3. Test Generation Strategies IV

Based on requirements

Andrea Polini

Software Engineering II – Software Testing
MSc in Computer Science
University of Camerino

November 13th, 2014

Cause-Effect Graphing

CEG aka Dependency Modeling

The very general idea is to make explicit the relation among input conditions (causes) and output conditions (effects) and to exploit such relations for testing purposes.

An effect is not necessarily visible to the external user, while it can be retrieved introducing suitable probes (test points)

Example

The LED close to the product description should be switched on when the credit becomes greater then the price of the snack

Cause-Effect Graphing

CEG aka Dependency Modeling

The very general idea is to make explicit the relation among input conditions (causes) and output conditions (effects) and to exploit such relations for testing purposes.

An effect is not necessarily visible to the external user, while it can be retrieved introducing suitable probes (test points)

Example

The LED close to the product description should be switched on when the credit becomes greater then the price of the snack

Cause-Effect Graphing

CEG aka Dependency Modeling

The very general idea is to make explicit the relation among input conditions (causes) and output conditions (effects) and to exploit such relations for testing purposes.

An effect is not necessarily visible to the external user, while it can be retrieved introducing suitable probes (test points)

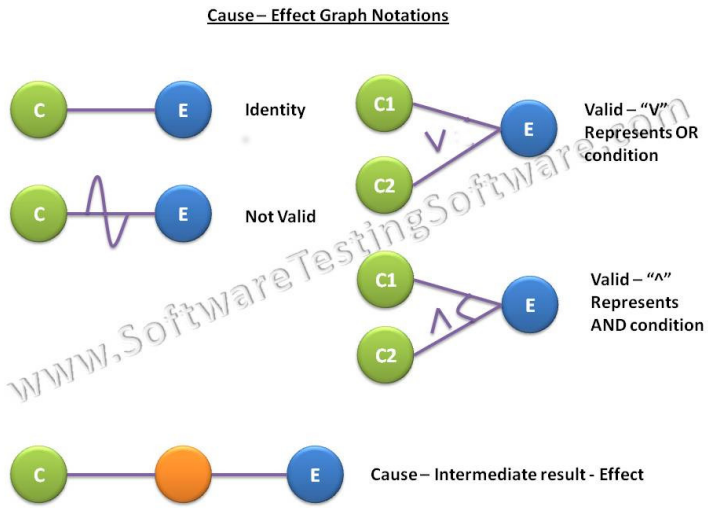
Example

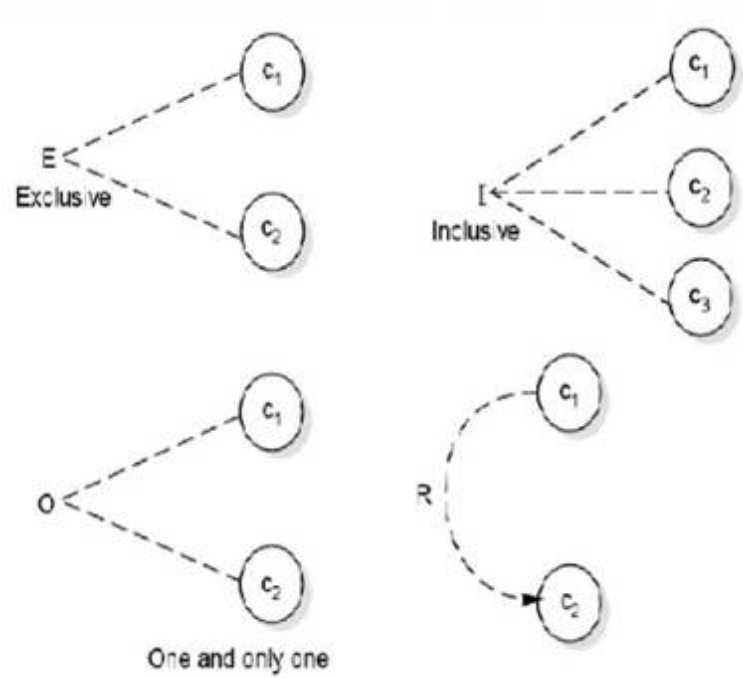
The LED close to the product description should be switched on when the credit becomes greater then the price of the snack

Test generation from CEG

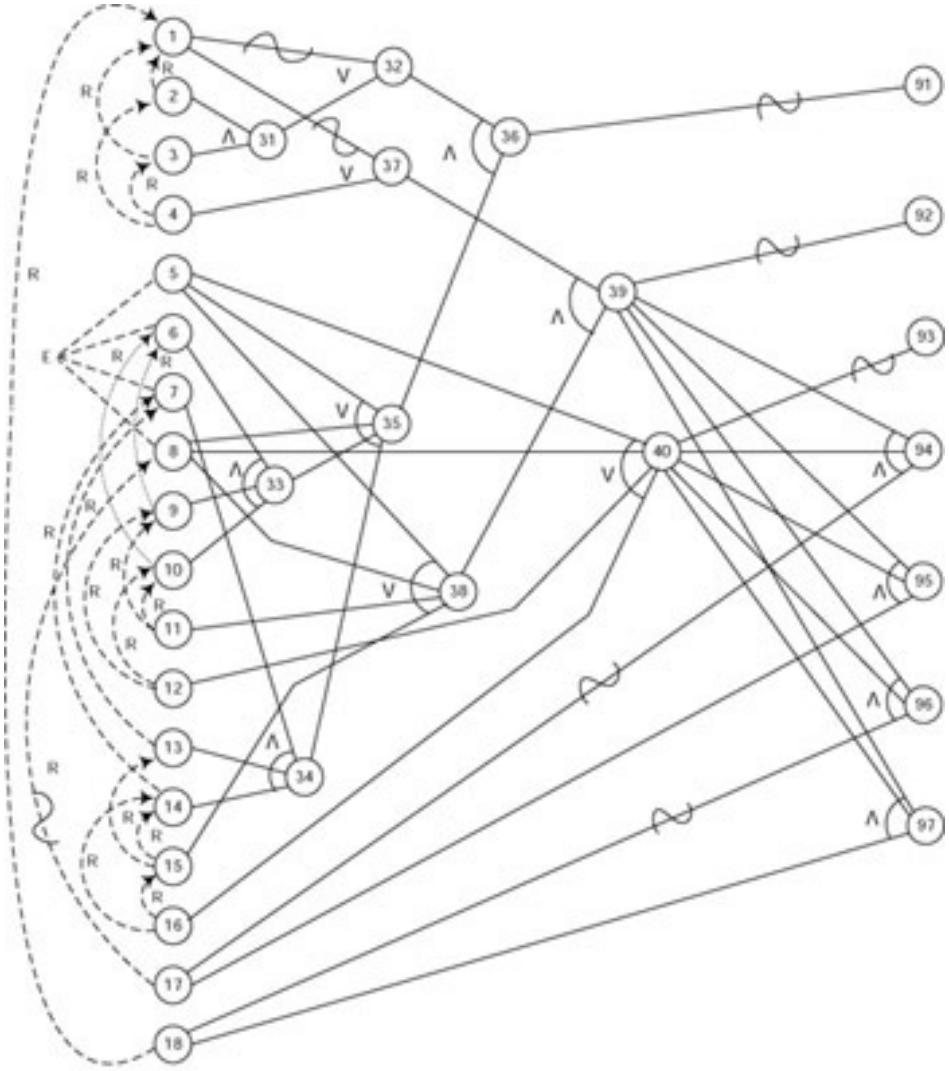
- Identify cause and effects reading the requirements
- Express the relationship between causes and effects using a CEG
- Tranform the CEG into a decision table
- Generate tests from the decision table

CEG Notation





Example


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
31
32
33
34
35
36
37
38
39
40
91
92
93
94
95
96
97
R
∨
∧

DT from a CEG

Decision tables

For each cause and effect use a row and put test as columns of the matrix. Each entry in the decision table is a 0 or a 1 depending on whether or not the corresponding condition is false or true, respectively.

How to derive a DT

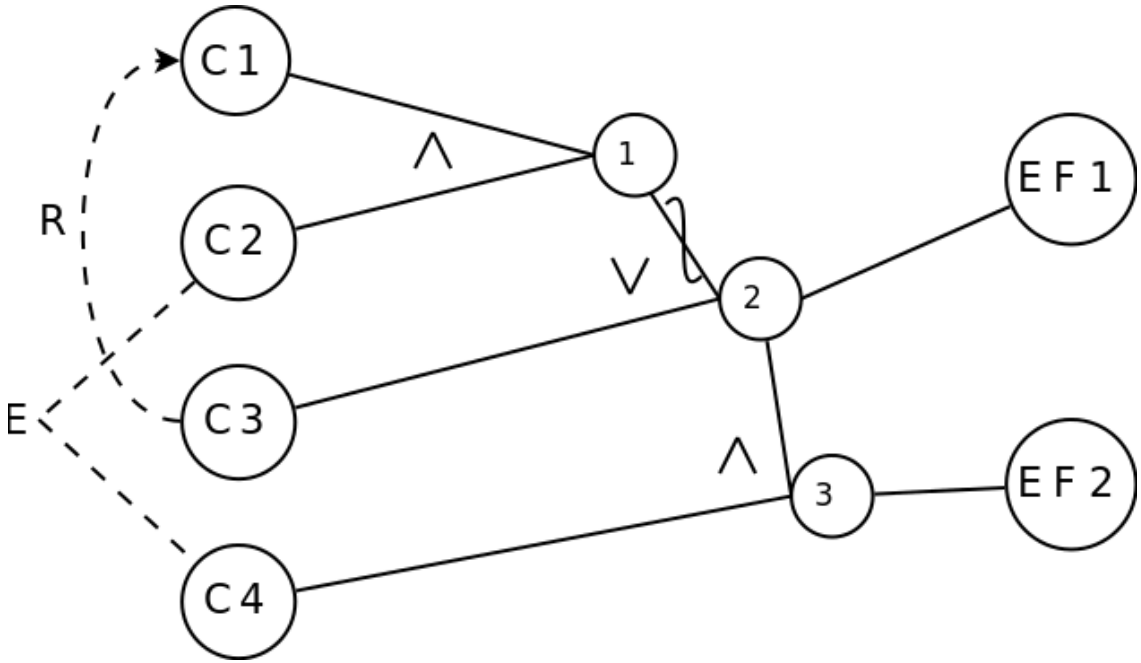
input: A CEG containing causes $C_1, C_2, ..., C_p$ and effects $Ef_1, Ef_2, ..., Ef_q$
output: A decision table containing $p + q$ rows and M columns where M depends on relationship between causes and effects.
Step1: Initialize DT to an empty DT
Step2: Execute the following steps for i=1 to q

2.1 Select the next effect

2.2 Find combinations of conditions that cause e to be present and store the m generated vector

2.3 update the decision table adding the generated vectors

DT derivation



Heuristic to avoid combinatorial explosion

The described approach could lead to exponential generation on the number of tests with respect to causes.

- For or relations: enemerate just those situations in which two causes are both false or one of them true
- For and relations: enumerate those situations ifor which causes assume different values, and those in which all of them are true.

Test generation

Tests from a DT

Each column of the DT constitutes the source for generating tests. Consdier that each condition could be satisfied by more assignement to the variable leading to the generation of more than one test for each column

Test generation from predicates

Predicate testing

if the printer is ON and has paper then send the document for printing

`pr:(printer_status=ON) ∧ printer_tray=!empty`

Consider the following predicate:

$$(a < b) \vee (c > d) \wedge e$$

The following test:

$$t = (a = 1, b = 2, c = 4, d = 2, e = true)$$

results in

$$p(t) = true$$

Fault model

Which kind of faults are generally targetted:

- Boolean operator fault
  - incorrect boolean operator used
  - negation missing or placed incorrectly
  - parentheses are incorrect
  - incorrect Boolean variable used
  - missing or extra Boolean variable
- relational operator fault
  - incorrect relational operator is used
- arithmetic expression fault
  - arithmetic expression is off by an amount equal to ϵ (off-by-ϵ,off-by-ϵ^+,off-by-ϵ^*)

Objective of predicate testing

To generate a test set $\mathcal{T}$ such that there is at least one test cast $t \in \mathcal{T}$ for which p_c and its faulty version p_i are distinguishable

Fault model

Which kind of faults are generally targetted:

- Boolean operator fault
  - incorrect boolean operator used
  - negation missing or placed incorrectly
  - parentheses are incorrect
  - incorrect Boolean variable used
  - missing or extra Boolean variable
- relational operator fault
  - incorrect relational operator is used
- arithmetic expression fault
  - arithmetic expression is off by an amount equal to ϵ (off-by-ϵ,off-by-ϵ^+,off-by-ϵ^*)

Objective of predicate testing

To generate a test set $\mathscr{T}$ such that there is at least one test cast $t \in \mathscr{T}$ for which p_c and its faulty version p_i are distinguishable

Predicate testing criteria

Three common criteria:

- BOR (Boolean Operator): A test set $\mathscr{T}$ that satisfied the BOR-testing criterion for a compound predicate p_r, guarantees the detection of single or multiple Boolean operator faults in the implementation of p_r. $\mathscr{T}$ is referred to as a BOR-adequate test set and sometimes written as $\mathscr{T}_{BOR}$.
- BRO (Boolean and relational Operator): A test set $\mathscr{T}$ that satisfied the BRO-testing criterion for a compound predicate p_r, guarantees the detection of single or multiple Boolean operator and relational operator faults in the implementation of p_r. $\mathscr{T}$ is referred to as a BRO-adequate test set and sometimes written as $\mathscr{T}_{BRO}$.
- BRE (Boolean and relational expression): A test set $\mathscr{T}$ that satisfied the BRE-testing criterion for a compound predicate p_r, guarantees the detection of single or multiple Boolean operator, relational operator and arithmetic expression faults in the implementation of p_r. $\mathscr{T}$ is referred to as a BRO-adequate test set and sometimes written as $\mathscr{T}_{BRE}$.

BOR example

Let $p_r : a < b \wedge c > d$ and $\mathscr{S}$ constraints on p_r where $\mathscr{S} = \{(\mathbf{t}, \mathbf{t}), (\mathbf{t}, \mathbf{f}), (\mathbf{f}, \mathbf{t})\}$ the following test set $\mathscr{T}$ satisfies constraint set $\mathscr{S}$ and the BOR-testing criterion:

$$\begin{aligned}\mathscr{T} = \{ & t_1 :< a = 1, b = 2, c = 1, d = 0 >; \\ & t_2 :< a = 1, b = 2, c = 1, d = 2 >; \\ & t_3 :< a = 1, b = 0, c = 1, d = 0 >; \\ \}\end{aligned}$$

Generating BOR.BRO,BRE adequate tests

There exist algorithms for the generation of adequate tests given constraints on the predicate. They are based on the definition of:

- Cartesian product of sets
- *onto* set product operator
- $AST(p_r)$