Black Box Testing Techniques

Dr. John H Robb, PMP, IEEE SEMC
UTA Computer Science and Engineering

Black Box Testing Techniques

- Typical “black-box” test analysis and design techniques include:
  - Equivalence partitioning
  - Boundary value analysis
  - Decision table testing
  - State transition analysis
  - Use Case testing
  - Decision logic and Karnaugh maps

- Black box is a bit of a misnomer – we don’t and can’t test strictly black box

- So these are correctly called Specification based test analysis and design techniques, but I use the term “black box” because it is so commonly used

A Brief Taxonomy of Software Bug Types

- We've talked about Requirements elicitation and challenges associated with requirements elicitation and specification
  - Incomplete requirements
  - Incorrect/erroneous requirements
  - Missing requirements (errors of omission)

- A discussion of these helps to identify typical problems in requirements so that we can avoid them if possible and deal with them if need be

- Before we begin software testing techniques we want to understand some of the issues with software test - this will help us understand why we are doing these things

- We are going to utilize specific techniques intended to address these

A Brief Taxonomy of Software Bug Types (cont.)

- First we need to shed any idea that software will never have bugs - we already discussed both latent defects and deficiencies
  - Both of these are present in even the most mature and high-quality software
  - Even the most mature software teams and processes generate plenty of bugs - they are just very good at eliminating them

- Software is a creative enterprise - it will always have bugs - we need to expect them and devise ways to eliminate them

- An assessment of software bug types is important for two reason
  1. It will help us to tailor the test techniques to a specific area of software development
  2. Understanding the kinds of defects that we typically generate will help us become better software developers

A Brief Taxonomy of Software Bug Types (cont.)

- Here is a brief list of the different kinds of software bugs - these are not limited to code or specification - most represent a type of defect that can be present in many types of software artifacts

- Logic error
- semantic error
- syntactic error
- off-by-one error
- infinite loop error
- memory corruption
- memory leak
- out of bounds index
- uninitialized pointers
- uninitialized variables
- uninitialized objects
- priority inversion
- buffer overflow
- buffer overread
- race condition
- pigeon hole principle
- zero based numbering
- deadlock
- handle leak

A Few Examples

- Off by one (OBE) error
  - I need to build 40 feet of fence with 8 foot fence panels - how many fence posts do I need?
  - I worked April 8th to April 11th - how many days did I work?
- Pigeonhole principle
  - I have If you have 3 books and you want to put them into 2 drawers - how many drawers will have 2 books? 1 book?
  - A program has inputs *a, b, c which represent three edges of a* triangle, the output is “not a triangle”, “normal”, “isosceles”, “equilateral triangle”.
  - If a tester said that
  - he used 5 inputs to test, we know that at least one kind of triangle is tested at least twice
  - If he used 3, we know at least one is not tested at all.

A More Detailed Look

- Syntax (grammatical) error - an error that does not compile or does not do what the language intended
  - invalid or other usage intended in the language
  - use of a "=" instead of a "==" in an if statement
  - uninitialized local variable, object never created, etc
- Semantic error - an error that both compiles and does what the language intended
  - Logic error (general) - missing brackets in calculations, accidental infinite loops, see dead code next, erroneous logic
  - Dead code/path/requirement - unreachable due to design problems
  - Off by one error - classic fencepost error - this can be either on indexes, or on boundary conditions. Zero based numbering.
  - Out of bounds indexes in arrays - causing either over-write or under-write. Can also cause over or under-reads.
  - Division by zero - overflow - underflow - NaN issues
  - Priority inversion - lower priority tasks starving off higher priority tasks

A More Detailed Look (cont.)

- Semantic error - an error that both compiles and does what the language intended (cont.)
  - Logical expressions - errors in the use of wrong operands, operators, sequences, or expressions
  - Race condition - a timing defect in some internal logic that causes the incorrect output (typically occasionally)
  - Deadlock - when two or more contending actions are waiting for each other - more than just a priority issue
  - Handle leak - when a software resource allocates a handle to a resource but does not free it

- Semantic errors are not constrained to code - they can reside anywhere!

- Anywhere starts with requirements and so will we

Another Interesting Taxonomy of Defects

- Testing Computer Software by Kaner, Falk, and Nguyen contains a detailed taxonomy consisting of over 400 types of defects and are summarized below.

User Interface Errors	**Functionality**
	Communication
	Command structure
	Missing commands
	Performance
	Output
Error Handling	**Error prevention**
	Error detection
	Error recovery
Boundary-related errors	**Numeric boundaries**
	Boundaries in space, time
	Boundaries in loops
Calculation errors	**Outdated constants**
	Calculation errors
	Wrong operation order
	Overflow and underflow
Initial and Later States	**Failure to set a data item to 0**
	Failure to initialize a loop control variable
	Failure to clear a string
	Failure to reinitialize
Control flow errors	**Program runs amok**
	Program stops
	Loops
	If, Then, Else or maybe not

Another Interesting Taxonomy of Defects (cont.)

Errors in handling or interpreting data	Data type errors
	Parameter list variables out of order or missing
	Outdated copies of data
	Wrong value from a table
	Wrong mask in a bit field
Race Conditions	Assuming one event always finishes before another
	Assuming that input will not occur in a specific interval
	Tasks starts before its prerequisites are met
Load conditions	Required resource not available
	Doesn't return unused memory
Hardware	Device unavailable
	Unexpected end of file
Source and vision control	Old bugs mysteriously reappear
	Source doesn't match binary
Documentation	None
Testing errors	Failure to notice a problem
	Failure to execute a planned test
	Failure to use the most promising test cases
	Failure to file a defect report

The techniques we address in this class are specifically geared to address these!

Why Requirements?

- Why are requirements a problem in terms of specification - one major problem is that they are typically language based

- Any language has communications issues - here are some snippets of classified ads - see if you can spot the problems
  - 2 female Boston Terrier puppies, 7 weeks old, Perfect markings, 555-1234. Leave mess.
  - Dinner Special -- Turkey $2.35; Chicken or Beef $2.25; Children $2.00.
  - For sale: an antique desk suitable for lady with thick legs and large drawers.
  - Now is your chance to have your ears pierced and get an extra pair to take home, too.
  - We do not tear your clothing with machinery. We do it carefully by hand.
  - Tired of cleaning yourself? Let me do it.
  - Mt. Kilimanjaro, the breathtaking backdrop for the Serena Lodge. Swim in the lovely pool while you drink it all in.
  - Get rid of aunts: Zap does the job in 24 hours.

Why Requirements (cont.)?

- We use very imprecise words in requirements
  - Do not use generalities where numbers are really required such as
    - "immediate", "large", "rapid", "many", "timely", "most", or "close".
  - Avoid fuzzy words that have relative meanings such as
    - "accommodate", "easy", "normal", "adequate", " sufficient" or
    - "effective", "be able to", "may", "can", "when necessary"
  - Avoid unclear boundary values such as:
    - "not limited to", "at a minimum", "maximize", "minimize", "between", "above", "below", "throughout"

Why Requirements (cont.)?

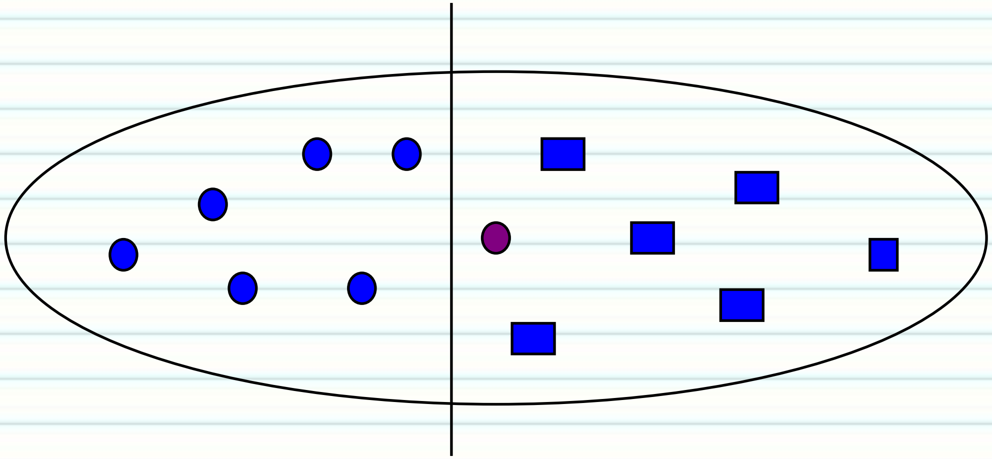
- We use very imprecise words in requirements (cont.)
- Until
  - Out of the office until 2/2/16
  - In the office until 2/2/16
  - She studied until dawn.
  - She kept studying right up until dawn.
  - He did not give in, even until his last breath.
  - He did not give in, even up until his last breath.
- From to
  - The museum is open daily from Monday to Thursday
  - The museum is closed daily from Friday to Sunday
- Through
  - I am out of the office through 2/2/16
- Since
  - I have started homework 1 since yesterday

Off By One Error

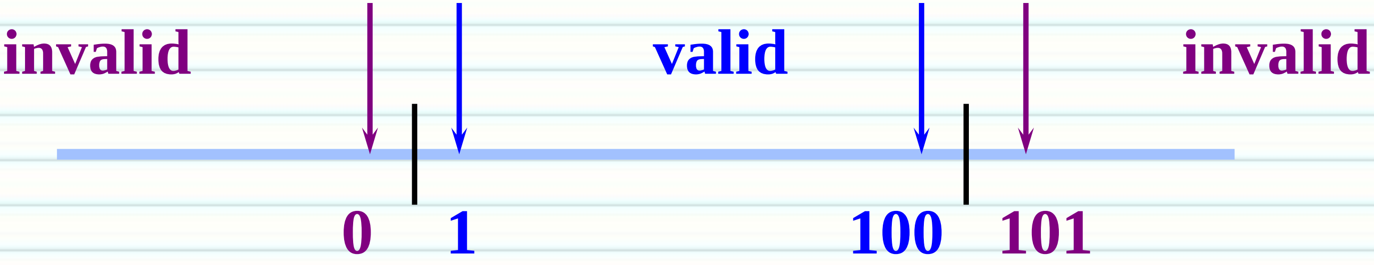
- This is the most prevalent error in software and is caused by one or more of the following
  - Logical error of inclusion or exclusion "fencepost error"
  - Use of incorrect operators
  - Zero based vs. physical numbering
  - Imprecise prepositional words/phrases

- We can avoid this error by one of several techniques
  - Knowledge - being aware that it is a major problem
  - Term Replacement - replace the usage of terms that are unclear with more specific indicators (including/excluding)
  - Test - we can design our tests to detect these types of errors - one such test design strategy is call "boundary value analysis"

Boundary value analysis (BVA)



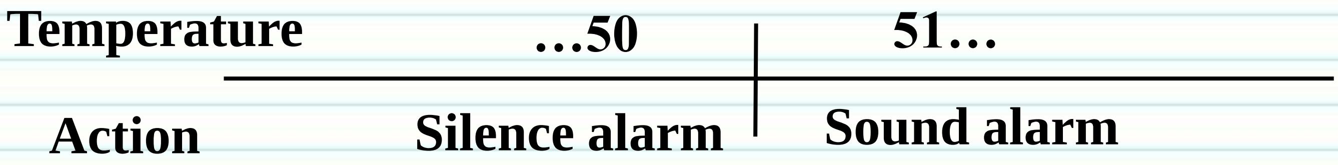
- faults tend to lurk near boundaries
- good place to look for faults
- test values on both sides of boundaries
- they test the use of correct logical operators - e.g. ">" vs. ">="
- they test the presence of off-by one errors



(c) JRCS 2016

Boundary Value Analysis

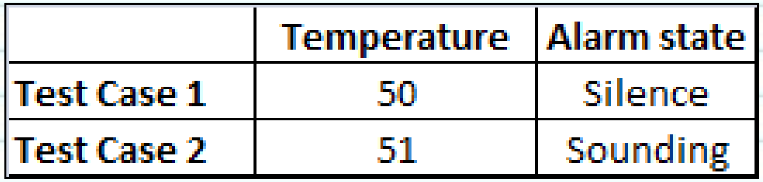
- Which of the following is more precise as a requirement statement?
  - If the temperature is greater than 50 (degrees F) then sound the refrigerator alarm otherwise silence the alarm, or
  - refrigerator alarm = (temp>50)
  - (in both cases assume temperature is an int)
- We set inputs based on each “border” of the partition and check actual outputs to the expected condition



Temperature ...50 | 51...

Action Silence alarm | Sound alarm

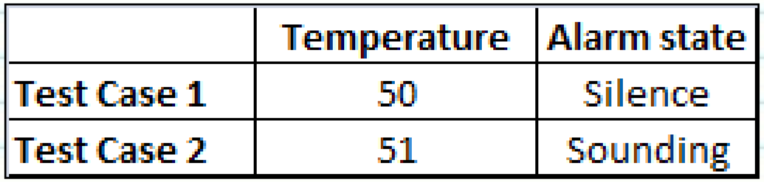
- We will have **a minimum of** two test cases



	Temperature	Alarm state
Test Case 1	50	Silence
Test Case 2	51	Sounding

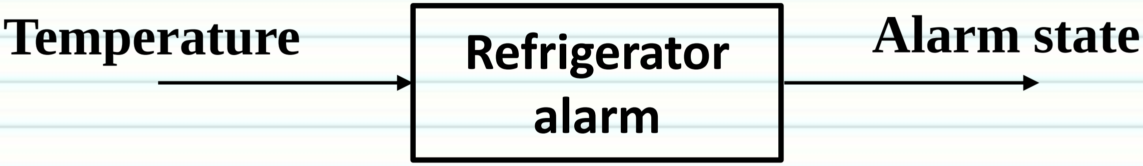
Let's Discuss Test Cases

- In the previous example we said that we would have a **minimum of** two test cases and we showed



	Temperature	Alarm state
Test Case 1	50	Silence
Test Case 2	51	Sounding

- If we drew this function as a black box it would look something like this:



- From a requirements perspective it has an input (temperature) and an output (alarm state)
- From a test perspective it has three values: an input, an output, and an expected output
- We develop the test cases using the input and expected output. The latter comes from the test oracle (which is the requirement here).
- We execute the test and compare the actual and expected output. If they disagree the test fails - otherwise it passes.

Class Exercises

- Develop the boundary values for the following:

1. alarm = (temp>=50)
2. alarm = (temp<-32)
3. alarm = (temp<=-32)
4. alarm = (temp > 50.0), assume a 0.1 significance
5. alarm = (temp >= 50.0) , assume a 0.1 significance
6. alarm = (temp <=-32.1) , assume a 0.1 significance
7. alarm = ((temp > 50) & (temp <60))
8. alarm = ((temp > 50) & (temp > 60)
9. alarm = ((temp < 50) & (temp < 60))
10. alarm = ((temp < 50) & (temp > 60))

Class Exercises

- Develop the boundary values for the following:

1. alarm = (temp>=50) 49, 50
2. alarm = (temp<-32) -33, -32
3. alarm = (temp<=-32) -32, -31
4. alarm = (temp > 50.0), assume a 0.1 significance, 50.0, 50.1
5. alarm = (temp >= 50.0), assume a 0.1 significance, 49.9, 50.0
6. alarm = (temp <=-32.1) , assume a 0.1 significance, -32, -32.1
7. alarm = ((temp > 50) & (temp <60)), 50, 51, 59, 60
8. alarm = ((temp > 50) & (temp > 60), 60, 61
9. alarm = ((temp < 50) & (temp < 60)), 49, 50
10. alarm = ((temp < 50) & (temp > 60)), {null}

Boundary Value Analysis (cont.)

- Examining the full range of the input

Temperature	**…50**	**51…**
Action	**Silence alarm**	**Sound alarm**

- I made some assumptions about the partitions and boundaries above – what are they?
- So let’s say the range of temperatures that we measure is from

  -100 <= Temp <= 100 C
- Assume a significance of 0.1, what are the partitions?
- Partition 1: Temp = -100.1
- Partition 2: Temp = -100,50
- Partition 3: Temp = 51,100
- Partition 4: Temp = 100.1
- For Floating Point values we always need a significance and range
- For Integer values we always need a range

Boundary Value Analysis (cont.)

- There are different thoughts about testing boundary values
  - Some use **three** values to test boundary conditions – at the boundary and some incremental distance either side (e.g. BS7925-2)
  - The **two** test values approach is more efficient – we will use two point strategies in this course, but be aware that **you need to understand and be prepared to use either approach in your career**
  - Remember the number we are working toward is the **minimum number of test cases** needed to achieve coverage

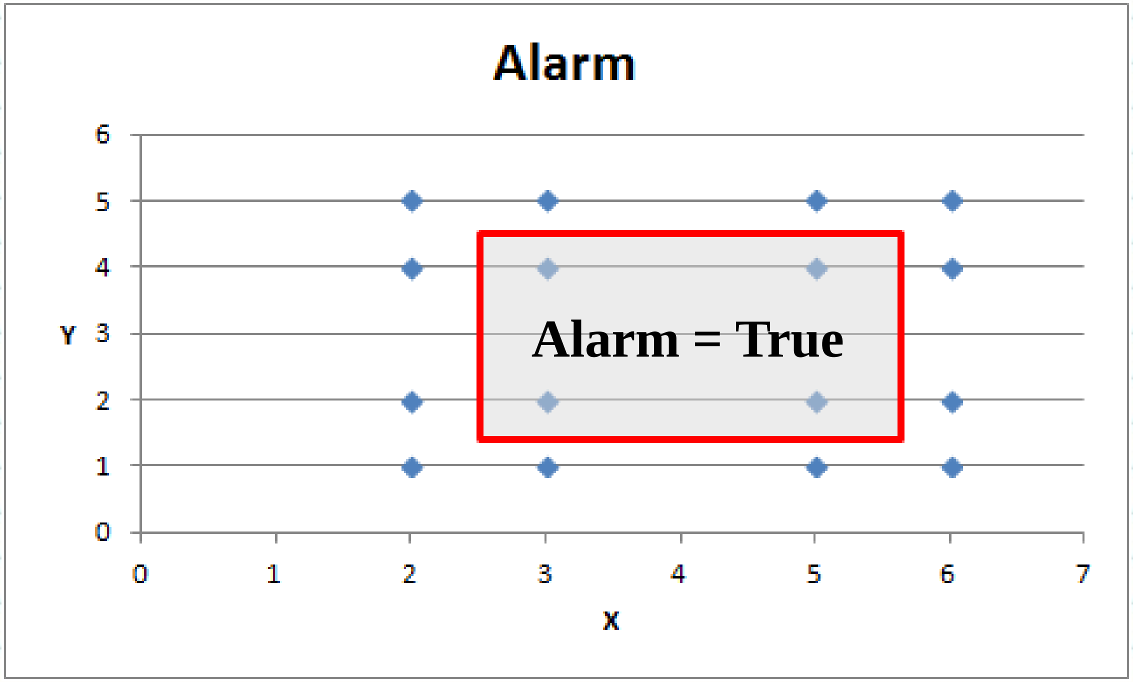
- Another example Alarm = (temp >= 50) – assume integer
  - Partitions are …49 and 50… - what is the boundary value?
  - So two points would be 49 and 50, three points would be 49, 50, 51
  - What if I change this back to Alarm = (temp > 50) – what is the boundary value? What two points and three points do I use?
  - What additional question should I ask if the example above is made a floating point value? See next

Boundary Value Analysis (cont.)

- Let’s apply this to Two-dimensional Specifications
- Suppose we have a specification that says
- For {(2 < x < 6) and (1 < y < 5)} alarm =True, otherwise alarm=false
- Alarm = ((x>2) && (x<6)) && ((y>1) && (y<5)), assume x,y are integers
- What are the boundaries for x? What are the boundaries for y?
- Class exercise – what are the boundaries for x & y? There are 16.
- If we assume x and y can range from (0 < x <=10) and (0 < y <=10) how many total test cases are in our problem?
- How can I get this down to a reasonable set?

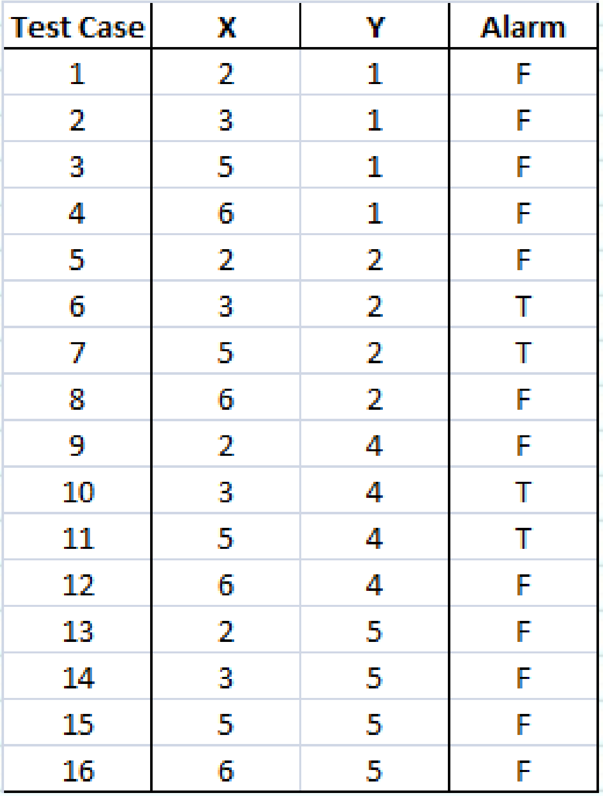
Boundary Value Analysis (cont.)

- Here is what the specification looks like (is this easier to visualize?)



Boundary Value Analysis (cont.)

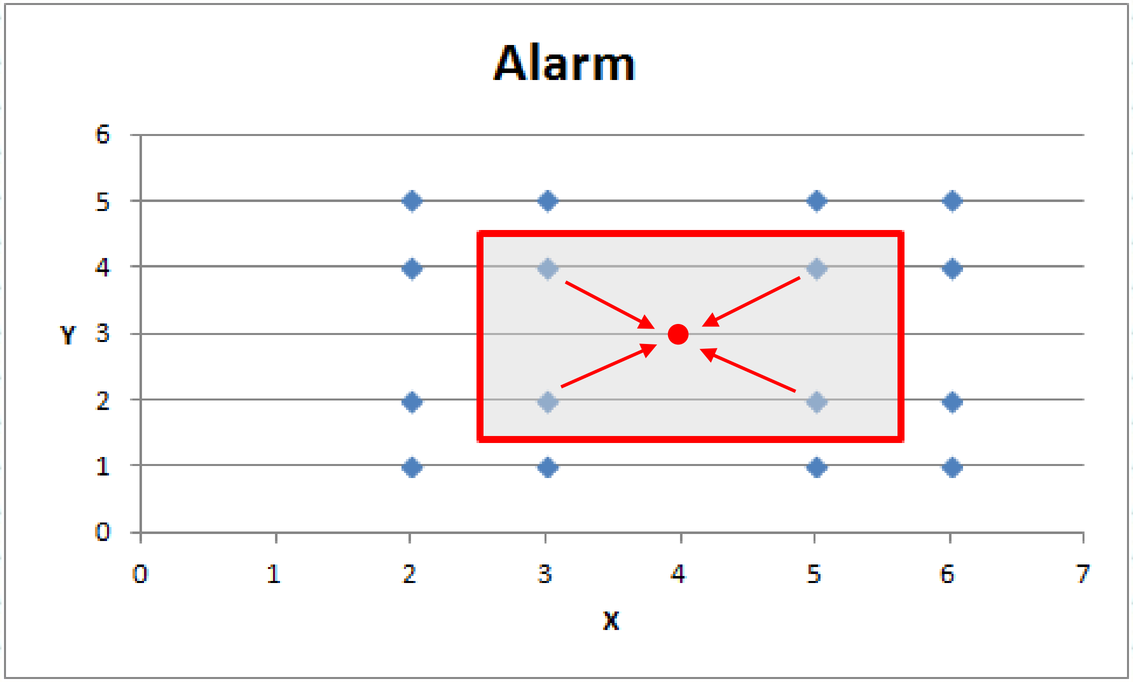
– Here is the corresponding Truth Table



Test Case	X	Y	Alarm
1	2	1	F
2	3	1	F
3	5	1	F
4	6	1	F
5	2	2	F
6	3	2	T
7	5	2	T
8	6	2	F
9	2	4	F
10	3	4	T
11	5	4	T
12	6	4	F
13	2	5	F
14	3	5	F
15	5	5	F
16	6	5	F

Boundary Value Analysis (cont.)

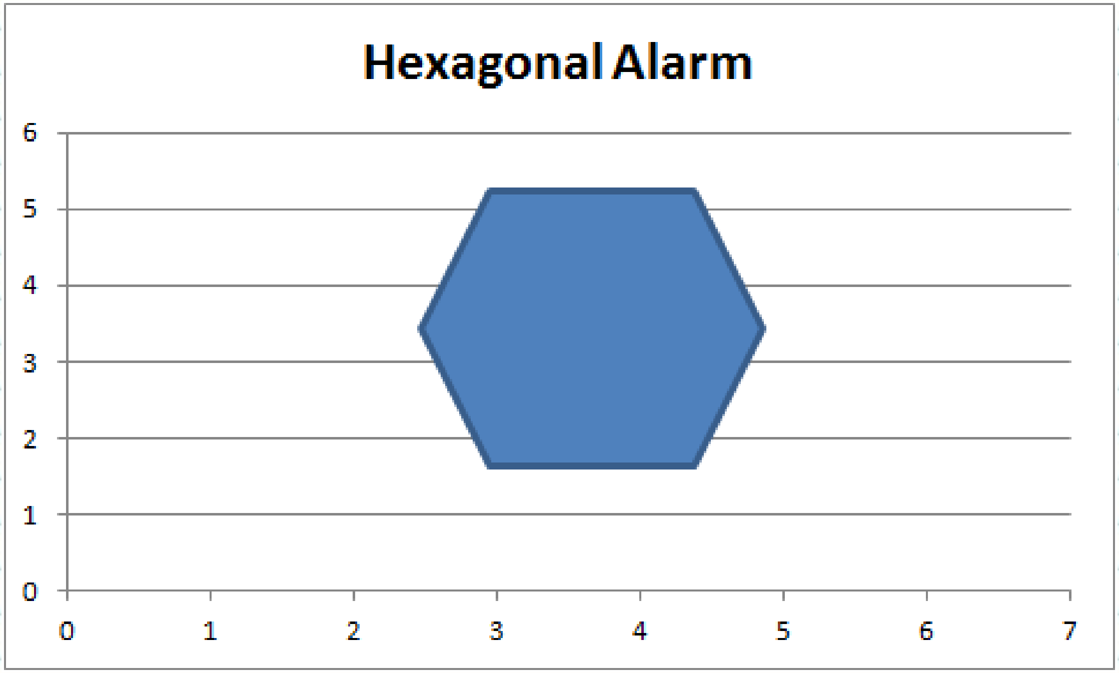
- Could I reduce the number of Test Cases?



- Is there a problem with doing this?
- So what is the minimum number of test cases?

Boundary Value Analysis (cont.)

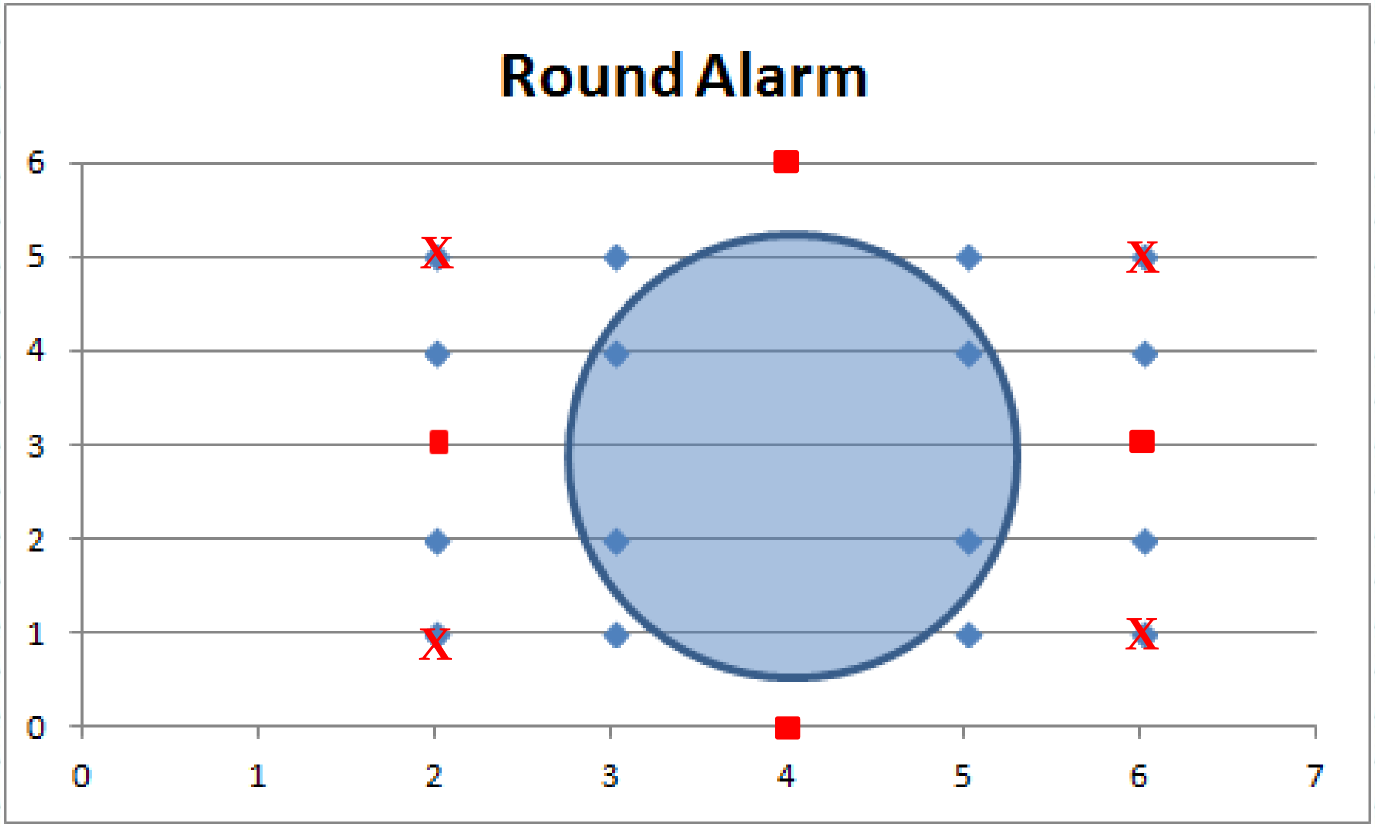
– Can I extend this to other shapes?



– As with the square I will use 4 points on edge corner.

Boundary Value Analysis (cont.)

- Can I extend this to other shapes?



- What values would I use?

Boundary Value Analysis (cont.)

- What are the limits of boundary values?
- What if I have a range of extensions in an office: 100-199?
- Area codes: what are the valid area codes for DFW?
  - 817,214,672, etc
- What about enumeration types:
  - {freezing, very cold, cold, warm, hot, very hot}?

- There are more advanced types of BVA that we are not going to use in this class
  - Worst case BVAs

Equivalence Classes or Partitions

- Contrary to BVA (used to test off-by-one errors) - ECPs do not check for error conditions!
- They are a test minimization technique - used to reduce a large input space down into a more manageable set
- Let's use our previous example

Temperature	**…50**	**51…**
Action	**Silence alarm**	**Sound alarm**

- I have two equivalence classes based on the actions needed – they are equivalent because the actions taken are equivalent within the same partition **Equivalence Classes are always based on actions**
- A domain partition is a partition of the input domain into a number of sub-domains - this is also known as **Input Space Partitioning**
- A boundary is where two sub-domains meet

Equivalence Classes or Partitions (cont.)

- So instead of testing from
  - -min temp to +max temp one degree at a time, I can minimize the number of tests based on the equivalent actions
  - reduction of test is a very important issue that we will be addressing throughout the semester
  - we are always making an engineering trade-off between test effort and coverage - using our bug taxonomies to guide us

Temperature	min…50	51…max
Action	Silence alarm	Sound alarm

min 50	51 max

- By combining BVA and ECP I can develop four tests that fully test all actions and test both boundary values

	Temperature	Alarm State
Test Case 1	min	silence
Test Case 2	50	silence
Test Case 3	51	sounding
Test Case 4	max	sounding

Equivalence Classes or Partitions (cont.)

- A savings account in a bank has a different rate of interest depending on the balance in the account.

1. 5% rate of interest is given if the balance in the account is between $100 to $1000
2. 3% rate of interest is given if the balance in the account is less than $100
3. 7% rate of interest is given if the balance in the account is at least $1000

Student exercise - identify the BVAs and ECPs

Equivalence Classes or Partitions (cont.)

- A savings account in a bank has a different rate of interest depending on the balance in the account.

1. 5% rate of interest is given if the balance in the account is between $100 to $1000
2. 3% rate of interest is given if the balance in the account is less than $100
3. 7% rate of interest is given if the balance in the account is at least $1000

Balance	0:99.99	100:999.99	1000.00:+
Interest	3%	5%	7%

- We need to pay very careful attention to the range of values
- What would happen to the ranges above if this was float but not dollars and cents? How would this change when we represent this in code?
- Here we have three ECPs based on the set of actions

Equivalence Classes or Partitions (cont.)

- A savings account in a bank has a different rate of interest depending on the balance in the account.

1. 5% rate of interest is given if the balance in the account is between $100 to $1000
2. 3% rate of interest is given if the balance in the account is less than $100
3. 7% rate of interest is given if the balance in the account is at least $1000

Balance	0:99.99	100:999.99	1000.00:+
Interest	3%	5%	7%

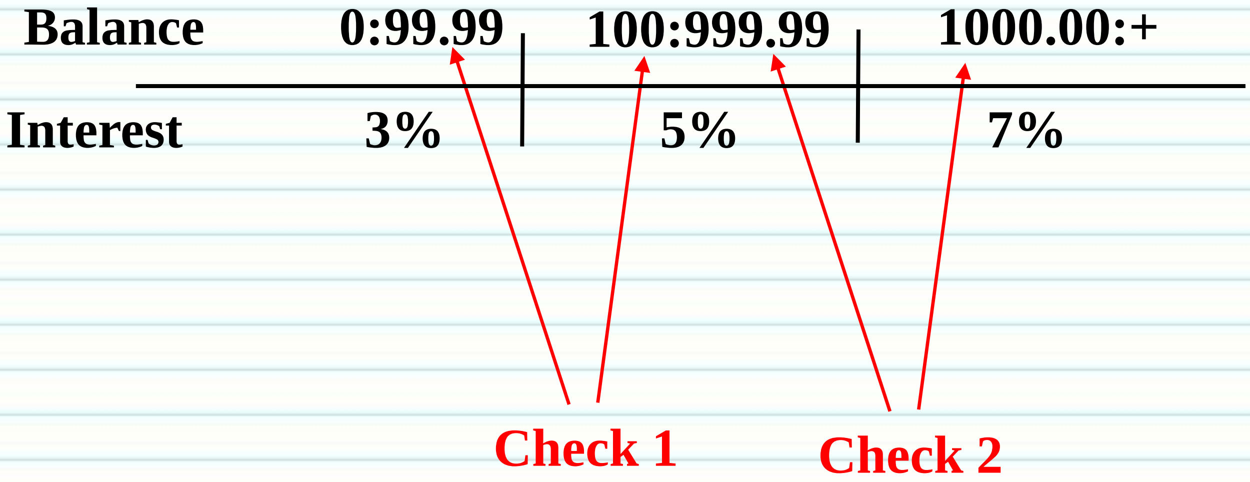
- Student exercise - develop the test cases needed to test this ECP/BVA solution (do not round up currency values!)

Equivalence Classes or Partitions (cont.)

	Input	Expected Output	
Test Case	Balance	Interest	Balance
1	$0.00	$0.00	$0.00
2	$99.99	$2.99	$102.98
3	$100.00	$5.00	$105.00
4	$999.99	$49.99	$1049.98
5	$1,000.00	$70.00	$1,070.00
6	2,000.00	$140.00	$2,140.00

Equivalence Classes or Partitions (cont.)

- One quick check of a ECP/BVA depiction is to look at the edges of each partition - the end should be right up against the start of the next
- If not, there is an error in the boundary values

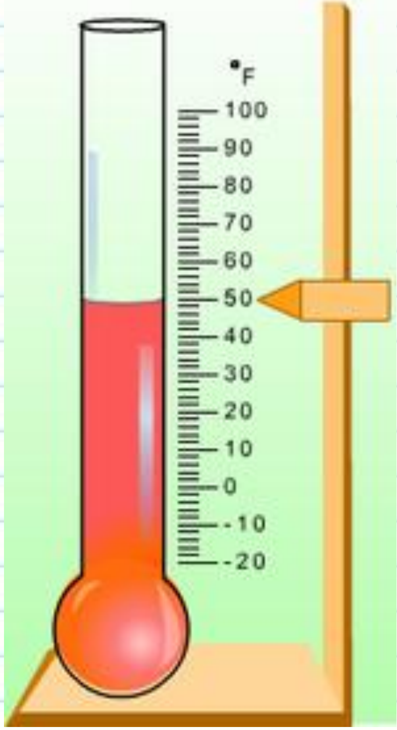


- For n ECPs, there will be n-1 checks - always do these checks - it is too easy to miss the BVs

ECPs and Invalid Data

- So far we have only looked at valid data for our ECP - ECPs are also useful for examining the invalid data values as well

- From the thermometer for our alarm problem
  - alarm = (temp>50), assume integers
- We can see that the valid ranges of data are
  - -20 to 100 degrees F
- So the valid partitions would be
  - -20:50 inclusive
  - 51:100 inclusive
- Invalid partitions are
  - -min int:-21 inclusive
  - 101: +max int
- Notice we have two types of invalid boundary values:
  - those driven by the thermometer and
  - those driven by the numerical representation (here int)



Class Exercise

- Student exercise
- Given the following enumeration for months, provide the ECPs (both invalid and valid) for the number of days in the given month - ignore leap-year
  - {Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec}

Class Exercise

- Given the following enumeration for months, provide the ECPs (both invalid and valid) for the number of days in the given month - ignore leap-year
  - {Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec}

- Valid ECPs are: 28, 30, 31
- Invalid ECPs would be -min int:27, inclusive and 32:max int inclusive
- Typically for robustness (or stress) testing - where we try to break or stress the software, we would test with Invalid partitions also.
- For this class I only want the valid ECPs unless otherwise stated.
- What is the minimum number of valid ECP test cases and what are they?
- Is this sufficient? It depends on the type of application - for safety or security critical it may not be enough.

ECPs and Types

- We have seen that ECPs are not like BVs and are limited to numerical types - this is because the action is what drives ECPs - so anything with an equivalent action is in the same ECP.

- ECPs can apply to numerical, enumeration, strings, area code, extensions, etc - any kind of representation that has an equivalent action.

Applying ECP/BVA Elsewhere

- We can apply Boundary Value and Equivalence classes/partitions to other areas as well
- Suppose an internal phone system for a company with 200 telephones has a 3-digit extension numbers from 100 to 699, we can identify the following classes/partitions and boundaries (Example from ISTQB Foundations)
- Digits: Valid partition 0-9, Invalid partition otherwise
- Number of Digits: Valid partition number 3 digits, Invalid boundary values of 2 and 4 digits
- Range of extension numbers: Valid partition 100 to 699, Two invalid boundaries of 099 and 700
- Assigned extensions: at least two partitions – Valid=assigned and Invalid=not assigned
- Boundary value of assigned extensions: lowest and highest could also be used
- There are more advanced cases of ECPs that are outside this class: strong and weak EC testing for example.

Mental Break 2

- A programmer and a software engineer are sitting next to each other on a long flight from San Jose to Bangalore. The programmer leans over to the software engineer and asks if he would like to play a fun game. The software engineer just wants to take a nap, so he politely declines and rolls over to the window to catch a few winks.

- The programmer persists and explains that the game is easy and fun. He explains "I ask you a question, and if you don't know the answer, you pay me $10. Then you ask me a question, and if I don't know the answer, I'll pay you $10." Again, the software engineer politely declines and tries to get to sleep.

- The programmer, now some what agitated, says, "OK, if you don't know the answer you pay me $10, and if I don't know the answer, I'll pay you $100!" This catches the software engineer's attention, and he sees no end to this torment unless he plays, so he agrees to the game. The programmer asks the first question. "What's the distance from the earth to the moon?" The software engineer doesn't say a word, but reaches into his wallet, pulls out a ten dollar bill and hands it to the programmer. Now, it's the software engineer's turn. He asks the programmer "What goes up a hill with three legs, and comes down on four?"

- The programmer looks up at him with a puzzled look. He takes out his laptop computer and searches all of his references. He taps into the air phone with his modem and searches the net and the Library of Congress. Frustrated, he sends e-mail to his co-workers all to no avail. After about an hour, he wakes the software engineer and hands him $100. The software engineer politely takes the $100 and turns away to try to get back to sleep. The programmer, more than a little frustrated, shakes the software engineer and asks "Well, so what's the answer?"

- Without a word, the software engineer reaches into his wallet, hands the programmer $10, and goes back to sleep.

Introduction to Decision Tables

- Decision tables work because they remedy one of requirements specifications biggest problem areas - textual requirements

- Decision tables can easily represent paragraphs worth of text in a single table and so are a powerful technique to reduce some of the software bug types discussed earlier.

- The types of systems that make good use of decision table representation are called "rule based" systems. This can apply to a large number of domains - business, e-commerce, web development, embedded, etc

- These have been used extensively in very complex domains and are worth addressing for their ease of specification

Decision Tables

- If you are a new customer and you want to open a credit card account then there are three condition.
  1. You will get a 15% discount on all your purchases today.
  2. If you are an existing customer and you hold a loyalty card, you get a 10% discount.
  3. If you have a coupon, you can get 20% off today (but it can’t be used with the ‘new customer’ discount). discount amounts are added, if applicable.
- Some interesting combinations…
  1. What happens if I am a new customer? Obvious!
  2. What happens if I am a new customer with a coupon?
  3. What happens if I am an existing customer with a coupon **and** a loyalty card?
  4. What happens if I am a new customer with a loyalty card and no coupon?
  5. What happens if I am a new customer with a loyalty card and a coupon?

Decision Tables (cont.)

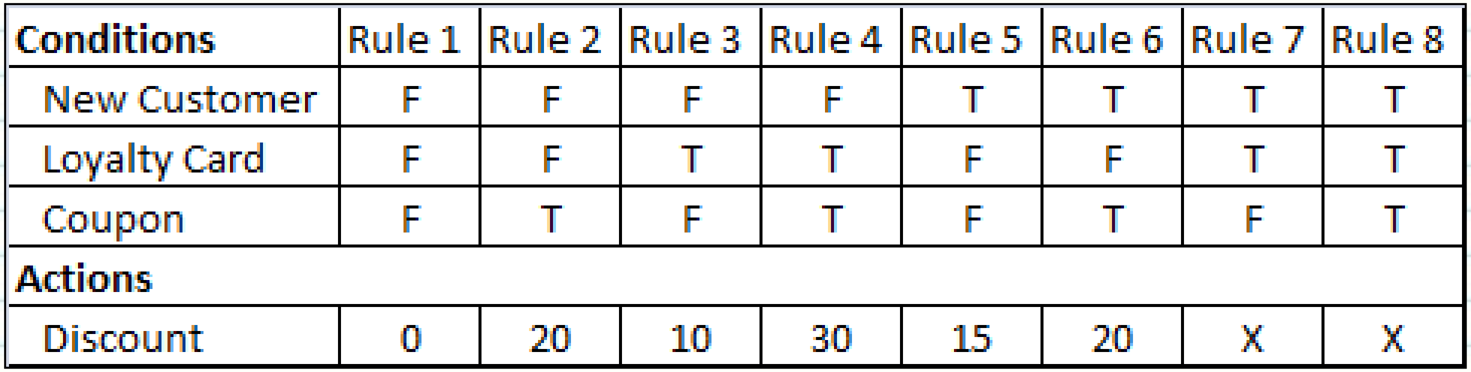
- I would be tempted to code the logic as follows:

```
if (new_customer) {
  if (!loyalty_card) {
    if (coupon)
      discount = 0.2;
    else discount = 0.15; }
  else error;}
else {
  if (loyalty_card)
    discount += 0.1;
  if (coupon)
 discount += 0.2;}
```

- Is this technically correct? Is it understandable?

Decision Tables (cont.)

- We can make this much more concise and understandable.
- How many conditions do we have? How many possible combinations does that give us?
- How many actions do we have?



Conditions	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
New Customer	F	F	F	F	T	T	T	T
Loyalty Card	F	F	T	T	F	F	T	T
Coupon	F	T	F	T	F	T	F	T
Actions								
Discount	0	20	10	30	15	20	X	X

- I have two cases where the requirements are not complete. I need to update the requirements to address what action I do with these.
- The best way to test this is to develop a test case for each column

Decision Tables (cont.)

- Loan Calculation
- The input has three fields and a Submit button, you can enter
  1. The amount of the total loan
  2. Interest rate
  3. The number of months you want to pay it off (term)
- Rules

  The monthly payment is calculated when the submit button is pressed if:

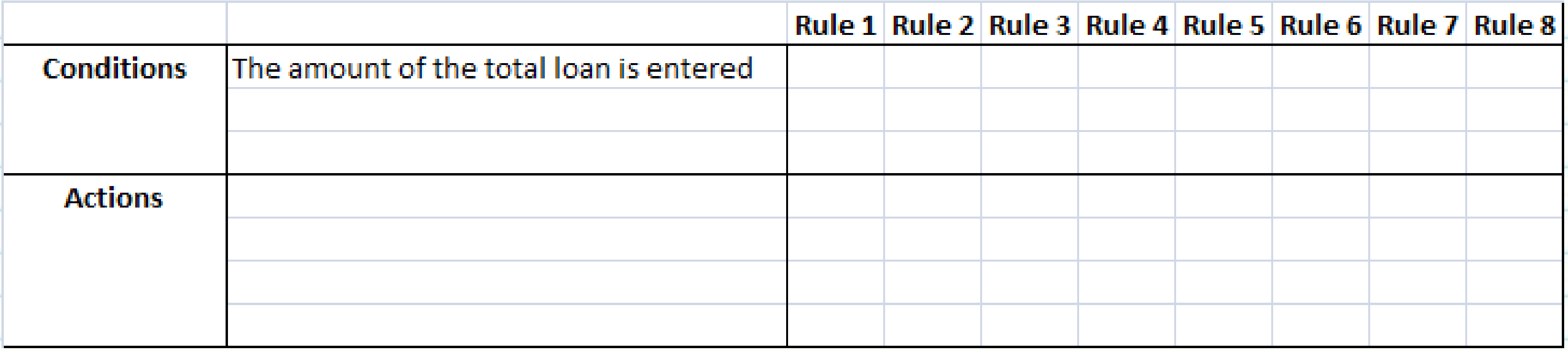
  1. All three are given
  2. Based on 5.5 % interest if the total loan and term is given
  3. Based on 48 months term if the total loan and interest are given

  An error condition is indicated:

  1. When the total loan amount has not been given

Class exercise – develop your own decision table – see the next chart for a blank decision table

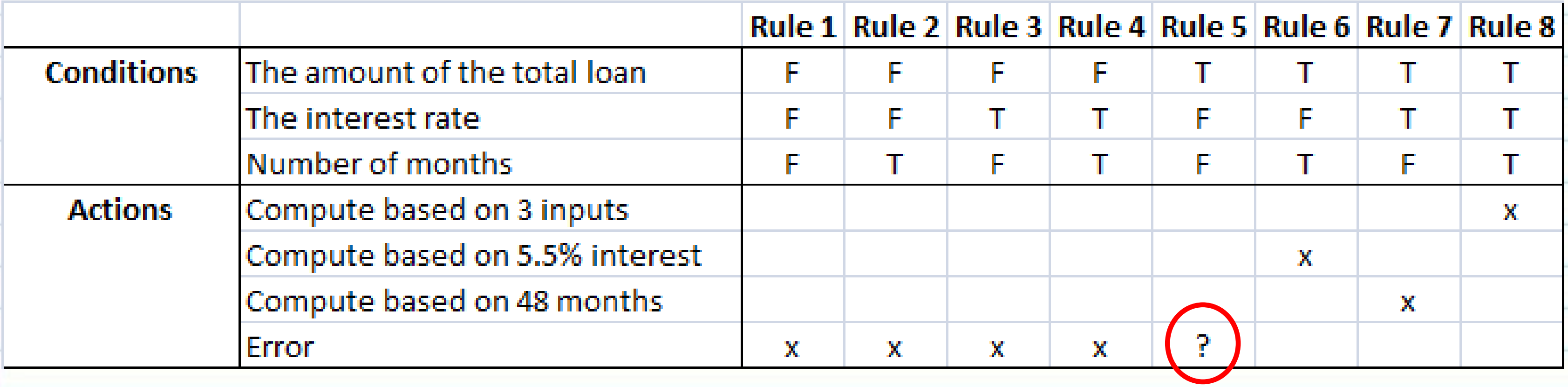
Decision Tables (cont.)



		Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
Conditions	The amount of the total loan is entered								
Actions									

Decision Tables (cont.)

- First I did not list the Submit button as an action – why?
- Second, the requirements are incomplete – a condition in the decision table has not been specified



		Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
Conditions	The amount of the total loan	F	F	F	F	T	T	T	T
	The interest rate	F	F	T	T	F	F	T	T
	Number of months	F	T	F	T	F	T	F	T
Actions	Compute based on 3 inputs								x
	Compute based on 5.5% interest						x		
	Compute based on 48 months							x	
	Error	x	x	x	x	?			

Decision Tables (cont.)

- Loan Calculation
- The input has four fields and a Submit button, you can enter
  1. The amount of the total loan
  2. Interest rate
  3. The number of months you want to pay it off (term)
- Rules

  The monthly payment is calculated when the submit button is pressed if:

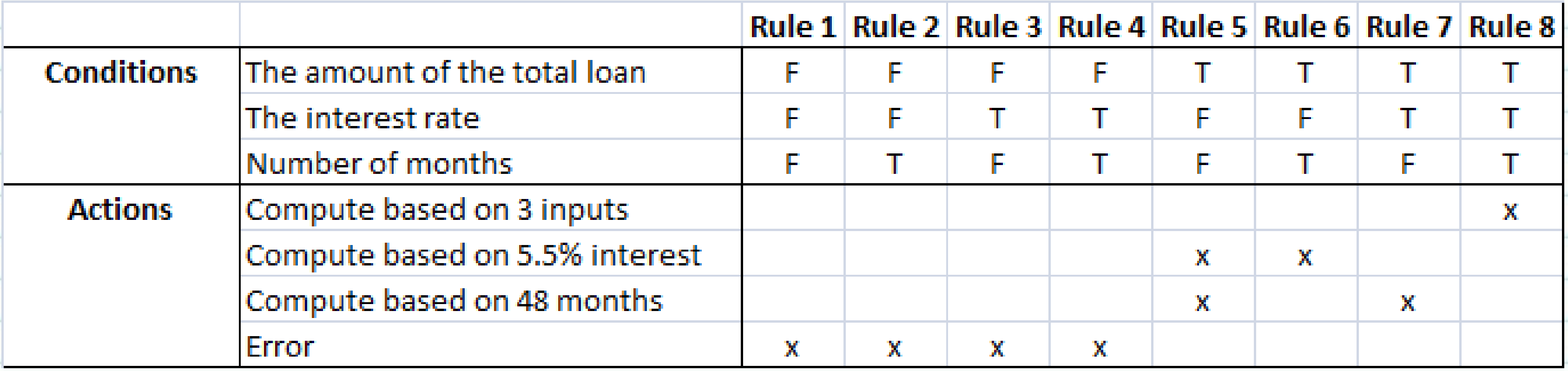
  1. All three are given
  2. The total loan and term is given then it is based on a 5.5 % interest
  3. The total loan and interest are given then it is based on a 48 months term
  4. Only the total loan amount then it is based off of 5.5% interest and 48 months term

  An error condition is indicated:

  1. When the total loan amount has not been given

Decision Tables (cont.)

- Now the requirements are complete – all conditions in the decision table have been specified



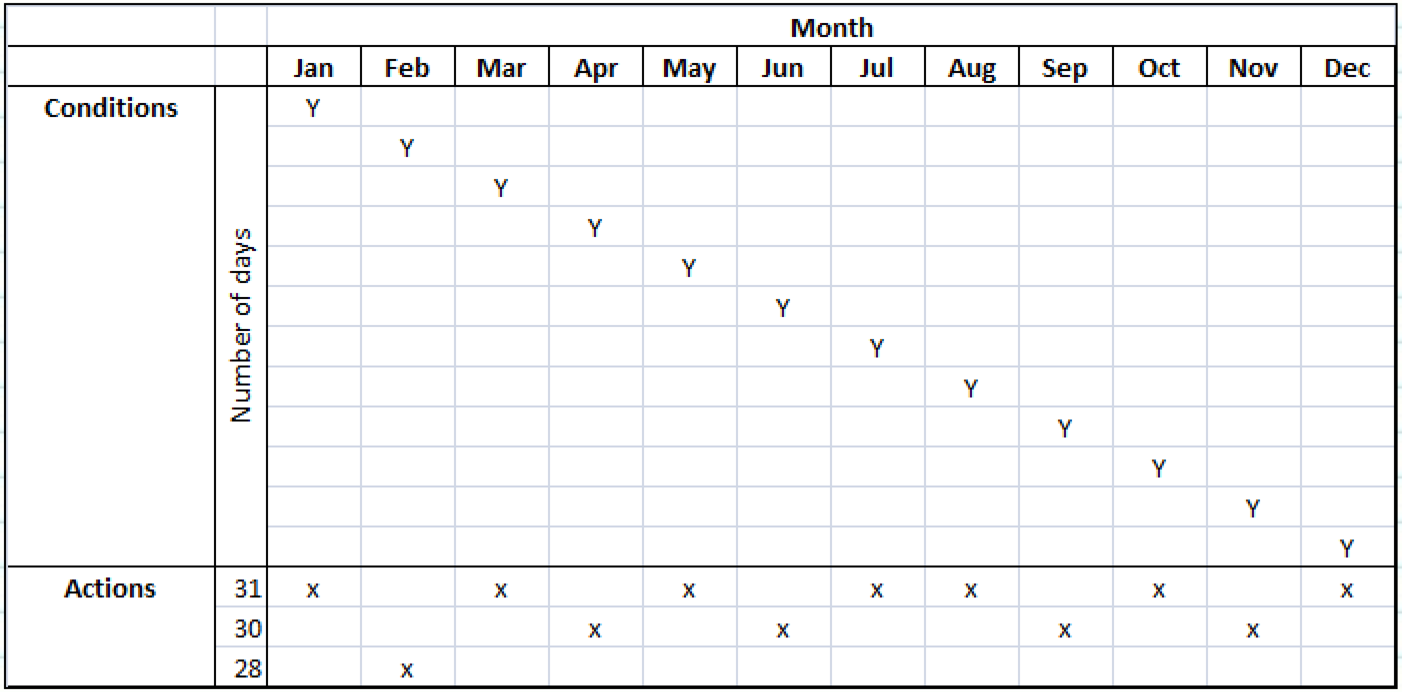
		Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
Conditions	The amount of the total loan	F	F	F	F	T	T	T	T
	The interest rate	F	F	T	T	F	F	T	T
	Number of months	F	T	F	T	F	T	F	T
Actions	Compute based on 3 inputs								x
	Compute based on 5.5% interest					x	x		
	Compute based on 48 months					x		x	
	Error	x	x	x	x				

Decision Tables (cont.)

- Incomplete and Over-specification of decision tables
- A decision table is complete if every possible set of conditions has a corresponding action prescribed (otherwise I have a case where an action is missing) – we have already looked at these
- An over-specified decision table results in a logic error – e.g., accidentally performing multiple actions where unintended

Decision Tables (cont.)

- Is this a correct representation of the requirements?



		Month											
		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Conditions	Number of days	Y											
			Y										
				Y									
					Y								
						Y							
							Y						
								Y					
									Y				
										Y			
											Y		
												Y	
													Y
Actions	31	x		x		x		x	x		x		x
	30				x		x			x		x	
	28		x										

- Balanced vs. unbalanced (latter typically has a “first of” or "all" clause)
- Can I make this table more concise without losing information?

Decision Tables (cont.)

- Class exercise - capture these rules in a decision table
- If the printer does **not** print, the warning light is flashing and the printer is **un**recognized, I need to:
  - check the printer-computer cable to make sure it is receiving data
  - ensure printer software is installed so that it can print
  - check or replace the ink
- If the printer does **not** print, the warning light is **not** flashing and the printer is recognized, I need to: check for a paper jam
- If the printer does **not** print, the warning light is flashing and the printer is recognized, I need to:
  - check or replace the ink
  - check for a paper jam
- If the printer does **not** print, the warning light is **not** flashing and the printer is **un**recognized, I need to:
  - Check the power cable
  - Check the printer-computer cable
  - Ensure printer software is installed
- If the printer does print, the warning light is flashing and the printer is recognized, I need to: Check/replace ink
- If the printer does print, the warning light is flashing and the printer is **un**recognized, I need to:
  - Ensure printer software is installed
  - Check/replace ink
- If the printer does print, the warning light is **not** flashing and the printer is **un**recognized, I need to: Ensure printer software is installed

Answer

		Rules							
Conditions	Printer does not print	N	N	N	N	Y	Y	Y	Y
	Warning light is flashing	N	N	Y	Y	N	N	Y	Y
	Printer is unrecognized	N	Y	N	Y	N	Y	N	Y
Actions	Check the power cable						X		
	Check the printer-computer cable						X		X
	Ensure printer software is installed		X		X		X		X
	Check/replace ink			X	X			X	X
	Check for paper jam					X		X	

What is the minimum number of test cases needed to test this?

Why?

Typical Decision Tables

One sided conditions	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
Conditions						
Sales Volume < $0	Y	N	N	N	N	N
$0.00 <= Sales Volume <= $49,999.99	N	Y	N	N	N	N
$50,000.00 <= Sales Volume <= $199,999.99	N	N	Y	N	N	N
$200,000.00 <= Sales Volume <= $300,000.00	N	N	N	Y	N	N
$300,000.01 <= Sales Volume <= $500,000.00	N	N	N	N	Y	N
Sales Volume > $500,00.00	N	N	N	N	N	Y
Actions						
Commission rate = 0%	X					
Commission rate = 2.1%		X				
Commission rate = 2.5%			X			
Commission rate = 3.05%				X		
Commission rate = 3.75%					X	
Commission rate = 5%						X
Fee ($300)	X					
No fee		X	X	X	X	X
Status = No sales	X					
Status = Ordinary sales		X	X	X	X	
Status = Top Seller						X
Gift Card = Yes						X
Gift Card = No	X	X	X	X	X	

Typical Decision Tables (cont.)

How Many ECPs?	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
Conditions						
1 Sales Volume < $0	Y	N	N	N	N	N
2 $0.00 <= Sales Volume <= $49,999.99	N	Y	N	N	N	N
3 $50,000.00 <= Sales Volume <= $199,999.99	N	N	Y	N	N	N
4 $200,000.00 <= Sales Volume <= $300,000.00	N	N	N	Y	N	N
5 $300,000.01 <= Sales Volume <= $500,000.00	N	N	N	N	Y	N
6 Sales Volume > $500,00.00	N	N	N	N	N	Y
Actions						
Commission rate = 0%	X					
Commission rate = 2.1%		X				
					X	
						X
					X	X
Status = Ordinary sales		X	X	X	X	
Status = Top Seller						X
Gift Card = Yes						X
Gift Card = No	X	X	X	X	X	

How Many Test Cases? Take the maximum of:

1) BVs - 2 tests * 6 ECPs = 12 tests

2) Number of Rules = 6

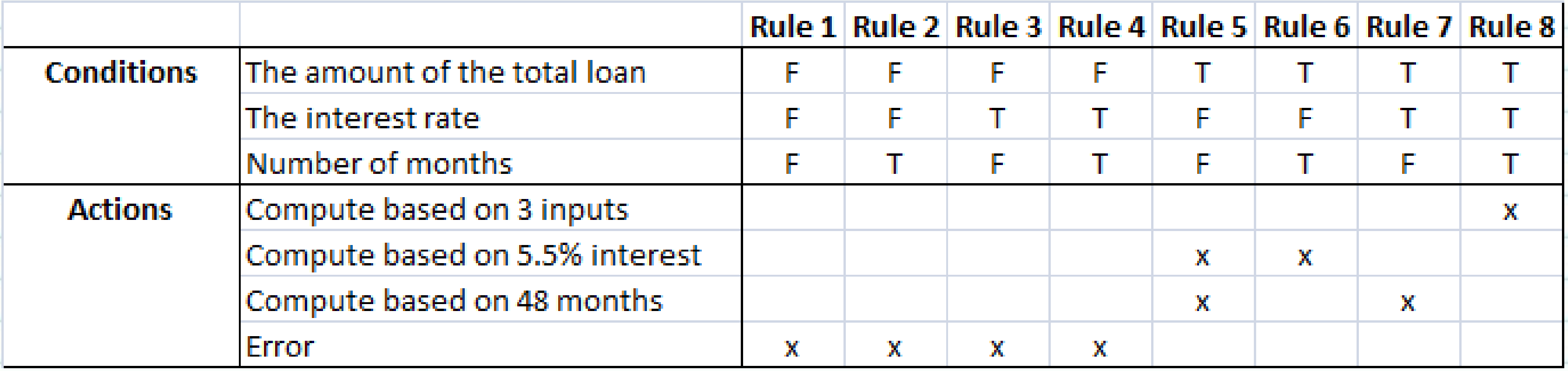
Why? 6 test cases would leave a minimum of 6 BVs untested

(c) JRCS 2016

Typical Decision Tables (cont.)

How Many unique ECPs?	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
Conditions						
Sales Volume < $0	Y	N	N	N	N	N
$0.00 <= Sales Volume <= $49,999.99	N	Y	N	N	N	N
$50,000.00 <= Sales Volume <= $199,999.99	N	N	Y	N	N	N
$200,000.00 <= Sales Volume <= $300,000.00	N	N	N	Y	N	N
$300,000.01 <= Sales Volume <= $500,000.00	N	N	N	N	Y	N
Sales Volume > $500,00.00	N	N	N	N	N	Y
Actions						
Commission rate = 0%	X					
Commission rate = 2.1%		X				
Commission rate = 2.5%			X			
Commission rate = 3.05%	1			X		
Commission rate = 3.75%		2	3		X	
Commission rate = 5%				4		X
Fee ($300)	X				5	6
No fee		X	X	X	X	X
Status = No sales	X					
Status = Ordinary sales		X	X	X	X	
Status = Top Seller						X
Gift Card = Yes						X
Gift Card = No	X	X	X	X	X	

More Tables



		Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
Conditions	The amount of the total loan	F	F	F	F	T	T	T	T
	The interest rate	F	F	T	T	F	F	T	T
	Number of months	F	T	F	T	F	T	F	T
Actions	Compute based on 3 inputs								x
	Compute based on 5.5% interest					x	x		
	Compute based on 48 months					x		x	
	Error	x	x	x	x				

What is the minimum number of test cases needed to test this?

8 test cases

How many unique ECPs? 5

(c) JRCS 2016

Unique ECPs vs. ECPs

- When the number between these two is different, this means that we have common actions across different ECPs
  - Much of the industry tests only the unique ECPs
  - This is not a complete coverage of the table - we will test the maximum of the number of BVs and the number of rules

- Having common ECPs does let us re-use expected outputs (being careful of copy and paste errors)

- It also let's us mentally reduce the problem scope further

Proper Decision Tables

- A completely specified decision table needs to show all possible conditions

All combinations are not specified

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
Conditions *						
Sales Volume < $0	Y	N	N	N	N	N
$0.00 <= Sales Volume <= $49,999.99	N	Y	N	N	N	N
$50,000.00 <= Sales Volume <= $199,999.99	N	N	Y	N	N	N
$200,000.00 <= Sales Volume <= $300,000.00	N	N	N	Y	N	N
$300,000.01 <= Sales Volume <= $500,000.00	N	N	N	N	Y	N
Sales Volume > $500,00.00	N	N	N	N	N	Y
Actions						
Commission rate	0%	2.10%	2.50%	3.05%	3.75%	5%
Fee	$300	$0	$0	$0	$0	$0
Status=None (NS), Ordinary (OS), Top (TS)	NS	OS	OS	OS	OS	TS
Gift Card	N	N	N	N	N	Y

* All other rules illegal

All other combinations must be addressed either this way or with a rule

Making Tables Concise

The previous table can be specified as the following which is much more concise and succinct - leading to fewer errors

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
Conditions	1) All conditions two-sided (where possible)					
-$∞ <= Sales Volume <= -$0.01	Y					
$0.00 <= Sales Volume <= $49,999.99		Y				
$50,000.00 <= Sales Volume <= $199,999.99			Y			
$200,000.00 <= Sales Volume <= $300,000.00				Y		
$300,000.01 <= Sales Volume <= $500,000.00					Y	
$500,000.01 <= Sales Volume <= $∞						Y
Actions	2) Write actions concisely					
Commission rate	0%	2.10%	2.50%	3.05%	3.75%	5%
Fee	$300	$0	$0	$0	$0	$0
Status=None (NS), Ordinary (OS), Top (TS)	NS	OS	OS	OS	OS	TS
Gift Card	N	N	N	N	N	Y

Table implements first-of rule	3) Use an unbalanced table

Common Problems With Decision Tables

- Faulty specification of conditions

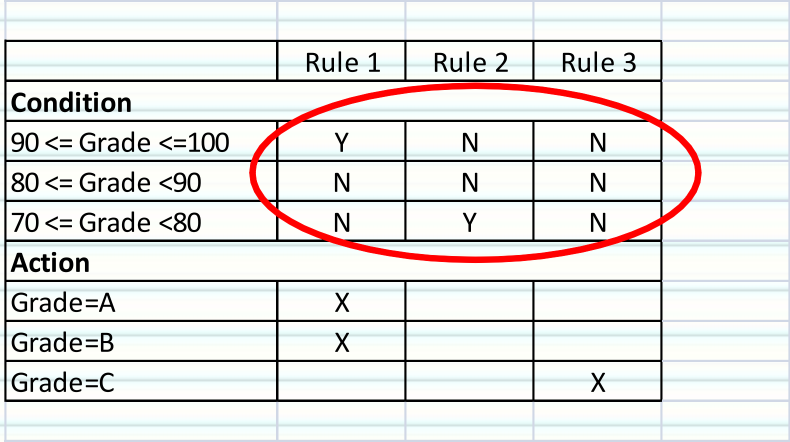
	Rule 1	Rule 2	Rule 3
Condition			
90 <= Grade <=100	Y	N	N
80 <= Grade <90	Y	N	N
70 <= Grade <80	N	Y	N
Action			
Grade=A	X		
Grade=B		X	
Grade=C			X

- Faulty specification of actions (over-specification)

	Rule 1	Rule 2	Rule 3
Condition			
90 <= Grade <=100	Y	N	N
80 <= Grade <90	N	N	N
70 <= Grade <80	N	Y	N
Action			
Grade=A	X		
Grade=B	X		
Grade=C			X

Common Problems With Decision Tables (cont.)

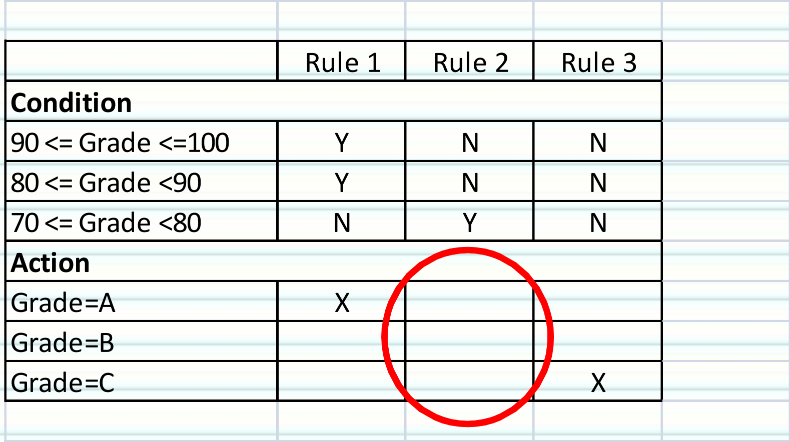
- Incomplete specification of conditions



	Rule 1	Rule 2	Rule 3
Condition			
90 <= Grade <=100	Y	N	N
80 <= Grade <90	N	N	N
70 <= Grade <80	N	Y	N
Action			
Grade=A	X		
Grade=B	X		
Grade=C			X

Needs to map the full truth table on conditions

- Incomplete specification of actions

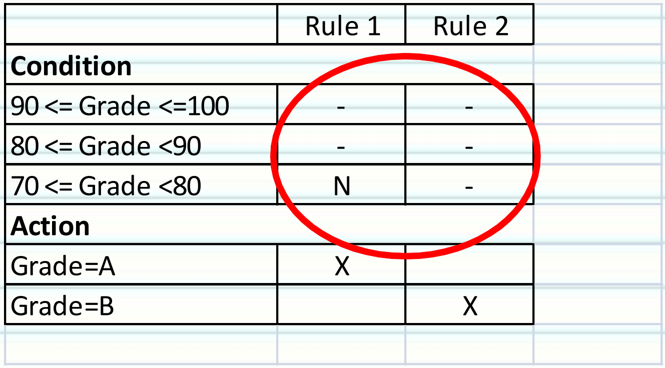


	Rule 1	Rule 2	Rule 3
Condition			
90 <= Grade <=100	Y	N	N
80 <= Grade <90	Y	N	N
70 <= Grade <80	N	Y	N
Action			
Grade=A	X		
Grade=B			
Grade=C			X

Needs to map the full truth table on actions

Common Problems With Decision Tables (cont.)

- Over specification of conditions

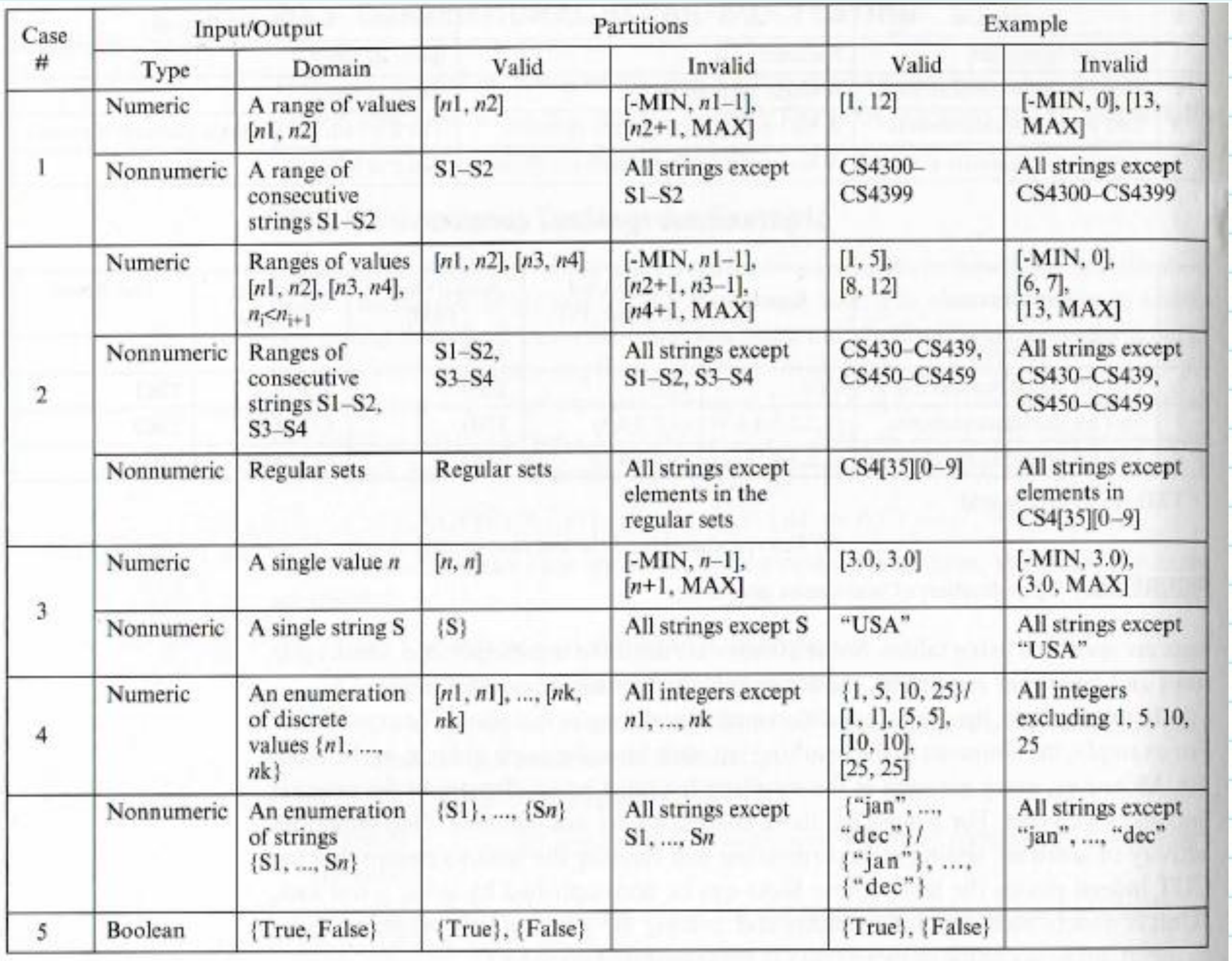


	Rule 1	Rule 2
Condition		
90 <= Grade <=100	-	-
80 <= Grade <90	-	-
70 <= Grade <80	N	-
Action		
Grade=A	X	
Grade=B		X

Rules sets overlap across the condition space

(c) JRCS 2016

The Full Rule Set



Case #	Input/Output		Partitions		Example	
	Type	Domain	Valid	Invalid	Valid	Invalid
1	Numeric	A range of values [n1, n2]	[n1, n2]	[-MIN, n1–1], [n2+1, MAX]	[1, 12]	[-MIN, 0], [13, MAX]
	Nonnumeric	A range of consecutive strings S1–S2	S1–S2	All strings except S1–S2	CS4300–CS4399	All strings except CS4300–CS4399
2	Numeric	Ranges of values [n1, n2], [n3, n4], $n_i < n_{i+1}$	[n1, n2], [n3, n4]	[-MIN, n1–1], [n2+1, n3–1], [n4+1, MAX]	[1, 5], [8, 12]	[-MIN, 0], [6, 7], [13, MAX]
	Nonnumeric	Ranges of consecutive strings S1–S2, S3–S4	S1–S2, S3–S4	All strings except S1–S2, S3–S4	CS430–CS439, CS450–CS459	All strings except CS430–CS439, CS450–CS459
	Nonnumeric	Regular sets	Regular sets	All strings except elements in the regular sets	CS4[35][0–9]	All strings except elements in CS4[35][0–9]
3	Numeric	A single value n	[n, n]	[-MIN, n–1], [n+1, MAX]	[3.0, 3.0]	[-MIN, 3.0), (3.0, MAX]
	Nonnumeric	A single string S	{S}	All strings except S	"USA"	All strings except "USA"
4	Numeric	An enumeration of discrete values {n1, ..., nk}	[n1, n1], ..., [nk, nk]	All integers except n1, ..., nk	{1, 5, 10, 25}/ [1, 1], [5, 5], [10, 10], [25, 25]	All integers excluding 1, 5, 10, 25
	Nonnumeric	An enumeration of strings {S1, ..., Sn}	{S1}, ..., {Sn}	All strings except S1, ..., Sn	{"jan", ..., "dec"}/ {"jan"}, ..., {"dec"}	All strings except "jan", ..., "dec"
5	Boolean	{True, False}	{True}, {False}		{True}, {False}	

FIGURE 20.2 Rules for partitioning the input space