Cause Effect Graph to Decision Table Generation

Praveen Ranjan Srivastava[1], Parshad Patel[2], Siddharth Chatrola[3]
Computer Science and information system Group
Birla Institute of Technology and Science
Pilani , Rajasthan-333031

praveenrsrivastava@gmail.com

Abstract-Cause-Effect Graphing (CEG) is used to identify test cases from a given specification to validate its corresponding implementation. This paper gives detail about this technique of software testing. It also shows how the CEG technique can be used to test that software fulfill requirement specification or not. This paper surveys how CEG converted into decision table. The aim of this paper is to overcome existing algorithm's shortcomings and generate all possible test cases.

Keywords -*Cause effect graphing (CFG), Requirement specification.*

I. INTRODUCTION

Testing is generally described as a group of procedures carried out to evaluate some aspect of a piece of software. Also Testing can be described as a process used for revealing defects in software, and for establishing that the software has attained a specified degree of quality with respect to selected attributes. They cover both validation and verification activities, and include in the testing domain all of the following: technical reviews, test planning, test tracking, test case design, unit test, integration test, system test, acceptance test, and usability test. Testing is dual-purpose Process-one that reveals defects, as well as one that is used to evaluate quality attributes of the software such as reliability, security, usability, and correctness [1].

White Box Testing, where we focus on developing test cases to cover the logical paths through the code (e.g., conditional statements, loops, etc.) [1,2]

Testing techniques consist of Black Box testing, where we focus on testing the software functional requirements, and testing the input/output interfaces. Black Box testing have Equivalence Partitioning Partition the input space into equivalence classes and develop a test case for each class of inputs. Boundary value analysis develops test cases at the boundaries of the possible input ranges (minimum and maximum values). The above techniques are important for data processing intensive applications. Cause-effect graphing used for control intensive applications, Develops test cases to represent input events (or causes) and the corresponding actions (or effects) [1,2,3].

II. CAUSE-EFFECT GRAPHING

Cause-Effect Graphing is basically a hardware testing technique adapted to software testing and further developed by others [1, 2]. The CEG technique is a black-box method, i.e., it considers only the desired external behavior of a system. As well, it is the only black-box test design technique that considers combinations of causes of system behaviors.

In CEG analysis, first, identify causes[*], effects[**] and constraints[***] in the (natural language) specification. Second, construct a CEG as a combinational logic network which consists of nodes, called causes and effects, arcs with Boolean operators (and, or, not) between causes and effects, and constraints. Finally, trace this graph to build a decision table which may be subsequently converted into use cases and eventually, test cases [4].

Cause-effect graphing, also known as dependency modeling , focuses on modeling dependency relationship among program input conditions known as output condition known as effects. The relationship is expressed visually in terms of cause-effect graph. [5] The graph is visual representation of a logical relationship among 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 [1].

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.

The following generic procedure is used for the generation of tests using cause-effect graphing. 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 s cause –effect graph. Transform the cause-effect graph into a limited entry decision table, hereafter referred to simply as decision table. Generate tests from the decision table [3].As with causes, examining the specification, or other similar artifact, word-by-word and underlining words or phrases that describe outputs or system transformations helps to identify the effects[5].

* A cause represents a distinct input condition or an equivalence class of input conditions. A cause can be interpreted as an entity which brings about an internal change in the system. In a CEG, a cause is always positive and atomic.

** An effect represents an output condition or a system transformation which is observable. An effect can be a state or a message resulting from a combination of causes.

***Constraints represent external constraints on the system.

1. Phd student (BITS PILANI)
2, 3. Master of Technology student. (BITS PILANI)

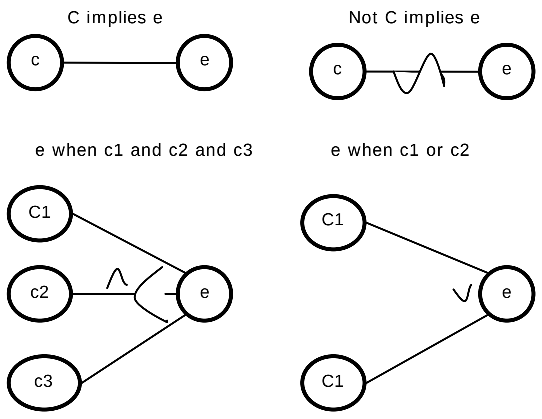


Figure 1. Notation

Constraint Symbol:

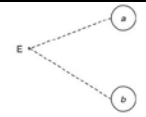
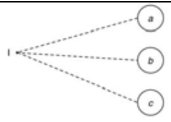
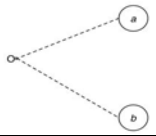
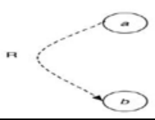
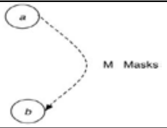
Constraint Symbol	Definition
	**The "E"(Exclusive) constraint states that one and only one of the causes a and b can be true.**
	**The "I" (Inclusive (at least one)) constraint states that at least one of the causes a, b and c must always be true (a, b and c cannot be false simultaneously).**
	**The "O"(One and Only One) constraint states that one and only one of the causes a and b can be true.**
	**The "R"(Requires) constraint states that for cause a to be true, than cause b must be true. In other words, it is impossible for cause a to be true and cause b to be false.**
	**The "M"(Mask) constraint states that if effect x is true; effect y is forced to false. (Note that the mask constraint relates to the effects and not the causes like the other constraints)**

Figure 2. Constrain Symbols

Consider the following set of requirements as an example:

- Display message A if input one and two.
- Display message B if input two and three.
- Display message C if input one and three.
- Display message D if input one and two and flag is set.
- Only one possible either Flag is set or input is three.

When examining these requirements, we see that there are two variables that will determine which are inputs. If inputs are one and two the message A is displayed that is cause1 then effect E1. If inputs are two and three the message B is displayed that is cause2 then effect E2 and so on. Here O (One and Only One) constraint between cuase3 and cause4 that is only one true at time. Also when e1 is true then effect 3 is true. Shown in Table 1 below, each cause and each effect is assigned an arbitrary unique number as part of this process step. [5]

TABLE 1 CAUSE AND EFFECT

Causes(input conditions)	**Effects(output conditions)**
C1: Input One	E1: Message A is displayed.
C2: Input Two	E2: Message B is displayed.
C3: Input Three	E3: Message C is displayed.
C4: Flag is set.	E4: Message D is displayed.

Semantics, in this step's instructions, reflect the meaning of the programs or functions. This meaning is discerned from the specification and transformed into a boolean graph that maps the causes to the resulting effects. It is easier to derive the boolean function for each effect from their separate CEGs, then these can be overlaid to produce a single decision table for all effects (see step 5). For example, the following separate CEGs (see Table 2) can be derived from above example.

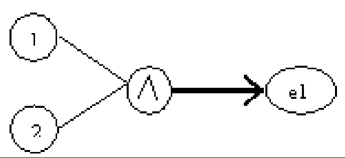
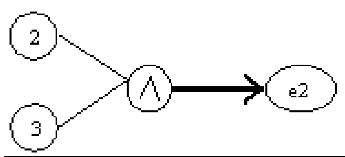
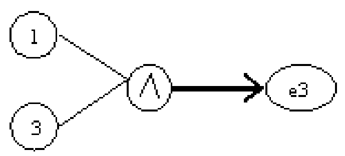
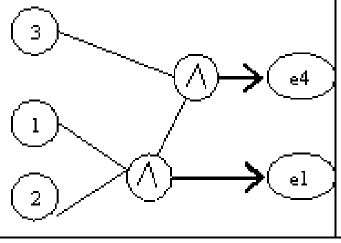
Constraint Symbol	Definition
CEG#1: 	**Causes :** **1. Input is one** **And** **2. Input is two** **Effect:** **e1: Display message A.**
CEG#2: 	**Causes :** **2. Input is two** **And** **3. Input is three** **Effect:** **e2: Display message B.**
CEG#3: 	**Causes :** **1. Input is one** **And** **3. Input is three** **Effect:** **e3. Display message C.**
CEG#4: 	**Causes :** **1. Input is one** **And** **2. Input is two** **&** **3. Flag is set** **Effect:** **e1: Display message A.** **e4. Display message D.**

Figure 3. Cause Effect Graph

The ones (1) in the limited entry decision table column indicate that the cause (or effect) is true in the CEG and zeros (0) indicate that it is false. Table 4 below illustrates the limited-entry decision table created by converting the CEG from the above example. For example, the CEG #1, from Table 2 in step3, converts into test case column 1 in the table below. From CEG 1, causes 1 and 3 being true result in effect 101 being true. [6]

Table 2 – DECISION TABLE

c1	0	1	1	1	1	1
c2	1	1	0	1	1	1
c3	1	0	1	0	1	1
c4	0	0	0	1	0	1
e1	0	1	0	1	1	1
e2	1	0	0	0	1	1
e3	0	0	1	0	1	1
e4	0	0	0	1	0	1

III. BACKGROUND WORK

Myers proposed a CEG tracing algorithm to systematically select a high-yield 4 set of test cases [1]. Myers' approach has some ambiguities. Some investigates Myers' approach; illustrate strengths and weaknesses, and present possible corrections [4, 7, 8]. But this solution is not efficient.

In recent works [3], first of all each effect in CFG are traced backward and combination of causes are identifies and this all entry put up in decision table. At the end of this procedure numbers of columns give number of tests generated.

This algorithm [3] has up to our point of view are suffering some of drawbacks. There are some cases where two or more effects simultaneously true, these combinations are not derived by this algorithm .Complexity($O(n^3)$) of this algorithm is also very high.

This paper gives extension of algorithm by overcoming drawbacks of algorithm and generates all possible test cases in efficient way.

IV. PROPOSED APPROCH

Proposed algorithm: Procedure for generating a decision table from cause-effect a graph.

Input: (a) A template containing causes C_1, C2…$C_{p,}$ effects E_1, E2... $Eq_{, their}$ relationship and Constrains.

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.

Procedures: CEG_TO_DT

```
Step 1: Read causes and effects from template and for each of the effect construct binary tree that specifies that effect in terms of cause's relationship.
Step 2: Initialize DT to an empty decision table.
Step 3: For each of the $E_i$ create decision table $EDT_i$ as empty decision table.
Step 4 Execute the following steps for i=1 to q.
        4.1 Select the next effect to be processed.
        Let $e=E_i$ .
        4.2 Starting at e, traverse the effect tree created in previous step and identify the causes $C_1$, C2… etc that are there in tree. Identify all possible combination of these causes that lead e to be present. Make sure that the combinations satisfy any constraints.
        4.3 Add all these possible entries into the decision t able $EDT_{i.}$
Step 5 Do OR-Operation between all effects that gives all combination of effects. For all these effects combination of effects do following.
        5.1 Select the decision tables $EDT_i$ for all effects that are present in this combination.
        5.2 Take all possible combination of entries of different tables. In this combination if there is a cause that is there in two or more effect decision table then take combinations in which cause values are same.
        5.3 Enter all these entries into decision table that satisfy all constrain.
```

End of Procedures: CEG_TO_DT

V. WORKING OF PROPOSED ALGORITHM

First for each effect binary tree is generated. Each effect binary tree is traced backwardly and possible combinations of causes are identifies. Each entry is entered in individual decision table of each effect. After that binary tree is generated for effects with OR operation between effects. Now this binary tree traced backwardly and we will get all possible effect combination. For each possible combination of effects, we take all possible combination of test cases of their decision table and get test cases. This purposed algorithm take care of all combination of causes and as well as of effects. Simultaneously true effect cases also considered.

For example, there are two effects e1 and e2.c1, c2, c3, c4 are four causes. Effect e1 is true if c1 and c2 is true. Effect e2 is true if either c3 is true or c2 is true or both are true. So binary tree for e1, e2 and binary tree for effects are as follows

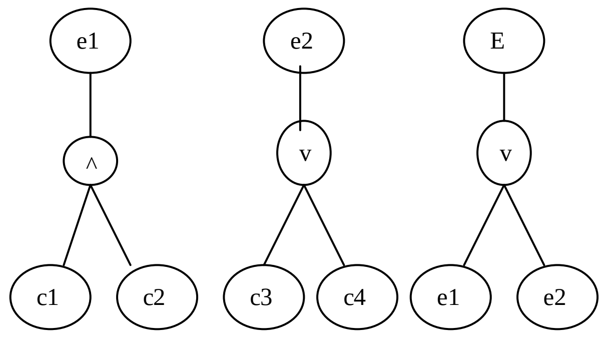


Figure 4

Now by traversing effect e1 an e2 tree backward we got following two decision table of e1 and e2 as shown below.

Table 3 – DECISION TABLE FOR e1 AND e2

c1	1
c2	1
e1	1

c3	0	1	1
c4	1	0	1
e2	1	1	1

Now by traversing last tree for effects combination we get three combination (e1,e2) = {(0,1),(1,0,),(1,1)}. For each of these combinations, we get different combination of entries of decision table. So final decision table is as follows.

Table 4 – DECISION TABLE GENERATED BY PROPOSED ALGORITHM

c1	1	0	0	0	**1**	**1**	**1**
c2	1	0	0	0	**1**	**1**	**1**
c3	0	0	1	1	**0**	**1**	**1**
c4	0	1	0	1	**1**	**0**	**1**
e1	1	0	0	0	**1**	**1**	**1**
e2	0	1	1	1	**1**	**1**	**1**

VI. ANALYSES OF PREVIOUS AND PROPOSED ALGORITHM

In purposed algorithm for each effect binary tree is generated so complexity (($O(n^2)$) for finding all possible combination is less in comparison with previous algorithm [3]. Algorithm given [3] is not generating all possible test cases in all case whereas our algorithm generates all possible test cases in all the cases. Decision Table show in Table 4 is generated by proposed algorithm which has all possible test cases where as Decision Table generated according to previous algorithm[3] has less number of test cases. Test cases shown in bold are not generated by previous algorithm [3], which are important test cases.

VII. CONCLUSION

In this paper we have shown lack ness of the current existing algorithm. This paper gives extension for that. Several CEG tools exist presently on the market, for example, the softest tool [9].However, their objective is to derive test cases from the cause-effect graph in order to validate an implementation. Our goal is to validate the informal specification as well. We developed a tool to derive the decision table and the customer-directed scenarios automatically from a given Cause-Effect Graph. The CEG is input in a text format; our tool gives outputs the decision table and the customer-directed scenarios. However, it does possess some drawbacks too. The main drawback is probably the up-front cost of deriving causes, effects, and constraints from a given informal specification. This process requires domain knowledge and training in the CEG method. However, these up-front costs are small compared to the potential major downstream savings because they avoid unnecessary rework and operational problems.

We will try to further reduce the complexity of algorithm. Apply Fuzzy logic and other heuristic approach to reduce number of test cases.

REFERENCES

[1] Myers, G. L., "The Art of Software Testing", Wiley-Interscience, New-York.
[2] Ilene Burnstein , “Practical Software Testing”
[3] Aditya P. Mathur , “Software Testing”, 1st Edition, Pearson Publication 2008.
[4] Khenaidoo Nursimulu, Robert L. Probert, “Cause-Effect Graphing Analysis and Validation of Requirements”
[5] The Westfall Team , “Cause-Effect Graphing By Theresa Hunt”
[6] Cai Ferriday , “A Review Paper on Decision Table-Based Testing”
[7] Kuo-Chung Tai 1, Amit Paradkar 2, Hsun-Kang Su, and Mladen A. Vouk , “Fault—Based Test Generation for Cause—Effect Graphs”.
[8] Amit Paradkar, K.C. Tai and M.A. Vouk, “Specification-based testing using cause-effect graphs”
[9] William R. Elmendorf , “Automated Design of Program Test Libraries”