4.5.4 Predicate Coverage Criterion

We refer to the partial CFG of Figure 4.9*a* to explain the concept of predicate coverage. OB1, OB2, OB3, and OB are four Boolean variables. The program computes the values of the individual variables OB1, OB2, and OB3— details of their computation are irrelevant to our discussion and have been omitted. Next, OB is computed as shown in the CFG. The CFG checks the value of OB and executes either OBlock1 or OBlock2 depending on whether OB evaluates to true or false, respectively.

We need to design just two test cases to achieve both statement coverage and branch coverage. We select inputs such that the four Boolean conditions in Figure 4.9*a* evaluate to the values shown in Table 4.6. The reader may note that we have shown just one way of forcing OB to true. If we select inputs so that these two cases hold, then we do not observe the effect of the computations taking place in nodes **2** and **3**. There may be faults in the computation parts of nodes **2** and **3** such that OB2 and OB3 always evaluate to false.

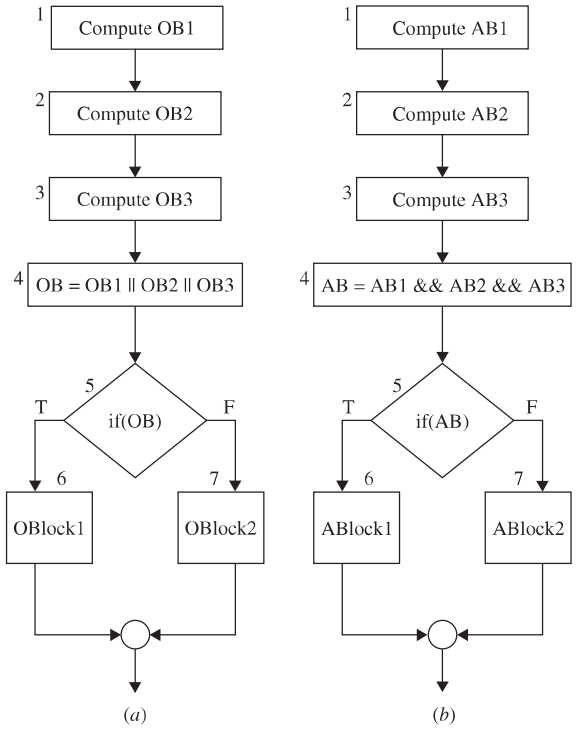

Figure 4.9 Partial CFG with (*a*) OR operation and (*b*) AND operation.

TABLE 4.6 Two Cases for Complete Statement and Branch Coverage of CFG of Figure 4.9*a*

Cases	OB1	OB2	OB3	OB
1	T	F	F	T
2	F	F	F	F

Therefore, there is a need to design test cases such that a path is executed under all possible conditions. The False branch of node **5** (Figure 4.9*a*) is executed under exactly one condition, namely, when OB1 = False, OB2 = False, and OB3 = False, whereas the true branch executes under *seven* conditions. If all possible combinations of truth values of the conditions affecting a selected path have been explored under some tests, then we say that *predicate coverage* has been achieved. Therefore, the path taking the true branch of node **5** in Figure 4.9*a* must be executed for all seven possible combinations of truth values of OB1, OB2, and OB3 which result in OB = True.

A similar situation holds for the partial CFG shown in Figure 4.9*b*, where AB1, AB2, AB3, and AB are Boolean variables.

4.6 GENERATING TEST INPUT

In Section 4.5 we explained the concept of path selection criteria to cover certain aspects of a program with a set of paths. The program aspects we considered were all statements, true and false evaluations of each condition, and combinations of conditions affecting execution of a path. Now, having identified a path, the question is how to select input values such that when the program is executed with the selected inputs, the chosen paths get executed. In other words, we need to identify inputs to force the executions of the paths. In the following, we define a few terms and give an example of generating test inputs for a selected path.

1. *Input Vector:* An input vector is a collection of all data entities read by the routine whose values must be fixed prior to entering the routine. Members of an input vector of a routine can take different forms as listed below:

- Input arguments to a routine
- Global variables and constants
- Files
- Contents of registers in assembly language programming
- Network connections
- Timers

A file is a complex input element. In one case, mere existence of a file can be considered as an input, whereas in another case, contents of the file are considered