Week 15: Lecture Notes

Minimization of Finite State Machines

- **Synchronous sequential Machine**

$M = \langle Q, \Sigma, \Delta, \delta, \lambda \rangle$

Q → states, Σ → input alphabet, Δ → output alphabet

δ ⇒ state transition function

λ → Output function

$\delta: Q \times \Sigma \to Q$

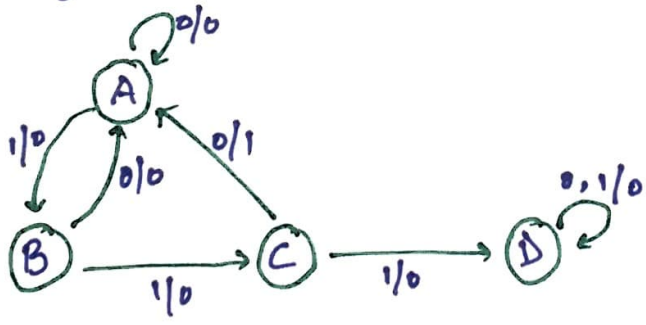
$\lambda: Q \times \Sigma \to \Delta$ for Mealy machines

$\lambda: Q \to \Delta$ for Moore machines

- **X- successor**

S_j is x-successor of S_i, $S_i, S_j \in Q$, $x \in \Sigma^*$ if

input sequence x takes the machine from state S_i to state S_j



- D is terminal state (sink - no outgoing edge from it to other vertex)

111 - successor of A is D

10 - successor of B is A

10 - successor of C is D

- State transition graphs may contain redundant states
  i.e. states whose function can be accomplished by other states

- State minimization is the transformation of a given machine into an equivalent machine with no redundant states

- Distinguishable states
- Distinguishable sequences
- k-distinguishable states

Example 1

- A, B → 1-distinguishable (dist. seq • X = 1)
  A → 1, B → 0
- A, E → 3-distinguishable (dist. seq X = 111)
  A → 100, E → 101

- Equivalent states
- Equivalence classes

Strongly Connected Machine

M is strongly connected if for every pair of states S_i, S_j there exists an input sequence that takes M from S_i to S_j

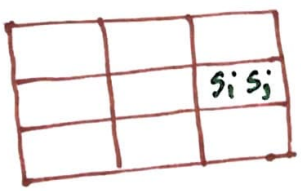
- A machine that has a terminal state is NOT strongly connected.
- Sink vertex: No outgoing edges that emanate from it terminate in other vertices
- Source Vertex: No edges that emanate from other vertices terminate in it.

State equivalence and machine minimization

- Two states S_i, S_j of a machine M are distinguishable iff $\exists$ atleast one finite input sequence that, when applied to M, causes different output sequences, depending on whether S_i or S_j is in the initial state.
- S_i, S_j are said to be K-distinguishable if $\exists$ a distinguishing input sequence of length K for the pair S_i, S_j (for which the output sequence is different).

- K-equivalent: states that are NOT K-distinguishable are said to be K-equivalent
  - S_i, S_j K-equivalent $\Rightarrow$ S_i, S_j r-equivalent for $r<K$
  - S_i, S_j K-equivalent $\forall K \Rightarrow$ S_i, S_j are equivalent

- Equivalent: The states S_i, S_j of machine are said to be equivalent iff for every possible input sequence, the same output sequence is produced regardless of whether S_i or S_j is the initial state.



$S_i, S_j \in Q$ in the same class iff S_i, S_j are equivalent

$S_i, S_j \in Q$ in different class iff S_i, S_j are distinguishable

- S_i, S_j equivalent states
  $\Rightarrow$ their corresponding X-successor, $\forall X$ are also equivalent

- **Property**

  $S_i \sim S_j \Rightarrow$ their corresponding X-successor for all input X are also equivalent

- **Procedure**

  Group states of M so that two states are in the same group iff they are equivalent (forms a partition of the states)

- $P_K \rightarrow$ partition using distinguishing sequence of length K
- Construct P_{K+1} from P_K
- Terminate when $P_K = P_{K+1}$

Example: (Moore reduction procedure)

PS	NS, z	
	$x=0$	$x=1$
A	E, 0	D, 1
B	F, 0	D, 0
C	E, 0	B, 1
D	F, 0	B, 0
E	C, 0	F, 1
F	B, 0	C, 0

PS - present state
NS - next state
z - output symbol
x - input symbol

- A, B - 1-distinguishable (X=1, is a distinguishing sequence)
- A, E - 3-distinguishable (X=111 is a distinguishing sequence)
- A, E - is 2-equivalent and so also 1-equivalent.

Partitioning (P_K to P_{K+1} construction)

$P_0 = (ABCDEF)$

$P_1 = (ACE), (BDF)$

$P_2 = (ACE), (BD), (F)$

$P_3 = (AC), (E), (B,D), (F)$

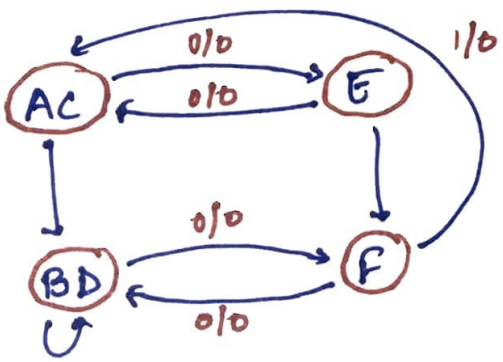
$P_4 =$

- P_{k+1} is obtained from P_k by

  placing in the same block of P_{k+1} those states that are in the same block of P_k, and whose I_i-successor for every possible I_i are also in the common block of P_k

Minimal reduced machine M^*

PS	NS, Z	
	x = 0	x = 1
AC	E, 0	BD, 1
E	AC, 0	f, 1
BD	f, 0	BD, 0
f	BD, 0	AC, 0



Example:

Machine

PS	NS, Z	
	x = 0	x = 1
A	E, 0	C, 0
B	C, 0	A, 0
C	B, 0	G, 0
D	G, 0	A, 0
E	F, 1	B, 0
F	E, 0	D, 0
G	D, 0	G, 0

Dist Sequence (X)

X = 0

X = 00

X = 100

X = 1100

State Partition

P_0 : (ABCDEFG)

P_1 : (ABCDEG) (F)

P_2 : (AF) (BCDG) (E)

P_3 : (AF) (BD) (CG) (E)

P_4 : (A) (F) (BD) (CG) (E)

P_5 : (A) (F) (BD) (CG) (E)

Minimal Machine

PS	NS, Z	
	x = 0	x = 1
A	E, 0	CG, 0
F	E, 0	BD, 0
BD	CG, 0	A, 0
CG	BD, 0	CG, 0
E	f, 1	BD, 0

- To every machine M there corresponds a minimal machine M^* that is equivalent to M and is unique upto isomorphism.

- Two machines M_1, M_2 are said to be equivalent iff for every state in M_1 there is a corresponding equivalent state in M_2 and vice-versa

Theorem

The equivalence partition is unique

Theorem

If two states S_i and S_j of a machine M are distinguishable then they are $(n-1)$ distinguishable, where n is the number of states in M

Definition

Two machines M_1, M_2 are equivalent ($M_1 \sim M_2$) iff for every state in M_1, there is a corresponding equivalent state in M_2 and vice-versa

Theorem

For every machine M, there is a minimum machine

$M_{red} \sim M$

- M_{red} is unique upto isomorphism

- $M_{red} \rightarrow$ Reduced quotient machine

Specification of incompletely specified machines

Example:

PS	NS, Z	
	$x=0$	$x=1$
A	B, 1	—
B	—, 0	C, 0
C	A, 1	B, 0

Equivalent description where all state transitions are specified

only output symbols are partially defined

PS	NS, Z	
	$x=0$	$x=1$
A	B, 1	T, -
B	T, 0	C, 0
C	A, 1	B, 0
T	T, -	T, -

Cover

A state S_i of machine M_1 is said to cover or contain state S_j of machine M_2 iff every input sequence applicable to S_j is also applicable to S_i and it's application to both M_1, M_2 when they are initially in S_i, S_j respectively results in identical output sequence whenever the output symbols of M_2 are specified.

- Machine M_1 covers machine M_2 iff $\overset{\text{for}}{\wedge}$ every state S_j in M_2 there is a corresponding state S_i in M_1 such that S_i covers S_j.

Compatible States

Two states S_i, S_j of a machine M are compatible iff for every input sequence to both S_i, S_j, the same output sequence will be produced whenever both input symbols are specified and regardless of whether S_i or S_j is the initial state.

i.e. S_i, S_j is compatible iff their output sequences are not conflicting (i.e. identical when specified) and their I_k-successor for every I_k for which both are specified, are either same or also compatible.

$S_i \xrightarrow{I_k} S_p$
$S_j \xrightarrow{I_k} S_q$

} same output sequence and either $S_p = S_q$ or S_p, S_q are compatible.

- $S_i, S_j, \ldots S_k \ldots$ are compatible iff for every applicable input sequence no two conflicting output sequences will be produced
- Maximal compatible not covered by any other compatible

Non uniqueness of the minimal machine in case of incompletely specified machines

Example:

M_1

PS	NS, Z	
	$x=0$	$x=1$
A	C, 1	E, –
B	C, –	E, 1
C	B, 0	A, 1
D	D, 0	E, 1
E	D, 1	A, 0

M_2. • Replace both '–' by '1'

AB becomes equivalent

PS	NS, Z	
	$x=0$	$x=1$
A	C, 1	E, 1
C	B, D	A, 1
D	D, 0	E, 1
E	D, 1	A, 0

- Replace both –'s by 0's

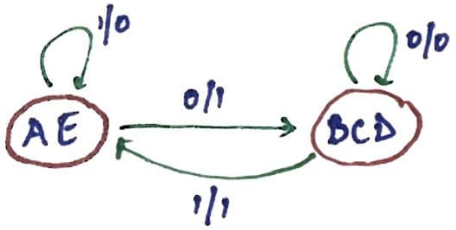
M_3

P_0: (ABCDE)

P_1: (AE), (BCD)

P_2: (AE), (BCD)

PS	NS, Z	
	$x=0$	$x=1$
AE	BCD, 1	AE, 0
BCD	BCD, 0	AE, 1



- Compatibility relation is not an equivalent relation because transitivity doesnot hold

for example in M_1

- A, B is compatible
- A, E is compatible if C, D is compatible
- But B, E I- distinguishable ⇒ not transitive

- A set of states is compatible iff every pair of states in that set is compatible.

  eg.- states B, C, D form the compatible (BCD) as (BC), (BD), and (CD) are compatibles

- We get two reduced machines M_2, M_3 covering M, and their number of states are each smaller than M,

Goal: To find a reduced machine that not only covers the original machine, but also has a minimum number of states.

Example (State spliting)

	NS, Z	
PS	x=0	x=1
A	A, 0	C, 0
B	B, 0	B, -
C	B, 0	A, 1

- A, B equivalent iff B, C are equivalen
- A, B to be equivant, - must be replaced by 0
- B, C to be equivalent, - must be replaced by 1

Conclusion: M is in reduced form (cannot be minimized further)

↓

false conclusion, WHY?

- Given an incompletely specified machine, attempt to reduce it to state minimization of completely specified machine
  - force the don't care conditions to all their possible values and choose the smallest of the completely specified machine so obtained
  - Can this be always done?

Augmented Machine (splitting state B by two states B', B'')

PS	NS, Z $x=0$	$x=1$
A	A, 0	C, 0
B'	B', 0	B'', –
B''	B^+, 0	B', –
C	B^+, 0	A, 1

setting $B^+ = B'$

PS	NS, Z $x=0$	$x=1$
A	A, 0	C, 0
B'	B', 0	B'', – ← 0
B''	B', 0	B', – ← 1
C	B', 0	A, 1

setting $B^+ = B''$

PS	NS, Z $x=0$	$x=1$
A	A, 0	C, 0
B'	B', 0	B'', – ← 0
B''	B'', 0	B', – ← 1
C	B'', 0	A, 1

Partition

P_0: $(AB'\ B''C)$

P_1: $(AB')\ (B''C)$

PS	NS, Z $x=0$	$x=1$
AB'	AB', 0	$B''C$, 0
$B''C$	AB', 0	AB', 1

Partition

P_0: $(AB'\ B''C)$

P_1: $(AB')\ (B''C)$

PS	NS, Z $x=0$	$x=1$
AB'	AB', 0	$B''C$, 0
$B''C$	$B''C$, 0	AB', 1

- Equivalence partition consists of disjoint blocks
- The subsets of compatibles may be overlapping

Note: We want to "merge two states" when for any input sequence they generate the same output sequence, but only where both outputs are specified.

Definition: A set of states is compatible iff they agree of outputs where they are specified.

PS	NS, Z x=0	x=1
A	A, 0	C, 0
B	B, 0	B, -
C	B, 0	A, 1

yields two compatible sets

$S_1 = (A, B)$

$S_2 = (B, C)$

PS	NS, Z x=0	x=1
S_1	S_1, 0	S_2, 0
S_2	S_2, 0 / S_1, 0	S_1, 1

Brute force method:

- specify don't care entries
- transform the incompletely specified machine into completely specified one
- such specification may not be optimal
- drastically reduces our freedom to simplify a machine.

- Instead, first generate all compatible pairs using Merge graph
- A set of states is compatible iff every pair of states in that set is compatible

$S_i \xrightarrow{I_k\text{-successor}} S_p$ $\quad$ $S_j \xrightarrow{I_k\text{-successor}} S_q$

- (S_p, S_q) implied by (S_i, S_j)
- if (S_i, S_j) compatible pair then (S_p, S_q) is referred as implied pair
- A set of states P is implied by a set of states Q if for some input seq I_k, P is the set of all I_k-successors of the states in Q

Merge Graph of an n-state machine M (undirected graph)

1. Consists of n vertices, each of which corresponds to a state of M
2. for each pair of states (S_i, S_j) in M, whose next state and the output entries are not conflicting, an undirected edge is drawn between the vertices S_i and S_j
3. If, for a pair of states (S_i, S_j), the corresponding output symbols under all input symbols are not conflicting, but the successor are not the same, an interrupted edges is drawn between S_i, S_j and the implied pairs are entered in the space.

- In the merger graphs, look for complete polygons that are not contained within any higher order complete polygon
  - complete polygon has all possible $\frac{n(n-3)}{2}$ diagonals, where n is the number of sides in the polygon

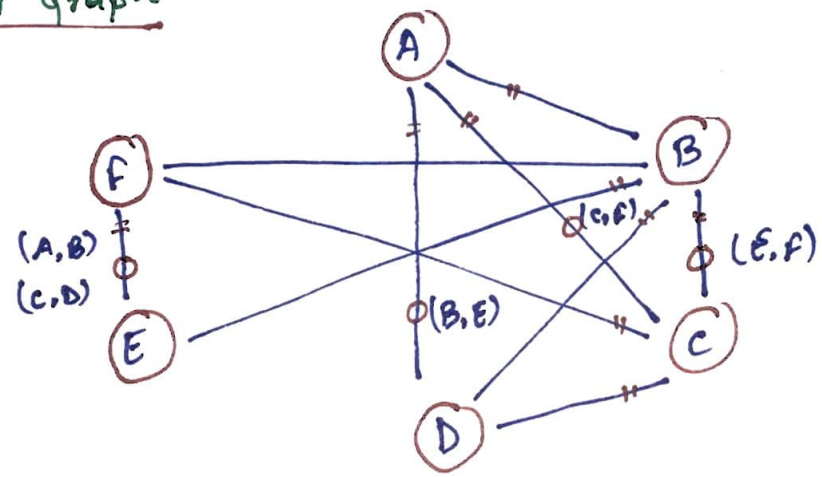
- States covered ~~in~~ by a complete polygon are all pairwise compatible
  ⇒ Complete polygons constitute a compatible

- If a polygon is not contained in any higher order complete polygon, then the states constitute a maximal compatible.

Example

	NS, Z			
PS	I_1	I_2	I_3	I_4
A	–	C, 1	E, 1	B, 1
B	E, 0	–	–	–
C	f, 0	f, 1	–	–
D	–	–	B, 1	–
E	–	f, 0	A, 0	D, 1
f	C, 0	–	B, 0	C, 1

Merger Graph



- nine compatible pairs: (AB), (AC), (AD) (BC), (BD), (BE), (CD), (C,F), (EF)
- (AB) (AC) (BC) compatible ⇒ (ABC) compatible
- The entire set of compatibles can be generated in this way

To find a minimal set of compatibles covering the original machine (which can be used as a basis of ~~consulting~~ constructing a minimal machine) it is often useful to find the set of maximal compatibles.

- Consider the set of compatibles $S = \{(ABCD), (EF)\}$
  - → minimal number of compatibles covering all the states of the above machine M
  - → provides a lower bound on the number of states in the minimal machine that covers M
- If maximal compatible (ABCD) is chosen, then (CF), (BE) must also be selected which are not in S
  - → S cannot be used to define the states of a minimal machine that covers M
  - → lower bound cannot be achieved in this case

Closed set of compatibles

A set of compatibles for a machine M is said to be closed if for every compatible contained in the set, all implied compatible are also contained in the set

Example (Merge graph of the above example)

$\{(AD), (BE), (CD)\}$ is closed (but not closed covering)

$\{(AB), (CD), (EF)\}$ is closed covering

- A closed set of compatibles that contains all the states of M is called a closed covering

Upper bounds on the number of states in the machine covering the original machine.

The set containing all the maximal compatibles
→ a closed covering as it covers all the states of the machine and every implied compatible is contained in the set
→ gives an upper bound

Example: Upper bound for the given machine M
→ 4, as the set of maximal compatibles for M is {(ABCD), (BE), (CF), (EF)}
→ gives a 4-state machine covering M

Trial and error for obtaining minimal closed covering

- For incompletely specified machines, the closed covering serves the same function as that served by the equivalence partition for completely specified machines.

Goal: Select a closed covering which has a minimum number of compatibles.
→ defines a minimal-state machine that covers the original machine.

Example (See Merger Graph)

1. A, B → can be covered by the compatible pair (AB)
   C, D → by (CD)
   The pair (Ef) is compatible and implies the (AB), (C, D)
   ⇒ {(AB), (CD), (Ef)} is a closed covering
   → yields a minimal three state machine covering the original machine.

	NS, Z			
PS	I_1	I_2	I_3	I_4
AB	Ef, 0	CD, 1	Ef, 1	AB, 1
CD	Ef, 0	Ef, 1	AB, 1	—
Ef	CD, 0	Ef, 0	AB, 0	CD, 1

3-state machine covering M

2. {(AD), (BE), (CF)} → another closed covering that corresponds to a minimal machine containing the given machine

A more systematic procedure for obtaining minimal closed covering

Compatibility graph: A digraph whose vertices corresponds to all compatible pairs and for which an edge leads from vertex $(S_i S_j)$ to vertex $(S_p S_q)$ iff $(S_i S_j)$ implies $(S_p S_q)$

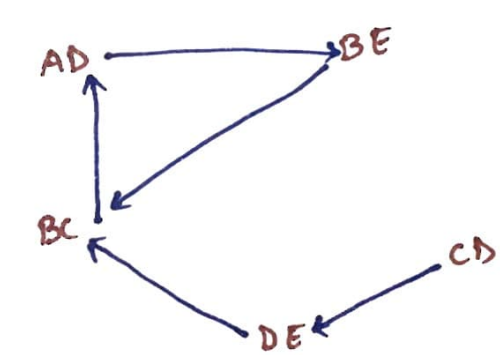
Example:

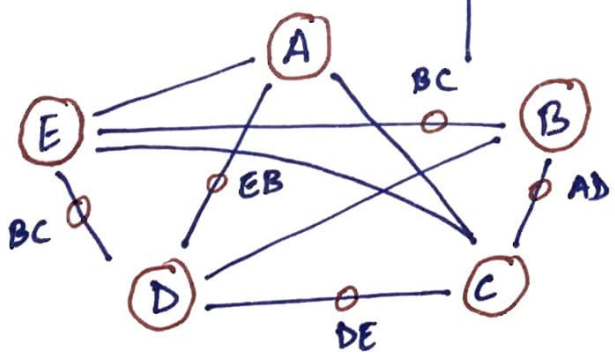
	NS, Z			
PS	I_1	I_2	I_3	I_4
A	-	-	E, 1	-
B	C, 0	A, 1	B, 0	-
C	C, 0	D, 1	-	A, 0
D	-	E, 1	B, -	-
E	B, 0	-	C, -	B, 0

Merger Graph

	NS, Z			
PS	I_1	I_2	I_3	I_4
AD	-	BE, 1	BE	-
BC	BC, 0	AD, 1	BC, BE	AD, 0
BE	BC, 0	AD, 1	BC, 0	BC, BE, 0

compatibility graph.





compatible pairs: (AD), (ED), (AC), (BE), (BC), (CD)

- A subgraph of a compatibility graph is said to be closed if for every vertex in the subgraph, all outgoing edges and their terminating vertices also belong to the graph
- If every state of the machine is covered by atleast one vertex of the subgraph, then the subgraphs forms a closed covering for the machine.

Example: {(BC), (AD), (BE)}, {(AC)}

{(AC), (BC), (AD), (BE)}, the graph itself.

{(DE), (BC), (AD), (BE)}

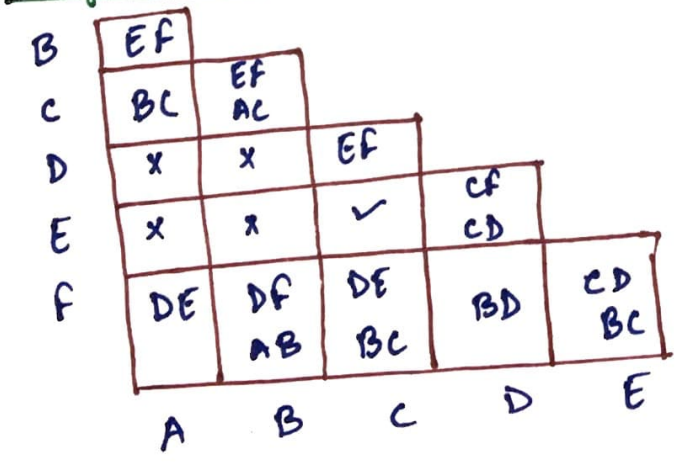
Merger Table (when dealing with machines with a large number of states)

Example:

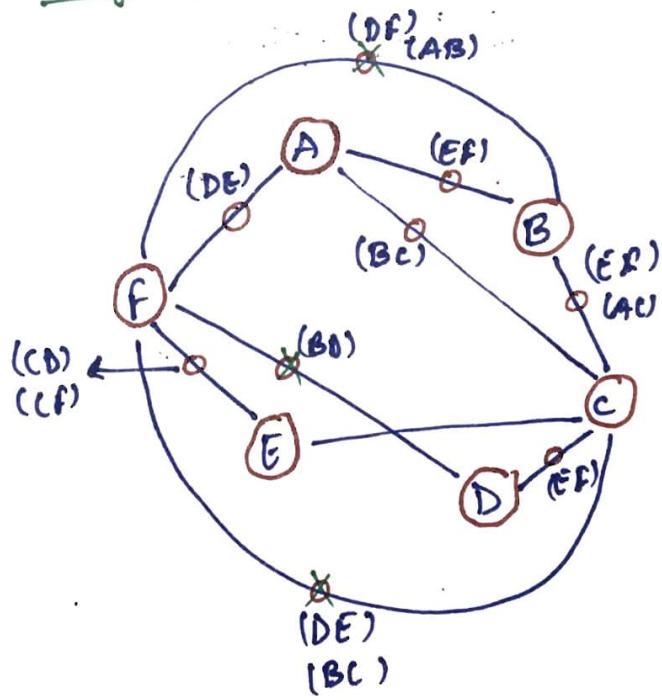
M

	NS, Z	
PS	I_1	I_2
A	E, 0	B, 0
B	f, 0	A, 0
C	E, -	C, 0
D	f, 1	D, 0
E	C, 1	C, 0
F	D, -	B, 0

Merge Table



Merge Graph.



Finding the set of maximal compatibles

Column E : (EF)

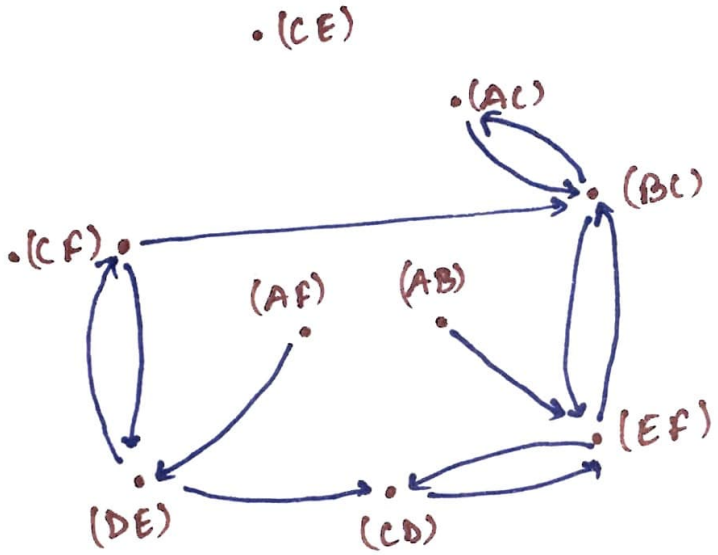
Column D : (DE), (EF)

Column C : (CDE), (CEF)

Column B : (CDE), (CEF), (BC)

Column A : (CDE), (CEF), (ABC), (AFC)

Compatibility Graph



- Set of maximal compatibles
  - → {(CDE), (CEF), (ABF), (AEF)}
  - → upper bound for states in a machine covering M is 4
  - → cannot be covered by a 2-state machine