FSM Testing and Checking Sequences

Radek Mařík

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

January 23, 2019

Outline

1. Finite State Machine
   - Definitions
2. Finite state machine testing
   - Terminology
   - Formal FSM Testing
   - Example
   - Characterization Set Construction

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 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.
- *Finite state machine* over $Input$ consists of the following items:
  1. A finite set Q of elements called *states*.
  2. A subset I of the set Q containing *initial states*.
  3. A subset T of the set Q containing *end states*.
  4. A finite set of *transitions*, that returns a set of all possible next states for each state and each symbol of the input alphabet.

Transition function

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

- $\mathbf{F}(q, input)$ contains all possible states of the automaton, to which it is possible to make a transition if the input symbol $input$ is accepted in state q.
- $\mathcal{P}Q$ denotes a set of all subsets of the set Q (*a power set of the set* Q, CZ potenční množina množiny).

Finite State Machine [HI98]

- Let $Input$ be a finite alphabet.
- *Finite state machine* over $Input$ consists of the following items:
  1. A finite set Q of elements called *states*.
  2. A subset I of the set Q containing *initial states*.
  3. A subset T of the set Q containing *end states*.
  4. A finite set of *transitions*, that returns a set of all possible next states for each state and each symbol of the input alphabet.

Transition function

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

- $\mathbf{F}(q, input)$ contains all possible states of the automaton, to which it is possible to make a transition if the input symbol $input$ is accepted in state q.
- $\mathcal{P}Q$ denotes a set of all subsets of the set Q (*a power set of the set* Q, CZ potenční množina množiny).

Finite State Machine with Output (Mealy) [HI98]

- Let $Input$ be a finite alphabet.
- *Finite state machine* over $Input$ consists of the following items:
  1. A finite set Q of elements called *states*.
  2. A subset I of the set Q containing *initial states*.
  3. A subset T of the set Q containing *end states*.
  4. A set $Output$ of all possible output symbols.
  5. A finite set of *transitions*, that returns a set of all possible next states for each state and each symbol of the input alphabet.

Output function

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

- $\mathbf{G}(q, input)$ determines an output symbol for each state and for each input symbol.
- $\mathbf{F}$ and $\mathbf{G}$ might be partial functions.

Finite State Machine with Output (Mealy) [HI98]

- Let $Input$ be a finite alphabet.
- *Finite state machine* over $Input$ consists of the following items:
  1. A finite set Q of elements called *states*.
  2. A subset I of the set Q containing *initial states*.
  3. A subset T of the set Q containing *end states*.
  4. A set $Output$ of all possible output symbols.
  5. A finite set of *transitions*, that returns a set of all possible next states for each state and each symbol of the input alphabet.

Output function

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

- $\mathbf{G}(q, input)$ determines an output symbol for each state and for each input symbol.
- $\mathbf{F}$ and $\mathbf{G}$ might be partial functions.

Finite State Machines Examples [HI98]

A set $Input$ of input symbols

- Actions or commands of the user entered through a keyboard,
- Mouse clicks or moves,
- Signals accepted by a sensor.

A set Q of states

- Values of certain important variables of the system,
- A behavioral model of the system,
- A formular type visible on the monitor,
- Whether devices are active or not.

Finite State Machines Examples [HI98]

A set $Input$ of input symbols

- Actions or commands of the user entered through a keyboard,
- Mouse clicks or moves,
- Signals accepted by a sensor.

A set Q of states

- Values of certain important variables of the system,
- A behavioral model of the system,
- A formular type visible on the monitor,
- Whether devices are active or not.

State Diagram [Bei95]

- **Nodes:** represent states (a state of the software application).
- **Edges:** represent transitions (a menu item selection).
- **Edge attributes (input symbols):** e.g. mouse actions, Alt+Key, function keys, keyboard keys of cursor movement.
- **Edge attributes (output symbols):** e.g. a menu presentation or a next window open.

Space ship model *Enterprise*

- three modes of the impulse engine:
  move forward(d), neutral(n), and move backward(r).
- three possible state of movement:
  forward(F), stop(S), and backward(B).
- their combinations creates nine states:
  DF, DS, DB, NF, NS, NB, RF, RS, and RB.
- possible inputs: $d > d$, $r > r$, $n > n$, $d > n$, $n > d$, $n > r$, $r > n$.

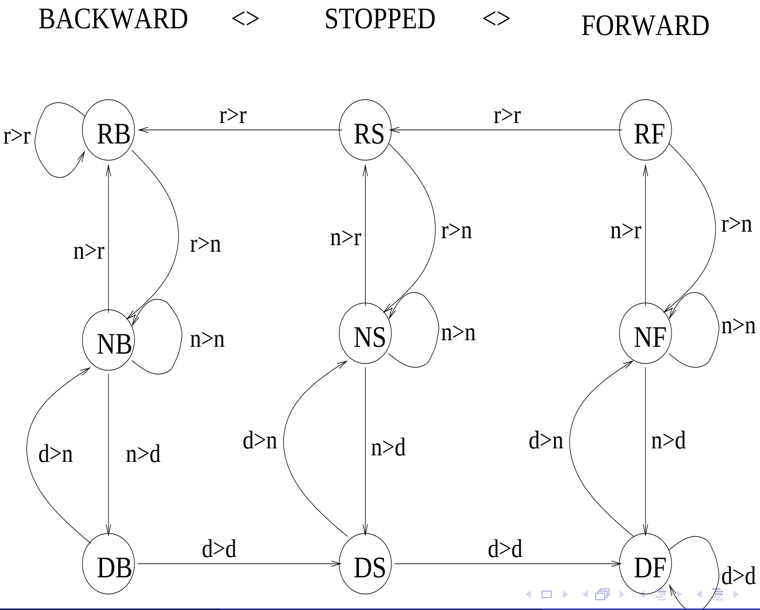
State Diagram [Bei95]

- **Nodes:** represent states (a state of the software application).
- **Edges:** represent transitions (a menu item selection).
- **Edge attributes (input symbols):** e.g. mouse actions, Alt+Key, function keys, keyboard keys of cursor movement.
- **Edge attributes (output symbols):** e.g. a menu presentation or a next window open.

Space ship model *Enterprise*

- three modes of the impulse engine:
  move forward(d), neutral(n), and move backward(r).
- three possible state of movement:
  forward(F), stop(S), and backward(B).
- their combinations creates nine states:
  DF, DS, DB, NF, NS, NB, RF, RS, and RB.
- possible inputs: $d > d$, $r > r$, $n > n$, $d > n$, $n > d$, $n > r$, $r > n$.

Enterprise State Space [Bei95]



State Diagram Properties [Bei95]

Properties

- A strong connected graph,
- State graphs grow very quickly,
- All possible and impossible inputs are considered in every state
  - the implementation of the system might be incorrect.
- Nice symmetry is a very rare case in real life.

Transition Table [Bei95]

A transition table

- has a row for each state
- has a column for each input.
- In fact, there are two tables with the same shape:
  - a transition table,
  - an output table.
- A value in the transition table represents the next state.
- A value in the output table is the output code for a given transition.
- **Hierachical (nested) automata** are the only way how huge tables can be avoided (e.g. statechart, starchart, etc.)

Enterprise Transition Table [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		

Outline

State Reachability [Bei95]

- **Reachable state:** a state B is reachable from a state A, if there is a input sequence such that the system is transferred from the state A to the state B.
- **Unreachable state:** a state is unreachable if it is not reachable, especially from the initial state. Unreachable states implies typically a mistake.
- **Strong connectivity:** all state of the finite automaton are reachable from the initial state. Most practical models are strongly connected if they do not contain mistakes.
- **Isolated states:** a set of states that are not reachable from the initial state. If they exist, then they are very suspicious, mistaken states.
- **Reset:** a special input symbol/action causing the transition from any state to the initial state.

State Categories [Bei95]

- **The set of the initial state:** If a transition leading from this set is performed, then there is no way back to this set (e.g. a boot of the system).
- **Working states:** When the set of the initial state is left, then the system works in a strongly connected set of states in which a majority of testing is performed.
- **The initial state of the working set:** a state of the working set which can be considered as the "initial state".
- **The set of ending state:** If the system reaches this set, then there is not way back to the working set, e.g. a finalizing sequence, a shutdown.
- The system is **fully specified** if transitions and output symbols are defined for all combinations of input symbols and states.
- **A round trip of the state A:** a sequence of transitions going from the state A to a state B and back to the state A.

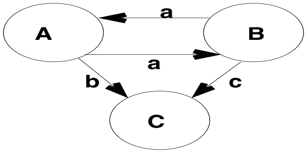
Test Design [Bei95]

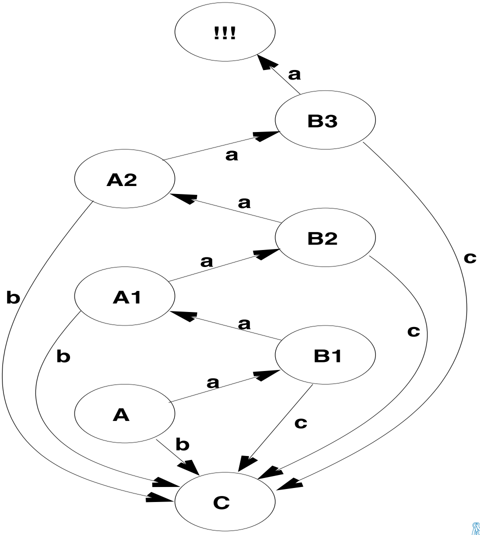
- Each state begins in the initial state.
- The system is transferred
  - from the initial state using the shortest path to the selected state,
  - the given transition is performed,
  - and the system is transferred using the shortest path back into the initial state,
  - i.e. we create a round trip.
- Each test is build upon the preceding simpler tests.
- The input symbol is determined for each transition of the round trip.
- The output symbol is determined for all associated transitions of the round trip.
- **We verify**
  - input codes,
  - output codes,
  - states,
  - each transition.
- **Are all end states reachable?**

Hidden States

- **Is the system in the initial state?**
  - A test cannot be started if there is no confirmation that the system is in the initial state.
  - Applications store their settings in a persistent way.
  - If a previous test fails, in what state is the application?
- **Hidden state:** an unknown state that is different from a given state but it has all transitions with the same input and output codes, i.e. it cannot be distinguish from the given state.
- **Has the implementation hidden states?**
  - During the software testing we might assume conditions that are not valid generally.
    - e.g. we know in which state the state is.
  - Often, we do not dealt with one or two hidden states, but the state space doubles and is multiplied in other way.

Hidden States - Example

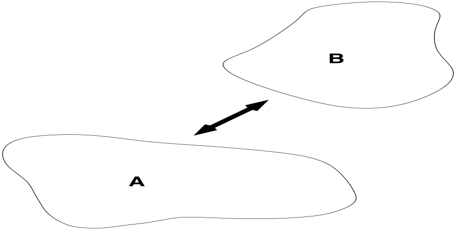




Outline

Finite State Machine Testing [HI98]

- Based on the isomorphism of finite state machines,
- $\mathcal{A} = (Input, Q, \mathbf{F}, q_0)$
- $\mathcal{A}' = (Input, Q', \mathbf{F}', {q_0}')$
- $g : \mathcal{A} \to \mathcal{A}'$
- $g : Q \to Q'$
  1. $g(q_0) = {q_0}'$
  2. $\forall q \in Q, input \in Input,$
     $g(\mathbf{F}(q, input)) = \mathbf{F}'(g(q), input)$



Test Set Construction [HI98, Cho78]

Chow's W method

- Let L be a set of input sequences and q, q' be two states.
- L *distinguishes (CZ rozliší)* the state q from q' if there is a sequence $k \in L$ such that the output sequence obtained by the application of k to the machine in the state q is different from the output sequence obtained by the application of k to the state q'.
- The machine is *minimal* if it does not contain redundant states.
- A set of input sequences W is called *a characterization set* if it can distinguish any two state of the machine.
- **A state cover** is a set L of input sequences such that it is possible to find an element of L using which we can reach the given state from the initial state q_0.
- **A transition cover** of the minimal machine is a set T of input sequences such that it is a state cover closed under the right composition with the input set $Input$.
  - $sequence \in T = L \bullet (Input^1 \cup \{<>\})$

Test Set Generation [HI98, Cho78]

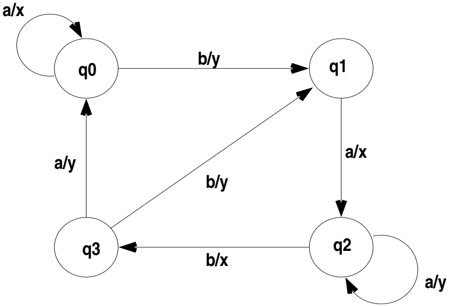
- How many times are there more states than in the specification? (k)
- $Z = Input^k \bullet W \cup Input^{k-1} \bullet W \cup \cdots \cup Input^1 \bullet W \cup W$
  - If A and B are two sets of sequences over the same alphabet, then $A \bullet B$ denotes a set of sequences composed from the sequences of the set A followed by a sequence from B.
  - k steps into an "unknown" space are performed followed by the verification of the state.
- Finite **test set**:

$$T \bullet Z$$

- Transition cover ensures
  - that all state and transition of the specification are implemented.
  - The set Z ensures that the implementation is in the same state as specified.
  - The parameter k ensures that all hidden states into the level k are tested.

Outline

A Simple Example [HI98]



- $Input = \{a, b\}$
- $L = \{<>, b, b :: a, b :: a :: b\}$, $<>$ . . . null input sequence
- $T = \{<>, a, b, b :: a, b :: b, b :: a :: a, b :: a :: b, b :: a :: b :: a, b :: a :: b :: b\}$
- $W = \{a, b\}$ [Chy84], pp. 31–34
- $$\begin{aligned} Z &= Input \bullet W \cup W \\ &= \{a, b\} \bullet \{a, b\} \cup \{a, b\} \\ &= \{a, b, a :: a, a :: b, b :: a, b :: b\} \end{aligned}$$

Test Set of the Example [HI98]

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

Test Set of the Example [HI98]

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

Test Set of the Example [HI98]

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

Test Set of the Example [HI98]

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

Test Set of the Example [HI98]

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

Applications [Bei95]

- Menu driven software,
- Object-oriented software,
- Protocols,
- Device drivers,
- Legacy hardware,
- Microcomputers of industrial and home devices,
- Software instalation,
- Archive and backup software,
- Safety software models,
- WEB applications.

Outline

1 Finite State Machine
- Definitions

2 Finite state machine testing
- Terminology
- Formal FSM Testing
- Example
- Characterization Set Construction

Mealy Machine [Mea55, Mat13]

Definition 1 (Mealy machine with a finite number of states is)

- 6-tuple $M(X, Y, Q, q_0, \delta, \lambda)$:
- X is a finite set of input symbols (the input alphabet),
- Y is a finite set of output symbols (the output alphabet)
- Q is a finite set of state,
- $q_0 \in Q$ is the initial state,
- $D \subseteq Q \times X$ is a specification domain,
- $\delta : D \to Q$ is a state transition function,
- $\lambda : D \to Y$ is an output function.

- If $D = Q \times X$, then M is a **complete** Mealy machine [SP10].
- A sequence $\alpha = x_1 \ldots x_k, \alpha \in I^*$ is **a defined input sequence** for a state $q \in Q$ if there are $q_1, \ldots, q_{k+1} \in Q$, where $q_1 = q$ such that $(q_i, x_i) \in D$ and $\delta(q_i, x_i) = q_{i+1}$ for all $1 \leq i \leq k$.

machine Minimality [SP10, Mat13]

Let $M(X, Y, Q, q_0, \delta, \lambda)$ be a Mealy machine with a finite number of states.

- Extended transition and state functions applied to an input symbol x of a defined input sequence α including the empty sequence ϵ:
  - for $q \in Q$, $\delta(q, \epsilon) = q$ and $\lambda(q, \epsilon) = \epsilon$
  - $\delta(q, \alpha x) = \delta(\delta(q, \alpha), x)$
  - $\lambda(q, \alpha x) = \lambda(\delta(q, \alpha), x)$
- $\Omega(q)$ is the set of all defined input sequences for state $q \in Q$.
- Two states $q, q' \in Q$ are **distinguishable**,
  if there is $\gamma \in \Omega(q) \cap \Omega(q')$ such that $\lambda(q, \gamma) \neq \lambda(q', \gamma)$.
  Then, we say that γ **distinghishes** the states q and q'.
- Two states $q_1, q_2 \in Q; q_1 \neq q_2$ are **state equivalent**,
  if they lead to the same of equivalent states after an application of any input sequence.
- M is **minimal** if no its two states are equivalent [Ner58, Gil60].

C-equivalence of States [SP10, Mat13]

Let $M(X, Y, Q, q_0, \delta, \lambda)$ be a Mealy machine with a finite number of states.

- Let $C \subseteq \Omega(q) \cap \Omega(q')$ be a set.
- The states $q_1, q_2 \in Q$ are C-**equivalent**, if $\lambda(q, \gamma) \neq \lambda(q', \gamma)$ for all $\gamma \in C$.

Two machines $M_1(X, Y, Q_1, q_0^1, \delta_1, \lambda_1)$ and $M_2(X, Y, Q_2, q_0^2, \delta_2, \lambda_2)$ are **equivalent**, if

1. for each state $q \in M_1$ there is $q' \in M_2$ such that q and q' are equivalent and
2. for each state $q \in M_2$ there is $q' \in M_1$ such that q and q' are equivalent.

k-**equivalence**

- Let $M_1(X, Y, Q_1, q_0^1, \delta_1, \lambda_1)$ and $M_2(X, Y, Q_2, q_0^2, \delta_2, \lambda_2)$ be two machines.
- The states $q_i \in Q_1$ and $q_j \in Q_2$ are considered to be k-**equivalent**, if they produce identical output sequences after excited with any input sequence of the length k.

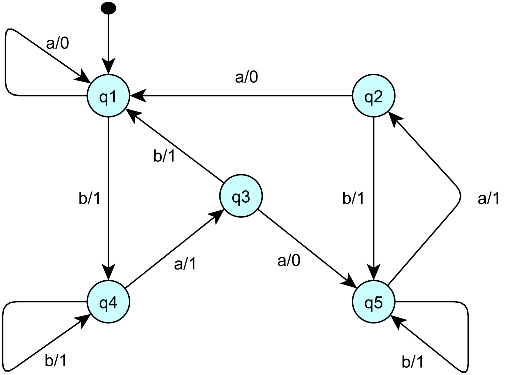
Characterization set W [SP10, Mat13]

Let $M(X, Y, Q, q_0, \delta, O)$ be a minimal and complete Mealy machine with a finite number of states.

- W is a finite set of input sequences that distinguishes any pair of different states $q_i, q_j \in Q$.
- Each input sequence $\gamma \in W$ has a finite length.
- For each pair of different states $q_i, q_j \in Q$ the set W contains at least one input sequence γ such that

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

Characterization Set Example [HI98]



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

k-equivalence Partition of States Q [Mat13]

- k-equivalence partition of states Q, denoted as P_k, is a collection of n finite sets $\Sigma_{k,1}, \Sigma_{k,2}, \ldots, \Sigma_{k,n}$ such that

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

- The states in $\Sigma_{k,i}$ are k-equivalent.
- If states $q_{\ell_1} \in \Sigma_{k,j}$ and $q_{\ell_2} \in \Sigma_{k,j}$ for $i \neq j$, then q_{ℓ_1} and q_{ℓ_2} are k-distinguishable.

W Set Construction [Mat13]

The Algorithm

1. Create a sequence of k-equivalence partitions of states Q denoted as $P_1, P_2, \ldots, P_m, m > 0$
2. Backward search k-equivalence partitions while constructing distinguishing sequences for each pair of the states.

- Algorithm convergence is guaranteed.
- When the algorithm stops each class $\Sigma_{K,j}$ of the finite partition P_K defines a class of equivalent states (1 for minimal machines).

Informally:

- First, find what can be distinguished in one step,
- then in two steps,
- etc.

W Set Construction [Mat13]

Tabular representation M.
0-equivalence partition $P_0 = \{\Sigma_1 = \{q_1, q_2, q_3, q_4, q_5\}\}$

Current state	**Output**		**Next state**	
	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

1-equivalence Partition P_1 Construction [Mat13]

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

Σ	**Current state**	**Output**		**Next state**	
		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

2-equivalence Partition Construction: P_1 Rewrite [Mat13]

Rewrite P_1, state q_i is replaced by $q_{i,j}$ if $q_i \in \Sigma_j$.

Σ	Current state	Next state	
		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	$q_{2,1}$	$q_{5,2}$

2-equivalence Partition Construction: P_2 Construction [Mat13]

Construct P_2. Divide $\Sigma_{1,j}$ with regard to the groups of next states.

Σ	Current state	Next state	
		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	$q_{2,1}$	$q_{5,3}$

3-equivalence Partition Construction: P_3 Construction [Mat13]

Construct P_3. Divide $\Sigma_{2,j}$ with regard to the groups of next states.

Σ	**Current state**	**Next state** a	**Next state** 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	$q_{2,1}$	$q_{5,4}$

4-equivalence Partition Construction: P_4 Construction [Mat13]

Construct P_4. Divide $\Sigma_{3,j}$ with regard to the groups of next states.

Σ	Current state	Next state a	Next state 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	$q_{2,2}$	$q_{5,5}$

Distinguishing Sequence Construction: Example [Mat13]

1. Find a distinguishing sequence of the states q_1 a q_2.
2. Init the distinguishing sequence: $z = \epsilon$.
3. Find tables P_i and P_{i+1} such that (q_1, q_2) are in the same group in P_i and in different groups in P_{i+1}:
   - P_3 and P_4 are obtained.
4. Find the input symbol distinguishing q_1 and q_2 in table P_3
   - b is the distinguishing symbol.
   - Update the distinguishing sequence: $z := z.b = \epsilon.b = b$.
5. Find the next states if the symbol b is applied to q_1 and q_2,
   - q_4 and q_5 are obtained.
6. Find tables P_i and P_{i+1} such that (q_4, q_5) are in the same group in P_i and in different groups in P_{i+1}:
   - P_2 and P_3 are obtained.
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. Repeat for each pair (q_i, q_j): $W = \{a, aa, aaa, baaa\}$

Summary

- Finite state machines
- How to test finite state machines
- Test set construction using Chow's W method
- Characterization set construction

Competencies

- Define finite state machine.
- Describe the concept of hidden states.
- Describe Chow's W method of test set construction.
- Define characterization set and describe its construction algorithm.

References 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.

[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.

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