UNIT –V
STATES, STATE GRAPHS, AND TRANSITION TESTING

(1) State Graphs:

(i) States(public question)

- State is a condition or situation during which an object undergoes throughout its life time.
- States are represented by nodes.
- States are numbered or identified by characters or words or whatever else is convenient.
- A state graph consists of a set of states in order to represent the behavior of the system.
- To understand the concept of states let us consider the following examples.

Example 1: A program that detects the character sequence ZCZC can be in the following states.

1. Neither ZCZC nor any part of it has been detected.
2. Z has been detected.
3. ZC has been detected.
4. ZCZ has been detected.
5. ZCZC has been detected.

Example 2: A moving automobile whose engine is running can have the following states with respect to transmission.

1. Reverse gear.
2. Neutral gear.
3. First gear.
4. Second gear.
5. Third gear.
6. Four gear.

Example 3: A person's checkbook can have the following states with respect to bank balance.

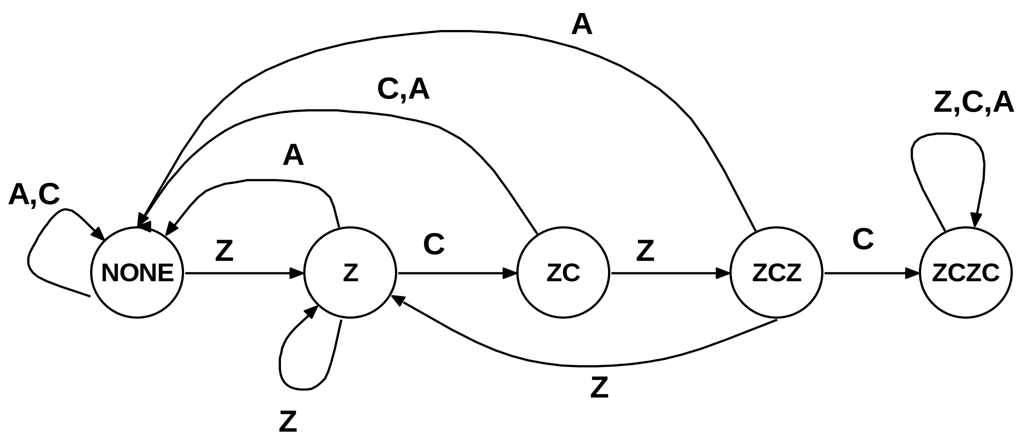
1. Equal.
2. Less than.
3. Greater than.

Example 4: A word processing program menu can be in the following states with respect to file transmission.

1. Create document.
2. Copy document.
3. Delete document.
4. Rename document.
5. Compress document.
6. Saving document
7. Copy disc.
8. Format disc
9. Backup disc
10. Recover from backup

(ii) Inputs and Transitions:(public question)

- Some thing is modeled and given is called input. Input may be values or variables.
- A state graph takes input provided to states.
- As a result of these inputs the state changes is known as transition.
- That is changing from one state to other state is called transition.
- Transitions are denoted by links that join the states.
- The input that causes the transition is represented on the link. So the inputs are link weights.
- A finite state machine is represented by a state graph having a finite number of states and a finite number of transitions between states.
- The ZCZC detection example can have the following types of inputs.
    1. Z
    2. C
    3. Any character other than Z or C which will be denoted by A.



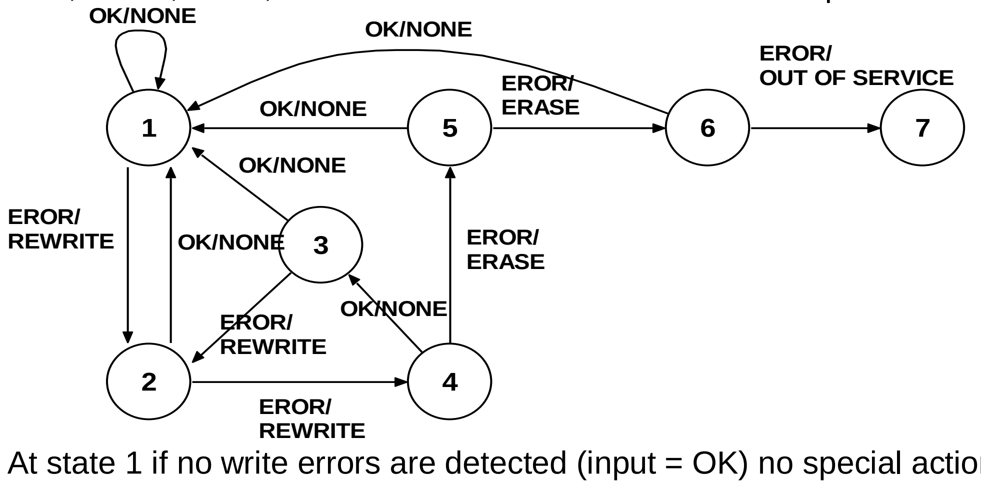
<u>The above state graph is interpreted as follows</u>.

1. If a system is in the NONE state, and it receives A or C then it is in NONE state only.
2. In NONE state if Z is received, the system enters into Z state. In Z state if it receives Z it will remain in the same state. If C is received it will go to the ZC state or if any other character say A is received then it will go back to the NONE state.
3. In ZC state if it receives Z it will enter into ZCZ state. If C or A is received it enter into NONE state.
4. In ZCZ state if it receives Z it enter into the Z state. If A is received it enters into the NONE state.
5. In ZCZ state if it receives C it enter into the ZCZC state. In ZCZC state if it receives Z or C or A then it will remain in the same state only.

<u>(iii) Outputs:</u>

- Outputs are based on the input values.
- When an input is applied to a state it is processed in order to produce an output.
- Each input and output of the state graph is separated by a slash '/' symbol.
- Outputs are also link weights. If more than one input having the same output than it can be represented by input1, input 2, input 3…/output.

<u>Example</u>: Let us consider a tape control recovery system. This system contains two inputs OK & Error. OK means "No write errors". Error means "There may be write errors". The outputs are Rewrite, Erase, None, Out of service. Here None means no special action is taken.



- At state 1 if no write errors are detected (input = OK) no special action is taken (output=NONE). If error is detected (input=ERROR) backspace the tape one block and rewrite the block (output =REWRITE) i.e. enter into state 2.

- At state 2 if the rewrite is successful (input= OK) no action is taken (output=NONE) and return to state 1.
- If the rewrite is not successful try another back space and rewrite (output=REWRITE) i.e. enter into state 4.
- If there are two successive rewrites and a third error occurs then backspace ten centimeters and erase (output=ERASE) i.e. from state 4 to state 5.
- If there are two successive rewrites and a third no error occurs then it enter into state 3 & then state 1. At state 3 if any error is detected then it enter into state 2 and rewrite.
- At state 5 if the erasure works (input=OK) no action is taken and return to initial state.
- If it does not work, backspace another ten centimeters and erase. i.e. enter into state 6.
- At state 6 if the erasure works (input=OK) no action is taken and return to initial state
- If the second erasure does not work put the tape control out of service i.e enter into state 7

(iv) State Table:

- If state graph has a large number of states and transitions, then it is difficult to follow them.
- Therefore a state table is used, as an easiest way to represent all the states, inputs, transitions and outputs of the state graph.
- A state table is defined as a tabular representation of a state graph.
- It consists of
  1. Each row represents a state.
  2. Each column represents an input condition.
  3. The box at the intersection of row and column represents the next state and the output.
- The state table for the tape control system is shown below.

STATE	OK	ERROR
1	1/NONE	2/REWRITE
2	1/NONE	4/REWRITE
3	1/NONE	2/REWRITE
4	3/NONE	5/ERASE
5	1/NONE	6/ERASE
6	1/NONE	7/OUT
7	....	....

. **(v) Time Versus Sequence:**

- State graphs don't represent time-they represent sequence.
- A transition might take microseconds or centuries.
- A system may be in one state for milliseconds or years.
- The finite state machine model can be elaborated to include notions of time in addition to sequence, such as Petri nets.

(vi) Software Implementation(public question)

1. Implementation and Operation:

- Here four tables are involved.

1. First table encode the input value. i.e. INPUT_TABLE_CODE.
2. A table that specifies the next state i.e. TRANSITION_TABLE
3. A table that specifies the output. i.e. OUTPUT_TABLE
4. A table that stores the present state of every device. i.e. DEVICE_TABLE.

This routine operates as follows.

```
        BEGIN
        PRESENT_STATE:=DEVICE_TABLE
        ACCEPT INPUT_VALUE
        INPUT_CODE:=INPUT_CODE_TABLE
```

```
POINTER:=INPUT_CODE#PRESENT_STATE
NEW_STATE:=TRANSITION_TABLE
OUTPUT_CODE:=OUTPUT_TABLE
CALL OUTPUT_HANDLER
DEVICE_TABLE:=NEW_STATE
END
```

Steps:

1. The present state is fetched from memory.
2. The present input value is fetched. If it is numerical it can be used directly. If it is not numerical encode into a numerical value.
3. The present state and input code are combined.
4. The output table contains a pointer to the routine to be executed.
5. The same pointer is used to fetch the new state value, which is then stored.

2. Input encoding and Input Alphabet:

- Only the simplest finite state machines can use the inputs directly.
- In ZCZC detector there are 256 possible ASCII characters. But we are taken Z, C and OTHER.
- The input encoding here is for OTHER=0, for Z=1, for C=2.
- The different encoded input values are called the input alphabet.

3. Output encoding and Output Alphabet:

- A single character output for a link is rare.
- So we want to output a string of characters.
- These can be encode into a convenient output alphabet.

4. State codes and State-Symbol products:

- The term state-symbol product is used to convert the combined state and input code into a pointer to compact table.

5. Application Comments for Designers:

- An explicit state table implementation is advantageous when either the control function is likely to change in the future or when the system has many similar, but slightly different control functions.

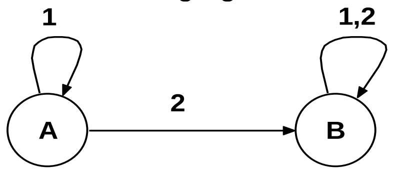
6. Application Comments for Testers(Public Question)

- Independent testers are not usually taken with either implementation details or the economics of this approach.
- If the programmers have implemented an explicit finite state machine then much of our work has been done for us.
- Sometimes showing the programmers the kinds of tests developed from a state graph description can lead them to consider it as an implementation technique.

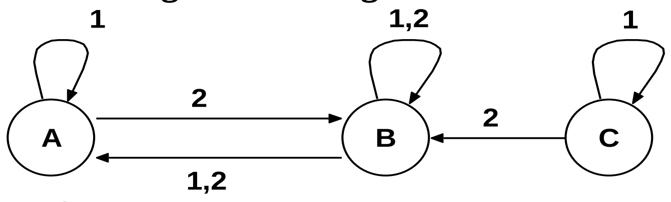
(2) Good State Graphs and Bad State Graphs: (public question)

(i) General:

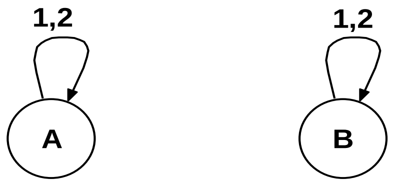
- In testing we deal with a good state graph and also with a bad one.
- The following figure shows examples of improper or bad state graphs.



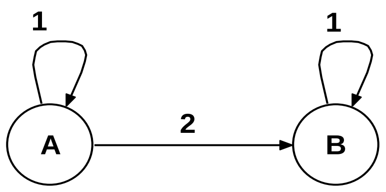
In state B the initial state can never be entered again



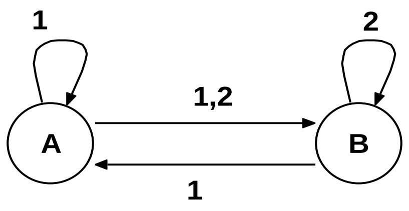
State C cannot be entered



State A,B are not reachable



No transition is specified for an input of 2 when in state B



Two transitions are specified for an input of 1 in state A

(2) State Bugs(public question)

- The bugs in states are called state bugs. The state bugs arise due to the following reasons.

1. Number of States:

- ❖ A State graph consists of the number of states. It represents behavior of the system.
- ❖ In practice the state is directly or indirectly recorded.
- ❖ State table is used to record the number of states of the state graph.
- ❖ In state table the state bugs are occurred because of missing states.
- ❖ That is in state table if the number of states are not recorded or missed then the result might be the bugs.
- ❖ To find the missing states, first find the number of states
- ❖ The number of states is founded by as follows.
  1. Identify all the component factors of the state.
  2. Identify all the allowable values for each factor.
  3. Now the number of states is the product of the factors and allowable values.
- ❖ Functional specifications are used to find the factors of the state. They may also helpful to find the number of possible values for each factor.

2. Impossible States:

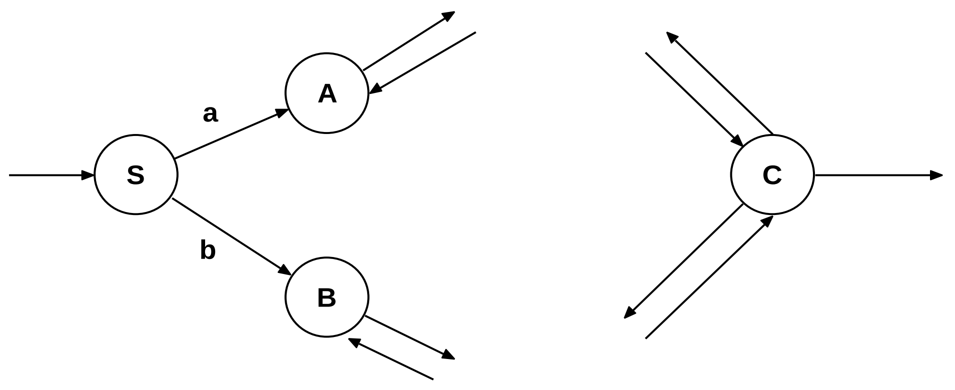
- ❖ A state that is not possible is called impossible states.
- ❖ For example a broken engine cannot run, so running a broken engine state is impossible state.
- ❖ There are some combination of factors that are impossible, they are
  GEAR: R, N, 1, 2, 3, 4 = 6 factors
  DIRECTION: forward, reverse, stopped = 3 factors
  ENGINE: running, stopped = 2 factors
  TRANMISSION: ok, broken = 2 factors
  ENGINE: ok, broken = 2 factors

TOTAL = 6 x 3 x 2 x 2 x2 =144 states.

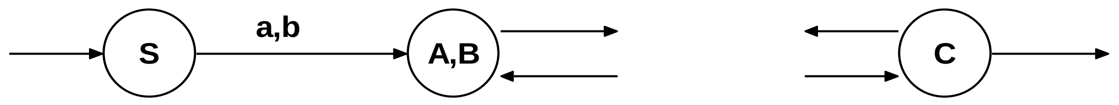
- ❖ A broken engine cannot run so the combination of engine is 3 states. Therefore the total number of states is 108. A car with a broken transmission does not move for long, there by further decreasing the number of states.

3. Equivalent States:

- ❖ Two states A, B are equivalent if every sequence of inputs starting from one state (s) produces exactly the same sequence of outputs.
- ❖ Let us take an example of two equivalent states.
- ❖ In the below figure, let us assume the system is in state S.
- ❖ An input of 'a' begins a transition to state A and an input of 'b' begins a transition to state B from S.
- ❖ If all the sequence of inputs from the state A generates exactly the same sequence of outputs as the other state B, then we say that these two states are equivalent.

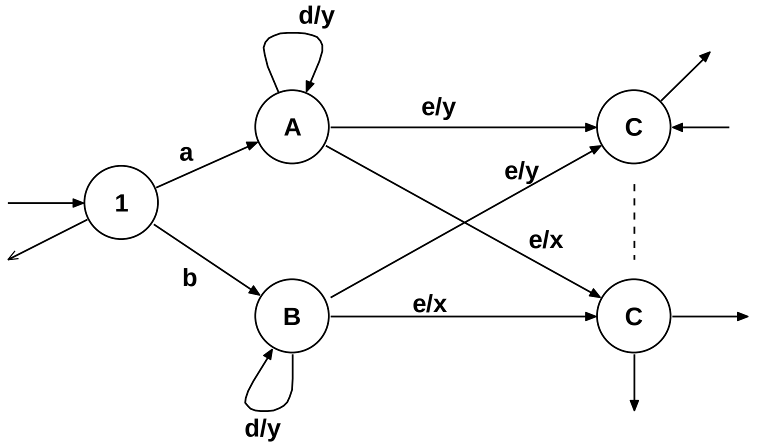


- ❖ Because these two states are treated equally, the state graph can be minimized by combining these two equivalent states as shown in the following figure.



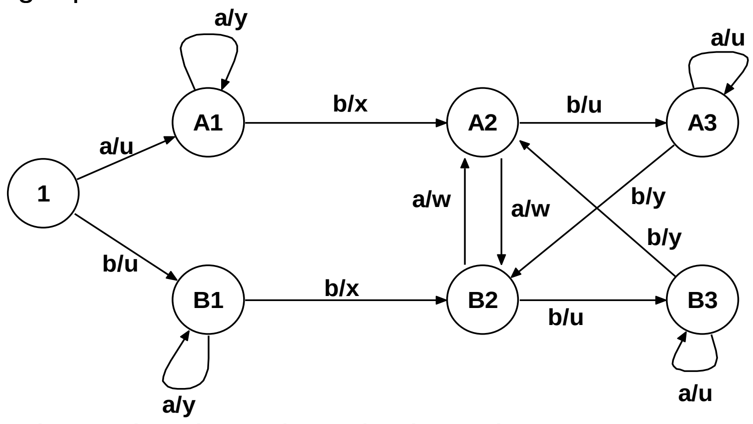
- ❖ Equivalent states can be recognized by the following procedure.

1. The two states are differentiated only by the different input values. For example Consider the following figure.



Here except a, b inputs, the system behavior in two states A, B are identical for every input sequence.

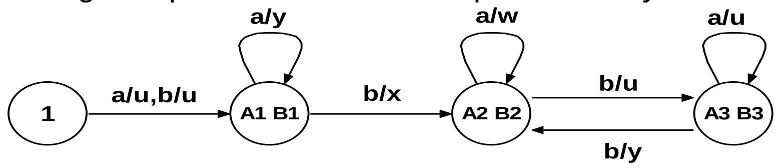
2. There are two set of rows which except for the state name, have identical state graphs with respect to transitions and outputs. The two sets can be merged. Let consider the following equivalent states.



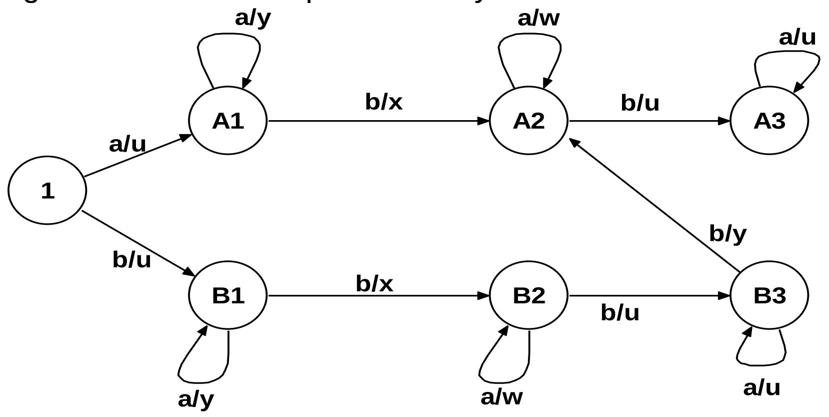
The Decision table to the above figure is shown below:

STATE	OK	ERROR
1	A1/u	B1/u
A1	A1/y	A2/x
B1	B1/y	B2/x
A2	B2/w	A3/u
B2	A2/w	B3/u
A3	A3/u	B2/y
B3	B3/u	A2/y

The merged equivalent states are represented by as follows



The Unmergeable states are represented by as follows



(3) Transition Bugs(public question)

- The connectivity between two or more states is known as transition.
- The bug in transition is called Transition Bug.

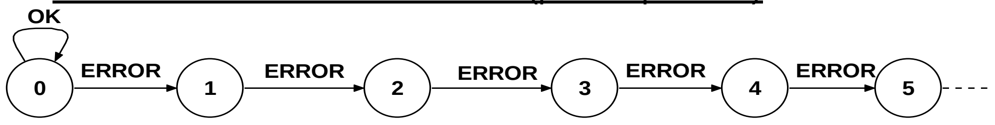
1. Unspecified and Contradictory Transitions:

- ❖ A transition is specified between states. If a transition may occur between states and not specified (i.e. unspecified transition) then the transition bug occurs.
- ❖ If a transition is not possible in the state then there must be a method that prevents the occurrence of input in that state.

- ❖ If there is no such method available then the occurrence of input becomes inefficient.
- ❖ So to avoid transition bug one transition must be specified for every input state combination.
- ❖ A program does not contain contradictions, if one input must be processed at a time to produce desired output. If a transition does not possible between states and the transition is specified then a contradictory transition may occur.
- ❖ That is if a programmer does not take all the measures of a program then contradictory transitions may occur because of transitions may not be possible between some of the states. For example if a single bit of a state is misplaced by the programmer then it doubles the number of states in the state graph and performs the contradictory transitions. This contradiction gives a transition bug.

2. Example(public question)

- ❖ The following example shows how to convert a specification into a state graph and how contradictions can come out.(public question)



Rule 1:

- ❖ The program will maintain an error counter which will be incremented whenever there is an error. Here there are only two input values OK, ERROR.
- ❖ These values make it easier to detect ambiguities and contradictions in a state table.

INPUT

STATE	OK	ERROR
0	0/NONE	1/
1		2/
2		3/
3		4/
4		5/
5		6/
6		7/
7		8/

Rule 2: If there is an error rewrite the block.

INPUT

STATE	OK	ERROR
0	0/NONE	1/REWRITE
1		2/REWRITE
2		3/REWRITE
3		4/REWRITE
4		5/REWRITE
5		6/REWRITE
6		7/REWRITE
7		8/REWRITE

Rule 3: If there are three errors, erase 10 centimeters of tape and rewrite the block.

INPUT

STATE	OK	ERROR
0	0/NONE	1/REWRITE
1		2/REWRITE
2		3/REWRITE,ERASE,REWRITE

3		4/REWRITE,ERASE,REWRITE
4		5/REWRITE,ERASE,REWRITE
5		6/REWRITE,ERASE,REWRITE
6		7/REWRITE,ERASE,REWRITE
7		8/REWRITE,ERASE,REWRITE

Rule 4: If there are three erasures and another error occur, then put out of service.

INPUT

STATE	OK	ERROR
0	0/NONE	1/REWRITE
1		2/REWRITE
2		3/ERASE,REWRITE
3		4/ERASE,REWRITE
4		5/ERASE,REWRITE
5		6/OUT
6		
7		

Rule 5:

❖ If the erasure was successful return to the normal state and clear the error counter.

INPUT

STATE	OK	ERROR
0	0/NONE	1/REWRITE
1		2/REWRITE
2		3/ERASE,REWRITE
3	0/NONE	4/ERASE,REWRITE
4	0/NONE	5/ERASE,REWRITE
5	0/NONE	6/OUT
6		

Rule 6:

❖ If the rewrite was unsuccessful increment the error counter, and try another rewrite.

Rule 7:

❖ If the rewrite was successful decrement the error counter and return to the previous state.

INPUT

STATE	OK	ERROR
0	0/NONE	1/REWRITE
1	0/NONE	2/REWRITE
2	1/NONE	3/ERASE,REWRITE
3	0/NONE 2/NONE	4/ERASE,REWRITE
4	0/NONE 3/NONE	5/ERASE,REWRITE
5	0/NONE 4/NONE	6/OUT
6		

Rule 7 A:

❖ If there have been no erasures and the rewrite is successful return to the previous state.

3. Unreachable States:

- ❖ An un reachable state is like unreachable code If a transition is not specified between two states then that states are unreachable. That is if any incorrect transition occurs then the state becomes unreachable.
- ❖ There may be a transition from unreachable states to other states.

4. Dead States:

- ❖ A dead state is a state that once entered cannot be left.
- ❖ In programming, a set of states may be dead because a program has two stages.
- ❖ In the first stage an initialization process takes place that consists of number of states to be initialized.
- ❖ In the second stage strongly connected set of functional states takes place in which operations of the states cannot be completed. So the functional states become dead states. The only solution to this problem is system restart.

(4) Output Errors:

- ➢ The errors in output are called output errors.
- ➢ The states, the transitions, and the inputs may be correct & there may be no dead or unreachable states, but the output for the transition may be incorrect.
- ➢ Output actions must be verified independently for states and transitions.

(5) Encoding Bugs:(public question)

- ➢ Encoding is a process of converting or coding the inputs, transitions, and outputs of the state.
- ➢ Encoding process is applied in both explicit and implicit finite state machines.
- ➢ The encoding bugs are more common at the time of input coding, output coding and state coding in an explicit state machine.
- ➢ The encoding bugs may also exist in an implicit finite state machine, because of different views made by programmer and tester.
- ➢ The behavior of a finite state machine is invariant under all encodings.
- ➢ That is say that the states are numbered 1 to n.
- ➢ If you renumber the states by an arbitrary permutation, the finite state machine is unchanged. Similarly for input and output code is unchanged.
- ➢ Therefore if you present your version of the finite state machine with a different encoding and if the programmer objects to renaming then there is encoding bugs.
- ➢ You may have to look at the implementation for these, especially the data dictionary.
- ➢ The implementation of the fields as bunch of bits tells you the potential size of the code.
- ➢ If the number of code value is less than this potential, there is an encoding process.
- ➢ In strongly typed languages with user defined semantic types the encoding process is probably a type conversion a set membership to integer.
- ➢ Again you may have to look at the program to spot potential bugs of this kind.

(3) State Testing:

(i) Impact of Bugs:

- ➢ Let us say that a routine is specified as a state graph that has been verified as correct in all details.
- ➢ From the following the bugs may occur.
  1. Wrong number of states
  2. Wrong transition
  3. Wrong output for a given transition
  4. Pair of states are wrongly made equivalent
  5. Set of states are split to create in equivalent duplicates.
  6. Set of states become dead.

7. Set of states become unreachable.

(ii) Principles:(public question)

- State testing is defined as a functional testing technique to test the functional bugs in the entire system.
- The principles for state testing are very similar to the principles of path testing.
- For path testing it is not possible to test every possible path in a flowgraph.
- Similarly for state testing it is not possible to test every possible path in a state graph.
- In a state graph a path is a sequence of transitions caused by a sequence of inputs.
- In state testing the primary interest is given to the states and transitions rather than outputs.
- In state testing define a set of covering input sequences and for each step in each input sequence define the expected next state, the expected transition and the expected output code.
- A set of tests consists of three sets of sequences
  1. Input sequences.
  2. Corresponding transitions
  3. Output sequences.

(iii) Limitations and extensions:

The limitation is: State transition coverage in a state graph does not guarantee complete testing.

The extension:

- Chow defines a hierarchy of paths and methods for combining paths.
- The simplest is called a 0 switch which corresponds to test each transition independently.
- The next level consists of testing transition sequences consisting of two transitions called 1 switch. The maximum length switch is an n-1 switch where n is the number of states.

The different advantages are

- State testing is useful when the error corrections are less expensive.
- State testing is also useful when the testers want to detect a specified input.
- A state testing is specifically designed for catching the deep bugs.
- A state testing provides easiness during the design of tests.

The different disadvantages are

- State testing does not provide through testing because when a test is completed there might be some bugs remains in the system. Testers require large number of input sequences to catch transition errors, missing states etc..

(iv) What to model:

- Combination of hardware & software can be modeled sufficiently complicated state graph.
- The state graph is behavioral model that is it is functional rather than structural.

(v) Getting the data:

- Here labor intensive data gathering is needed and needs more meetings to resolves issues.

(vi) Tools:

- Tools for hardware logic designs are needed.

(4) Testability tips:

(i) A balm for programmers:

- The key to testability design is easy that is we can easily build explicit finite state machines.

(ii) How big How small:

- For two finite state machines there are only eight good and bad ones.
- For three finite state machines there are eighty possible good and bad one.
- Similarly for Four state machines 2700 most of which are bad and for five state machines 275000 most of which are bad. For six state machines 100 millions most of which are bad.

(iii) Switches, Flags and unachievable paths :

- The functionality of switches and flags are almost similar in the state testing.

- Switches or flags are used as essential tool for state testing to test the finite state machine in every possible state.
- A flag or switch value is set at the initial process of finite state machine, and then this value is evaluated and tested.
- Depending on its value the specific path is selected to find an easiest way for testing the finite state machine.
- Mostly the switch or flag works on true or false condition.
- In figure a flag is set to p in the program. This p variable is assigned to some value which can be evaluated.
- Depending on its value a path is separated into branches in order to proceed testing in either way that is u or x
- This also can be done by removing a flag and separating v path into two different paths w,y as shown in the above figure.
- Unachievable paths those paths which don't interact with each other.
- Here there are four paths u,w,x,y in that two are not achievable and two are achievable.
- That is u is not achievable to path y and path x is not achievable to path w & u is achievable to path w and path x is achievable to path y.
- Finally both the paths uw and xy are needed to cover the branches.
- In the above figure there are three flag variables p,q,r in the program.
- These variables are assigned some values that can be evaluated and based on which the paths are separated into branches.
- The main benefit of using this implementation is to remove the unnecessary combination from the decision tree as shown in the figure c.

<u>(iv) Essential and inessential finite state behavior:</u>

- To understand an essential and inessential finite state behavior, we need to know the concept of finite state machines and combinational machines.
- There is a difference between finite state machines and combinational machines in terms of quality.
- In combinational machines a path is chosen depending on the truth values of predicates.
- The predicate truth values are the values which once determined will never change and always remains constant.
- In these machines a path is equivalent to a Boolean algebraic expression over the predicates
- Further more it does not matter in which order the decisions are made.

<u>(v) Design guide lines:</u>

- Fine state machine is represented by a state graph having a finite number of states and a finite number of transitions between states
- Finite state machine (FSM) is a functional testing tool and programming testing tool.
- That is it is an essential tool for state testing in order to identify or model the behavior of software.
- The different guide lines are given below.
  1. Initially learn the procedure of finite state machine that are used in both hardware and software.
  2. Design an abstract machine in such a way that it works properly and satisfies the user requirements.
  3. Design an explicit finite state machine.
  4. Prototype test must be conducted thoroughly to determine the processing time and space of explicit finite state machine design.
  5. If time or space is more effecting the overall system, then use shortcuts to complete the design process.

6. If there are more than a few numbers of states then use hierarchical design to represent them.
7. If there is large number of states then software tools and programming languages must be developed.
8. The capability to initialize to an arbitrary state must be inbuilt together with the transition verification instrumentation.

GRAPH MATRICES AND APPLICATIONS

(1) Motivational Overview:

(1) What are the problems with pictorial graphs?

Problems with pictorial graphs:

1. Tracing a path in a pictorial graph is difficult task.
2. There is every possibility of having an error while tracing i.e. we can miss a link or cover some links twice.
3. Even yellow marking pen also not be reliable because once the concentration is lost during marking; we will lose the position to be marked.
4. It is very difficult to generate test cases for a pictorial graph
5. The time is also wasted if pictorial graphs are used.

(2) What are the graph matrices and their applications?

(i) Graph Matrices:

- The matrix in which every node of a graph is represented by one row and one column is called a graph matrix. or The matrix that represents the structure of a graph is known as graph matrix.
- In a graph matrix each row and each column intersection represents, the relationship between the respective row nodes and column nodes.
- A graph is an abstract representation of a software structure.
- A graph can be traced thoroughly to perform a check for covering paths, sensitizing paths, predicate expressions etc.
- Here we use either pictorial graphs or graph matrices.
- Tracing a path in a pictorial graph is difficult task.
- There is every possibility of having an error while tracing i.e. we can miss a link or cover some links twice. Even yellow marking pen also not be reliable because once the concentration is lost during marking; we will lose the position to be marked.
- Graph matrices are introduced to overcome these problems.
- A graph matrix is purely based on matrix methods.

(ii) Applications:

(i) Tool Building:

- Using matrix representation and its methods we construct test tools.
- It is more difficult to generate test cases for a pictorial graph than the graph matrix.

(ii) Doing and understanding testing theory:

- Theoretically speaking, graphs are the simple structures but when used in theorem proving we use graph matrices because pictorial graphs will omit some important algorithms.

(iii) The Basic Algorithms:

- The basic algorithms represent a basic tool kit. The basic tool kit consists of
  1. Matrix multiplication is used to derive the path expression from every node to every other node.
  2. A partitioning algorithm is used for eliminating loops from graphs.
  3. A collapsing process is used to get the path expression.

(3) Write relative merits and demerits of different Graph Matrix representations?

(i) Merits:

1. Using matrix representation and its methods we construct test tools.
2. Matrix representation gives the best results.
3. Graph matrices are used for developing algorithms and proving theorems of graphs.
4. Linked list representation is used to represent graph matrices.

(ii) Demerits:

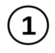
1. Graph matrix representation for two dimensional arrays is useful only for small graphs with simple link weights, however with large graphs; this matrix representation gives inconvenience.
2. Matrix representation requires a large storage space.
3. An additional weight matrix is also needed.
4. Since many entries of the graph matrices are null, the time taken to process such entries is a waste of time.

(2) The Matrix of a graph:

(1) Explain about the matrix of a graph?

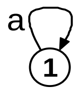
(i) Basic Principles:

- A graph matrix is array representation of nodes. In a graph matrix each row and each column intersection represents, the relationship between the respective row nodes and column nodes.
- Some examples of graphs and their associated matrices are given by.



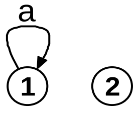
[0]

Figure (a)



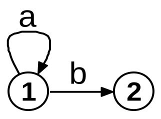
[a]

Figure (b)



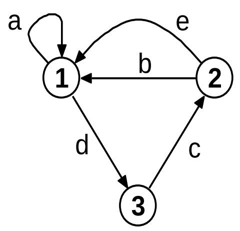
a	

Figure (c)



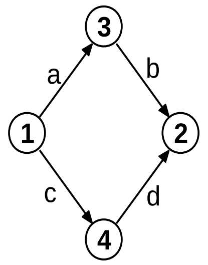
a	b

Figure (d)



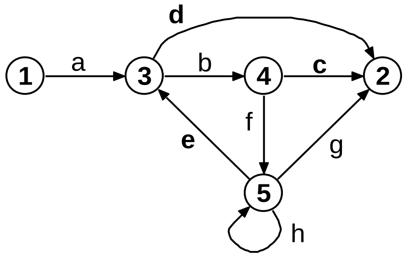
	1	2	3
1	**a**		**d**
2	**b+c**		
3		**c**	

Figure (e)



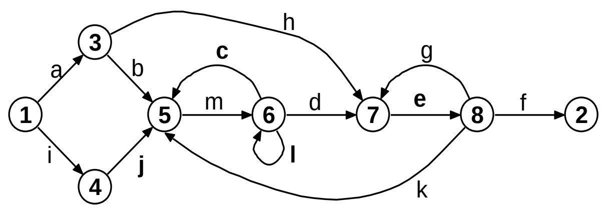
	1	2	3	4
1			**a**	**c**
2				
3		**b**		
4		**d**		

Figure (f)



	1	2	3	4	5
1			**a**		
2					
3		**d**		**b**	
4		**c**			**f**
5		**g**	**e**		**h**

Figure (f)



	1	2	3	4	5	6	7	8
1			**a**	**i**				
2								
3					**b**		**h**	
4					**j**			
5						**m**		
6					**c**	**l**	**d**	
7								**e**
8		**f**			**k**		**g**	

Figure (h)

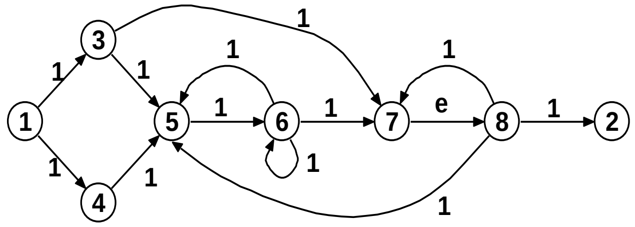
- Now observe the following
  1. The size of the matrix is equal to the number of nodes.
  2. There is a place to put every link weight between any node and any other node. i.e. The entry at a row and column intersection is the link weight of the link.
  3. A connection from node i to node j does not same that a connection from node j to node i. For example in figure (h) the (5,6) entry is m but the (6,5) entry is c..

(ii) A simple weight:

- Let '1' means that there is a connection and '0' means that there is no connection.
- The different arithmetic rules are

1+1=1	1+0=1	0+0=0
1x1=1	1x0=0	0x0=0

- A matrix with link weights defined with 1 or 0 is called a connection matrix.
- Consider the following flowgraph and its matrix representation.



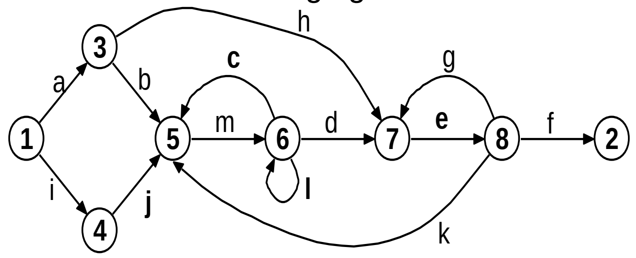
	1	2	3	4	5	6	7	8	
1			**1**	**1**					**2-1=1**
2									
3					**1**		**1**		**2-1=1**
4					**1**				**1-1=0**
5						**1**			**1-1=0**
6					**1**	**1**	**1**		**3-1=2**
7								**1**	**1-1=0**
8		**1**			**1**		**1**		**3-1=2**
									6+1=7

- Each row of a matrix denotes the outlinks corresponding to that node and each column denotes the inlinks corresponding to that node.
- A branch node is a node with more than one non zero entry in its row. For example rows 1,3,6, and 8 of the above figure have more than one entry, so these nodes are branch nodes.
- A junction node is a node with more than one non zero entry in its column. For example columns 5,6 and 7 of the above figure have more than one entry, so these nodes are junction nodes
- By subtracting 1 from the total number of entries in each row and ignoring rows with no entries we obtain the number of decisions for each row. Adding these decision values and then adding 1 to the sum gives the graph's cyclomatic complexity.
- In the above figure the graph's cyclomatic complexity is 7.

(iii) Further notation:

- The link weight between node i and node j, is denoted by a_{ij}.

- A self loop at node i is denoted by a_{ii} The link weight for the link between nodes j and i is denoted by a_{ji}.
- Consider the following figure.



- From the above figure

  abmd=$a_{13}\, a_{35}\, a_{56}\, a_{67}$

  degef=$a_{67}\, a_{78}\, a_{87}\, a_{78}\, a_{82}$

  ahekmlld=$a_{13}\, a_{37}\, a_{78}\, a_{85}\, a_{56}\, a_{66}\, a_{66}\, a_{67}$
- The expression $a_{ij}\, a_{jj}\, a_{jm}$ denotes that a path from node i to j, with a self loop at j and then a link from node j to m.
- The transpose of a matrix is the matrix with rows and columns interchanged.
- It is denoted by A^T.
- If $C=A^T$ then $c_{ij}=a_{ji}$
- The intersection of two matrices is denoted by A#B. If C=A # B then $c_{ij}=a_{ij}$ # b_{ij}.

(3) Node Reduction Algorithm:

Write the steps involved in Node Reduction Algorithm. Illustrate with an example?

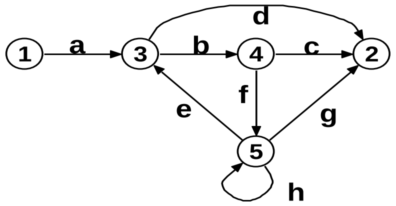
Node Reduction Algorithm:

Steps:

1. The reduction is done one node at a time by combining the elements in the last column with the elements in the last row and putting the result into the entry at the corresponding intersection. This step is called cross-term reduction. After cross term reduction the matrix size is reduced by one.
2. If there is one entry in one position and we want to enter another entry in that same position then add that two entries. This step is called parallel reduction.
3. If there is entry in principle diagonal then it represents a self loop. To remove that self loop, multiply every term in that row by the loop term. This step is called loop reduction.
4. By using the above three steps a 2x2 size matrix is obtained with the path expression. This path expression is the required path expression from node 1 to node 2.

Example:

- Consider the following flow graph.



- Specify the above flowgraph in the matrix format.

	1	2	3	4	5
1	·		a		
2		·			
3		d	·	b	
4		c		·	f
5		g	e		h

- Remove the self loop at node 5 by applying the loop reduction step.

	1	2	3	4	5
1	·		**a**		
2		·			
3		**d**	·	**b**	
4		**c**		·	**f**
5		**h*g**	**h*e**		·

- Combine the elements in the last column with the elements in the last row by applying cross-term reduction and parallel reduction steps.

	1	2	3	4
1	·		**a**	
2		·		
3		**d**	·	**b**
4		**c+fh*g**	**fh*e**	·

- Combine the elements in the last column with the elements in the last row by applying cross-term reduction and parallel reduction steps.

	1	2	3
1	·		**a**
2		·	
3		**d+b(c+fh*g)**	**bfh*e**

- Again a self loop occurred at node 3. So Remove the self loop by applying loop reduction step.

	1	2	3
1	·		**a**
2		·	
3		**(bfh*e)*(d+b(c+fh*g))**	

- Combine the elements in the last column with the elements in the last row by applying cross-term reduction.

	1	2
1	·	**a(bfh*e)*(d+b(c+fh*g))**
2		·

<u>Note: Refer other four examples from class notes</u>

<u>(4) Applications:</u>

<u>(1) Illustrate the applications of Node Reduction Algorithm:</u>

<u>(i) Maximum Path Count Arithmetic:</u>

- For theory refer unit-5 material.
- For example refer unit-8 notes.

<u>(ii) Probability of path expressions:</u>

- For theory refer unit-5 material.
- For example refer unit-8 notes.

(5) Relations:

(1) What is a Relation? What are the different properties of Relations?

Relation:

- The property by which two nodes are interconnected is called a relation.
- A relation can be represented by a link with connecting nodes.
- A link represented with link weight.
- This link weight can be numerical, logical, illogical, or whatever.
- The graph matrix which consists of unweighted simple links is called a connection matrix
- The graph matrix which consists of weighted simple links is called a relation matrix.

Different properties of relations:

- The different properties of relations are.

(i) Transitive Relations:

- A Relation R is transitive if *a*R*b* and *b*R*c* then *a*R*c*.
- Examples of transitive relations are: is connected to, is greater than or equal to, is less than or equal to, is a relative of, etc.
- Examples of intransitive relations are: is a friend of, is a neighbor of, etc.

(ii) Reflexive Relations:

- A relation R is reflexive if for every *a*, *a*R*a*. This relation represents a self-loop at every node.
- Examples of reflexive relations are: equals, is a relative of, etc.
- Examples of irreflexive relations include: not equals, is a friend of, etc.

(iii) Symmetric Relations:

- A relation R is symmetric if *a*R*b* then *b*R*a*. This relation represents if a link from a to b then there is also a link from b to a.
- A graph whose relations are symmetric is called an undirected graph and a graph whose relations are not symmetric is called a directed graph
- Examples of symmetric relations are: a relative of, is brother of, etc.
- Examples of asymmetric relations are: is the boss of, is greater than, etc.

(iv) Antisymmetric Relations:

- A relation R is antisymmetric, if *a*R*b* and *b*R*a*, then *a = b.*
- Examples of antisymmetric relations are: is greater than or equal to, is a subset of, etc.
- Examples of nonantisymmetric relations are: is connected to, is a relative of, etc.

(2) What are Equivalence Relations and Partial Ordering Relations?

(i)Equivalence Relations:

- A relation is said to be an equivalence relation if it satisfies transitive, reflexive, and symmetric properties. If a set of objects satisfy an equivalence relation, then it forms an equivalence class.
- The idea behind a partition-testing is that we can partition the input space into equivalence classes.

(ii)Partial Ordering Relations:

- A partial ordering relation satisfies the reflexive, transitive, and antisymmetric properties.
- A graph which shows partial ordering relation between its nodes is said to be partial ordered graph. Partial ordered graphs have different properties. They are
  i. loop free,
  ii. There is at least one maximum element.
  iii. There is at least one minimum element.

iv. If you reverse all the arrows the resultant graph is also partial ordered.

- The maximum element '*a*' represents the relation *x*R*a.* Similarly, the minimum element '*a*', represents a relation *a*R*x*. Examples are Trees and loop-free graphs.

(6) The Powers of a Matrix:

(i) Explain about Matrix Powers and Products?

Matrix Powers and Products:

- A graph matrix is array representation of nodes. In a graph matrix each row and each column intersection represents, the relationship between the respective row nodes and column nodes.
- The square of the matrix represents all path segments with two links long. Similarly the third power represents all path segments with three links long and so on.
- Let A be a matrix whose entries are a_{ij}. The set of all paths between any node i and any node j is given by

$$a_{ij} + \sum_{k=1}^{n} a_{ik} a_{kj} + \sum_{k=1}^{n} \sum_{m=1}^{n} a_{ik} a_{km} a_{mj} + \sum_{k=1}^{n} \sum_{m=1}^{n} \sum_{l=1}^{n} a_{ik} a_{km} a_{ml} a_{lj}$$

$$+ \cdots \sum_{k=1}^{n} \sum_{m=1}^{n} \sum_{l=1}^{n} \cdots \sum_{p=1}^{n} a_{ik} a_{km} a_{ml} \cdots a_{qp}\, a_{pj}$$

- Given a matrix whose entities are a_{ij} the square of that matrix is given by

$$a_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$$

- When given two matrices A,B with entries a_{ik} and b_{kj} respectively. The product of AB is a new matrix C whose entries are c_{ij}.

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

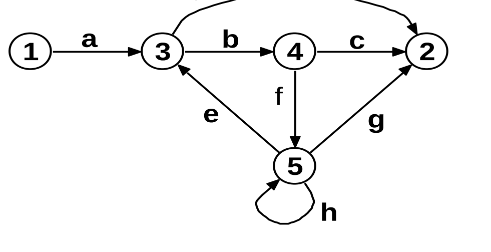
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41}$$
$$c_{12} = a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42}$$
$$c_{13} = a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33} + a_{14}b_{43}$$
$$c_{32} = a_{41}b_{12} + a_{42}b_{22} + a_{43}b_{32} + a_{44}b_{42}$$
$$c_{44} = a_{41}b_{14} + a_{42}b_{24} + a_{43}b_{34} + a_{44}b_{44}$$

- The c_{32} entry is obtained by combining, the entries in the third row of the A matrix, with the corresponding elements in the second column of the B matrix.

Example:

- Consider the following flowgraph and its graph matrix.



Let A =

	1	2	3	4	5
1			a		
2					
3		d		b	
4		c			f
5		g	e		h

$A^2 = A * A$

$$A^2 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & & & a & & \\ 2 & & & & & \\ 3 & & d & & b & \\ 4 & & c & & & f \\ 5 & & g & e & & h \end{array} \times \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & & & a & & \\ 2 & & & & & \\ 3 & & d & & b & \\ 4 & & c & & & f \\ 5 & & g & e & & h \end{array} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & & ad & & ab & \\ 2 & & & & & \\ 3 & & bc & & & bf \\ 4 & & fg & fe & & fh \\ 5 & & ed+hg & he & eb & h^2 \end{array}$$

$A^3 = A^2 * A$

$$A^3 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & & ad & & ab & \\ 2 & & & & & \\ 3 & & bc & & & bf \\ 4 & & fg & fe & & fh \\ 5 & & ed+hg & he & eb & h^2 \end{array} \times \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & & & a & & \\ 2 & & & & & \\ 3 & & d & & b & \\ 4 & & c & & & f \\ 5 & & g & e & & h \end{array} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & & abc & & & abf \\ 2 & & & & & \\ 3 & & bfg & bfe & & bfh \\ 4 & & fed+fhg & fhe & feb & fh^2 \\ 5 & & hed+ebc+h^2g & h^2e & heb & ebf+h^3 \end{array}$$

(ii) Explain about the set of all paths and the algorithm for finding set of all paths?

(a) The set of all paths:

- The set of all paths is given by the following infinite series of matrix powers.

$$\sum_{i=1}^{\infty} A^i = A^1+A^2+A^3+\ldots+A^\infty$$

- Let I be an n x n diagonal matrix where n is the number of nodes, then the above expression becomes

$$\sum_{i=1}^{\infty} A^i = A(I+A^1+A^2+A^3+\ldots+A^\infty)$$

- We know that $(A+A) = A$

  So $(A+I)^2 = A^2 + 2AI + I^2 = A^2 + 2A + I^2 = A^2 + A+A + I^2= A^2+A+I$. (Since A + A = A)

- Similarly

  $(A+I)^n = I+A^1+A^2+A^3+\ldots A^n$

- Now the original expression becomes

$$\sum_{i=1}^{\infty} A^i = A(I+A^1+A^2+A^3+\ldots+A^\infty) = A(A+I)^\infty$$

- If the paths of length n-1, where n is the number of nodes, the set of all such paths is

$$\sum_{i=1}^{n-1} A^i = A(A+I)^{n-2}$$

(b) The algorithm for finding set of all paths:

- The algorithm for finding set of all paths
  1. Express n-2 as a binary number.
  2. Calculate the successive squares of (A+I) matrix, that is $(A+I)^2$, $(A+I)^4$, $(A+I)^8$, $(A+I)^{16}$ and so on.
  3. Select only the binary powers of (A+I) matrix that correspond to a value 1 in the binary representation of (n-2).
  4. The set of all paths of length less than or equal to (n-1) is obtained from the original matrix as a result of step 3.

- For example the set of all paths for 16 nodes is given by

$$\sum_{i=1}^{15} A^i = A(A+I)^8(A+I)^4(A+I)^2$$

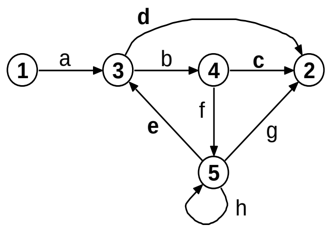
- A matrix for which $A^2=A$ is said to be idempotent matrix. A matrix whose successive power gives an idempotent matrix is called idempotent generator. The nth power of a matrix A + I over a transitive relation is called the transitive closure of the matrix.

(iii) What are the loops? How to convert graphs with loops into loop-free graphs:

- Loops are infinite sum of matrix powers.
- The way to handle loops is similar like handling loops in regular expressions.
- Loop terms are displayed on the principle diagonal of the graph matrix. A loop can be removed from a graph by using loop reduction step of Node Reduction Algorithm.

Example:

- Consider the following flowgraph and its graph matrix



$$\text{Let } A = \begin{array}{c|c|c|c|c|c|} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & & & a & & \\ \hline 2 & & & & & \\ \hline 3 & & d & & b & \\ \hline 4 & & c & & & f \\ \hline 5 & & g & e & & h \\ \hline \end{array} \qquad A + I = \begin{array}{c|c|c|c|c|c|} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 1 & & a & & \\ \hline 2 & & 1 & & & \\ \hline 3 & & d & 1 & b & \\ \hline 4 & & c & & 1 & f \\ \hline 5 & & g & e & & h+1 \\ \hline \end{array}$$

- In (A+I) matrix there is a self loop at node 5. Now we can obtain (A+I)* after removing the self loop at node 5 by applying loop reduction step.
- i.e. To remove self loop at node 5 multiply loop term h* with all row elements of row 5.

$$(A+I)^* = \begin{array}{c|c|c|c|c|c|} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 1 & & a & & \\ \hline 2 & & 1 & & & \\ \hline 3 & & d & 1 & b & \\ \hline 4 & & c & & 1 & f \\ \hline 5 & & h^*g & h^*e & & 1 \\ \hline \end{array}$$

$(A+I)^{2*} = (A+I)^* \ (A+I)^*$

$$= \begin{array}{c|c|c|c|c|c|} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 1 & & a & & \\ \hline 2 & & 1 & & & \\ \hline 3 & & d & 1 & b & \\ \hline 4 & & c & & 1 & f \\ \hline 5 & & h^*g & h^*e & & 1 \\ \hline \end{array} \times \begin{array}{c|c|c|c|c|c|} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 1 & & a & & \\ \hline 2 & & 1 & & & \\ \hline 3 & & d & 1 & b & \\ \hline 4 & & c & & 1 & f \\ \hline 5 & & h^*g & h^*e & & 1 \\ \hline \end{array} = \begin{array}{c|c|c|c|c|c|} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 1 & ad & a & ab & \\ \hline 2 & & 1 & & & \\ \hline 3 & & d+bc & 1 & b & bf \\ \hline 4 & & c+fh^*g & fh^*e & 1 & f \\ \hline 5 & & h^*g+h^*ed & h^*e & h^*eb & 1 \\ \hline \end{array}$$

$(A+I)^{3*} = (A+I)^{2*} \ (A+I)^*$

$$= \begin{array}{c|c|c|c|c|c|} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 1 & ad & a & ab & \\ \hline 2 & & 1 & & & \\ \hline 3 & & d+bc & 1 & b & bf \\ \hline 4 & & c+fh^*g & fh^*e & 1 & f \\ \hline 5 & & h^*g+h^*ed & h^*e & h^*eb & 1 \\ \hline \end{array} \times \begin{array}{c|c|c|c|c|c|} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 1 & & a & & \\ \hline 2 & & 1 & & & \\ \hline 3 & & d & 1 & b & \\ \hline 4 & & c & & 1 & f \\ \hline 5 & & h^*g & h^*e & & 1 \\ \hline \end{array} = \begin{array}{c|c|c|c|c|c|} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 1 & ad+abc & a & ab & abf \\ \hline 2 & & 1 & & & \\ \hline 3 & & d+bc+bfh^*g & 1+bfh^*e & b & bf \\ \hline 4 & & c+fh^*g+fh^*ed & fh^*e & 1+fh^*eb & f \\ \hline 5 & & h^*g+h^*ed+h^*e(d+bc) & h^*e & h^*eb & 1+h^*ebf \\ \hline \end{array}$$

- No new loops were found for the second power matrix

- But the third power matrix has a loop term bfh*e at node 3. So all other entries in that row are multiplied by (bfh*e)*. Similarly there is a loop term (fh*eb) at node 4.
- So all other entries in that row are multiplied by (fh*eb)*. Also a loop term (h*ebf)* at node 5.
- So all other entries in the fifth row is multiplied by (h*ebf)*

(iv) Explain about Partitioning Algorithm in detail?

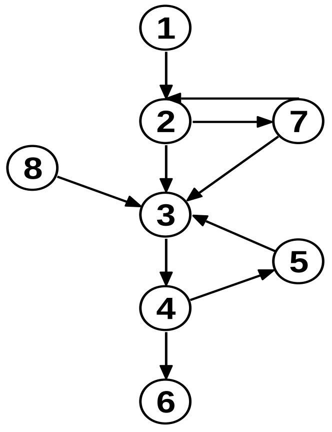
Partitioning Algorithm:

- It is an algorithm which is used to transform the graphs with loops into loop free graphs.
- There are certain points to remember here. They are
  1. A graph is loop free at the top level.
  2. Many graphs with loops are easy to analyze, if you know where to break the loops.
  3. This algorithm is used to develop a tool which can identify the loops.
- The partition algorithm represents.

  $(A+I)^n \# (A+I)^{nT}$

Example:

- Consider the following flowgraph and its graph matrix (A + I).



$$A+I = \begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & 1 & 1 & & & & & & \\ 2 & & 1 & 1 & & & & 1 & \\ 3 & & & 1 & 1 & & & & \\ 4 & & & & 1 & 1 & 1 & & \\ 5 & & & 1 & & 1 & & & \\ 6 & & & & & & 1 & & \\ 7 & & 1 & 1 & & & & 1 & \\ 8 & & & 1 & & & & & 1 \end{array}$$

- The transitive closure matrix $(A+I)^n$ can be obtained by using the following steps.

  Step:1: Mark all diagonal entries by 1

  Step:2: The flow from node 1 to node 6 is 1-2-7-2-3-4-5-3-4-6
  So mark nodes 1,2,3,4,5,6,7 by 1 in the first row

  Step:3: The flow from node 2 to node 6 is 2-7-2-3-4-5-3-4-6
  So mark nodes 2,3,4,5,6 by 1 in the second row.

  Step:4: The flow from node 3 to node 6 is 3-4-5-3-4-6.
  So mark nodes 3,4,5,6 by 1 in the third row

  Step:5: The flow from node 4 to node 6 is 4-5-3-4-6.
  So mark nodes 3,4,5,6 by 1 in the fourth row

. Step:6: The flow from node 5 to node 6 is 5-3-4-6.
  So mark nodes 3,4,5,6 by 1 in the fifth row.

  Step:7: The flow from node 6 is only 6. So mark node 6 by 1 in the sixth row.

  Step:8: The flow from node 7 to node 6 is 7-2-3-4-5-3-4-6
  So mark nodes 2,3,4,5,6,7 by 1 in the seventh row

. Step:9: The flow from node 8 to node 6 is 8-3-4-5-3-4-6
  So mark nodes 3,4,5,6,8 by 1 in the eighth row

- Therefore the transitive closure matrix is.

$(A+I)^n$ =

	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	
2		1	1	1	1	1	1	
3			1	1	1	1		
4			1	1	1	1		
5			1	1	1	1		
6						1		
7		1	1	1	1	1	1	
8			1	1	1	1		1

- The transpose of $(A+I)^n$ is.

$(A+I)^{nT}$ =

	1	2	3	4	5	6	7	8
1	1							
2	1	1					1	
3	1	1	1	1	1		1	1
4	1	1	1	1	1		1	1
5	1	1	1	1	1		1	1
6	1	1	1	1	1	1	1	1
7	1	1					1	
8								1

- The intersection of transitive closure matrix $(A+I)^n$ and transpose matrix $(A+I)^{nT}$ is given by, identifying similar rows and column entries from $(A+I)^n$ and $(A+I)^{nT}$

$(A+I)^n\#(A+I)^{nT}$=

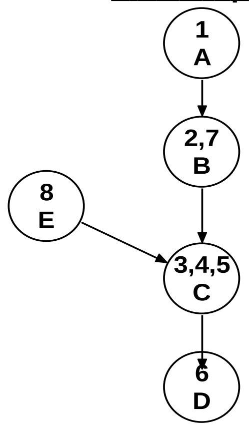
	1	2	3	4	5	6	7	8
1	1							
2		1					1	
3			1	1	1			
4			1	1	1			
5			1	1	1			
6						1		
7		1					1	
8								1

- From the above matrix, by comparing a row/column with other rows/columns, the equivalent nodes to be grouped.
- After grouping

  Let A=[1]   B=[2,7]   C=[3,4,5]   D=[6]   E=[8]

- The graph and graph matrix representation to the above values is given by

<u>FlowGraph</u> .



<u>Graph Matrix</u>

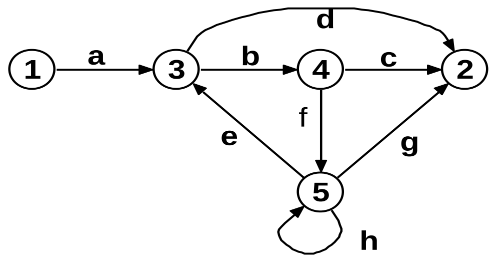
	A	B	C	D	E
A	1	1			
B		1	1		
C			1	1	
D				1	
E			1		1

(v) Explain about Breaking Loops And Applications:

- Consider the matrix format of a flowgraph.
- If there are any entries on the principal diagonal, then break the loop for that link.
- Here we use successive powers of the matrix.
- Another way to break the loop is applying the node reduction algorithm.
- Here we break the loop or we remove the self loop at any node is, by multiplying loop term with all other entries in that corresponding row.

(vi) Explain about Some matrix properties?

- To interchange the node names in the flowgraph, we must interchange both the corresponding rows and the corresponding columns of the node names in the graph matrix.
- Consider the following flowgraph and its Graph matrix.



	1	2	3	4	5
1			a		
2					
3		d		b	
4		c			f
5		g	e		h

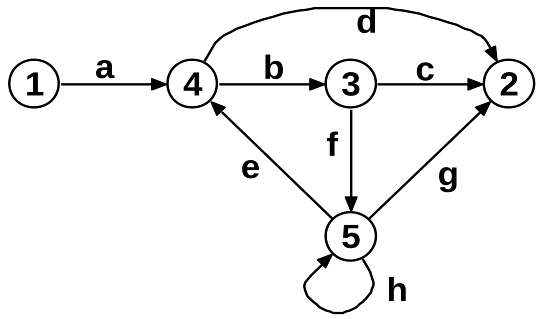
- Interchange rows 3 and 4 to the above graph matrix.

	1	2	3	4	5
1			a		
2					
3		c			f
4		d		b	
5		g	e		h

- Interchange columns 3 and 4 to the above graph matrix.

	1	2	3	4	5
1				a	
2					
3		c			f
4		d	b		
5		g		e	h

- The flowgraph to the above graph matrix is given by.



- By comparing the above flowgraph with the given flowgraph, it is proved that nodes 3,4 are interchanged

(7) Building Tools:

Explain about building tools of graph matrices?

i. Matrix Representation in software:

- A graph is an abstract representation of the software structure.
- Theoretically speaking, graphs are the simple structures but when used in theorem proving we have to apply graph matrices.
- It consists of

a) Overview:

- We prove theorems and develop graph algorithms by using graph matrices. When we want to process graphs in a computer we represent them as linked lists.

b) Node degree and graph density:

- Each intermediate node may have any number of inner links and outer links.
- The inner links of a node represents the in degree of the node.
- Similarly the outer links of a node represents the out degree of a node.
- The sum of in degree and out degree of a particular node is the degree of that node.
- The average degree of a node for a graph is between 3 and 4.
- The degree of a simple branch and simple junction is 3.
- The degree of a loop in a graph is 4.
- Any graph with average degree of 5 or 6 is said to be a very busy flowgraph.

c) What is wrong with arrays:

- We can represent the matrix as a two-dimensional array for small graphs with simple weights, but this is not convenient for larger graphs because

  (i) Space:
  - Matrix representation requires a large storage space.
  - Hence a linked list representation is used which requires less storage space.

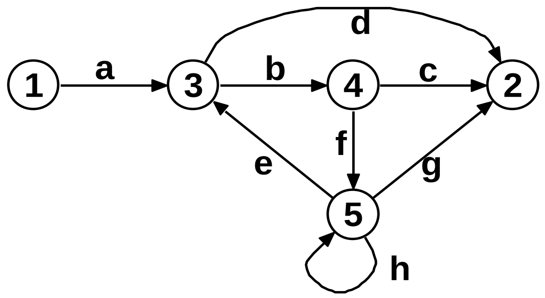
  (ii) Weights:
  - An additional weight matrix is required for complicated link weights.

  (iii) Variable-Length Weights:
  - The link weights in a flow graphs are represented in a two dimensional string array (matrix format), in which most of entries are null.

  (iv) Processing time:
  - Since many entries of the graph matrices are null, the time taken to process such entries are more.

d) Linked-list Representation:

- Consider following the flowgraph.



- The linked list for the above flowgraph is

1,3;a
2,exit
3,2;d
3,4;b
4,2;c
4,5;f
5,2;g
5,3;e
5,5;h

- ❖ List entries are usually placed in lexicographic (dictionary) order.
- ❖ The link weight expressions are stored in a string array to which link names act as pointers.
- ❖ If the link weights are values then, they are stored in an array of fixed length.

List Entry	Content
1	node 1,3;a
2	node 2,exit
3	node 3,2;d ,4;b
4	node 4,2;c ,5;f
5	node 5,2;g ,3;e ,5;h

- ❖ The node names appear only once, at the first link entry. A node name i.e starting node is used for the list entry.
- ❖ And there are back pointers for the inlinks. So we get

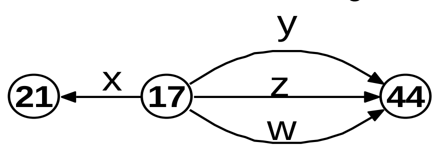
List Entry	Content	List Entry	Content
1	node 1,3;a	4	node 4,2;c ,5;f 3,
2	node 2,exit 3, 4, 5,	5	node 5,2;g ,3;e ,5;h 4, 5,
3	node 3,2;d ,4;b 1, 5,		

- ❖ It is important to keep the lists sorted in lexicographic order with the following priorities: node names or pointer outlink names, inlink names.

2. Matrix Operations:

a) Parallel Reduction:

- ❖ Parallel links after sorting are adjacent entries with the same pair of node names. Ex:



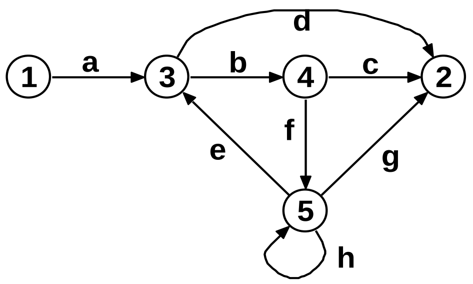
node 17,21;x
,44;y
,44;z
,44;w

- ❖ We have three parallel links from node 17 to 44. So

Node 17,21;x
,44;y(where y=y+z+w)

b)Loop Reduction:

- The loop reduction step is used to remove the loop. Here self link represents the loop.
- For removing a loop, the loop term is multiplied with all the outer links of the node at which the loop presents. Consider the following example.



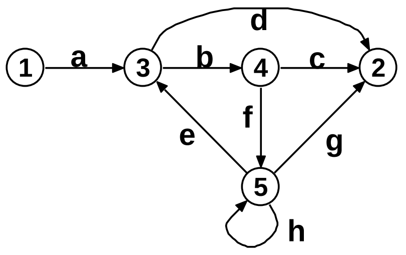
List Entry	Content Before	Content After
5	node 5,2;g	node 5,2;h*g
	,3;e	,3;h*e
	,5;h	
	4,	4,
	5,	

c) Cross term reduction:

- Select a node for reduction.
- The cross term reduction represents that we combine every inlink to the node with every outlink from that node after removing that node.
- The links created by node removal are stored in a separate list which is then sorted and thereafter merged into the master list.

d) Addition, Multiplication and other operations:

- Here addition of two matrices is simple. Multiplication is more complicated.
- Transposition is done by reversing the pointer directions, resulting in a list that is not correctly sorted. Sorting that list provides the transpose. All other matrix operations can be easily implemented by sorting, merging, and combining parallels.



List Entry	Content Before	Content After
2	node 2,exit	node 2,exit
	3,	3,
	4,	4,
	5,	5,
3	node 3,2;d	node 3,2;d
	,4;b	,2;bc
		,5;bf
	1,	1,
	5,	5,
4	node 4,2;c	
	,5;f	
	3,	
5	node 5,2;h*g	node 5,2;h*g
	,3;h*e	,3;h*e
	4,	

3. Node Reduction Optimization:

- The optimum order for node reduction is to do lowest-degree nodes first. The idea is to get the lists as short as possible as quickly as possible. Nodes of degree 3 reduce the total link count by one link when removed. A degree-4 node keeps the link count the same, and all higher-degree nodes increase the link count.
- For large graphs with 500 or more nodes and an average degree of 6 or 7, the difference between not optimizing the node-reduction order and optimizing it was about 50: 1 in processing time.