A	C	E	1
B	C	A	1
C	E	B	1
D	A	B	1
E	D	D	2



	ϵb	ϵbb	ϵa	ϵaa	ϵ
A	*12*	121	11	112	1
B	11	*112*	11	112	1
C	11	111	*12*	121	1
D	11	111	11	*111*	1
E	21	211	21	211	*2*

State Characterizing Set [Sou14]

- For all states different from s there exists an input sequence x_k, such that the output sequences produced by the machine in response to x_k is different, i.e., $\forall s_i \neq s \; \exists x_k \in \Lambda_s : \lambda^*(s_i, x_k) \neq \lambda^*(s, x_k)$.

Definition 1.4

A **state characterizing set** Λ_s of a state $s \in S$ is a set of input sequences $x_k \in \Sigma^*$, such that the set of output sequences produced by the machine in response to all x_k from any state other than s is different than that from s, i.e., $\{\lambda^*(s_i, x_k) \,|\, x_k \in \Lambda_s\} \neq \{\lambda^*(s, x_k) \,|\, x_k \in \Lambda_s\}$ for any $s_i \neq s$.

- Each state of a reduced FSM has a state characterizing set (SCSet).
- Minimization of number of sequences in a state characterizing set can be proved to be NP-hard[HMU06] because Set cover problem which is NP-complete can be reduced polynomially to the minimization of sequence number problem.

Characterization Set [Sou14]

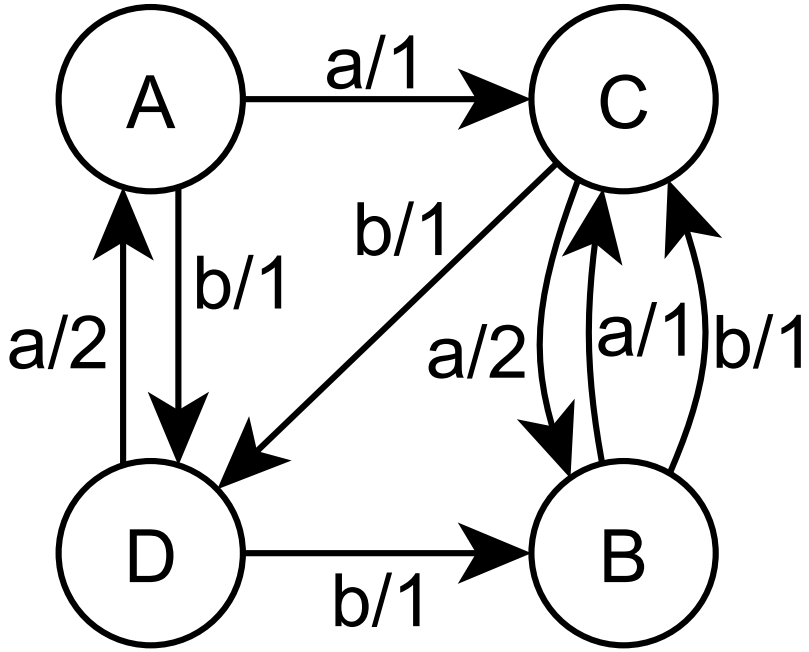
- A **characterizing set** W is a set of input sequences $x_k \in \Sigma^*$, such that for each pair of states $(s_i, s_j), s_i \neq s_j$, there is sequence $x_k \in W$ that distinguishes this pair, i.e., $\lambda^*(s_i, x_k) \neq \lambda^*(s_j, x_k)$.
- W is also known as **characterization set** [Cho78].
- The characterizing set (CSet) can be obtained as union of SCSets of all states

SCSet Algorithm [Sou14]

1. For each pair of states $(s_i, s_j) \in S \times S$ apply each input $a \in \Sigma$. If on some a states s_i and s_j produce different output, store this input a as the shortest distinguishing sequence for pair (s_i, s_j). For Moore machines distinguish pairs of states first by the empty string ϵ.
2. For each pair (s_i, s_j) distinguished by the input sequence w in the previous step try to find an undistinguished pair (s_k, s_l) and an input a such that the pair of the next states $(\delta(s_k, a), \delta(s_l, a)) = (s_i, s_j)$. If there are such pair and input, store $a \cdot w$ as a distinguishing sequence for the pair(s_k, s_l)
3. If some pair of different states is undistinguished, go to step 2. Otherwise create a characterizing set as a set of all distinguishing sequences stored in the previous steps. A state characterizing set of state s is a set of all sequences distinguished pairs (s, s_k), where $s_k \neq s$, i.e., $\{w_k \in \Sigma^* \,|w_k$ distinguishes $(s, s_k) \in S \times S, s \neq s_k\}$.

SCSet Example - Mealy Machine [Sou14]

	a	b
A	C / 1	D / 1
B	C / 1	C / 1
C	B / 2	D / 1
D	A / 2	B / 1



	A	B	C	D	SCSet
A	–	bba	a	a	{a, bba}
B	bba	–	a	a	{a, bba}
C	a	a	–	ba	{a, ba}
D	a	a	ba	–	{a, ba}

Homing Sequence [DH94, Sou14]

- A homing sequence (HS) guides FSM to some specific states.

Definition 1.5

An input sequence x is said to be a **homing sequence** if the final state of the machine can be determined uniquely from the machine's response to x, regardless of the initial state. These final states of the machine are determined by observing the output sequence produced by applying a homing sequence to the machine.

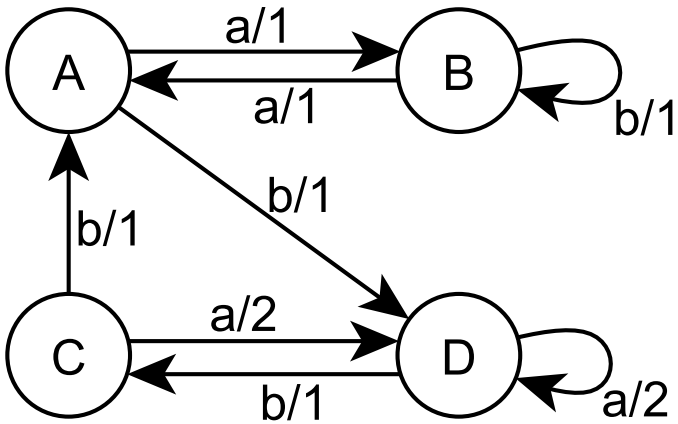
- A homing sequence exists for all reduced FSM.
- If the current state of FSM is unknown, HS is applied and according to the output sequence a final state is determined.
- An adaptive form of sequence can rapidly reduce the length of homing sequence in some cases.

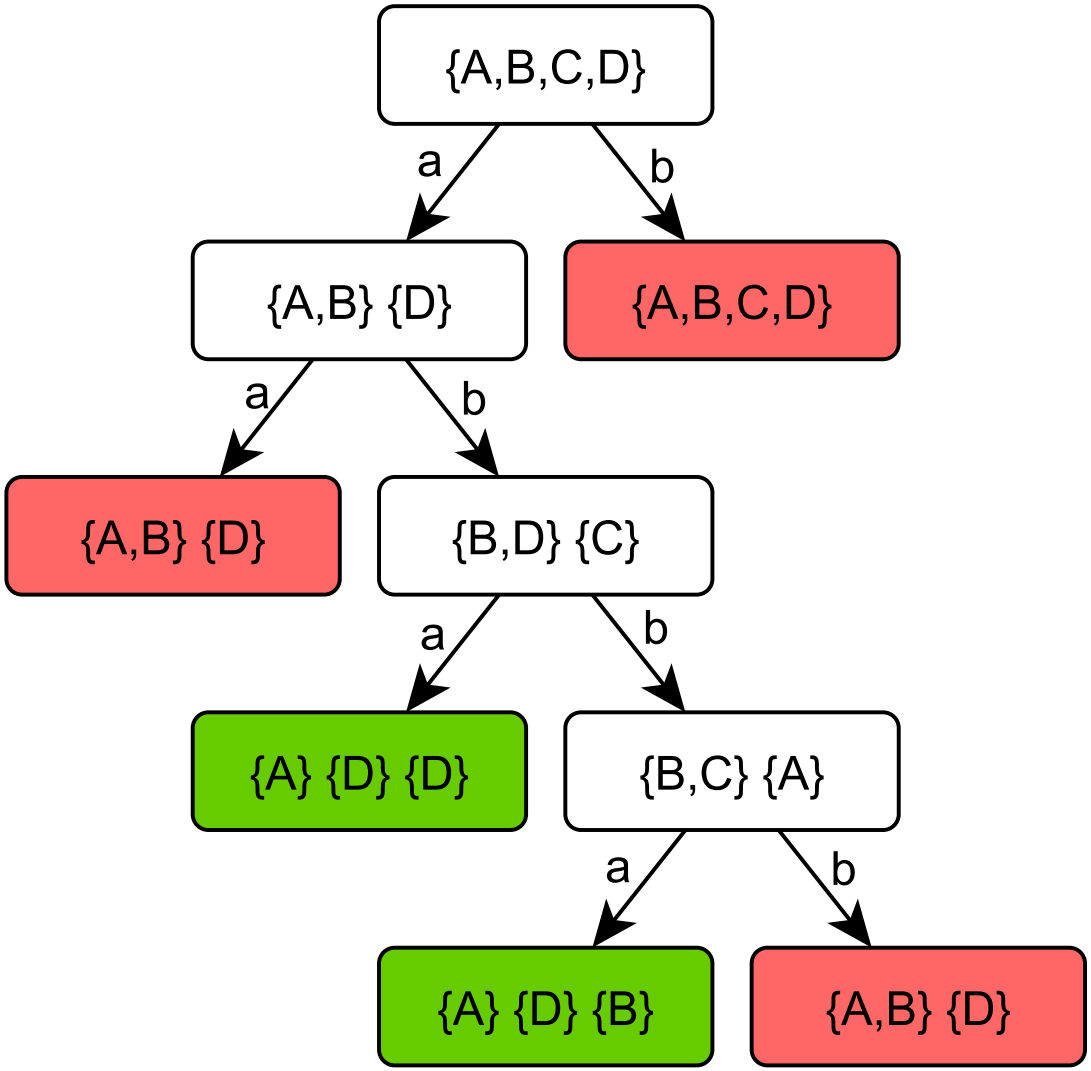
HS algorithm [DH94, Sou14]

1. The homing tree has an root node labeled with the set Q of all states of the machine.
2. For each input $a \in \Sigma$, construct a branch from Q to a successor node which represents the set of all next states, if the present state is in Q and the input a is applied. Group these nodes according to outputs associated with the transitions to the states. Within any group, no state need be repeated.
3. Determine terminal nodes in the tree according to the following rules:
   a. A node which is identical to a node at an earlier level is a terminal node.
   b. A node in which each group is a single state is a terminal node.
4. If one or more nodes are terminal nodes by rule b), a sequence of inputs from the root node to such a terminal node is a homing sequence. Note that all nodes cannot be terminal by rule a) since a homing sequence always exists. If there are some nonterminal nodes in the tree, go to step 5.
5. For each nonterminal node Q_i and each input a, construct a branch from Q_i to a successor node, representing the next state of Q_i for input a, grouping them by outputs and not grouping together states that are generated by different subgroups of Q_i. Go to step 3.

HS Example - Mealy Machine [Sou14]

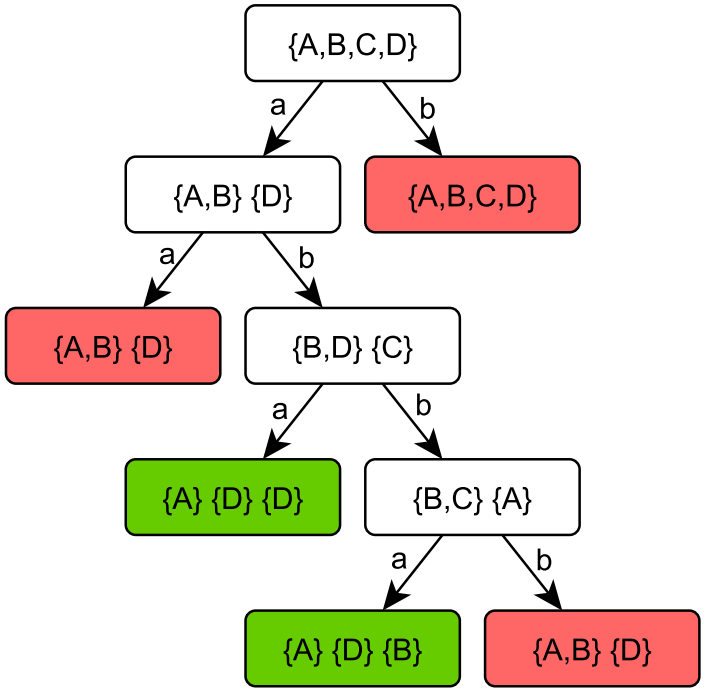
	a	b
A	B / 1	D / 1
B	A / 1	B / 1
C	D / 2	A / 1
D	D / 2	C / 1





HS Example Sequences [Sou14]

	a	b
A	B / 1	D / 1
B	A / 1	B / 1
C	D / 2	A / 1
D	D / 2	C / 1



initial state	response to aba	final state	response to $abba$	final state
A	111	A	1111	A
B	112	D	1112	D
C	212	D	2111	B
D	212	D	2111	B

Synchronizing Sequence [DH94, Sou14]

- Some FSM can be synchronized to a particular state which means that FSM is in this state after applying a specific input sequence.
- The input sequence is called a synchronizing sequence (SS).

Definition 1.6

A **synchronizing sequence** is an input sequence which takes the machine to a unique final state independent of its initial state.

- This sequence has not an adaptive form because the decision is made regardless of the output.
- It is guaranteed that SS takes FSM into one state unlike HS which can take machine into more than one final state.
- SS has always at least the same length as HS
- FSM may not even have an SS.
- When output of FSM cannot be observed but a representation of FSM is known SS still can be used to determine the current state.

SS algorithm I [DH94, Sou14]

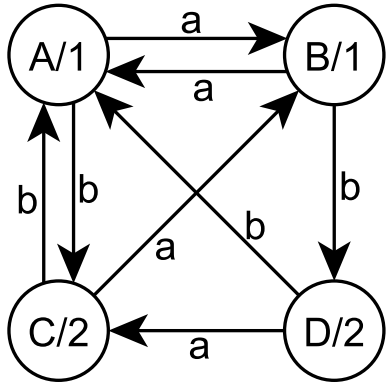
- The synchronizing sequences can be found from a synchronizing tree which is a successor tree.
- The synchronizing tree is similar to the homing tree except that the states represented by a node are not grouped according to outputs since the final state must be determined independently of the output.
- The following steps are followed to build a synchronizing tree and derive all minimal length synchronizing sequences.

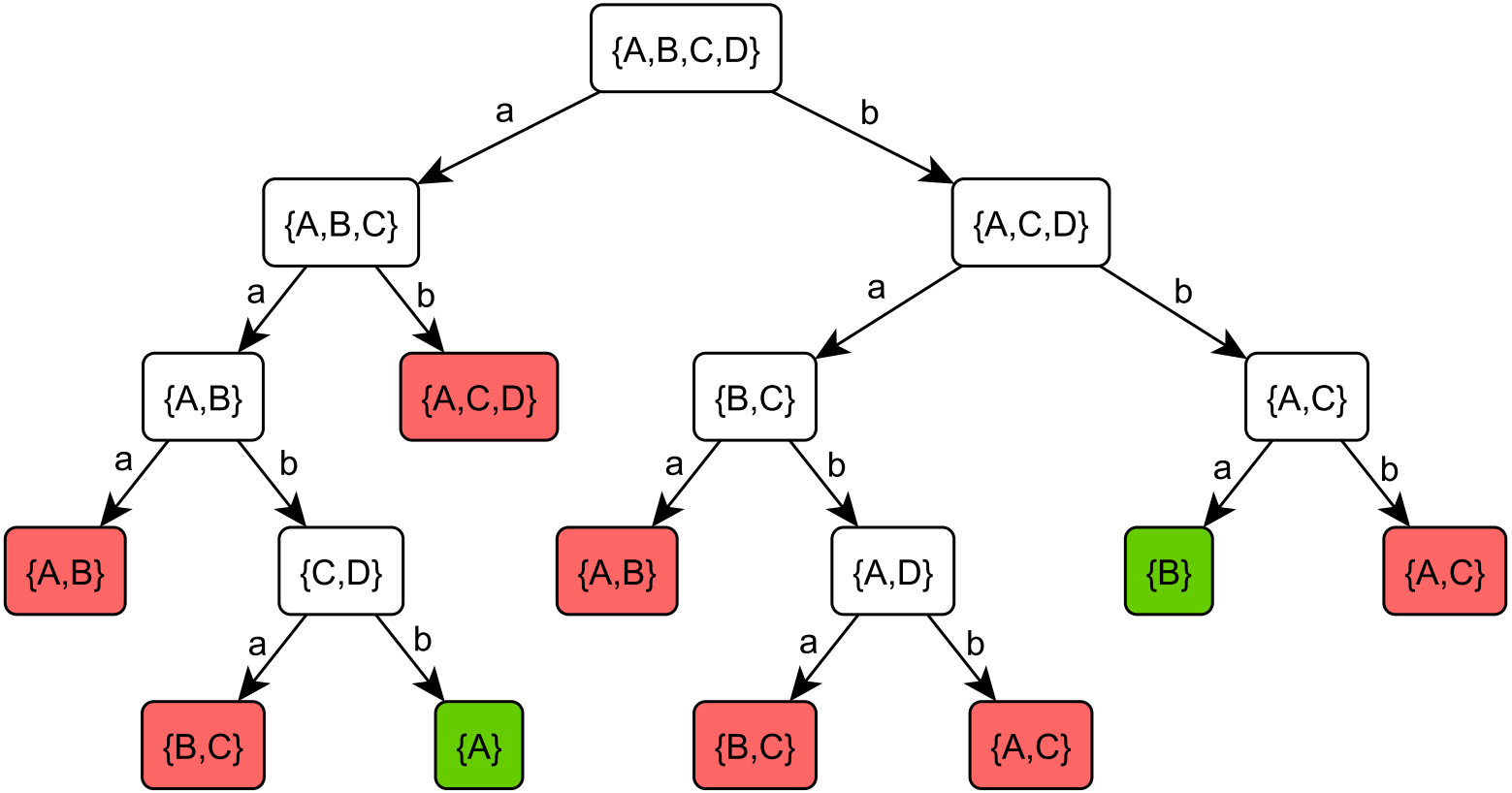
SS algorithm II [DH94, Sou14]

1. The synchronizing tree has an root node labeled with the set Q of all machine states.
2. For each input $a \in \Sigma$, construct a branch from Q to a successor node which represents the set of all next states, if the current state is in Q and the input a is applied. Group these nodes disregarding the outputs associated with the transition to the states. Within the group, no state need to be repeated.
3. Terminal nodes in the tree are determined according to the following rules:
   a. A node which is identical to a node at an earlier level is a terminal node.
   b. A node in which the group is a single state is a terminal node.
4. If one or more nodes are terminal nodes by rule b), the sequence of inputs from the root node to such a terminal node is a synchronizing sequence. If all nodes are terminated by rule a), the machine has no synchronizing sequence. If there are some nonterminal nodes in the tree, go to step 5.
5. For each nonterminal node Q_i and each input a, construct a branch from Q_i to a successor node, representing the next states of Q_i for input a. Go to step 3.

SS Example - Moore Machine [Sou14]

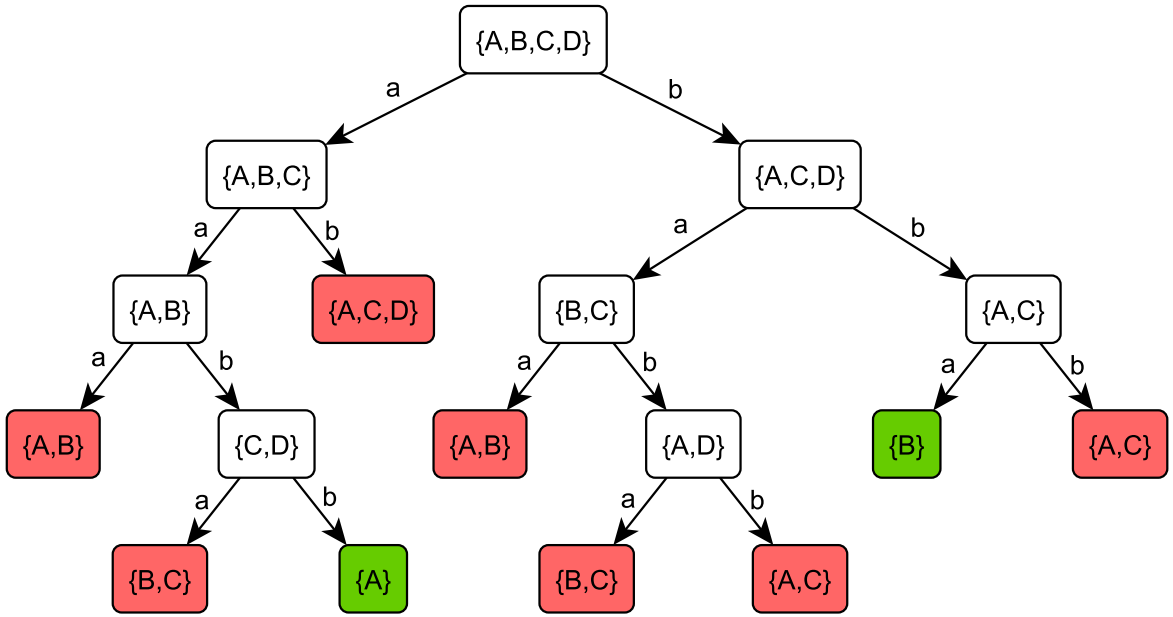
	a	b	ϵ
A	B	C	1
B	A	D	1
C	B	A	2
D	C	A	2





SS Example Sequences [Sou14]

	a	b	ϵ
A	B	C	1
B	A	D	1
C	B	A	2
D	C	A	2

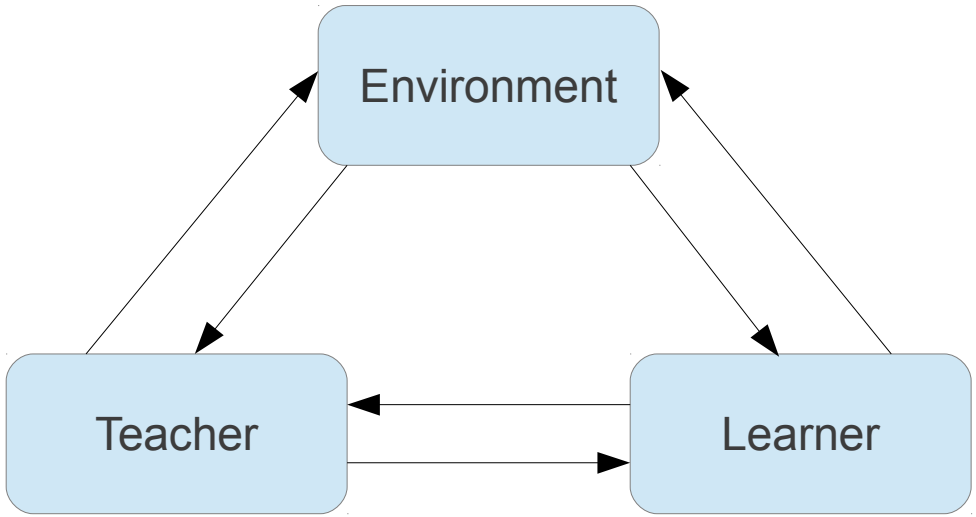


initial state	response to bba	final state	response to $aabb$	final state
A	211	B	1121	A
B	211	B	1121	A
C	121	B	1121	A
D	121	B	2121	A

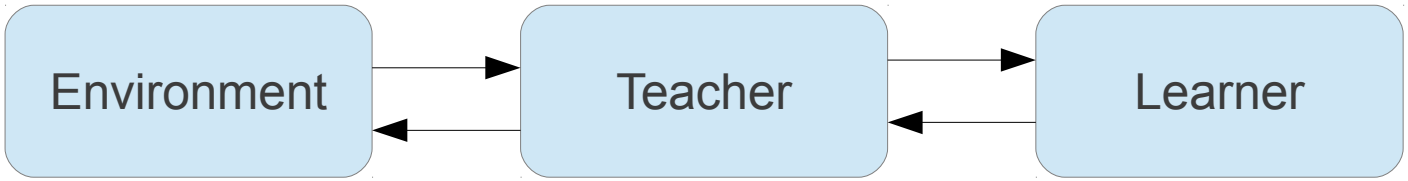
Aktivní učení [Hon13]

- **Pasivní učení** - množina X je dána a nemůžeme ji jakkoliv upravovat,
- **Aktivní učení** - množinu X si můžeme zvolit a v průběhu procesu učení dále upravovat.

Aktivní učení s učitelem [Hon13]



Architektura učení s minimálním adekvátním učitelem.



Architektura s degradovaným učitelem pracujícím jako rozhraní.

Tabulka pozorování [Ang86, Sha08, Hon13]

Definition 2.1

Nechť $\mathcal{E} = (A, \text{accept})$ je akceptační prostředí.
Tabulkou pozorování prostředí $\mathcal{E}$ nazýváme uspořádanou trojici $OT = (S, E, \mathrm{T})$, kde

- $S \subseteq A^*$, $S \neq \emptyset$, S konečná, S je uzavřená na prefixy,
- $E \subseteq A^*$, $E \neq \emptyset$, E konečná, E je uzavřená na sufixy,
- T je funkce $(S \cup S \cdot A) \times E \to \{0, 1\}$.

- Množinu S nazýváme *vstupní množinou*.
- E *rozlišovací množinou*.

Počáteční tabulku pozorování [Hon13]

- V L^* algoritmu nejprve inicializujeme počáteční tabulku pozorování $OT = (S, E, \mathrm{T})$ tak, že $S = \{\epsilon\}$, $E = \{\epsilon\}$.
- Dále vytvoříme *frontu otázek příslušnosti*, kterou tvoří všechny dvojice $s \cdot e$, kde $s \in S \cup S \cdot A$ a $e \in E$.
- Pomocí učitele dostane odpověď z množiny $\{0, 1\}$, zda-li $s \cdot e$ patří do rozeznávaného jazyka a tuto hodnotu uložíme na místo $\mathrm{T}(s, e)$ v tabulce pozorování.
- Odlišné řádky v sekci S tabulky definují stavy možného automatu.

		E
		ϵ
S	ϵ	1
$S \cdot A$	a	0
	b	0

Tabulku pozorování - uzavřenost, konzistence [Hon13]

Definition 2.2

Říkáme, že **tabulka pozorování** $OT = (S, E, \mathrm{T})$ **je uzavřena**, pokud $(\forall t \in S \cdot A)(\exists s \in S)(s \overset{E}{\sim} t)$.
Dále říkáme, že **tabulka je konzistentní**, pokud $(\forall s, t \in S, s \overset{E}{\sim} t) \implies (\forall a \in A)(s \cdot a \overset{E}{\sim} t \cdot a)$.

- Kontrolu uzavřenosti a konzistence provádíme po vyprázdnění fronty otázek příslušnosti.

Tabulku pozorování - modifikace [Hon13]

- Pokud není $OT = (S, E, \mathrm{T})$ uzavřená, pak
  1. najdeme $t \in S \cdot A$, že $s \overset{E}{\not\sim} t$ pro všechna $s \in S$.
  2. toto t pak přidáme do množiny S a frontu otázek příslušnosti rozšíříme o $t \cdot a \cdot e$ pro všechna $a \in A$ a $e \in E$.
- Jestliže není OT konzistentní,
  1. najedeme $s, t \in S$, $e \in E$ a $a \in A$, že $s \overset{E}{\sim} t$, ale $T(s \cdot a, e) \neq T(t \cdot a, e)$.
  2. do rozlišovací množiny E přidáme slovo $a \cdot e$
  3. frontu otázek příslušnosti rozšíříme o $s' \cdot e$ pro všechna $s' \in S \cup S \cdot A$.
  4. Je zřejmé, že po tomto zásahu již nebude v nové tabulce pozorování platit $s \overset{E}{\sim} t$.

L^* algoritmus [Ang86, Sha08, Hon13]

1. Inicializace počáteční tabulky pozorování $OT = (S, E, \mathrm{T})$.
2. Pomocí fronty otázek příslušnosti vyplníme celou tabulku pozorování.
3. Kontrola uzavřenosti a konzistence tabulky.
   1. Pokud není OT uzavřená, rozšíříme množinu S o $t \in S \cdot A$, že $s \overset{E}{\not\sim} t$ pro všechna $s \in S$. Rozšíříme frontu otázek příslušnosti a pokračujeme bodem 2.
   2. Pokud není OT konzistentní, rozšíříme množinu E o slovo $a \cdot e$, $e \in E$ a $a \in A$ tak, že existují $s, t \in S$, že $s \overset{E}{\sim} t$, ale $\mathrm{T}(s \cdot a, e) \neq \mathrm{T}(t \cdot a, e)$. Rozšíříme frontu otázek příslušnosti a pokračujeme bodem 2.
4. Vytvoříme návrh $\mathcal{A}$ prostředí a zeptáme se učitele na jeho správnost.
5. Pokud učitel vrátí protipříklad $c \in A^+$, smažeme návrh $\mathcal{A}$, přidáme do množiny S všechny prvky množiny $\mathrm{pref}(c)$, rozšíříme frontu otázek příslušnosti a pokračujeme bodem 2.
6. Návrh $\mathcal{A}$ přijímáme za automat realizující prostředí $\mathcal{E}$.

Literatura I

[Ang86] Dana Angluin. Learning regular sets from queries and counter-examples. Technical Report YALEU/DCS/TR-464, Yale University, Department of Computer Science, March 1986.

[Cho78] Tsun S. Chow. Testing software design modeled by finite-state machines. *Software Engineering, IEEE Transactions on*, (3):178–187, 1978. Test: P.Z, W=characterization set Transfer and operation error, extra state Comparison: branch, switch and boundary-interior covers.

[DH94] RG Deshmukh and GN Hawat. An algorithm to determine shortest length distinguishing, homing, and synchronizing sequences for sequential machines. In *Southcon/94. Conference Record*, pages 496–501. IEEE, 1994. Def and algorithms: DS, HS, SS - as successor tree.

[HMU06] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. *Introduction to Automata Theory, Languages, and Computation*. Prentice Hall, 2006. Def: DFA, (eps-)NFA, RegEx, RegLang, CFL, Pushdown Automata, TM Undecidability, Intractable, P, (Co-)NP, NPC, (N)PSpace(-Complete), RP, ZPP.

[Hon13] Marek Honzírek. Aktivní učení a automaty. Master's thesis, Katedra matematiky, Fakulta jaderná a fyzikálně inženýrská, ČVUT, Praha, 2013.

[Hsi71] E. P. Hsieh. Checking experiments ror sequential machines. *IEEE Transactions on Computers*, C-20(10):1152–1166, Oct 1971.

[LY94] David Lee and Mihalis Yannakakis. Testing finite-state machines: State identification and verification. *Computers, IEEE Transactions on*, 43(3):306–320, 1994. Def: state identification and verification Test whether FSM has a PDS is PSPACE-Complete There are FSM with exponential-long PDS as the shortest one Polynomial existence and constructing algorithm for ADS (based on Hopcroft minimization).

[Sha08] Muzammil Muhammad Shahbaz. *Reverse Engineering Enhanced State Models of Black Box Software Components to support Integration Testing*. PhD thesis, Institut Polytechnique de Grenoble, 2008.

[Sou14] Michal Soucha. Sequences of finite-state machines, BSc. thesis. Master's thesis, Department of Cybernetics, Faculty of Electrical Engineering, CTU, Prague, 2014. Department of Cybernetics, Faculty of Electrical Engineering, CTU, Prague.