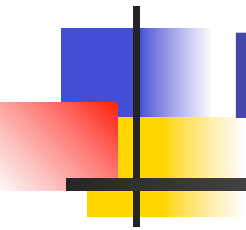
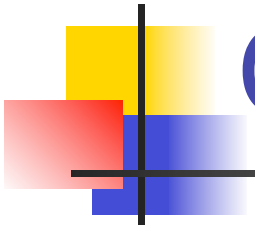


Black Box Testing (revisited)

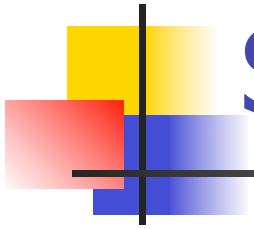
Csci 565

Spring 2007



Objectives

- Cause-Effect Graphs in Functional testing
- Input validation and Syntax-driven Testing
- Decision Table-Based Testing
- State transition testing (revisited)



Syntax-Driven testing (SDT)

- Applicable to
  - Specification validation written by a certain grammar (BNF)
  - Input data validation
- Test generation
  - Generate test cases such that each production rule is tested at least once

Example: BNF of simple Arithmetic expressions

Example

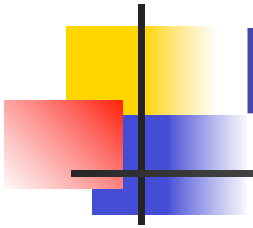
<exp> ::= <exp> + <term>|

<exp> - <term>| <term>

<term> ::= <term> * <factor> | <term> / <factor>|<factor>

<factor> ::= <id> | <exp>

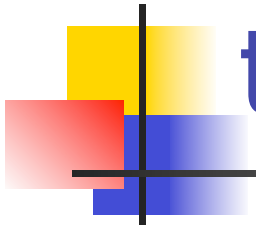
<id>::= a|b|…|z



More on SDT

- The set of test cases for SDT should include expressions that exercise the production rules
- Examples:
  - a+b * c
  - (w+v) + z

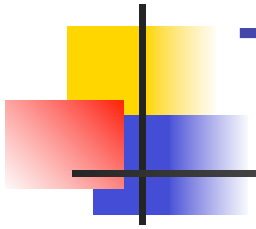
Input validation and syntax testing



- Data validation is the first line of defense against a hostile world
  - Casual
  - Malicious users
- Good designs won’t accept garbage
- Good testers will test system against possible garbage

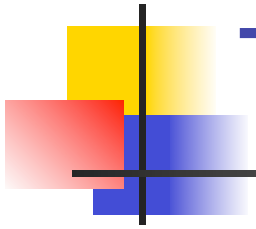
Formal speciation of input

- Use Backus-Naur Form (BNF) to specify data input
  - E.g.,
    - line::= (alpha:char)$^{7..80}$ end:line
    - alpha:char::= A/B/C/…/Z
    - end:line ::= CR CR LF/ CR LF LF



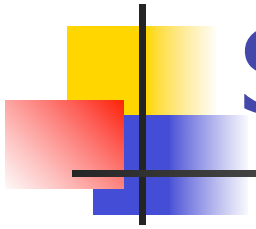
Test Case generation

- A data validation routine is designed to recognize strings defined by BNF
- String recognizer routine
  - Accepts the string (or recognizes)
  - Rejects the string
- String generator
  - When test designer attempts to generate strings



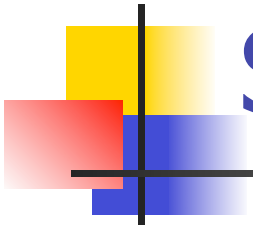
Types of incorrect actions

- There are three possible kinds of incorrect actions
  - The recognizer does not recognize a good string
  - It accepts a bad string
  - It may accept or reject a good string or a bad string, but in so doing, it fails



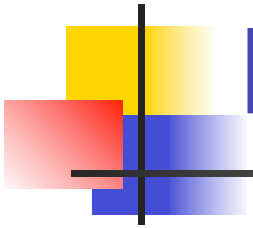
Strings' errors

- Even small specification may result in many strings
- Strings' errors can be classified as
  - Syntax error
  - Delimiter errors
  - Field-value errors



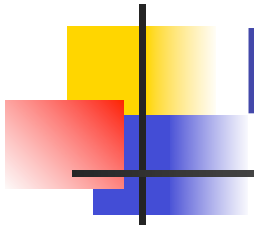
Syntax error

- E.g.,
  - Item ::= atype/btype/ctype/dtype etype
- Possible test cases
  - Wrong combination
    - E.g, /dtype atype
  - Don’t do enough
    - Dtype/ etype
  - Don’t do nothing
  - Do too much
    - Atype btype ctype dtype etype/ atype atype atype/ dtype etype atype/dtype etype



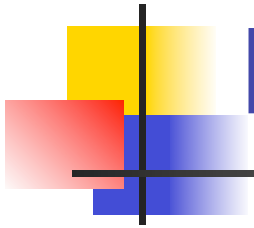
Delimiter errors

- Delimiters are chars placed between two field
- Excellent source of test cases
- Test for
  - Missing delimiter
  - Wrong delimiter
  - Not a delimiter
  - Too many delimiters
  - Paired delimiters



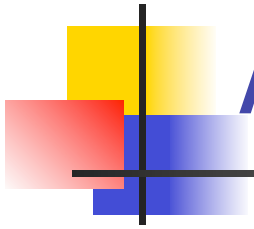
Field-value errors

- Fields have meaning and must be tested
  - Boundary values
  - Excluded values
  - Troublesome values



Problems with SDT

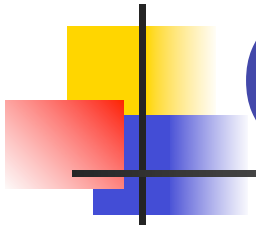
- It is easy to forget the normal cases
- Don’t get carry a way with combinations
- Don't ignore structure



Applications of SDT

- Here are some examples
  - All user interfaces
  - Operating command processing
  - All communications interfaces and protocols
  - Internal interface and protocol (call sequences)

Decision table-based Testing (DTT)

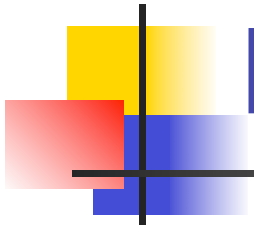


- Applicable to the software requirements written using “if-then” statements
- Can be automatically translated into code
- Assume the independence of inputs
- Example
  - If c1 AND c2 OR c3 then A1

Sample of Decision table

- A decision table is consists of a number of columns (rules) that comprise all test situations
- Action a_i will take place if c1 and c2 are true

	1	2					rn
c1	1	1					0
c2	1	0					0
	x	x					1
cn	0	0					0
a1	1	0					0
a2	0	1					1
…	0	0					0
an	o	o					1



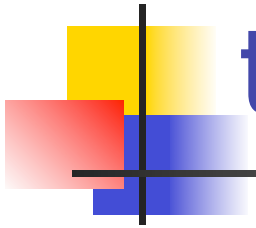
Example: Simple editor

- A simple text editor should provide the following features
  - Copy
  - Paste
  - Boldface
  - Underline
  - select

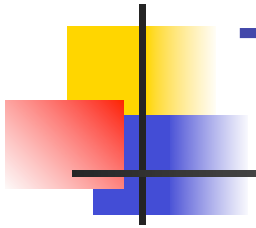
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
copy	1															
paste	0															
Undrln	0															
Bold	0															
C-test	1															
P-text	0															
U-text	0															
B-text	0															

In general, for n conditions, we need 2^n rules

Decision tables as a basis for test case design



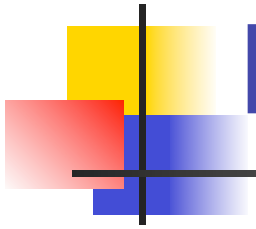
- The use of decision-table model to design test cases is applicable
  - The spec is given by table or is easily converted to one
  - The order in which the conditions are evaluated does not affect the interpretation of rules or the resulting action
  - The order in which the rules are evaluated does not affect the resulting action
  - Once a rule has been satisfied and an action is selected, no other rule need be examined
  - If multiple actions result from the satisfaction of a rule, the order in which the actions take place is not important



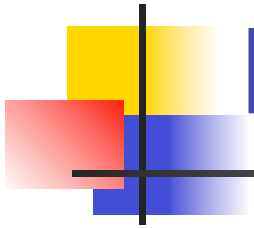
The implications of rules

- The above conditions have the following implications
  - Rules are complete (i.e., every combination of decision table values including default combinations are inherent in the decision table)
  - The rules are consistent (i.e, there is not two actions for the same combinations of conditions)

Cause-effect graphs in black box testing

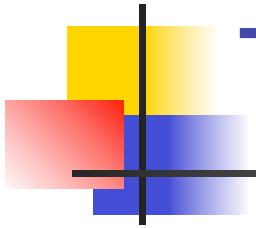


- Captures the relationships between specific combinations of inputs (causes) and outputs (effects)
  - Deals with specific cases,
  - Avoids combinatorial explosion
  - Explore combinations of possible inputs
- Causes/effects are represented as nodes of a cause effect graph
- The graph also includes a number of intermediate nodes linking causes and effects



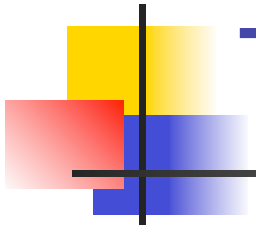
Example 2:

- Consider the following spec:
  - The character in column 1 must be an “A” or a “B”.
  - The character in column 2 must be a digit
    - In that case, the file update is made
  - If the first character is incorrect, message X12 is issued
  - If the second character is not a digit, message X13 is issued



The causes

1. Character in column 1 is “A”
2. Character in column 1 is “B”
3. Character in column 2 is a digit

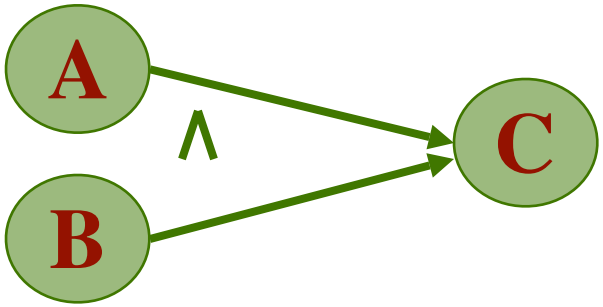


The effects

- The effects are
  - 70: update made
  - 71: message X12 is issued
  - 72: message X13 is issued

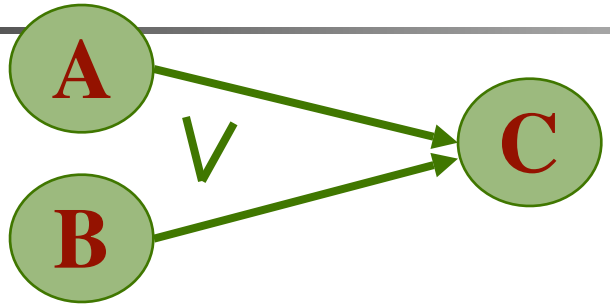
Drawing Cause-Effect Graphs

If A then B (identity)

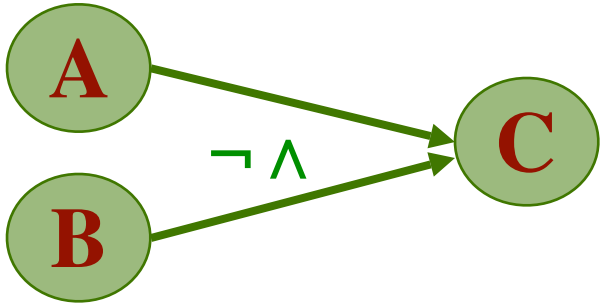


If (A and B)then C

Drawing Cause-Effect Graphs

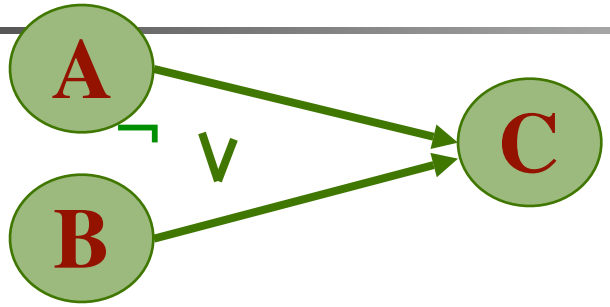


If (A or B) then C



If (not(A and B)) then C

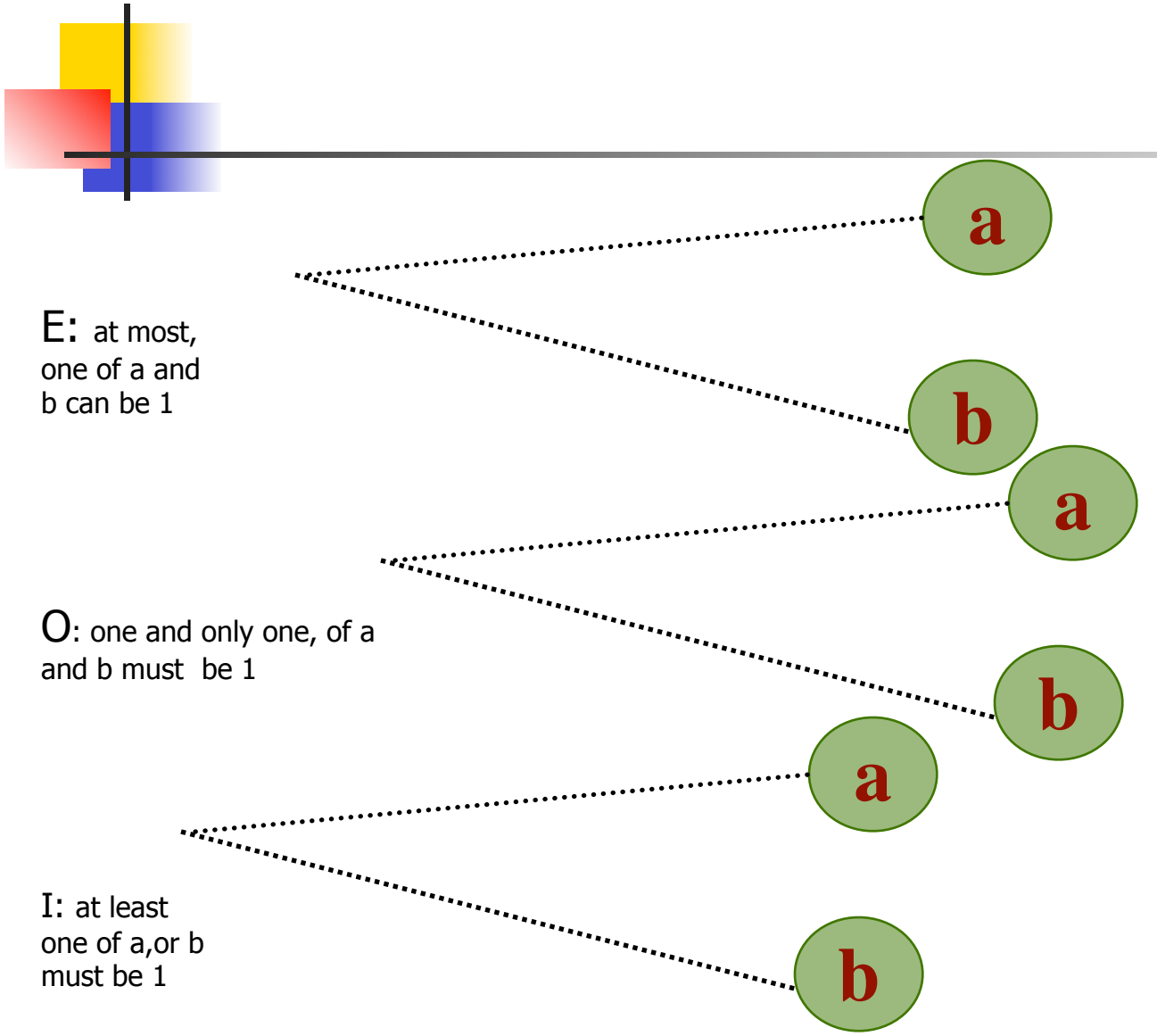
Drawing Cause-Effect Graphs



If (not (A or B))then C

If (not A) then B

Constraint Symbols

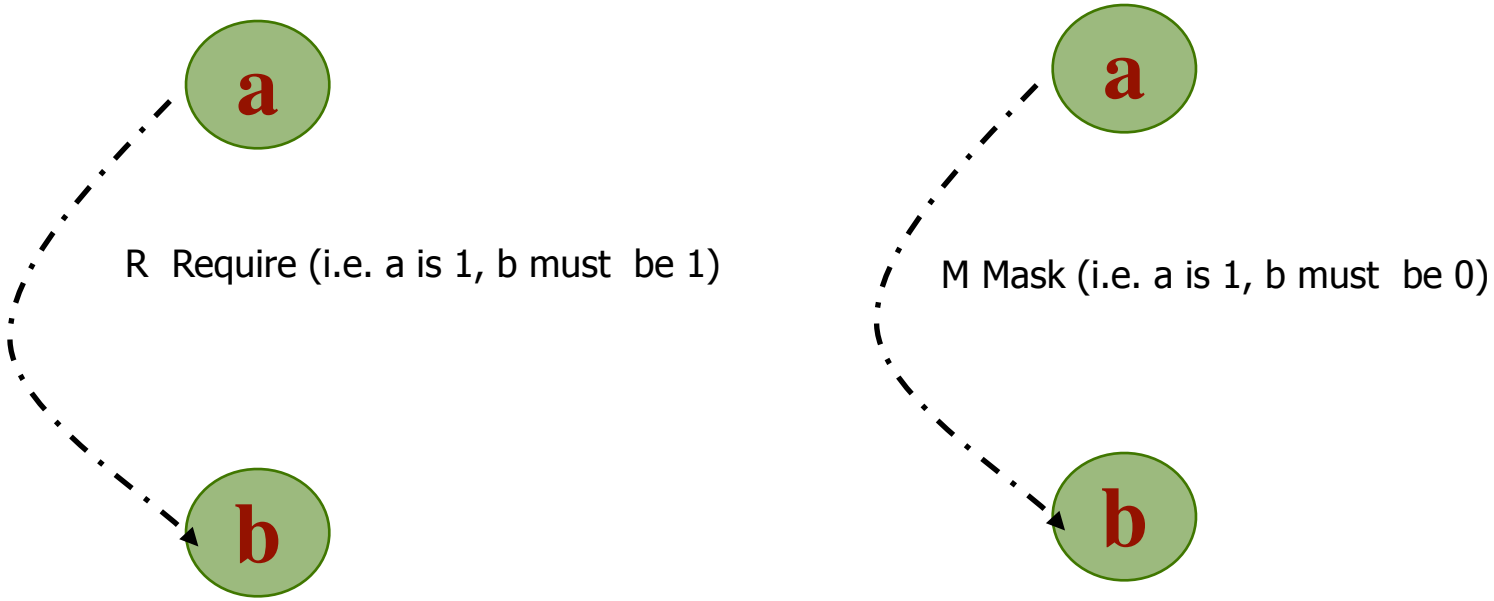


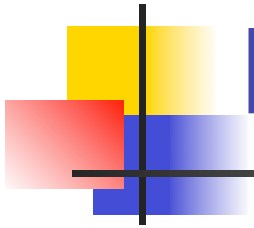
E: at most, one of a and b can be 1

O: one and only one, of a and b must be 1

I: at least one of a,or b must be 1

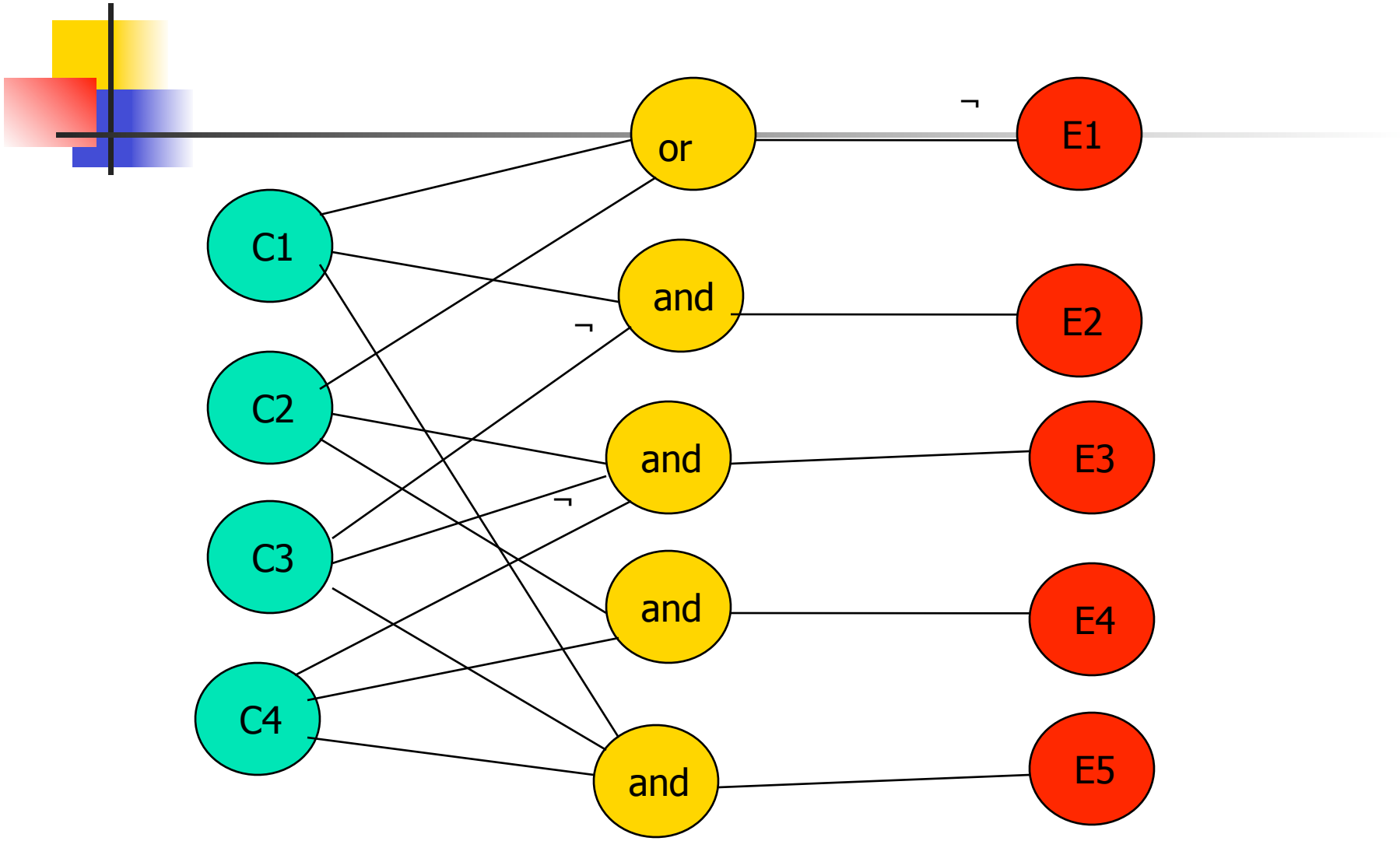
More Constraint symbols





Example: ATM

- For a simple ATM banking transaction system
  - Causes (inputs)
    - C1: Command is credit
    - C2: command is debit
    - C3: account number is valid
    - C4: transaction amount is valid
  - Effects
    - E1: Print “invalid command”
    - E2: Print “ invalid account number”
    - E3: Print “debit amount not valid”
    - E4: debit account
    - E5: Credit account


or
E1
C1
and
E2
C2
and
E3
C3
and
E4
C4
and
E5

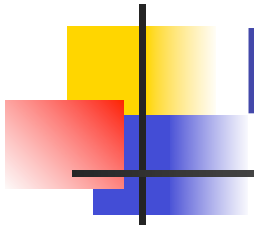
Description of processing nodes used in a Cause-effect graph

Type of processing node	description
AND	Effect occurs if all input are true (1)
OR	Effect occurs if both or one input is true
XOC	Effect occurs if one input is true
Negation ¬	Effect occurs if input are false (0)

ATM Cause-effect decision table

Don’t Care condition

Cause\effect	R1	R2	R3	R4	R5
C1	0	1	x	x	1
C2	0	x	1	1	x
C3	x	0	1	0	1
C4	x	x	0	1	1
E1	1	0	0	0	0
E2	0	1	0	0	0
E3	0	0	1	0	0
E4	0	0	0	1	0
E5	0	0	0	0	1



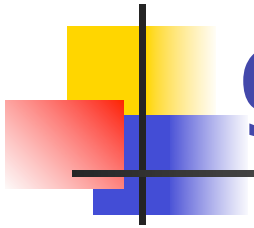
Example 3:

- Consider the following NL specifications:
  - The boiler should be shut down if any of the following conditions occurs:
    - If the water level in a boiler is below the 20000lb mark
    - The water level is above the 120000 lb, or
    - The boiler is operating in degraded mode and the steam meter fails
      - Being in degraded mode means
        - If either water pump or a pump monitors fails

Example 3: Translate NL into workable pieces (atomic specifications)

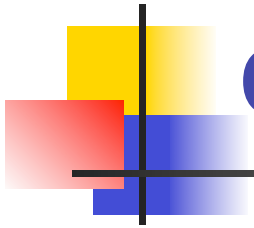
- Atomic sentences are
  - C1. the water level in boiler is below the 20000 lb (F/T)
  - C2. the water level is above the 120000 lb mark (F/T)
  - C3. A water pump has failed (F/T)
  - C4. A pump monitor has failed (F/T)
  - C5. Steam meter has failed/T
  - E. Shut the boiler down

Steps to create cause-effect graph



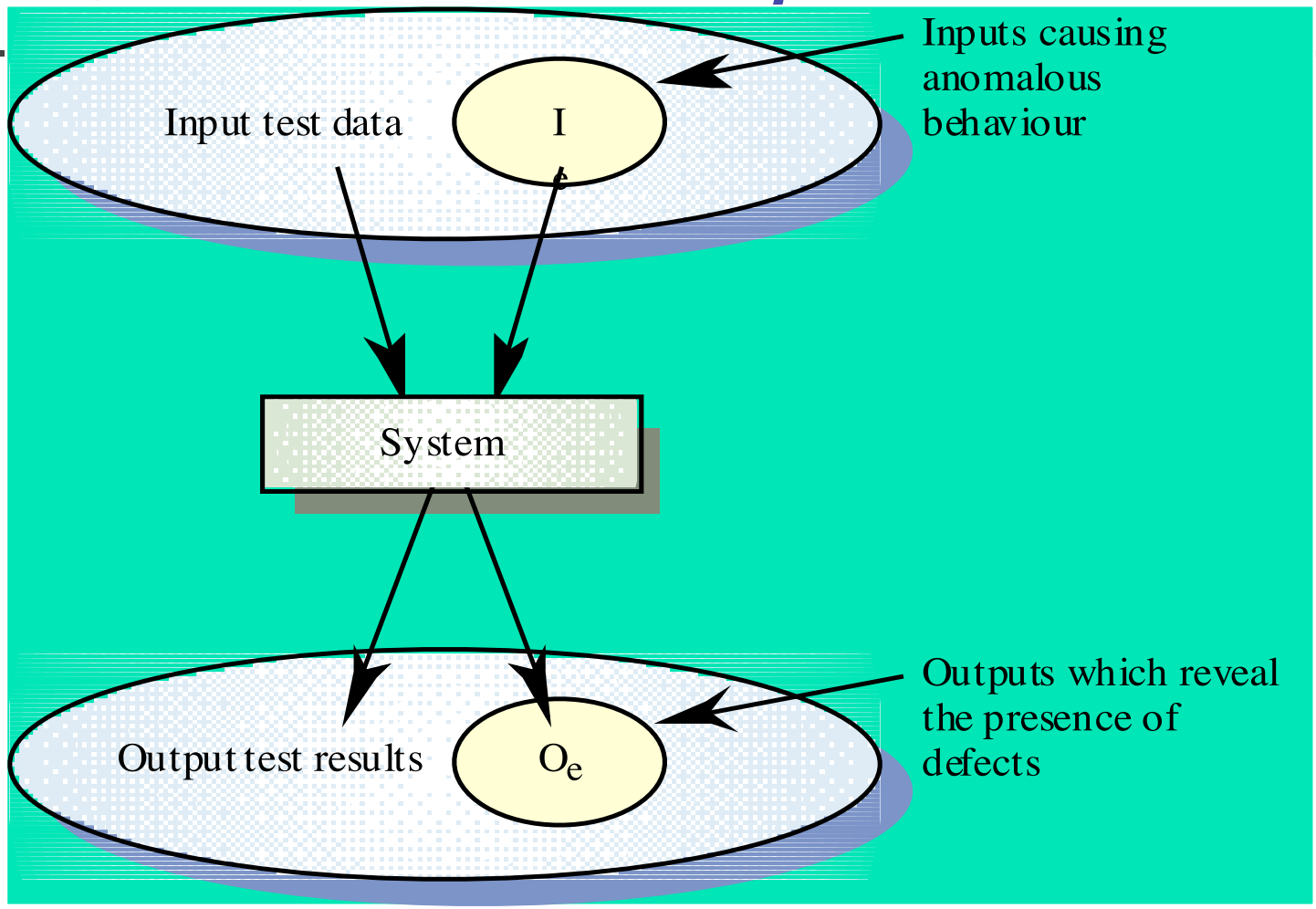
1. Study the functional requirements and divide the requirements into workable pieces
    1. E.g. of workable piece, in eCom, can be verifying a single item placed in shopping cart
2. Identify causes and effects in the specifications
    1. Causes: distinct input conditions
    2. Effects: an output condition or a system transformations.
3. Assign unique number to each cause and effect
4. Use the semantic content of the spec and trasnform it into a Boolean graph
5. Annotate the graph with constrains describing combinations of causes and/or effects
6. Convert the graph into a limited-entry decision table
7. Use each column as a test case

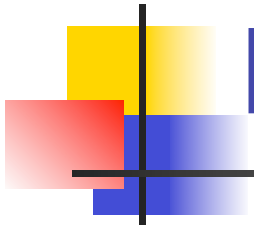
Possible research topics based on CEG



- Comparison of CEG ,FSM-based test sets, and randomly generated test cases (functional)
  - For effectiveness in terms of cost and fault detection capabilities
    - For fault detection capabilities
    - For number of test cases generated (cost)
  - For automatic generation of actual test cases

Black-box testing

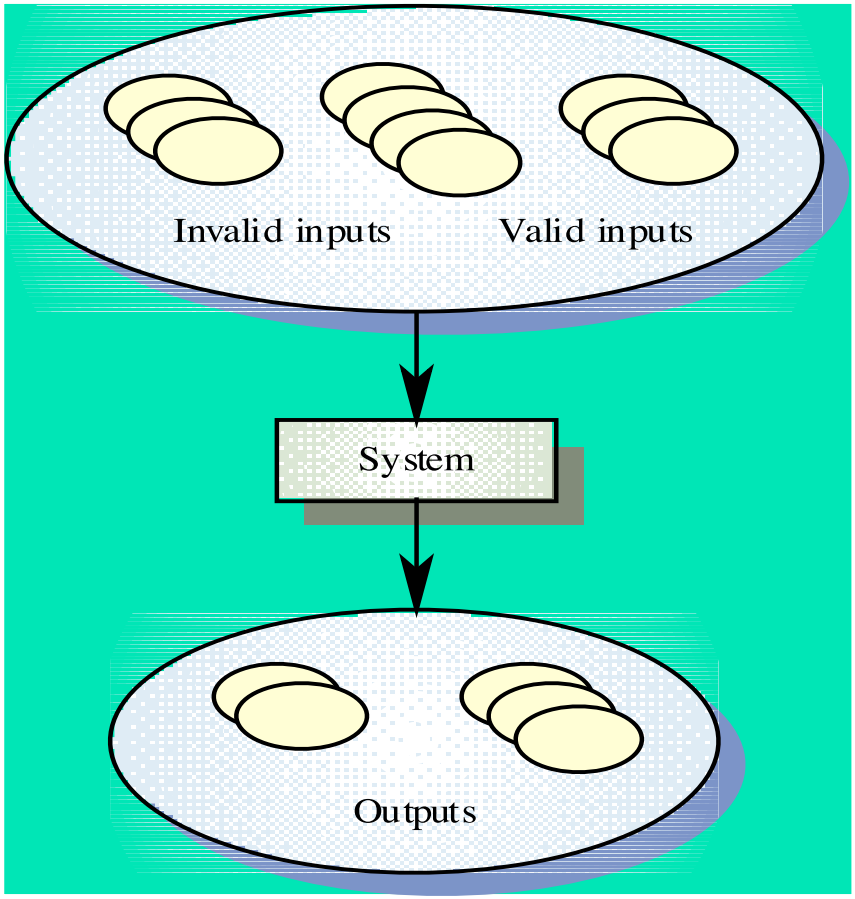


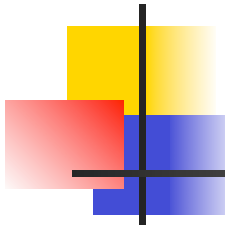


Equivalence Partitioning (EP)

- Input data and output results often fall into different classes
  - where all members of a class are related
- Each of these classes is an equivalence partition where the program behaves in an equivalent way for each class member
- The entire set (union of all classes serves
  - Completeness and non-redundancy
- Test cases should be chosen from each partition (or class)

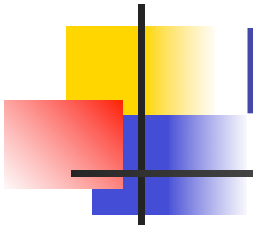
Equivalence partitioning





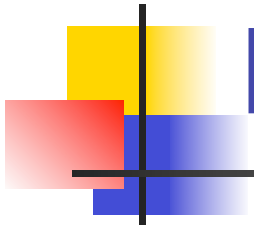
Guidelines for equivalence classes

1. If an input condition specifies range,
    1. one valid and two invalid equivalence classes are needed
2. If a condition requires a specific value,
    1. then one valid and two invalid equivalence classes are needed
3. If an input condition specifies a member of a set,
    1. one valid and one invalid equivalence class are needed
4. If an input condition is Boolean,
    1. one valid and one invalid class are needed



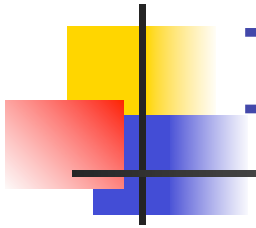
Example: ATM

- Consider data maintained for ATM
  - User should be able to access the bank using PC and modem
  - User should provide six-digit password
  - Need to follow a set of typed commands



Data format

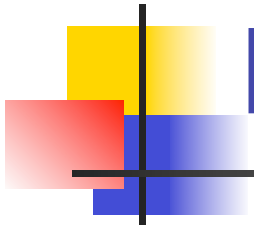
- Software accepts
  - Area code:
    - Might be blank or three-digit
  - Prefix:
    - three-digit number not beginning with 0 or 1
  - Suffix:
    - four digits number
  - Password: six digit alphanumeric value
  - Command:
    - {“check”, “deposit,” “ bill pay”, “transfer” etc.}



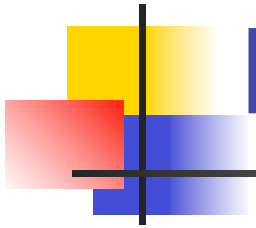
Input conditions for ATM

- Input conditions
  - Area code:
    - Boolean: the area code may or may not be present
    - Range: values defined between 200-999
    - Specific value: no value > 905
  - Prefix: range –specific value >200
  - Suffix: value (four-digit length)
  - Password:
    - Boolean: password may or may not be present
    - Or value – six char string
  - Command: set containing commands noted previously

Boundary Value Analysis (BVA)



- complements equivalence partitioning
- Works with single fault assumption in reliability theory
- Focuses is on the boundaries of the input
  - If input condition specifies a range bounded by a certain values, say, a and b, then test cases should include
    - The values for a and b
    - The values just above and just below a and b
- If an input condition specifies any number of values, test cases should be exercise
  - the minimum and maximum numbers,
  - the values just above and just below the minimum and maximum values



More on BVA

- Generalizing BVA are done
  - Using the number of variables
    - Example for Function of F with n variables, need at least 4n+1 test cases
  - The type of ranges
    - Logical quantity vs physical bounded quantities
      - E.g. of discrete and bounded: months of year, commissions
      - E.g., discrete and unbounded: length of sides in triangle problems
        - Need to create artificial min/max bounds

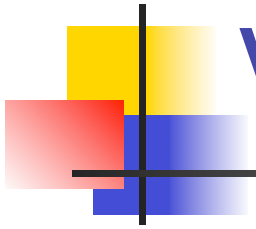
Example 2: Equivalence Partitioning

Consider a component, *generate_grading,* with the following specification:

The component is passed an exam mark (out of 75) and a coursework (c/w) mark (out of 25), from which it generates a grade for the course in the range 'A' to 'D'. The grade is calculated from the overall mark which is calculated as the sum of the exam and c/w marks, as follows:

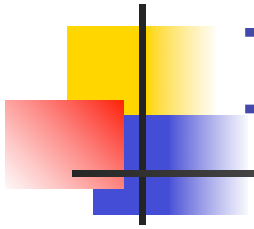
greater than or equal to 70	-	*'A'*
greater than or equal to 50, but less than 70	-	*'B'*
greater than or equal to 30, but less than 50	-	*'C'*
less than 30	-	*'D'*

Where a mark is outside its expected range then a fault message ('FM') is generated. All inputs are passed as integers.



Valid partitions

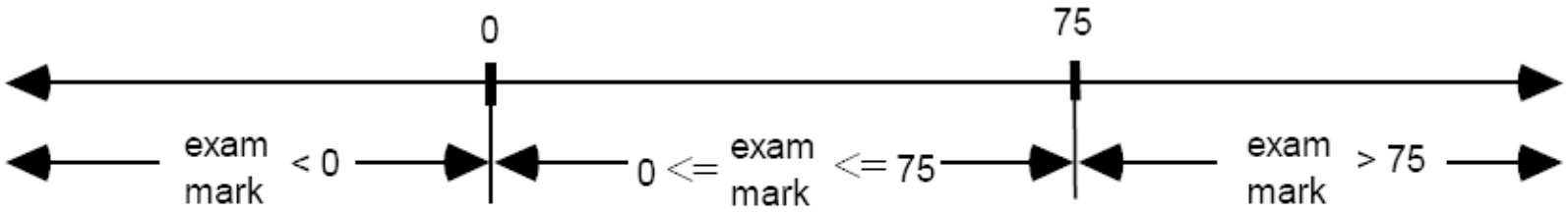
- The valid partitions can be
  - 0<=exam mark <=75
  - 0<=coursework <=25



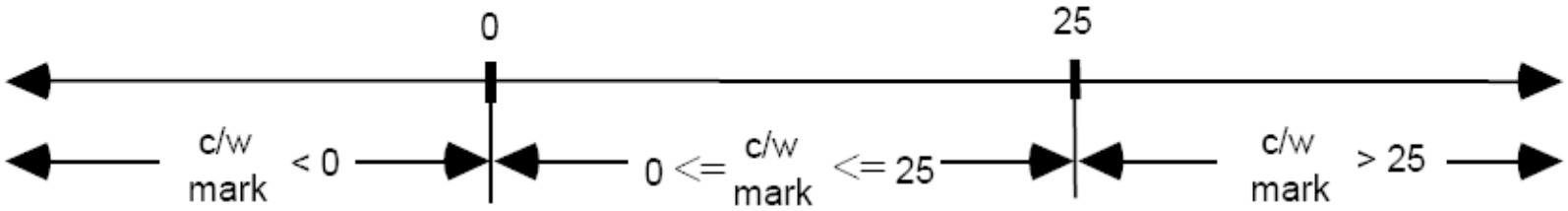
Invalid partitions

- The most obvious partitions are
  - Exam mark > 75
  - Exam mark < 0
  - Coursework mark > 25
  - Coursework nark <0

Exam mark and c/w mark



And for the input, coursework mark, we get:



Less obvious invalid input EP

- invalid INPUT EP should include

exam mark = real number (a number with a fractional part)
exam mark = alphabetic
coursework mark = real number
coursework mark = alphabetic

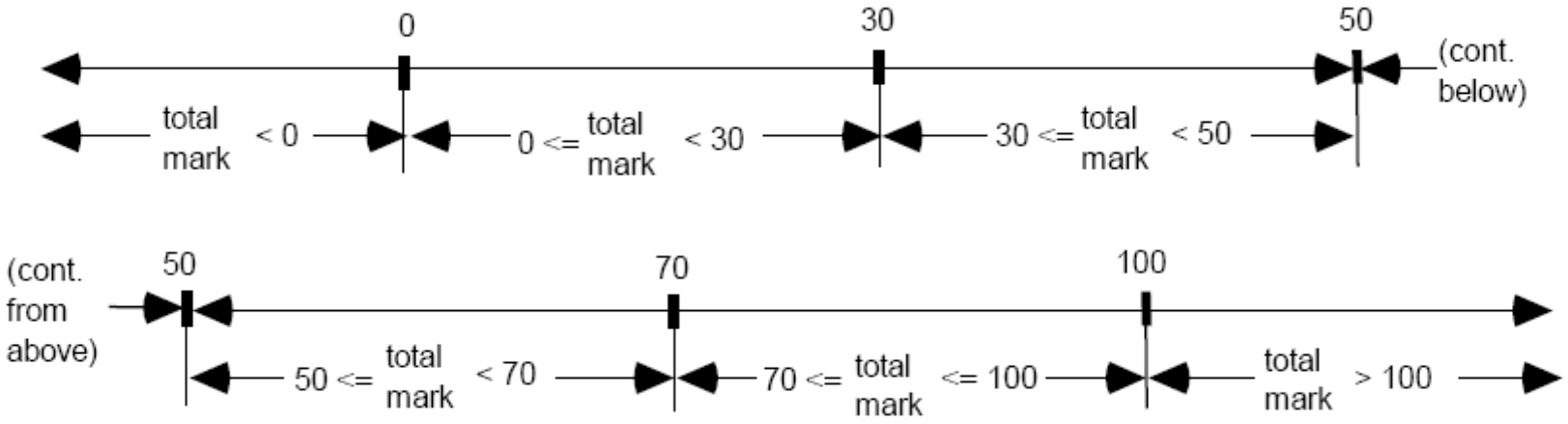
Partitions for the OUTPUTS

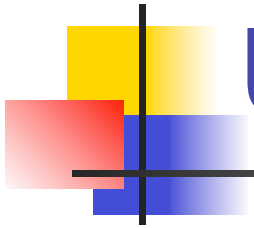
- EP for valid OUTPUTS should include

'A'	is induced by	70 <= total mark <= 100
'B'	is induced by	50 <= total mark < 70
'C'	is induced by	30 <= total mark < 50
'D'	is induced by	0 <= total mark < 30
'Fault Message'	is induced by	total mark > 100
'Fault Message'	is induced by	total mark < 0

The EP and boundaries

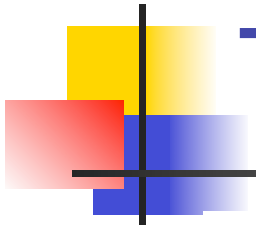
- The EP and boundaries for total mark





Unspecified Outputs

- Three unspecfied Outputs can be identified (very subjective)
  - Output = “E”
  - Output = “A+”
  - Output = “null”



Total PE

0 <= exam mark <= 75
exam mark > 75
exam mark < 0
0 <= coursework mark <= 25
coursework mark > 25
coursework mark < 0
exam mark = real number
exam mark = alphabetic
coursework mark = real number
coursework mark = alphabetic
70 <= total mark <= 100
50 <= total mark < 70
30 <= total mark < 50
0 <= total mark < 30
total mark > 100
total mark < 0
output = 'E'
output = 'A+'
output = 'null'

Test Cases corresponding to PE exam mark (INPUT)

Test Case	1	2	3
Input (exam mark)	44	-10	93
Input (c/w mark)	15	15	15
total mark (as calculated)	59	5	108
Partition tested (of exam mark)	0 <= e <= 75	e < 0	e > 75
Exp. Output	'B'	'FM'	'FM'

Test Case 4-6 (coursework)

Test Case	4	5	6
Input (exam mark)	40	40	40
Input (c/w mark)	8	-15	47
total mark (as calculated)	48	25	87
Partition tested (of c/w mark)	0 <= c <= 25	c < 0	c > 25
Exp. Output	'C'	'FM'	'FM'

test case for Invalid inputs

The test cases corresponding to partitions derived from possible invalid inputs are:

Test Case	7	8	9	10
Input (exam mark)	48.7	q	40	40
Input (c/w mark)	15	15	12.76	g
total mark (as calculated)	63.7	not applicable	52.76	not applicable
Partition tested	exam mark = real number	exam mark = alphabetic	c/w mark = real number	c/w mark = alphabetic
Exp. Output	'FM'	'FM'	'FM'	'FM'

Test cases for invalid outputs: 1

The test cases corresponding to partitions derived from the valid outputs are:

Test Case	11	12	13
Input (exam mark)	-10	12	32
Input (c/w mark)	-10	5	13
total mark (as calculated)	-20	17	45
Partition tested (of total mark)	t < 0	0 <= t < 30	30 <= t < 50
Exp. Output	'FM'	'D'	'C'

Test cases for invalid outputs: 2

Test Case	14	15	16
Input (exam mark)	44	60	80
Input (c/w mark)	22	20	30
total mark (as calculated)	66	80	110
Partition tested (of total mark)	50 <= t < 70	70 <= t <= 100	t > 100
Exp. Output	'B'	'A'	'FM'

Test cases for invalid outputs: 3

The test cases corresponding to partitions derived from the invalid outputs are:

Test Case	17	18	19
Input (exam mark)	-10	100	null
Input (c/w mark)	0	10	null
total mark (as calculated)	-10	110	null+null
Partition tested (output)	'E'	'A+'	'null'
Exp. Output	'FM'	'FM'	'FM'

Minimal Test cases:1

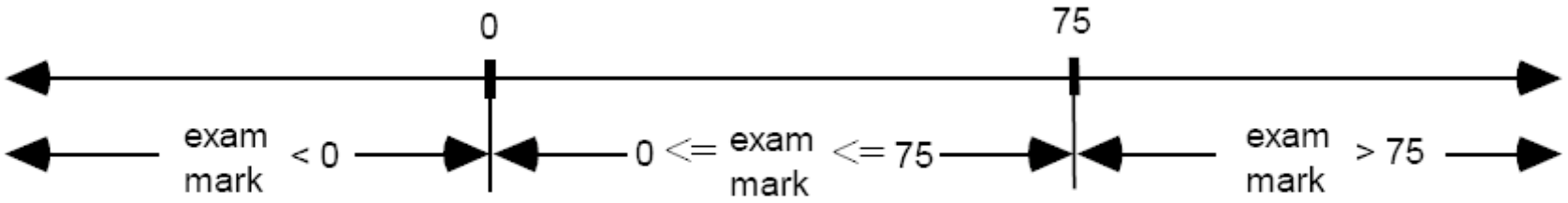
Test Case	1	2	3	4
Input (exam mark)	60	40	25	15
Input (c/w mark)	20	15	10	8
total mark (as calculated)	80	55	35	23
Partition (of exam mark)	0 <= e <= 75	0 <= e <= 75	0 <= e <= 75	0 <= e <= 75
Partition (of c/w mark)	0 <= c <= 25	0 <= c <= 25	0 <= c <= 25	0 <= c <= 25
Partition (of total mark)	70 <= t <= 100	50 <= t < 70	30 <= t < 50	0 <= t < 30
Exp. Output	'A'	'B'	'C'	'D'

Test Case	5	6	7	8
Input (exam mark)	-10	93	60.5	q
Input (c/w mark)	-15	35	20.23	g
total mark (as calculated)	-25	128	80.73	-
Partition (of exam mark)	e < 0	e > 75	e = real number	e = alphabetic
Partition (of c/w mark)	c < 0	c > 25	c = real number	c = alphabetic
Partition (of total mark)	t < 0	t > 100	70 <= t <= 100	-
Exp. Output	'FM'	'FM'	'FM'	'FM'

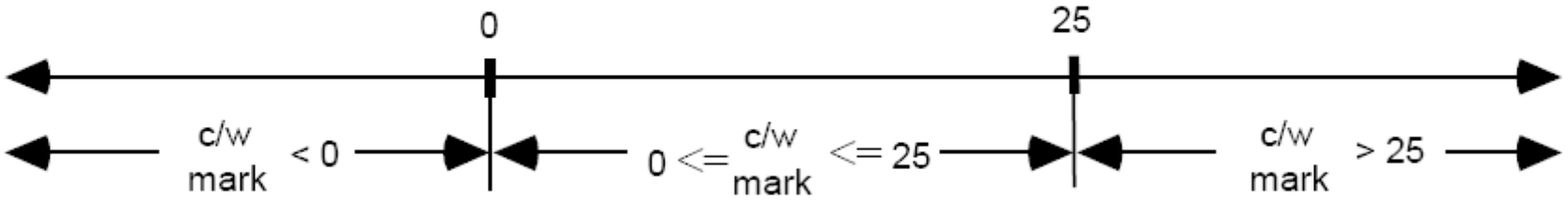
Minimal Test cases:2

Test Case	9	10	11
Input (exam mark)	-10	100	'null'
Input (c/w mark)	0	10	'null'
total mark (as calculated)	-10	110	null+null
Partition (of exam mark)	e < 0	e > 75	-
Partition (of c/w mark)	0 <= c <= 25	0 <= c <= 25	-
Partition (of total mark)	t < 0	t > 100	-
Partition (of output)	'E'	'A+'	'null'
Exp. Output	'FM'	'FM'	'FM'

For Boundary Value analysis (BVA)

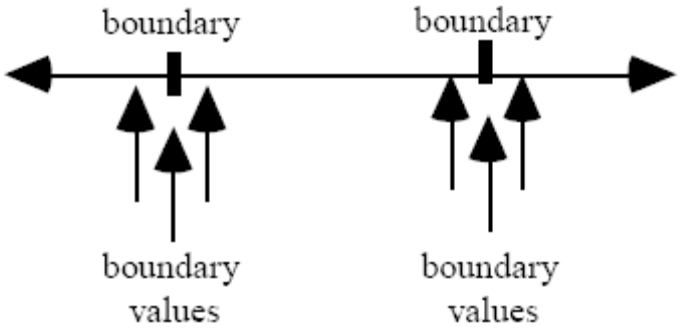


And for the input, coursework mark, we get:



BVA

- For each boundary three values are used
  - One on the boundary
  - One either side of it



Test cases corresponding to BVA (Exam mark)

Thus the six test cases derived from the input exam mark are:

Test Case	1	2	3	4	5	6
Input (exam mark)	-1	0	1	74	75	76
Input (c/w mark)	15	15	15	15	15	15
Boundary tested (exam mark)		0			75	
Exp. Output	'FM'	'D'	'D'	'A'	'A'	'FM'

Note that the input coursework (c/w) mark has been set to an arbitrary valid value of 15.

Test cases corresponding to BVA (Course work)

The test cases derived from the input coursework mark are thus:

Test Case	7	8	9	10	11	12
Input (exam mark)	40	40	40	40	40	40
Input (c/w mark)	-1	0	1	24	25	26
Boundary tested (c/w mark)		0			25	
Exp. Output	'FM'	'C'	'C'	'B'	'B'	'FM'

Note that the input exam mark has been set to an arbitrary valid value of 40.

PE + BVA

'A'	is induced by	70 <= total mark <= 100
'B'	is induced by	50 <= total mark < 70
'C'	is induced by	30 <= total mark < 50
'D'	is induced by	0 <= total mark < 30
'Fault Message'	is induced by	total mark > 100
'Fault Message'	is induced by	total mark < 0

where total mark = exam mark + coursework mark.

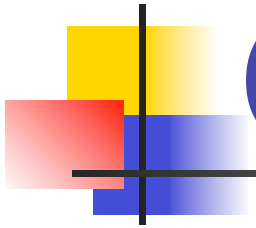
Test case derived from valid outputs:1

Test Case	13	14	15	16	17	18	19	20	21
Input (exam mark)	-1	0	0	29	15	6	24	50	26
Input (c/w mark)	0	0	1	0	15	25	25	0	25
total mark (as calculated)	-1	0	1	29	30	31	49	50	51
Boundary tested (total mark)		0			30			50	
Exp. Output	'FM'	'D'	'D'	'D'	'C'	'C'	'C'	'B'	'B'

Test case derived from valid outputs:2

Test Case	22	23	24	25	26	27
Input (exam mark)	49	45	71	74	75	75
Input (c/w mark)	20	25	0	25	25	26
total mark (as calculated)	69	70	71	99	100	101
Boundary tested (total mark)		70			100	
Exp. Output	'B'	'A'	'A'	'A'	'A'	'FM'

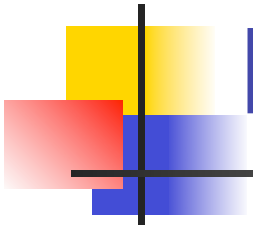
State Transition Testing (revisited)



- States
  - The word “state” is used in much the same way as in “state of the Union” or “state of economy”
  - Refers to a combination of circumstances (or snapshots of program attributes)

Number of states

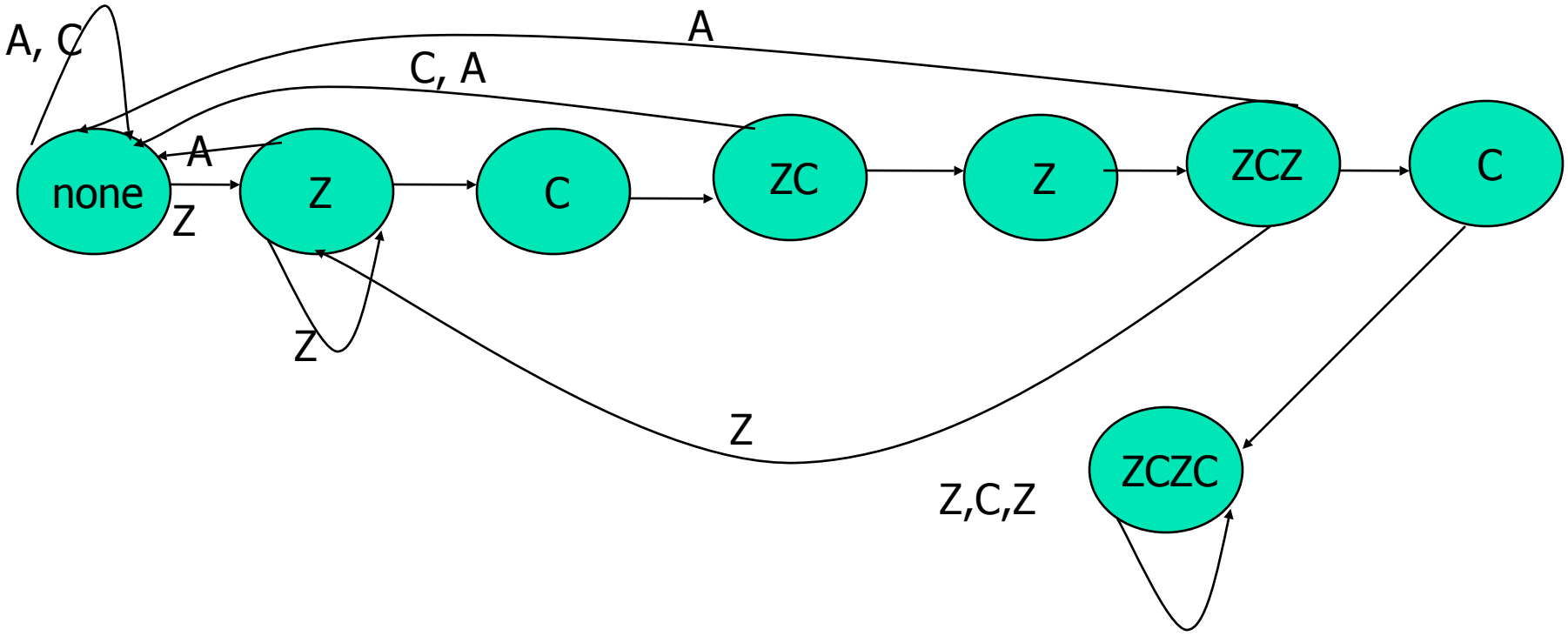
- Suppose the following factors are identified for a car
  - GEAR (R,N, 1,2,3, 4) = 6 values
  - Direction (Forward, reveres, stopped) =3 values
  - Engines Operation (Running, ideal) = 2 values
  - Engine condition (Okay, broken) = 2 values
  - Total number of states
    - 6*3*2*2*=72
  - Impossible states
    - Can broken engine run?
    - A car with a broken transmission won't move

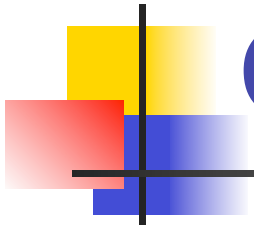


Example of state graph

- A program that detects the char sequence “ZCZC” cab be in the following states
  - Neither ZCZC nor any part of it has been detected
  - Z has been detected
  - ZC has been visited
  - ZCZ has been visited or detected
  - ZCZC has been detected

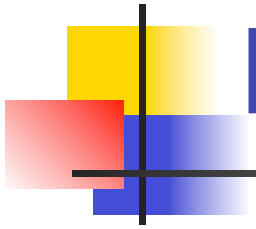
ZCZC sequence detection state graphs





GOOD STATE GRAPHS

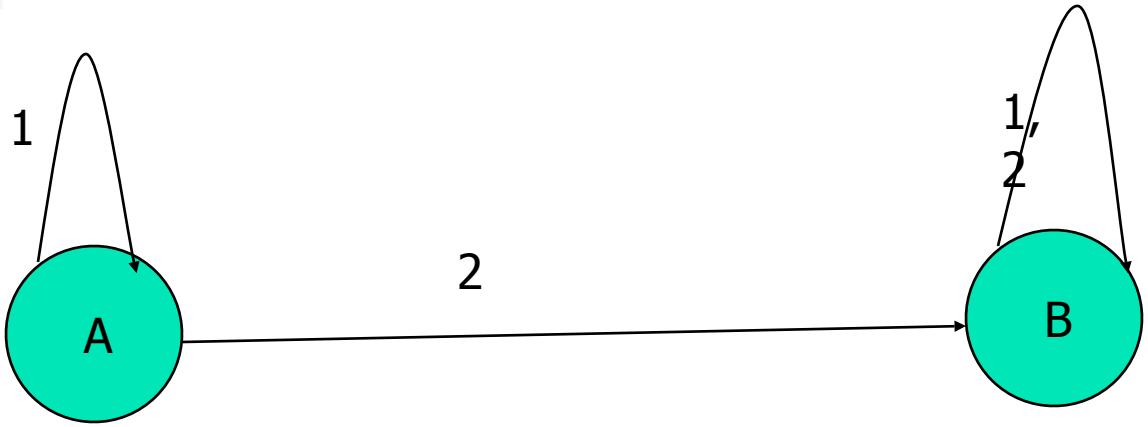
- Some general rules
  1. The total number of states is equal to the product of the possibilities of factors that make up the states
  2. Deterministic behavior
  3. For every transition there is one outcome specified
  4. For every state there is a sequence of inputs that will drive the system back to the same state



More on states

- impossible states
- Equivalent states
  - Two states are equivalent if every sequence of inputs starting from one state produces exactly the same sequence of outputs when stared from the other state
- Unreachable states
  - A state that no input sequence will reach
- Dead states
  - A state which once entered cannot be left

Dead State



State B can never be left (Dead State)

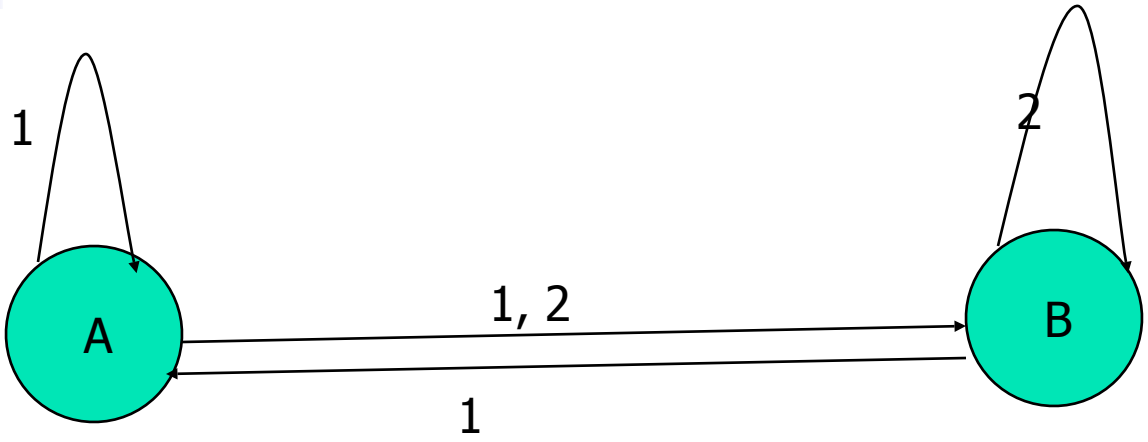
The initial state never be entered gain

Unreachable State

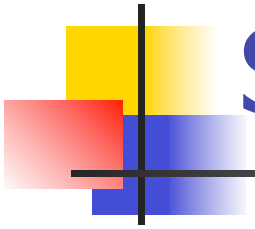
State B can never be left (Dead State)

The initial state never be entered gain

Ambitious state

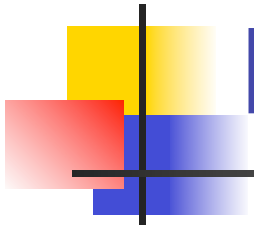


Two transitions are specified for an input of 1 in state A



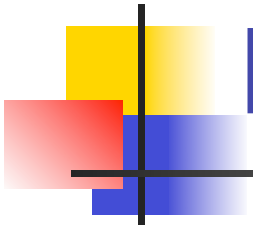
State testing

- The starting point of state testing is
  - Define a set of covering input sequence that get back to the initial state when starting from the initial state
  - For each step in each input sequence, define the expected next state, the expected transition, and the expected outcome



More on state testing

- A set of tests consists of
  - Input sequence
  - Corresponding transitions or next-state names
  - Outcome sequences



References

- 1. G. Myers. The Art of Software Testing, second edition, Wiley 2004
- 2. Standard for Software Component Testing Produced by the British Computer Society, April 2001,
  - www.testingstandards.co.uk