UT ECE | THE UNIVERSITY OF TEXAS AT AUSTIN
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

EE360T/382V Software Testing

khurshid@ece.utexas.edu

February 28, 2018

Overview

Last class – Exam 1 handed back

Today – Complete Input space partitioning

Next class – Start Syntax-based testing

Problem Set 3 is out

- **Due**: Mar 9, 2018, 11:59pm

Recall: Criteria based on structures

The textbook focuses on four kinds of structures to define criteria:

- Graphs (Chapter 2)
  - E.g., control-flow graphs (CFGs)
- Logical expressions (Chapter 3)
  - E.g., if-conditions
- Input domain characterization (Chapter 4)
  - E.g., sorted array
- Syntactic structures (Chapter 5)
  - E.g., mutation

EE360T/382V Software Testing

khurshid@ece.utexas.edu

Input Space Partitioning (Chapter 4)*

*Introduction to Software Testing by Ammann and Offutt

Combination strategies criteria

How do we consider multiple partitions together?

What combination of blocks do we choose values from?

All Combinations Coverage (ACoC) – All combinations of blocks from all characteristics must be used

- Number of tests is product of number of blocks in each characteristic: $\prod_i |bi|$
  - E.g., the second interface-based IDM for the triangle classifier results in 4 x 4 x 4 = 64 tests

Each Choice Coverage (ECC)

One value from each block for each characteristic must be used in at least one test case

Number of tests is at least the number of blocks in the largest characteristic: $\text{Max}_i \, |b_i|$

Example tests for the triangle classifier:

(2, 2, 2)
(1, 1, 1)
(0, 0, 0)
(-1, -1, -1)

Pair-Wise Coverage (PWC)

A value from each block for each characteristic must be combined with a value from every block for each other characteristic

Number of tests is at least the product of two largest characteristics: $\text{Max}_i \ |b_i| \ \text{x} \ \text{Max}_{j \ != i} \ |b_j|$

Example tests for triangle classifier:

2, 2, 2	2, 1, 1	2, 0, 0	2, -1, -1
1, 2, 1	1, 1, 0	1, 0, -1	1, -1, 2
0, 2, 0	0, 1, -1	0, 0, 2	0, -1, 1
-1, 2, -1	-1, 1, 2	-1, 0, 1	-1, -1, 0

T-Wise Coverage (TWC)

A value from each block for each group of *t* characteristics must be combined

If all characteristics are the same size, number of tests is at least $(\text{Max}_i\ |b_i|)^t$

If *t* is the number of characteristics, TWC is equivalent to all combinations coverage (ACoC)

Benefits of *t*-wise (over pair-wise) are not clear

Base Choice Coverage (BCC)

A base choice block is chosen for each characteristic, and a base test is formed by using the base choice for each characteristic. Subsequent tests are chosen by holding all but one base choice constant and using each non-base choice in each other characteristic.

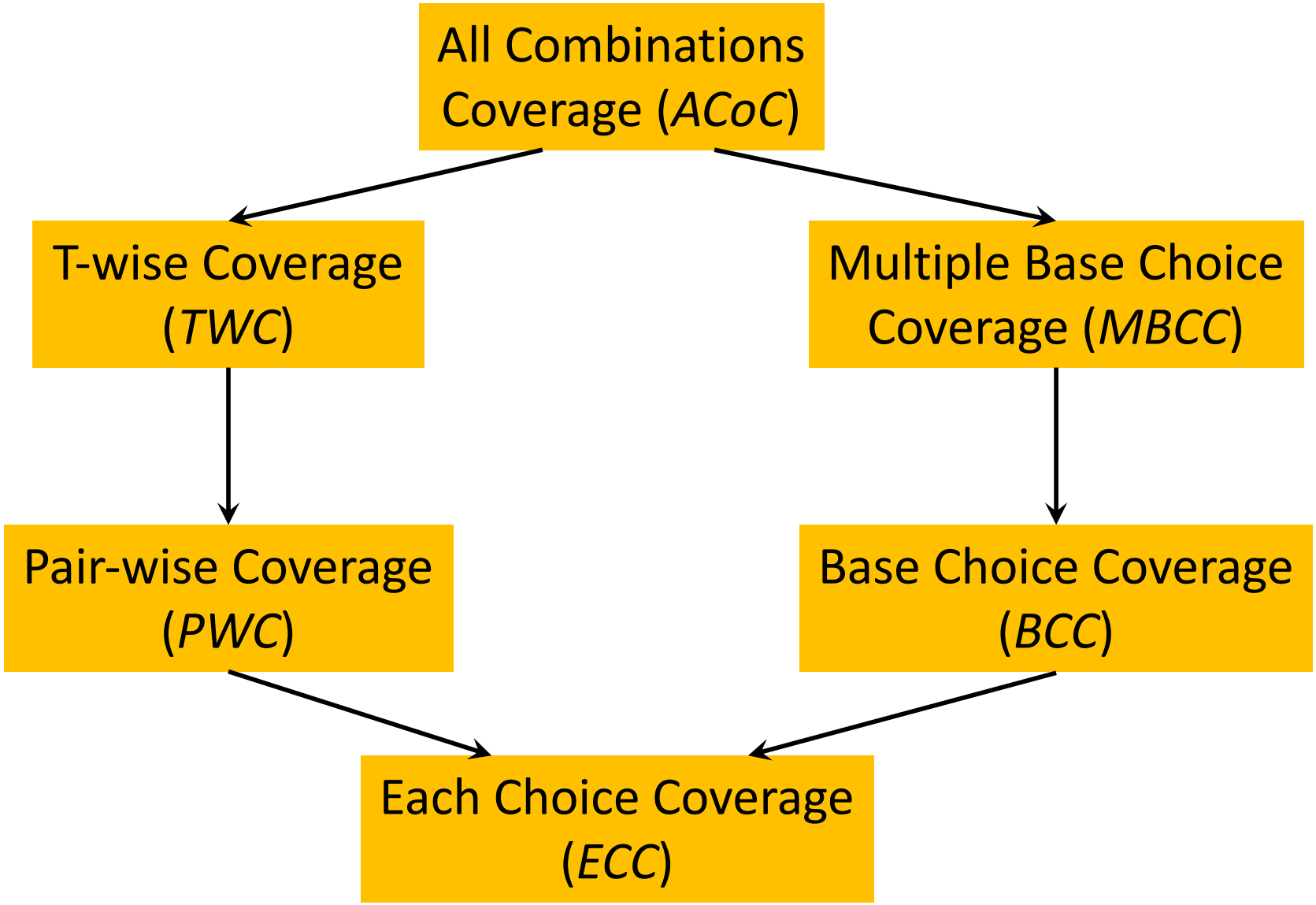
Number of tests is one base test + one test for each other block: $1 + \sum_i(|bi| - 1)$

Example using triangle classifier: >1 as base choice block

2, 2, 2	2, 2, 1	2, 1, 2	1, 2, 2
	2, 2, 0	2, 0, 2	0, 2, 2
	2, 2, -1	2, -1, 2	-1, 2, 2

Can be generalized to multiple base choice blocks (*MBCC*)

Subsumption



Constraints among partitions

Some block combinations may be infeasible

- E.g., valid = false and scalene = true not possible

Constraints capture feasibility properties

Two basic types of constraints

- A block from one characteristic **cannot** be combined with a specific other block
- A block from one characteristic **can only** be combined with a specific other block

Handling constraints depends on specific criteria

- *ACoC*, *PWC*, *TWC*: drop infeasible combinations
- *BCC*, *MBCC*: change a value to another non-base choice to try to find a feasible combination

Example constraint handling

Sorting an array

Input: array with elements of some arbitrary type

Output: array in sorted order, largest, smallest value

Characteristics:

- Length of
- Type of e
- Max valu
- Min value
- Position c
- Position c

Partitions:

- Len { 0, 1, 2..100, 101..MAXINT }
- Type { int, char, string, other }
- Max { ≤0, 1, >1, 'a', 'Z', 'b', ..., 'Y' }
- Min { ... }
- Max Pos { 1, 2 .. Len-1, Len }
- Min Pos { 1, 2 .. Len-1, Len }

Blocks from other characteristics are irrelevant

Blocks must be combined

Applying input domain modeling to an example recursive structure

Recall our singly linked-list example:

```
public class SLList { // invariant: acyclicity
    Node header;
    int size;

    static class Node {
        int elem;
        Node next;
    }

    int removeFirst() {
        // precondition: header != null
        // postcondition: returns the element in the first
        //   node and removes that node if header != null;
        //   else, throws NullPointerException
        ... }
}
```

One way to form characteristics and apply coverage criteria

Observe – *removeFirst*'s behavior depends on **one input**: receiver object, which is a list

- Note: empty list violates method precondition

Consider properties of lists (based on type declaration or functionality) to form characteristics

1. List has repetitions
   - 2 blocks: T; F
2. Length of list
   - 3 blocks: length == 1; length == 2; length > 2

Can apply combination strategies criteria to blocks for these two characteristics

Another way to form characteristics and apply coverage criteria

Observe – *removeFirst*'s behavior depends on **values of object fields reachable from *this***

- Each reachable object field is effectively an input

Consider properties of fields of *this* to define characteristics

1. Value of *this.header* (assume non-null)
   - 2 blocks
     - *header.next* == *header*; *header.next* != *header*
2. Value of *this.size* (assume non-zero)
   - 3 blocks: 1; 2; >2

Again, can apply combination strategies criteria

?/!