BLG311E Formal Languages and Automata

Finite State Machines

A.Emre Harmancı Tolga Ovatman Ö.Sinan Saraç

2015

Outline

Computing Machines

Computer

A computer is a general purpose *machine* which can be programmed to carry out a finite set of arithmetic or logical operations(*computation*).

Computing Machines

Machine

A machine is a tool consisting of one or more generally moving parts that is constructed to achieve a particular goal. However, the advent of electronics technology has led to the development of devices without moving parts that are considered machines.

Computing Machines

Abstract Machine

An abstract machine, is a theoretical model of a computer hardware or software system.

Finite State Machine(FSM)

An FSM has a mathmematical model defined by a quintuple $(S, I, O, \delta, \omega)$, where:

- S: Set of states.
- I: Input alphabet (a finite, non-empty set of symbols)
- O: Output alphabet (a finite, non-empty set of symbols)
- δ: The transition function defined on $I \times S \rightarrow S$
- ω: The output function defined on either $S \rightarrow O$ or $I \times S \rightarrow O$

FSM properties

A finite state machine should hold the following properties

1. It has finite input and output alphabets
2. It is deterministic

Deterministic Machine

For a deterministic machine the outcome of a transition from one state to another given a certain input can be predicted for every occurrence. In a deterministic finite state machine, for each pair of state and input symbol there is one and only one transition to a next state.

FSM properties

A finite state machine should hold the following properties

1. It has finite input and output alphabets
2. It is deterministic

Deterministic Machine

For a deterministic machine the outcome of a transition from one state to another given a certain input can be predicted for every occurrence. In a deterministic finite state machine, for each pair of state and input symbol there is one and only one transition to a next state.

FSM properties

A finite state machine should hold the following properties

1. It has finite input and output alphabets
2. It is deterministic

Deterministic Machine

For a deterministic machine the outcome of a transition from one state to another given a certain input can be predicted for every occurrence. In a deterministic finite state machine, for each pair of state and input symbol there is one and only one transition to a next state.

FSM properties

A finite state machine should hold the following properties

3. It has transducer capability.

Transducer

A transducer machine has two alphabets: an input alphabet and an output alphabet. Transducers are said to be able to *transform* inputs to output.

FSM properties

A finite state machine should hold the following properties

3. It has transducer capability.

Transducer

A transducer machine has two alphabets: an input alphabet and an output alphabet. Transducers are said to be able to *transform* inputs to output.

Transducers

When realizing transducers with digital circuits the concept of discrete-time is used.

Discrete Time

Discrete time is the discontinuity of a function's time domain that results from sampling a variable at a finite interval. One of the fundamental concepts behind discrete time is an implied (actual or hypothetical) system clock.

Transducers

In sequential synchronous digital circuits both the input and output functions are determined by combinational circuits. On the other hand, in determining next state of the circuit the characteristic equation of the flip-flop becomes effective as well. In a formal way
$S(t^+) = Q(S(t), \delta(S(t), I(t)))$

- SR type: $q^+ = S + R'q$
- D type: $q^+ = D$
- JK type: $q^+ = Jq' + K'q$
- T type: $q^+ = T \oplus q$

For the case of a D flip-flop the next state is determined by
$S(t^+) = \delta(S(t), I(t))$

Transducers

In sequential synchronous digital circuits both the input and output functions are determined by combinational circuits. On the other hand, in determining next state of the circuit the characteristic equation of the flip-flop becomes effective as well. In a formal way
$S(t^+) = Q(S(t), \delta(S(t), I(t)))$

- SR type: $q^+ = S + R'q$
- D type: $q^+ = D$
- JK type: $q^+ = Jq' + K'q$
- T type: $q^+ = T \oplus q$

For the case of a D flip-flop the next state is determined by
$S(t^+) = \delta(S(t), I(t))$

Transducers

Transducers holds some certain properties in digital discrete-time systems.

- Sequence: Discretization is performed by the sequence order of the labels. n is the length of the sequence.
- $[\Lambda]$: A singleton set which only includes an *empty string*.
- I^*: Set of input sequences : $I^* = [\Lambda] \cup I \cup I^2 \cup \ldots \cup I^n \ldots$
- O^*: Set of output sequences : $O^* = [\Lambda] \cup O \cup O^2 \cup \ldots \cup O^n \ldots$
- Suppose that the transducer performs $w = f(x)$ transformation where $w \in O$ and $x \in I$. Function f holds following properties:
  - Length preservation: $|w| = |x| = n; n \in \mathbb{N}$
  - Prefix inclusivity:
    $(x = x_1x_2) \wedge (w = w_1w_2) \wedge (|x_1| = |w_1|) \Rightarrow w_1 = f(x_1)$

An example Machine

Suppose we want to design a machine that reads numerical codes input from a keypad and accepts *only one* certain key combination. For this particular case$(S, I, O, \delta, \omega)$ can be defined as:

- $I \in \{0, 1, 2, \ldots, 9\}$. We may want to restrict the number of digits to 4, for example $1234 \in I^4$
- S: At each keystroke we need to control if the correct digit is input. In our case $|S| = 5$ or $s_i \in S : 0 \leq i \leq 4$
- $O = \{open, closed\}$ is the output alphabet.
- Our machine is going to accept the code after 4 correct inputs. The final state $F = s_4$. Let's ignore any wrong combinations at this moment for the sake of simplicity.
- Since we only have one correct combination δ can be defined as follows δ :
  $(s_0, 1) \rightarrow s_1$
  $(s_1, 9) \rightarrow s_2$
  $(s_2, 0) \rightarrow s_3$
  $(s_3, 3) \rightarrow s_4$
- Our output function ω maps $s_i \rightarrow closed$ where $0 \leq i \leq 3$ and $s_4 \rightarrow open$

An example Machine

Suppose we want to design a machine that reads numerical codes input from a keypad and accepts *only one* certain key combination. For this particular case$(S, I, O, \delta, \omega)$ can be defined as:

- $I \in \{0, 1, 2, \ldots, 9\}$. We may want to restrict the number of digits to 4, for example $1234 \in I^4$
- S: At each keystroke we need to control if the correct digit is input. In our case $|S| = 5$ or $s_i \in S : 0 \leq i \leq 4$
- $O = \{open, closed\}$ is the output alphabet.
- Our machine is going to accept the code after 4 correct inputs. The final state $F = s_4$. Let's ignore any wrong combinations at this moment for the sake of simplicity.
- Since we only have one correct combination δ can be defined as follows δ :
  $(s_0, 1) \rightarrow s_1$
  $(s_1, 9) \rightarrow s_2$
  $(s_2, 0) \rightarrow s_3$
  $(s_3, 3) \rightarrow s_4$
- Our output function ω maps $s_i \rightarrow closed$ where $0 \leq i \leq 3$ and $s_4 \rightarrow open$

An example Machine

Suppose we want to design a machine that reads numerical codes input from a keypad and accepts *only one* certain key combination. For this particular case$(S, I, O, \delta, \omega)$ can be defined as:

- $I \in \{0, 1, 2, \ldots, 9\}$. We may want to restrict the number of digits to 4, for example $1234 \in I^4$
- S: At each keystroke we need to control if the correct digit is input. In our case $|S| = 5$ or $s_i \in S : 0 \leq i \leq 4$
- $O = \{open, closed\}$ is the output alphabet.
- Our machine is going to accept the code after 4 correct inputs. The final state $F = s_4$. Let's ignore any wrong combinations at this moment for the sake of simplicity.
- Since we only have one correct combination δ can be defined as follows δ :
  $(s_0, 1) \rightarrow s_1$
  $(s_1, 9) \rightarrow s_2$
  $(s_2, 0) \rightarrow s_3$
  $(s_3, 3) \rightarrow s_4$
- Our output function ω maps $s_i \rightarrow closed$ where $0 \leq i \leq 3$ and $s_4 \rightarrow open$

An example Machine

Suppose we want to design a machine that reads numerical codes input from a keypad and accepts *only one* certain key combination. For this particular case$(S, I, O, \delta, \omega)$ can be defined as:

- $I \in \{0, 1, 2, \ldots, 9\}$. We may want to restrict the number of digits to 4, for example $1234 \in I^4$
- S: At each keystroke we need to control if the correct digit is input. In our case $|S| = 5$ or $s_i \in S : 0 \leq i \leq 4$
- $O = \{open, closed\}$ is the output alphabet.
- Our machine is going to accept the code after 4 correct inputs. The final state $F = s_4$. Let's ignore any wrong combinations at this moment for the sake of simplicity.
- Since we only have one correct combination δ can be defined as follows δ :
  $(s_0, 1) \rightarrow s_1$
  $(s_1, 9) \rightarrow s_2$
  $(s_2, 0) \rightarrow s_3$
  $(s_3, 3) \rightarrow s_4$
- Our output function ω maps $s_i \rightarrow closed$ where $0 \leq i \leq 3$ and $s_4 \rightarrow open$

An example Machine

Suppose we want to design a machine that reads numerical codes input from a keypad and accepts *only one* certain key combination. For this particular case$(S, I, O, \delta, \omega)$ can be defined as:

- $I \in \{0, 1, 2, \dots, 9\}$. We may want to restrict the number of digits to 4, for example $1234 \in I^4$
- S: At each keystroke we need to control if the correct digit is input. In our case $|S| = 5$ or $s_i \in S : 0 \leq i \leq 4$
- $O = \{open, closed\}$ is the output alphabet.
- Our machine is going to accept the code after 4 correct inputs. The final state $F = s_4$. Let's ignore any wrong combinations at this moment for the sake of simplicity.
- Since we only have one correct combination δ can be defined as follows δ :
  $(s_0, 1) \rightarrow s_1$
  $(s_1, 9) \rightarrow s_2$
  $(s_2, 0) \rightarrow s_3$
  $(s_3, 3) \rightarrow s_4$
- Our output function ω maps $s_i \rightarrow closed$ where $0 \leq i \leq 3$ and $s_4 \rightarrow open$

An example Machine

Suppose we want to design a machine that reads numerical codes input from a keypad and accepts *only one* certain key combination. For this particular case$(S, I, O, \delta, \omega)$ can be defined as:

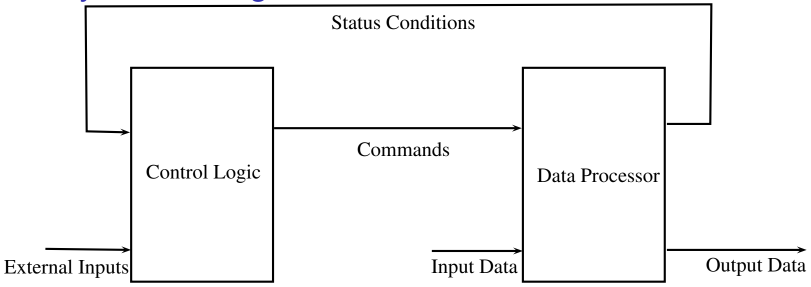
- $I \in \{0, 1, 2, \ldots, 9\}$. We may want to restrict the number of digits to 4, for example $1234 \in I^4$
- S: At each keystroke we need to control if the correct digit is input. In our case $|S| = 5$ or $s_i \in S : 0 \leq i \leq 4$
- $O = \{open, closed\}$ is the output alphabet.
- Our machine is going to accept the code after 4 correct inputs. The final state $F = s_4$. Let's ignore any wrong combinations at this moment for the sake of simplicity.
- Since we only have one correct combination δ can be defined as follows δ :
  $(s_0, 1) \rightarrow s_1$
  $(s_1, 9) \rightarrow s_2$
  $(s_2, 0) \rightarrow s_3$
  $(s_3, 3) \rightarrow s_4$
- Our output function ω maps $s_i \rightarrow closed$ where $0 \leq i \leq 3$ and $s_4 \rightarrow open$

An example Machine

Suppose we want to design a machine that reads numerical codes input from a keypad and accepts *only one* certain key combination. For this particular case$(S, I, O, \delta, \omega)$ can be defined as:

- $I \in \{0, 1, 2, \ldots, 9\}$. We may want to restrict the number of digits to 4, for example $1234 \in I^4$
- S: At each keystroke we need to control if the correct digit is input. In our case $|S| = 5$ or $s_i \in S : 0 \le i \le 4$
- $O = \{open, closed\}$ is the output alphabet.
- Our machine is going to accept the code after 4 correct inputs. The final state $F = s_4$. Let's ignore any wrong combinations at this moment for the sake of simplicity.
- Since we only have one correct combination δ can be defined as follows δ :
  $(s_0, 1) \rightarrow s_1$
  $(s_1, 9) \rightarrow s_2$
  $(s_2, 0) \rightarrow s_3$
  $(s_3, 3) \rightarrow s_4$
- Our output function ω maps $s_i \rightarrow closed$ where $0 \le i \le 3$ and $s_4 \rightarrow open$

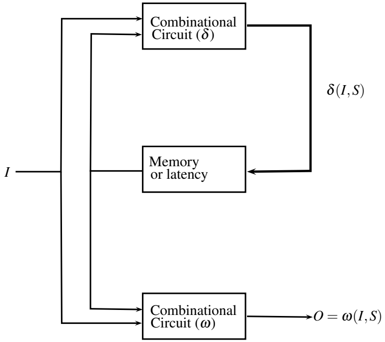
Digital System Design



- When designing digital hardware, a general approach is to handle data processing and control operations separately.
- The control logic and data processing tasks of a digital system are specified by means of a hardware algorithm.
- One of the common ways in representing an algorithm is by using a flowchart.

Machine Types

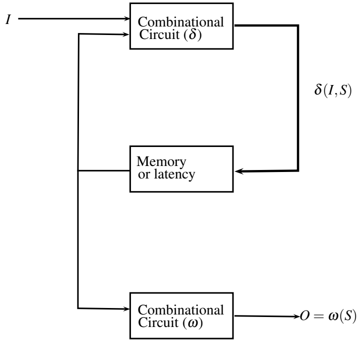
A Mealy machine[1] is a finite-state machine whose output values are determined both by its current state and the current inputs $I \times S \rightarrow O$.


Combinational Circuit (δ)
$\delta(I,S)$
I
Memory or latency
Combinational Circuit (ω)
$O = \omega(I,S)$

[1] The Mealy machine is named after George H. Mealy, who presented the concept in a 1955 paper, "A Method for Synthesizing Sequential Circuits"

Machine Types

A Moore machine[2] is a finite-state machine, whose output values are determined solely by its current state $S \rightarrow O$.

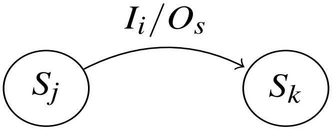

I
Combinational Circuit (δ)
$\delta(I,S)$
Memory or latency
Combinational Circuit (ω)
$O = \omega(S)$

[2]The Moore machine is named after Edward F. Moore, who presented the concept in a 1956 paper, "Gedanken-experiments on Sequential Machines".

Modeling FSMs

Following elements can be used in modeling FSMs:
Mealy Machine

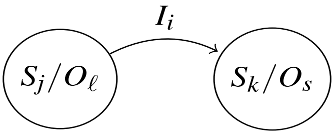
	I_1	$\dots$	I_i	$\dots$	I_m
S_1					
$\dots$					
S_j			S_k/O_s		
$\dots$					
S_n					



$$\begin{tikzcd} S_j \arrow[r, bend left, "I_i/O_s"] & S_k \end{tikzcd}$$

Moore Machine

	I_1	$\dots$	I_i	$\dots$	I_m	O
S_1						O_1
$\dots$						$\dots$
S_j			S_k			O_ℓ
$\dots$						$\dots$
S_n						O_p



$$\begin{tikzcd} S_j/O_\ell \arrow[r, bend left, "I_i"] & S_k/O_s \end{tikzcd}$$

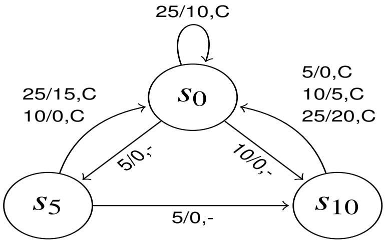
Example

A coffee vending machine that accepts coins of 5, 10 and 25 cents, gives coffe for 15 cents and returns the change. Let's build Mealy and Moore models of this machine.
Mealy model, $S_x/i,j$: x denotes money input so far, i change return and j holds C for coffee output and $-$ for no output.

State table:

	5	10	25
S_0	$S_5/0,-$	$S_{10}/0,-$	$S_0/10,C$
S_5	$S_{10}/0,-$	$S_0/0,C$	$S_0/15,C$
S_{10}	$S_0/0,C$	$S_0/5,C$	$S_0/20,C$

State diagram:



Example

For Moore model the number of states need to be at least the number of different State/output pairs in the Mealy model. States should not be assigned to the coins input. When no coins are input current state is preserved.

State table:

	5	10	25	Output
S_0	S_5	S_{10}	S_{25}	0,-
S_5	S_{10}	S_{15}	S_{30}	0,-
S_{10}	S_{15}	S_{20}	S_{35}	0,-
S_{15}	S_5	S_{10}	S_{25}	0,C
S_{20}	S_5	S_{10}	S_{25}	5,C
S_{25}	S_5	S_{10}	S_{25}	10,C
S_{30}	S_5	S_{10}	S_{25}	15,C
S_{35}	S_5	S_{10}	S_{25}	20,C

We can map Mealy model transitions to Moore states of this machines as follows

$S_5/0,- \rightarrow S_5$
$S_{10}/0,- \rightarrow S_{10}$
$S_0/0,C \rightarrow S_{15}$
$S_0/5,C \rightarrow S_{20}$
$S_0/10,C \rightarrow S_{25}$
$S_0/15,C \rightarrow S_{30}$
$S_0/20,C \rightarrow S_{35}$

Example

For Moore model the number of states need to be at least the number of different State/output pairs in the Mealy model. States should not be assigned to the coins input. When no coins are input current state is preserved.

State table:

	5	10	25	Output
S_0	S_5	S_{10}	S_{25}	0,-
S_5	S_{10}	S_{15}	S_{30}	0,-
S_{10}	S_{15}	S_{20}	S_{35}	0,-
S_{15}	S_5	S_{10}	S_{25}	0,C
S_{20}	S_5	S_{10}	S_{25}	5,C
S_{25}	S_5	S_{10}	S_{25}	10,C
S_{30}	S_5	S_{10}	S_{25}	15,C
S_{35}	S_5	S_{10}	S_{25}	20,C

We can map Mealy model transitions to Moore states of this machines as follows

$S_5/0,- \rightarrow S_5$

$S_{10}/0,- \rightarrow S_{10}$

$S_0/0,C \rightarrow S_{15}$

$S_0/5,C \rightarrow S_{20}$

$S_0/10,C \rightarrow S_{25}$

$S_0/15,C \rightarrow S_{30}$

$S_0/20,C \rightarrow S_{35}$

Outline

State equivalence

During a system's design iterations designers may come up with non-optimal state tables containing equivalent states. In order to overcome this situation mathematical definition of equivalency and state reduction principles are used.

Equivalence Relation

A given binary relation $\sim$ on a set S is said to be an equivalence relation if and only if it is reflexive, symmetric and transitive. Equivalently, $\forall s_i, s_j, s_k \in S$:

- $s_i \sim s_i$
- $s_i \sim s_j \Rightarrow s_j \sim s_i$
- $s_i \sim s_k \wedge s_k \sim s_j \Rightarrow s_i \sim s_j$

State equivalence

Using the mathematical definition of equivalency the set of machine states can be partitioned into equivalence classes. Equivelance classes should include

- Explicitly equivalent states
- Implicitly equivalent states, whose equivalency depends other states' equivalency.

State Equivalency Conditions

The necessary and sufficient conditions for two states in a state table to be equivalent are: For all the inputs of those states

- Outputs should be the same
- Successing conditions should
  - be explicitly or implicitly equivalent
  - not have any outputs that doesn't conform equivalency

Dependency identification of states can be performed using dependency tables and undirected relation graphs. Let's consider the following state table.

S/I	I_1	I_2	I_3	I_4
S_1	$S_1/0$	$S_2/1$	$S_5/1$	$S_1/0$
S_2	$S_2/1$	$S_2/0$	$S_6/0$	$S_3/0$
S_3	$S_3/1$	$S_7/0$	$S_4/0$	$S_3/0$
S_4	$S_1/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_5	$S_5/0$	$S_6/1$	$S_5/1$	$S_5/0$
S_6	$S_2/1$	$S_6/0$	$S_2/0$	$S_3/0$
S_7	$S_8/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_8	$S_8/0$	$S_6/1$	$S_5/1$	$S_1/0$

We shall build a dependency table step by step based on the state dependencies.

S/I	I_1	I_2	I_3	I_4
S_1	$S_1/0$	$S_2/1$	$S_5/1$	$S_1/0$
S_2	$S_2/1$	$S_2/0$	$S_6/0$	$S_3/0$
S_3	$S_3/1$	$S_7/0$	$S_4/0$	$S_3/0$
S_4	$S_1/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_5	$S_5/0$	$S_6/1$	$S_5/1$	$S_5/0$
S_6	$S_2/1$	$S_6/0$	$S_2/0$	$S_3/0$
S_7	$S_8/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_8	$S_8/0$	$S_6/1$	$S_5/1$	$S_1/0$

	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_2							
S_3							
S_4							
S_5							
S_6							
S_7							
S_8							

S/I	I_1	I_2	I_3	I_4
S_1	$S_1/0$	$S_2/1$	$S_5/1$	$S_1/0$
S_2	$S_2/1$	$S_2/0$	$S_6/0$	$S_3/0$
S_3	$S_3/1$	$S_7/0$	$S_4/0$	$S_3/0$
S_4	$S_1/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_5	$S_5/0$	$S_6/1$	$S_5/1$	$S_5/0$
S_6	$S_2/1$	$S_6/0$	$S_2/0$	$S_3/0$
S_7	$S_8/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_8	$S_8/0$	$S_6/1$	$S_5/1$	$S_1/0$

	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_2	**X**						
S_3							
S_4							
S_5							
S_6							
S_7							
S_8							

S/I	I_1	I_2	I_3	I_4
S_1	$S_1/0$	$S_2/1$	$S_5/1$	$S_1/0$
S_2	$S_2/1$	$S_2/0$	$S_6/0$	$S_3/0$
S_3	$S_3/1$	$S_7/0$	$S_4/0$	$S_3/0$
S_4	$S_1/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_5	$S_5/0$	$S_6/1$	$S_5/1$	$S_5/0$
S_6	$S_2/1$	$S_6/0$	$S_2/0$	$S_3/0$
S_7	$S_8/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_8	$S_8/0$	$S_6/1$	$S_5/1$	$S_1/0$

	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_2	X						
S_3	**X**						
S_4	(2,7)-(4,5)-**(1,3)**						
S_5							
S_6							
S_7							
S_8							

S/I	I_1	I_2	I_3	I_4
S_1	$S_1/0$	$S_2/1$	$S_5/1$	$S_1/0$
S_2	$S_2/1$	$S_2/0$	$S_6/0$	$S_3/0$
S_3	$S_3/1$	$S_7/0$	$S_4/0$	$S_3/0$
S_4	$S_1/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_5	$S_5/0$	$S_6/1$	$S_5/1$	$S_5/0$
S_6	$S_2/1$	$S_6/0$	$S_2/0$	$S_3/0$
S_7	$S_8/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_8	$S_8/0$	$S_6/1$	$S_5/1$	$S_1/0$

	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_2	X						
S_3	X						
S_4	X						
S_5	**(2,6)**						
S_6							
S_7							
S_8							

S/I	I_1	I_2	I_3	I_4
S_1	$S_1/0$	$S_2/1$	$S_5/1$	$S_1/0$
S_2	$S_2/1$	$S_2/0$	$S_6/0$	$S_3/0$
S_3	$S_3/1$	$S_7/0$	$S_4/0$	$S_3/0$
S_4	$S_1/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_5	$S_5/0$	$S_6/1$	$S_5/1$	$S_5/0$
S_6	$S_2/1$	$S_6/0$	$S_2/0$	$S_3/0$
S_7	$S_8/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_8	$S_8/0$	$S_6/1$	$S_5/1$	$S_1/0$

	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_2	X						
S_3	X	(2,7)(4,6)					
S_4	X	X					
S_5	(2,6)	X					
S_6	X	○					
S_7	(1,8)(2,7)(4,5)(1,3)						
S_8	(2,6)						

S/I	I_1	I_2	I_3	I_4
S_1	$S_1/0$	$S_2/1$	$S_5/1$	$S_1/0$
S_2	$S_2/1$	$S_2/0$	$S_6/0$	$S_3/0$
S_3	$S_3/1$	$S_7/0$	$S_4/0$	$S_3/0$
S_4	$S_1/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_5	$S_5/0$	$S_6/1$	$S_5/1$	$S_5/0$
S_6	$S_2/1$	$S_6/0$	$S_2/0$	$S_3/0$
S_7	$S_8/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_8	$S_8/0$	$S_6/1$	$S_5/1$	$S_1/0$

	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_2	X						
S_3	X	(2,7)(4,6)					
S_4	X	X					
S_5	(2,6) ○	X					
S_6	X	○					
S_7	(1,8)(2,7)(4,5)(1,3)						
S_8	(2,6) ○						

S/I	I_1	I_2	I_3	I_4
S_1	$S_1/0$	$S_2/1$	$S_5/1$	$S_1/0$
S_2	$S_2/1$	$S_2/0$	$S_6/0$	$S_3/0$
S_3	$S_3/1$	$S_7/0$	$S_4/0$	$S_3/0$
S_4	$S_1/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_5	$S_5/0$	$S_6/1$	$S_5/1$	$S_5/0$
S_6	$S_2/1$	$S_6/0$	$S_2/0$	$S_3/0$
S_7	$S_8/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_8	$S_8/0$	$S_6/1$	$S_5/1$	$S_1/0$

	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_2	X						
S_3	X	(2,7)(4,6)**X**					
S_4	X	X					
S_5	(2,6) ○	X					
S_6	X	○					
S_7	(1,8)(2,7)(4,5)(1,3)**X**	**X**					
S_8	(2,6) ○						

S/I	I_1	I_2	I_3	I_4
S_1	$S_1/0$	$S_2/1$	$S_5/1$	$S_1/0$
S_2	$S_2/1$	$S_2/0$	$S_6/0$	$S_3/0$
S_3	$S_3/1$	$S_7/0$	$S_4/0$	$S_3/0$
S_4	$S_1/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_5	$S_5/0$	$S_6/1$	$S_5/1$	$S_5/0$
S_6	$S_2/1$	$S_6/0$	$S_2/0$	$S_3/0$
S_7	$S_8/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_8	$S_8/0$	$S_6/1$	$S_5/1$	$S_1/0$

	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_2	X						
S_3	X	**X**					
S_4	X	X	X				
S_5	(2,6) ○	X	X				
S_6	X	○	**(2,3)(6,7)(2,4)**				
S_7	X	X					
S_8	(2,6) ○	X					

S/I	I_1	I_2	I_3	I_4
S_1	$S_1/0$	$S_2/1$	$S_5/1$	$S_1/0$
S_2	$S_2/1$	$S_2/0$	$S_6/0$	$S_3/0$
S_3	$S_3/1$	$S_7/0$	$S_4/0$	$S_3/0$
S_4	$S_1/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_5	$S_5/0$	$S_6/1$	$S_5/1$	$S_5/0$
S_6	$S_2/1$	$S_6/0$	$S_2/0$	$S_3/0$
S_7	$S_8/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_8	$S_8/0$	$S_6/1$	$S_5/1$	$S_1/0$

	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_2	X						
S_3	X	X					
S_4	X	X	X				
S_5	(2,6) ○	X	**X**	(1,5)(6,7)**(3,5)**			
S_6	X	○	X				
S_7	X	X	X				
S_8	(2,6) ○	X	X				

S/I	I_1	I_2	I_3	I_4
S_1	$S_1/0$	$S_2/1$	$S_5/1$	$S_1/0$
S_2	$S_2/1$	$S_2/0$	$S_6/0$	$S_3/0$
S_3	$S_3/1$	$S_7/0$	$S_4/0$	$S_3/0$
S_4	$S_1/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_5	$S_5/0$	$S_6/1$	$S_5/1$	$S_5/0$
S_6	$S_2/1$	$S_6/0$	$S_2/0$	$S_3/0$
S_7	$S_8/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_8	$S_8/0$	$S_6/1$	$S_5/1$	$S_1/0$

	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_2	X						
S_3	X	X					
S_4	X	X	X				
S_5	(2,6) ○	X	X	X			
S_6	X	○	X	X			
S_7	X	X	X	**(1,8)**○			
S_8	(2,6) ○	X	X				

S/I	I_1	I_2	I_3	I_4
S_1	$S_1/0$	$S_2/1$	$S_5/1$	$S_1/0$
S_2	$S_2/1$	$S_2/0$	$S_6/0$	$S_3/0$
S_3	$S_3/1$	$S_7/0$	$S_4/0$	$S_3/0$
S_4	$S_1/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_5	$S_5/0$	$S_6/1$	$S_5/1$	$S_5/0$
S_6	$S_2/1$	$S_6/0$	$S_2/0$	$S_3/0$
S_7	$S_8/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_8	$S_8/0$	$S_6/1$	$S_5/1$	$S_1/0$

	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_2	X						
S_3	X	X					
S_4	X	X	X				
S_5	(2,6) ○	X	X	**X**			
S_6	X	○	X	X			
S_7	X	X	X	(1,8)○			
S_8	(2,6) ○	X	X	(1,8)(7,6)**(4,5)**			

S/I	I_1	I_2	I_3	I_4
S_1	$S_1/0$	$S_2/1$	$S_5/1$	$S_1/0$
S_2	$S_2/1$	$S_2/0$	$S_6/0$	$S_3/0$
S_3	$S_3/1$	$S_7/0$	$S_4/0$	$S_3/0$
S_4	$S_1/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_5	$S_5/0$	$S_6/1$	$S_5/1$	$S_5/0$
S_6	$S_2/1$	$S_6/0$	$S_2/0$	$S_3/0$
S_7	$S_8/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_8	$S_8/0$	$S_6/1$	$S_5/1$	$S_1/0$

	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_2	X						
S_3	X	X					
S_4	X	X	X				
S_5	(2,6) ○	X	X	**X**			
S_6	X	○	X	X	X		
S_7	X	X	X	(1,8)○	(5,8)**(4,5)(6,7)(3,5)**		
S_8	(2,6) ○	X	X	X			

S/I	I_1	I_2	I_3	I_4
S_1	$S_1/0$	$S_2/1$	$S_5/1$	$S_1/0$
S_2	$S_2/1$	$S_2/0$	$S_6/0$	$S_3/0$
S_3	$S_3/1$	$S_7/0$	$S_4/0$	$S_3/0$
S_4	$S_1/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_5	$S_5/0$	$S_6/1$	$S_5/1$	$S_5/0$
S_6	$S_2/1$	$S_6/0$	$S_2/0$	$S_3/0$
S_7	$S_8/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_8	$S_8/0$	$S_6/1$	$S_5/1$	$S_1/0$

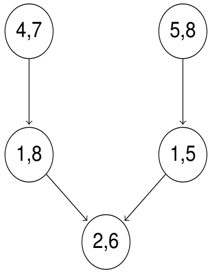
	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_2	X						
S_3	X	X					
S_4	X	X	X				
S_5	(2,6) ○	X	X	X			
S_6	X	○	X	X	X		
S_7	X	X	X	(1,8)○	X		
S_8	(2,6) ○	X	X	X	(1,5) ○		

S/I	I_1	I_2	I_3	I_4
S_1	$S_1/0$	$S_2/1$	$S_5/1$	$S_1/0$
S_2	$S_2/1$	$S_2/0$	$S_6/0$	$S_3/0$
S_3	$S_3/1$	$S_7/0$	$S_4/0$	$S_3/0$
S_4	$S_1/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_5	$S_5/0$	$S_6/1$	$S_5/1$	$S_5/0$
S_6	$S_2/1$	$S_6/0$	$S_2/0$	$S_3/0$
S_7	$S_8/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_8	$S_8/0$	$S_6/1$	$S_5/1$	$S_1/0$

	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_2	X						
S_3	X	X					
S_4	X	X	X				
S_5	(2,6) ○	X	X	X			
S_6	X	○	X	X	X		
S_7	X	X	X	(1,8)○	X	X	
S_8	(2,6) ○	X	X	X	(1,5)○	**X**	**(6,7)(4,5)(1,3)**

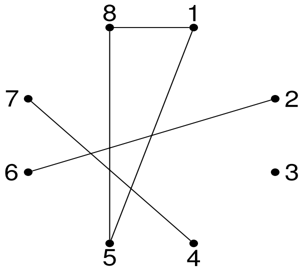
In the dependency table below we can see that some equivalencies depend on others. We can sketch a directed graph based on those dependencies.

	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_2	X						
S_3	X	X					
S_4	X	X	X				
S_5	(2,6) ○	X	X	X			
S_6	X	○	X	X	X		
S_7	X	X	X	(1,8)○	X	X	
S_8	(2,6) ○	X	X	X	(1,5)○	X	X


$$\begin{tikzcd}
4,7 \arrow[d] & & 5,8 \arrow[d] \\
1,8 \arrow[dr] & & 1,5 \arrow[dl] \\
& 2,6 &
\end{tikzcd}$$

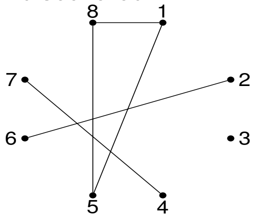
We can sketch an undirected dependency graph using directed dependencies and find the connected components inside the graph. We can combine these cliques and discover equivalent states.

$$\begin{tikzcd}
4,7 \arrow[d] & & 5,8 \arrow[d] \\
1,8 \arrow[dr] & & 1,5 \arrow[dl] \\
& 2,6 &
\end{tikzcd}$$



S/I	I_1	I_2	I_3	I_4
S_1	$S_1/0$	$S_2/1$	$S_5/1$	$S_1/0$
S_2	$S_2/1$	$S_2/0$	$S_6/0$	$S_3/0$
S_3	$S_3/1$	$S_7/0$	$S_4/0$	$S_3/0$
S_4	$S_1/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_5	$S_5/0$	$S_6/1$	$S_5/1$	$S_5/0$
S_6	$S_2/1$	$S_6/0$	$S_2/0$	$S_3/0$
S_7	$S_8/0$	$S_7/1$	$S_4/1$	$S_3/0$
S_8	$S_8/0$	$S_6/1$	$S_5/1$	$S_1/0$

If we rebuild our state table using the equivalence classes we have just discovered



		I_1	I_2	I_3	I_4
S'_1	(S_1, S_5, S_8)	$S'_1/0$	$S'_2/1$	$S'_1/1$	$S'_1/0$
S'_2	(S_2, S_6)	$S'_2/1$	$S'_2/0$	$S'_2/0$	$S'_3/0$
S'_3	(S_3)	$S'_3/1$	$S'_4/0$	$S'_4/0$	$S'_3/0$
S'_4	(S_4, S_7)	$S'_1/0$	$S'_4/1$	$S'_4/1$	$S'_3/0$

Outline

State compatibility

A machine can sometimes be defined *incompletely*, either willingly or by the lack of information. An incomplete FSM transitions under some inputs lead to unspecified states or unspecified outputs. In order to eliminate the redundant states in such machines the concept of *compatibility relation* can be used.

Compatibility Relation

A given binary relation γ on a set S is said to be a compatibility relation if and only if it is reflexive, symmetric and non-transitive.
For example let function $d(x,y)$ denote the distance between points x and y. If we define a relation $R_\gamma = \{(a,b) | d(a,b) \leq 2, a,b \in \mathbb{N}\}$.
For this relation $1\gamma3$ and $3\gamma5$ holds but $1\gamma5$ doesn't. On the other hand transitive pairs can also be found like $1\gamma2$ and $2\gamma3$. γ is a compatibility relation.

Compatibility Class

A compatibility relation forms the compatiblity classes over the set it has been defined. Each compatibility class is transitive inside. Differing from equivalence classes, compatibility classes may not be distinct.

Maximal Compatibility Class

A compatible class is said to be maximal if it is not covered by any other compatible class. Class' graph is not a subgraph of another compatibility class.
For the following three compatibility classes $(a,b,c,d) \supseteq (a,b,c) \supseteq (a,b)$, (a,b,c,d) is the maximal compatibility class.

Cover

A set of compatible classes covers machine $\mathbb{M}$ if it contains all the states of the machine.

Complete Cover

The set of all the maximal compatible classes that covers machine $\mathbb{M}$.

Cover

A set of compatible classes covers machine $\mathbb{M}$ if it contains all the states of the machine.

Complete Cover

The set of all the maximal compatible classes that covers machine $\mathbb{M}$.

Cover

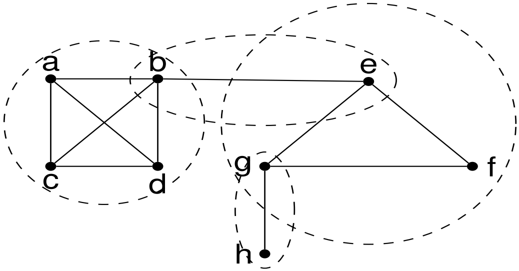
A set of compatible classes covers machine $\mathbb{M}$ if it contains all the states of the machine.

Complete Cover

The set of all the maximal compatible classes that covers machine $\mathbb{M}$.

Example

For the relation γ defined as
$R_\gamma = \{a\gamma b, a\gamma c, a\gamma d, b\gamma c, b\gamma d, c\gamma d, b\gamma e, e\gamma f, e\gamma g, g\gamma f, g\gamma h\}$
Let's sketch the relation graph.



The complete cover of the graph is: $\{\{a,b,c,d\}\{b,e\}\{e,f,g\}\{g,h\}\}$.
For this example one can find a cover that doesn't contain $\{b,e\}$ however the complete cover should contain $\{b,e\}$ too.

Compatibility of States

States s_i and s_j are compatible, if and only if, for every possible input sequence applicable to them, the same output sequence is produced ignoring the undefined states. Compatible states are denoted as $s_i \gamma s_j$.

Let's consider the three states below

$s_1 = ab\emptyset ef$

$s_2 = a\emptyset fef$

$s_3 = acfef$

There exists two compatible classes s_1,s_2 and s_2, s_3 in this example.

State Compatibility Conditions

The necessary and sufficient conditions for two states in a state table to be compatible are: For all the inputs of those states

- Outputs should be the compatible
- Successing conditions should
  - be explicitly or implicitly compatible
  - not have any outputs that doesn't conform compatiblity

State reduction using a complete cover may yield to a non-optimal result. In order to achieve optimal state reduction minimal closed covers should be used.

Implied Compatible

Let's assume R is a set of next states for input i from a compatible set of states S. If S is a compatible, then R is called an implied compatible of S under input i. Implied compatibles can be seen in compatibility dependency graph.

Closed Cover

A set of compatibles is closed if for every compatible contained in the set, all its implied compatibles are also contained in the same set.

State reduction using a complete cover may yield to a non-optimal result. In order to achieve optimal state reduction minimal closed covers should be used.

Implied Compatible

Let's assume R is a set of next states for input i from a compatible set of states S. If S is a compatible, then R is called an implied compatible of S under input i. Implied compatibles can be seen in compatibility dependency graph.

Closed Cover

A set of compatibles is closed if for every compatible contained in the set, all its implied compatibles are also contained in the same set.

State reduction using a complete cover may yield to a non-optimal result. In order to achieve optimal state reduction minimal closed covers should be used.

Implied Compatible

Let's assume R is a set of next states for input i from a compatible set of states S. If S is a compatible, then R is called an implied compatible of S under input i. Implied compatibles can be seen in compatibility dependency graph.

Closed Cover

A set of compatibles is closed if for every compatible contained in the set, all its implied compatibles are also contained in the same set.

State reduction using a complete cover may yield to a non-optimal result. In order to achieve optimal state reduction minimal closed covers should be used.

Minimal Closed Cover

A set of k compatibles forming the set $\mathbb{S}$ is called a minimal closed cover if and only if $\mathbb{S}$ satisfies

1. Covering condition: $\mathbb{S}$ covers the machine $\mathbb{M}$
2. Closure condition: $\mathbb{S}$ is closed
3. Minimal condition: A set of $k-1$ or less compatibles does not satisfy both covering condition and closure condition.

Example

Let's consider the following incomplete state table:

	I_1	I_2	I_3	I_4
S_1	-	-	$S_5/1$	-
S_2	-	$S_5/1$	$S_4/-$	-
S_3	$S_3/0$	$S_2/1$	-	$S_1/0$
S_4	$S_3/0$	$S_1/1$	$S_4/0$	-
S_5	$S_4/0$	$-/1$	$S_3/-$	$S_4/-$

and its corresponding dependency table

	S_1	S_2	S_3	S_4
S_2	$S_4 - S_5$			
S_3	$\circ$	$S_2 - S_5$		
S_4	X	$S_1 - S_5$X	$S_1 - S_2$	
S_5	$S_3 - S_5$X	$S_3 - S_4$	$S_3 - S_4, S_1 - S_4$X	$S_3 - S_4$

Example

Let's consider the following incomplete state table:

	I_1	I_2	I_3	I_4
S_1	-	-	$S_5/1$	-
S_2	-	$S_5/1$	$S_4/-$	-
S_3	$S_3/0$	$S_2/1$	-	$S_1/0$
S_4	$S_3/0$	$S_1/1$	$S_4/0$	-
S_5	$S_4/0$	$-/1$	$S_3/-$	$S_4/-$

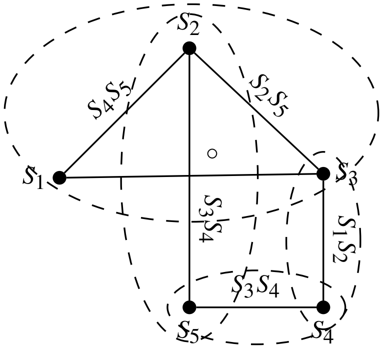
and its corresponding dependency table

	S_1	S_2	S_3	S_4
S_2	$S_4 - S_5$			
S_3	$\circ$	$S_2 - S_5$		
S_4	X	$S_1 - S_5$X	$S_1 - S_2$	
S_5	$S_3 - S_5$X	$S_3 - S_4$	$S_3 - S_4, S_1 - S_4$X	$S_3 - S_4$

Example

Let's build the relation graph of the state compatibilities. We label edges as compatibility dependencies.

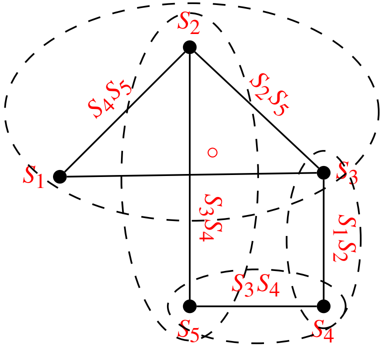
	S_1	S_2	S_3	S_4
S_2	$S_4 - S_5$			
S_3	$\circ$	$S_2 - S_5$		
S_4	X	$S_1 - S_5$X	$S_1 - S_2$	
S_5	$S_3 - S_5$X	$S_3 - S_4$	$S_3 - S_4, S_1 - S_4$X	$S_3 - S_4$



This machine's maximal compatibility class is $\{(S_1, S_2, S_3), (S_3, S_4), (S_2, S_5), (S_4, S_5)\}$

Example

The relation graph points out a four state machine, it may be approporiate to examine dependency graph of the relation as well.



$$\begin{tikzcd}
S_1S_3 & & \\
S_1S_2 \arrow[r] & S_3S_4 \arrow[d] & \\
& S_4S_5 \arrow[ul] & S_2S_5 \arrow[l] \\
& & S_2S_3 \arrow[u]
\end{tikzcd}$$

For the graph above, (S_1, S_2), (S_3, S_4) and (S_4, S_5) forms a compatibility class having closure property and covering all the states of the system.

Example

At the end of the process we can build a new state table for the machine.

	I_1	I_2	I_3	I_4
S_1	-	-	$S_5/1$	-
S_2	-	$S_5/1$	$S_4/-$	-
S_3	$S_3/0$	$S_2/1$	-	$S_1/0$
S_4	$S_3/0$	$S_1/1$	$S_4/0$	-
S_5	$S_4/0$	$-/1$	$S_3/-$	$S_4/-$

A Mealy model reduced machine

		I_1	I_2	I_3	I_4
(S_1, S_2)	a	-	c/1	c/1	-
(S_3, S_4)	b	b/0	a/1	b,c/0	a/0
(S_4, S_5)	c	b/0	a/1	b/0	b,c/-

A Moore model machine

		I_1	I_2	I_3	I_4	O
u	(a/1)	-	z	z	-	1
v	(a/0)	-	z	z	-	0
w	(b/0)	w	u	w	v	0
z	(c/1)	w	u	w	w	1