Homework #1 is posted.

Specification-based testing.

Equivalence partitioning testing.

1. Normal Eq. testing
   valid sub-domains

   (a) weak normal eq. testing.
   1-dim subdomains

   (b) strong normal eq. testing.
   multi-dim subdomains

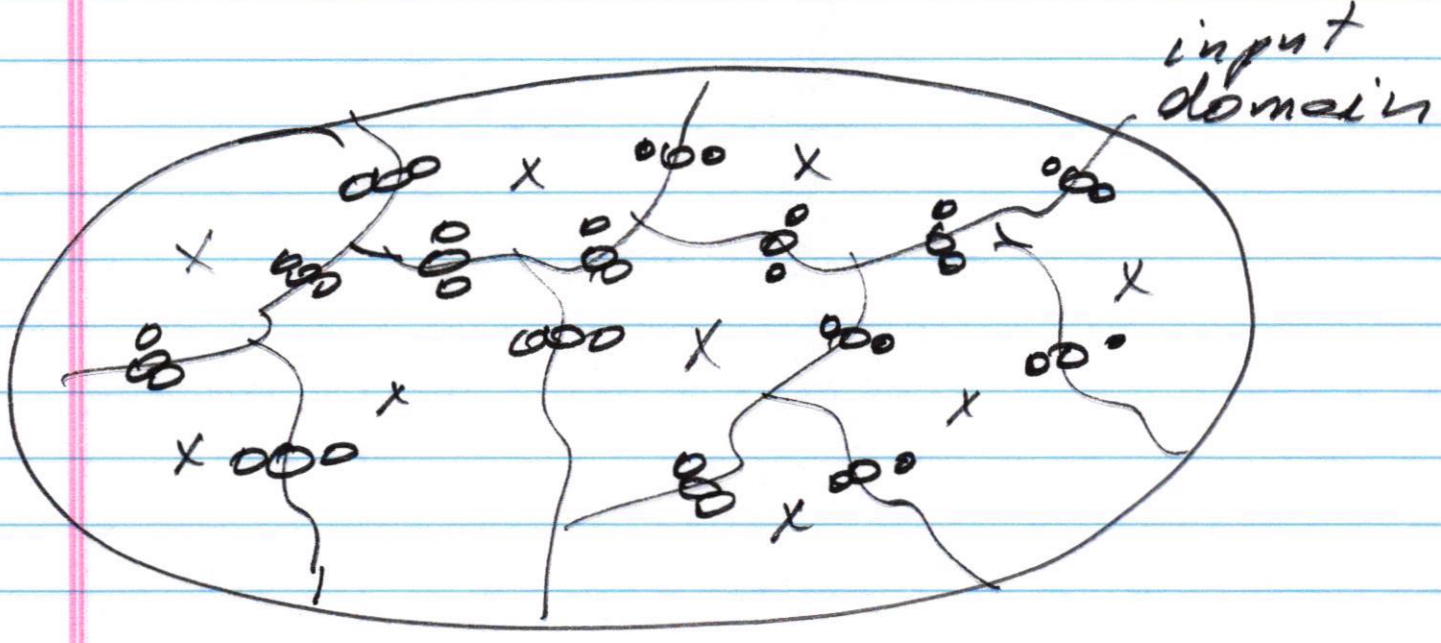
2. Robust Eq. testing.
   invalid subdomains.

   (a) weak robust eq. testing.
   1-dim subdomains.

   (b) strong robust eq. testing.
   multi-dim subdomains.

Testing boundaries

between subdomains.



Idea: select test

(a) on boundaries

(b) around — " —

1. Normal Boundary Value Analysis (BVA)

valid boundaries

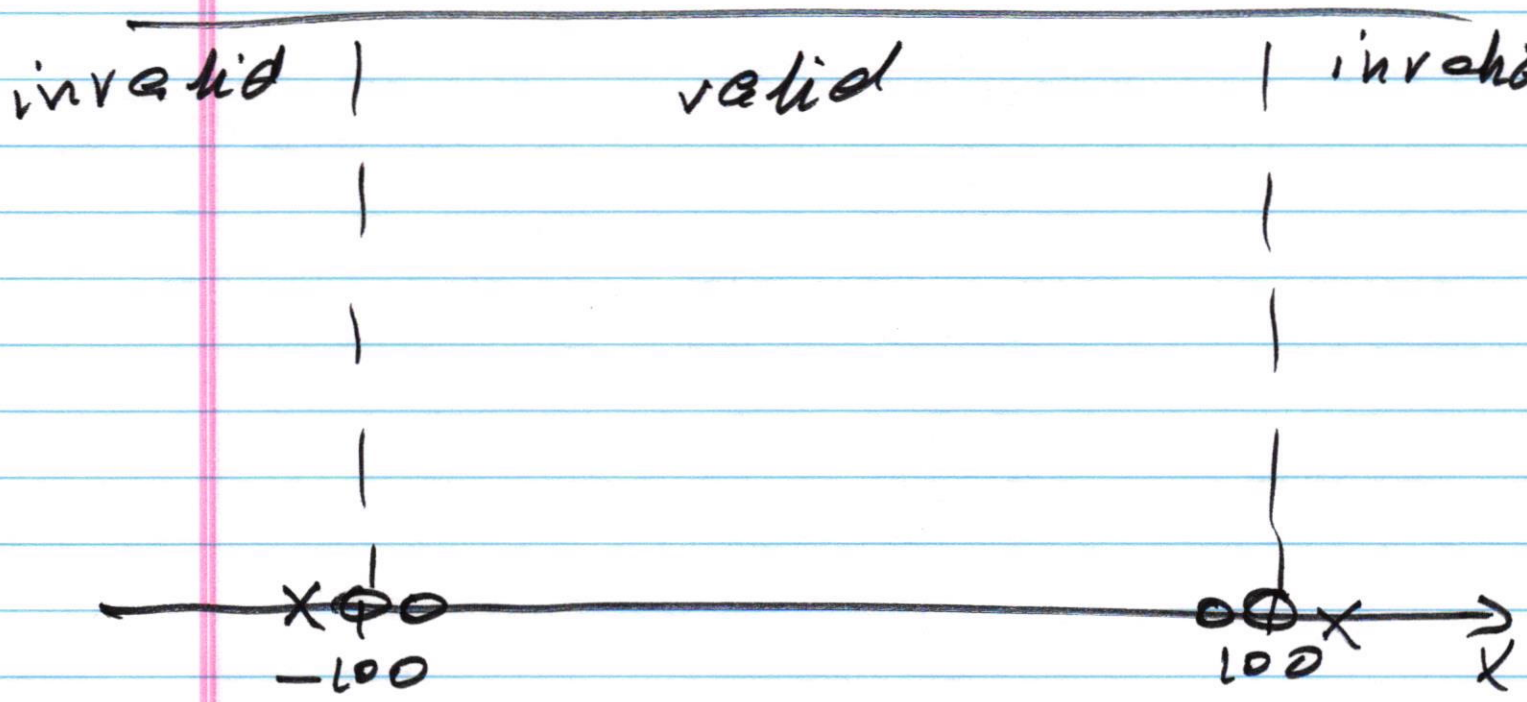
2. Robust boundary value testing.

boundaries of invalid subdomains.

1-dim subdomains.

x : integer

$-100 \leq x \leq 100$



o Normal BVA tests
x Robust boundary tests

Normal BVA tests	Robust boundary tests
T1: x = 100	T5: x = 101
T2: x = -100	T6: x = -101
T3: x = 99	
T4: x = -99	

Integer array dimension declaration

int n(d [,d]...)

n is the symbolic name of the array

d is a dimension declaration

Symbolic name can have 1-6 letters or digits, where the first character must be a letter.

The minimum # of dimensions = 1

The maximum # of dimensions = 7

The format of a dimension declaration

[lb:] ub

lb: lower bound; ub: upper bound

The bound may be in the range

-65,534 to +65,535

If lb is not specified, lb =1

ub $\geq$ lb

input conditions	valid sub-domain(s)	invalid sub-domain(s)
size of array name	1 - 6 (1)	0; (2) > 6 (3)
array name has letters/digits	has letters (4) has digit (5)	something (6) else
array name starts with a letter	yes (7) .	no (8) .
# of dimensions	1 - 7 (9)	0; (10) > 7 (11)
lower bound	-65,534 - +65,535 (12)	<-65,534 - (13)>+65,535 (14)
upper bound	-65,534 - +65,535 (15)	<-65,534 - (16)>+65,535 (17)
lower bound specified	yes (18) no (19)	
upper bound to lower bound	ub > lb (20) ub = lb (21)	ub < lb (22)

1. Normal boundary tests.

(a) Size of array name.

T1: int A(1:5)
T2: int ABCDE(1:5)
T3: int A7(1:5)
T4: int ABcDE(2:5)

(b) # of dimensions

T1
T5: int A(1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5)
T6: int A(1:5, 2:4)
T7: int A(1:5, 1:5, 1:5, 1:5, 1:5, 1:5)

(c) lower bound

T8: int A(65534:10)
T9: int A(65535:65535)
T10: int A(-65533:10)
T11: int A(65534:65535)

(d) upper bound

T12: int A(-65534:-65534)

T13: int A(1 :65535)

T14: int A(-65534:-65533)

T15: int A(1 :65534)

(e) ub to lb

ub > lb

ub = lb ✓

T16: int A(5:5)

T17: int A(4:5)

Robust BVA tests

(a) size of array name.

T1: int (1:5) already exists

T2: int ABCDEFG(1:5)

(b) # of dimensions.

T3: int A() already exists

T4: int A(1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5)

(c) . . . . . .

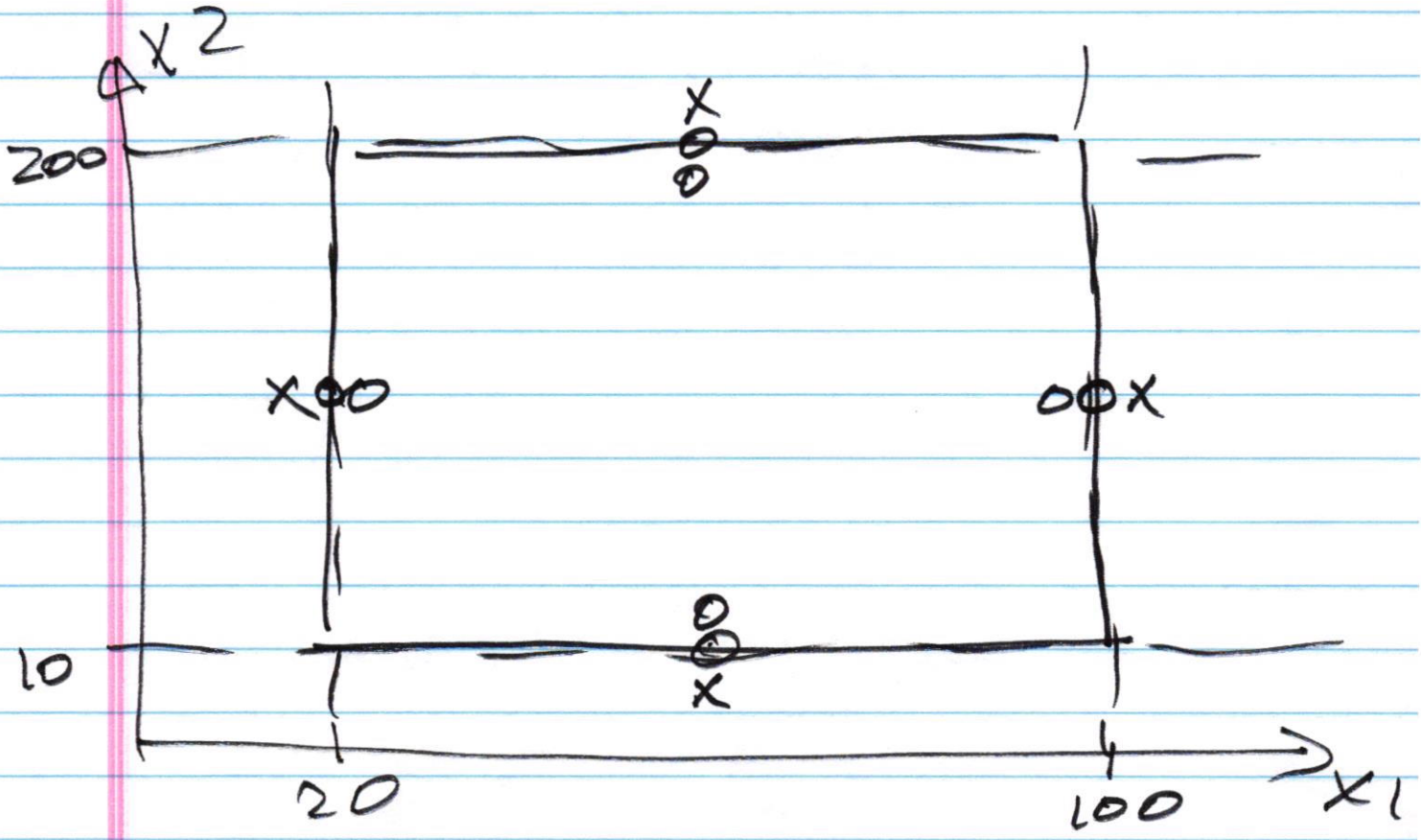
(e) ub < lb

T15: int A(5:4)

multiple inputs
multiple input conditions

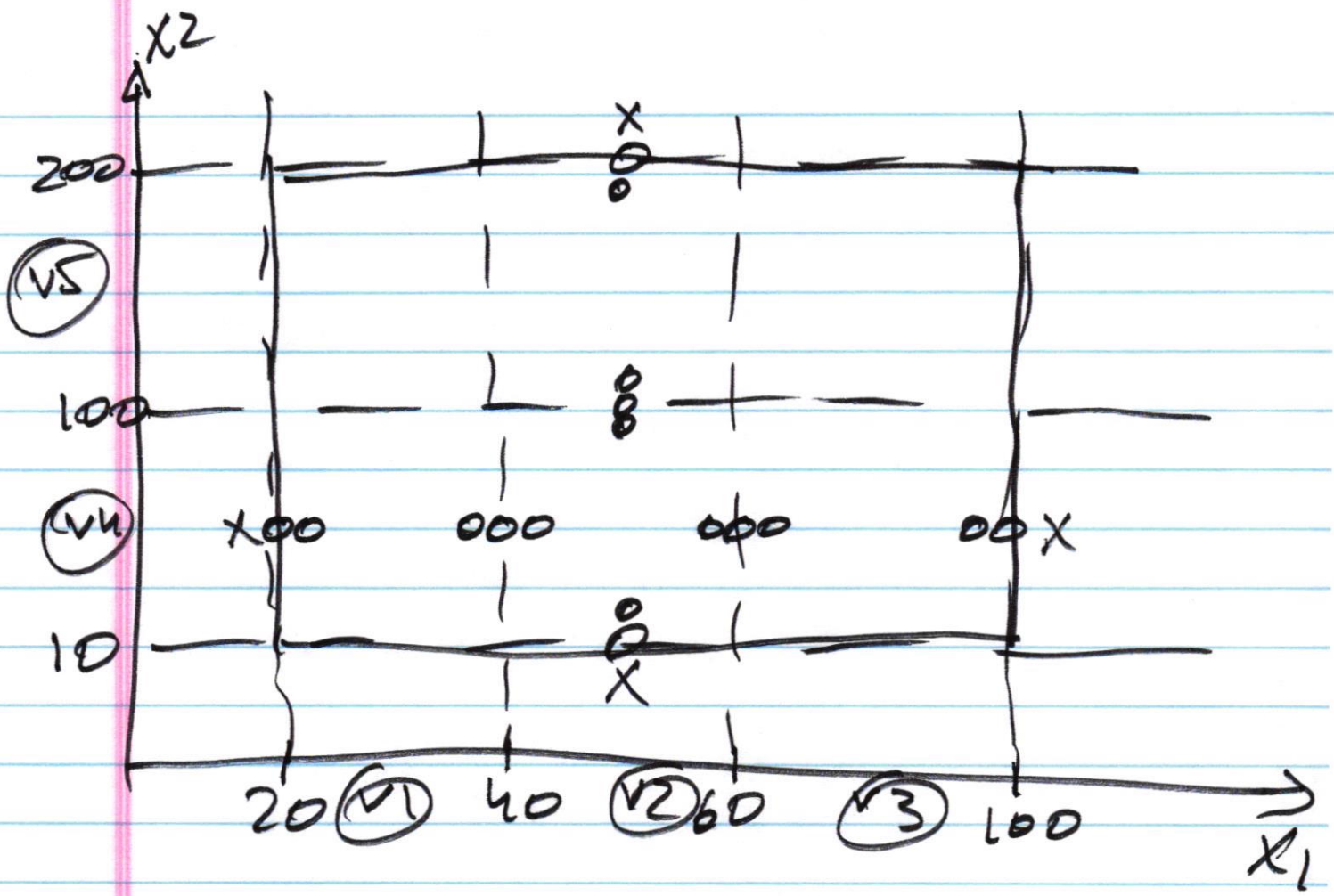
$20 \leq X_1 \leq 100$

$10 \leq X_2 \leq 200$



o Normal BVA tests

x Robust boundary tests



o Normal BVA tests

x robust boundary tests.

These methods assume 1-dim subdomains.

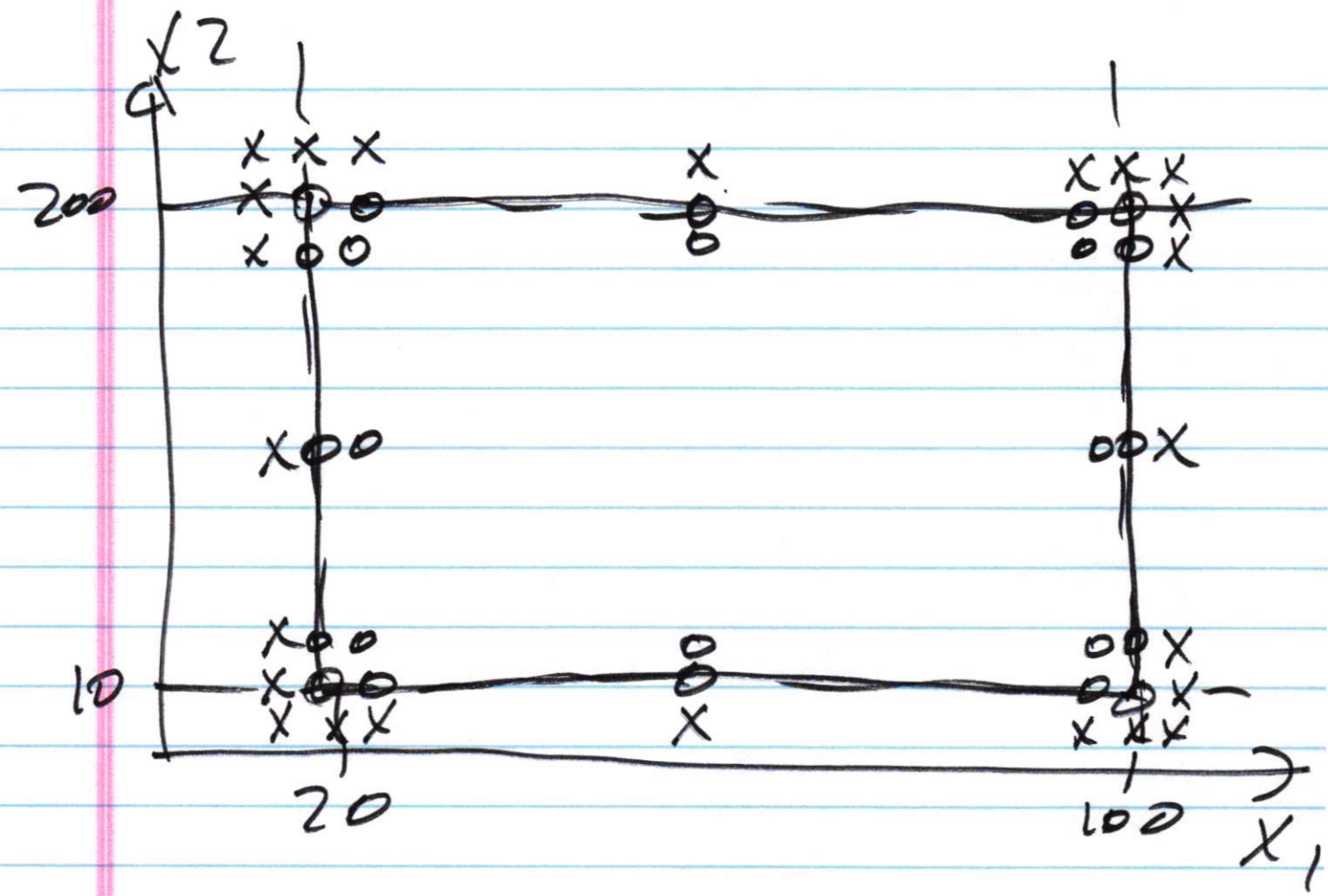
multi-dim subdomains/
boundaries

(1) worst case boundary
value testing.

valid multi-dim
boundaries

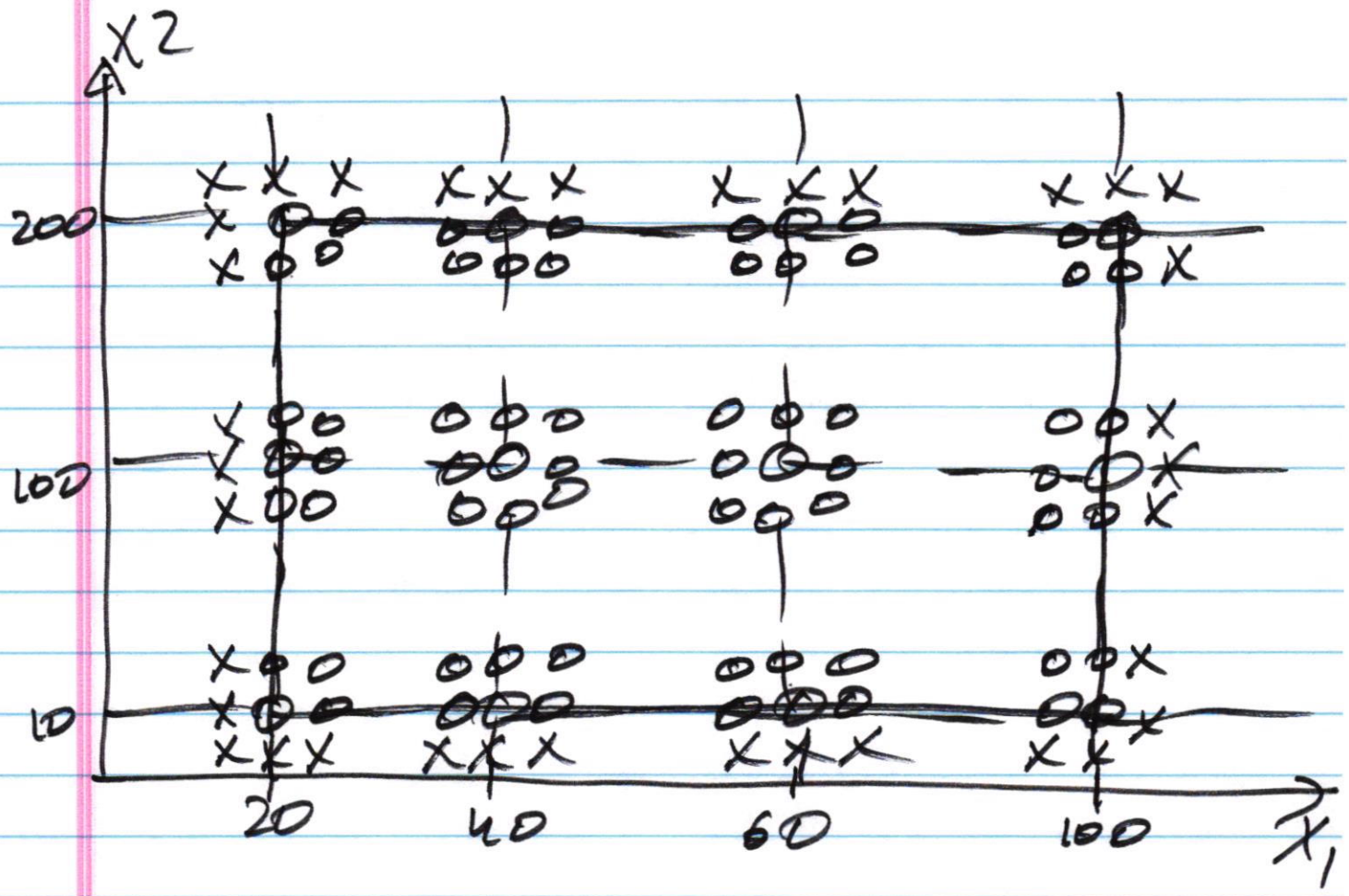
(2) Robust worst case,
boundary value testing.

invalid multi-dim
boundaries.



o worst case boundary value tests ✓

x robust worst case boundary tests



O worst case boundary tests

X robust worst case boundary tests

Decision Table testing.

specification

↓

input conditions

+

actions

outputs/outcomes

↓

decision table

↓

test cases

Decision Table

provides a simple tabular representation of complex logical decisions.

(1) conditions
(2) actions
(3) rules

condition = { True (T) / False (F) }

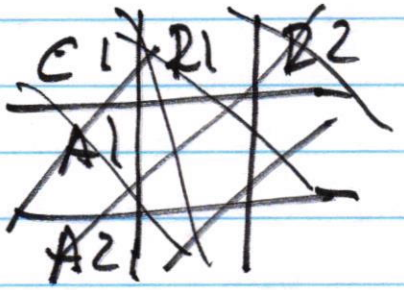
	R1	R2	R3
C1	T	F	T
C2	F	T	F
C3	T	T	F
A1	X		X
A2		X	X
A3		X	

C: condition
A: Action

completness

N: # of conditions

N = 1



	R1	R2
C1	T	F
A1	X	
A2		X

N = 2

	R1	R2	R3	R4
C1	T	T	F	F
C2	T	F	T	F
A1	X	X		X
A2		X	X	X

N	# of rules
N = 1	2
N = 2	4
N = 3	8
⋮	
N = 10	1,024
N	2^N

don't care cases

	R1	R2		R12
C1	T	T		T
C2	F	F		F
C3	T	F		—
C4	F	F		F
A1				
A2	X	X		X
A3				

don't care.

Decision-table testing.

specification

↓

input conditions
+
actions

↓

decision table
(a set of rules)

↓

test cases

at least one test per every rule.

input: 2 characters

(1) 1-st character must be "A" or "B"

(2) 2nd character must be a digit

(3) if 1st char is "A" or "B" and 2nd char is a digit, a file is updated

(4) if 1st char is not "A" or "B", msg12 is displayed

(5) if 2nd char is not a digit, msg13 is displayed

			T1	T2	T3	T4	T5	T6
	R1	R2	R3	R4	R5	R6	R7	R8
C1: 1st char is "A"	T	T	T	T	F	F	F	F
C2: 1st char B	T	F	F	F	T	T	F	F
C3: 2nd char is digit	T	F	T	F	T	F	T	F
A1: file is updated			X		X			
A2: msg 12							X	X
A3: msg 13				X		X		X
A4: impossible	X	X						

no tests

6 tests.

Tests

T1:	A5	R3
T2:	AC	R4
T3:	B7	R5
T4:	BC	R6
T5:	C4	R7
T6:	CX	R8

HOMEWORK ASSIGNMENT #1

CS589; Fall 2021
Due Date: **September 22, 2021**
Late homework 50% off
After **September 26,** the homework assignment will not be accepted.

This is an **individual** assignment. **Identical or similar** solutions will be penalized.

Submission: All homework assignments must be submitted on the Blackboard. The hardcopy submissions will not be accepted.

SPECIFICATION-BASED TESTING

Suppose a software component (called a Car Insurance System) has been implemented to handle processing the annual renewal of a hypothetical auto insurance policy. The following are requirements for the component:

> If the insured made one claim in the last year and is age 25 or older, the increase is $60 and no letter is sent. If the insured had no claims in the last year and is age 24 or younger, the increase is $60 and no letter is sent. If the insured had no claims in the last year and is age 25 or older, the increase is $35 and no letter is sent. If the insured made two, three, or four claims in the last year and is age 24 or younger, the increase is $310 and a warning letter is sent. If the insured made one claim in the last year and is age 24 or younger, the increase is $110 and a warning letter is sent. If the insured made two, three, or four claims in the last year and is age 25 or older, the increase is $210 and a warning letter is sent. If the insured made five or more claims in the last year, the policy is canceled.

The component accepts the input in the following format (6 input variables):

last name
first name
Age
car type
VIN
# of claims

The maximum size of the "*first name*" is 10 characters and "*last name*" is 15 characters. The *car type* can be: *sedan, mini-van*, *truck*, *SUV, Taxi*

Assumptions:

- *last name* and *first name* contain only letter characters.
- *age* is an integer. The minimum *age* is 18 and the maximum *age* is 110.
- The maximum # *of claims* is 10.
- VIN (a vehicle identification number) contains 17 characters (letters and digits) and has the following format: DLLLLDDLLLLDDDDDD, where D is a digit and L is a letter.

A sample component test:
Test #1:

last name	Smith
first name	John
Age	27
car type	Truck
VIN	1HGBH41JXMN109186
# of claims	3

PROBLEM #1 (35 points): Equivalence partition testing
For the Car Insurance System identify input conditions related to:

1. last name
2. first name
3. Age
4. Car type
5. VIN
6. # of claims

From the identified input conditions list valid and invalid sub-domains (equivalence classes). Based on identified sub-domains design test cases using:

a. Strong normal equivalence testing. — multi-dim
b. Weak robust equivalence testing. — 1-dim

Hint: Before designing test cases, identify related/unrelated input conditions.

PROBLEM #2 (30 points): Boundary-value testing
Based on identified sub-domains in Problem #1 design: 1-dim

1. Normal Boundary-Value Analysis test cases.
2. Robust Boundary test cases.

PROBLEM #3 (35 points): Decision-Table based testing

Suppose a software component (called a Grader component) has been implemented to automatically compute a grade in a course. A course taught at a university has two components: (1) two exams and (2) a project. To pass the course with grade C a student must score at least 50 points in Exam-1, 60 points in Exam-2, and 50 points in the Project. Students pass the course with grade B if they score at least 60 points in Exam-1, 65 points in Exam-2, and 60 points in the Project. If, in addition to this, the average of the exams and the Project is at least 75 points then students are awarded a grade A. Final grades for the course are: A, B, C, and E. The Grader component accepts four inputs:

Student #
Exam-1
Exam-2
Project

Assumptions:

- Assume *Exam-1, Exam-2,* and *Project* are integers and represent the scores of the exams and the project.
- The ranges for the exam scores and the project score are:
  - $0 \leq Exam\text{-}1 \leq 100$
  - $0 \leq Exam\text{-}2 \leq 100$
  - $0 \leq Project \leq 100$
- *Student #* is a number represented as a 9-character string in the following format: AXXXXXXXX, where X is a digit.

Sample test cases for the Grader component:

Test #1: *Student #*=A11112222, *Exam-1*=57, *Exam-2* = 64, *Project*=55
Test #2: *Student #*=A42312242, *Exam-1*=75, *Exam-2* = 24, *Project*=85

Use **decision-table** based testing to design test cases to test the GRADER program. Provide a decision table and test cases derived from the decision table.

Note: In your solution conditions cannot be complex logical expressions. For example:

(Exam-1<50) and (Exam-2 ≥ 60)

is **not acceptable** as a condition in the decision table. However, "*Exam-1 <50*" is a condition; "*Exam-2 ≥ 60*" is a condition.