(5) UNIT-VII

STM MAM

States, State graphs and Transaction Testing

Introduction:-

The state graph and its state table are useful for describing S/w behaviour. Here finite state machine is a functional testing tool.

A finite state machine is as fundamental as boolean algebra These are used for knowing the structure of S/w, behaviours & specifications.

STATE GRAPHS:-

States:- States are denoted by nodes and each node can be identified by either number (or) char's.

Ex:- Car gear System

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

Ex:- Computer system that has several states with respect to load operating System.

1) Power on
2) Boot
3) Drivers load
4) operating system run.

Ex:
→ Word Processing menu for file manipulation.

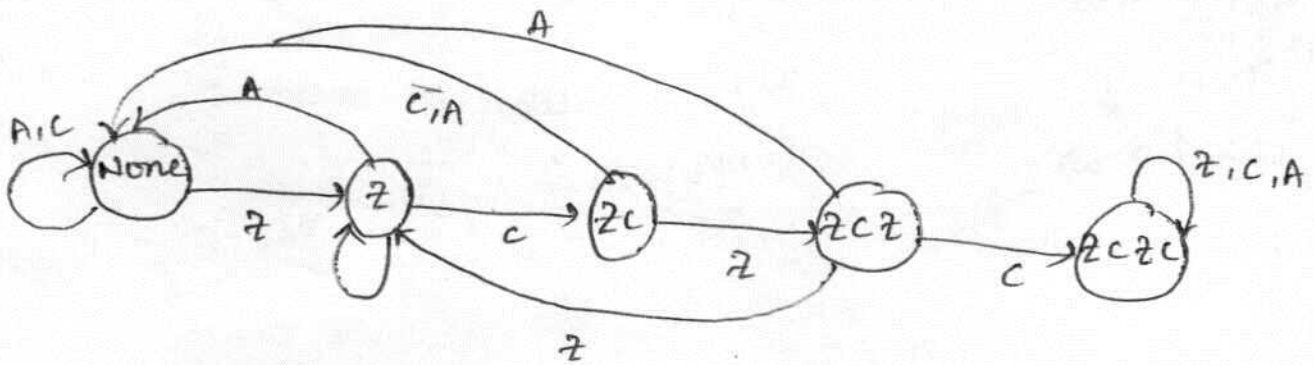
1) Copy Document
2) Delete document
3) Rename document
4) Create document
5) Compress document
6) Copy disc
7) Format disc
8) Backup disc
9) Recover from backup.

Ex: Thread Prog that has several states for executing a Process

1) New
2) Ready
3) Running
4) Block
5) Exit.

A Prog 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 " " "
5) ZCZC " " "



- **Inputs and Transitions:**

A state graph takes i/p's that may be values or variables provided to state. The changes to the state are based on the i/p's. They can be denoted by numbers (or) characters.

The "zczc" detection example can have the following kinds of i/p

1) If the system is in the "NONE" state, any i/p other than a z will keep it in that state.

2) If a z is received, the system transitions to the "z" state.

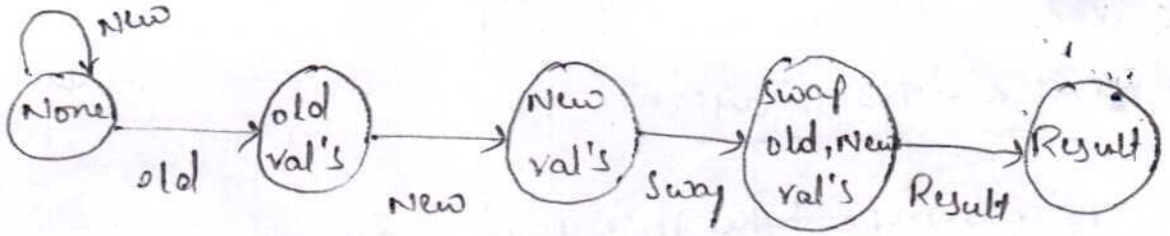
3) If the system is in the 'z' state and 'z' is received, it will remain in the 'z' state. If a 'c' is received, it will go to the 'zc' state.

4) A z received in the 'zc' state progresses to the 'zcz' state, but any other char breaks the sequence and causes a return to the "None" state.

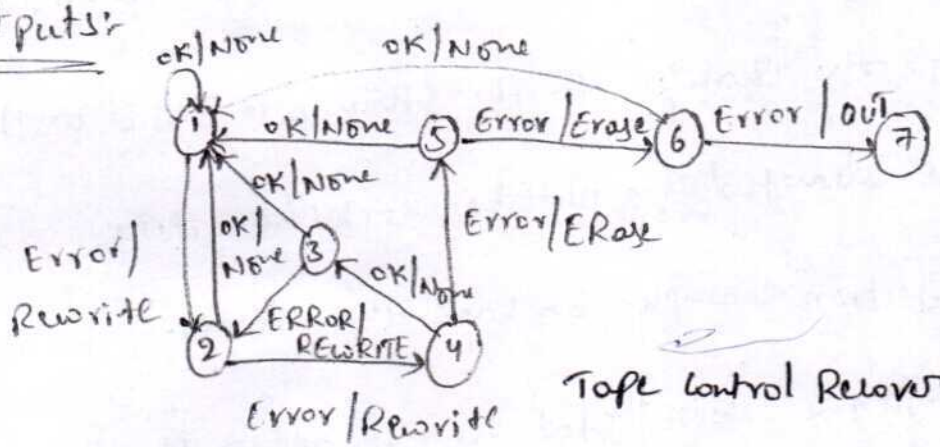
5) A c received in the 'zcz' state completes the sequence and the system enters the "zczc" state. A z breaks the sequence and causes a transition back to the 'z' state.

6) The system stays in the "zczc" state no matter what is received.

Ex:- 2)



outputs:-



Tape control Recovery Routine State graph.

outputs are denoted by letters (or) words and are separated from i/p's by a slash as follows: i/p / output.

If no write are detected (input = OK), no special action is taken (output = None). If a write error is detected (input = ERROR) back space the tape one block and rewrite the block (output = REWRITE). If the rewrite is successful (input = ok), ignore the fact that there has been rewrite. If the rewrite is not successful try another backspace and Rewrite.

There are only two kinds of i/p's (OK, ERROR) and four kinds of o/p's (REWRITE, ERASE, NONE, OUT-OF-SERVICE).

State Table:- 1) Each row of the table corresponds to a state

2) Each column corresponds to an i/p condition.

3) The box at the intersection of a row and column specifies the next state (the transition) and the o/p, if any

next state (the transition) and the o/p, if any.

STATE	OKAY	ERROR
1	1/None	2/write
2	1/None	4/write
3	1/None	2/write
4	3/None	5/Erase
5	1/None	6/Erase
6	1/None	7/OUT
7		

Ex: An example of hard disk recovery state graph.

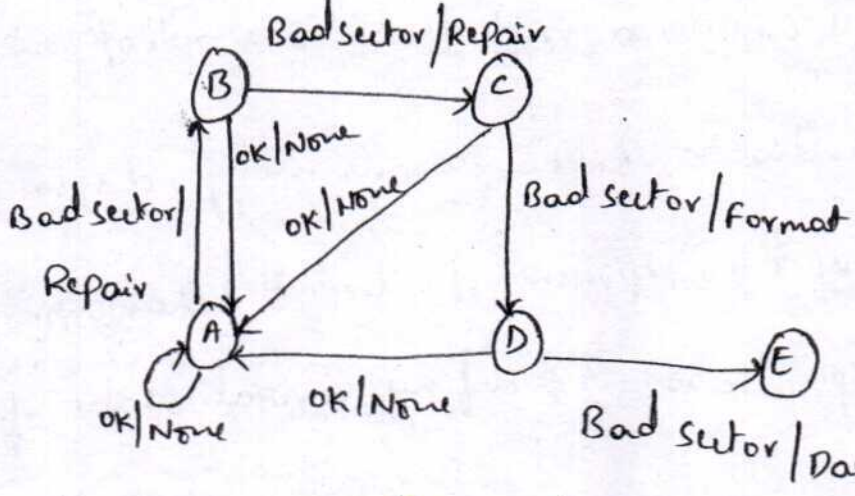


fig: Hard disk Recovery state graph.

State Table:

State	Inputs	
	OK	Bad sector
A	A/None	B/Repair
B	A/None	C/Repair
C	A/None	D/Format
D	A/None	E/Damage.
E		

i/p's: OK, Bad sector.

o/p's: None, Repair, Format, Damage.

Time versus Sequence:-

State graphs don't represent time, they represent sequence. A transition might take microseconds or centuries, a system could be in one state for milliseconds and another for eons. Long period of time.

Software Implementation of State graph:-

Implementation & operation:-

Stategraph is simply a graphical representation of overall system behaviour. It represents every state, input, transition and output of system in order to specify its behaviour.

Let us consider a real time example of mobile phone which consists of hardware, software, memory, display and so etc. A state graph is used to represent the overall behaviour of a mobile Phone in a graphical representation. It defines several states of a mobile Phone such as call, receive, talk, listen. These states are involved with diff i/p's, transitions and o/p's. The transitions involve linking b/w two states such as calling at one side and receiving at the other. A state table is a tabular representation of stategraph used to implement mobile phone with the following prog.

Begin using these writing a prog way is easier by current_state: following four tables.

1) Input code Table

2) Transition Table

3) output Table 4) Device Table.

```
Begin
current_state := Mobile_device_Table(Talk_Name)
Accept Input_Number
Input_Encode := Input_Table(Input_Number)
Pointer := Input_Encode + current_State  --> Transition table.
New_state := Transition_Table(Pointer).
output_code := output_Table(Pointer)
Invoke output_controller (output_code)
Mobile_device_Table (Talk_Name) := New_State.
End.
```

There are 4 tables involved in the above prog. They are

1) Mobile_Table Contains device information that stores the current State of every talk such as calling, receiving etc.

2) Input_Table Contains i/p information that are encoded into a list (convert into coded form) for State Processing.

3) Transition_Table defines Pointer to represent the next State for every State i/p combination.

4) o/p table specifies the o/p that relates to every state i/p combination.

Input Encoding and Input Alphabet:-

only the simplest finite state machine such as a character sequence detector. For example in the 'zczc' detector, although there are 256 possible ASCII characters, we are only interested in three diff types; 'z', 'c' and 'OTHER'. The i/p encoding could be implemented as a table that contained the following codes; 'OTHER' = 0, 'z' = 1 and 'c' = 2. Alternatively we could implement it as a PROCESS: IF INPUT = 'z' THEN code = 1 ELSE IF INPUT = 'c' THEN CODE = 2 ELSE CODE = 0 ENDIF.

The set of diff encoded i/p values is called the 'input alphabet'.

o/p Encoding and o/p Alphabet:-

There can be many diff incompatible, kinds of o/p's for transitions of finite state machine: a single character o/p for a link are in actual applications, we might want to o/p a string of char's, call a subroutine whatever we might want to do, there are only a finite number of such distinct actions, which we can encode into a convenient o/p alphabet.

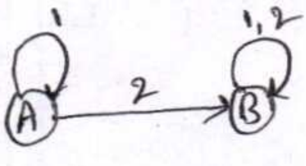
Application Comments for Tutors:- A state table or state graph representation of the behaviours of a prog or system can help us to design effective.

Good State graph and Bad :-

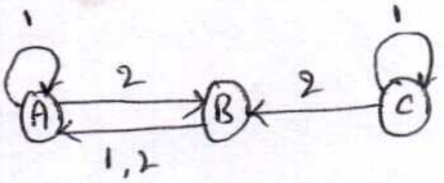
Judging Principle :-

1) Total no of states is equal to product of possibilities of factors that make up state.
2) For every state and i/p there is exactly one transition specified to exactly one, possibly the same state.
3) For every transition, there is one o/p action specified. The o/p can be trivial, but atleast one o/p does something sensible.
4) For every state, there is a sequence of i/p's that will drive the system back to the same state.

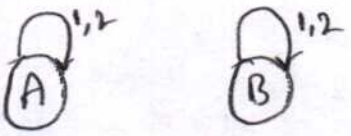
Improper State graphs :-



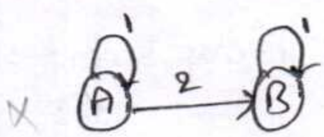
A – Not entered again
B – can't be left



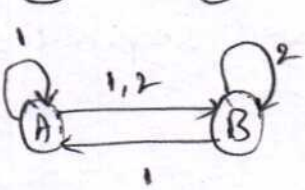
C – can't be entered.



States A and B are not reachable.



No i/p 2 for B



Two transitions for 1 in A.

"Thus results to has."

Bugs are two types in state graph.

1) State Bugs

2) Transition Bugs.

State Bugs:-

1) ~~No of~~ No of States:- The No of States in a state graph is number of States we choose.

→ Calculate States using.

1) Identify all component factors of state.

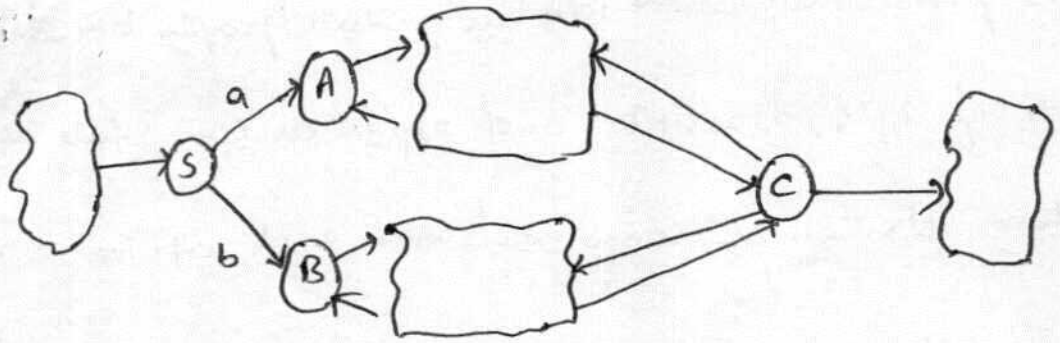
2) Identify all possible values for each factor.

3) Product of above. (multiply the no of possible values of all the factors to obtain the no of states)

2) Impossible States:- Some combination of factors may appear to be impossible.

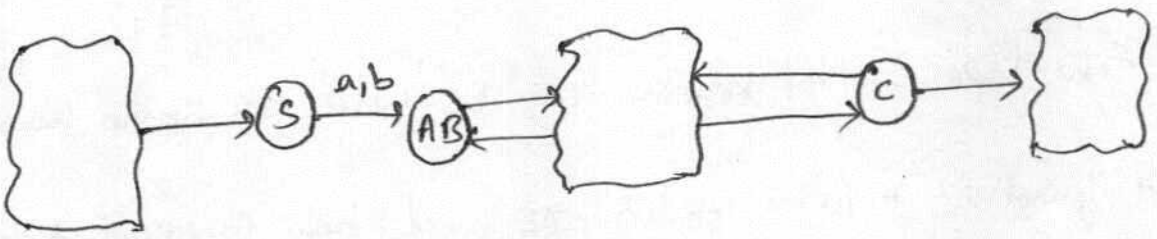
Gear	R, N, 1, 2, 3, 4	6
Direction	Forward, reverse, Stop	3
Engine	Running, Stop	2
Transmission	Okay, broken	2
Engine	Okay, broken	2
Total no of states		144

3) Equivalent States:- Two states are equivalent if every seq of i/p's starting from one state produce exactly the same sequence of o/p's when started from other state.



Equivalent States.

Let us assume that the system is in the state S, where an i/p of 'a' begins a transition to state A. On the other hand, an i/p of 'b' begins a transition to state B. The sub state graphs are represented by the above box shape whose internal information is not necessary for the equivalent process. Because these two states are treated equally, the state graph can be minimized by combining these two equivalent states.



Merged Equivalent States.

The following procedures are used to identify the equivalent states. They are.

1) If one state of row has the same sequence of i/p's, transitions and of the other state of row except state names and i/p's, then the behaviour of sy in these two sets of rows is identical. They these two states can be combined with each other.

2) If two sets of rows have same identical state graphs based on the same sequence of i/p's, transitions and o/p's except state names, then these two sets can be combined with each other.

Transition Bugs:- (The connectivity b/w two or more states is known as transition)

1) Unspecified and contradictory (opposite) transitions:-

Every i/p state state combination should have specified transition.

1) If transition is impossible, then the mechanism may cause mal (failure) function.

2) Exactly one transition must be specified for every combination of i/p and state.

Ex:- The tape control routine can be used to know how to convert a specification into a stategraph and how contradiction can come about. Start with the first statement in the specification and add to the state graph one statement at a time.

There are some rules for converting a specification into a state graph.

Rule 1: An error counter is used in a program, which will be incremented whenever there's an error.

Rule 2: Rewrite the block, if there is an error.

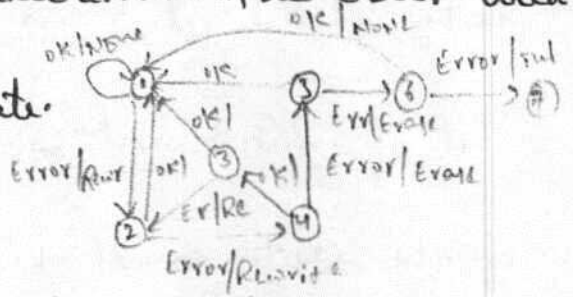
Rule 3: If there is three successive errors, erase 10 centimetery of tape and rewrite the block.

Rule 4: If there is three successive erasures and another erro occurs, Put the unit out of service.

Rule 5: Return to the normal state if the erasure was successfu Now clear the error counter.

Rule 6: If the rewrite was unsuccessful, increment the error counter, advance the state, and try another rewrite.

Rule 7: If the rewrite was successful, decrement the error coun and return to the previous state.



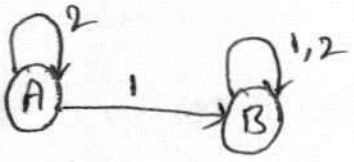
2) Unreachable Staty:

An unreachable state is like unreachable code - a state that no i/p sequence can reach. An unreachable state is not impossible. Furthermore, there may be transitions from the unreachable state to other staty, there usually are because the state become unreachable as a result of incorrect transitions.

Unreachable Staty come about from previously "impossik Staty. Some of these staty correspond to previously "impossible"

cont.

3) Dead State: A state in a state graph that once entered cannot be left is called a dead state. In a dead state, inputs cannot be processed further. For eg consider the fig:



B - can't be left

State B is a dead state.

In s/w design programming, a set of ~~values~~ states may be dead because a prog has two diff stages. In the first stage, an initialization process takes place that consists of a number of stat to be initialized. In the second stage, the strongly connected set of functional states takes in which the routine of operations of the states cannot be completed.

The initialization states are waiting to send the initialize states for the functional process, but they are unable to reach to t functional states become dead states to the initialization states because the routine of operations are not being completed. The only solution to this problem may be system crash or system restart.

Error System booting.

* Differences b/w good state graphs and Bad state graphs. (8)

* Good state graphs	Bad state graphs
1) A state graph is said to be good, when every state, i/p, transition & o/p is specified clearly and understandable.	1) A state graph is said to be bad when every state, i/p, transition o/p is not specified clearly and difficult to understand.
2) In good state graph, the sequence of i/p's is specified for every state in order to perform some action that will help the system back to the initial state.	2) In bad state graph, there is no sequence of i/p's specified, that result to the incorrect o/p's.
3) In good state graph, there is exactly one transition specified for every state and i/p combination. So that the transition bugs may not occur.	3) In bad state graph, there might be either none or more than one transitions specified for every state and i/p combinat that results to the transition
4) In good state graph, the bugs are less and easy to identify.	4) In bad state graph, the bugs are more and difficult to identify.

* State Testing:

State testing is defined as a functional testing technique to test the functional bugs in the entire system.

For path testing it is not possible to test every possible path in a flow graph. Similarly, for state testing, it is not possible to test every possible path is a state graph. In a state graph a path is a sequence of transitions caused by a sequence of i/p's like path testing, state testing examines each transition in a state graph to guarantee complete testing.

A bug can manifest itself as one or more of the following symptoms:

1) Wrong no of states
2) Wrong transition for a given state i/p combination.
3) Wrong o/p for a given transition.
4) States or sets of set states that are split to create inequivalent duplicates.
5) States or sets of states that have become dead.
6) States or sets of states that have become unreachable.

Advantages:

The advantages of state testing are as follows,

1) State testing is useful when the error corrections are less expensive.
2) A state testing is also useful when the testing [illegible]

Disadvantages:

The disadvantages of state testing are as follows,

1) State testing does not provide through testing because when a test is completed, there might be some bugs remained in the system.

2) Testers requiring large no of i/p sequences to catch transition error missing states, extra states and the link.

The starting point of state testing is:

1) Define a set of covering i/p sequences that get back to the initial state when starting from the initial state.

2) For each step each i/p sequence, define the expected next state, the expected transition, and the expected o/p code.

Set of tests, then consist of 3 sets of sequences:

1) i/p sequences.

2) Corresponding transitions or next state names

3) o/p sequences.

* Testability Tips:-

As per the state testing is concerned, there are lot of views or tips from different programmers and testers.

1) The primary key to testing is to build explicit finite state machine and implementation of it helps programmers to get relief themselves from diff testers.

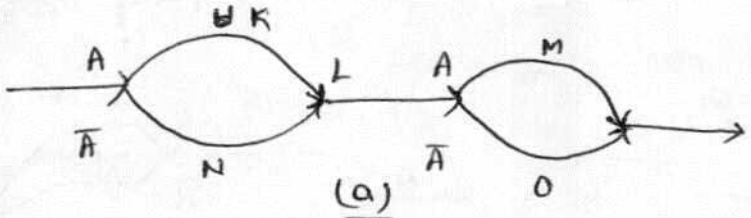
2) A finite state machine and identify how to implement each part of that model.

3) To test, eliminate the switches and flags from the machine.

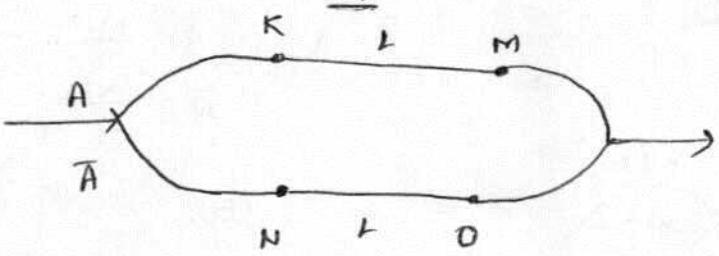
4) Identify the essential and inessential behaviour of the finite state machine and then don't implement inessential behaviour of the machine.

1) Switches, flags and unachievable paths:-

The functionality of switches and flags is almost similar as both refer to the same term in the state testing. Switches or flags are u as an essential tool for state testing to cover ~~the~~ or test the fini state machine, then this value is evaluated and tested. Depending on its value the specific path is selected to find an easiest way fi testing the finite state machine.



(a)

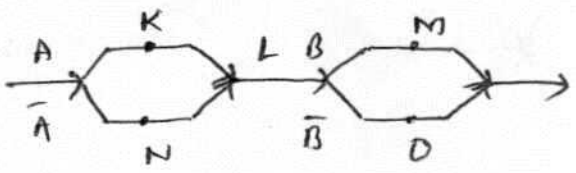


(b)

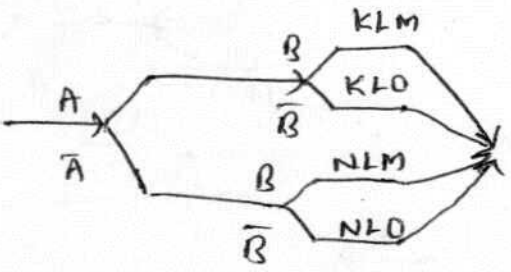
one flag variable Program.

Fig(a) a flag variable is set as 'A' in the program. This 'A' variat is assigned some value which can be evaluated. Depending on it result a path is seperated into branches in order to proceed testi in either way. This path can also be done by removing a flag and Separating L path in to two diff Pathy ~~in~~ as shown in the fig(b Unachievable Pathy are those pathy which do not interact with eac other. It is important to note that there are four pathy 'KMNO', where two pathy are not achievable to other two pathy such as Path 'K' is not achievable to path 'O' and path 'N' is not achieval to Path 'M'. Two pathy are achievable to each other such as Path 'K' is achievable to path 'M' and path is 'N' is achievable to Pat 'O'. Finally, both the pathy 'KM' and 'NO' are needed to cover branches.

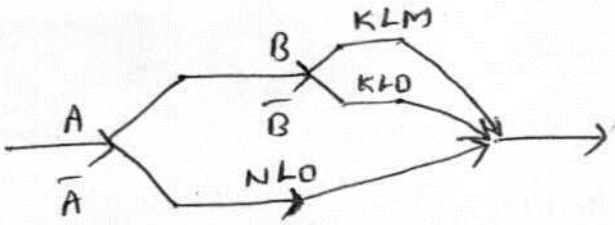
Let us take another ex: of a Prog with two flag variables.



(a)



(b)



(c)

Above fig there are two flag variables A and B in the Prog. These variables are assigned some values that can be evaluated and based on which the paths are separated into branches. The decision is taken to select an easiest path for testing. Therefore, all the paths of a tree are achievable such as 'KMNO' and all the paths are nec to cover the branches.

2) Essential and Inessential finite state ~~machine~~. Behaviour:

To understand an essential and inessential finite state behaviour, we need to know the concept of finite state machines and combinatio machines. There is a diff b/w finite state machines and combinatio machines in terms of quality. In combinational machines, a path is

rely on the Previous Processing. The predicate truth values are the values which once determined will never change and always remain constant.

For all possible i/p's, the combinational machine has exactly one state and one transition that leads to itself.

The implementation of combinational prog is based on decision table or decision tree to classify its control logic.

Essential finite state machine is purely based on acknowledgment and has no specific logic for implementation without acknowledgements.

Ex:- Flip-Flop.

The following are the condition of inessential finite state behaviours.

i) If behaviour can be obtained by parallel prog in data flow machine.

ii) If behaviour is obtained from decision table or decision tr

iii) If expr of emit is equal to unity for a prog including lo

iv) If the prog's emit expression is not equal to unity but we don't want to loop under the looping conditions.