Subsumption revisited

• **Criteria Subsumption**: Test criterion $C_1$ subsumes $C_2$ iff every set of test cases that satisfies $C_1$ also satisfies $C_2$

• Question from last class:
Subsumption ≠ Subset

• Subsumption cannot always be explained using subsets, e.g.,
  – C1: {Lemon, Pistachio, Cantaloupe, Pear, Tangerine, Apricot}
  – C2: {Yellow, Green, Orange, While}
• From last class: C1 subsumes C2. But TR(C1) ⊈ TR (C2)
• There is a many-to-one mapping from TR(C1) to TR(C2)
More on Subsumption

- Can we always show subsumption as subsets or many-to-one mappings?

- $C_1 = \{\text{program branches}\} = \{B, !B\}$
- $C_2 = \{\text{program statements}\} = \{\text{line 1, line 2, line 3, line 4}\}$

```java
int stringFactor(String i, int n) {
    1. if (i != null || n != 0) // --> B
       return i.length() / n;
    2. else
    3.     return -1;
    4. return -1;
}
```
Subsumption: wrong definition?

- **Criteria Subsumption**: Test criterion $C_1$ subsumes $C_2$ iff every set of test cases that satisfies $C_1$ also satisfies $C_2$.

- Comment from last class: definition should be, “$C_1$ subsumes $C_2$ iff every set of test cases that satisfies $C_2$ also satisfies $C_1$”.

- Which definition do you now think is correct?

- Hint: replace $C_1$ with “Branch Coverage” and $C_2$ with “Statement Coverage”.
Summary on subsumption

• Formally, subsumption is a relation between two sets of test requirements

• Goal: given a test set T that satisfies criterion C1, what can we say about T with respect to another criterion C2?

• There are many ways to show a subsumption relation
  – Subset
  – Many-to-one mapping
  – One-to-one mapping
  – …
Hands-on Demo

- Maven
Input Space Partitioning

Owolabi Legunsen

The following are modified versions of the publicly-available slides for Chapter 6 in the Ammann and Offutt Book, “Introduction to Software Testing” (http://www.cs.gmu.edu/~offutt/softwaretest)
1st of four structures we’ll cover

Four Structures for Modeling Software

- **Input Space**
  - Source
  - Use cases
  - Design
  - Specs

- **Graphs**
  - Source
  - FSMs
  - Specs

- **Logic**
  - Source
  - FSMs
  - DNF

- **Syntax**
  - Source
  - Models
  - Integ
  - Input
Why Input Space Partitioning?

• No implementation knowledge is needed
  – Just the input space

• Easy to apply without automation

• Can adjust the procedure to get more or fewer tests

• Equally applicable at several levels of testing
  – Unit, Integration, System, etc.
An industrial study of applying input space partitioning to test financial calculation engines

Jeff Offutt · Chandra Alluri

Published online: 23 September 2012
© Springer Science+Business Media, LLC 2012
Editor: James Miller
Input Domains and ISP

- **Input domain**: all possible inputs to a program
  - Most input domains are so large that they are effectively *infinite*
- **Input parameters** define the scope of the input domain
  - Parameter values to a method, data from a file, global variables, user inputs
- **ISP**: First *partition* input domain into *regions* (called *blocks*)
  - Values in each block are assumed equally useful for testing
- **ISP**: Then choose at least *one value* from each block

**Input domain**: Alphabetic letters

**Partitioning characteristic**: Case of letter

- Block 1: upper case
- Block 2: lower case