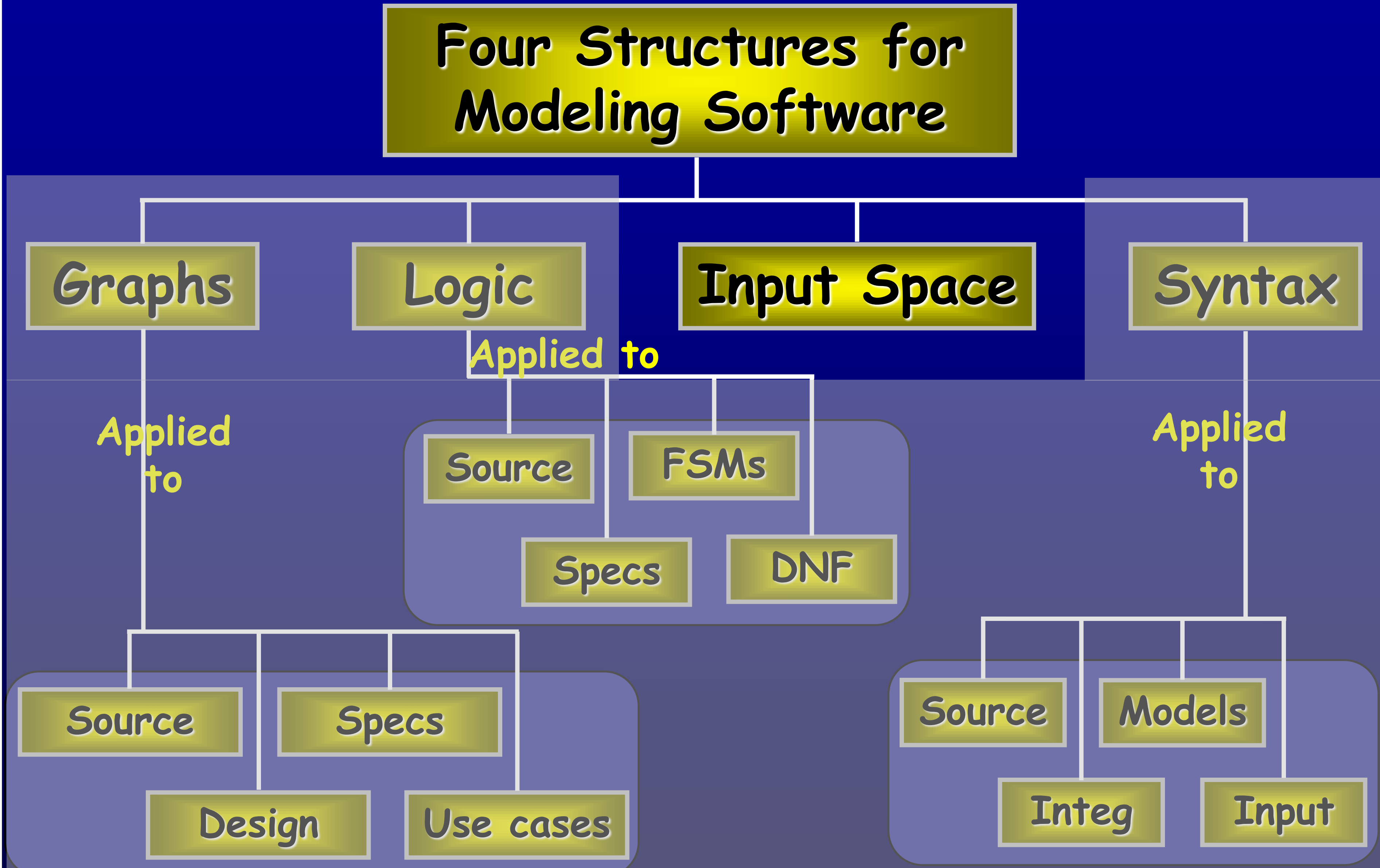
Ch. 4 : Input Space Coverage



Input Domains

- **The <u>input domain</u> to a program contains all the possible inputs to that program**

- **Testing is fundamentally about <u>choosing finite sets</u> of values from the input domain**

The Source of Input Domain
at different abstraction levels

- **System level**
  - **Number of students** { *0, 1, >1* }
  - **Level of course** { *600, 700, 800* }
  - **Major** { *swe, cs, isa, infs* }

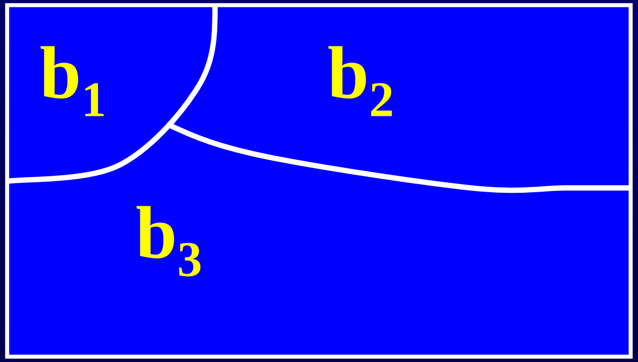
- **Unit level**
  - **Parameters** *F (int X, int Y)*
  - **Possible values** X*: { <0, 0, 1, 2, >2 }, Y : { 10, 20, 30 }*
  - **Tests** *F (-5, 10), F (0, 20), F (1, 30), F (2, 10), F (5, 20)*

Input Domain Partitioning

- **Domain for each input parameter is partitioned into regions**
- **At least one value is chosen from each region**

Partitioning Domains

- **Domain *D***
- **Partition scheme *q* of *D***
- **The partition *q* defines a set of blocks, $Bq = b_1 , b_2 , \ldots b_Q$**
- **The partition must satisfy two properties :**
  1. **blocks must be *pairwise disjoint* (no overlap)**
  2. **together the blocks *cover* the domain *D* (complete)**



Choosing Partitions

- Choosing (or defining) partitions seems easy, but is easy to get wrong
- Consider the "order of file F"

b_1 = sorted in ascending order

b_2 = sorted in descending order

b_3 = arbitrary order

but … something's fishy …

What if the file is of length 1?

The file will be in all three blocks …

That is, disjointness is not satisfied

Solution:

Each characteristic should address just one property

C1: File F sorted ascending

- c1.b1 = true
- c1.b2 = false

C2: File F sorted descending

- c2.b1 = true
- c2.b2 = false

Properties of Partitions

- If the partitions are not **complete** or **disjoint**, that means the partitions have not been considered carefully enough
- They should be reviewed carefully, like any **design** attempt
- Different **alternatives** should be considered

Exercise

- Consider the following characteristics and blocks for an input parameter, **car**:
  - **Where Made**
    - **b1: North America**
    - **b2: Europe**
    - **b3: Asia**
  - **Energy Source**
    - **b1: Gas Only**
    - **b2: Electric Only**
    - **b3: Hybrid**
  - **Size**
    - **b1: 2-door**
    - **b2: 4-door**
    - **b3: hatch-back**

What are the mistakes here?

How can they be corrected?

Exercise

– **Where Made**
  - **b1: North America**
  - **b2: Europe**
  - **b3: Asia**
– **Energy Source**
  - **b1: Gas Only**
  - **b2: Electric Only**
  - **b3: Hybrid**
– **Size**
  - **b1: 2-door**
  - **b2: 4-door**
  - **b3: hatch-back**

Not complete!
May be an additional block, "other"?

Not disjoint!
A hatchback can be 2-door or 4-door
Either add more blocks such as
2-door+hatch-back and 4-door+hatch-back
or
Add more characteristics:
Side Doors; b1: 2, b2: 4
Hatch-back; b1: True, b2: False

Input Domain Modeling

- Step 1: Identify the input domain
- Step 2: Identify equivalence classes (partitioning)

- Two basic approaches
  - Interface Based
  - Functionality Based

Interface Based Input Domain Modeling

- Each parameter is considered in isolation
- Each characteristic is related to a single parameter

- Simple, can be automated 

- Ignores parameter interactions/relations
- Can lead to incomplete domain modeling 

Functionality Based Input Domain Modeling

- Use semantics and domain knowledge
- Characteristics correspond to intended functionality

- Modeling is based on requirements; can start early
- Takes parameter relationships into account

- Hard to identify characteristics
- Hard to translate values to concrete test cases

Example

- public boolean findElement(List list, Object element)
  - If list or element is null, throw NullPointerException
  - Else if element is in the list, return true
  - Else, return false

- Interface-based characteristics for list:
  - list is null
    - b1 = True
    - b2 = False
  - list is empty
    - b1 = True
    - b2 = False

Functionality-based characteristics:

- Number of occurences of element in list
  - b1 = 0
  - b2 = 1
  - b3 = more than 1
- Element occurs first in list
  - b1 = True
  - b2 = False

Choosing Blocks and Values

- **A creative design step not to unnecessarily increase the number of test cases and capture all faults at the same time**

- **General strategy for indentifying values**
  - **Partition valid values for different part of the functionality**
  - **Check for completeness (missing partitions)**
  - **Check for disjointness (overlapping partitions)**

- **Selection**
  - **Valid values**
  - **Boundaries**
  - **Normal use**
  - **Invalid values**

Running Example

- **TriTyp program**
  - **Input: Side1, Side2, Side3**
    - **3 integer values that represent the lengths of the sides of a triangle**
  - **Output: the category of triangle**
    - **Scalene**
    - **Isosceles**
    - **Equilateral**
    - **Invalid**

TriTyp: Interface-Based Modeling

- **Relation of a variable with respect to some special value, 0**

First Characterization of TriTyp's Inputs

Characteristic	b_1	b_2	b_3
q_1 = "Relation of Side 1 to 0"	greater than 0	equal to 0	less than 0
q_2 = "Relation of Side 2 to 0"	greater than 0	equal to 0	less than 0
q_3 = "Relation of Side 3 to 0"	greater than 0	equal to 0	less than 0

- A maximum of 3*3*3 = 27 tests
- Some triangles are valid, some are invalid
- Refining the characterization can lead to more tests …

TriTyp: Interface-Based Modeling

Second Characterization of TriTyp Inputs

Characteristic	b_1	b_2	b_3	b_4
q_1 = "Refinement of q_1"	greater than 1	equal to 1	equal to 0	less than 0
q_2 = "Refinement of q_2"	greater than 1	equal to 1	equal to 0	less than 0
q_3 = "Refinement of q_3"	greater than 1	equal to 1	equal to 0	less than 0

- **A maximum of 4*4*4 = 64 tests**
- **Complete because the inputs are integers (0 . . 1)**

Possible values for partition q_1

Characteristic	b_1	b_2	b_3	b_4
Side1	2	1	0	-1

Test boundary conditions

TriTyp: Funtionality-Based Modeling

- **First two characterizations are based on syntax–parameters and their type**
- **A semantic level characterization could use the fact that the three integers represent a triangle**

Geometric Characterization of *TriTyp* Inputs				
Characteristic	b_1	b_2	b_3	b_4
q_1 = "Geometric Classification"	scalene	isosceles	equilateral	invalid

- Oops … something's fishy … equilateral is also isosceles !
- We need to refine the example to make characteristics valid

Correct Geometric Characterization of *TriTyp* Inputs				
Characteristic	b_1	b_2	b_3	b_4
q_1 = "Geometric Classification"	scalene	isosceles, not equilateral	equilateral	invalid

TriTyp: Funtionality-Based Modeling

- **Values** for this partitioning can be chosen as

Possible values for geometric partition q_1				
Characteristic	b_1	b_2	b_3	b_4
Triangle	(4, 5, 6)	(3, 3, 4)	(3, 3, 3)	(3, 4, 8)

TriTyp: Funtionality-Based Modeling

- A **different approach** would be to break the geometric characterization into four separate characteristics

Four Characteristics for *TriTyp*		
Characteristic	b_1	b_2
q_1 = "Scalene"	True	False
q_2 = "Isosceles"	True	False
q_3 = "Equilateral"	True	False
q_4 = "Valid"	True	False

- Use constraints to ensure that
  - Equilateral = True implies Isosceles = True
  - Valid = False implies Scalene = Isosceles = Equilateral = False

Choosing Combinations of Values

- Once characteristics and partitions are defined, the next step is to **choose test values**
- We use **criteria** – to choose **effective** subsets
- The most obvious criterion is to choose all combinations

<u>All Combinations (ACoC)</u> : All combinations of blocks from all characteristics must be used.

- Number of tests is the product of the number of blocks in each characteristic : $\prod_{i=1}^{Q}(B_i)$
- The second characterization of TriTyp results in 4*4*4 = 64 tests
  - Too many ?

Input Space Partitioning (ISP) Coverage Criteria – All Combinations

- Consider the "second characterization" of TriTyp as given before:

Characteristic	b_1	b_2	b_3	b_4
q_1 = "Refinement of q_1"	greater than 1	equal to 1	equal to 0	less than 0
q_2 = "Refinement of q_2"	greater than 1	equal to 1	equal to 0	less than 0
q_3 = "Refinement of q_3"	greater than 1	equal to 1	equal to 0	less than 0

- For convenience, we relabel the blocks:

Characteristic	b_1	b_2	b_3	b_4
A	A1	A2	A3	A4
B	B1	B2	B3	B4
C	C1	C2	C3	C4

ISP Criteria – ACoC Tests

A1 B1 C1	A2 B1 C1	A3 B1 C1	A4 B1 C1
A1 B1 C2	A2 B1 C2	A3 B1 C2	A4 B1 C2
A1 B1 C3	A2 B1 C3	A3 B1 C3	A4 B1 C3
A1 B1 C4	A2 B1 C4	A3 B1 C4	A4 B1 C4
A1 B2 C1	A2 B2 C1	A3 B2 C1	A4 B2 C1
A1 B2 C2	A2 B2 C2	A3 B2 C2	A4 B2 C2
A1 B2 C3	A2 B2 C3	A3 B2 C3	A4 B2 C3
A1 B2 C4	A2 B2 C4	A3 B2 C4	A4 B2 C4
A1 B3 C1	A2 B3 C1	A3 B3 C1	A4 B3 C1
A1 B3 C2	A2 B3 C2	A3 B3 C2	A4 B3 C2
A1 B3 C3	A2 B3 C3	A3 B3 C3	A4 B3 C3
A1 B3 C4	A2 B3 C4	A3 B3 C4	A4 B3 C4
A1 B4 C1	A2 B4 C1	A3 B4 C1	A4 B4 C1
A1 B4 C2	A2 B4 C2	A3 B4 C2	A4 B4 C2
A1 B4 C3	A2 B4 C3	A3 B4 C3	A4 B4 C3
A1 B4 C4	A2 B4 C4	A3 B4 C4	A4 B4 C4

ACoC yields 4*4*4 = **64 tests** for **TriTyp**!

This is almost certainly more than we need

Only **8 are valid** (all sides greater than zero)

ISP Criteria – Each Choice

- **64 tests for TriTyp is almost certainly way too many**
- **One criterion comes from the idea that we should try at least one value from each block**

> **<u>Each Choice Coverage (ECC)</u> : One value from each block for each characteristic must be used in at least one test case.**

- Number of tests is the number of blocks in the largest characteristic : $\mathbf{Max}_{i=1}^{Q}(B_i)$

For TriTyp :	Substituting values:
A1, B1, C1	2, 2, 2
A2, B2, C2	1, 1, 1
A3, B3, C3	0, 0, 0
A4, B4, C4	-1, -1, -1

ISP Criteria – Pair-Wise

- Each choice yields few tests—cheap but maybe ineffective
- Another approach combines values with other values

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 $(\mathrm{Max}_{i=1}^{Q}(B_i)) * (\mathrm{Max}_{j=1,\, j!=i}^{Q}(B_j))$

For TriTyp :	A1, B1, C1	A1, B2, C2	A1, B3, C3	A1, B4, C4
	A2, B1, C2	A2, B2, C3	A2, B3, C4	A2, B4, C1
	A3, B1, C3	A3, B2, C4	A3, B3, C1	A3, B4, C2
	A4, B1, C4	A4, B2, C1	A4, B3, C2	A4, B4, C3

ISP Criteria –T-Wise

- A natural extension is to require combinations of *t* values instead of *2*

t-Wise Coverage (TWC) :A value from each block for each group of t characteristics must be combined.

- Number of tests is at least the product of *t* largest characteristics
- If all characteristics are the same size, the formula is

$$\left(\mathrm{Max}_{i=1}^{Q}(B_i)\right)^t$$

- If *t* is the number of characteristics *Q*, then all combinations
- That is … *Q-WC = ACoC*
- *t*-wise is expensive and benefits are not clear

ISP Criteria – Base Choice

- Testers sometimes recognize that certain values are **important**
- This uses **domain knowledge** of the program

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 $1 + \sum_{i=1}^{Q} (B_i - 1)$

For TriTyp : Base	A1, B1, C1	A1, B1, C2	A1, B2, C1	A2, B1, C1
		A1, B1, C3	A1, B3, C1	A3, B1, C1
		A1, B1, C4	A1, B4, C1	A4, B1, C1

Base Choice Notes

- The base test must be **feasible**
  - That is, all base choices must be **compatible**
- **Base choices** can be
  - Most likely from an end-use point of view
  - Simplest
  - Smallest
  - First in some ordering
- **Happy path** tests often make good base choices
- The base choice is a **crucial design** decision
  - Test designers should **document** why the choices were made

ISP Criteria – Multiple Base Choice

- **We sometimes have more than one logical base choice**

<u>Multiple Base Choice Coverage (MBCC)</u> : At least one, and possibly more, base choice blocks are chosen for each characteristic, and base tests are formed by using each base choice for each characteristic at least once. Subsequent tests are chosen by holding all but one base choice constant for each base test and using each non-base choice in each other characteristic.

- **If *M* base tests and m_i base choices for each characteristic:**

$$M + \sum_{i=1}^{Q} (M * (B_i - m_i))$$

ISP Criteria – Multiple Base Choice

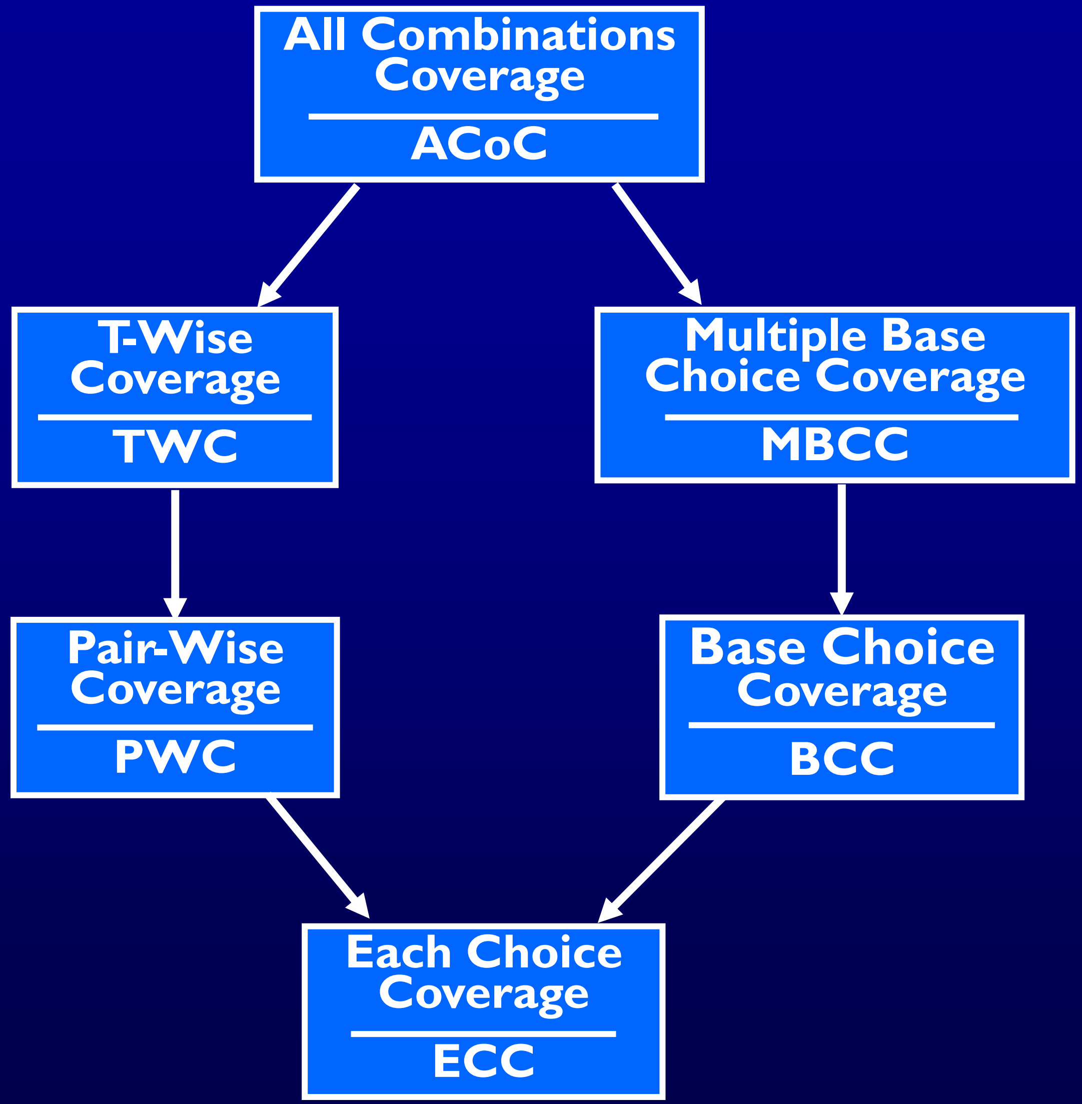
- **We sometimes have more than one logical base choice**

<u>Multiple Base Choice Coverage (MBCC)</u> : At least one, and possibly more, base choice blocks are chosen for each characteristic, and base tests are formed by using each base choice for each characteristic at least once. Subsequent tests are chosen by holding all but one base choice constant for each base test and using each non-base choice in each other characteristic.

For TriTyp : <u>Bases</u>

A1, B1, C1	**A1, B1, C3**	**A1, B3, C1**	**A3, B1, C1**
	A1, B1, C4	**A1, B4, C1**	**A4, B1, C1**
	A1,B1,C2	**A1, B2, C1**	**A2, B1, C1**
A2, B2, C2	**A2, B2, C3**	**A2, B3, C2**	**A3, B2, C2**
	A2, B2, C4	**A2, B4, C2**	**A4, B2, C2**
	A2, B2, C1	**A2, B1, C2**	**A1, B2, C2**

ISP Coverage Criteria Subsumption



Exercise 1.1

- **Consider the following characteristics and blocks**

Characteristics	Block 1	Block 2	Block 3	Block 4
Value 1	< 0	0	> 0	
Value 2	< 0	0	> 0	
Operation	$+$	$-$	$\times$	$\div$

- **Provide test cases that satisfy Each Choice coverage**

Exercise 1.1

- **Consider the following characteristics and blocks**

Characteristics	Block 1	Block 2	Block 3	Block 4
Value 1	< 0	0	> 0	
Value 2	< 0	0	> 0	
Operation	$+$	$-$	$\times$	$\div$

- **4 test cases can satisfy Each Choice coverage**

V1	*V2*	*Op*
-2	*-2*	$+$
0	*0*	$-$
2	*2*	$\times$
2	*2*	$\div$

Exercise 1.2

- Consider the following characteristics and blocks

Characteristics	Block 1	Block 2	Block 3	Block 4
Value 1	< 0	0	> 0	
Value 2	< 0	0	> 0	
Operation	$+$	$-$	$\times$	$\div$

- Provide test cases that satisfy **Base Choice** coverage
  - Assume base choices as
    - Value 1: > 0
    - Value 2: > 0
    - Operation: +

Exercise 1.2

- Consider the following characteristics and blocks

Characteristics	Block 1	Block 2	Block 3	Block 4
Value 1	< 0	0	> 0	
Value 2	< 0	0	> 0	
Operation	$+$	$-$	$\times$	$\div$

- 8 test cases can satisfy **Base Choice** coverage
  - Assuming base choices as
    - Value 1 => 0
    - Value 2 => 0
    - Operation = +

V1	*V2*	*Op*
2	2	+
-2	2	+
0	2	+
2	-2	+
2	0	+
2	2	-
2	2	$\times$
2	2	$\div$

Exercise 1.3

- **Consider the following characteristics and blocks**

Characteristics	Block 1	Block 2	Block 3	Block 4
Value 1	< 0	0	> 0	
Value 2	< 0	0	> 0	
Operation	$+$	$-$	$\times$	$\div$

- **How many test cases are needed to satisfy the All Combinations (ACoC) criterion?**

Exercise 1.3

- **Consider the following characteristics and blocks**

Characteristics	Block 1	Block 2	Block 3	Block 4
Value 1	< 0	0	> 0	
Value 2	< 0	0	> 0	
Operation	$+$	$-$	$\times$	$\div$

- **How many test cases are needed to satisfy the All Combinations (ACoC) criterion?**
  - **3 * 3 * 4 = 36**

Exercise 1.4

- **Consider the following characteristics and blocks**

Characteristics	Block 1	Block 2	Block 3	Block 4
Value 1	< 0	0	> 0	
Value 2	< 0	0	> 0	
Operation	$+$	$-$	$\times$	$\div$

- **Provide test cases that satisfy Pair-Wise coverage**

Exercise 1.4

- Consider the following characteristics and blocks

Characteristics	Block 1	Block 2	Block 3	Block 4
Value 1	< 0	0	> 0	
Value 2	< 0	0	> 0	
Operation	$+$	$-$	$\times$	$\div$

- Number of pairs = 7 + 7 + 7 + 4 + 4 + 4 = 33
- Each test case can include 3 pairs;
- So, at least 11 test cases are required

Exercise 1.4

- Consider the following characteristics and blocks

Characteristics	Block 1	Block 2	Block 3	Block 4
Value 1	< 0	0	> 0	
Value 2	< 0	0	> 0	
Operation	$+$	$-$	$\times$	$\div$

- 12 test cases can satisfy **Pair-Wise** coverage

-2 -2 +,	0 0 +,	2 2 +,
0 -2 -,	2 0 -,	-2 2 -,
2 -2 ×,	-2 0 ×,	0 2 ×,
-2 -2 ÷,	0 0 ÷,	2 2 ÷,

Constraints Among Characteristics (6.3)

- Some combinations of blocks are **infeasible**
  - "less than zero" and "scalene" … not possible at the same time
- These are represented as **constraints** among blocks
- Two general types of constraints
  - A block from one characteristic **cannot be** combined with a specific block from another
  - A block from one characteristic can **ONLY BE** combined with a specific block form another characteristic
- Handling constraints depends on the criterion used
  - **ACC**, **PWC**, **TWC** : Drop the infeasible pairs
  - **BCC**, **MBCC** : Change a value to another non-base choice to find a feasible combination

Example Handling Constraints

- Sorting an array
  - **Input** : variable length array of arbitrary type
  - **Outputs** : sorted array, largest value, smallest value

Characte

- Length
- Type of
- Max val
- Min valu
- Position
- Position

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

Blocks must be combined

Input Space Partitioning Summary

- Fairly easy to apply, even with <u>no automation</u>
- Convenient ways to <u>add more or less</u> testing
- Applicable to <u>all levels</u> of testing – unit, class, integration, system, etc.
- Based only on the <u>input space</u> of the program, not the implementation

Simple, straightforward, effective, and widely used in practice