FSM Checking Sequences

Radek Mařík

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

December 12, 2017

Outline

Finite Machine in Applications [Bei95, HI98]

- a model for testing of applicaiton driven using menu
- a model of communication protocols
- a model used in object-oriented design

Finite State Machine

- an abstract machine which the number of states and input symbols is finite and constant.
- consists of
  - states (nodes) . . . future behavior is fully determined by a given state,
  - transitions (edges) . . . behavioral rules,
  - input symbols (labels of edges) . . . environmental stimuli, and
  - output symbols (labels of edges or nodes) . . . external reactions.

Finite State Machine [HI98]

- Let $Input$ be a finite alphabet.
- *Konečný stavový automat* nad $Input$ obsahuje následující položky:
  1. konečnou množinu Q prvků nazývanou *stavy*.
  2. podmnožinu I množiny Q obsahující *počáteční stavy*.
  3. podmnožinu T množinu Q obsahující *konečné stavy*.
  4. konečnou množinu *přechodů*, které pro každý stav a každý symbol vstupní abecedy vrací následující stav.

Přechodová funkce

$$\mathbf{F} : Q \times Input \rightarrow \mathcal{P}Q$$

- $\mathbf{F}(q, input)$ obsahuje možné stavy automatu, do kterých lze přejít ze stavu q po přijmutí symbolu $input$.
- $\mathcal{P}Q$ označuje množinu všech podmnožin Q (*potenční množina množiny* Q).

Konečný automat s výstupem [HI98]

- $Input$ konečná abeceda.
- *Konečný automat* nad množinou $Input$ obsahuje následující komponenty:
  1. Konečná množina Q prvků nazývaných *stavy*.
  2. Podmnožina I množiny Q obsahující *počáteční stavy*.
  3. Podmnožina T množiny Q obsahující *koncové stavy*.
  4. Množina $Output$ možných výstupů.
  5. Konečná množina *přechodů*, které pro každý stav a každý symbol vstupní abecedy vrací množinu možných následujících stavů.

Výstupní funkce

$$\mathbf{G} : Q \times Input \rightarrow Output$$

- pro každý stav a pro každý vstupní symbol určuje výstupní symbol.
- $\mathbf{F}$ a $\mathbf{G}$ mohou být parciální funkce.

Příklady konečných automatů [HI98]

Množina $Input$

- akce či příkazy uživatele zadaných na klávesnici,
- kliky či pohyby myše,
- přijmutí signálu ze senzoru.

Množina stavů Q

- hodnoty jistých důležitých proměnných systému,
- mód chování systému,
- druh formuláře, který je viditelný na monitoru,
- zda jsou zařízení aktivní či ne.

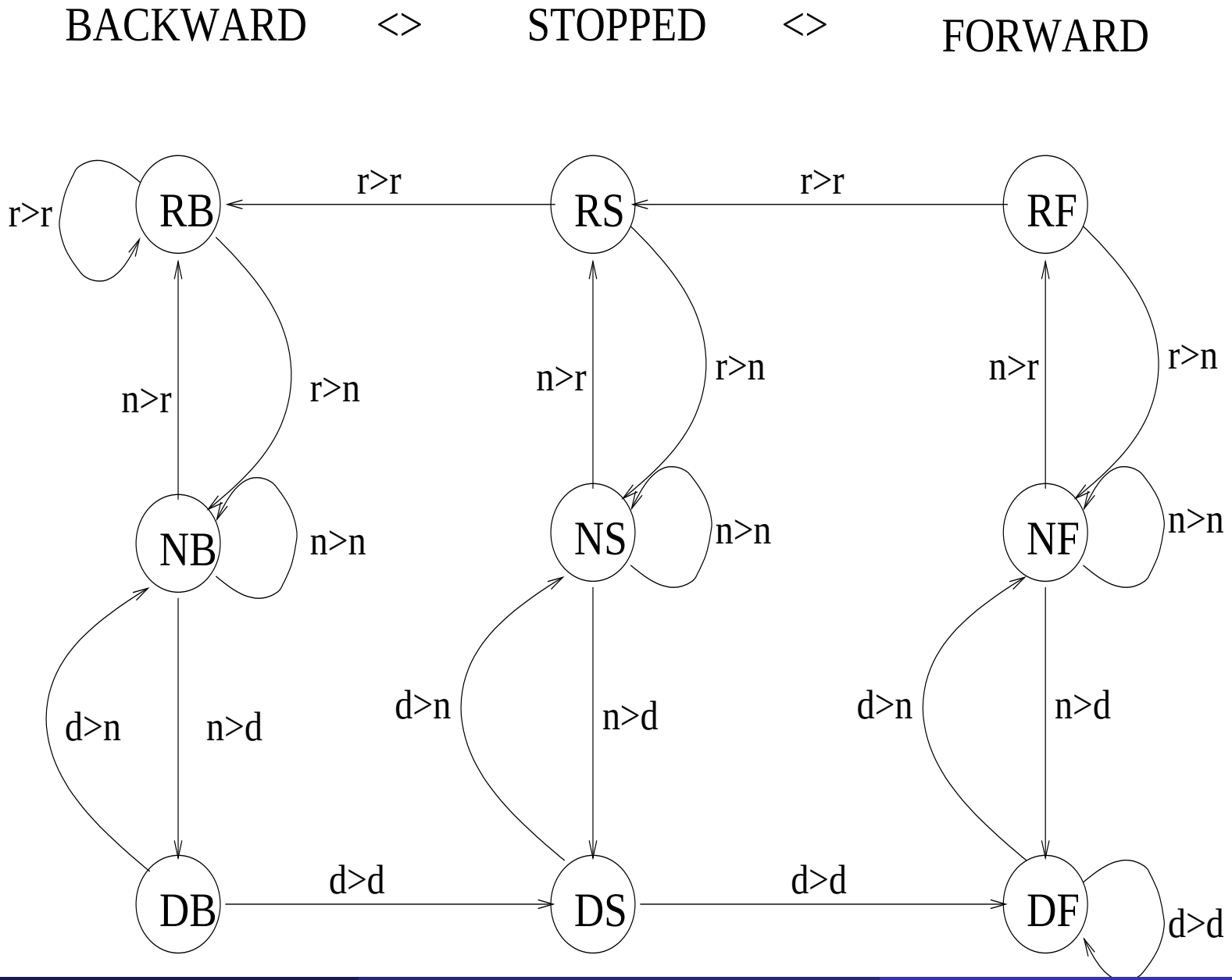
Stavový diagram [Bei95]

- **Vrcholy:** zobrazují stavy (stav softwarové aplikace).
- **Hrany:** znázorňují přechody (výběr položky v menu).
- **Atributy hran (vstupní kódy):** např. akce myší, Alt+Key, funkční klíče, klávesy pohybu kursoru.
- **Atributy hran (výstupní kódy):** např. zobrazení jiného menu či otevření dalšího okna.

Model vesmírné lodi *Enterprise*

- tři nastavení impulsního motoru:
  tah vpřed(d), neutrál(n), a zpětný tah(r).
- tři možné stavy pohybu:
  pohyb dopředu(F), zastavena(S), a pohyb vzad(B).
- kombinace vytvoří devět stavů:
  DF, DS, DB, NF, NS, NB, RF, RS, a RB.
- možné vstupy: $d > d$, $r > r$, $n > n$, $d > n$, $n > d$, $n > r$, $r > n$.

Stavový prostor Enterprise [Bei95]



Vlastnosti stavových diagramů [Bei95]

Vlastnosti

- silně souvislý graf,
- stavové grafy rostou velmi rychle,
- typicky se uvažují všechny možné i nemožné vstupy v daném stavu - implementace systému nemusí být správná.
- pěkná symetrie je velmi řídký jev v praxi.

Přechodové tabulky [Bei95]

- má pro každý stav jeden řádek a pro každý vstup jeden sloupec,
- ve skutečnosti jsou tabulky dvě s stejným tvarem:
  - tabulka přechodů
  - tabulka výstupů
- hodnotou pole v tabulce přechodů je příští stav,
- hodnotou pole v tabulce výstupů je výstupní kód pro daný přechod.
- **hierarchické (vnořené) automaty** jsou jedinou cestou, jak se vyhnout obrovským tabulkám (např. stavová schémata, angl. statechart, starchart, atd.)

Přechodová tabulka Enterprise [Bei95]

Enterprise

STATE	$r > r$	$r > n$	$n > n$	$n > r$	$n > d$	$d > d$	$d > n$	$r > d$	$d > r$
RB	RB	NB							
RS	RB	NS							
RF	RS	NF							
NB			NB	RB	DB				
NS			NS	RS	DS				
NF			NF	RF	DF				
DB						DS	NB		
DS						DF	NS		
DF						DF	NF		

Dosažitelnost stavů [Bei95]

- **Dosažitelný stav:** stav B je dosažitelný ze stavu A, jestliže existuje sekvence vstupů taková, která převede systém ze stav A do stavu B.
- **Nedosažitelný stav:** stav je nedosažitelný, pokud není dosažitelný, zvláště z počátečního stavu. Nedosažitelné stavy znamenají typicky chybu.
- **Silně souvislý:** všechny stavy konečného automatu jsou dosažitelné z počátečního stavu. Většina modelů v praxi je silně souvislá, pokud neobsahují chyby.
- **Isolované stavy:** množina stavů, které nejsou dosažitelné z počátečního stavu. Pokud existují, jedná se o velmi podezřelé, chybové stavy.
- **Reset:** speciální vstupní akce způsobující přechod z jakéhokoliv stavu do počátečního stavu.

Rozdělení stavů [Bei95]

- **Množina počátečního stavu:** Jakmile se provede přechod z této množiny, pak se do této množiny již nelze vrátit (např. boot systému).
- **Pracovní stavy:** po opuštění množiny počátečního stavu, se systém pohybuje v silně souvislé množině stavů, kde se provádí většina testování.
- **Počáteční stav pracovní množiny:** stav pracovní množiny, který je možné považovat za "výchozí stav".
- **Množina koncových stavů:** dostane-li se systém do této množiny, nelze se zpět vrátit do pracovní množiny, např. ukončovací sekvence programu.
- **Úplně specifikovaný:** je systém, pokud je přechody a výstupní kódy definovány pro jakoukoliv kombinaci vstupního kódu a stavu.
- **Okružní cesta stavu A:** sekvence přechodů jdoucí ze stavu A do stavu B a zpět do A.

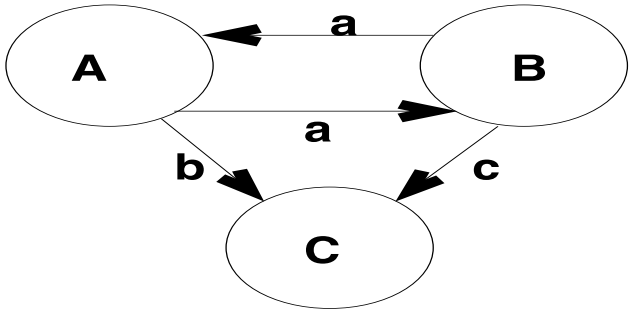
Návrh testů [Bei95]

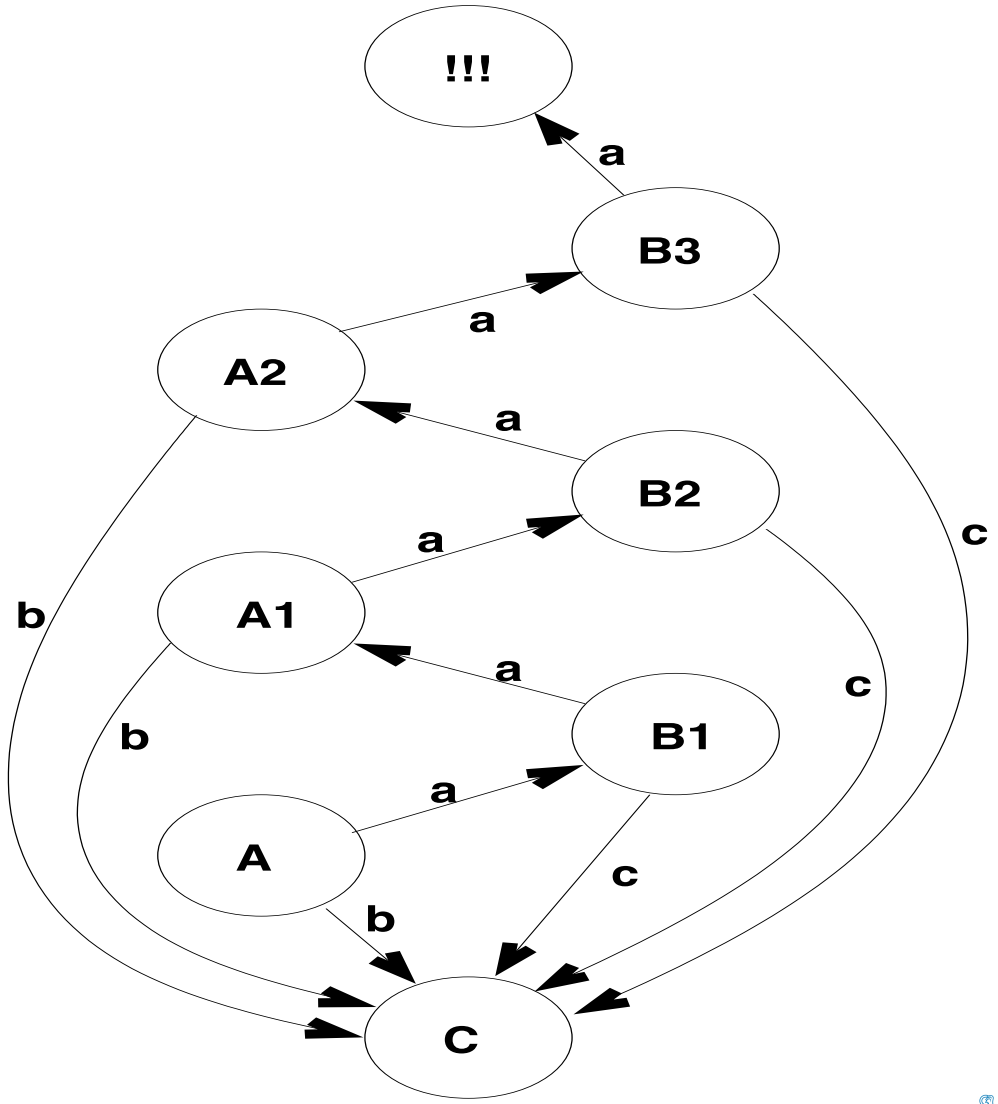
- Každý test začíná v počátečním stavu.
- Z počátečního stavu se systém přivede nejkratší cestou k vybranému stavu, provede se zadaný přechod a systém se nejkratší možnou cestou přivede opět do počátečního staru; vytváříme tzv. okružní cestu.
- Každý test staví na předchozích jednodušších testech.
- Určíme vstupní kód pro každý přechod okružní cesty.
- Určíme výstupní kódy asociované s přechody okružní cesty.
- **Ověříme**
  - kódování vstupů,
  - kódování výstupů,
  - stavy,
  - každý přechod.
- **Je každý koncový stav dosažitelný?**

Skryté stavy

- **Je systém v počátečním stavu?**
  - Test nelze zahájit, pokud systém není potvrzeným způsobem v počátečním stavu.
  - Aplikace si uchovávají persistentně své nastavení.
  - Jestliže předchozí test selže, v jakém stavu se aplikace nachází?
- **Má implementace skryté stavy?**
  - Při testování softwaru můžeme předpokládat věci, které nemusí obecně platit.
    - např. že víme, ve kterém stavu se systém nachází.
  - Typicky se nejedná o jeden či dva skryté stavy, ale stavový prostor se zdvojnásobuje či jinak násobí.

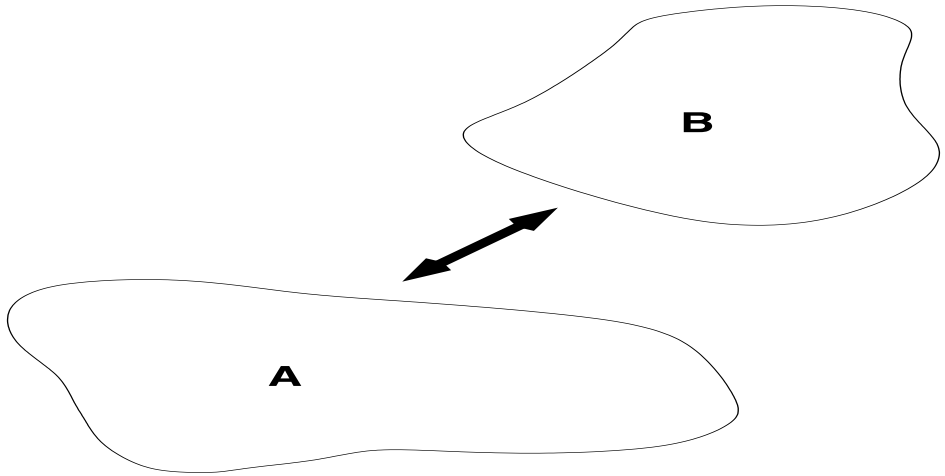
Skryté stavy





Testování konečného automatu [HI98]

- založeno na izomorfismu konečných automatů,
- $\mathcal{A} = (Input, Q, \mathbf{F}, q_0)$
- $\mathcal{A}' = (Input, Q', \mathbf{F}', {q_0}')$
- $g : \mathcal{A} \to \mathcal{A}'$
- $g : Q \to Q'$
  1. $g(q_0) = {q_0}'$
  2. $\forall q \in Q, input \in Input,$
     $g(\mathbf{F}(q, input)) = \mathbf{F}'(g(q), input)$



Konstrukce množiny testů [HI98, Cho78]

Chowova W metoda

- Nechť L je množina vstupních sekvencí a q, q' dva stavy. L *rozliší* stav q od q', jestliže existuje sekvence k v L taková, že výstup získaný apliací k na automat ve stavu q je různý od výstupu získaný aplikací k na stav q'.
- Automat je *minimální*, pokud neobsahuje redundantní stavy.
- Množina vstupních sekvencí W se nazývá *charakterizační množina*, jestliže může rozlišit jakékoliv dva stavy stavy automatu.
- **Pokrytí stavu** je množina vstupních sekvencí L taková, že lze nalézt prvek množiny L, kterým se lze dostat do jakéhokoliv žádaného stavu z počátečního stavu q_0.
- **Pokrytí přechodů** minimálního automatu je množina vstupních sekvencí T, která je pokrytím stavů a uzavřená z hlediska pravé kompozice s množinou vstupů $Input$.
  - $sequence \in T = L \bullet (Input^1 \cup \{<>\})$

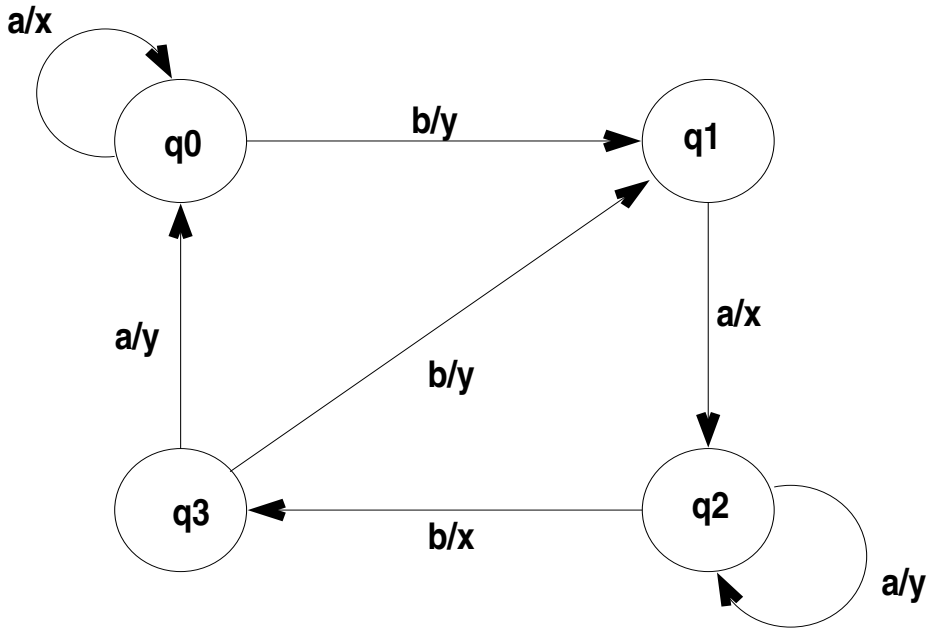
Generování množiny testů [HI98, Cho78]

- O kolik je v implementaci více testů než ve specifikaci? (k)
- $Z = Input^k \bullet W \cup Input^{k-1} \bullet W \cup \cdots \cup Input^1 \bullet W \cup W$
  - Jestliže A a B jsou množiny sekvencí stejné abecedy, pak $A \bullet B$ značí množinu sekvencí, složených ze sekvencí množiny A následující sekvencí z B.
  - k kroků do "neznámého" prostoru následovaných ověřením stavu
- Konečná **množina testů**:

$$T \bullet Z$$

- Pokrytí přechodů zajisťuje,
  - že všechny stavy a přechody specifikace jsou implementovány,
  - množina Z zajišťuje, že implementace je ve stejném stavu, který určuje specifikace.
  - Parametr k jistí, že do jisté úrovně všechny skryté stavy implementace jsou testovány.

Jednoduchý příklad [HI98]



- $Input = \{a, b\}$
- $L = \{<>, b, b::a, b::a::b\}$, $<>$ . . . nulový vstup
- $T = \{<>, a, b, b::a, b::b, b::a::a, b::a::b, b::a::b::a, b::a::b::b\}$
- $W = \{a, b\}$ [Chy84], pp. 31–34

$$
\begin{aligned}
Z &= Input \bullet W \cup W \\
&= \{a, b\} \bullet \{a, b\} \cup \{a, b\} \\
&= \{a, b, a::a, a::b, b::a, b::b\}
\end{aligned}
$$

Testovací množina příkladu [HI98]

$$
\begin{aligned}
T \bullet Z = \\
= \quad & \{<>, a, b, b::a, b::b, b::a::a, b::a::b, b::a::b::a, b::a::b::b\} \\
& \bullet \{a, b, a::a, a::b, b::a, b::b\} \\
= \quad & \{a, b, a::a, a::b, b::a, b::b, \\
& a::a, a::b, a::a::a, a::a::b, a::b::a, a::b::b, \\
& b::a, b::b, b::a::a, b::a::b, b::b::a, b::b::b, \\
& b::a::a, b::a::b, b::a::a::a, b::a::a::b, b::a::b::a, b::a::b::b, \\
& b::b::a, b::b::b, b::b::a::a, b::b::a::b, b::b::b::a, b::b::b::b, \\
& b::a::a::a, b::a::a::b, b::a::a::a::a, b::a::a::a::b, b::a::a::b::a, b::a::a::b::b, \\
& b::a::b::a, b::a::b::b, b::a::b::a::a, b::a::b::a::b, b::a::b::b::a, b::a::b::b::b, \\
& b::a::b::a::a, b::a::b::a::b, b::a::b::a::a::a, \\
& b::a::b::a::a::b, b::a::b::a::b::a, b::a::b::a::b::b, \\
& b::a::b::b::a, b::a::b::b::b, b::a::b::b::a::a, \\
& b::a::b::b::a::b, b::a::b::b::b::a, b::a::b::b::b::b\} \\
= \quad & \ldots \text{simplification}
\end{aligned}
$$

Aplikace [Bei95]

- software řízený pomocí menu,
- objektově orientovaný software,
- protokoly,
- řadiče zařízení,
- starší hardware,
- mikropočítače průmyslových a domacích zařízení,
- instalace softwaru,
- software pro archivaci či obnovení.

Mealyho automat [Mea55, Mat13]

Definition 1 (Mealyho automat s konečným počtem stavů je)

- 6-tice $M(X, Y, Q, q_0, \delta, \lambda)$:
- X je konečná množina vstupních symbolů (vstupní abeceda),
- Y je konečná množina výstupních symbolů (výstupní abeceda),
- Q je konečná množina stavů,
- $q_0 \in Q$ je počáteční stav,
- $D \subseteq Q \times X$ je specifikační doména,
- $\delta : D \to Q$ je přechodová funkce,
- $\lambda : D \to Y$ je výstupní funkce.

- Jestliže $D = Q \times X$, potom M je **úplný** Mealyho automat [SP10].
- Řetězec $\alpha = x_1 \dots x_k, \alpha \in I^*$ je **definovaná vstupní sekvence** pro stav $q \in Q$, jestliže existují $q_1, \dots, q_{k+1}$, kde $q_1 = q$ takové, že $(q_i, x_i) \in D$ a $\delta(q_i, x_i) = q_{i+1}$ pro všechna $1 \leq i \leq k$.

Minimalita automatu [SP10, Mat13]

Dán Mealyho automat $M(X, Y, Q, q_0, \delta, \lambda)$ s konečným počtem stavu.

- Rozšíření přechodové a výstupní funkce aplikovanou na vstupní symbol x na definované vstupní sekvence α, zahrnující prázdnou sekvenci ϵ:
  - pro $q \in Q$, $\delta(q, \epsilon) = q$ a $\lambda(q, \epsilon) = \epsilon$
  - $\delta(q, \alpha x) = \delta(\delta(q, \alpha), x)$
  - $\lambda(q, \alpha x) = \lambda(\delta(q, \alpha), x)$
- $\Omega(q)$ je množina všech definovaných vstupních sekvencí pro stav $q \in Q$.
- Dva stavy $q, q' \in Q$ jsou **rozlišitelné**, jestliže existuje $\gamma \in \Omega(q) \cap \Omega(q')$ takové, že $\lambda(q, \gamma) \neq \lambda(q', \gamma)$. Pak říkáme, že γ **rozlišuje** stavy q a q'.
- Dva stavy $q_1, q_2 \in Q; q_1 \neq q_2$ jsou **stavově ekvivalentní**, jestliže po aplikaci jakékoliv vstupní sekvence vedou do stejných nebo ekvivalentních stavů.
- M je **minimální**, jestliže žádné jeho dva stavy nejsou ekvivalentní [Ner58, Gil60].

C-ekvivalence stavů [SP10, Mat13]

Dán Mealyho automat $M(X, Y, Q, q_0, \delta, \lambda)$ s konečným počtem stavu.

- Nechť je dána množina $C \subseteq \Omega(q) \cap \Omega(q')$.
- Stavy $q_1, q_2 \in Q$ jsou C-**ekvivalentní**, jestliže $\lambda(q, \gamma) \neq \lambda(q', \gamma)$ pro všechny $\gamma \in C$.

Dva automaty $M_1(X, Y, Q_1, q_0^1, \delta_1, \lambda_1)$ a $M_2(X, Y, Q_2, q_0^2, \delta_2, \lambda_2)$ jsou **ekvivalentní**, jestliže

1. pro každý stav $q \in M_1$ existuje $q' \in M_2$ takový, že q a q' jsou ekvivalentní a
2. pro každý stav $q \in M_2$ existuje $q' \in M_1$ takový, že q a q' jsou ekvivalentní.

k-**ekvivalence**

- Nechť $M_1(X, Y, Q_1, q_0^1, \delta_1, \lambda_1)$ a $M_2(X, Y, Q_2, q_0^2, \delta_2, \lambda_2)$ jsou dva automaty.
- Stavy $q_i \in Q_1$ a $q_j \in Q_2$ se považují za k-**ekvivalentní**, jestliže po aplikování jakékoliv vstupní sekvence délky k jsou produkovány identické výstupní sekvence.

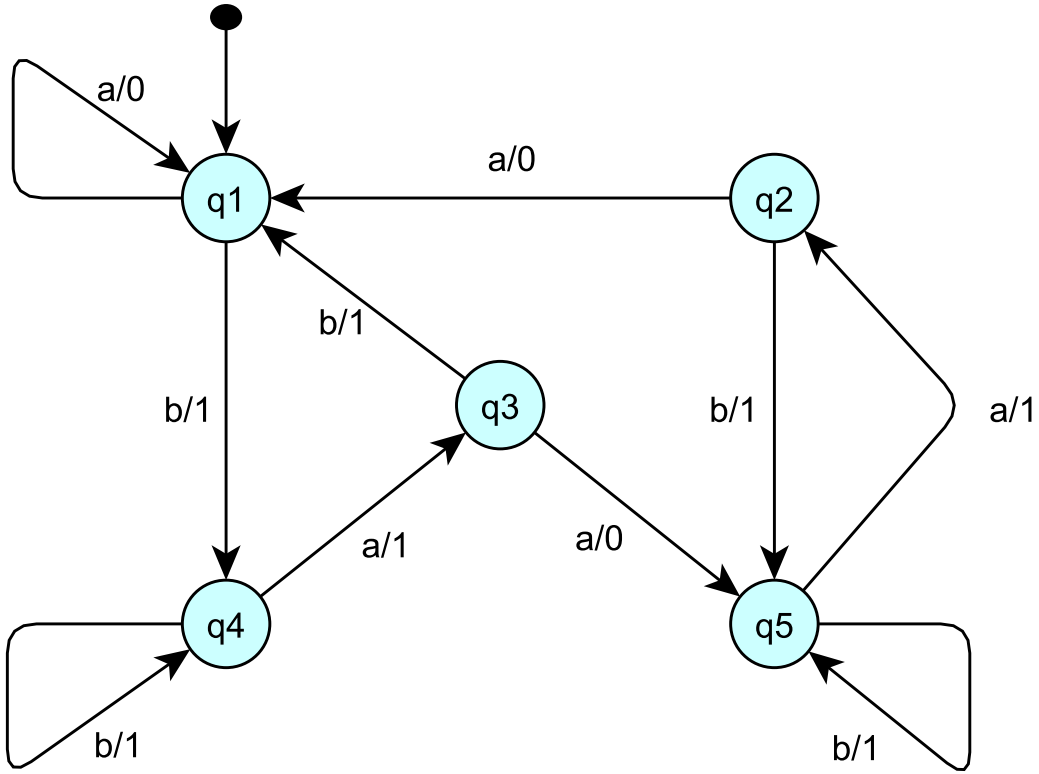
Charakterizační množina W [SP10, Mat13]

Nechť Mealyho automat $M(X, Y, Q, q_0, \delta, O)$ s konečným počtem stavu je minimální a úplný.

- W je konečná množina vstupních sekvencí, která rozliší jakýkoliv pár stavů $q_i, q_j \in Q$.
- Každá vstupní sekvence $\gamma \in W$ má konečnou délku.
- Pro každé dva stavy $q_i, q_j \in Q$ množina W obsahuje (alespoň jednu) vstupní sekvenci γ takovou, že

$$\lambda(q_i, \gamma) \neq \lambda(q_j, \gamma)$$

Příklad charakterizační množiny [HI98]



- $Input = \{a, b\}$
- $W = \{baaa, aa, aaa\}$
- $\lambda(q_1, baaa) = 1 \ldots (1101)$
- $\lambda(q_2, baaa) = 0 \ldots (1100)$
- $\lambda(q_1, baaa) \neq \lambda(q_2, baaa) \implies baaa$ rozlišuje stavy q_1 a q_2

k-ekvivalentní rozklad stavů Q [Mat13]

- k-ekvivalentní rozklad stavů Q označovaný jako P_k je soubor n konečných množin $\Sigma_{k,1}, \Sigma_{k,2}, \ldots, \Sigma_{k,n}$ takových

$$\cup_{i=1}^{n} \Sigma_{k,i} = Q$$

- Stavy v $\Sigma_{k,i}$ jsou k-ekvivalentní.
- Jestliže $q_{\ell_1} \in \Sigma_{k,j}$ a $q_{\ell_2} \in \Sigma_{k,j}$ pro $i \neq j$, potom q_{ℓ_1} a q_{ℓ_2} jsou k-rozlišitelné.

Konstrukce W množiny [Mat13]

Postup

1. Vytvoření sekvence k-ekvivalentních rozklad stavů Q označenou jako $P_1, P_2, \ldots, P_m, m > 0$
2. Prohledání k-ekvivalentních rozkladů v opačném pořadí se současnou kontrukcí rozlišujících sekvencí pro každou dvojici stavů.

- Je garantována konvergence postupu.
- Po skončení postupu každá třída $\Sigma_{K,j}$ konečného rozkladu P_K definuje třídu ekvivalentních stavů (typicky 1).

Neformálně:

- nejprve se zjistí, co lze rozlišit v jednom kroku
- po té ve dvou krocích
- atd.

Konstrukce W množiny [Mat13]

Tabulární reprezentace M.
0-ekvivalenční rozklad $P_0 = \{\Sigma_1 = \{q_1, q_2, q_3, q_4, q_5\}\}$

Současný stav	**Výstup**		**Následující stav**	
	a	b	a	b
q_1	0	1	q_1	q_4
q_2	0	1	q_1	q_5
q_3	0	1	q_5	q_1
q_4	1	1	q_3	q_4
q_5	1	1	q_2	q_5

Konstrukce 1-ekvivalenční rozklad P_1 [Mat13]

1-ekvivalenční rozklad $P_1 = \{\Sigma_1 = \{q_1, q_2, q_3\}, \Sigma_2 = \{q_4, q_5\}\}$.

Σ	**Současný stav**	**Výstup**		**Následující stav**	
		a	b	a	b
1	q_1	0	1	q_1	q_4
	q_2	0	1	q_1	q_5
	q_3	0	1	q_5	q_1
2	q_4	1	1	q_3	q_4
	q_5	1	1	q_2	q_5

Konstrukce 2-ekvivalenční rozklad: přepis P_1 [Mat13]

Přepis P_1, stav q_i je nahrazen $q_{i,j}$, přičemž $q_i \in \Sigma_j$.

Σ	**Současný stav**	**Následující stav**	
		a	b
1	q_1	$q_{1,1}$	$q_{4,2}$
	q_2	$q_{1,1}$	$q_{5,2}$
	q_3	$q_{5,2}$	$q_{1,1}$
2	q_4	$q_{3,1}$	$q_{4,2}$
	q_5	$q_{2,1}$	$q_{5,2}$

Konstrukce 2-ekvivalenční rozklad: konstrukce P_2 [Mat13]

Konstrukce P_2. Rozdělení $\Sigma_{1,j}$ podle skupin příštích stavů.

Σ	**Současný stav**	**Následující stav**	
		a	b
1	q_1	$q_{1,1}$	$q_{4,3}$
	q_2	$q_{1,1}$	$q_{5,3}$
2	q_3	$q_{5,3}$	$q_{1,1}$
3	q_4	$q_{3,2}$	$q_{4,3}$
	q_5	$q_{2,1}$	$q_{5,3}$

Konstrukce 3-ekvivalenční rozklad: konstrukce P_3 [Mat13]

Konstrukce P_3. Rozdělení $\Sigma_{2,j}$ podle skupin příštích stavů.

Σ	**Současný stav**	**Následující stav**	
		a	b
1	q_1	$q_{1,1}$	$q_{4,3}$
	q_2	$q_{1,1}$	$q_{5,4}$
2	q_3	$q_{5,4}$	$q_{1,1}$
3	q_4	$q_{3,2}$	$q_{4,3}$
4	q_5	$q_{2,1}$	$q_{5,4}$

Konstrukce 4-ekvivalenční rozklad: konstrukce P_4 [Mat13]

Konstrukce P_4. Rozdělení $\Sigma_{3,j}$ podle skupin příštích stavů.

Σ	**Současný stav**	**Následující stav**	
		a	b
1	q_1	$q_{1,1}$	$q_{4,4}$
2	q_2	$q_{1,1}$	$q_{5,5}$
3	q_3	$q_{5,5}$	$q_{1,1}$
4	q_4	$q_{3,3}$	$q_{4,4}$
5	q_5	$q_{2,2}$	$q_{5,5}$

Nalezení rozlišujících sekvencí: příklad [Mat13]

1. Nalezněme rozlišující sekvenci stavů q_1 a q_2.
2. Inicializace rozlišující sekvence: $z = \epsilon$.
3. Najdi tabulky P_i a P_{i+1} takové, že (q_1, q_2) jsou ve stejné skupině v P_i a v různých skupinách v P_{i+1}:
   - dostaneme P_3 a P_4.
4. Nalezni vstupní symbol rozlišující q_1 a q_2 v tabulce P_3
   - Rozlišujícím symbolem je b.
   - Prodluž rozlišující sekvenci: $z := z.b = \epsilon.b = b$.
5. Nalezni příští stavy stavů q_1 a q_2 po aplikaci symbolu b
   - dostaneme q_4 a q_5.
6. Najdi tabulky P_i a P_{i+1} takové, že (q_4, q_5) jsou ve stejné skupině v P_i a v různých skupinách v P_{i+1}:
   - dostaneme P_2 a P_3.
7. $(q_4, q_5) \rightarrow P_2, P_3 \rightarrow a \rightarrow z = ba$
8. $(q_3, q_2) \rightarrow P_1, P_2 \rightarrow a \rightarrow z = baa$
9. $(q_1, q_5) \rightarrow P_0, P_1 \rightarrow a \rightarrow z = baaa$
10. Opakuj pro každý pár (q_i, q_j): $W = \{a, aa, aaa, baaa\}$

Set Theory and Strings

Definition 3.1

A **cardinality** of a set A is the number of elements of the set A. It is denoted $|A|$.

Definition 3.2

A **partition** (CZ rozklad) P of a set A is a set of nonempty subsets of A such that every element $a \in A$ is in exactly one of these subsets, i.e., A is a disjoint union of the subsets.

String operations:

- ϵ is the empty symbol, every extended alphabet X_ϵ contains ϵ,
- $|\epsilon| = 0$.
- $x \cdot y$ means concatenation of strings (words) x and y.
  It can be also written as xy,
- $|x|$ means the length of string (word) x.

Finite State Machine

A **finite-state machine** is a sextuple $(S, \Sigma, \Gamma, s_0, \delta, \lambda)$, where

- S is a finite nonempty set of states,
- Σ is an input alphabet (a finite nonempty set of symbols),
- Γ is an output alphabet (a finite nonempty set of symbols),
- s_0 is an initial state, $s_0 \in S$,
- δ is a state-transition function: $\delta : S \times \Sigma \to S$,
- ω is an output function: $\lambda : S \times \Sigma_\epsilon \to \Gamma_\epsilon$.

Additional designations:

- Σ^* is the set of all strings (words) over the input alphabet,
- Γ^* is the set of all strings (words) over the output alphabet,
- Alphabet X^* always contains ϵ and $\forall x \in X^* : \epsilon \cdot x = x = x \cdot \epsilon$.
- Thus X^* is always nonempty and it is also countable because X is countable.

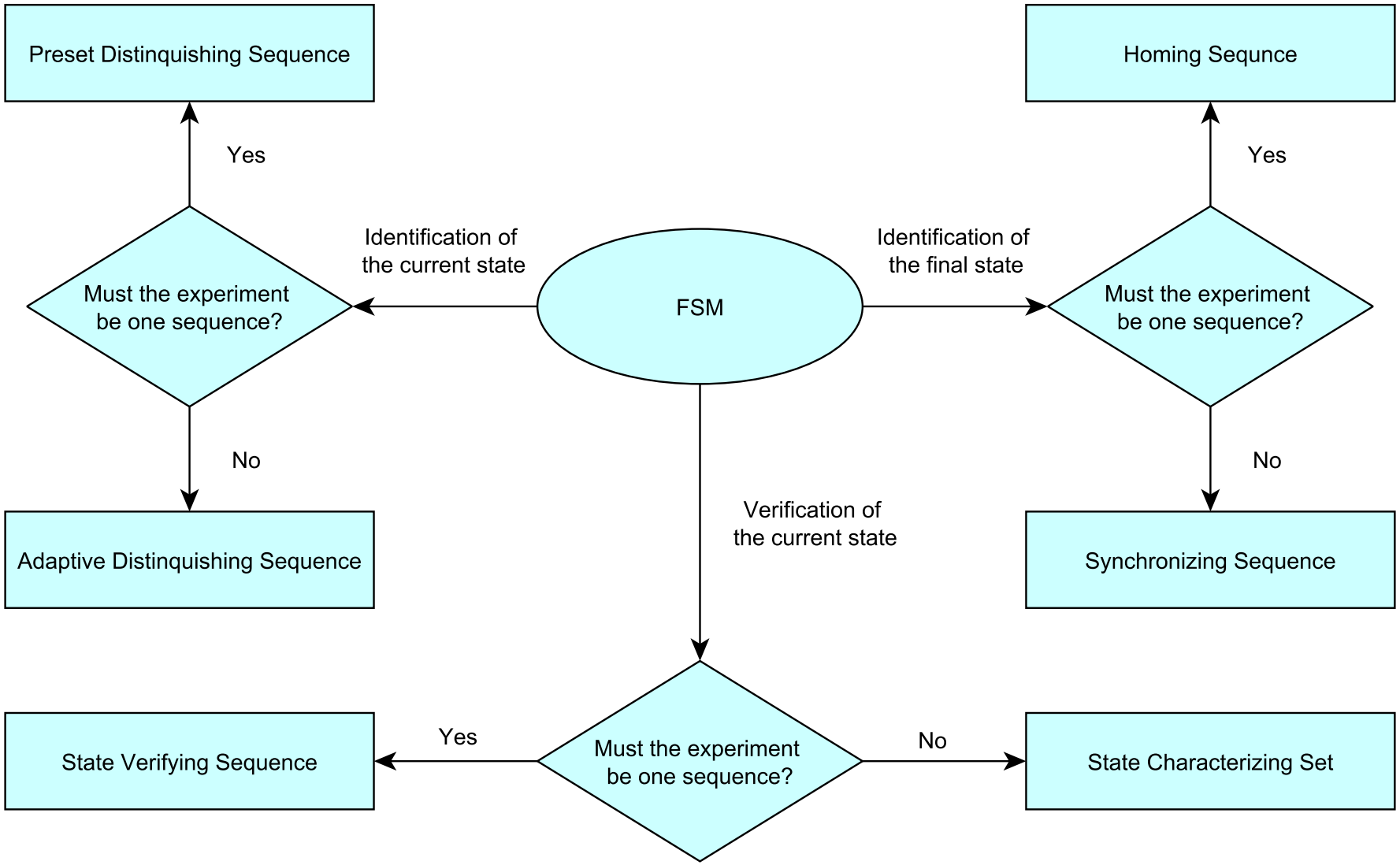
Input/Output Sequence

- An **input sequence** is a string of input symbols.
- An **output sequence** is a string of output symbols.
- An **sequence experiment** is an application of an input sequence to the given FSM from a given state and the output sequence is recorded.
  - The main purpose of the experiment is a possibility to claim something about a given initial state or a final state.
  - An experiment can be represented as a preset of adaptive sequence.
  - The **preset form** is one input sequence.
    - All symbols of the input sequence are applied and an output sequence is obtained.
    - A single decision is based on the entire output sequence.
  - The **adaptive form** is represented as a decision tree, where each internal node is an input symbol and edges to children are labeled by possible output symbols.
    - The next input symbol depends on the observed previous output symbol.
  - A FSM having a sequence in a preset form has always also an adaptive form of this sequence.

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 3.3

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 3.4

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.

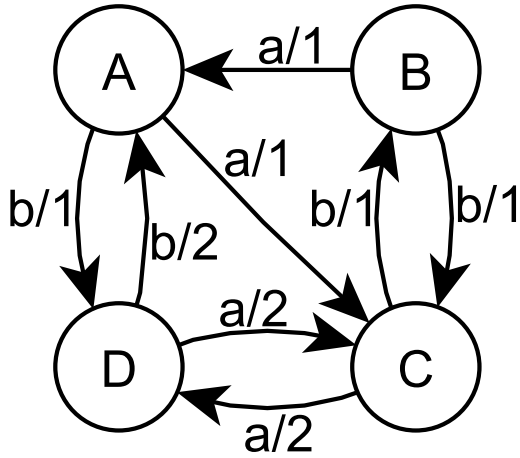
PDS algorithm I [DH94, Sou14]

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.

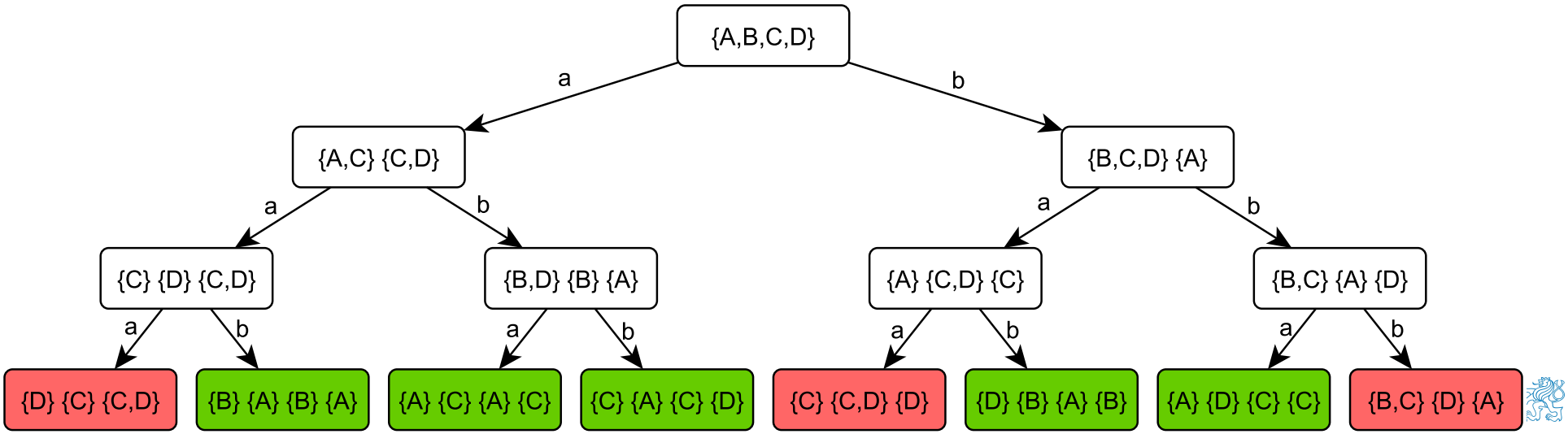
PDS algorithm II [DH94, Sou14]

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]

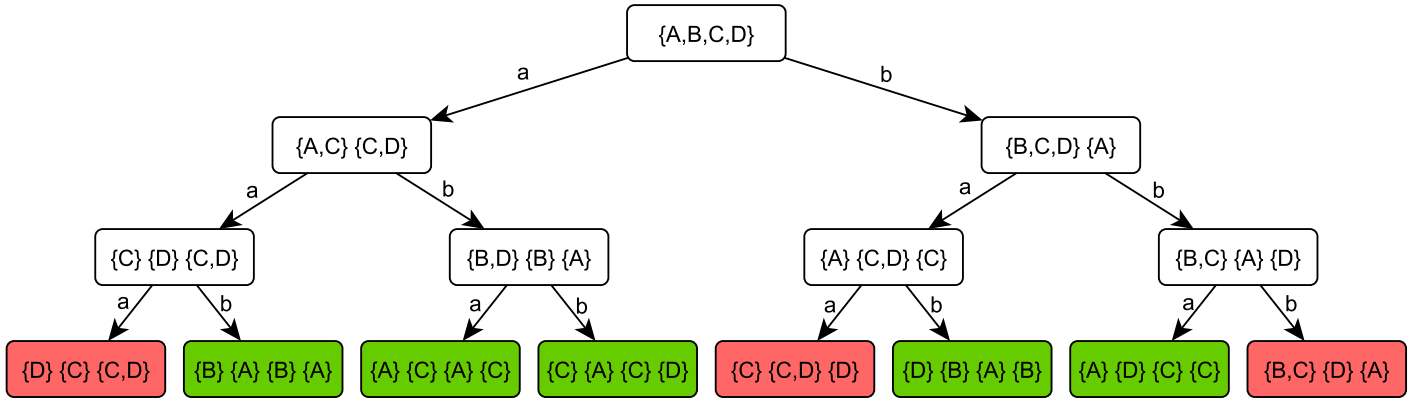


	a	b
A	C / 1	D / 1
B	A / 1	C / 1
C	D / 2	B / 1
D	C / 2	A / 2



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

Homing Sequence [DH94, Sou14]

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

Definition 3.5

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.

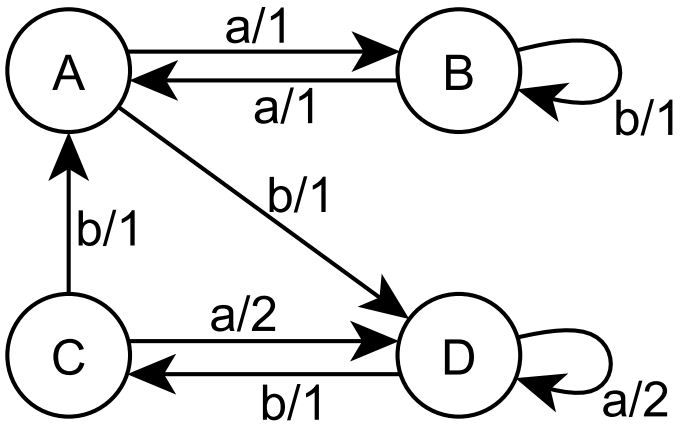
- A homing sequence exists for all reduced FSM.
- If the current state of FSM is unknown, HS is applied and according to the output sequence a final state is determined.
- An adaptive form of sequence can rapidly reduce the length of homing sequence in some cases.

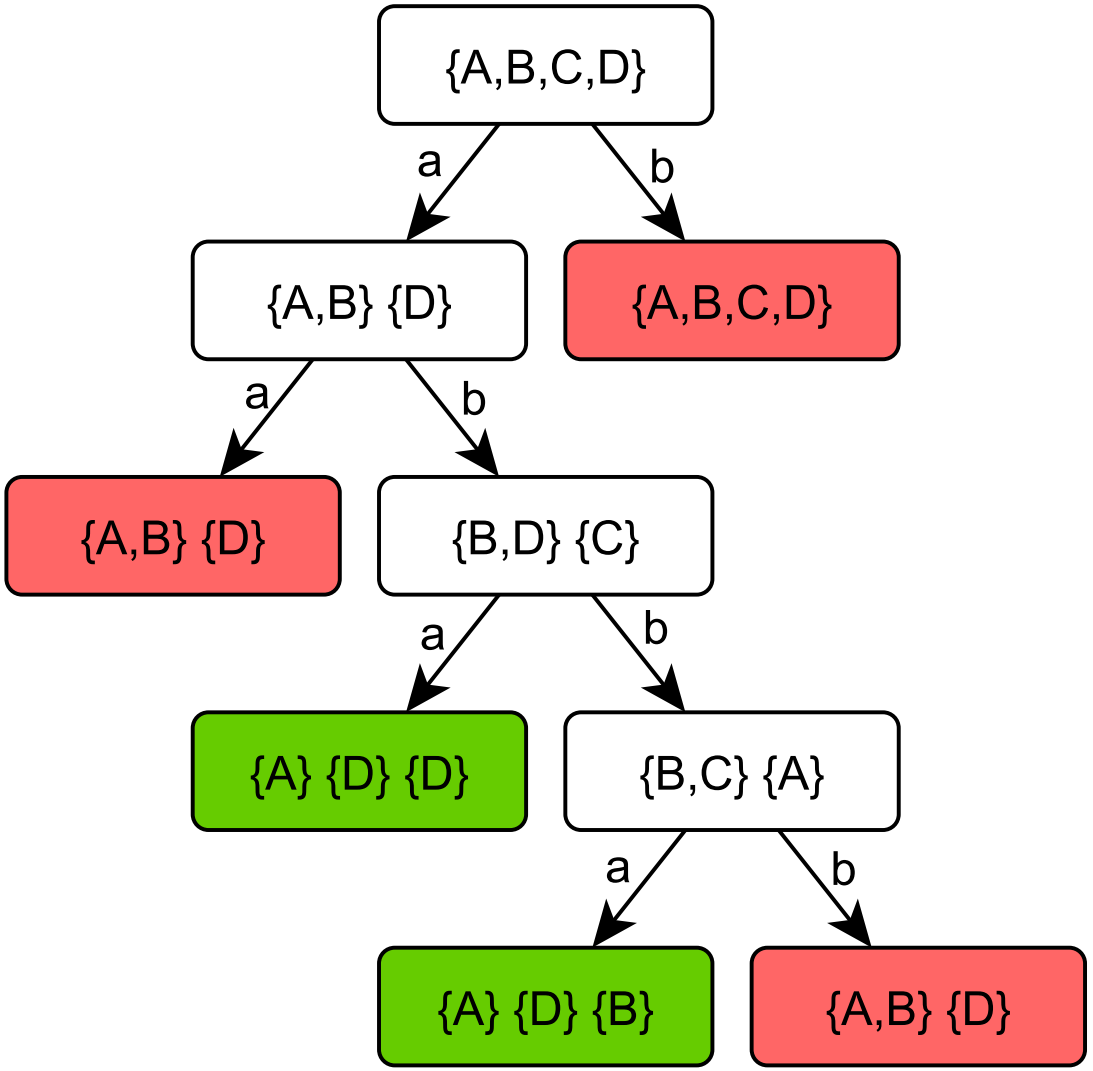
HS algorithm [DH94, Sou14]

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]

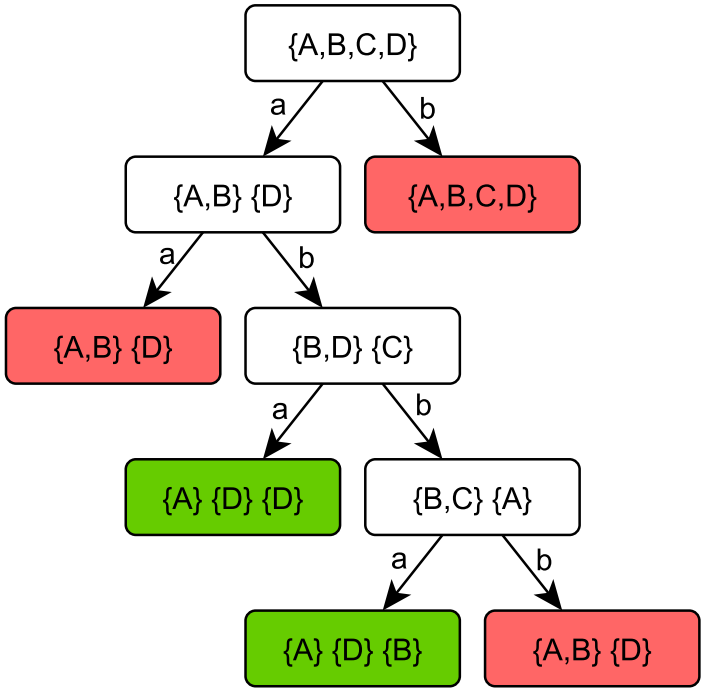
	a	b
A	B / 1	D / 1
B	A / 1	B / 1
C	D / 2	A / 1
D	D / 2	C / 1





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 3.6

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]

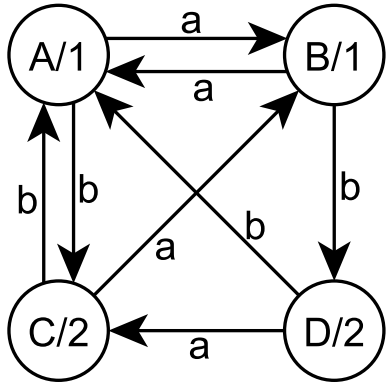
- The synchronizing sequences can be found from a synchronizing tree which is a successor tree.
- The synchronizing tree is similar to the homing tree except that the states represented by a node are not grouped according to outputs since the final state must be determined independently of the output.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

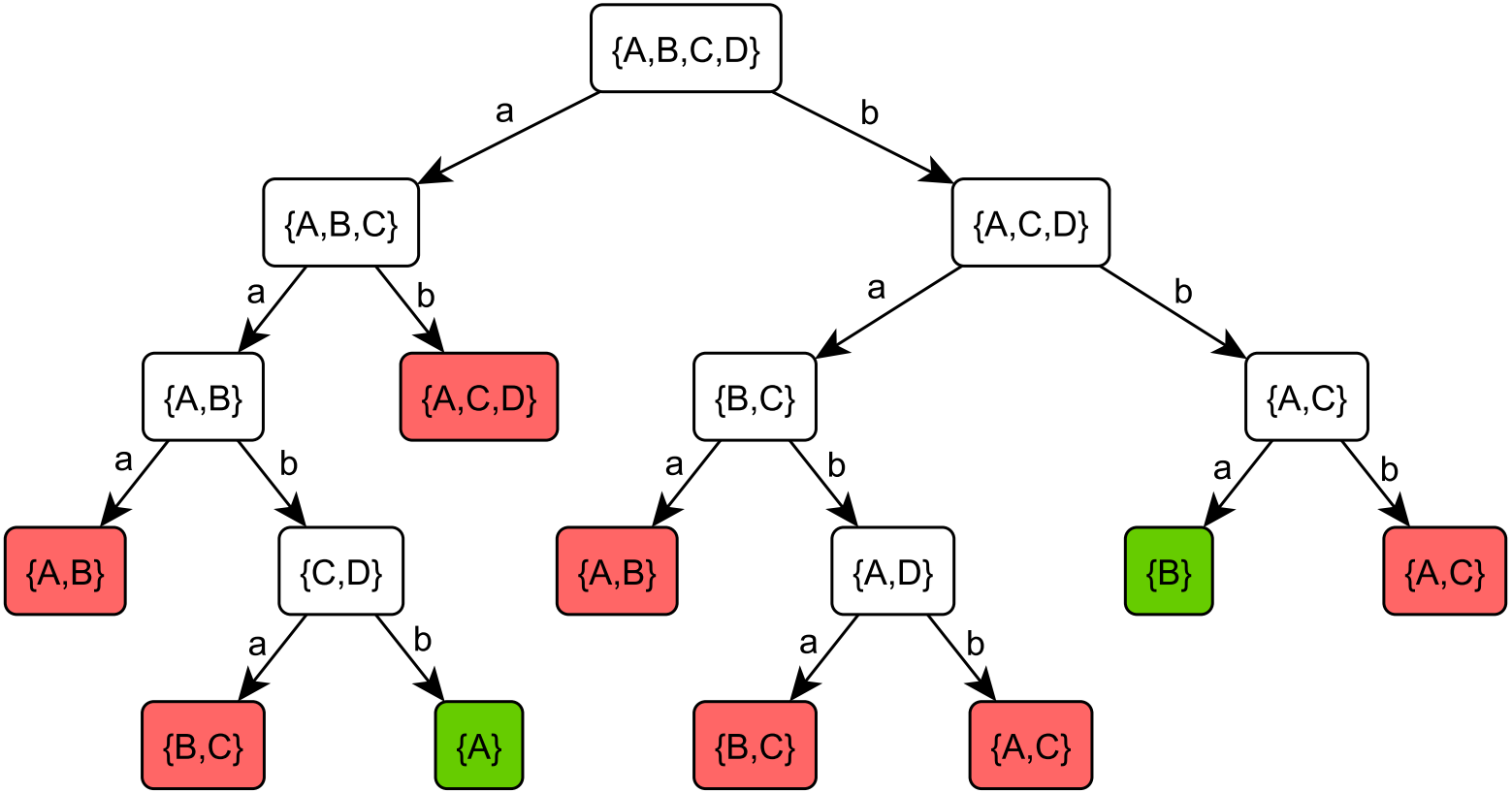
SS algorithm II [DH94, Sou14]

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]

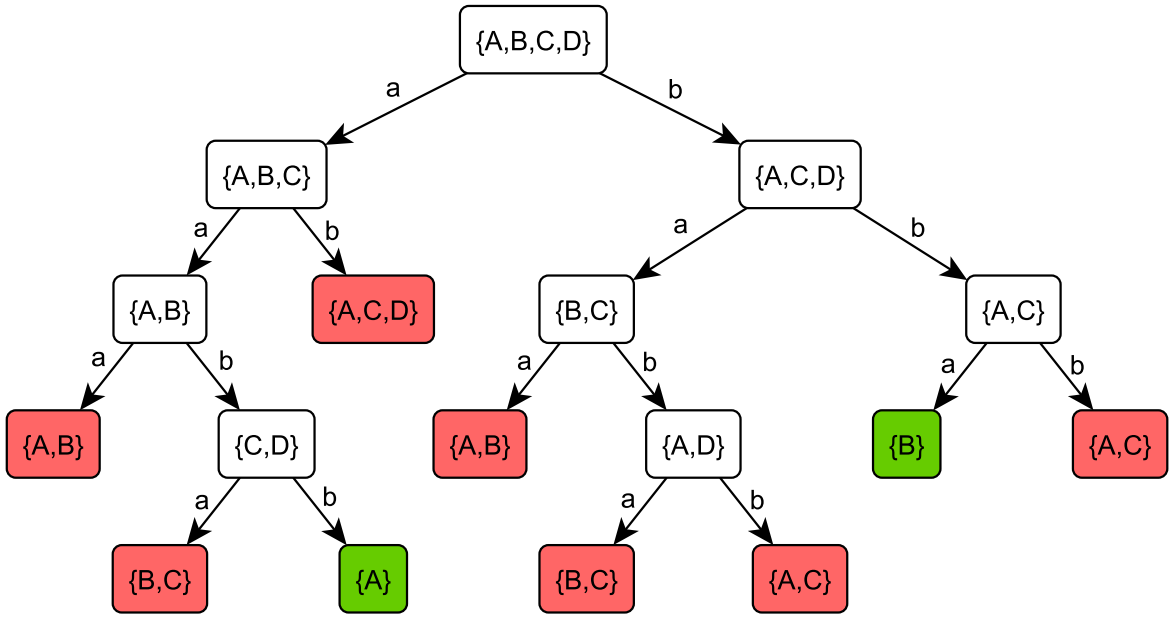
	a	b	ϵ
A	B	C	1
B	A	D	1
C	B	A	2
D	C	A	2





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

Literatura I

[Bei95] Boris Beizer. *Black-Box Testing, Techniques for Functional Testing of Software and Systems*. John Wiley & Sons, Inc., New York, 1995.

[Cho78] T.S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, SE-4(3):178–187, May 1978.

[Chy84] Michal Chytil. *Automaty a gramatiky*. SNTL Praha, 1984.

[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.

[Gil60] A. Gill. Characterizing experiments for finite-memory binary automata. *IRE Transactions on Electronic Computers*, EC-9(4):469–471, Dec 1960.

[HI98] Mike Holcombe and Florentin Ipate. *Correct Systems: Building a Business Process Solution*. Springer, 1998.

[Mat13] Aditya P. Mathur. Foundations of software testing 2e, slides, 2013.

[Mea55] George H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal, The*, 34(5):1045–1079, Sept 1955.

[Ner58] A. Nerode. Linear automaton transformations. *Proc. Amer. Math. Soc.*, 9:541–544, 1958.

[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.

[SP10] A. Simao and A. Petrenko. Checking completeness of tests for finite state machines. *IEEE Transactions on Computers*, 59(8):1023–1032, Aug 2010.