Chapter 7

Decision Table–Based Testing

Of all the functional testing methods, those based on decision tables are the most rigorous because decision tables enforce logical rigor. Two closely related methods are used: cause–effect graphing (Elmendorf, 1973; Myers, 1979) and the decision tableau method (Mosley, 1993). These are more cumbersome to use and are fully redundant with decision tables, so we will not discuss them here. Both are covered in Mosley (1993).

7.1 Decision Tables

Decision tables have been used to represent and analyze complex logical relationships since the early 1960s. They are ideal for describing situations in which a number of combinations of actions are taken under varying sets of conditions. Some of the basic decision table terms are illustrated in Table 7.1.

A decision table has four portions: the left-most column is the stub portion; to the right is the entry portion. The the condition portion is noted by c's, the action portion is noted by a's. Thus, we can refer to the condition stub, the condition entries, the action stub, and the action entries. A column in the entry portion is a rule. Rules indicate which actions, if any, are taken for the circumstances indicated in the condition portion of the rule. In the decision table in Table 7.1, when conditions c1, c2, and c3 are all true, actions a1 and a2 occur. When c1 and c2 are both true and c3 is false, actions a1 and a3 occur. The entry for c3 in the rule where c1 is true and c2 is false is called a "don't care" entry. The don't care entry has two major interpretations: the condition is irrelevant, or the condition does not apply. Sometimes people will enter the "n/a" symbol for this latter interpretation.

When we have binary conditions (true/false, yes/no, 0/1), the condition portion of a decision table is a truth table (from propositional logic) that has been rotated 90°. This structure guarantees that we consider every possible combination of condition values. When we use decision tables for test case identification, this completeness property of a decision table guarantees a form of complete testing. Decision tables in which all the conditions are binary are called limited entry decision tables. If conditions are allowed to have several values, the resulting tables are called extended entry decision tables. We will see examples of both types for the NextDate problem.

Table 7.1 Portions of a Decision Table

Stub	*Rule 1*	*Rule 2*	*Rules 3, 4*	*Rule 5*	*Rule 6*	*Rules 7, 8*
c1	T	T	T	F	F	F
c2	T	T	F	T	T	F
c3	T	F	—	T	F	—
a1	X	X		X		
a2	X				X	
a3		X		X		
a4			X			X

Decision tables are deliberately declarative (as opposed to imperative); no particular order is implied by the conditions, and selected actions do not occur in any particular order.

7.1.1 Technique

To identify test cases with decision tables, we interpret conditions as inputs and actions as outputs. Sometimes conditions end up referring to equivalence classes of inputs, and actions refer to major functional processing portions of the item tested. The rules are then interpreted as test cases. Because the decision table can mechanically be forced to be complete, we know we have a comprehensive set of test cases.

Several techniques that produce decision tables are more useful to testers. One helpful style is to add an action to show when a rule is logically impossible.

In the decision table in Table 7.2, we see examples of don't care entries and impossible rule usage. If the integers a, b, and c do not constitute a triangle, we do not even care about possible equalities, as indicated in the first rule. In rules 3, 4, and 6, if two pairs of integers are equal, by transitivity, the third pair must be equal; thus, the negative entry makes these rules impossible.

Table 7.2 Decision Table for the Triangle Problem

c1: a, b, c form a triangle?	F	T	T	T	T	T	T	T	T
c2: a = b?	—	T	T	T	T	F	F	F	F
c3: a = c?	—	T	T	F	F	T	T	F	F
c4: b = c?	—	T	F	T	F	T	F	T	F
a1: Not a Triangle	X								
a2: Scalene									X
a3: Isosceles					X		X	X	
a4: Equilateral		X							
a5: Impossible			X	X		X			

Table 7.3 Refined Decision Table for the Triangle Problem

c1: a < b + c?	F	T	T	T	T	T	T	T	T	T	T
c2: b < a + c?	—	F	T	T	T	T	T	T	T	T	T
c3: c < a + b?	—	—	F	T	T	T	T	T	T	T	T
c4: a = b?	—	—	—	T	T	T	T	F	F	F	F
c5: a = c?	—	—	—	T	T	F	F	T	T	F	F
c6: b = c?	—	—	—	T	F	T	F	T	F	T	F
a1: Not a Triangle	X	X	X								
a2: Scalene											X
a3: Isosceles							X		X	X	
a4: Equilateral				X							
a5: Impossible					X	X		X			

The decision table in Table 7.3 illustrates another consideration related to technique: the choice of conditions can greatly expand the size of a decision table. Here, we expanded the old condition (c1: a, b, c form a triangle?) to a more detailed view of the three inequalities of the triangle property. If any one of these fails, the three integers do not constitute sides of a triangle. We could expand this still further because there are two ways an inequality could fail: one side could equal the sum of the other two, or it could be strictly greater.

When conditions refer to equivalence classes, decision tables have a characteristic appearance. Conditions in the decision table in Table 7.4 are from the NextDate problem; they refer to the mutually exclusive possibilities for the month variable. Because a month is in exactly one equivalence class, we cannot ever have a rule in which two entries are true. The don't care entries (—) really mean "must be false." Some decision table aficionados use the notation F! to make this point.

Use of don't care entries has a subtle effect on the way in which complete decision tables are recognized. For limited entry decision tables, if n conditions exist, there must be 2^n rules. When don't care entries really indicate that the condition is irrelevant, we can develop a rule count as follows: rules in which no don't care entries occur count as one rule, and each don't care entry in

Table 7.4 Decision Table with Mutually Exclusive Conditions

Conditions	*R1*	*R2*	*R3*
c1: month in M1?	T	—	—
c2: month in M2?	—	T	—
c3: month in M3?	—	—	T
a1			
a2			
a3			

Table 7.5 Decision Table for Table 7.3 with Rule Counts

c1: a < b + c?	F	T	T	T	T	T	T	T	T	T	T
c2: b < a + c?	—	F	T	T	T	T	T	T	T	T	T
c3: c < a + b?	—	—	F	T	T	T	T	T	T	T	T
c4: a = b?	—	—	—	T	T	T	T	F	F	F	F
c5: a = c?	—	—	—	T	T	F	F	T	T	F	F
c6: b = c?	—	—	—	T	F	T	F	T	F	T	F
Rule count	32	16	8	1	1	1	1	1	1	1	1
a1: Not a Triangle	X	X	X								
a2: Scalene											X
a3: Isosceles							X		X	X	
a4: Equilateral				X							
a5: Impossible					X	X		X			

a rule doubles the count of that rule. The rule counts for the decision table in Table 7.3 are shown in Table 7.5. Notice that the sum of the rule counts is 64 (as it should be).

If we apply this simplistic algorithm to the decision table in Table 7.4, we get the rule counts shown in Table 7.6.

Table 7.6 Rule Counts for a Decision Table with Mutually Exclusive Conditions

Conditions	*R1*	*R2*	*R3*
c1: month in M1	T	—	—
c2: month in M2	--	T	—
c3: month in M3	—	—	T
Rule count	4	4	4
a1			

We should only have eight rules, so we clearly have a problem. To see where the problem lies, we expand each of the three rules, replacing the "—" entries with the T and F possibilities, as shown in Table 7.7.

Notice that we have three rules in which all entries are T: rules 1.1, 2.1, and 3.1. We also have two rules with T, T, F entries: rules 1.2 and 2.2. Similarly, rules 1.3 and 3.2 are identical; so are rules 2.3 and 3.3. If we delete the repetitions, we end up with seven rules; the missing rule is the one in which all conditions are false. The result of this process is shown in Table 7.8. The impossible rules are also shown.

The ability to recognize (and develop) complete decision tables puts us in a powerful position with respect to redundancy and inconsistency. The decision table in Table 7.9 is redundant — three conditions and nine rules exist. (Rule 9 is identical to rule 4.)

Notice that the action entries in rule 9 are identical to those in rules 1 to 4. As long as the actions in a redundant rule are identical to the corresponding part of the decision table, we do

Table 7.7 Expanded Version of Table 7.6

Conditions	*1.1*	*1.2*	*1.3*	*1.4*	*2.1*	*2.2*	*2.3*	*2.4*	*3.1*	*3.2*	*3.3*	*3.4*
c1: mo. in M1	T	T	T	T	T	T	F	F	T	T	F	F
c2: mo. in M2	T	T	F	F	T	T	T	T	T	F	T	F
c3: mo. in M3	T	F	T	F	T	F	T	F	T	T	T	T
Rule count	1	1	1	1	1	1	1	1	1	1	1	1
a1												

Table 7.8 Mutually Exclusive Conditions with Impossible Rules

	1.1	*1.2*	*1.3*	*1.4*	*2.3*	*2.4*	*3.4*	
c1: mo. in M1	T	T	T	T	F	F	F	F
c2: mo. in M2	T	T	F	F	T	T	F	F
c3: mo. in M3	T	F	T	F	T	F	T	F
Rule count	1	1	1	1	1	1	1	1
a1: Impossible	X	X	X		X			X

Table 7.9 A Redundant Decision Table

Conditions	*1–4*	*5*	*6*	*7*	*8*	*9*
c1	T	F	F	F	F	T
c2	—	T	T	F	F	F
c3	—	T	F	T	F	F
a1	X	X	X	—	—	X
a2	—	X	X	X	—	—
a3	X	—	X	X	X	X
	✓					✓

redundancy

Table 7.10 An Inconsistent Decision Table

Conditions	*1–4*	*5*	*6*	*7*	*8*	*9*
c1	T	F	F	F	F	T
c2	—	T	T	F	F	F
c3	—	T	F	T	F	F
a1	X	X	X	—	—	—
a2	—	X	X	X	—	X
a3	X	—	X	X	X	—
	✓					✓

"inconsistent" means same Conds diff action

not have much of a problem. If the action entries are different, as they are in Table 7.10, we have a bigger problem.

If the decision table in Table 7.10 were to process a transaction in which c1 is true and both c2 and c3 are false, both rules 4 and 9 apply. We can make two observations:

1. Rules 4 and 9 are inconsistent.
2. The decision table is nondeterministic.

Rules 4 and 9 are inconsistent because the action sets are different. The whole table is nondeterministic because there is no way to decide whether to apply rule 4 or rule 9. The bottom line for testers is that care should be taken when don't care entries are used in a decision table.

7.2 Test Cases for the Triangle Problem

Using the decision table in Table 7.3, we obtain 11 functional test cases: 3 impossible cases, 3 ways to fail the triangle property, 1 way to get an equilateral triangle, 1 way to get a scalene triangle, and 3 ways to get an isosceles triangle (see Table 7.11). Of course, we still need to provide actual values

Table 7.11 Test Cases from Table 7.3

Case ID	*a*	*b*	*c*	*Expected Output*
DT1	4	1	2	Not a Triangle
DT2	1	4	2	Not a Triangle
DT3	1	2	4	Not a Triangle
DT4	5	5	5	Equilateral
✓ DT5	?	?	?	Impossible
✓ DT6	?	?	?	Impossible
DT7	2	2	3	Isosceles
✓ DT8	?	?	?	Impossible
DT9	2	3	2	Isosceles
DT10	3	2	2	Isosceles
DT11	3	4	5	Scalene

"?" coming from impossible action

for the variables in the conditions. If we extended the decision table to show both ways to fail an inequality, we would pick up three more test cases (where one side is exactly the sum of the other two). Some judgment is required in this because of the exponential growth of rules. In this case, we would end up with many more don't care entries and more impossible rules.

7.3 Test Cases for the NextDate Function

The NextDate function was chosen because it illustrates the problem of dependencies in the input domain. This makes it a perfect example for decision table-based testing, because decision tables can highlight such dependencies. Recall that, in Chapter 6, we identified equivalence classes in the input domain of the NextDate function. One of the limitations we found in Chapter 6 was that indiscriminate selection of input values from the equivalence classes resulted in "strange" test cases, such as finding the next date to June 31, 1812. The problem stems from the presumption that the variables are independent. If they are, a Cartesian product of the classes makes sense. When logical dependencies exist among variables in the input domain, these dependencies are lost (suppressed is better) in a Cartesian product. The decision table format lets us emphasize such dependencies using the notion of the "impossible" action to denote impossible combinations of conditions (which are actually impossible rules). In this section, we will make three tries at a decision table formulation of the NextDate function.

7.3.1 First Try

Identifying appropriate conditions and actions presents an opportunity for craftsmanship. Suppose we start with a set of equivalence classes close to the one we used in Chapter 6:

M1 = {month : month has 30 days}
M2 = {month : month has 31 days}
M3 = {month : month is February}
D1 = {day : $1 \leq$ day ≤ 28}
D2 = {day : day = 29}
D3 = {day : day = 30}
D4 = {day : day = 31}
Y1 = {year : year is a leap year}
Y2 = {year : year is not a leap year}

If we wish to highlight impossible combinations, we could make a limited entry decision table with the following conditions and actions. (Note that the equivalence classes for the year variable collapse into one condition in Table 7.12.)

This decision table will have 256 rules, many of which will be impossible. If we wanted to show why these rules were impossible, we might revise our actions to the following:

a1: Day invalid for this month
a2: Cannot happen in a non-leap year
a3: Compute the next date

Table 7.12 First Try Decision Table with 256 Rules

Conditions			
c1: month in M1?	T		
c2: month in M2?		T	
c3: month in M3?			T
c4: day in D1?			
c5: day in D2?			
c6: day in D3?			
c7: day in D4?			
c8: year in Y1?			
a1: impossible			
a2: next date			

a1 & a2 can not happen at the same time

7.3.2 Second Try

If we focus on the leap year aspect of the NextDate function, we could use the set of equivalence classes as they were in Chapter 6. These classes have a Cartesian product that contains 36 entries (test cases), with several that are impossible.

To illustrate another decision table technique, this time we will develop an extended entry decision table, and we will take a closer look at the action stub. In making an extended entry decision table, we must ensure that the equivalence classes form a true partition of the input domain. (Recall from Chapter 3 that a partition is a set of disjoint subsets where the union is the entire set.) If there were any overlaps among the rule entries, we would have a redundant case in which more than one rule could be satisfied. Here, Y2 is the set of years between 1812 and 2012, evenly divisible by four, excluding the year 2000:

M1 = {month : month has 30 days}
M2 = {month : month has 31 days}
M3 = {month : month is February}
D1 = {day : 1 ≤ day ≤ 28}
D2 = {day : day = 29}
D3 = {day : day = 30}
D4 = {day : day = 31}
Y1 = {year : year = 2000}
Y2 = {year : year is a non-century leap year}
Y3 = {year : year is a common year}

In a sense, we could argue that we have a "gray box" technique, because we take a closer look at the NextDate problem statement. To produce the next date of a given date, only five possible manipulations can be used: incrementing and resetting the day and month, and incrementing the year. (We will not let time go backward by resetting the year.) To follow the metaphor, we still cannot see inside the implementation box—the implementation could be a table lookup.

These conditions would result in a decision table with 36 rules that correspond to the Cartesian product of the equivalence classes. Combining rules with don't care entries yields the decision

Extended entry as c3 can be Y1, Y2, Y3

Table 7.13 Second Try Decision Table with 36 Rules

	1	2	3	4	5	6	7	8
c1: month in	M1	M1	M1	M1	M2	M2	M2	M2
c2: day in	D1	D2	D3	D4	D1	D2	D3	D4
c3: year in	—	—	—	—	—	—	—	—
Rule count	3	3	3	3	3	3	3	3
Actions								
a1: impossible				X				
a2: increment day	X	X			X	X	X	
a3: reset day			X					X
a4: increment month			X					?
a5: reset month								?
a6: increment year								?

don't know if Dec.

	9	10	11	12	13	14	15	16
c1: month in	M3	M3	M3	M3	M3	M3	M3	M3
c2: day in	D1	D1	D1	D2	D2	D2	D3	D4
c3: year in	Y1	Y2	Y3	Y1	Y2	Y3	—	—
Rule count	1	1	1	1	1	1	3	3
Actions								
a1: impossible						X	X	X
a2: increment day		X						
a3: reset day	X		X	X	X			
a4: increment month	X		X	X	X			
a5: reset month								
a6: increment year								

table in Table 7.13, which has 16 rules. We still have the problem with logically impossible rules, but this formulation helps us identify the expected outputs of a test case. If you complete the action entries in this table, you will find some cumbersome problems with December (in rule 8) and other problems with February 28 in rules 9, 11, and 12. We fix these next.

7.3.3 Third Try

We can clear up the end-of-year considerations with a third set of equivalence classes. This time, we are very specific about days and months, and we revert to the simpler leap year or non-leap year condition of the first try—so the year 2000 gets no special attention. (We could do a fourth try, showing year equivalence classes as in the second try, but by now you get the point.)

M1 = {month : month has 30 days}
M2 = {month : month has 31 days except December}
M3 = {month : month is December}
M4 = {month : month is February}
D1 = {day : $1 \le$ day ≤ 27}
D2 = {day : day = 28}
D3 = {day : day = 29}
D4 = {day : day = 30}
D5 = {day : day = 31}
Y1 = {year : year is a leap year}
Y2 = {year : year is a common year}

The Cartesian product of these contains 40 elements. The result of combining rules with don't care entries is given in Table 7.14; it has 22 rules, compared with the 36 of the second try. Recall from Chapter 1 the question of whether a large set of test cases is necessarily better than a smaller

not dec 31 days

Table 7.14 Decision Table for the NextDate Function

30 days	1	2	3	4	5	6	7	8	9	10
c1: month in	M1	M1	M1	M1	M1	M2	M2	M2	M2	M2
c2: day in	D1	D2	D3	D4	D5	D1	D2	D3	D4	D5
c3: year in	—	—	—	—	—	—	—	—	—	—
Actions										
a1: impossible					X					
a2: increment day	X	X	X			X	X	X	X	
a3: reset day				X						X
a4: increment month				X						X
a5: reset month										
a6: increment year										

Feb

Dec	11	12	13	14	15	16	17	18	19	20	21	22
c1: month in	M3	M3	M3	M3	M3	M4	M4	M4	M4	M4	M4	M4
c2: day in	D1	D2	D3	D4	D5	D1	D2	D2	D3	D3	D4	D5
c3: year in	—	—	—	—	—	—	Y1	Y2	Y1	Y2	—	—
Actions												
a1: impossible										X	X	X
a2: increment day	X	X	X	X		X	X					
a3: reset day					X			X	X			
a4: increment month								X	X			
a5: reset month					X							
a6: increment year					X							

diff from previous try

set. Here, we have a 22-rule decision table that gives a clearer picture of the NextDate function than does the 36-rule decision table. The first five rules deal with 30-day months; notice that the leap year considerations are irrelevant. The next two sets of rules (6 to 15) deal with 31-day months, where rules 6 to 10 deal with months other than December and rules 11 to 15 deal with December. No impossible rules are listed in this portion of the decision table, although there is some redundancy that an efficient tester might question. Eight of the 10 rules simply increment the day. Would we really require eight separate test cases for this subfunction? Probably not; but note the insights we can get from the decision table. Finally, the last seven rules focus on February in common and leap years.

The decision table in Table 7.14 is the basis for the source code for the NextDate function in Chapter 2. As an aside, this example shows how good testing can improve programming. All the decision table analysis could have been done during the detailed design of the NextDate function.

Table 7.15 Reduced Decision Table for the NextDate Function

	1–3	*4*	*5*	*6–9*	*10*
c1: month in	M1	M1	M1	M2	M2
c2: day in	D1, D2, D3	D4	D5	D1, D2, D3, D4	D5
c3: year in	—	—	—	—	—
Actions					
a1: impossible			X		
a2: increment day	X			X	
a3: reset day		X			X
a4: increment month		X			X
a5: reset month					
a6: increment year					

	11–14	*15*	*16*	*17*	*18*	*19*	*20*	*21, 22*
c1: month in	M3	M3	M4	M4	M4	M4	M4	M4
c2: day in	D1, D2, D3, D4	D5	D1	D2	D2	D3	D3	D4, D5
c3: year in	—	—	—	Y1	Y2	Y1	Y2	—
Actions								
a1: impossible							X	X
a2: increment day	X		X	X				
a3: reset day		X			X	X		
a4: increment month					X	X		
a5: reset month		X						
a6: increment year		X						

We can use the algebra of decision tables to further simplify these 22 test cases. If the action sets of two rules in a limited entry decision table are identical, there must be at least one condition that allows two rules to be combined with a don't care entry. This is the decision table equivalent of the "treated the same" guideline that we used to identify equivalence classes. In a sense, we are identifying equivalence classes of rules. For example, rules 1, 2, and 3 involve day classes D1, D2, and D3 for 30-day months. These can be combined similarly for day classes D1, D2, D3, and D4 in the 31-day month rules, and D4 and D5 for February. The result is in Table 7.15.

The corresponding test cases are shown in Table 7.16.

Table 7.16 Decision Table Test Cases for NextDate

Case ID	*Month*	*Day*	*Year*	*Expected Output*
1–3	April	15	2001	April 16, 2001
4	April	30	2001	May 1, 2001
5	April	31	2001	Invalid Input Date
6–9	January	15	2001	January 16, 2001
10	January	31	2001	February 1, 2001
11–14	December	15	2001	December 16, 2001
15	December	31	2001	January 1, 2002
16	February	15	2001	February 16, 2001
17	February	28	2004	February 29, 2004
18	February	28	2001	March 1, 2001
19	February	29	2004	March 1, 2004
20	February	29	2001	Invalid Input Date
21, 22	February	30	2001	Invalid Input Date

7.4 Test Cases for the Commission Problem

The commission problem is not well served by a decision table analysis. This is not surprising because very little decisional logic is used in the problem. Because the variables in the equivalence classes are truly independent, no impossible rules will occur in a decision table in which conditions correspond to the equivalence classes. Thus, we will have the same test cases as we did for equivalence class testing.

7.5 Guidelines and Observations

As with the other testing techniques, decision table-based testing works well for some applications (such as NextDate) and is not worth the trouble for others (such as the commission problem). Not surprisingly, the situations in which it works well are those in which a lot of decision making takes place (such as the triangle problem), and those in which important logical relationships exist among input variables (the NextDate function).