CS 589

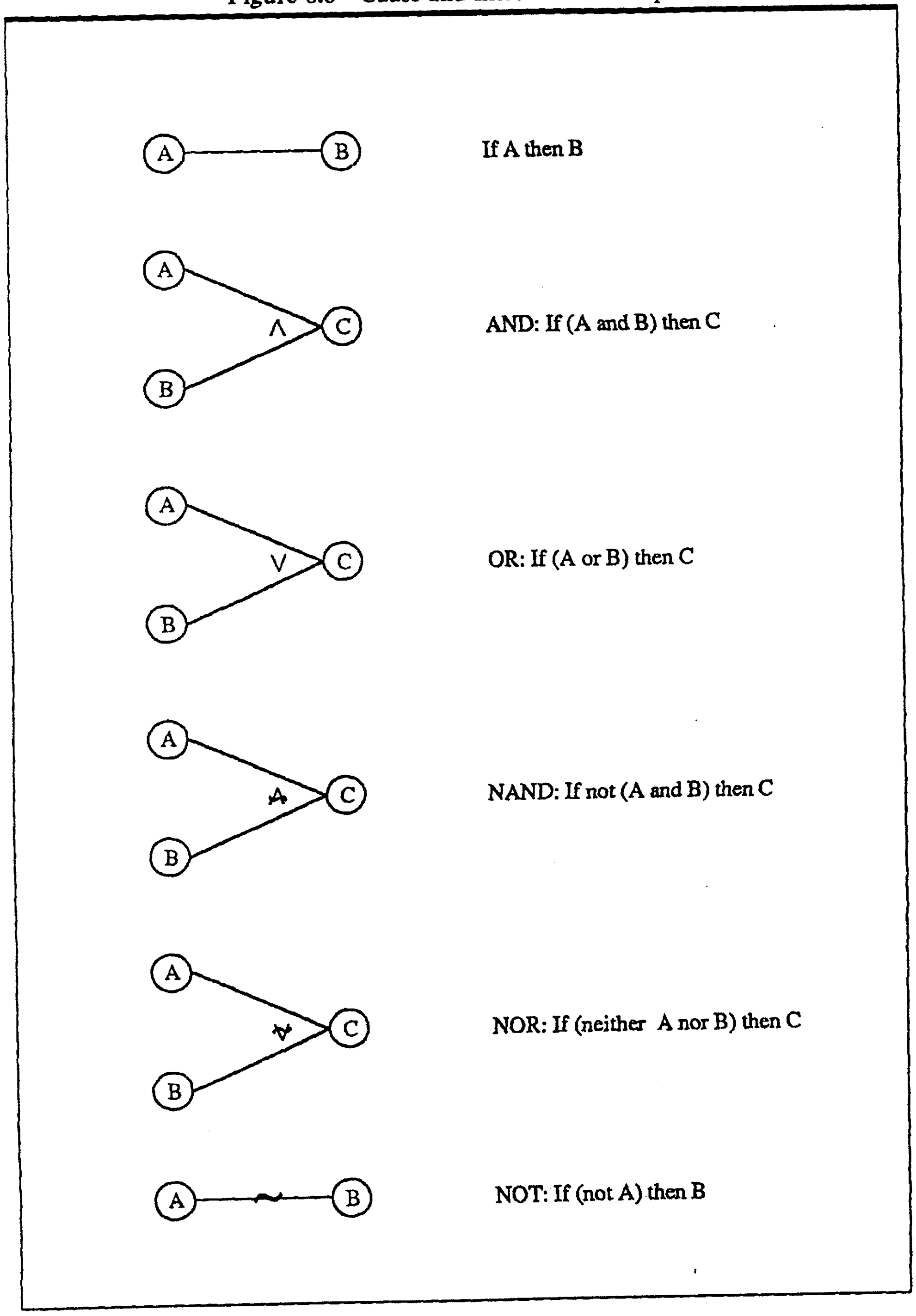
HANDOUT #1

Black-Box Testing

Cause-Effect Graphing[*]

* Shari Lawrence Pfleeger, Software Engineering: The Production of Quality Software, Macmillan Pulishing Company.

Figure 8.6 Cause and Effect Relationships



An example will show us how such a graph is built. Suppose we are testing a water level monitoring system. The requirements definition for one of the system functions reads as follows:

> The system sends a message to the dam operator about the safety of the lake level.

Corresponding to this requirement is a design description.

> INPUT: The syntax of the function is
>
> LEVEL(*A*, *B*)
>
> where *A* is the height in feet of the water behind the dam, and *B* is the number of inches of rain in the last twenty-four-hour period.
>
> PROCESSING: The function calculates whether the water level is within a safe range, is too high, or is too low.
>
> OUTPUT: The screen shows one of the following messages:
>
> 1. "LEVEL = SAFE" when result is safe or low
> 2. "LEVEL = HIGH" when result is high
> 3. "INVALID SYNTAX"
>
> depending on the result of the calculation.

We can separate these requirements into five causes:

Cause 1. The first five characters of the command are 'LEVEL'.

Cause 2. The command contains exactly two parameters separated by a comma and enclosed in parentheses.

Cause 3. The parameters A and B are real numbers such that the water level is calculated to be LOW.

Cause 4. The parameters A and B are real numbers such that the water level is calculated to be SAFE.

Cause 5. The parameters A and B are real numbers such that the water level is calculated to be HIGH.

We can also describe three effects.

Effect E1. The message "LEVEL = SAFE" is displayed on the screen.

Effect E2. The message "LEVEL = HIGH" is displayed on the screen.

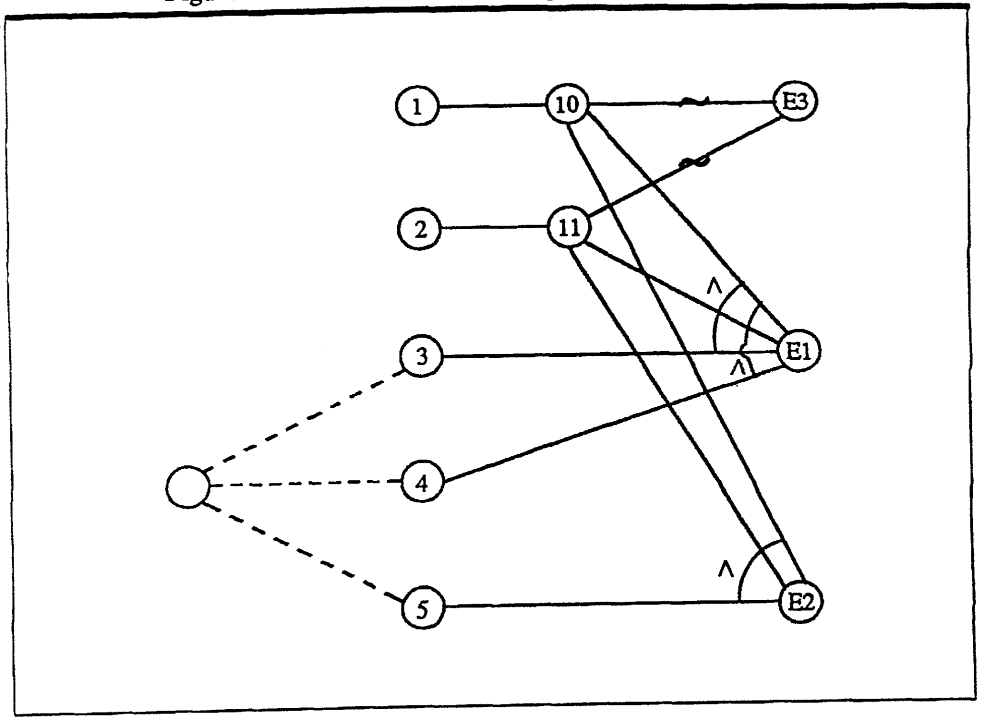
Effect E3. The message "INVALID SYNTAX" is printed out.

These become the nodes of our graph. However, the function includes a check on the parameters to be sure that they are passed properly. To reflect this, we establish two intermediate nodes:

Node 10. The command is syntactically valid.

Node 11. The operands are syntactically valid.

Figure 8.7 Cause and Effect Graph for LEVEL Function



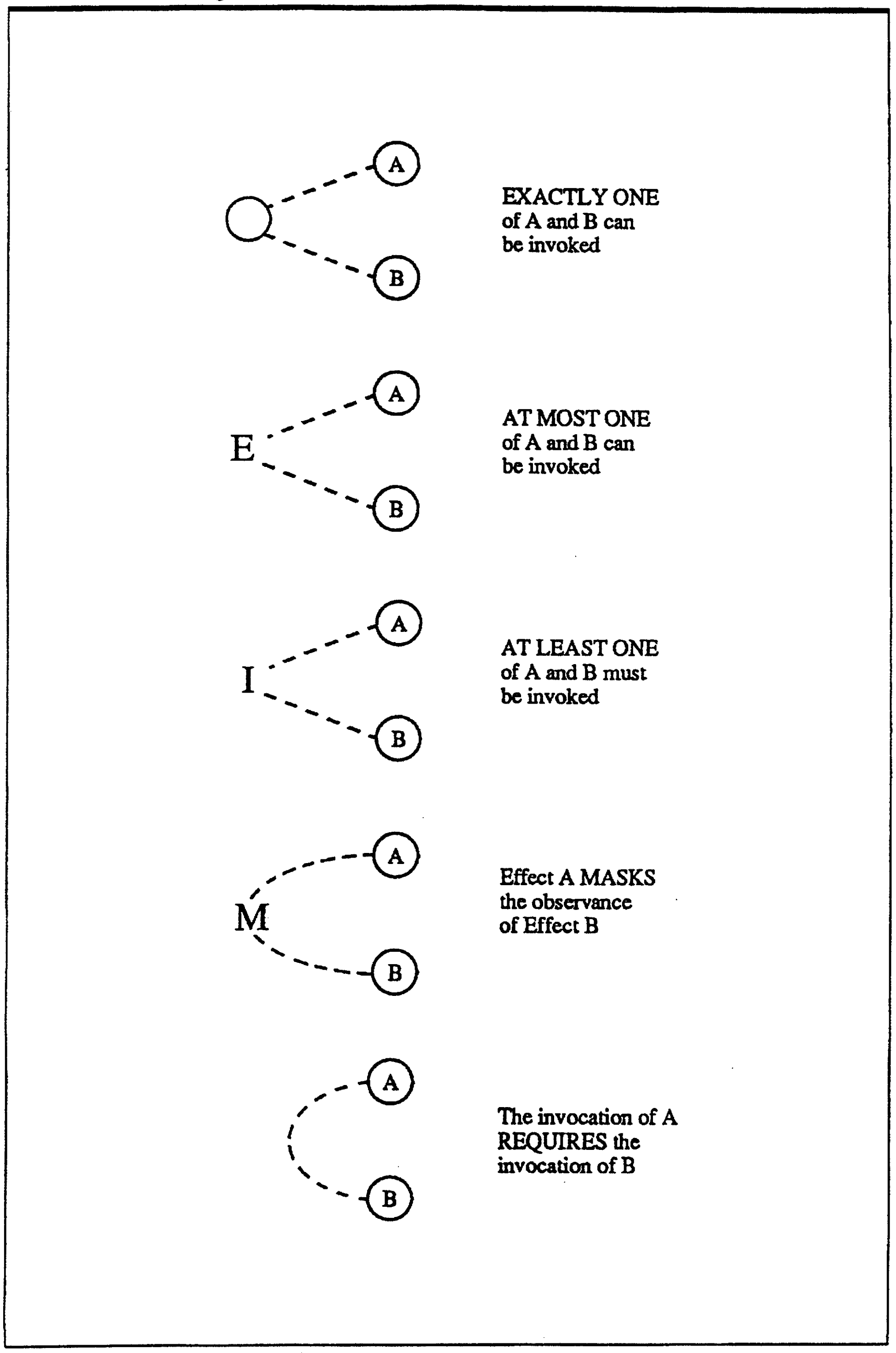
We can draw the relationships between cause and effect as shown in Figure 8.7. Notice that there are dashed lines to the left of the effects. These lines mean that exactly one effect can result. Other notations can be made on cause and effect graphs to provide additional information. Figure 8.8 illustrates some of the possibilities. By looking at the graph, we can tell if

- At most one of a set of conditions can be invoked
- At least one of a set of conditions must be invoked
- Exactly one of a set of conditions can be invoked
- One effect masks the observance of another effect
- Invocation of one effect requires the invocation of another

At this point, we are ready to define a *decision table* using the information from the cause and effect graph. We put a row in the table for each cause or effect. Thus, in our example, our decision table needs five rows for the causes and three rows for the effects. The columns of the decision table correspond to the test cases. We define the columns by examining each effect and listing all combinations of causes that can lead to that effect.

In our LEVEL example, we can determine the number of columns in the decision table by examining the lines flowing into the effect nodes of the cause and effect graph. We see in Figure 8.7 that there are two separate lines flowing into E3; each corresponds to a column. There are four lines flowing into E1, but only two

Figure 8.8 Additional Cause and Effect Notation


A
B
EXACTLY ONE of A and B can be invoked
E
A
B
AT MOST ONE of A and B can be invoked
I
A
B
AT LEAST ONE of A and B must be invoked
M
A
B
Effect A MASKS the observance of Effect B
A
B
The invocation of A REQUIRES the invocation of B

combinations yield the effect. Each of the combinations is a column in the table. Finally, only one combination of lines results in effect E2, so we have our fifth column.

Each column of the decision table represents a set of *states* of causes and effects. We keep track of the states of other conditions when a particular combination is invoked. We indicate the condition of the cause by placing an I in the table when the cause is invoked or true, or an S if the cause is suppressed or false. If we do not care whether the cause is invoked or suppressed, we can use an X to mark the "don't-care" state. Finally, we indicate whether a particular effect is absent (A) or present (P).

For testing the LEVEL function, the five columns in Table 8.2 display the relationship between invocation of causes and the resultant effects. If causes 1 and 2 are true (that is, the command and parameters are valid), then effect depends on whether causes 3, 4, or 5 are true. If cause 1 is true but cause 2 is false, the effect is already determined, and we don't care about the state of causes 3, 4, or 5. Similarly, if cause 1 is false, we no longer care about the states of other causes.

Note that theoretically we could have generated 32 test cases: five causes in each of two states yield 2^5 possibilities. Thus, the use of a cause and effect graph substantially decreases the number of test cases we must consider.

Table 8.2 Decision Table for Cause and Effect Graph

		Tests				
		1	2	3	4	5
	1	I	I	I	I	S
	2	I	I	I	X	I
Causes	3	I	S	S	X	X
	4	S	I	S	X	X
	5	S	S	I	X	X
	E1	P	P	A	A	A
Effects	E2	A	A	P	A	A
	E3	A	A	A	P	P

S = Suppressed I = Invoked A = Absent P = Present X = Don't Care

In general, we can reduce the number of test cases even more by using our knowledge of the causes to eliminate certain other combinations. For example, if the number of test cases is high, we may assign a priority to each combination of causes. Then, we can eliminate the combinations of low priority. Similarly, we can eliminate those combinations that are unlikely to occur or for which testing is not economically justifiable.

In addition to reducing the number of test cases to consider, cause and effect graphs help us predict the possible outcomes of exercising the system. At the same time, the graphs find unintended side effects for certain combinations of causes. However, cause and effect graphing has several limitations. The graphs are not practical for systems that include time delays, iterations, or loops where the system reacts to feedback from some of its processes to perform other processes.

R. Binder, Testing object-oriented systems, Addison Wesley

◆ 6.3.5 Cause–Effect Graphs

A **cause–effect graph** may be useful for identifying and analyzing the relationships to be modeled in a decision table [Myers 79]. I first encountered cause–effect graphs in 1982 and was initially put off by their "spaghetti and meatball" appearance. After developing several cause–effect models, however, I now think they provide an effective technique for analyzing and representing logic relationships.

A node is drawn for each cause (decision variable) and effect (output action). The cause and effect nodes are placed on opposite sides of a sheet. A line from a cause to an effect indicates that the cause is a necessary condition for the effect. If an effect has only a single cause (line), then that cause is also a sufficient condition to produce the effect. If a single effect has two or more causes (lines), the logical relationship of the causes is annotated by symbols for logical *and* ($\wedge$) and logical *or* ($\vee$) placed between the lines. A cause whose negation is a necessary condition is shown by the logical *not* (~) imposed on the line. A single cause may be necessary for many effects; a single effect may have many necessary causes. Intermediate nodes may be introduced to simplify the graph. Figure 6.8 shows a cause–effect graph for the insurance renewal example.

Cause–effect graphing can offer benefits for the initial development of a specification and for test design when no decision table is available. It requires systematic consideration of decision variables, conditions, and actions. The an-

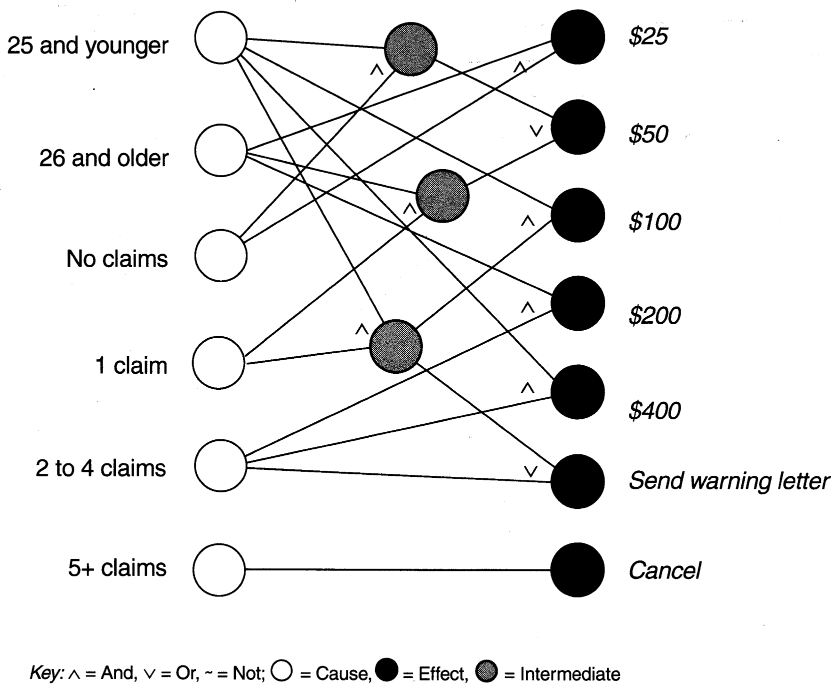


FIGURE 6.8 Insurance renewal cause–effect graph.

alyst may get visual clues about missing or incorrect relationships. A consistent, simple notation to record an evolving model is used. However, the general limitations of graphical techniques apply: As the complexity and scope of the modeled behavior increases, the graphs become busy and eventually intractable.

Boolean functions can be derived from a cause–effect graph. One function (a truth table) exists for each effect. If several effects are present, then the resulting decision table is a composite of several truth tables that happen to share decision variables and actions. It is usually easier to derive the function for each effect separately. These functions may then be overlaid to produce a single decision table for all effects.

Direct transcription of a cause–effect graph onto a column-wise decision table is summarized in Procedure 1. A more detailed example is presented in [Myers 79]. Direct transcription by inspection can be difficult and error-prone. An alternative is to transcribe Boolean formulas from a cause–effect graph and then rewrite them in sum-of-products form. This approach works for any arbitrarily complex cause–effect graph.

The first step in formula transcription is to write out the formula for each effect and intermediate node. The node name becomes the left side of the expression. The upstream node names and corresponding logical operators form the right side of the expression. These formulas are then reduced to a single sum-of-products formula by algebraic substitution and the rewrite laws given in Figure 6.5. A derivation of the sum-of-products function from the cause–effect graph in Figure 6.9 follows. Although this graph is not intimidating, the resulting formula is moderately complex.

1. *Transcribe node-to-node formulas from the graph.* Write the equation for effect Z and its predecessors P and R:

$$Z = PR \qquad (1)$$

Write the equation for intermediate node P and its predecessors:

$$P = A + {\sim}Q \qquad (2)$$

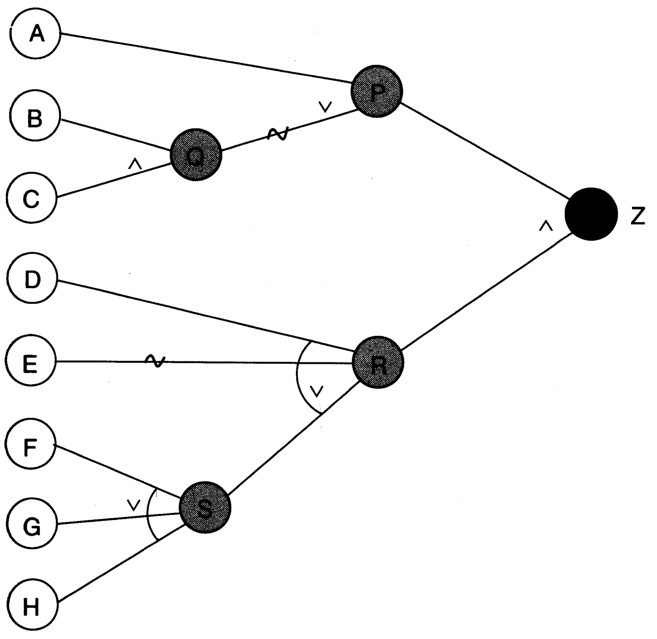


FIGURE 6.9 Cause–effect graph with eight variables.

```
Define and label a row in the condition section for each cause.
1.  Define and label a row in the action section for each effect.
2.  For each effect node:
        If there are one, two, or three causes for the effect:
            Try transcribing by inspection.
        fi
        If there are four or more causes for the effect:
        Transcribe node-to-node formulas from the graph.
            Write the formula for the effect and its predecessors.
            For each intermediate node:
                Write the formula for the intermediate node and its
                predecessors.
            rof
        Derive the complete Boolean formula.
            Replace intermediate variables by substitution until the ef-
            fect formula contains only cause variables.
            Factor and rewrite the effect formula into sum-of-
            products form using the Boolean laws (see Figure 6.5).
        fi
    rof
3.  Generate variant entries.
    For each effect formula:
        For each product term:
            Expand into a minterm
        rof
        Discard duplicate minterms.
        Generate a variant column for the minterm.
        Transcribe the truth values to the column.
    rof
```

Procedure 1 Transcribe Cause–Effect Graph to Decision Table

Write the equation for intermediate node Q and its predecessors:

$$Q = BC \tag{3}$$

Write the equation for intermediate node R and its predecessors:

$$R = D + {\sim}E + S \tag{4}$$

Write the equation for intermediate node S and its predecessors:

$$S = F + G + H \tag{5}$$

2. *Derive the complete Boolean formula in sum-of-products form.* Substitute Equation (2) for P and Equation (4) for R:

$$Z = (A + {\sim}Q)\ (D + {\sim}E + S) \tag{6}$$

Substitute Equation (3) for Q, and Equation (5) for S:

$$Z = (A + {\sim}(BC))\ (D + {\sim}E + (F + G + H)) \tag{7}$$

Factor product ${\sim}(BC)$ using De Morgan's law:

$$Z = (A + {\sim}B + {\sim}C)\ (D + {\sim}E + F + G + H) \tag{8}$$

Expand the implied products using the distributive law:

$$\begin{aligned} Z = {} & AD + A{\sim}E + AF + AG + AH + \\ & {\sim}BD + {\sim}B{\sim}E + {\sim}BF + {\sim}BG + {\sim}BH + \\ & {\sim}CD + {\sim}C{\sim}E + {\sim}CF + {\sim}CG + {\sim}CH \end{aligned} \tag{9}$$

The Boolean formula for the graph in Figure 6.9 is given by Equation (9). The fully enumerated truth table (not shown) would contain 2^8 variants.

Myers also provides a notation to model external constraints among causes. *Exclusive* causes are mutually exclusive but not required. If any is present, only one may be true; both may be false. *Inclusive* causes require that at least one must be true; all may be true. *Singleton* causes require that at least one but no more than one be true. Constraints are shown by dashed lines [Myers 79].

Modeling constraints apart from the decision table may be useful if it simplifies the testing problem. Constraints can also be modeled by adding a term for the constraint to the function. This representation explicitly incorporates external constraints and decision variables into the model. For example, to model an exclusive constraint on variables X and Y, $\sim(XY)$ is added to the formula. $X + Y$ is added to model an inclusive constraint, and $X \oplus Y$ (exclusive *or*) to model a singleton constraint. (The expansion of $X \oplus Y$ is $\sim XY + X\sim Y$). Adding explicit constraints does not change the relative size of the test suite, as it does not increase the number of variables. Because it is easy to make a mistake in complex algebraic substitution, the resulting formula should be verified (Figure 6.10). ■

A	B	C	D	Z = AB~C + AD	
0	0	0	0	0	@OR(@AND(+A2,+B2,@NOT(+C2)),@AND(+A2,+D2))
0	0	0	1	0	@OR(@AND(+A3,+B3,@NOT(+C3)),@AND(+A3,+D3))
0	0	1	0	0	@OR(@AND(+A4,+B4,@NOT(+C4)),@AND(+A4,+D4))
0	0	1	1	0	@OR(@AND(+A5,+B5,@NOT(+C5)),@AND(+A5,+D5))
0	1	0	0	0	@OR(@AND(+A6,+B6,@NOT(+C6)),@AND(+A6,+D6))
0	1	0	1	0	@OR(@AND(+A7,+B7,@NOT(+C7)),@AND(+A7,+D7))
0	1	1	0	0	@OR(@AND(+A8,+B8,@NOT(+C8)),@AND(+A8,+D8))
0	1	1	1	0	@OR(@AND(+A9,+B9,@NOT(+C9)),@AND(+A9,+D9))
1	0	0	0	0	@OR(@AND(+A10,+B10,@NOT(+C10)),@AND(+A10,+D10))
1	0	0	1	1	@OR(@AND(+A11,+B11,@NOT(+C11)),@AND(+A11,+D11))
1	0	1	0	0	@OR(@AND(+A12,+B12,@NOT(+C12)),@AND(+A12,+D12))
1	0	1	1	1	@OR(@AND(+A13,+B13,@NOT(+C13)),@AND(+A13,+D13))
1	1	0	0	1	@OR(@AND(+A14,+B14,@NOT(+C14)),@AND(+A14,+D14))
1	1	0	1	1	@OR(@AND(+A15,+B15,@NOT(+C15)),@AND(+A15,+D15))
1	1	1	0	0	@OR(@AND(+A16,+B16,@NOT(+C16)),@AND(+A16,+D16))
1	1	1	1	1	@OR(@AND(+A17,+B17,@NOT(+C17)),@AND(+A17,+D17))

FIGURE 6.10 Spreadsheet for checking the boiler control truth table.