中山大學
SUN YAT-SEN UNIVERSITY

Part II [Problem]
3. Test Adequacy (Blackbox, level-1)

SE-307 Software Testing Techniques

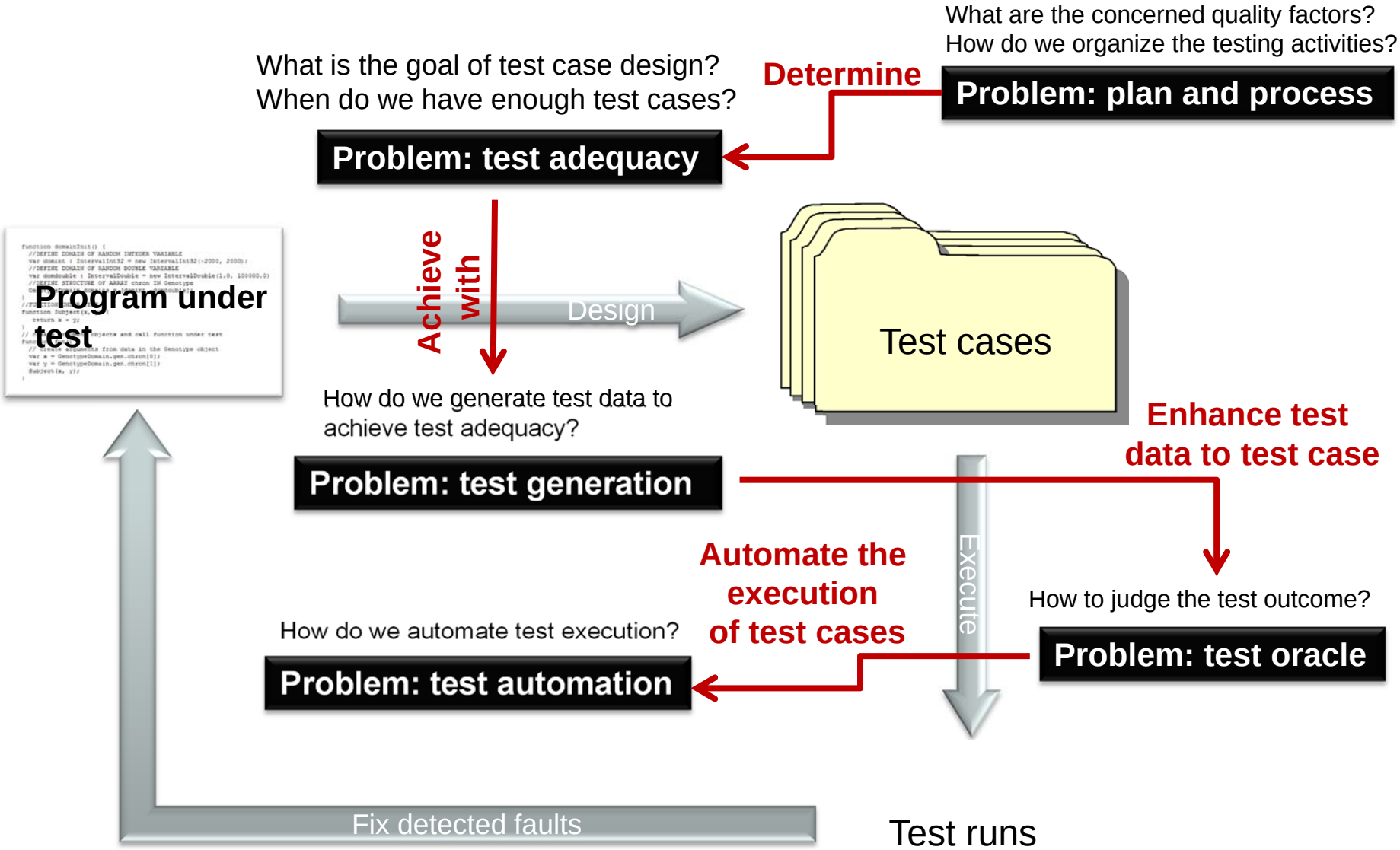
http://my.ss.sysu.edu.cn/wiki/display/SE307/Home

Instructor: Dr. Wang Xinming, School of Software, Sun Yat-Sen University

Review: Problems

- First dimension in PMOD: **Problems**
  - **Fundamental** problems
    - Test oracle
    - Test adequacy ← We are here!
    - Test generation
  - **Important** problems
    - Test process and plan
    - Test automation

Review: their relation



Review: What we have learnt

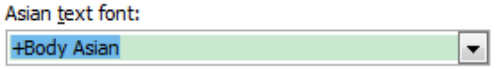
- Dimension one: Problems
  - Test oracle
  - Test adequacy
    - Whitebox (白盒测试充分性标准)
      - Control-flow coverage（控制流覆盖）
      - Logic coverage （逻辑覆盖）
      - Data-flow coverage（数据流覆盖）
      - Interface coverage （接口覆盖）
      - Mutant coverage（变异覆盖）
    - Blackbox (黑盒测试充分性标准)

Test Adequacy: Whitebox vs. Blackbox

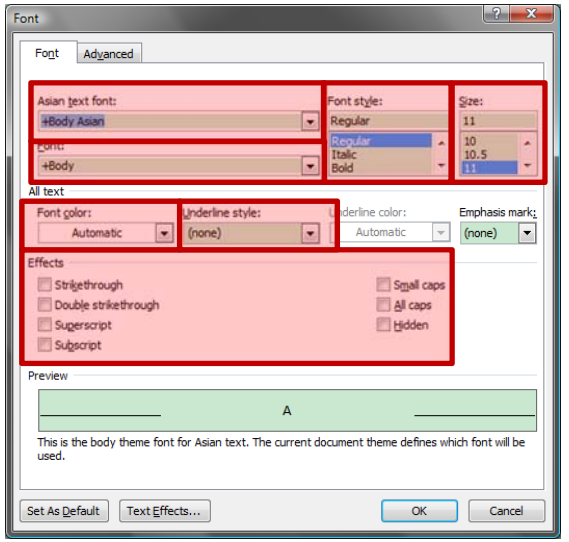
- Test adequacy is about the questions:
  - What is the goal of test suite design?
  - Is our test suite 'adequate'?
  - When can we stop designing new test cases?
- **Whitebox** approach to test adequacy:
  - The goal is to exercise elements in the source code. e.g. statements, basic blocks, branches, loops, basis-paths, conditions, du-paths, function-calls, etc.
  - Our test suite is considered adequate **if every concerned code element is covered by at least one test case**.
- **Blackbox** approach to test adequacy:
  - The goal is to exercise the behaviors of the units-under-test based on its requirement, without considering its implementation detail. Such a unit can be a method, a class, a sub-system, or the whole programs.
  - Our test suite is considered adequate **if it covers representative choices of input to the unit-under-test**.

Covering the Representative Choices of Input

- Level 1: ***Single*** input parameter
  - Cover the representative choices for **one single input parameter**.
  - e.g. the font type. (宋体 or 黑体 or Arial or ….)



- Level 2: ***Combination*** of input parameters
  - Cover the representative **combinations of multiple parameters**.
  - e.g. settings in the whole font dialog. (宋体-加粗-11号-红色 or 宋体-普通-小四-黑色 or 黑体-加粗-12号-黄色 or …)



- Level 3: ***Sequence*** of parameter combinations
  - Cover the representative **sequences of parameter combinations**.
  - e.g. font settings for multiple paragraphs. (e.g. 为不同的段落设置不同的字体)。

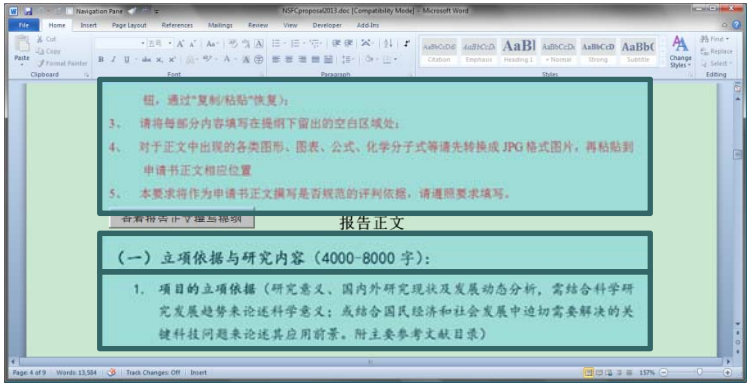


Illustration of The Three Levels

Level 1

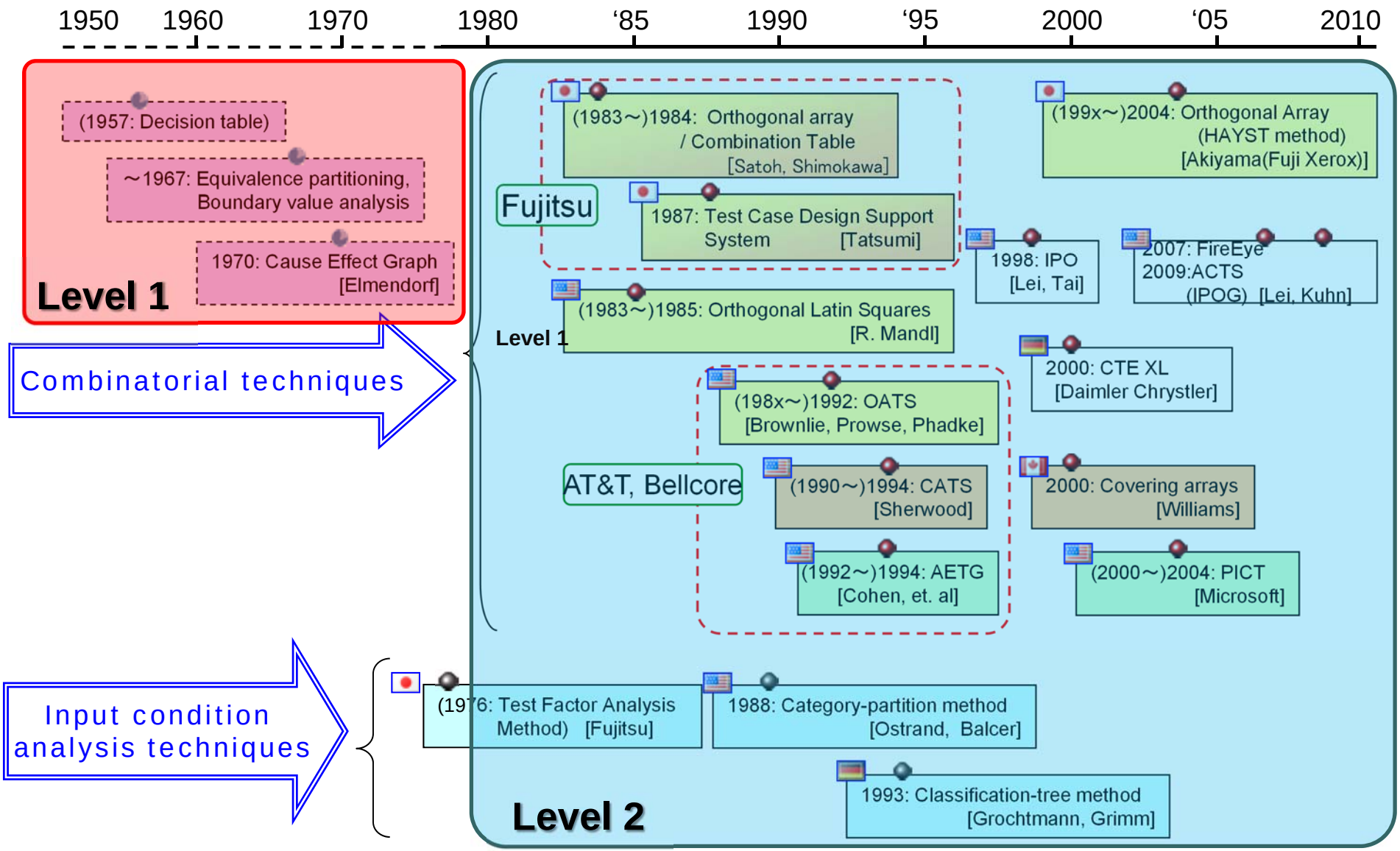
Level 2

Level 3

Level 1 or Level 2?

- When the input involves multiple elements
  - Treat them as one single parameter or multiple parameters?
- One single parameter if …
  - Elements are handled independently in the same way. *e.g. a list of user names to grant write-access*.
  - The order of the elements matters. *e.g. a list of number to sort.*
  - The elements are tightly coupled. e.g. *Coordinate (x, y).*
- Multiple parameters if…
  - Elements are loosely coupled. e.g. user name vs. server address
- In other cases, however, either way is OK.
  - Username/password.
  - The lengths of edges in a triangle.
  - Day/month/year.

History of Blackbox Testing Techniques



Level-1: Adequacy For One Parameter

Level-1 Techniques

- **Equivalence class partitioning （ECP, 等价类划分）**
  - Partition the input domain of the parameter into equivalence classes
  - Adequacy criterion: cover each partition at least once
- **Boundary value analysis （BVA, 边界值分析）**
  - Analyze the boundary cases for each equivalence class in ECP.
  - Adequacy criterion: cover each boundary case at least once.
- **Cause-effect graph and decision table（因果图和决策表）**
  - Analyze the causal relation between input and output as *edges*.
  - Adequacy criterion: cover each edge at least once.

Equivalence Class Partitioning (ECP)

- Suppose that we were going to test a method that implements the absolute value function for integers.

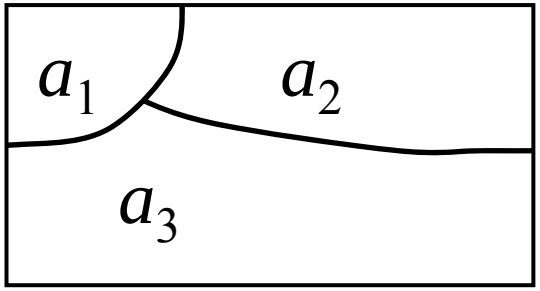
```
public int abs( int x )
```

- Exhaustive testing would require testing every possible value of the type `int`.
  - Not practical
- Instead, see if we can partition the input domain into equivalence classes.
  - A set or range of input domain values can be considered to be an equivalence class if they can reasonably be expected to **cause similar responses from the program**.
  - Need to consider both **valid classes** (合法输入) and **invalid classes** (非法输入).

Concept: Partitioning

- A partitioning of a set A is to divide A into subsets a_1, a_2, ..., a_n such that:
  - $a_1 \cup a_2 \cup \ldots \cup a_n = A$ (***completeness***)
  - for any i and j, $a_i \cap a_j = \emptyset$ (***disjoint***)

- Defined by a reflexive, symmetric, transitive relation between input data points (equivalence relation)



Rule to partition an input domain: group inputs that cause similar responses from the program as described in the requirement.

Input in the same group usually drive the program to execute the same paths – but we **shall not** partition input based on the source code.

Equivalence Class Partition (ECP)

`[Integer.MIN_VALUE, -1] [0] [1,Integer.MAX_VALUE]`

- Cover each equivalence class by choosing a "representative" value from each class. Any value ought to be as good as any other (for now... In BVA we require additional values at the boundary)

`[Integer.MIN_VALUE, -1]`:	Choose **`-34`**
`[0]`:	Choose **`0`**
`[1,Integer.MAX_VALUE]`:	Choose **`+42`**

How to Find Classes?

- Based on what the requirement says:
  - The parameter shall be within a range [a, b].
    - within range [a, b]: a valid class
    - too large (b, ∞]: an invalid class
    - too small [- ∞, a): a invalid class
  - The parameter shall be one in a set of values (e.g. ‘car’, ‘truck’):
    - One valid equivalence class for each value in the enumeration.
    - One invalid equivalence class: all values not in the set (i.e. everything else).
  - The parameter shall satisfy some conditions (e.g. “first character of the identifier must be a letter and the last one must be a number”) :
    - One valid class for satisfying the conditions (e.g. the first character is a letter).
    - One invalid class for violating each condition (e.g. the first character is not a letter).
  - The parameter shall be a list of values. (e.g. the input shall be a string of at least 8 characters and at most 12 characters)
    - Create one valid class for the correct number of values
    - Create two invalid classes – one for fewer values and one more values
- Enumerate the combination if more than one cases are matched.

Additional Classes

- They might not be explicitly mentioned in the requirement. But it is still necessary to test them:
  - Empty: value exists, but has no contents.
    - e.g. empty string “”, empty set, empty list.
  - Blank: value exists, and has content.
    - e.g. string that contains characters only “ ”
  - Null: value does not exist or is not allocated.
    - e.g. null reference, NULL pointer
  - None: does not provide value for the parameter, or when the parameter require selecting one item from a list, make no selection.
  - Very long: the length of the input is extremely long.
  - Special value: February 29
  - Encoding of string: utf-8 vs. GBK

Complexity in ECP (1)

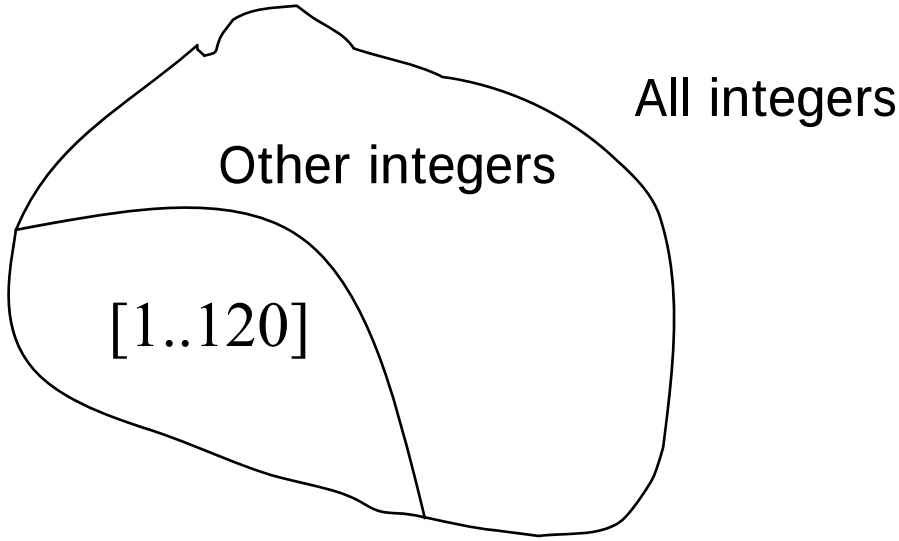
- Invalid Partitioning （无效的等价类划分）
  - The classes have overlapping part, or they do not cover the whole input domain.
  - Example: input parameter → a sorted list of numbers
    - Class 1: a list of numbers with ascending order
    - Class 2: a list of numbers with descending order
    - **Problem 1**: they are overlapping when the list contains *one number* or *numbers of the same value*.
    - **Problem 2**: they miss the case when the list contains *no number*.
- What shall we do?
  - Answer: create new classes for the overlapping and missing parts.
  - Example:
    - Class 1 & class 2: additionally require that the list contains more than one number and not all of they are the same.
    - Class 3: a list with one single number.
    - Class 4: a list with more than one numbers of the same value.
    - Class 5: an empty list.

Complexity in ECP (2)

- Partitioning that is too coarse （太粗略的等价类划分）
  - Some class shall have been further partitioned.
  - Example: the “file name” parameter when saving a file in Windows XP
    - Class 1: Valid file name
    - Class 2: Invalid file name
  - **Problem**: class 2 is too coarse.
    - There are different reasons why a file name is invalid.
- What shall we do?
  - (Obvious) answer: Further partition the class.
  - Example:
    - Class 2-1: device name ‘COM1’, ‘COM2’, … ‘COM9’, ‘LPT1’, … ’LPT9’
    - Class 2-2: contains an invalid character in ‘/\<>*?:;|”’ e.g. ‘test?’, ‘test*’
    - Class 2-3: empty name e.g. ‘’
    - Class 2-4: blank name e.g. ‘ ’
    - Class 2-5: name of a file already exists in the current directory

Example 1

Consider an application A that takes an integer denoted by age as input. Let us suppose that the only legal values of age are in the range [1..120]. The set of input values is now divided into a set E containing all integers in the range [1..120] and a set U containing the remaining integers.

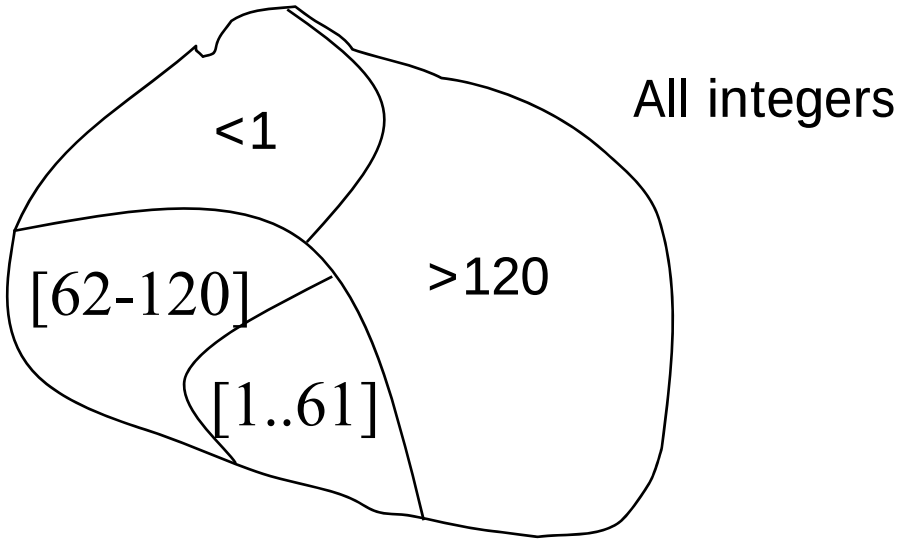


Example 1 (contd.)

Further, assume that the application is required to process all values in the range [1..61] in accordance with requirement R1 and those in the range [62..120] according to requirement R2.
Thus E is further subdivided into two regions depending on the expected behavior.

Similarly, it is expected that all invalid inputs less than or equal to 1 are to be treated in one way while all greater than 120 are to be treated differently. This leads to a subdivision of U into two categories.

Example 1 (contd.)



Example 1 (contd.)

Tests selected using the equivalence partitioning technique aim at targeting faults in the application under test with respect to inputs in any of the four regions, i.e. two regions containing expected inputs and two regions containing the unexpected inputs.

It is expected that any single test selected from the range [1..61] will reveal any fault with respect to R1. Similarly, any test selected from the region [62..120] will reveal any fault with respect to R2. A similar expectation applies to the two regions containing the unexpected inputs.

Example 2

This example shows a few ways to define equivalence classes based on the knowledge of requirements and the program text.

Consider that wordCount method takes a word w and a filename f as input and returns the number of occurrences of w in the text contained in the file named f. An exception is raised if there is no file with name f.

Example 2 (contd.)

```
begin
    String w, f
    Input w, f
    if (not exists(f) {raise exception; return(0);}
    if(length(w)==0)return(0);
    if(empty(f))return(0);
    return(getCount(w,f));
end
```

Using the partitioning method described in the examples above, we obtain the following equivalence classes.

Example 2 (contd.)

Equivalence class	w	f
E1	non-null	exists, not empty
E2	non-null	does not exist
E3	non-null	exists, empty
E4	null	exists, not empty
E5	null	does not exist
E6	null	exists, empty

Quiz 1: ECP

- A phone number with the following format:

  (XXX) XXX – XXXX

  Area Code (optional) **Prefix** **Suffix**

  - **Area Code**: 3-digit number [`200` … `999`] except `911`
  - **Prefix**: 3-digit number not beginning with `0` or `1`
  - **Suffix**: 4-digit number

- What are the equivalence classes for the three parameters?

Quiz 1: ECP

- Area Code: 3-digit number [200 … 999] except 911
  - Five Classes:
    - (- ∞ …199], [200 … 910], [911], [912 … 999], [1000 … ∞)
- Prefix: 3-digit number not beginning with 0 or 1
  - Three Classes:
    - (- ∞… 199], [200 … 999], [1000 … ∞)
- Suffix: 4-digit number
  - Three Classes:
    - (-∞ … -1], [0000 … 9999], [10000 … ∞)

Quiz 1: ECP

- Suppose we have a 学生管理系统
- And the requirement says: “用户凭自己的用户编号和口令进行登录；用户编号为对应的学号，学号为5位数字，由07001到07100；用户口令要求最少为8位，最多为12位，必须同时含有字符（字母或符号）和数字。”
  - What are the classes for the parameter “学号”?
  - What are the classes for the parameter “密码”?

学号:

- 五位数字
  - 07001~07100 间的5位数字：’07050’
  - 小于07001的五位数字：’06050’
  - 大于07001的五位数字：’08199’
- 小于五位，非空的数字：’010’
- 大于五位的数字：’070010’
- 有非数字
  - 5位：’010A0’
  - 小于5位，非空：’*000’
  - 大于5位：’700@01’
- 空：”
- 空串：’ ’
- 超长：’00000000000000000000…..0’

Quiz 1: ECP

- Suppose we have a 学生管理系统
- And the requirement says: “用户凭自己的用户编号和口令进行登录；用户编号为对应的学号，学号为5位数字，由07001到07100；用户口令要求最少为8位，最多为12位，必须同时含有字符（字母或符号）和数字。”
  - What are the classes for the parameter “学号”?
  - What are the classes for the parameter “密码”?

密码:

- 包含字符和数字
  - 8位到12位：’se307hello!’
  - 少于8位, 非空： ’se307!’
  - 多于12位： ’se307helloworld!’
- 只包含字符，非空串
  - 8位到12位：’sehelloworld!’
  - 少于8位，非空： ’sehello’
  - 多于12位： ’sehelloworld!’
- 只包含数字
  - 8位到12位：’307307307’
  - 少于8位，非空： ’307’
  - 多于12位：’307307307307307’
- 空：”
- 空串：’ ’
- 超长：’f9asdhqsd0vsdv….q394sdhk’

Level-1 Techniques

- **Equivalence class partitioning （ECP, 等价类划分）**
  - Partition the input domain of the parameter into equivalence classes
  - Adequacy criterion: cover each partition at least once
- **Boundary value analysis （BVA, 边界值分析）**
  - Analyze the boundary cases for each equivalence class in ECP.
  - Adequacy criterion: cover each boundary case at least once.
- **Cause-effect graph and decision table（因果图和决策表）**
  - Analyze the causal relation between input and output as *edges*.
  - Adequacy criterion: cover each edge at least once.

Motivation

- It has been found that most errors are caught at the **boundary** of the equivalence classes.
- Not surprising, as the end points of the boundary are usually used in the code for checking:
  - E.g., checking K is in range [X … Y):

```
if (K >= X && K <= Y)
    ...
```

Easy to make mistake on the comparison.

- Boundary value analysis requires that these boundary cases being covered by the test suite.
  - It is an extension and refinement of ECP.

Boundary Value Analysis (BVA)

- Given the equivalence classes of a parameter P, BVA requires that for each boundary between classes, cover:
  - The boundary
  - Just above the boundary by one min_unit
  - Just below the boundary by one min_unit
- **Note**: it is assumed that a nominal value of C has already been covered in ECP. BVA sough to enhance ECP with additional test cases.
  - These test cases are redundant in ECP, but necessary in BVA.
- The question, of course, is how to define ‘boundary’, ‘just above the boundary’.

How to Find Boundary Cases?

- If the parameter shall be within a range, [X … Y]
  - Four values should be tested:
    - Valid: X and Y
    - Invalid: Just below X (e.g., `X – min_unit`)
    - Invalid: Just above Y (e.g., `Y + min_unit`)
  - E.g., [1 … 12], and the min_unit is 1
    - Test Data: `{0, 1, 12, 13}`

- Similar for open interval (X … Y), i.e., X and Y not inclusive.
  - Four values should be tested:
    - Invalid: X and Y
    - Valid: Just **above** X (e.g., `X + min_unit`)
    - Valid: Just **below** Y (e.g., `Y – min_unit`)
  - e.g., (`100 … 200`) and the `min_unit` is 1
    - Test Data: `{100, 101, 199, 200}`

How to Find Boundary Cases?

- If the parameter is one from a list of values:
  - Four values should be tested:
    - Valid: `min, max`
    - Invalid: Just below `min` (e.g., `min – min_unit`)
    - Invalid: Just above `max` (e.g., `max + min_unit`)
  - E.g., the input parameter shall be one in `{2, 4, 6, 8}` and `min_unit` is 1.
    - Test Data: `{1, 2, 8, 9}`
- If the parameter is a data structure:
  - Define test data to exercise the data structure at the prescribed boundaries.
  - E.g.,
    - Array : Empty Array, Array with 1 element, Full Array
    - Set: Empty set, set with 1 element
- Boundary value analysis is not applicable for data with no meaningful boundary, e.g., the set *color* `{Red, Green, Yellow}`.

Quiz 2: BVA

- A mail order company charges $2.95 postage for deliveries if the package weighs less than 2 kg, $3.95 if the package weighs 2 kg or more but less than 5 kg, and $5 for packages weighing 5 kg or more.
  - What are the classes for the parameter “package weighs”?
  - What are the boundary cases that are required to be covered by BVA? (assume the min_unit is 1 gram).

Classes:

C1: 0 (invalid class)
C2: (0, 2kg)
C3: [2kg, 5kg)
C4: [5kg, ∞)

Boundary:

C2: 0.001kg, 1.999kg, 2kg
C3: 2.001kg, 5kg
C4: 5.001kg

Level-1 Techniques

- **Equivalence class partitioning （ECP, 等价类划分）**
  - Partition the input domain of the parameter into equivalence classes
  - Adequacy criterion: cover each partition at least once
- **Boundary value analysis （BVA, 边界值分析）**
  - Analyze the boundary cases for each equivalence class in ECP.
  - Adequacy criterion: cover each boundary case at least once.
- **Cause-effect Graph and Decision Table（因果图和决策表）**
  - Analyze the causal relation between input (cause) and output (effect).
  - Adequacy criterion: cover the cause-effect graph with decision table.

Motivation

- **Problem 1**: ECP asks us to partition the domain of the input parameter into different classes by the different ways the program handle it. But how do we know these ways?
- **Problem 2**: When the input is subject to complicated conditions and their combinations, how do we design the equivalence classes?
- **The idea**: Partition the input domain by how the program responds (generate output) to different input conditions.
  - Establish the relation between input and output by reading the requirement.

Cause-Effect Graph

- Cause-effect graph models dependency between program input conditions (known as *causes*) and output condition (known as *effects*).
  - A cause is any condition in the requirements that may affect the program output.
  - An effect is the response of the program to some combination of input conditions.
- Example: the input parameter is a list of integers
  - Display message A if input contains 1 and 2.
  - Display message B if input contains 2 and 3.
  - Display message C if input contains 1 or 3.
  - Display message D if input contains 1, 2, and 4.
  - One and only one of {3, 4} must be in the input.

Causes:
input conditions

Effects:
output conditions

Constraint between causes

Notes

- There are more than one way to define the causes and effects.
  - Define one cause “input contains 1 and 2” vs. define two causes ““input contains 1” and “input contains 2”.
  - Define one effect “display both A and B” vs. define two effects “display A” and “display B”.
  - Cause-effect graph does not have any restriction here. But we usually prefer the latter in order to make the dependencies and constraints simple.

- The causes/effects do not necessarily partition the input/output domain.
  - More than one cause/effects can occur. e.g. display both A and B.
  - It is also possible that no cause/effect occurs. e.g. no message is displayed, input is null.
  - Some causes/effects might overlap.

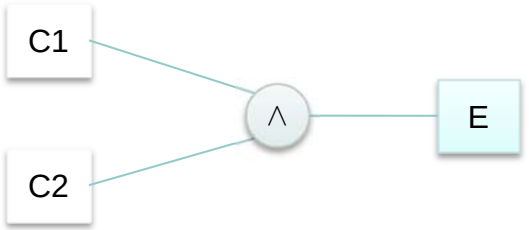
- Cause-effect graph can also apply to multiple parameters.
  - Causes can be used to describe input conditions of multiple parameters.
  - We will discuss it later as a technique for level 2.

Notations

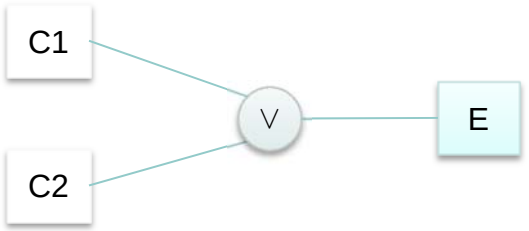
Dependencies

C causes E

¬C causes E

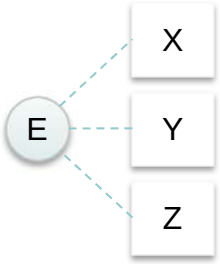


C1∧C2 cause E

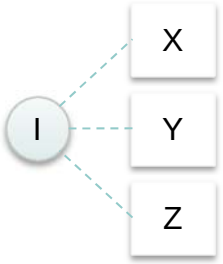


C1∨C2 cause E

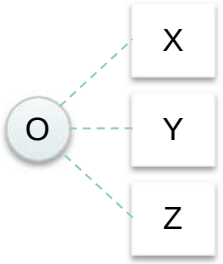
Constraints



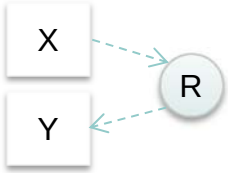
Exclusive:
X, Y, and Z cannot happens together



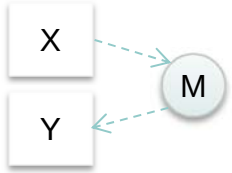
Inclusive:
At lease one of X, Y, and Z happens.



Only:
One and only one of X, Y, and Z happens.



Require:
If X is true, then Y must be true. i.e. X⇒Y



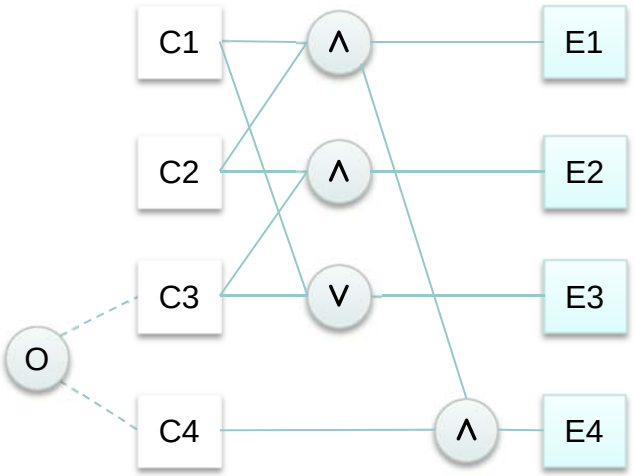
Mask:
If X is true, then Y must be false. i.e. X⇒¬Y

Example

- The input parameter is a list of integers
  - Display message A if input contains 1 and 2.
  - Display message B if input contains 2 and 3.
  - Display message C if input contains 1 or 3.
  - Display message D if input contains 1, 2, and 4.
  - One and only one of {3, 4} must be in the input.

C1: contains 1
C2: contains 2
C3: contains 3
C4: contains 4

E1: display A
E2: display B
E3: display C
E4: display D



Question: what do we need to cover in this graph?

Quiz 2: Cause-Effect Graph

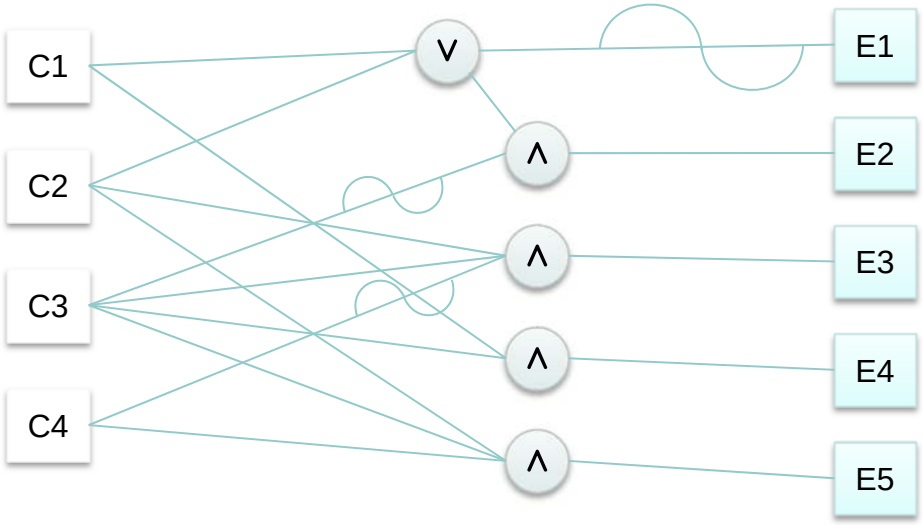
- Draw the cause-effect graph for an ATM transaction system's "withdraw money"(从储蓄账户或信用卡账户取款) feature.

Causes:

C1: Draw from `credit`(信用卡)account
C2: Draw from `debit`(储蓄) account
C3: Account No. is valid
C4: Transaction amount is no greater than the debit amount.

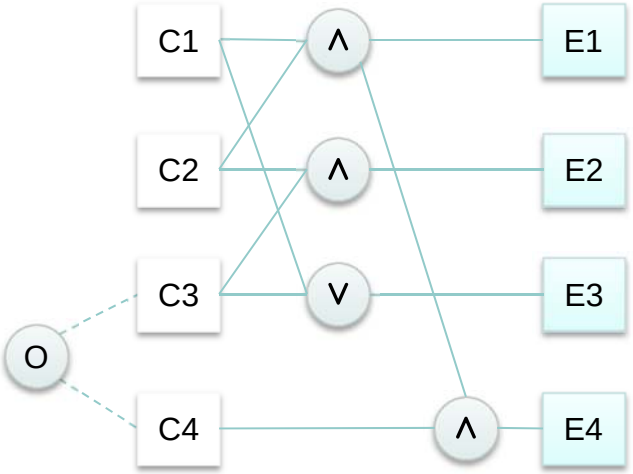
Effects:

E1: Print "invalid command",
E2: Print "invalid account No."
E3: Print "debit number is invalid" (存款余额不足)
E4: withdraw memory from credit account
E5: withdraw memory from debit account



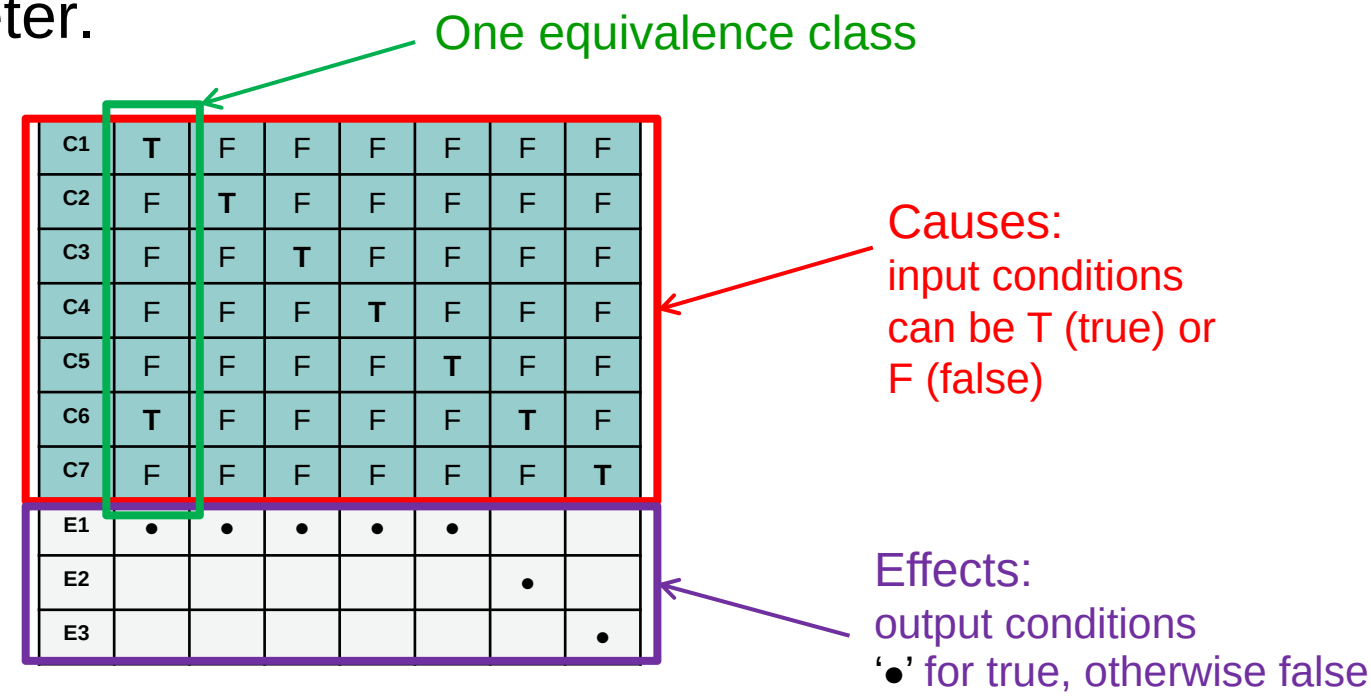
Coverage on Cause-Effect Graph

- **Cause-effect Coverage**
  - For each effect, cover its true and false case at least once.
  - For each '∧' node with *k* incoming values $n_1, n_2, \ldots n_k$, cover k+1 cases:
    - n_1=true, n_2=true, … n_k = true (all-true)
    - n_1=false, n_2=true, … n_k = true (all-but-n_1-true)
    - n_1=true, n_2=false, … n_k = true (all-but-n_2-true)
    - …
    - n_1=true, n_2=true, … n_k = false (all-but-n_k-true)
  - For each '∨' node with *k* incoming values $n_1, n_2, \ldots n_k$, cover k+1 cases:
    - n_1=false, n_2=false, … n_k = false (all-false)
    - n_1=true, n_2=false, … n_k = false (all-but-n_1-false)
    - n_1=false, n_2=true, … n_k = false (all-but-n_2-false)
    - …
    - n_1=true, n_2=true, … n_k = true (all-but-n_k-false)
- If some cases are infeasible as they violate the constraints, we don't need to cover them.



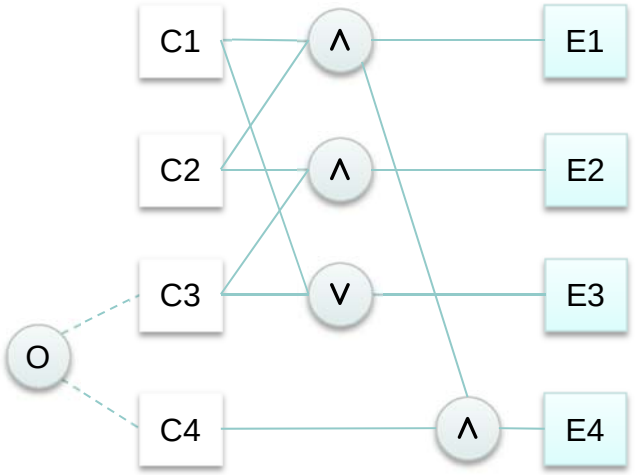
Decision Table

- A tool to plan the test cases for covering cause-effect graph.
- Consist of two parts:
  - The upper part shows the causes
  - The lower part shows the effects
- Each column represents one equivalence class for the input parameter.



Example

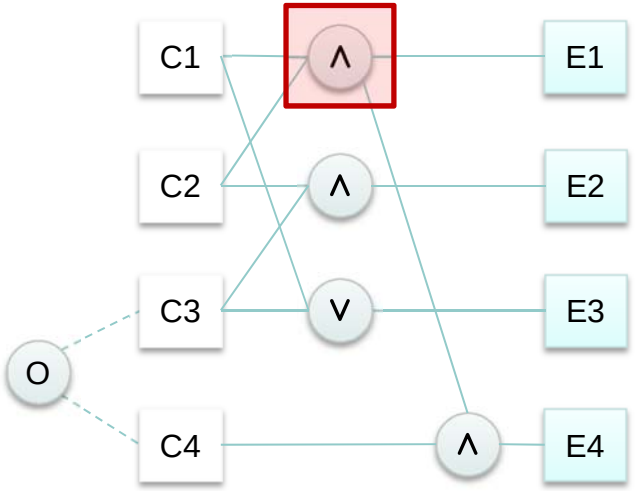
- For each effect, cover its true and false case at least once.



C1	**T**	F	**T**	**T**	**T**	F	F
C2	**T**	**T**	F	**T**	**T**	F	F
C3	F	**T**	**T**	**T**	F	F	**T**
C4	**T**	F	F	F	F	**T**	F
E1	•			•			
E2		•		•	•		
E3	•	•	•				•
E4	•						

Example

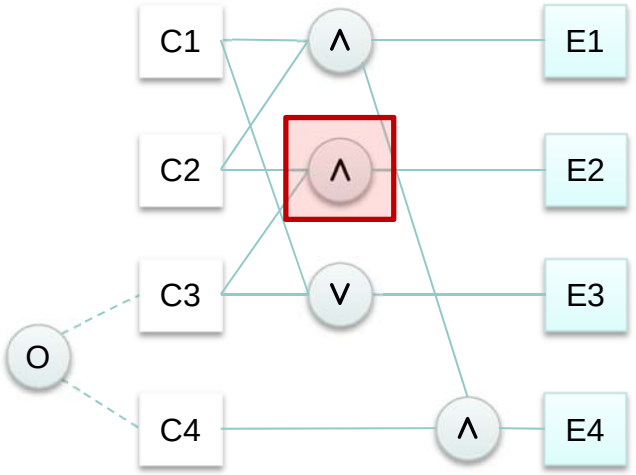
- For each '∧' node with *k* incoming values $n_1, n_2, \ldots n_k$, cover k+1 cases.
  - all-true
  - all-but-n_1-true
  - all-but-n_1-true
  - …
  - all-but-n_k-true



C1	T	F	T	T	T	F	F
C2	T	T	F	T	T	F	F
C3	F	T	T	T	F	F	T
C4	T	F	F	F	F	T	F
E1	•			•			
E2		•		•	•		
E3	•	•	•				•
E4	•						

Example

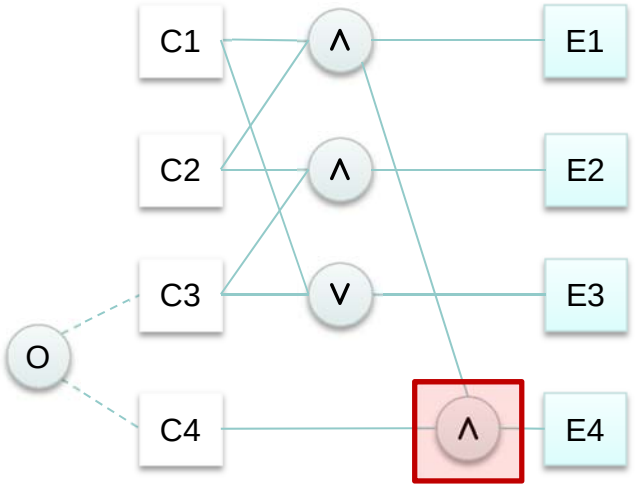
- For each '∧' node with *k* incoming values $n_1, n_2, \ldots n_k$, cover k+1 cases.
  - all-true
  - all-but-n_1-true
  - all-but-n_1-true
  - …
  - all-but-n_k-true



C1	**T**	F	**T**	**T**	**T**	F	F
C2	**T**	**T**	F	**T**	**T**	F	F
C3	F	**T**	**T**	**T**	F	F	**T**
C4	**T**	F	F	F	F	**T**	F
E1	•			•			
E2		•		•	•		
E3	•	•	•				•
E4	•						

Example

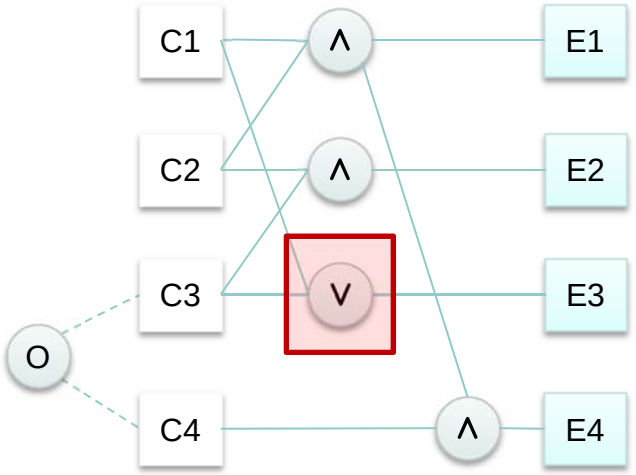
- For each '∧' node with *k* incoming values $n_1, n_2, \ldots n_k$, cover k+1 cases.
  - all-true
  - all-but-n_1-true
  - all-but-n_1-true
  - …
  - all-but-n_k-true



C1	T	F	T	T	T	F	F
C2	T	T	F	T	T	F	F
C3	F	T	T	T	F	F	T
C4	T	F	F	F	F	T	F
E1	•			•			
E2		•		•	•		
E3	•	•	•				•
E4	•						

Example

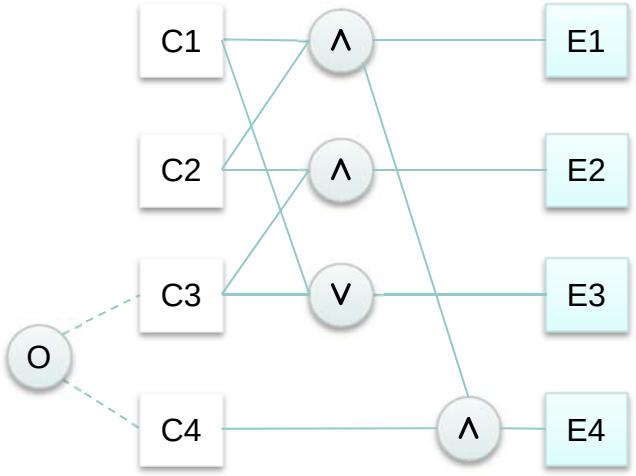
- For each '∨' node with *k* incoming values $n_1, n_2, \ldots n_k$, cover k+1 cases.
  - all-false
  - all-but-n_1-false
  - all-but-n_1-false
  - …
  - all-but-n_k-false



C1	T	F	T	T	T	F	F
C2	T	T	F	T	T	F	F
C3	F	T	T	T	F	F	T
C4	T	F	F	F	F	T	F
E1	•			•			
E2		•		•	•		
E3	•	•	•				•
E4	•						

Finding Cause-Effect Adequate Test Cases

- Generate a decision table from a cause-effect graph to achieve cause-effect coverage
- **Input**: A cause-effect graph with N causes C_1, C_2, … C_N, M effects E_1, E_2, … E_m, and their associated constraints
- **Output**: A decision table DT with N+M rows and W columns, where W depends on the cause-effect graph.



Algorithm

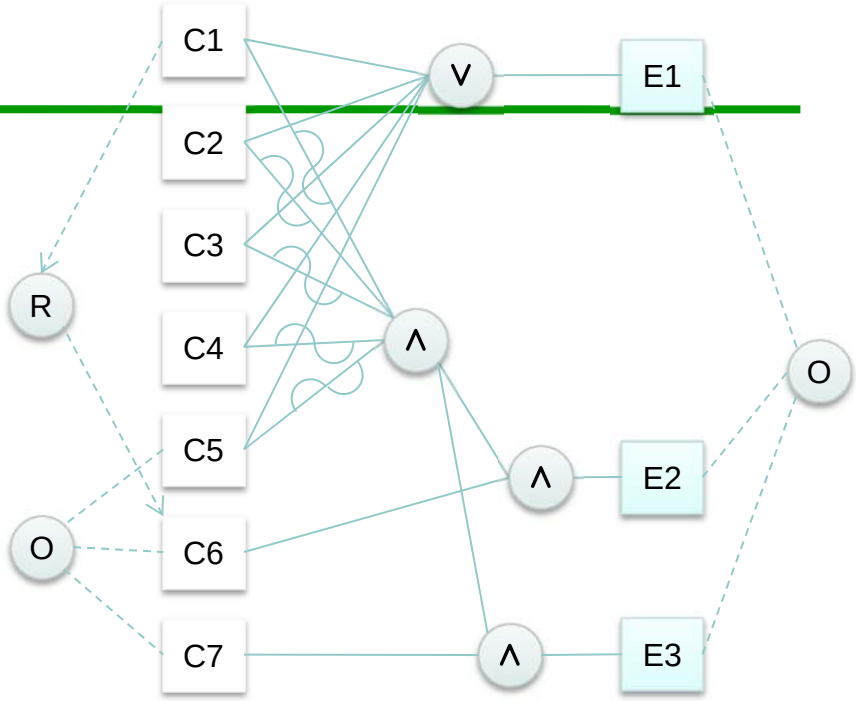
C1	T	F	T	T	T	F	F
C2	T	T	F	T	T	F	F
C3	F	T	T	T	F	F	T
C4	T	F	F	F	F	T	F
E1	•			•			
E2		•		•	•		
E3	•	•	•				•
E4	•						

Finding Cause-Effect Adequate Test Cases

- Initialize an empty decision table.
- For each of the effects E_i in E_1, E_2, … E_m
  - Starting from E_i, trace back through the graph, for each encountered node:
    - A '$\vee$' node with k precedents: find k+1 combinations of causes that cover the k+1 cases: all-false … all-but-n_k-false. Continue tracing each precedent.
    - A '$\wedge$' node with k precedents: find k+1 combinations of causes that cover the k+1 cases: all-true … all-but-n_k-true. Continue tracing each precedent.
    - The node has been covered in previous iteration: ignore this node.
  - Create a column in the decision table for each combination of causes. For causes not covered in the combination, freely set the as true/false as long as the constraints are satisfied.
  - Set the effects for each column.
- If the true/false value of an effect is not covered in the above loop, find a combination of causes to cover it and create a column in the decision table.
- Merge redundant columns in the decision table.

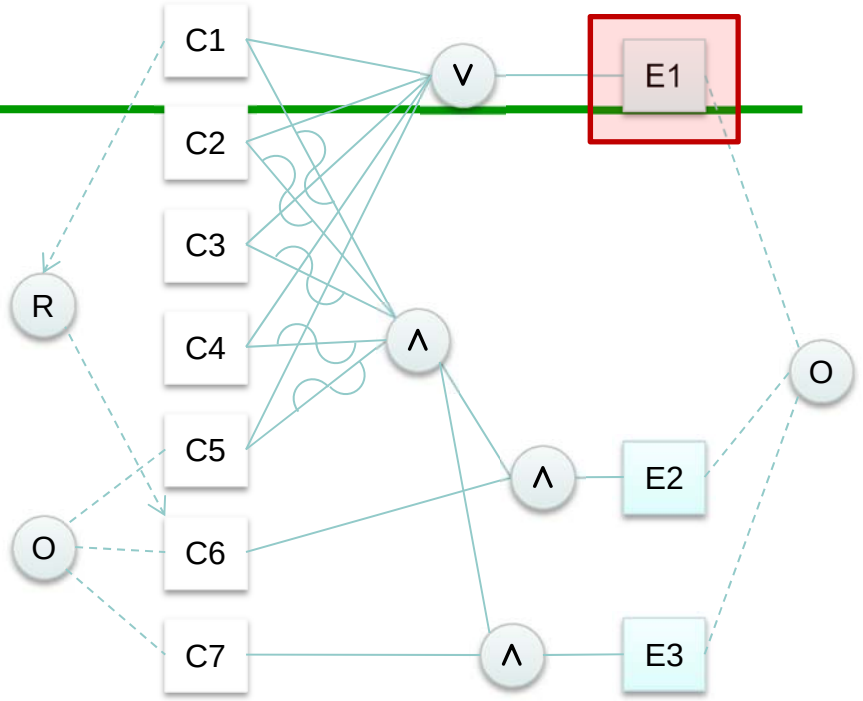
Example

- Movement of horse in Chinese chess（象棋中马的移动）
  - Input Parameter: the place where you want to move your horse.
- The effects:
  - E1: Illegal move
  - E2: Legal move into an empty space
  - E3: Legal move into an enemy's piece
- The causes:
  - C1: The place is outside the board.
  - C2: Does not follow the rule （非'日'字）.
  - C3: Another piece prevents the movement （绊马腿）.
  - C4: The move results in the king being checkmated.
  - C5: The place contains one's own piece.
  - C6: The place is empty.
  - C7: The place contains enemy's piece.



Example

- Movement of horse in Chinese chess（象棋中马的移动）
  - Input Parameter: the place where you want to move your horse.
- The effects:
  - E1: Illegal move
  - E2: Legal move into an empty space
  - E3: Legal move into an enemy's piece
- The causes:
  - C1: The place is outside the board.
  - C2: Does not follow the rule （非'日'字）.
  - C3: Another piece prevents the movement （绊马腿）.
  - C4: The move results in the king being checkmated.
  - C5: The place contains one's own piece.
  - C6: The place is empty.
  - C7: The place contains enemy's piece.

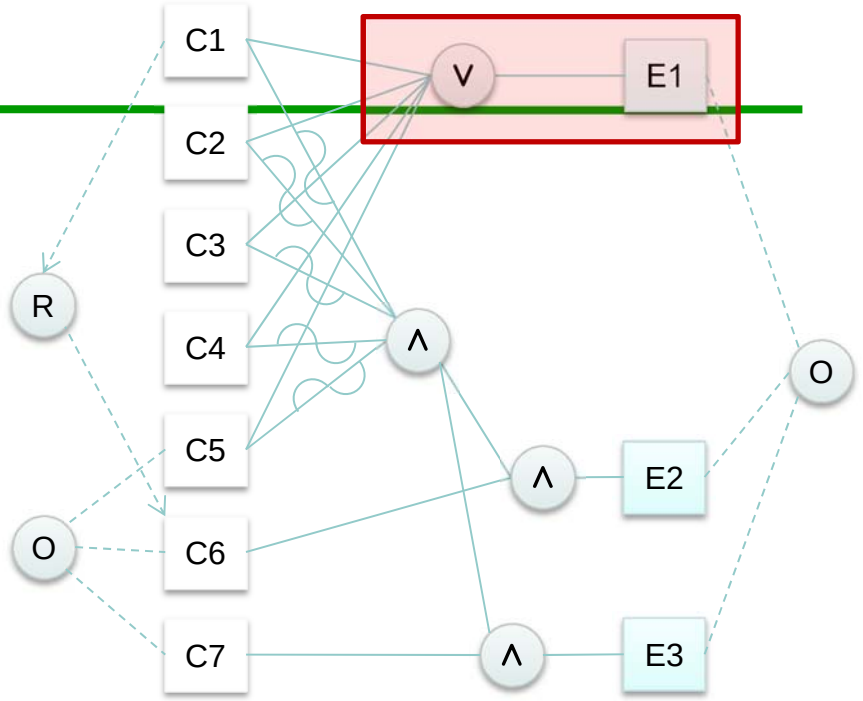


C1												
C2												
C3												
C4												
C5												
C6												
C7												
E1												
E2												
E3												

Consider the effect one by one.
Back trace to its causes.

Example

- Movement of horse in Chinese chess（象棋中马的移动）
  - Input Parameter: the place where you want to move your horse.
- The effects:
  - E1: Illegal move
  - E2: Legal move into an empty space
  - E3: Legal move into an enemy's piece
- The causes:
  - C1: The place is outside the board.
  - C2: Does not follow the rule （非'日'字）.
  - C3: Another piece prevents the movement （绊马腿）.
  - C4: The move results in the king being checkmated.
  - C5: The place contains one's own piece.
  - C6: The place is empty.
  - C7: The place contains enemy's piece.

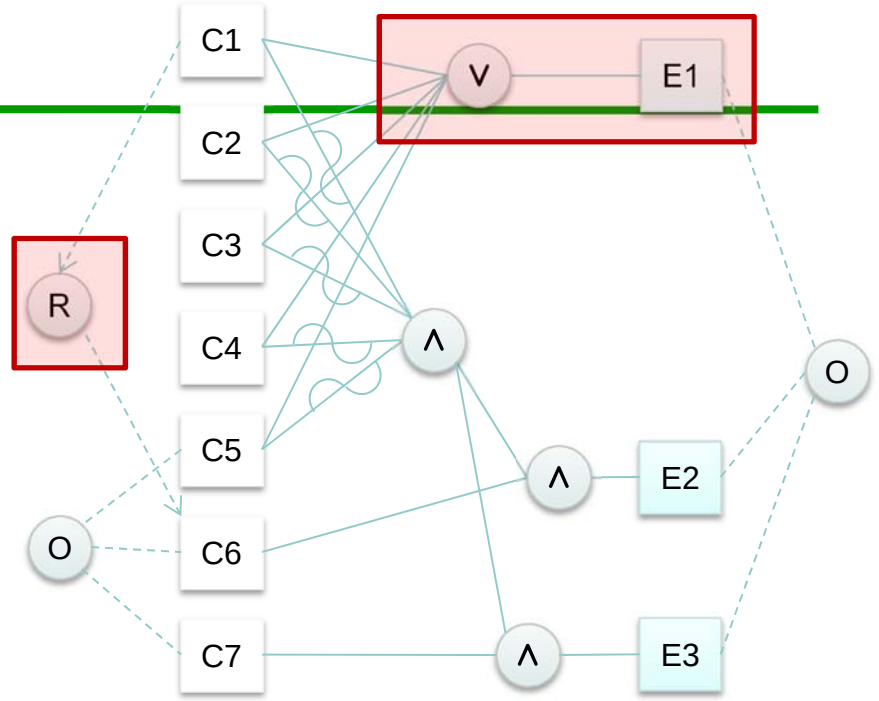


C1	T	F	F	F	F	F						
C2	F	T	F	F	F	F						
C3	F	F	T	F	F	F						
C4	F	F	F	T	F	F						
C5	F	F	F	F	T	F						
C6												
C7												
E1												
E2												
E3												

Cover the OR node with 6 combinations: all-false, … all-but-n_6-false

Example

- Movement of horse in Chinese chess（象棋中马的移动）
  - Input Parameter: the place where you want to move your horse.
- The effects:
  - E1: Illegal move
  - E2: Legal move into an empty space
  - E3: Legal move into an enemy's piece
- The causes:
  - C1: The place is outside the board.
  - C2: Does not follow the rule （非'日'字）.
  - C3: Another piece prevents the movement （绊马腿）.
  - C4: The move results in the king being checkmated.
  - C5: The place contains one's own piece.
  - C6: The place is empty.
  - C7: The place contains enemy's piece.

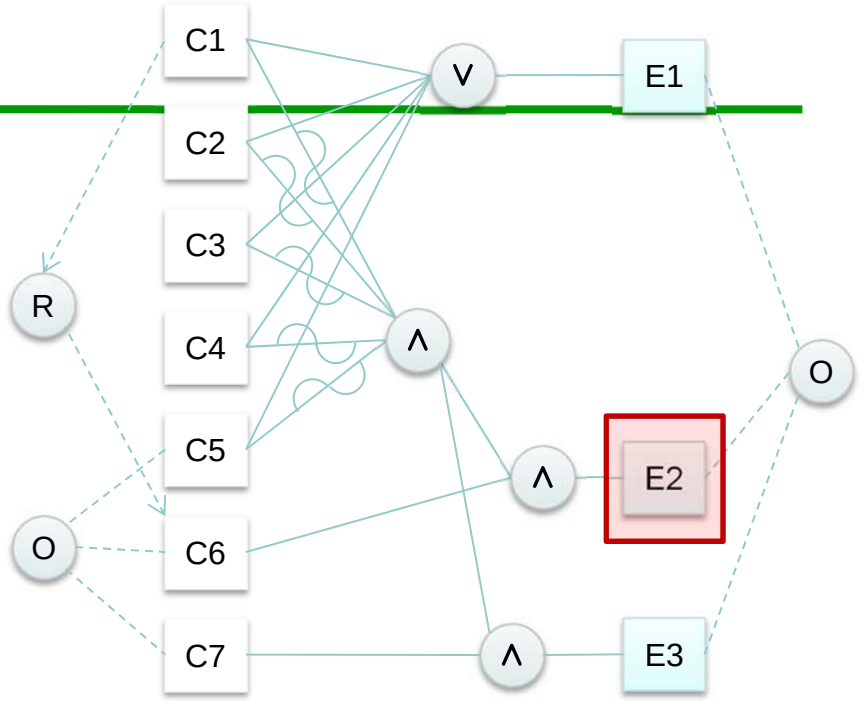


C1	T	F	F	F	F	F						
C2	F	T	F	F	F	F						
C3	F	F	T	F	F	F						
C4	F	F	F	T	F	F						
C5	F	F	F	F	T	F						
C6	T	T	T	T	F	T						
C7	F	F	F	F	F	F						
E1	•	•	•	•	•							
E2						•						
E3												

Freely set the other causes as long as they satisfy the constraint. Also, set the effects.

Example

- Movement of horse in Chinese chess（象棋中马的移动）
  - Input Parameter: the place where you want to move your horse.
- The effects:
  - E1: Illegal move
  - E2: Legal move into an empty space
  - E3: Legal move into an enemy's piece
- The causes:
  - C1: The place is outside the board.
  - C2: Does not follow the rule （非'日'字）.
  - C3: Another piece prevents the movement （绊马腿）.
  - C4: The move results in the king being checkmated.
  - C5: The place contains one's own piece.
  - C6: The place is empty.
  - C7: The place contains enemy's piece.

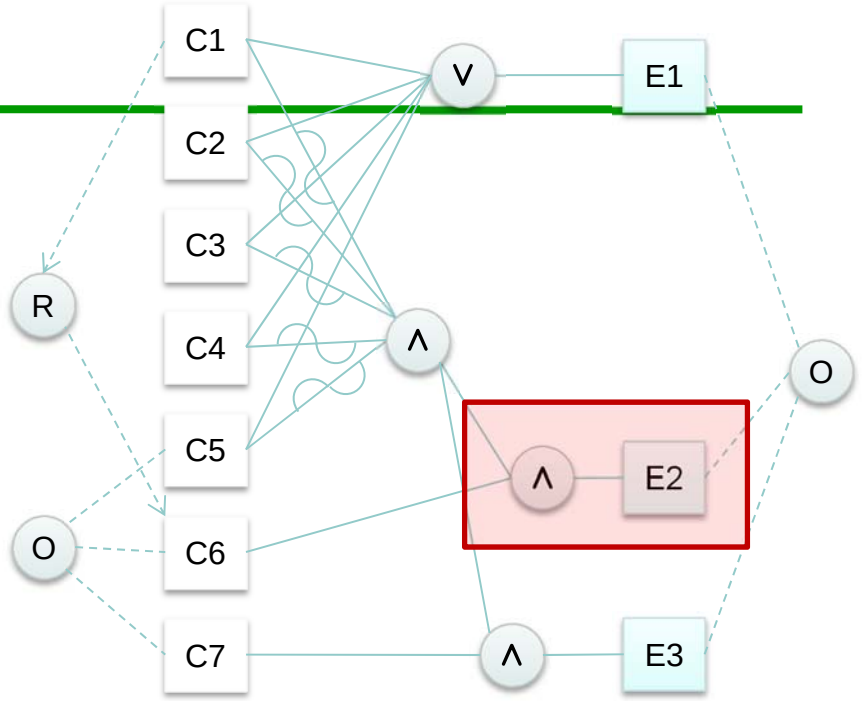


C1	**T**	F	F	F	F	F						
C2	F	**T**	F	F	F	F						
C3	F	F	**T**	F	F	F						
C4	F	F	F	**T**	F	F						
C5	F	F	F	F	**T**	F						
C6	**T**	**T**	**T**	**T**	F	**T**						
C7	F	F	F	F	F	F						
E1	•	•	•	•	•							
E2						•						
E3												

Consider the next effect E2

Example

- Movement of horse in Chinese chess（象棋中马的移动）
  - Input Parameter: the place where you want to move your horse.
- The effects:
  - E1: Illegal move
  - E2: Legal move into an empty space
  - E3: Legal move into an enemy's piece
- The causes:
  - C1: The place is outside the board.
  - C2: Does not follow the rule （非'日'字）.
  - C3: Another piece prevents the movement （绊马腿）.
  - C4: The move results in the king being checkmated.
  - C5: The place contains one's own piece.
  - C6: The place is empty.
  - C7: The place contains enemy's piece.

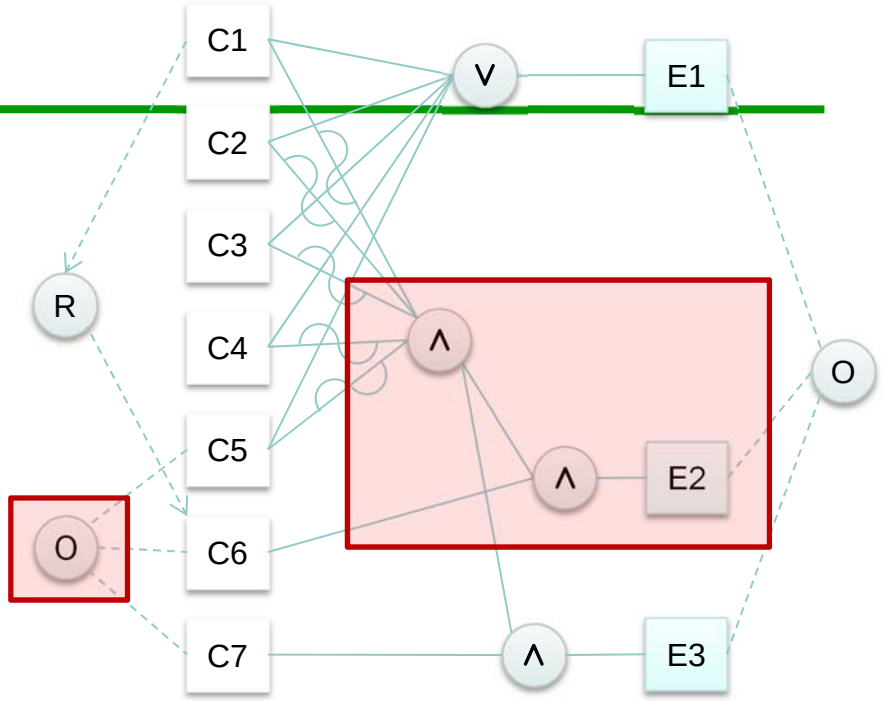


C1	T	F	F	F	F	F	F	F	T			
C2	F	T	F	F	F	F	F	F	F			
C3	F	F	T	F	F	F	F	F	F			
C4	F	F	F	T	F	F	F	F	F			
C5	F	F	F	F	T	F	F	F	F			
C6	T	T	T	T	F	T	T	F	T			
C7	F	F	F	F	F	F						
E1	•	•	•	•	•							
E2						•						
E3												

Cover the AND node with 3 combinations: all-true, all-but-n_1-true, all-but-n_2-true

Example

- Movement of horse in Chinese chess（象棋中马的移动）
  - Input Parameter: the place where you want to move your horse.
- The effects:
  - E1: Illegal move
  - E2: Legal move into an empty space
  - E3: Legal move into an enemy's piece
- The causes:
  - C1: The place is outside the board.
  - C2: Does not follow the rule （非'日'字）.
  - C3: Another piece prevents the movement （绊马腿）.
  - C4: The move results in the king being checkmated.
  - C5: The place contains one's own piece.
  - C6: The place is empty.
  - C7: The place contains enemy's piece.

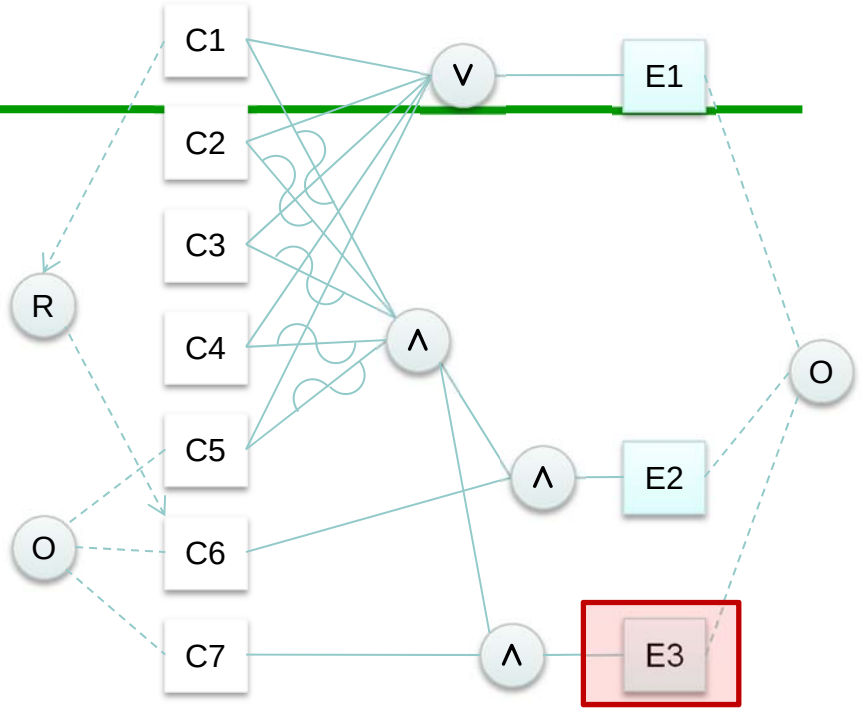


C1	T	F	F	F	F	F	F	F	T			
C2	F	T	F	F	F	F	F	F	F			
C3	F	F	T	F	F	F	F	F	F			
C4	F	F	F	T	F	F	F	F	F			
C5	F	F	F	F	T	F	F	F	F			
C6	T	T	T	T	F	T	T	F	T			
C7	F	F	F	F	F	F	F	T	F			
E1	•	•	•	•	•				•			
E2						•	•					
E3								•				

Ignore the next 'AND' node as it has been covered before. Again, set the other causes and the effects.

Example

- Movement of horse in Chinese chess（象棋中马的移动）
  - Input Parameter: the place where you want to move your horse.
- The effects:
  - E1: Illegal move
  - E2: Legal move into an empty space
  - E3: Legal move into an enemy's piece
- The causes:
  - C1: The place is outside the board.
  - C2: Does not follow the rule （非'日'字）.
  - C3: Another piece prevents the movement （绊马腿）.
  - C4: The move results in the king being checkmated.
  - C5: The place contains one's own piece.
  - C6: The place is empty.
  - C7: The place contains enemy's piece.



C1	T	F	F	F	F	F	F	F	T	F	F	F
C2	F	T	F	F	F	F	F	F	F	F	F	T
C3	F	F	T	F	F	F	F	F	F	F	F	F
C4	F	F	F	T	F	F	F	F	F	F	F	F
C5	F	F	F	F	T	F	F	F	F	F	F	F
C6	T	T	T	T	F	T	T	F	T	F	T	F
C7	F	F	F	F	F	F	F	T	F	T	F	T
E1	•	•	•	•	•				•			•
E2						•	•				•	
E3								•		•		

Repeat this for E3

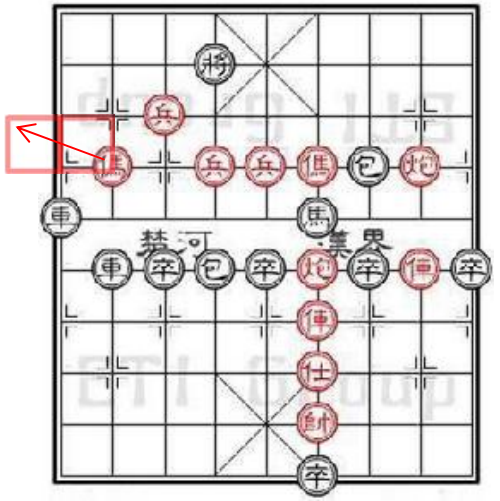
Example

- Movement of horse in Chinese chess（象棋中马的移动）
  - Input Parameter: the place where you want to move your horse.
- The effects:
  - E1: Illegal move
  - E2: Legal move into an empty space
  - E3: Legal move into an enemy's piece
- The causes:
  - C1: The place is outside the board.
  - C2: Does not follow the rule （非'日'字）.
  - C3: Another piece prevents the movement （绊马腿）.
  - C4: The move results in the king being checkmated.
  - C5: The place contains one's own piece.
  - C6: The place is empty.
  - C7: The place contains enemy's piece.

Reduce redundant test cases

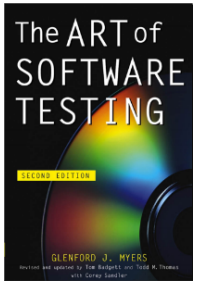
C1	T	F	F	F	F	F	F	F	T	F	F	F
C2	F	T	F	F	F	F	F	F	F	F	F	T
C3	F	F	T	F	F	F	F	F	F	F	F	F
C4	F	F	F	T	F	F	F	F	F	F	F	F
C5	F	F	F	F	T	F	F	F	F	F	F	F
C6	T	T	T	T	F	T	T	F	T	F	T	F
C7	F	F	F	F	F	F	F	T	F	T	F	T
E1	•	•	•	•	•				•			•
E2						•	•				•	
E3								•		•		

Generate one test case for each column
Example: column1 move from (1,3) to (-1,2)

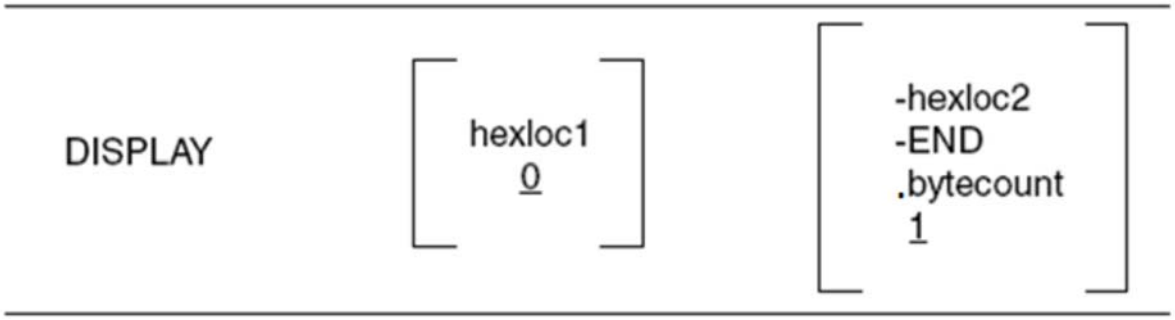


Case Study

- Analyse the requirement for a command in an interactive debugging system.
  - This example is from “The Art of Software Testing”



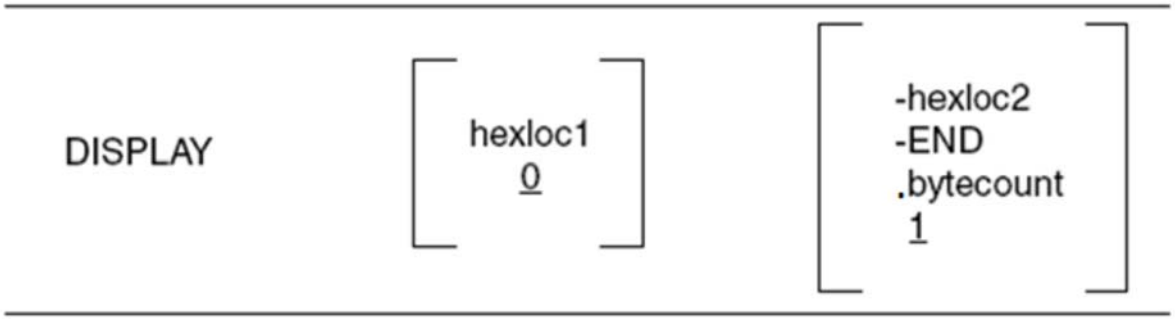
- The `DISPLAY` command is used to view from a terminal window the contents of memory locations. The command syntax is:



Requirement

- Brackets represent alternative optional operands.
- Capital letters represent operand keywords; lowercase letters represent operand values (i.e., actual values are to be substituted).
- Underlined operands represent the default values (i.e., the value used when the operand is omitted).

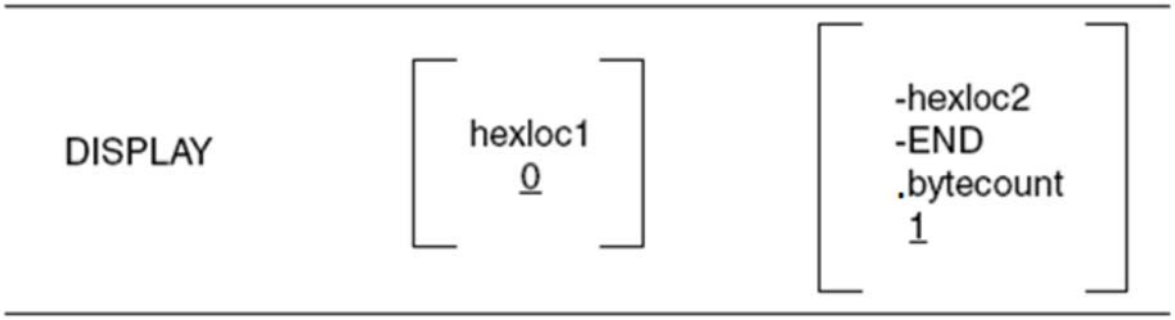
Example 1: `DISPLAY`
Example 2: `DISPLAY 77F`
Example 3: `DISPLAY 77F-407A`



Requirement: the first operand

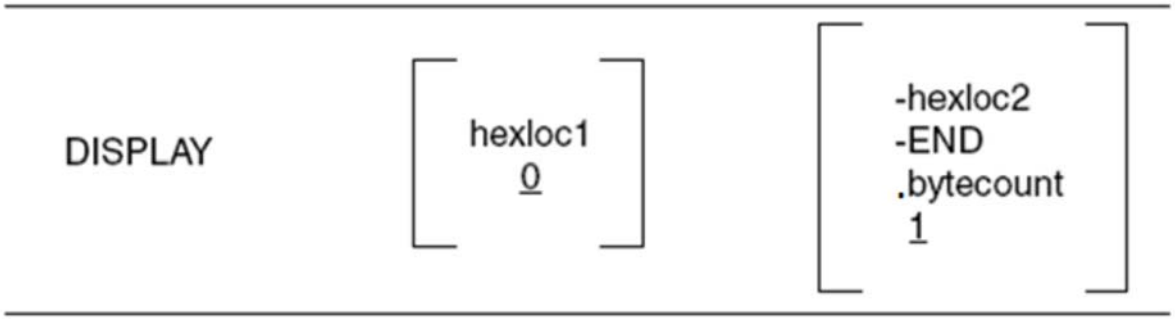
- The first operand (`hexloc1`) specifies the address of the first byte whose contents are to be displayed.
  - The address may be one to six hexadecimal digits (0–9, A–F) in length. If it is not specified, the address 0 is assumed.
  - The address must be within the actual memory range of the machine.

Example: `DISPLAY 77F-407A`
Example: `DISPLAY -407A`
Example: `DISPLAY`



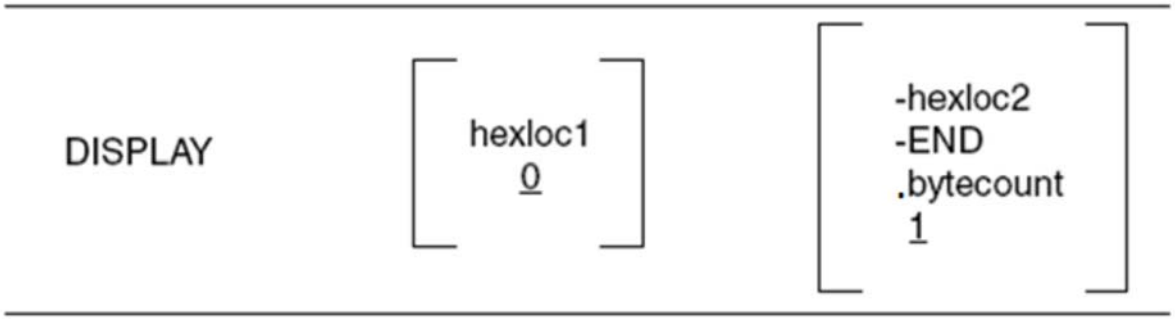
Requirement: the second operand

- The second operand specifies the amount of memory to be displayed.
  - If `hexloc2` is specified, it defines the address of the last byte in the range of locations to be displayed.
  - It may be 1~6 hexadecimal digits in length. The address must be greater than or equal to the starting address (`hexloc1`). Also, `hexloc2` must be within the actual memory range of the machine.
  - If `END` is specified, memory is displayed up through the last actual byte in the machine.



Requirement: the second operand

- The second operand specifies the amount of memory to be displayed.
  - If `bytecount` is specified, it defines the number of bytes of memory to be displayed (starting with the location specified in `hexloc1`).
  - The operand `bytecount` is a hexadecimal integer (one to six digits).
  - The sum of `bytecount` and `hexloc1` must not exceed the actual memory size plus 1, and `bytecount` must have a value of at least 1.



Requirement: Output

- When memory contents are displayed, the output format on the screen is one or more lines of the format

  ```
  xxxxxx = word1 word2 word3 word4
  ```

  where `xxxxxx` is the hexadecimal address of `word1`.

- An integral number of words (four-byte sequences, where the address of the first byte in the word is a multiple of four) is always displayed, regardless of the value of `hexloc1` or the amount of memory to be displayed.

- All output lines will always contain four words (16 bytes). The first byte of the displayed range will fall within the first word.

Requirement: Error Message

- The error messages that can be produced are:
  - M1: “invalid command syntax”
  - M2: “memory requested is beyond actual memory limit”
  - M3: “memory requested is a zero or negative range”

Example

DISPLAY

displays the first four words in memory (default starting address of 0, default byte count of 1)

```
0000000     0010  1211  4231  1A34
```

DISPLAY 77F

displays the word containing the byte at address 77F and the three subsequent words

```
000077c     312F  2F21  3DA2  2524
```

DISPLAY 77F.6

displays the words containing the six bytes starting at location 77F

```
000077c     312F  2F21  3DA2  2524
```

DISPLAY 77F-407A

displays the words containing the bytes in the address range 775- 407A

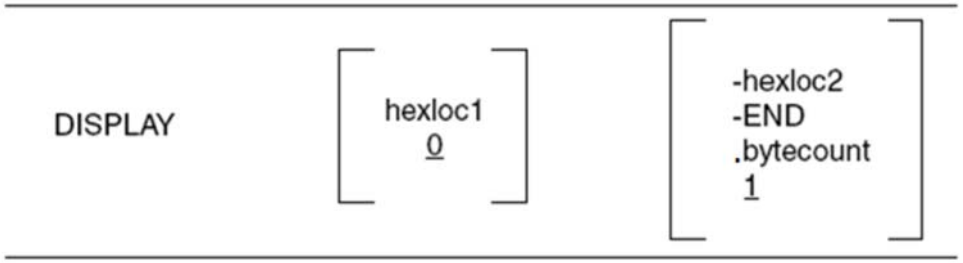
```
000077c     312F  2F21  3DA2  2524
000078c     F112  AE2F  3252  1233
              ...   ...
0004078     F112  AE2F  3252  1233
```

Identify the Causes

- Causes for operand 1
  - **C1**: First operand is present.
  - **C2**: The `hexloc1` operand contains only hexadecimal digits.
  - **C3**: The `hexloc1` operand contains one to six characters.
  - **C4**: The `hexloc1` operand is within the actual memory range of the machine.

The first operand (`hexloc1`*) specifies the address of the first byte whose contents are to be displayed.*

- *The address may be one to six hexadecimal digits (0–9, A–F) in length. If it is not specified, the address 0 is assumed.*
- *The address must be within the actual memory range of the machine.*



Identify the Causes

- Causes for operand 2
  - **C5**: Second operand is `END`.
  - **C6**: Second operand is `hexloc2`.
  - **C9**: The `hexloc2` operand contains only hexadecimal digits.
  - **C10**: The `hexloc2` operand contains one to six characters.
  - **C11**. The `hexloc2` operand is within the actual memory range of the machine.
  - **C12**. The `hexloc2` operand is greater than or equal to the `hexloc1` operand.

The second operand specifies the amount of memory to be displayed.

- *If* `hexloc2` *is specified, it defines the address of the last byte in the range of locations to be displayed.*
- *It may be 1~6 hexadecimal digits in length. The address must be greater than or equal to the starting address (*`hexloc1`*). Also,* `hexloc2` *must be within the actual memory range of the machine.*
- *If* `END` *is specified, memory is displayed up through the last actual byte in the machine.*

Identify the Causes

- Causes for operand 2
  - **C7**: Second operand is `bytecount`.
  - **C8**: Second operand is omitted.
  - **C13**. The `bytecount` operand contains only hexadecimal digits.
  - **C14**. The `bytecount` operand contains one to six characters.
  - **C15**. `bytecount` + `hexloc1` <=memory size + 1.
  - **C16**. `bytecount` >=1.

The second operand specifies the amount of memory to be displayed.

- *If* `bytecount` *is specified, it defines the number of bytes of memory to be displayed (starting with the location specified in* `hexloc1`*).*
- *The operand* `bytecount` *is a hexadecimal integer (one to six digits).*
- *The sum of* `bytecount` *and* `hexloc1` *must not exceed the actual memory size plus 1, and* `bytecount` *must have a value of at least 1.*
- *If operand 2 is not specified, the* `bytecount` *1 is assumed.*

Identify the Causes

- Causes identified from the output description
  - **C17.** Specified range is large enough to require multiple output lines.
  - **C18.** Start of range does not fall on a word boundary.

When memory contents are displayed, the output format on the screen is one or more lines of the format

```
xxxxxx = word1 word2 word3 word4
```

where `xxxxxx` *is the hexadecimal address of* `word1`*.*

An integral number of words (four-byte sequences, where the address of the first byte in the word is a multiple of four) is always displayed, regardless of the value of `hexloc1` *or the amount of memory to be displayed.*

All output lines will always contain four words (16 bytes). The first byte of the displayed range will fall within the first word.

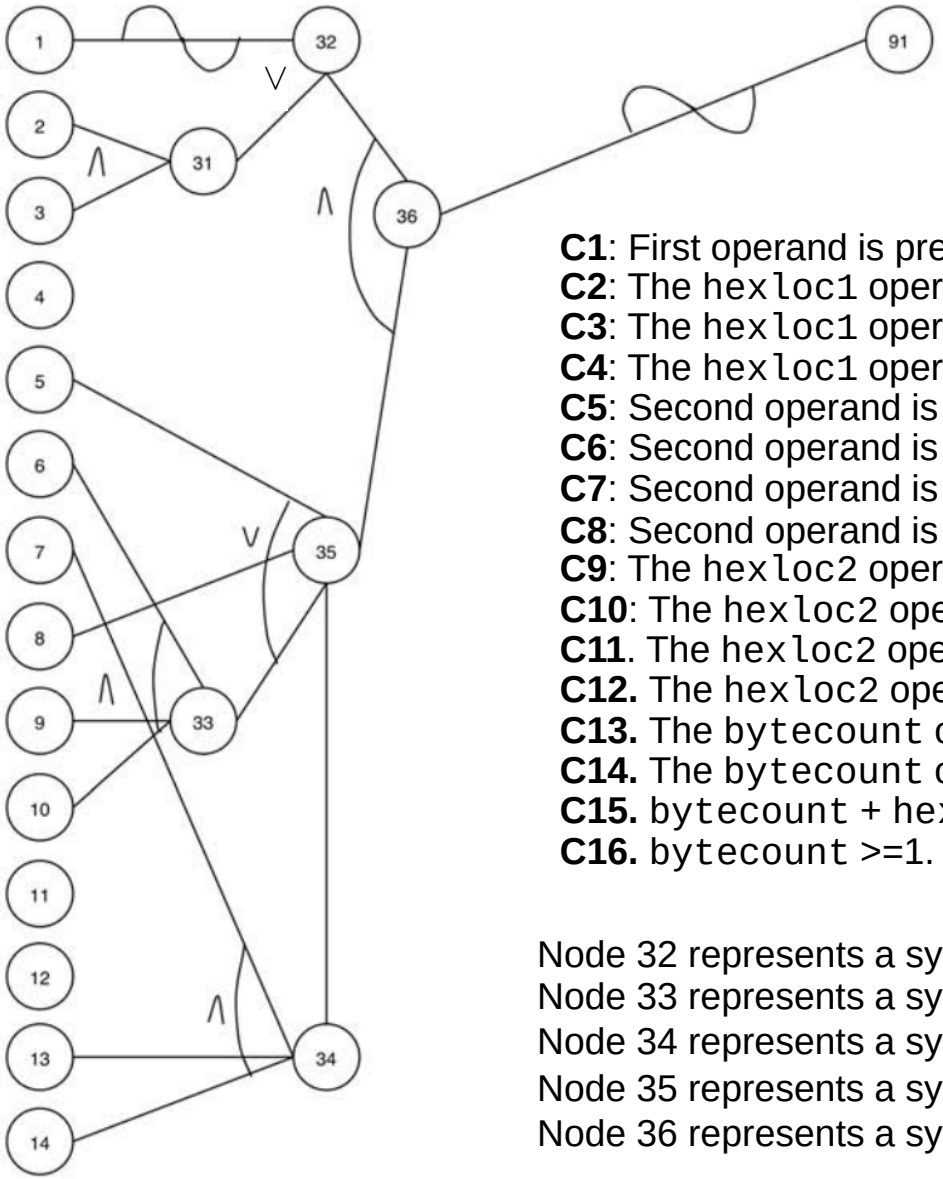
Identify the Effects

The effects:

- **E91.** Message M1 is displayed.
- **E92.** Message M2 is displayed.
- **E93.** Message M3 is displayed.

- **E94.** Memory is displayed on one line.
- **E95.** Memory is displayed on multiple lines.
- **E96.** First byte of displayed range falls on a word boundary.
- **E97.** First byte of displayed range does not fall on a word boundary.

Cause-Effect Graph: Partial



E91. Message M1 is displayed. "*invalid command syntax*"

C1: First operand is present.
C2: The `hexloc1` operand contains only hexadecimal digits.
C3: The `hexloc1` operand contains one to six characters.
C4: The `hexloc1` operand is within the actual memory range of the machine.
C5: Second operand is `END`.
C6: Second operand is `hexloc2`.
C7: Second operand is `bytecount`.
C8: Second operand is omitted.
C9: The `hexloc2` operand contains only hexadecimal digits.
C10: The `hexloc2` operand contains one to six characters.
C11. The `hexloc2` operand is within the actual memory range of the machine.
C12. The `hexloc2` operand is greater than or equal to the `hexloc1` operand.
C13. The `bytecount` operand contains only hexadecimal digits.
C14. The `bytecount` operand contains one to six characters.
C15. `bytecount` + `hexloc1` <=memory size + 1.
C16. `bytecount` >=1.

Node 32 represents a syntactically correct first operand
Node 33 represents a syntactically correct '`hexloc2`' second operand.
Node 34 represents a syntactically correct '`bytecount`' second operand.
Node 35 represents a syntactically correct second operand.
Node 36 represents a syntactically correct command

Cause-Effect Graph: All but Constraints

Example:

E95. Memory is displayed on one line.

E95 = ¬C17 ∧ N39 ∧ N40

C17: Specified range is large enough to require multiple output lines.
N39 represents request memory is within memory limit.
N40 represents request memory has a positive range.

N39 = N37 ∧ N38

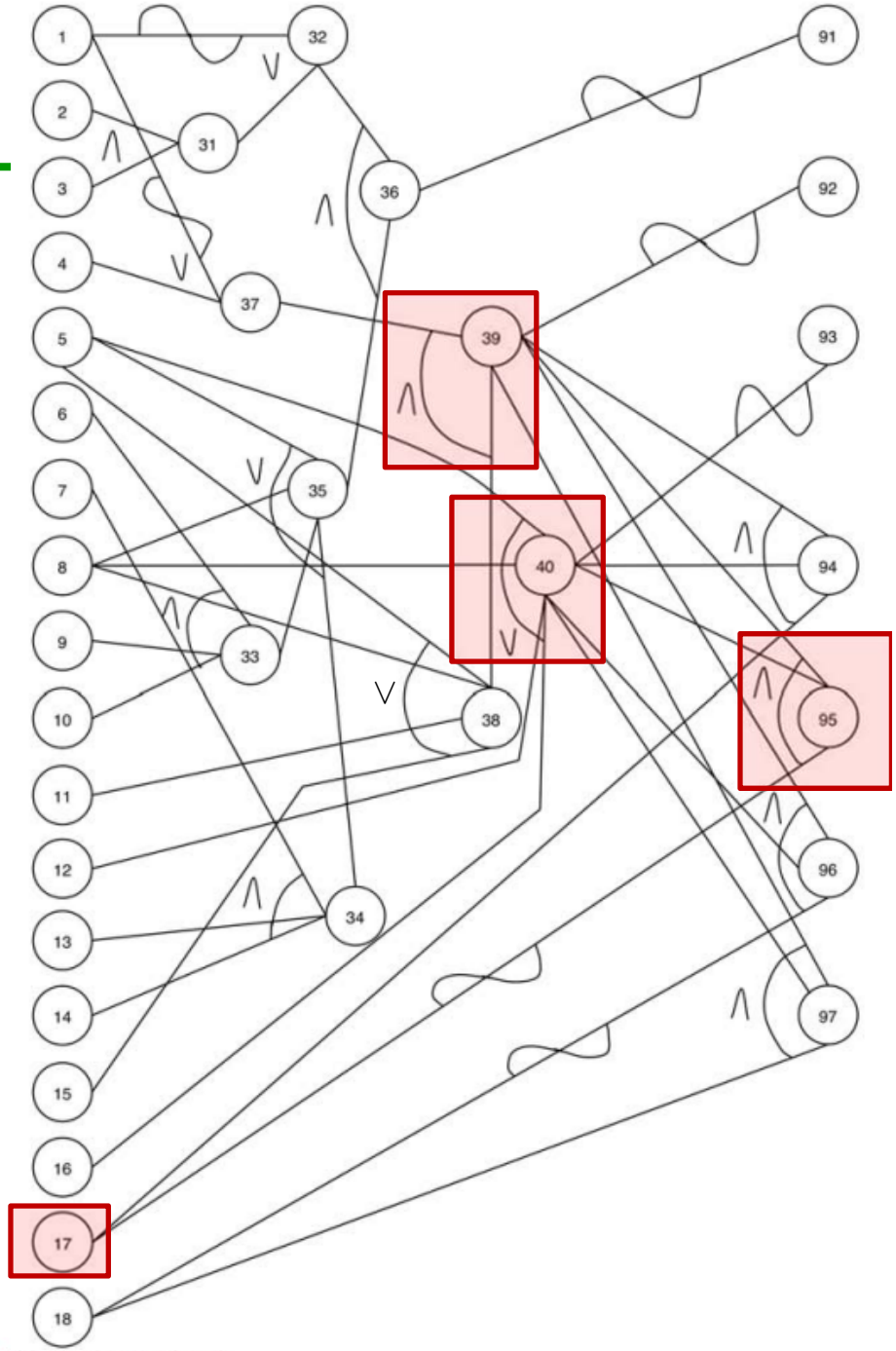
N37 represents valid first operand within range.
N38 represents valid second operand within range.

N37 = ¬C1 ∨ C4

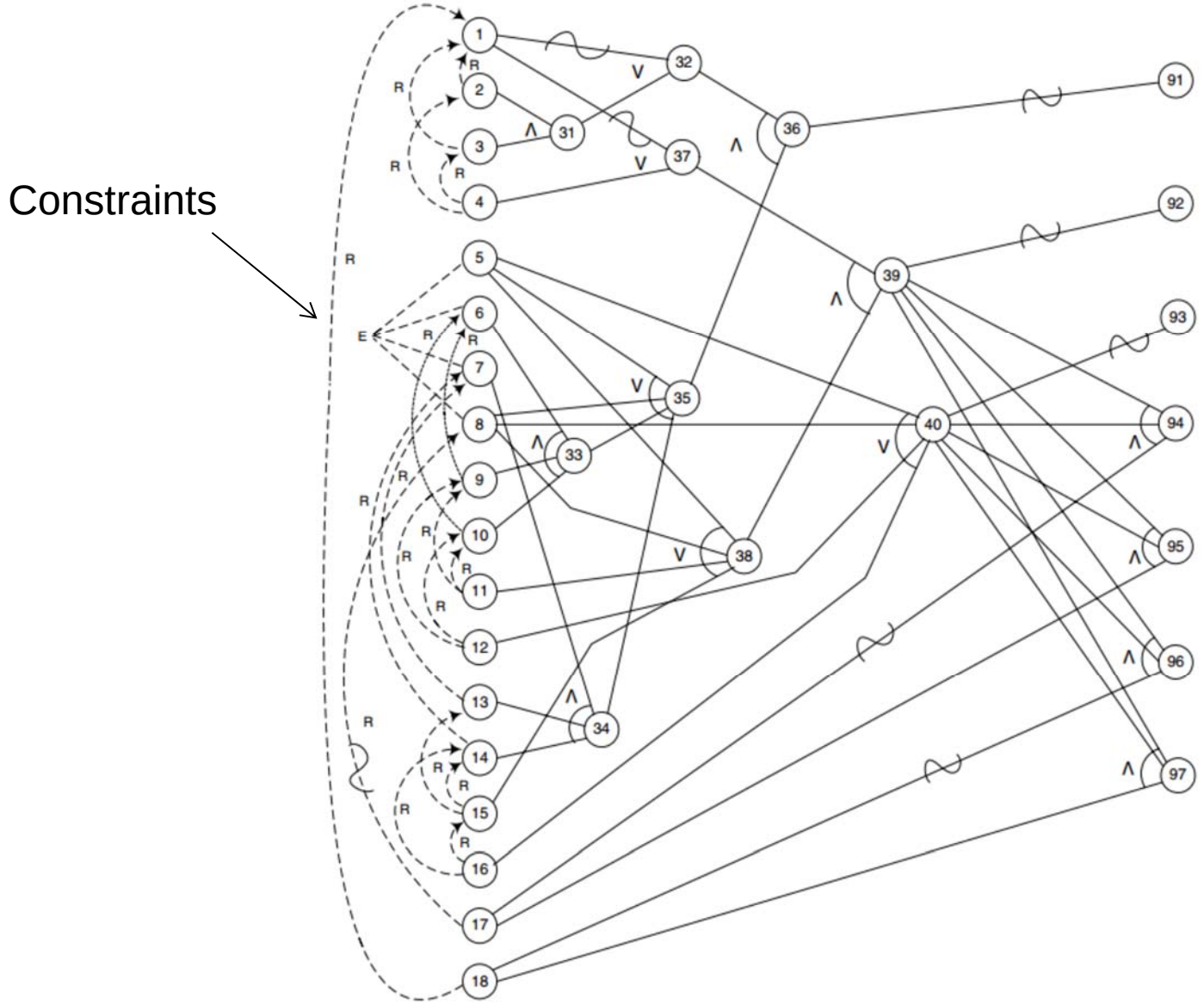
C1: First operand is present.
C4: The `hexloc1` operand is within the actual memory range of the machine.

N38 = C5 ∨ C8 ∨ C11 ∨ C15

C5: Second operand is `END`.
C8: Second operand is omitted.
C11. The `hexloc2` operand is within the actual memory range of the machine.
C15. `bytecount` + `hexloc1` <=memory size + 1.



Full Cause-Effect Graph



Decision Table

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	0	0	0	1	1	1
2	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1					1	1	1	1	1	1	1				1	1	1
3	0	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1					1	1	1	1	1	1	1				1	1	1
4												1	1	0	0	1	1	1	1	1	1					1	1	1	1	1	1	1				1	1	1
5				0										1				1				1				1				1			1			1		
6	1	1	1	0	1	1	1				1	1			1	1				1				1				1			1			1			1	
7				0				1	1	1			1				1				1				1				1			1			1			1
8				0															1				1				1										
9	1	1	1		1	0	0				0	1			1	1				1				1				1			1			1			1	
10	1	1	1		0	1	0				1	1			1	1				1				1				1			1			1			1	
11												0			0	1				1				1				1			1			1			1	
12																0				1				1				1			1			1			1	
13								1	0	0			1				1				1				1				1			1			1			1
14								0	1	0			1				1				1				1				1			1			1			1
15													0								1				1				1			1			1			1
16																	0				1				1				1			1			1			1
17																		0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
18																		1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
91	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
92	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
93	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
94	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
95	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1
97	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0

...And Test Cases

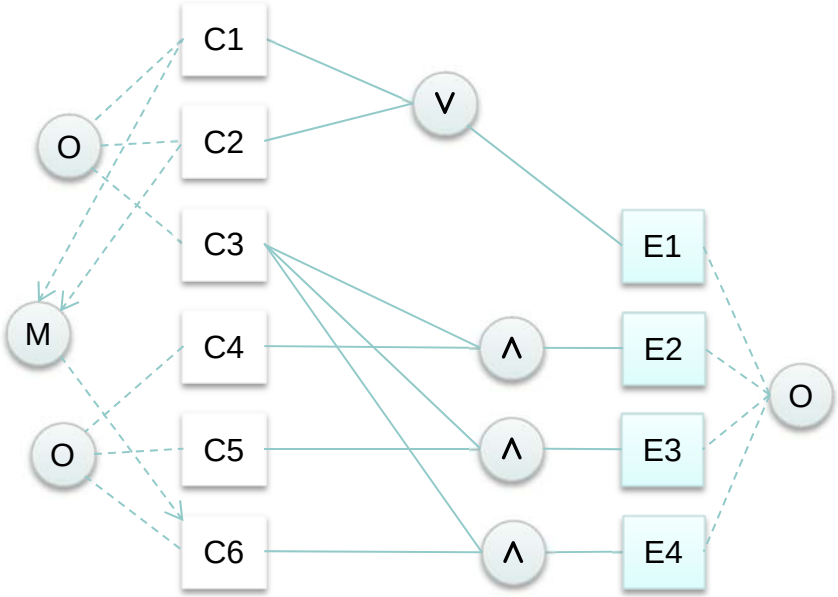
1	DISPLAY 234AF74-123	(91)	20	DISPLAY 21-29	(94, 97)
2	DISPLAY 2ZX4-3000	(91)	21	DISPLAY 4021.A	(94, 97)
3	DISPLAY HHHHHHHH-2000	(91)	22	DISPLAY –END	(94, 96)
4	DISPLAY 200 200	(91)	23	DISPLAY	(94, 96)
5	DISPLAY 0-22222222	(91)	24	DISPLAY –F	(94, 96)
6	DISPLAY 1-2X	(91)	25	DISPLAY .E	(94, 96)
7	DISPLAY 2-ABCDEFGHI	(91)	26	DISPLAY 7FF8-END	(94, 96)
8	DISPLAY 3.1111111	(91)	27	DISPLAY 6000	(94, 96)
9	DISPLAY 44.$42	(91)	28	DISPLAY A0-A4	(94, 96)
10	DISPLAY 100.$$$$$$$	(91)	29	DISPLAY 20.8	(94, 96)
11	DISPLAY 10000000-M	(91)	30	DISPLAY 7001-END	(95, 97)
12	DISPLAY FF-8000	(92)	31	DISPLAY 5-15	(95, 97)
13	DISPLAY FFF7001	(92)	32w	DISPLAY 4FF100	(95, 97)
14	DISPLAY 8000-END	(92)	33	DISPLAY –END	(95, 96)
15	DISPLAY 8000-8001	(92)	34	DISPLAY –20	(95, 96)
16	DISPLAY AA-A9	(93)	35	DISPLAY .11	(95, 96)
17	DISPLAY 7000.0	(93)	36	DISPLAY 7000-END	(95, 96)
18	DISPLAY 7FF9-END	(94, 97)	37	DISPLAY 4-14	(95, 96)
19	DISPLAY 1	(94, 97)	38	DISPLAY 500.11	(95, 96)

Quiz 3: Decision Table

- Input parameter: a list of three **positive** integers that represent the lengths of triangle edges.
- Output: the type of the triangle
  - invalid '非法' scalene '不等边' isosceles '等腰' equilateral '等边'
- Draw the cause-effect graph and the decision table.

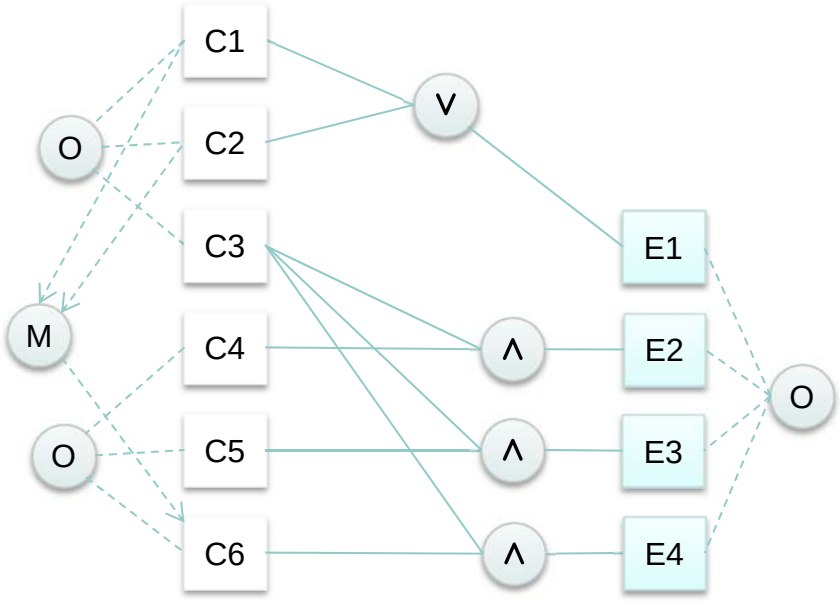
C1: sum of two edges smaller than the remaining one
C2: sum of two edges equal to the remaining one
C3: sum of two edges greater than the remaining one
C4: none of the edges have the same length
C5: two edges have the same length
C6: three edges have the same length

E1: invalid
E2: scalene
E3: isosceles
E4: equilateral



Quiz 3: Decision Table

- Input parameter: a list of three **positive** integers that represent the lengths of triangle edges.
- Output: the type of the triangle
  - invalid '非法' scalene '不等边' isosceles '等腰' equilateral '等边'
- Draw the cause-effect graph and the decision table.

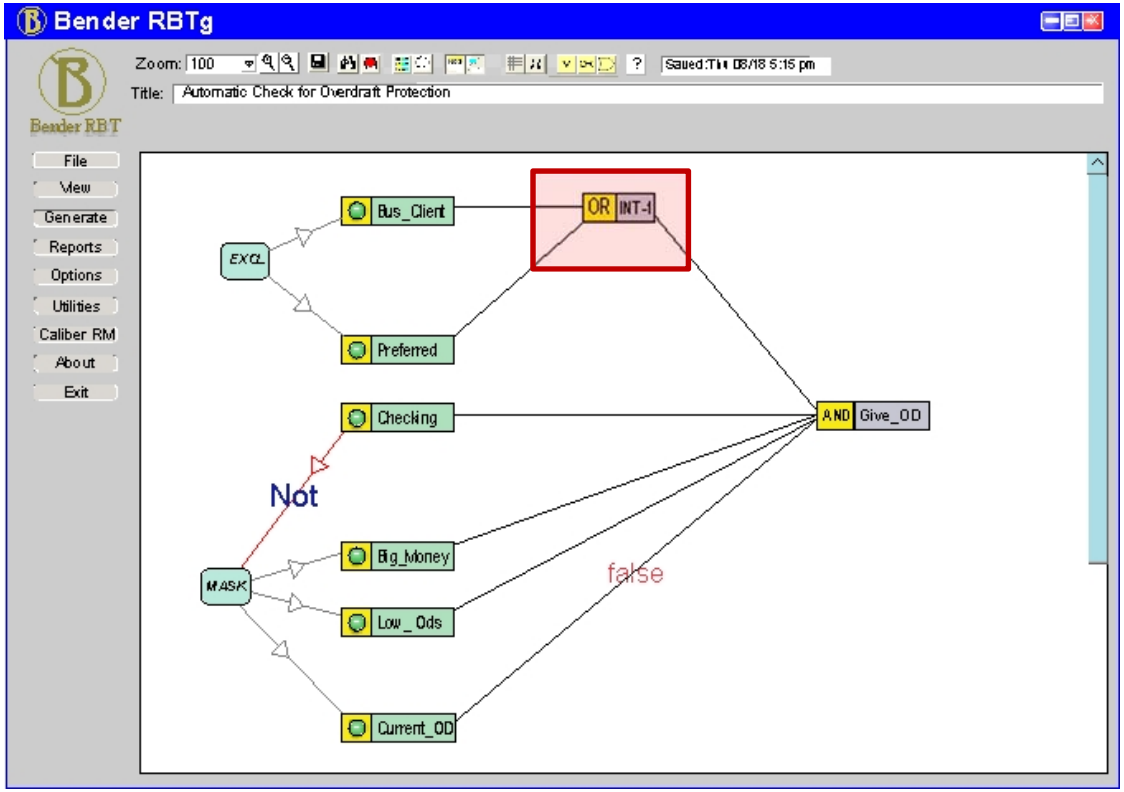


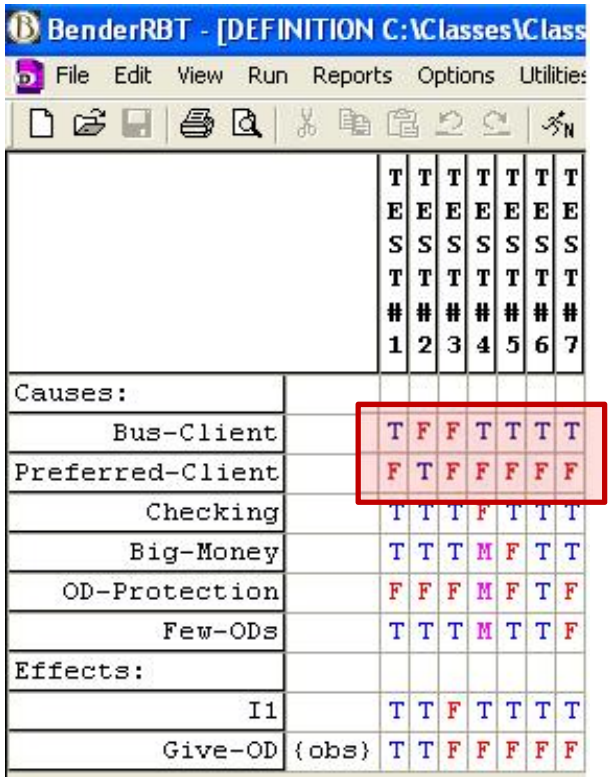
C1	**T**	F	F	F	F	F
C2	F	**T**	**T**	F	F	F
C3	F	F	F	**T**	**T**	**T**
C4	**T**	F	F	**T**	F	F
C5	F	**T**	F	F	**T**	F
C6	F	F	**T**	F	F	**T**
E1	•	•	•			
E2				•		
E3					•	
E4						•

Tool: BenderRBT

http://benderrbt.com

- Support the drawing of cause-effect graph and the generation of decision table.
- Measure Cause-effect Coverage of the test suite





Thank you!