4. Test Generation – Predicate Analysis

Andrea Polini

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

Cause-Effect Graphing

CEG aka Dependency Modeling

The very general idea is to make explicit, also through a graphical representation, the relation among input conditions (causes) and output conditions (effects) and to exploit such relations for testing purposes.
In any case the relation can be fruitfully represented by a boolean expression

Cause and effects

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

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, also through a graphical representation, the relation among input conditions (causes) and output conditions (effects) and to exploit such relations for testing purposes.
In any case the relation can be fruitfully represented by a boolean expression

Cause and effects

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

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, also through a graphical representation, the relation among input conditions (causes) and output conditions (effects) and to exploit such relations for testing purposes.
In any case the relation can be fruitfully represented by a boolean expression

Cause and effects

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

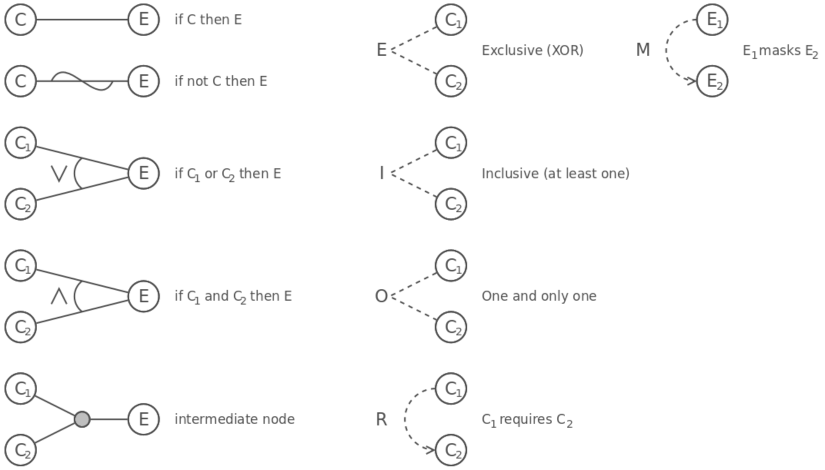
The LED close to the product description should be switched on when the credit becomes greater then the price of the snack

Test generation from CEG

CEG and test generation

- 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



Creating a CFG

Process

To create a CFG follow the process below:

- carefully identify causes and effects from a thoughtful analysis of the requirements.
- assign to each cause and each effect a unique identifier
- represent the identified relations in a CFG

Example

Computer purchase system

A web based company sells computers (CPU), printers (PR), monitors (M), and additional memories (RAM).

Conditions: For each order the buyer may select from 3 CPU, 2 PR, 3M. RAM one unit in one order. M20 and M23 any CPU or as stand alone. M30 only with CPU 3. PR 1 is available free with CPU 2 or 3. M and PR can pbe purchased as stand alone. Non M30. CPU 1 gets RAM 256 upgrade. CPU 2 o 3 gets RAM 512 upgrade. RAM 1G upgrade and free PR2 available if CPU 3 purchased with M30.

There is a window to make selection with menus in particular a widget displaying the free item available and a “Price” widget reports the calculation related to prices.

Causes are the purchase of the items

Effects are the status of the various windows

Example

Computer purchase system

A web based company sells computers (CPU), printers (PR), monitors (M), and additional memories (RAM).

Conditions: For each order the buyer may select from 3 CPU, 2 PR, 3M. RAM one unit in one order. M20 and M23 any CPU or as stand alone. M30 only with CPU 3. PR 1 is available free with CPU 2 or 3. M and PR can pbe purchased as stand alone. Non M30. CPU 1 gets RAM 256 upgrade. CPU 2 o 3 gets RAM 512 upgrade. RAM 1G upgrade and free PR2 available if CPU 3 purchased with M30.

There is a window to make selection with menus in particular a widget displaying the free item available and a “Price” widget reports the calculation related to prices.

Causes are the purchase of the items

Effects are the status of the various windows

Decision Tables from a CEG

CEG models relations among different aspects of the system. The derivation of test requires the definition of the corresponding decision table

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

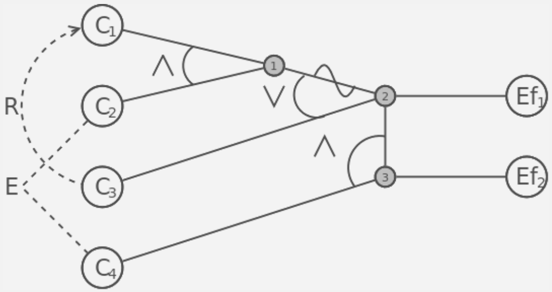
Input: A CEG containing causes $C_1, C_2, ..., C_p$ and effects $Ef_1, Ef_2, ..., Ef_q$
Output: A decision table containing $p + q$ rows and M columns where M depends on relationship between causes and effects.

Procedure CFG2DT
Step1: Initialize DT to an empty DT
Step2: Execute the following steps for i=1 to q

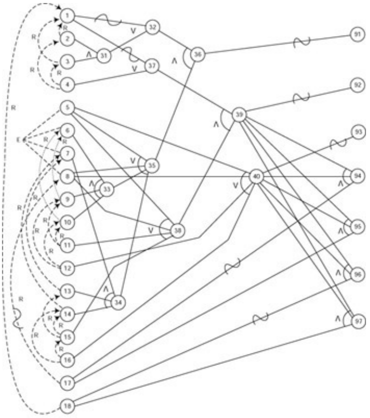
- **2.1** Select the next effect e
- **2.2** Find combinations of conditions that cause e to be present and store the m generated vector
- **2.3** update the decision table adding the generated vectors

DT derivation

Consider the following CEG and derive the corresponding decision table:



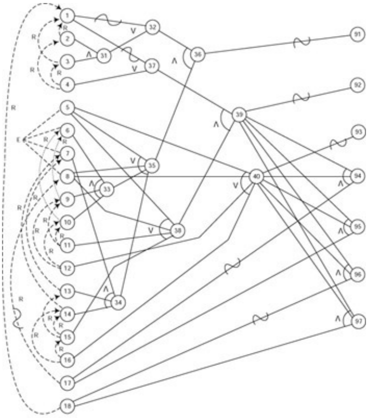
Apply the procedure to the following CEG:



You need to automatize the process. In your opinion which should be the main component included in a supporting tool?

Example

Apply the procedure to the following CEG:



You need to automatize the process. In your opinion which should be the main component included in a supporting tool?

Heuristic to avoid combinatorial explosion

The described approach could lead to exponential generation on the number of tests with respect to causes. Indeed having an effect depending on n causes can lead to the generation of a number of vectors in the order of 2^n

Reduction strategies

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

Test generation

Tests from a decision table

Each column of the decision table constitutes the source for generating tests. Consider 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

Techniques aiming at finding bugs in the coding of conditions

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 $\mathscr{T}$ 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

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:

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

Covered faults

To discover the covered faults lets modify the proposition introducing one or more operational fault

Generating BOR, BRO, BRE adequate tests

A predicate constraint C for predicate p_r is a sequence of $n+1$ boolean and relational symbols.
A test case t satisfies C for predicate p_r, if each component of p_r satisfies the corresponding constraint in C when evaluted against t.
e.g.: given $p_r = b \wedge r < s \vee u \geq v$ and $C : (t, =, >)$ the following test case satisfies C: $<b = true, r = 1, s = 1, u = 1, v = 0>$

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
- $AST(p_r)$

Onto Operator

Onto Operator

Given two sets A and B the onto operator constructs the minimal set of pairs $\langle a, b \rangle$ where $a \in A$ and $b \in B$ and each element of the two sets is used in at least one of the pairs in the onto set $A \otimes B$.
Which is the cardinality of the onto set?

Let $A = \{t, 0, >\}$ and $B = \{f, <\}$ lets derive the cartesian product and some example of onto product set

Abstract Syntax Tree for a Predicate p

AST

The abstract sintax tree provides a tree based representation of a predicate that is tipically useful for associating meaning to the predicate itself.
Leaf of the tre are atomic proposition while nodes are boolean operators

AST

Let's build the AST for the proposition:
$a < b \vee q \wedge \neg p \vee (a == c \wedge p)$

Generating the BOR-constraint set

Let p_r be a predicate and $AST(P_r)$ its abstract syntax tree, S_N the constraint set attached to a node N (where ${S^t}_N$ and ${S^f}_N$ are the true and false constraints associated with the node). The following alg. generates the BOR-constraint set for p_r

Input: $AST(p_r)$ (only singular expressions)

Output: BOR-Constraint set attached to the root node

1. Label each leaf node N of $AST(p_r)$ with its constraint set $S_N = \{t, f\}$
2. Visit the AST bottom-up. Let N_1 and N_2 direct descendants of node N and S_{N1} and S_{N2} the corresponding BOR-constraint set. S_N is computed as follows:

   2.1 N is an OR-node:
   - ${S^f}_N = {S^f}_{N1} \otimes {S^f}_{N2}$
   - ${S^t}_N = ({S^t}_{N1} \times \{f_2\}) \cup (\{f_1\} \times {S^t}_{N2})$ where $f_1 \in {S^f}_{N1}$ and $f_2 \in {S^f}_{N2}$

   2.2 N is an AND-node:
   - ${S^t}_N = {S^t}_{N1} \otimes {S^t}_{N2}$
   - ${S^f}_N = ({S^f}_{N1} \times \{t_2\}) \cup (\{t_1\} \times {S^f}_{N2})$ where $t_1 \in {S^t}_{N1}$ and $t_2 \in {S^t}_{N2}$

   2.3 N is NOT-node:
   - ${S^t}_N = {S^f}_{N1}$
   - ${S^f}_N = {S^t}_{N1}$

BOR-constraint set example

Let's apply the BOR-constraint procedure to:

- $(a + b < c) \wedge \neg p \vee (r > s)$

Generating the BRO-constraint set

Input: $AST(p_r)$ (only singular expressions)
Output: BRO-Constraint set attached to the root node

1. Label each leaf node N of $AST(p_r)$ with its constraint set S_N. For each leaf node that represents a Boolean variable $S_N = \{t, f\}$. For each leaf node that is a relational expression $S_N = \{(>), (=), (<)\}$.
2. Visit the AST bottom-up. Let N_1 and N_2 direct descendants of node N and S_{N1} and S_{N2} the corresponding BRO-constraint set. S_N is computed as done for the BOR procedure.

Let's apply the BRO-constraint procedure to:

- $(a + b < c) \wedge \neg p \vee (r > s)$

Generating the BRO-constraint set

Input: $AST(p_r)$ (only singular expressions)
Output: BRO-Constraint set attached to the root node

1. Label each leaf node N of $AST(p_r)$ with its constraint set S_N. For each leaf node that represents a Boolean variable $S_N = \{t, f\}$. For each leaf node that is a relational expression $S_N = \{(>), (=), (<)\}$.
2. Visit the AST bottom-up. Let N_1 and N_2 direct descendants of node N and S_{N1} and S_{N2} the corresponding BRO-constraint set. S_N is computed as done for the BOR procedure.

Let's apply the BRO-constraint procedure to:

- $(a + b < c) \wedge \neg p \vee (r > s)$

Generating the BRE-constraint set

Input: $AST(p_r)$ (only singular expressions)
Output: BRE-Constraint set attached to the root node

1. Label each leaf node N of $AST(p_r)$ with its constraint set S_N. For each leaf node that represents a Boolean variable $S_N = \{t, f\}$. For each leaf node that is a relational expression $S_N = \{(+\epsilon), (=), (-\epsilon)\}$.
2. Visit the AST bottom-up. Let N_1 and N_2 direct descendants of node N and S_{N1} and S_{N2} the corresponding BRE-constraint set. S_N is computed as done for the BOR procedure.

Constraint	**Satisfying condition**
$+\epsilon$	$0 < e_1 - e_2 \leq +\epsilon$
$-\epsilon$	$-\epsilon \leq e_1 - e_2 < 0$

Let's apply the BRE-constraint procedure to:

- $(a + b < c) \wedge \neg p \vee (r > s)$

Generating the BRE-constraint set

Input: $AST(p_r)$ (only singular expressions)
Output: BRE-Constraint set attached to the root node

1. Label each leaf node N of $AST(p_r)$ with its constraint set S_N. For each leaf node that represents a Boolean variable $S_N = \{t, f\}$. For each leaf node that is a relational expression $S_N = \{(+\epsilon), (=), (-\epsilon)\}$.
2. Visit the AST bottom-up. Let N_1 and N_2 direct descendants of node N and S_{N1} and S_{N2} the corresponding BRE-constraint set. S_N is computed as done for the BOR procedure.

Constraint	**Satisfying condition**
$+\epsilon$	$0 < e_1 - e_2 \leq +\epsilon$
$-\epsilon$	$-\epsilon \leq e_1 - e_2 < 0$

Let's apply the BRE-constraint procedure to:

- $(a + b < c) \wedge \neg p \vee (r > s)$

CEG and Predicate testing

CEG

- CEG strategy to define relations among causes and effects ("oracles")
- Decision table technique to identify test cases

Predicate testing

- Stategies for deriving test from predicates, fault coverage guarantees

"Better together"

CEG and Predicate testing

CEG

- CEG strategy to define relations among causes and effects (“oracles”)
- Decision table technique to identify test cases

Predicate testing

- Stategies for deriving test from predicates, fault coverage guarantees

“Better together”

CEG and Predicate testing

CEG

- CEG strategy to define relations among causes and effects ("oracles")
- Decision table technique to identify test cases

Predicate testing

- Stategies for deriving test from predicates, fault coverage guarantees

"Better together"

Usage of predicate testing techniques

Approaches to test set derivation from predicates can be applied considering different starting points:

- Specification based testing
- Program based testing

The different settings have different consequences