Combinatorial Testing

- Introduction
- Combinatorial Coverage Criteria
- Pairwise Test Generation
- Summary

Motivation

- The behavior of a software application may be affected by many factors, e.g., input parameters, environment configurations, and state variables.
- Techniques like equivalence partitioning and boundary-value analysis can be used to identify the possible values of individual factors.
- It is impractical to test all possible combinations of values of all those factors. (Why?)
**Combinatorial Explosion**

- Assume that an application has 10 parameters, each of which can take 5 values. How many possible combinations?

---

**Example - sort**

```bash
> man sort
Reformatting page. Wait... done

User Commands

```

NAME

`sort` - sort, merge, or sequence check text files

SYNOPSIS

```
/usr/bin/sort [ -cmu ] [ -o output ] [ -T directory ]
    [ -y [ kmem ]] [ -z recsz ] [ -dfiMnr ] [ -b ] [ -t char ]
    [ -k keydef ] [ +pos1 [-pos2 ] ] [ file...]

...
**Combinatorial Design**

- Instead of testing all possible combinations, a subset of combinations is generated to satisfy some well-defined combination strategies.

- A key observation is that not every factor contributes to every fault, and it is often the case that a fault is caused by interactions among a few factors.

- Combinatorial design can dramatically reduce the number of combinations to be covered but remains very effective in terms of fault detection.

**Fault Model**

- A \( t \)-way interaction fault is a fault that is triggered by a certain combination of \( t \) input values.

- A simple fault is a \( t \)-way fault where \( t = 1 \); a pairwise fault is a \( t \)-way fault where \( t = 2 \).

- In practice, a majority of software faults consist of simple and pairwise faults.
**Example – Pairwise Fault**

```c
begin
  int x, y, z;
  input (x, y, z);
  if (x == x1 and y == y2)
    output (f(x, y, z));
  else if (x == x2 and y == y1)
    output (g(x, y));
  else
    output (f(x, y, z) + g(x, y))
end

Expected: x = x1 and y = y1 => f(x, y, z) - g(x, y); x = x2, y = y2 => f(x, y, z) + g(x, y)
```

**Example – 3-way Fault**

```c
// assume x, y ∈ {-1, 1}, and z ∈ {0, 1}
begin
  int x, y, z, p;
  input (x, y, z);
  p = (x + y) * z // should be p = (x - y) * z
  if (p >= 0)
    output (f(x, y, z));
  else
    output (g(x, y));
end
```
Combinatorial Testing

- Introduction
- Combination Strategies Criteria
- Pairwise Test Generation
- Summary

All Combinations Coverage

- Every possible combination of values of the parameters must be covered
- For example, if we have three parameters $P_1 = (A, B)$, $P_2 = (1, 2, 3)$, and $P_3 = (x, y)$, then all combinations coverage requires 12 tests: $\{(A, 1, x), (A, 1, y), (A, 2, x), (A, 2, y), (A, 3, x), (A, 3, y), (B, 1, x), (B, 1, y), (B, 2, x), (B, 2, y), (B, 3, x), (B, 3, y)\}$
Each Choice Coverage

- Each parameter value must be covered in at least one test case.
- Consider the previous example, a test set that satisfies each choice coverage is the following: \{(A, 1, x), (B, 2, y), (A, 3, x)\}

Pairwise Coverage

- Given any two parameters, every combination of values of these two parameters are covered in at least one test case.
- A pairwise test set of the previous example is the following:

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>x</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
<td>x</td>
</tr>
<tr>
<td>A</td>
<td>-</td>
<td>y</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>y</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>y</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>y</td>
</tr>
<tr>
<td>B</td>
<td>-</td>
<td>x</td>
</tr>
</tbody>
</table>
**T-Wise Coverage**

- Given any $t$ parameters, every combination of values of these $t$ parameters must be covered in at least one test case.
- For example, a 3-wise coverage requires every triple be covered in at least one test case.
- Note that all combinations, each choice, and pairwise coverage can be considered to be a special case of $t$-wise coverage.

**Base Choice Coverage**

- For each parameter, one of the possible values is designated as a base choice of the parameter.
- A base test is formed by using the base choice for each parameter.
- Subsequent tests are chosen by holding all base choices constant, except for one, which is replaced using a non-base choice of the corresponding parameter:

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>x</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
<td>x</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>y</td>
</tr>
</tbody>
</table>
**Multiple Base Choices Coverage**

- At least one, and possibly more, base choices are designated for each parameter.

- The notions of a base test and subsequent tests are defined in the same as Base Choice.

---

**Subsumption Relation**

```
   All Combinations
     /\       T>=2
  T-Wise   Multiple Base Choices
           /\       /\  
  Pairwise Base Choice
           /\       /\  
  Each Choice
```
Combinatorial Testing

- Introduction
- Combination Strategies Criteria
- Pairwise Test Generation
- Summary

Why Pairwise?

- Many faults are caused by the interactions between two parameters
  - 92% statement coverage, 85% branch coverage
- Not practical to cover all the parameter interactions
  - Consider a system with $n$ parameter, each with $m$ values. How many interactions to be covered?
- A trade-off must be made between test effort and fault detection
  - For a system with 20 parameters each with 15 values, pairwise testing only requires less than 412 tests, whereas exhaustive testing requires $15^{20}$ tests.
**Example (1)**

Consider a system with the following parameters and values:

- parameter A has values A1 and A2
- parameter B has values B1 and B2, and
- parameter C has values C1, C2, and C3

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>C1</td>
</tr>
<tr>
<td>A1</td>
<td>B2</td>
<td>C2</td>
</tr>
<tr>
<td>A2</td>
<td>B1</td>
<td>C3</td>
</tr>
<tr>
<td>A2</td>
<td>B2</td>
<td>C1</td>
</tr>
<tr>
<td>A2</td>
<td>B1</td>
<td>C2</td>
</tr>
<tr>
<td>A1</td>
<td>B2</td>
<td>C3</td>
</tr>
</tbody>
</table>

**Example (2)**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>C1</td>
</tr>
<tr>
<td>A1</td>
<td>B2</td>
<td>C1</td>
</tr>
<tr>
<td>A2</td>
<td>B1</td>
<td>C2</td>
</tr>
<tr>
<td>A2</td>
<td>B2</td>
<td>C2</td>
</tr>
<tr>
<td>A2</td>
<td>B1</td>
<td>C1</td>
</tr>
<tr>
<td>A1</td>
<td>B1</td>
<td>C3</td>
</tr>
<tr>
<td>A2</td>
<td>B2</td>
<td>C3</td>
</tr>
</tbody>
</table>
The IPO Strategy

- First generate a pairwise test set for the first two parameters, then for the first three parameters, and so on
- A pairwise test set for the first \( n \) parameters is built by extending the test set for the first \( n - 1 \) parameters
  - **Horizontal growth**: Extend each existing test case by adding one value of the new parameter
  - **Vertical growth**: Adds new tests, if necessary

---

Algorithm IPO\(_H\) (\( T, p_1 \))

Assume that the domain of \( p_i \) contains values \( v_1, v_2, ..., v_q \);
\[ \pi = \{ \text{pairs between values of } p_i \text{ and values of } p_1, p_2, ..., p_{i-1} \} \]

if \( |T| \leq q \)

for \( 1 \leq j \leq |T| \), extend the \( j^{\text{th}} \) test in \( T \) by adding value \( v_j \) and remove from \( \pi \) pairs covered by the extended test

else

for \( 1 \leq j \leq q \), extend the \( j^{\text{th}} \) test in \( T \) by adding value \( v_j \) and remove from \( \pi \) pairs covered by the extended test;

for \( q < j \leq |T| \), extend the \( j^{\text{th}} \) test in \( T \) by adding one value of \( p_i \) such that the resulting test covers the most number of pairs in \( \pi \), and remove from \( \pi \) pairs covered by the extended test
**Algorithm IPO-V(T, π)**

let $T'$ be an empty set;
for each pair in $\pi$
    assume that the pair contains value $w$ of $p_k$, $1 \leq k < i$, and value $u$ of $p_i$;
    if ($T'$ contains a test with "-" as the value of $p_k$ and $u$ as the value of $p_i$)
        modify this test by replacing the "-" with $w$
    else
        add a new test to $T'$ that has $w$ as the value of $p_k$, $u$ as the value of $p_i$, and "-" as the value of every other parameter;
$T = T \cup T'$

**Example Revisited**

Show how to apply the IPO strategy to construct the pairwise test set for the example system.
**Combinatorial Testing**

- Introduction
- Combinatorial Coverage Criteria
- Pairwise Test Generation
- Summary

**Summary**

- Combinatorial testing makes an excellent *trade-off* between test effort and test effectiveness.
- Pairwise testing can often reduce the number of dramatically, but it can still detect faults effectively.
- The IPO strategy constructs a pairwise test set incrementally, one parameter at a time.
- In practice, some combinations may be invalid from the domain semantics, and must be excluded, e.g., by means of constraint processing.