Cause-Effect Graphing

Dr. Durga Prasad Mohapatra
Professor
CSE Department
NIT Rourkela
durga@nitrkl.ac.in

Plan of the Talk

- Introduction to Testing
- Cause Effect Graphing
- Procedure used for the generation of tests
- Basic elements of a cause-effect graph
- Constraints amongst causes
- Constraint amongst effects
- Creating Cause-Effect Graph
- Test generation from a decision table
- Decision Table from cause-effect graph

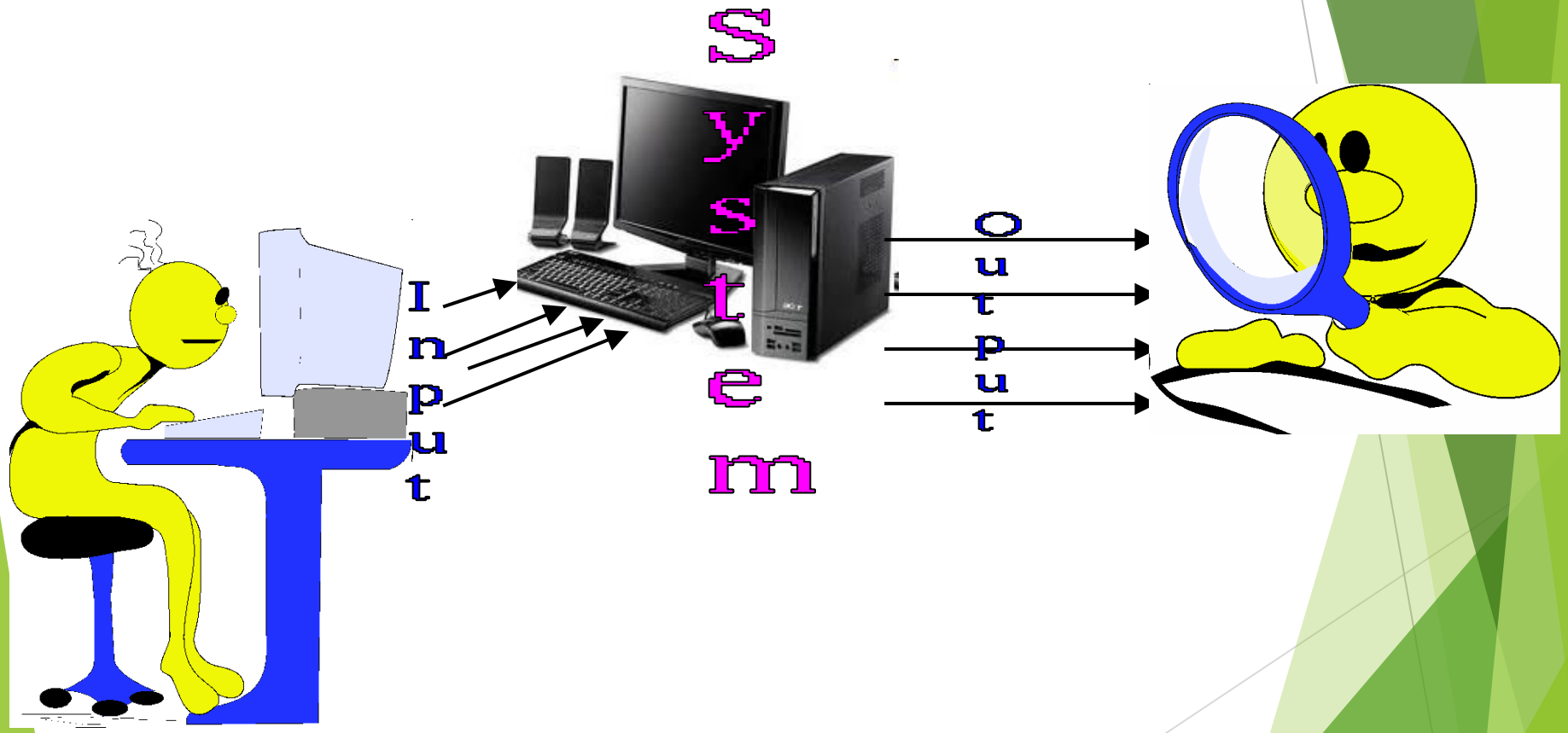
Why Test?

- Ariane 5 rocket self-destructed 37 seconds after launch
- Reason: A control software bug that went undetected
  - Conversion from 64-bit floating point to 16-bit signed integer value had caused an exception
    - The floating point number was larger than 32767
    - Efficiency considerations had led to the disabling of the exception handler.
- Total Cost: over $1 billion

How Do You Test a Program?

- Input test data to the program.
- Observe the output:
  - Check if the program behaved as expected.

How Do You Test a Program?



How Do You Test a Program?

- If the program does not behave as expected:
  - Note the conditions under which it failed.
  - Later debug and correct.

What's So Hard About Testing ?

- Consider **int proc1(int x, int y)**
- Assuming a 64 bit computer
  - Input space = 2^{128}
- Assuming it takes 10secs to key-in an integer pair
  - It would take about a billion years to enter all possible values!
  - Automatic testing has its own problems!

Testing Facts

- Consumes largest effort among all phases
  - Largest manpower among all other development roles
  - Implies more job opportunities
- About 50% development effort
  - But 10% of development time?
  - How?

Testing Facts

- Testing is getting more complex and sophisticated every year.
  - Larger and more complex programs
  - Newer programming paradigms

Overview of Testing Activities

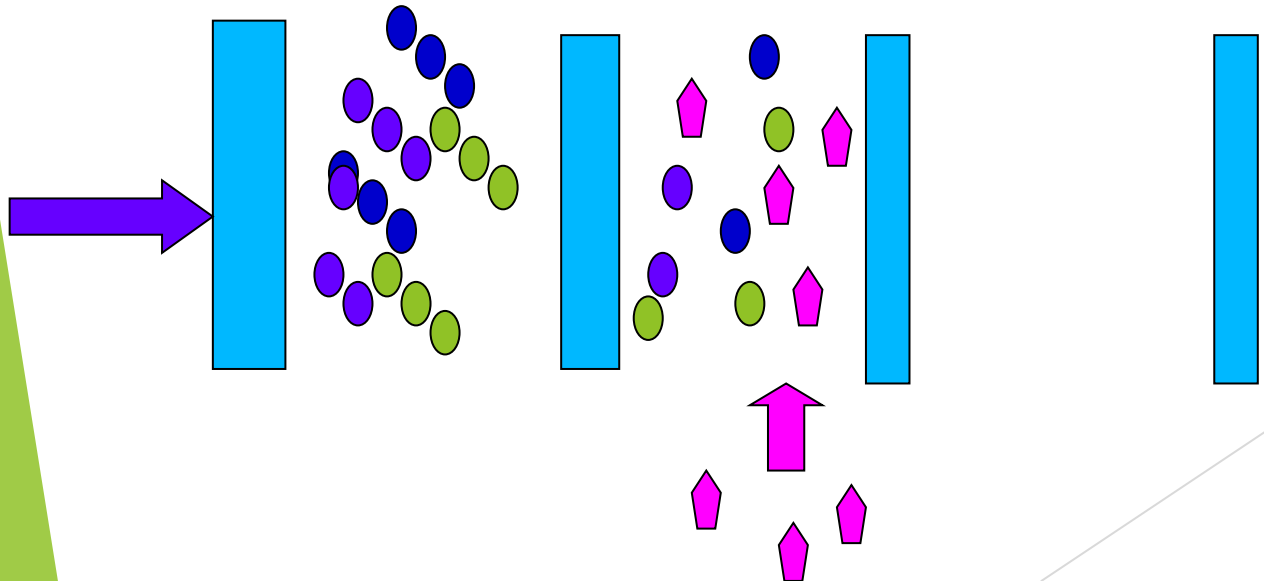
- Test Suite Design
- Run test cases and observe results to detect failures.
- Debug to locate errors
- Correct errors.

Error, Faults, and Failures

- A failure is a manifestation of an error (also defect or bug).
  - Mere presence of an error may not lead to a failure.

Pesticide Effect

- Errors that escape a fault detection technique:
  - Can not be detected by further applications of that technique.



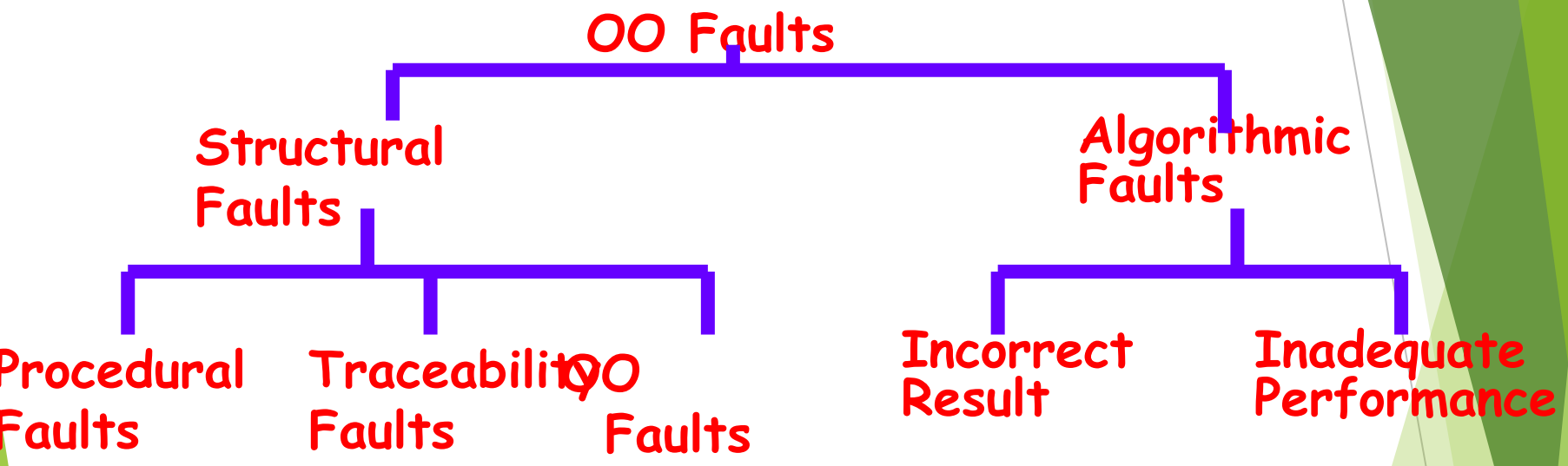
Pesticide Effect

- Assume we use 4 fault detection techniques and 1000 bugs:
  - Each detects only 70% bugs
  - How many bugs would remain
  - $1000*(0.3)^4=81$ bugs

Fault Model

- Types of faults possible in a program.
- Some types can be ruled out
  - Concurrency related-problems in a sequential program

Fault Model of an OO Program



Hardware Fault-Model

- Simple:
  - Stuck-at 0
  - Stuck-at 1
  - Open circuit
  - Short circuit
- Simple ways to test the presence of each
- Hardware testing is fault-based testing

Software Testing

- Each test case typically tries to establish correct working of some functionality
  - Executes (covers) some program elements
  - For restricted types of faults, fault-based testing exists.

Test Cases and Test Suites

- Test a software using a set of carefully designed test cases:
  - The set of all test cases is called the test suite

Test Cases and Test Suites

- A test case is a triplet [I,S,O]
  - I is the data to be input to the system,
  - S is the state of the system at which the data will be input,
  - O is the expected output of the system.

Verification versus Validation

- Verification is the process of determining:
  - Whether output of one phase of development conforms to its previous phase.
- Validation is the process of determining:
  - Whether a fully developed system conforms to its SRS document.

Verification versus Validation

- Verification is concerned with phase containment of errors,
  - Whereas the aim of validation is that the final product be error free.

Design of Test Cases

- Exhaustive testing of any non-trivial system is impractical:
  - Input data domain is extremely large.
- Design an optimal test suite:
  - Of reasonable size and
  - Uncovers as many errors as possible.

Design of Test Cases

- If test cases are selected randomly:
  - Many test cases would not contribute to the significance of the test suite,
  - Would not detect errors not already being detected by other test cases in the suite.
- Number of test cases in a randomly selected test suite:
  - Not an indication of effectiveness of testing.

Design of Test Cases

- Testing a system using a large number of randomly selected test cases:
  - Does not mean that many errors in the system will be uncovered.
- Consider following example:
  - Find the maximum of two integers x and y.

Design of Test Cases

- The code has a simple programming error:
- ```
  if (x>y) max = x;
        else max = x;
  ```
- Test suite {(x=3,y=2);(x=2,y=3)} can detect the error,
- A larger test suite {(x=3,y=2);(x=4,y=3); (x=5,y=1)} does not detect the error.
  ```

Design of Test Cases

- Systematic approaches are required to design an optimal test suite:
  - Each test case in the suite should detect different errors.

Design of Test Cases

- There are essentially three main approaches to design test cases:
  - Black-box approach
  - White-box (or glass-box) approach
  - Grey-box testing

Black-Box Testing

- Test cases are designed using only functional specification of the software:
  - Without any knowledge of the internal structure of the software.
- For this reason, black-box testing is also known as functional testing.

Black Box Testing

- Approaches to design black box test cases
  - Requirements Based Testing
  - Positive and Negative Testing
  - Boundary Value Analysis (BVA)
    - Boundary Value Checking (BVC)
    - Robustness Testing
    - Worst-Case Testing
  - Equivalence Partitioning
  - State Table Based Testing
  - Decision Table Based
  - Cause-Effect Graphing
  - Compatibility Testing
  - User Documentation Testing
  - Domain Testing

Cause-Effect Graphing

- Cause-effect graphing, also known as *dependency modeling*,
  - focuses on modelling dependency relationships amongst
    - program input conditions, known as *causes*, and
    - output conditions, known as *effects*.
- The relationship is expressed visually in terms of a cause-effect graph.
- The graph is a visual representation of a logical relationship amongst inputs and outputs that can be expressed as a Boolean expression.
- The graph allows selection of various combinations of input values as tests.
- The combinatorial explosion in the number of tests is avoided by using certain heuristics during test generation.

Cause-Effect Graphing (Contd..)

- A cause is any condition in the requirements that may effect the program output.
- An effect is the response of the program to some combination of input conditions.
  - For example, it may be
    - An error message displayed on the screen
    - A new window displayed
    - A database updated.
- An effect need not be an "output" visible to the user of the program.
- Instead, it could also be an internal *test point* in the program that can be probed during testing to check if some intermediate result is as expected.
  - For example, the intermediate test point could be at the entrance into a method to indicate that indeed the method has been invoked.

Example

- Consider the requirement "Dispense food only when the DF switch is ON"
  - Cause is "DF switch is ON".
  - Effect is "Dispense food".
- This requirement implies a relationship between the "DF switch is ON" and the effect "Dispense food".
- Other requirements might require additional causes for the occurrence of the "Dispense food" effect.

Cause and Effect Graphs

- Testing would be a lot easier:
  - if we could automatically generate test cases from requirements.
- Work done at IBM:
  - Can requirements specifications be systematically used to design functional test cases?

Cause and Effect Graphs

- Examine the requirements:
  - restate them as logical relation between inputs and outputs.
  - The result is a Boolean graph representing the relationships
    - called a cause-effect graph.

Cause and Effect Graphs

- Convert the graph to a decision table:
  - each column of the decision table corresponds to a test case for functional testing.

Steps to create cause-effect graph

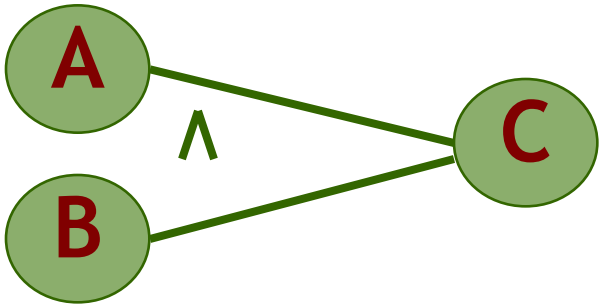
- Study the functional requirements.
- Mark and number all causes and effects.
- Numbered causes and effects:
  - become nodes of the graph.

Steps to create cause-effect graph

- Draw causes on the LHS
- Draw effects on the RHS
- Draw logical relationship between causes and effects
  - as edges in the graph.
- Extra nodes can be added
  - to simplify the graph

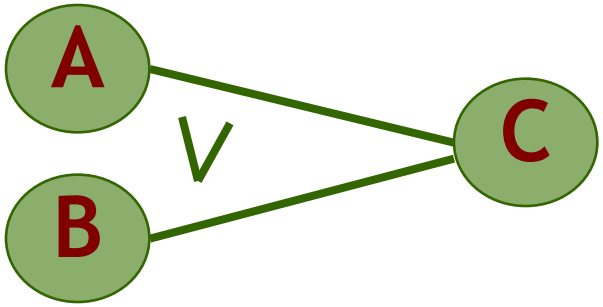
Drawing Cause-Effect Graphs

If A then B

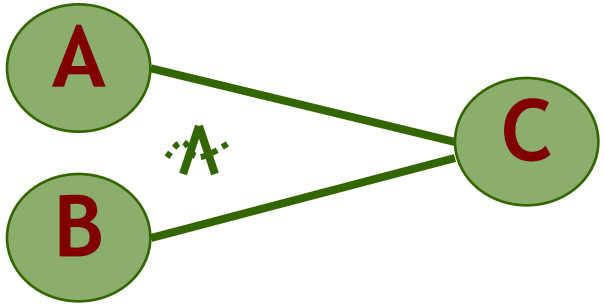


If (A and B)then C

Drawing Cause-Effect Graphs

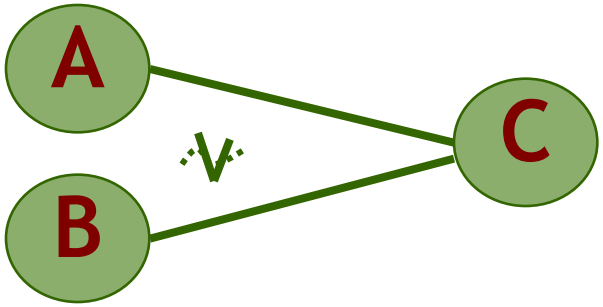


If (A or B)then C



If (not(A and B))then C

Drawing Cause-Effect Graphs



If (not (A or B))then C

If (not A) then B

Cause effect graph- Example

- A water level monitoring system
  - used by an agency involved in flood control.
  - Input: level(a,b)
    - a is the height of water in dam in meters
    - b is the rainfall in the last 24 hours in cms

Cause effect graph- Example

- Processing
  - The function calculates whether the level is safe, too high, or too low.
- Output
  - message on screen
    - level=safe
    - level=high
    - invalid syntax

Cause effect graph- Example

- We can separate the requirements into 5 clauses:
  1. first five letters of the command is “level”
  2. command contains exactly two parameters
     - separated by comma and enclosed in parentheses

Cause effect graph- Example

- Parameters A and B are real numbers:

3
  - such that the water level is calculated to be low

4
  - or safe.

- The parameters A and B are real numbers:

5
  - such that the water level is calculated to be high.

Cause effect graph- Example

10

11

- Command is syntactically valid
- Operands are syntactically valid.

Cause effect graph- Example

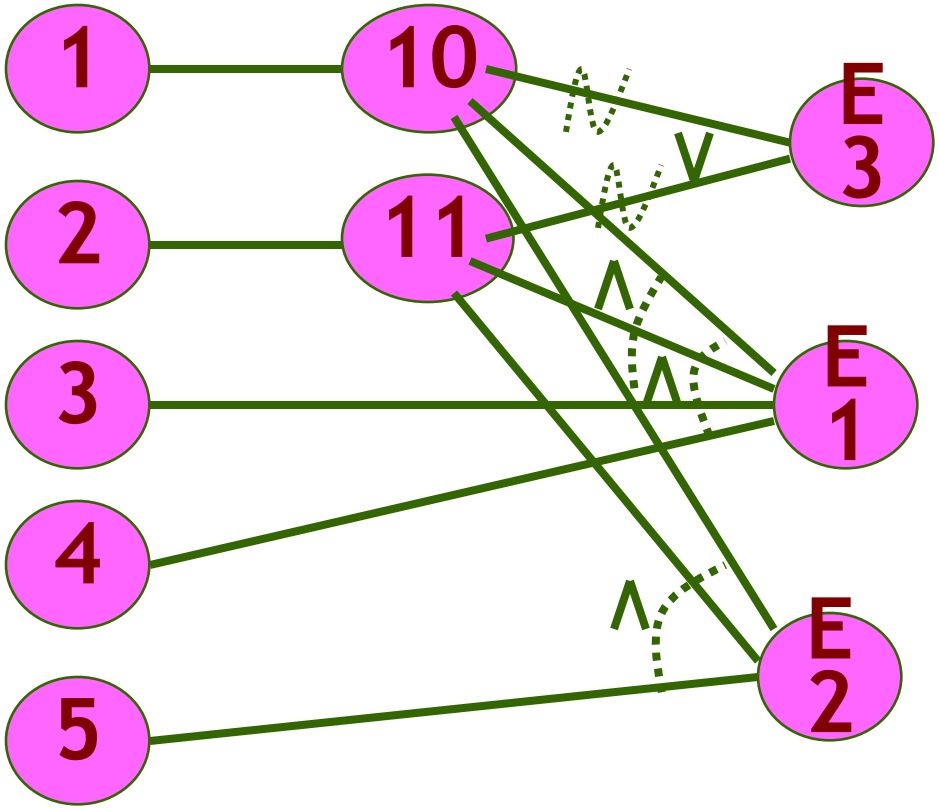
- Three effects
  - level = safe
  - level = high
  - invalid syntax

E1

E2

E3

Cause effect graph- Example



Cause effect graph- Decision table

	Test 1	Test 2	Test 3	Test 4	Test 5
Cause 1	I	I	I	S	I
Cause 2	I	I	I	X	S
Cause 3	I	S	S	X	X
Cause 4	S	I	S	X	X
Cause 5	S	S	I	X	X
Effect 1	P	P	A	A	A
Effect 2	A	A	P	A	A
Effect 3	A	A	A	P	P

I = Invoked

x = don't care

s = supressed

P = present

A = absent

Cause effect graph- Example

- Put a row in the decision table for each cause or effect:
  - in the example, there are five rows for causes and three for effects.

Cause effect graph- Example

- The columns of the decision table correspond to test cases.
- Define the columns by examining each effect:
  - list each combination of causes that can lead to that effect.

Cause effect graph- Example

- We can determine the number of columns of the decision table
  - by examining the lines flowing into the effect nodes of the graph.

Cause effect graph- Example

- Theoretically we could have generated $2^5=32$ test cases.
  - Using cause effect graphing technique reduces that number to 5.

Cause effect graph

- Not practical for systems which:
  - include timing aspects
  - feedback from processes is used for some other processes.

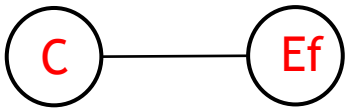
Procedure used for the generation of tests

- Identify causes and effects by reading the requirements. Each cause and effect is assigned a unique identifier. Note that an effect can also be a cause for some other effect.
- Express the relationship between causes and effects using a cause-effect graph.
- Transform the cause-effect graph into a limited entry decision table, hereafter referred to as decision table.
- Generate tests from the decision table.

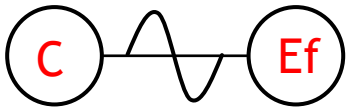
Basic elements of a cause-effect graph

- implication
- not (~)
- and (^)
- or (v)

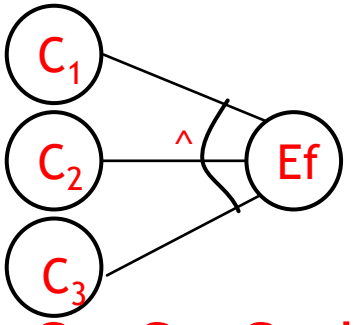
C implies Ef



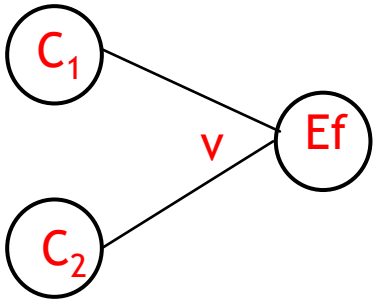
not C implies Ef



Ef when C_1 and C_2 and C_3



Ef when C_1 or C_2



- C, C_1, C_2, C_3 denote causes.
- Ef denotes an effect.

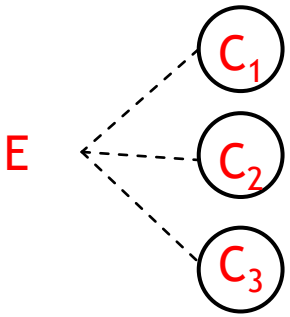
Semantics of basic elements

- *C* implies *Ef*: if(*C*) then *Ef*;
- not *C* implies *Ef*: if(¬*C*) then *Ef*;
- *Ef* when C_1 and C_2 and C_3: if(C_1&&C_2&&C_3) then *Ef*;
- *Ef* when C_1 or C_2: if(C_1||C_2) then *Ef*;

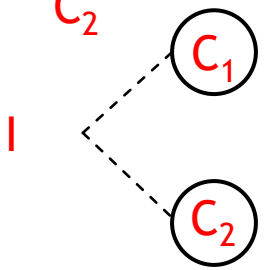
Constraints amongst causes (E,I,O,R)

- Constraints show the relationship between the causes.
- Exclusive (E)
- Inclusive (I)
- Requires (R)
- One and only one (O)

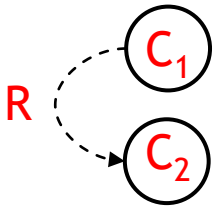
Exclusive: either C_1 or C_2 or C_3



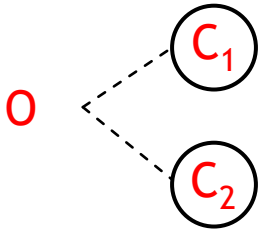
Inclusive: at least C_1 or C_2



C_1 requires C_2



One and only one, of C_1 and C_2



Constraints amongst causes (E,I,O,R)

- Exclusive (E) constraint between three causes C_1, C_2 and C_3 implies that exactly one of C_1, C_2, C_3 can be true.
- Inclusive (I) constraint between two causes C_1 and C_2 implies that at least one of the two must be present.
- Requires (R) constraint between C_1 and C_2 implies that C_1 requires C_2.
- One and only one (O) constraint models the condition that one, and only one, of C_1 and C_2 must hold.

Possible values of causes constrained by E, I, R,O

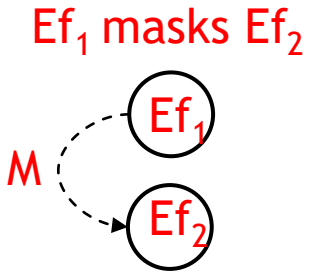
- A 0 or 1 under a cause implies that the corresponding condition is, respectively, false and true.
- The arity of all constraints, except R, is greater than 1, i.e., all except the R constraint can be applied to two or more causes; the R constraint is applied to two causes.
- A condition that is false (true) is said to be in the “0-state” (1 state).
- Similarly, an effect can be “present” (1 state) or “absent” (0 state).

Possible values of causes constrained by E, I, R,O

Constraint	Arity	Possible values		
		C1	C2	C3
$E(C_1,C_2,C_3)$	$n\geq 2$	0	0	0
		1	0	0
		0	1	0
		0	0	1
$I(C_1,C_2)$	$n\geq 2$	1	0	-
		0	1	-
		1	1	-
$R(C_1,C_2)$	$n=2$	1	1	-
		0	0	-
		0	1	-
$O(C_1,C_2,C_3)$	$n\geq 2$	1	0	0
		0	1	0
		0	0	1

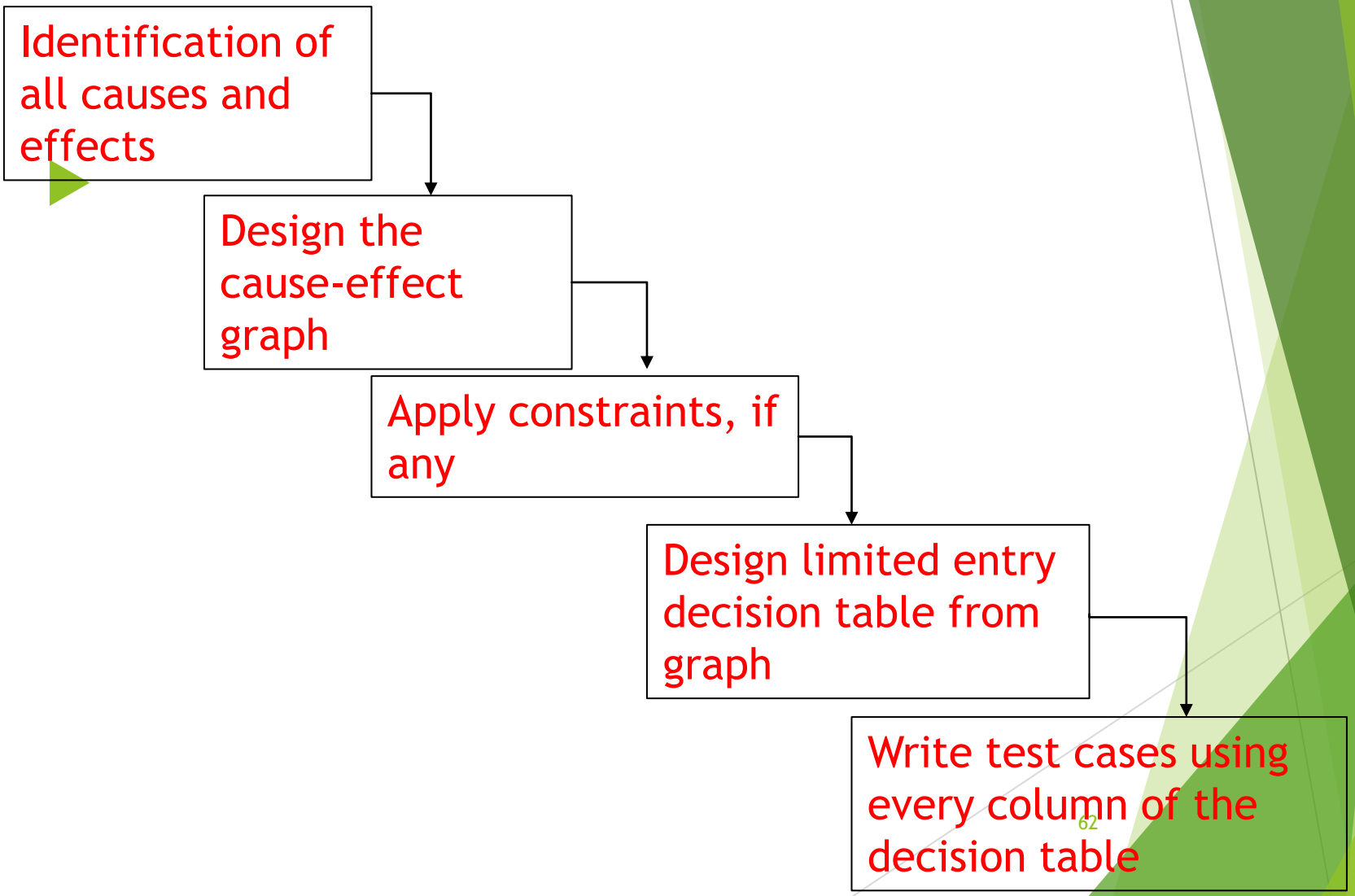
Constraint amongst effects

- Masking (M)



- Masking (M) constraint between two effects Ef_1 and Ef_2 implies that if Ef_1 is present, then Ef_2 is forced to be absent.

Steps for generating test cases using Cause-Effect Graph

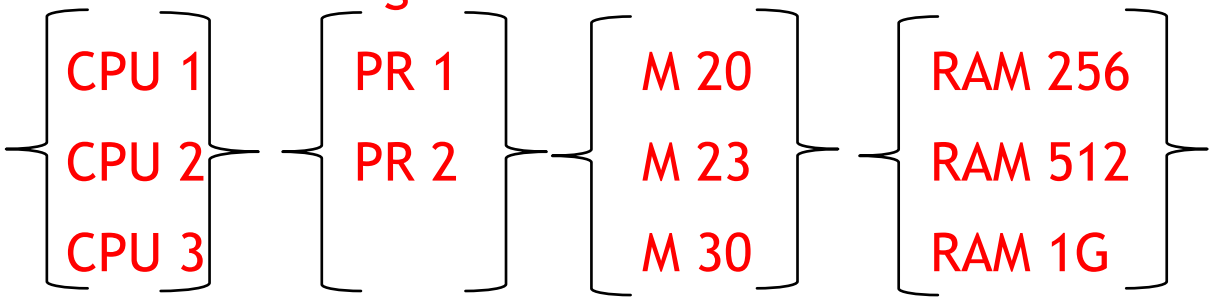


Creating Cause-Effect Graph

- The process of creating a cause-effect graph consists of two major steps.
- The causes and effects are identified by a careful examination of the requirements.
  - This process also exposes the relationships amongst various causes and effects as well as constraints amongst the causes and effects.
  - Each cause and effect is assigned a unique identifier for ease of reference in the cause-effect graph.
- The cause-effect graph is constructed to
  - express the relationships extracted from the requirements.
- When the number of causes and effects is large, say over 100 causes and 45 effects,
  - it is appropriate to use an incremental approach.

Example

- Consider the task of test generation for a GUI based computer purchase system.
- A web-based company is selling computers (CPU), printers (PR), monitors (M), and additional memory (RAM).
- An order configuration consists of one to four items as shown below.



- The GUI consists of four windows for displaying selections from CPU, Printer, Monitor and RAM and one window where any free giveaway items are displayed.
- For each order, the buyer may select from three CPU models, two printer models, and three monitors.
- There are separate windows one each for CPU, printer, and monitor that show the possible selections.
- For simplicity, assume that RAM is available only as an upgrade and that only one unit of each item can be purchased in one order.

Example

- Monitors M 20 and M 23 can be purchased with any CPU or as a stand-alone item.
- M 30 can only be purchased with CPU 3.
- PR 1 is available free with the purchase of CPU 2 or CPU 3.
- Monitors and printers, except for M 30, can also be purchased separately without purchasing any CPU.
- Purchase of CPU 1 gets RAM 256 upgrade.
- Purchase of CPU 2 or CPU 3 gets a RAM 512 upgrade.
- The RAM 1G upgrade and a free PR 2 is available when CPU 3 is purchased with monitor M 30.
- When a buyer selects a CPU, the contents of the printer and monitor windows are updated. Similarly, if a printer or a monitor is selected, contents of various windows are updated.

Example

- Any free printer and RAM available with the CPU selection is displayed in a different window marked “Free”.
- The total price, including taxes, for the items purchased is calculated and displayed in the “Price” window.
- Selection of a monitor could also change the items displayed in the “Free” window.
- Sample configurations and contents of the “Free” window are given below.

Items purchased	“Free” window	Price
CPU 1	RAM 256	$499
CPU 1, PR 1	RAM 256	$628
CPU 2, PR 2, M 23	PR 1, RAM 512	$2257
CPU 3, M 30	PR 2, RAM 1G	$3548

Example

- The first step in creating cause-effect graphing is to read the requirements carefully and make a list of causes and effects.
- A unique identifier, C_1 through C_8, has been assigned to each cause.
- Each cause listed below is a condition that can be true or false.
- C_1: Purchase CPU 1.
- C_2: Purchase CPU 2.
- C_3: Purchase CPU 3.
- C_4: Purchase PR 1.
- C_5: Purchase PR 2.
- C_6: Purchase M 20.
- C_7: Purchase M 23.
- C_8: Purchase M 30.
- For example, C_8 is true if monitor M 30 is purchased.

Example

- Note that while it is possible to order any of the items listed above, the GUI will update the selection available depending on which CPU, or any Other item, is selected.
- For example, if CPU 3 is selected for purchase then monitors M 20 and M 23 will not be available in the monitor selection window.
- Similarly, if monitor M 30 is selected for purchase, then CPU 1 and CPU 2 will not be available in the CPU window.
- Next, we identify the effects.
- In this example, the application software calculates and displays the list of items available free with the purchase and the total price.
- Hence, the effect is in terms of the contents of the "Free" and "Price" windows.

Example

- Calculation of the total purchase price depends on the items purchased and the unit price of each item.
- The unit price is obtained by the application from a price database.
- The price calculation and display is a cause that creates the effect of displaying the total price.
- For simplicity, we ignore the price related cause and effect.
- The set of effects in terms of the contents of the “Free” display window are listed below.

  Ef_1: RAM 256

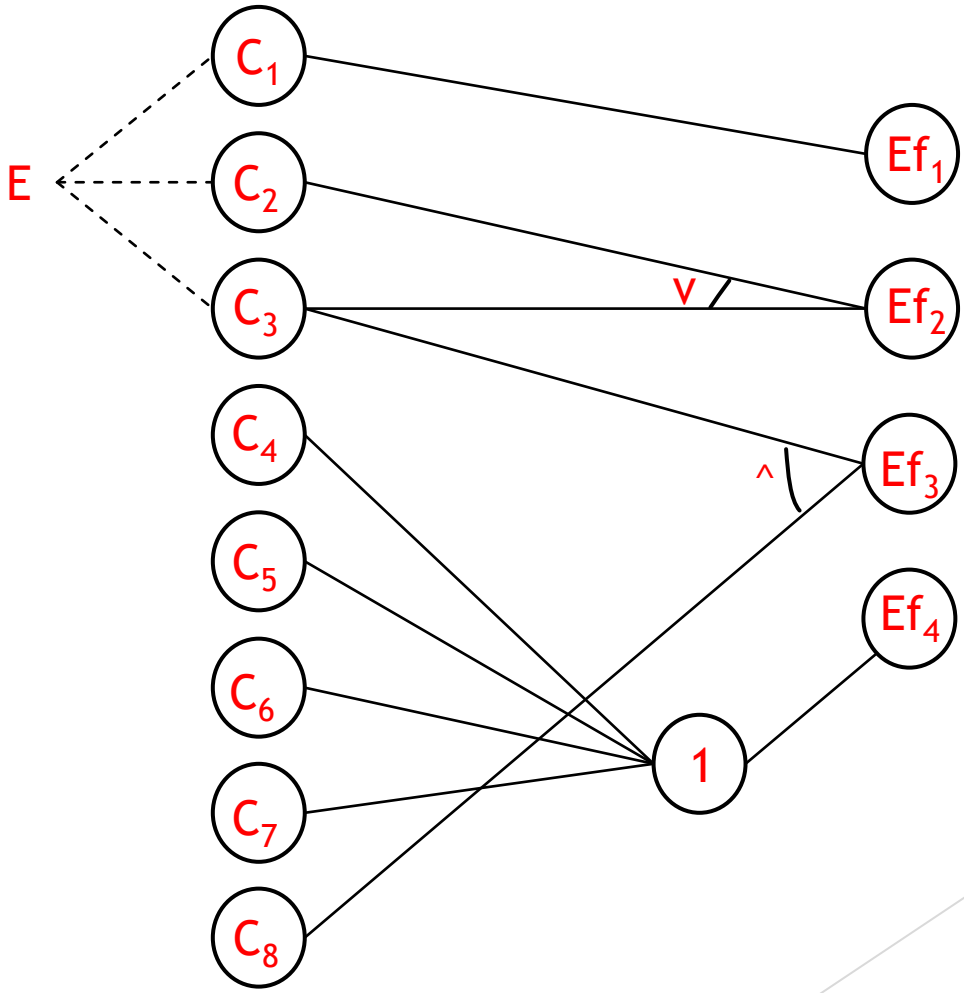
  Ef_2: RAM 512 and PR 1.

  Ef_3: RAM 1G and PR 2.

  Ef_4: No giveaway with this item.

Example

- Following shows the complete graph that expresses the relationships between C_1 through C_8 and effects Ef_1 through Ef_4.



Example

- From the above cause-effect graph, followings can be inferred.
  - C_1, C_2 and C_3 are constrained using the E (exclusive) relationship.
  - This expresses the requirement that only one CPU can be purchased in one order.
  - Similarly, C_3 and C_8 are related via the R (requires) constraint to express the requirement that monitor M 30 can only be purchased with CPU3.
  - Relationships amongst causes and effects are expressed using the basic elements.
- There is an intermediate node labelled 1 in the graph.
- Such intermediate nodes are often useful when an effect depends on conditions combined using more than one operator, for example, $(C_1 \wedge C_2) \vee C_3$.
- Also it can be noted that
  - Purchase of printers and monitors without any CPU leads to no free item (Ef_4).

Example

- The relationships between effects and causes shown in the graph can be expressed in terms of Boolean expressions as follows:

$Ef_1 = C_1$

$Ef_2 = C_2 \vee C_3$

$Ef_3 = C_3 \wedge C_8$

$Ef_4 = C_4 \wedge C_5 \wedge C_6 \wedge C_7$

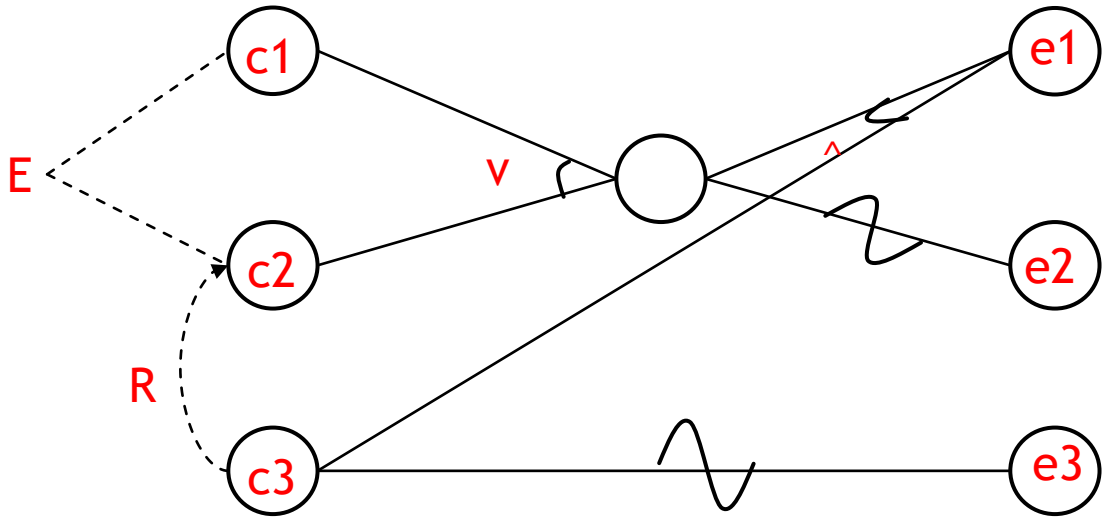
Another example

- Consider the example of keeping the record of marital status and number of children of a citizen.
- The value of marital status must be `U' or `M'.
- The value of the number of children must be digit or null in case a citizen is unmarried.
- If the information entered by the user is correct then an update is made.
- If the value of marital status of the citizen is incorrect, then the error message 1 is issued.
- Similarly, if the value of the number of children is incorrect, then the error message 2 is issued.

Answer

- Causes are
  - c1: marital status is U
  - c2: marital status is M
  - c3: number of children is a digit
- Effects are
  - e1: updation made
  - e2: error message 1 is issued
  - e3: error message 2 is issued

Answer



- There are two constraints
  - Exclusive (between c1 and c2) and
  - Requires (between c3 and c2)
- Causes c1 and c2 cannot occur simultaneously.
- For cause c3 to be true, cause c2 has to be true.

Heuristics to avoid combinatorial explosion

- While tracing back through a cause-effect graph,
  - we generate combinations of causes that set an intermediate node, or an effect, to a 0 or 1 state.
- Doing so in a brute force manner could lead to a large number of combinations.
- In the worst case, if *n* causes are related to an effect *e*, then the maximum number of combinations that bring *e* to a 1-state is 2^n.
- As tests are derived from the combinations of causes,
  - large values of *n* could lead to an exorbitantly large number of tests.
- We avoid such a combinatorial explosion by using simple heuristics related to the "AND" (^) and "OR" (v) nodes.
- Heuristics are based on the assumption that
  - certain types of errors are less likely to occur than others.

Heuristics to avoid combinatorial explosion

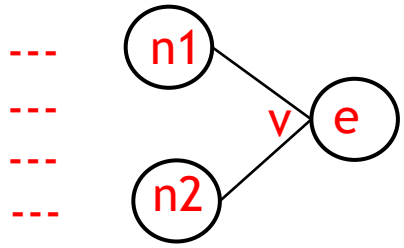
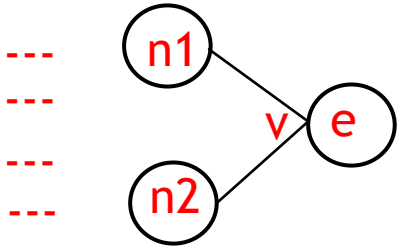
- Thus, while applying the heuristics to generate test
  - inputs will likely lead to a significant reduction in the number of tests generated,
  - it may also discard tests that would have revealed a program error.
- Hence, one must apply the heuristics with care and only when
  - the number of tests generated without their application is too large to be useful in practice.

Heuristics used during the generation of input combinations from a cause-effect graph

- The heuristics are labelled H_1 through H_4.
- The leftmost column shows the node type in the cause-effect graph.
- The center column is the desired state of the dependent node.
- The rightmost column is the heuristics for generating combinations of inputs to the nodes that effect the dependent node *e*.
- For simplicity, only two nodes n_1 and n_2 and the corresponding effect *e* are shown.

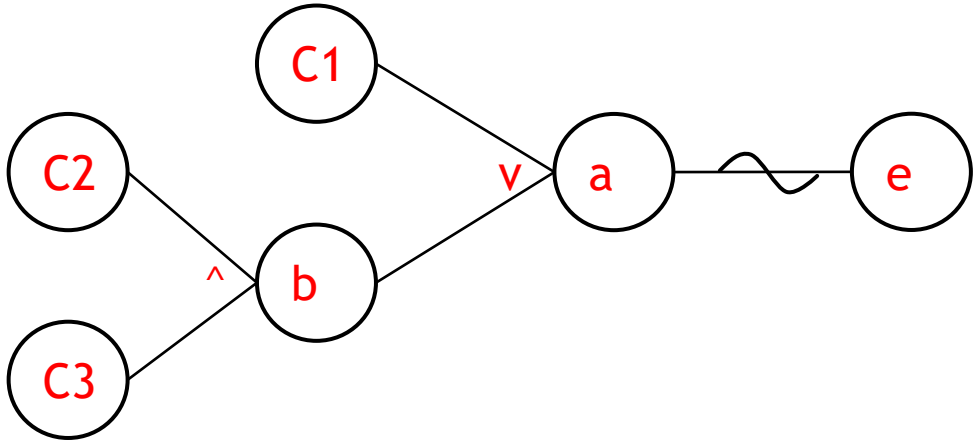
Heuristics used during the generation of input combinations from a cause-effect graph

Table 1

Node type	Desired state of *e*	Input combinations
	0	**H1:** Enumerate all combinations of inputs to *n1* and *n2* such that *n1*=*n2*=0.
	1	**H2:** Enumerate all combinations of inputs to *n1* and *n2* other than those for which *n1*=*n2*=0.
	0	**H3:** Enumerate all combinations of input to n_1 and n_2 such that each of the possible combinations of n_1 and n_2 appears exactly once and n_1=n_2=1 does not appear. Note that for two nodes, there are three such combinations: (0,0), (0,1) and (1,0). In general, for *k* nodes combined using the logical and operator to form *e*, there are 2^k-1 such combinations.
	1	**H4:** Enumerate all combinations of inputs to *n1* and *n2* such that *n1*=*n2*=1.

Example

- Consider the following cause-effect graph.
- We have at least two choices while tracing backwards to derive the necessary combinations.
  - Derive all combinations first and then apply the heuristics.
  - Derive combinations while applying the heuristics.



Example

- Suppose we require node *e* to be 1.
- Tracing backwards, this requirement implies that node *a* must be a 0 (zero).
- If we trace backwards further, without applying any heuristic, we obtain the following seven combinations of causes that bring *e* to 1 state.
- The last column lists the inputs to node *a*.
- The combinations that correspond to the inputs to node *a* listed in the rightmost column are separated by horizontal lines.

Example: Generating tests using heuristics

- First we note that node *a* matches the OR-node shown in the top half of Table 1.
- As we want the state of node *a* to be 0, heuristic H1 applies in this situation.
- H1 asks to enumerate all combinations of inputs to node *a* such that C1 and node *b* are 0.
- (0,0) is the only such combination and is listed in the last column of the following table.

	C1	C2	C3	Inputs to node *a*
1	0	0	0	C1=0, b=0
2	0	0	1	
3	0	1	0	
4	0	1	1	C1=0, b=1
5	1	0	0	C1=1, b=0
6	1	0	1	
7	1	1	0	

Example: Generating tests using heuristics

- Let us begin with (0,0)
- No heuristic applies to C1 as it has no preceding nodes.
- Node *b* is an AND-node as shown in the bottom half of Table.
- We want node *b* to be 0 and therefore H3 applies.
- In accordance with H3 we generate three combinations of inputs to node b: (0,0), (0,1) and (1,0).
- Notice that combination (1,1) is forbidden.
- Joining these combinations of C2 and C3 with C1=0,
  - we obtain the first three combinations listed in the preceding table.

Example: Generating tests using heuristics

- Though not required here, suppose that we were to consider the combination C1=0,*b*=1.
- Heuristic H4 applies in this situation.
- As both C2 and C3 are causes with no preceding nodes,
  - the only combination we obtain now is (1,1).
- Combining this with C1=0 we obtain sequence 4 listed in the preceding table.

Example: Generating tests using heuristics

- We have completed derivation of combinations using the heuristics listed in Table 1.
- Note that the combinations listed above for C1=1,*b*=0 are not required.
- Thus, we have obtained only three combinations instead of the seven enumerated earlier.
- The reduced set of combinations is listed below.

	C1	C2	C3	Inputs to node *a*
1	0	0	0	C1=0, b=0
2	0	0	1	
3	0	1	0	

Example: Generating tests using heuristics

- Let us examine the rationale underlying the various heuristics for reducing the number of combinations.
- Heuristics H1 does not save us on any combinations.
- The only way an OR-node can cause its effect *e* to be 0 is for all its inputs to be 0.
- H1 suggests that we enumerate all such combinations.
- Heuristics H2 suggests we use all combinations that cause e to be 1 except those that cause n1=n2=0.
- To understand the rationale underlying H2 consider a program required to generate an error message when condition c1 or c2 is true.
- A correct implementation of this requirement is given below.

```
if(c1 v c2)printf("Error");
```

Example: Generating tests using heuristics

- Now consider the following erroneous implementation of the same requirement.

  if(c1 v ¬c2)printf("Error");

- A test that sets both c1 and c2 true
  - will not be able to detect an error in the implementation above
    - if short circuit evaluation is used for Boolean expressions.
- However, a test that sets c1=0 and c2=1 will be able to detect this error.
- Hence H2 saves us from generating all input combinations that generate the pair (1,1) entering an effect in an OR-node.

Example: Generating tests using heuristics

- Heuristics H3 saves us from repeating the combinations of n1 and n2.
- Once again this could save us a lot of tests.
- The assumption here is that
  - any error in the implementation of *e* will be detected by
  - tests that cover different combinations of n1 and n2.
- Thus, there is no need to have two or more tests that
  - contain the same combination of n1 and n2.

Example: Generating tests using heuristics

- Lastly, H4 for the AND-node is analogous to H1 for the OR-node.
- The only way an AND-node can cause its effect *e* to be 1
  - is for all its inputs to be 1.
- H4 suggests that we enumerate all such combinations.
- We stress, once again, that
  - while the heuristics will likely reduce the set of tests generated using cause-effect graphing,
    - they might also lead to useful tests being discarded.
- Of course, in general and prior to the start of testing,
  - it is almost impossible to know which of the test cases discarded will be useless and which ones useful.

Decision Table from cause-effect graph

- Each column of the decision table represents a combination of input values, and hence a test.
- There is one row for each condition and effect.
- Thus the table decision table can be viewed as an N X M matrix with
  - N being the sum of the number of conditions and effects and
  - M the number of tests.
- Each entry in the decision table is a 0 or 1
  - depending on whether or not the corresponding condition is false or true, respectively.
- For a row corresponding to an effect, an entry is 0 or 1
  - if the effect is not present or present, respectively.

Procedure for generating a decision table from a cause-effect graph

- **Input:** A cause-effect graph containing causes C_1, C_2, …, C_p and effects Ef_1, Ef_2, …, Ef_q.
- **Output:** A decision table DT containing $N=p+q$ rows and M columns, where M depends on the relationship between the causes and effects as captured in the cause-effect graph.
- **Procedure:** CEGDT
- /*
- i is the index of the next effect to be considered.
- *next_dt_col* is the next empty column in the decision table.
- V_k: a vector of size $p+q$ containing 1's and 0's. V_j, $1 \le j \le p$, indicates the state of condition C_j and V_l, $p < l \le p+q$, indicates the presence or absence of effect Ef_{l-p}.
- */

Procedure for generating a decision table from a cause-effect graph

- **Step 1** *Initialize DT to an empty decision table.*

  next_dt_col=1.

- **Step 2** Execute the following steps for *i*=1 to *q*.

  2.1 *Select the next effect to be processed.*

  Let $e=Ef_i$.

  2.2 *Find combinations of conditions that cause e to be present.*

  Assume that *e* is present. Starting at *e*, trace the cause-effect graph backwards and determine the combinations of conditions C_1, C_2, …, C_p that lead to e being present.

  Let V_1, V_2, …, V_{mi} be the combinations of causes that lead to *e* being present, i.e. in 1 state, and hence $m_i \geq 1$. Set $V_k(l), p<l\leq p+q$ to 0 or 1 depending on whether effect Ef_{l-p} is present or not for the combination of all conditions in V_k.

Procedure for generating a decision table from a cause-effect graph

2.3 *Update the decision table.*

Add V_1, V_2, …, V_{mi} to the decision table as successive columns starting at *next_dt_col*.

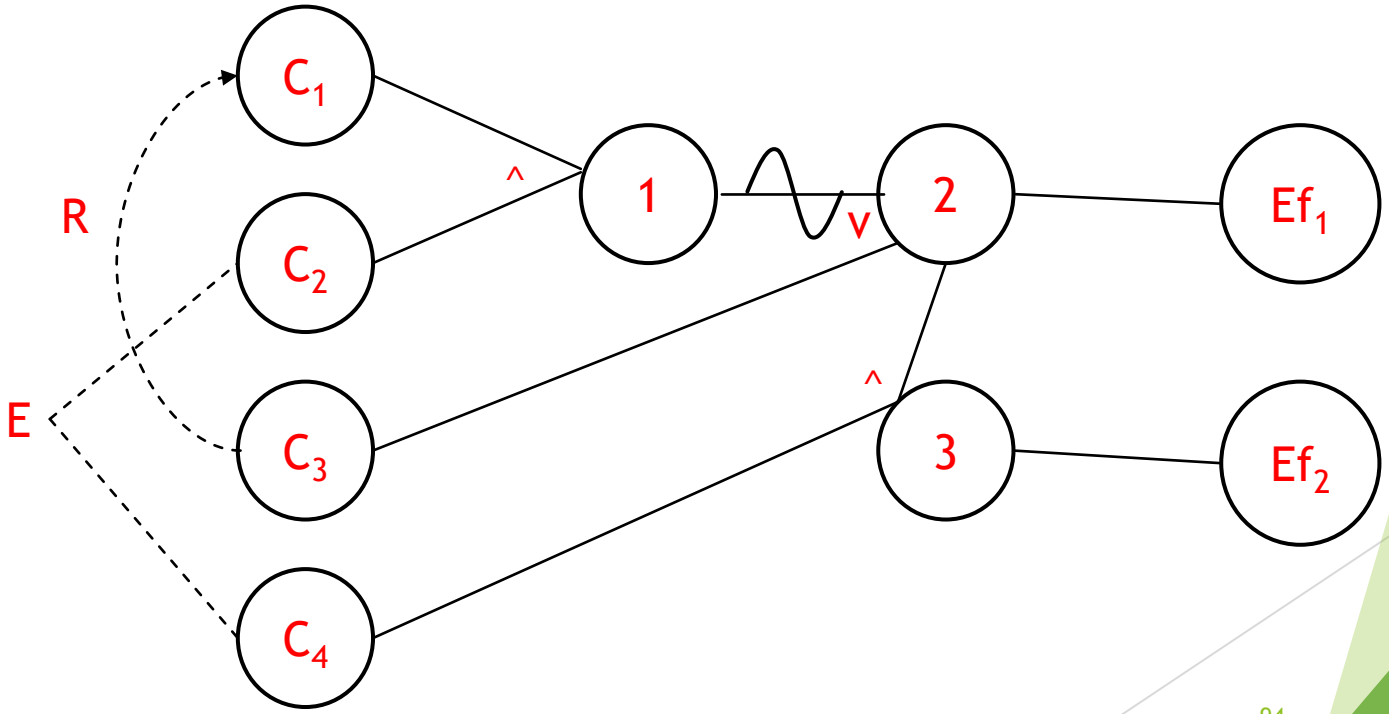
2.4 *Update the next available column in the decision table.*

next_dt_col=next_dt_col+m_i. At the end of this procedure, *next_dt_col*-1 is the number of tests generated.

End of Procedure CEGDT

Example

- Consider the cause effect graph shown below.
- It shows four causes labelled C_1, C_2, C_3 and C_4 and two effects labeled Ef_1 and Ef_2.
- There are three intermediate nodes labeled 1, 2 and 3.



Example

- Step 1: Set *next_dt_col*=1 to initialize the decision table to empty.
- Next, *i*=1 and in accordance with Step 2.2, $e=Ef_1$.
- In accordance with Step 2.2, trace backwards from *e* to determine combinations that will cause *e* to be present.
- *e* must be present when node 2 is in 1-state.
- Moving backwards from node 2 in the cause-effect graph, any of the following three combinations of states of nodes 1 and C_3 will lead to *e* being present: (0,1), (1,1) and (0,0).
- Node 1 is also an internal node and hence move further back to obtain the values of C_1 and C_2 that effect node 1.
- Combinations of C_1 and C_2 that brings node 1 to the 1-state is (1,1) and combinations that bring it to 0-state are (1,0),(0,1) and (0,0).

Example

- Combining this information with that derived earlier for nodes 1 and C_3, we obtain the following seven combinations of C_1, C_2 and C_3 that cause e to be present.

1	0	1
0	1	1
0	0	1
1	1	1
1	0	0
0	1	0
0	0	0

- Next, C_3 requires C_1 which implies that C_1 must be in 1-state for C_3 to be in 1-state.
- This constraint makes infeasible the second and third combinations above.

Example

- In the end, we obtain the following five combinations of the four causes that lead to *e* being present.

1	0	1
1	1	1
1	0	0
0	1	0
0	0	0

- Setting C_4 to 0 and appending the values of Ef_1 and Ef_2, we obtain the following five vectors.

V_1	1	0	1	0	1	0
V_2	1	1	1	0	1	0
V_3	1	0	0	0	1	0
V_4	0	1	0	0	1	0
V_5	0	0	0	0	1	0

Example

- Note that m_1=5 in Step 2.
- This completes the application of Step 2.2.
- The five vectors are transposed and added to the decision table starting at column *next_dt_col* which is 1.
- The decision table at Step 2.3 follows

	1	2	3	4	5
C_1	1	1	1	0	0
C_2	0	1	0	1	0
C_3	1	1	0	0	0
C_4	0	0	0	0	0
Ef_1	1	1	1	1	1
Ef_2	0	0	0	0	0

- We update *next_dt_col* to 6, increment *i* to 2 and get back to Step 2.1.

Example

- We now have $e=Ef_2$.
- Tracing backwards, we find that for e to be present, node 3 must be in the 1-state.
- This is possible with only one combination of node 2 and C_4, which is (1,1).
- Earlier we derived the combinations of C_1, C_2 and C_3 that lead node 2 into the 1-state.
- Combining these with the value of C_4, we arrive at the following combination of causes that lead to the presence of Ef_2.

1	0	1	1
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	1

Example

- From the cause-effect graph, note that C_2 and C_4 cannot be present simultaneously.
- Hence, we discard the second and fourth combinations from the list above and obtain the following three feasible combinations.

	1	0	1	1
	1	0	0	1
	0	0	0	1

- Appending the corresponding values of Ef_1 and Ef_2 to each of the above combinations, we obtain the following three vectors.

V_1	1	0	1	1	1	1
V_2	1	0	0	1	1	1
V_3	0	0	0	1	1	1

Example

- Transposing the vectors listed above and appending them as three columns to the existing decision table, we obtain the following

	1	2	3	4	5	6	7	8
C_1	1	1	1	0	0	1	1	0
C_2	0	1	0	1	0	0	0	0
C_3	1	1	0	0	0	1	0	0
C_4	0	0	0	0	0	1	1	1
Ef_1	1	1	1	1	1	1	1	1
Ef_2	0	0	0	0	0	1	1	1

- Next, we update *next_dt_col* to 9.
- Of course, doing so is useless as the loop set up in Step 2 is now terminated.
- The decision table listed above is the output.

Test generation from a decision table

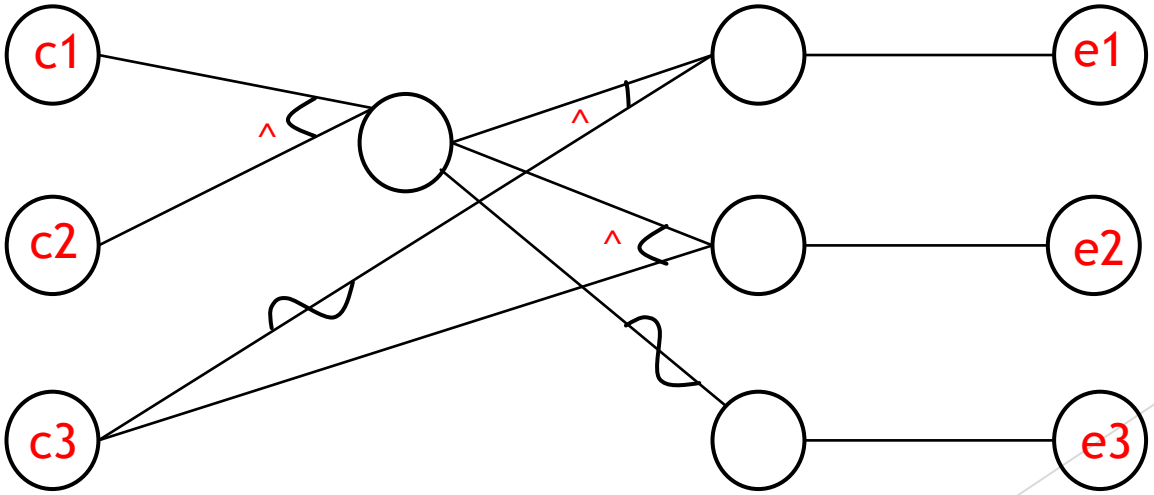
- Test generation from a decision table is relatively forward.
- Each column in the decision table generates at least one test input.
- Note that each combination might be able to generate more than one test when a condition in the cause-effect graph can be satisfied in more than one way.
- For example, consider the following cause:
- *C*: $x<99$
- The condition above can be satisfied by many values such as $x=1$ and $x=49$.
- Also, *C* can be made false by many values of *x* such as $x=100$ and $x=999$.
- Thus, one might have a choice of values of input variables while generating tests using columns from a decision table

Example

- A tourist of age greater than 21 years and having a clean driving record is supplied a rental car.
- A premium amount is also charged if the tourist is on business,
- Otherwise, it is not charged.
- If the tourist is less than 21 year old, or does not have a clean driving record,
  - The system will display the following message: “Car cannot be supplied”.

Answer

- Causes are
  - c1: Age is over 21
  - c2: Driving record is clean
  - c3: Tourist is on business
- Effects are
  - e1: Supply a rental car without premium charge
  - e2: Supply a rental car with premium charge
  - e3: Car cannot be supplied



Decision Table and Test Cases

	1	2	3	4
c1: Over 21?	F	T	T	T
c2: Driving record clean?	-	F	T	T
c3: On business?	-	-	F	T
e1: Supply a rental car without premium charge			X	
e2: Supply a rental car with premium charge				X
e3: Car cannot be supplied	X	X		

Test Case	Age	Driving_record_clean	On_business	Expected Output
1	20	Yes	Yes	Car cannot be supplied
2	26	No	Yes	Car cannot be supplied
3	62	Yes	No	Supply a rental car without premium charge
4	62	Yes	Yes	Supply a rental car with premium charge

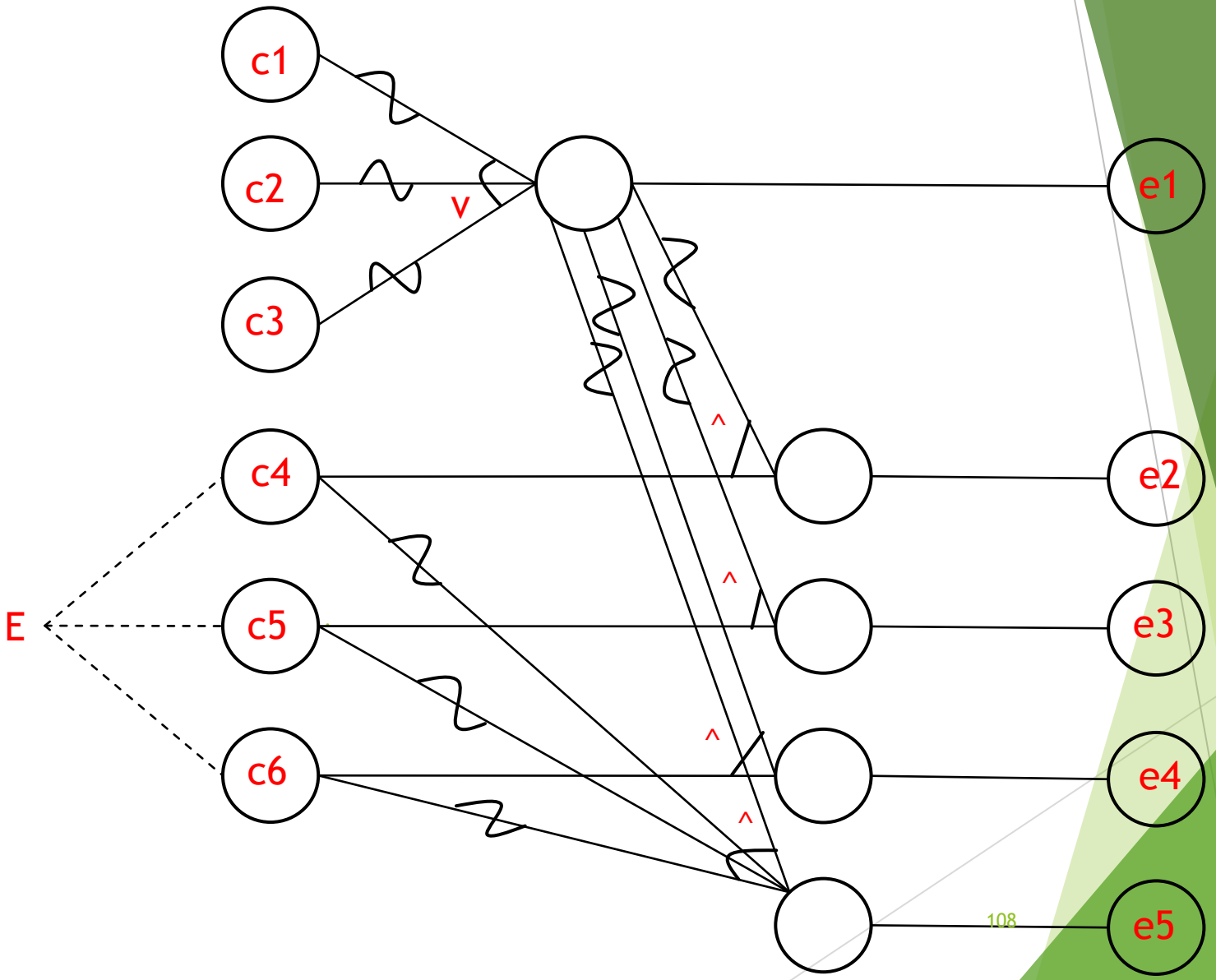
Example 2: Triangle Classification Problem

- Consider a program for classification of a triangle.
- Its input is a triple of positive integers (say a, b and c) and the input values are greater than zero and less than or equal to 100.
- The triangle is classified according to the following rules:
  - Right angled triangle: $c^2=a^2+b^2$ or $a^2=b^2+c^2$ or $b^2=c^2+ a^2$
  - Obtuse angled triangle: $c^2>a^2+b^2$ or $a^2>b^2+c^2$ or $b^2>c^2+ a^2$
  - Acute angled triangle: $c^2<a^2+b^2$ or $a^2<b^2+c^2$ or $b^2<c^2+a^2$
  - The program output may have one of the following words: [Acute angled triangle, Obtuse angled triangle, Right angled triangle, Invalid triangle, Input values are out of range]

Answer

- Causes are:
  - c1: side “a” is less than the sum of sides “b” and “c”.
  - c2: side “b” is less than the sum of sides “a” and “c”.
  - c3: side “c” is less than the sum of sides “a” and “b”.
  - c4: square of side “a” is equal to the sum of squares of sides “b” and “c”.
  - c5: square of side “a” is greater than the sum of squares of sides “b” and “c”.
  - c6: square of side “a” is less than the the sum of squares of sides “b” and “c”.
- Effects are:
  - e1: Invalid triangle
  - e2: Right angle triangle
  - e3: Obtuse angled triangle
  - e4: Acute angled triangle
  - e5: Impossible stage

Cause-Effect Graph



Decision Table

	1	2	3	4	5	6	7	8	9	10	11
c1: $a<b+c$	F	T	T	T	T	T	T	T	T	T	T
c2: $b<a+c$	-	F	T	T	T	T	T	T	T	T	T
c3: $c<a+b$	-	-	F	T	T	T	T	T	T	T	T
c4: $a^2=b^2+c^2$	-	-	-	T	T	T	T	F	F	F	F
c5: $a^2>b^2+c^2$	-	-	-	T	T	F	F	T	T	F	F
c6: $a^2<b^2+c^2$	-	-	-	T	F	T	F	T	F	T	F
e1: Invalid triangle	X	X	X								
e2: Right angle triangle							X				
e3: Obtuse angled triangle									X		
e4: Acute angled triangle										X	
e5: Impossible				X	X	X		X			X

Thank You