## Input Variables vs. Test Parameters

In the example above, we assumed that the parameters to be included in tests were taken from function calls in the program, *f* (*p*1, *p*2, …, *pn*), where each parameter had defined values or a range of values. In many cases, it will not be so obvious how to identify what should be included in the covering array and tests. The classic Ostrand and Balcer [151] category partitioning paper illustrates this common situation with the example of a “find” command, which takes user input of a string and a file name and locates all lines containing the string. The format of the command is “find <*string*> <*filename*>, where <*string*> is one or more quoted strings of characters such as “john”, “john smith”, or “john” “smith”. Search strings may include the escape character (backslash) for quotes, to select strings with embedded quotes in the file, such as “\”john\”” to report the presence of lines containing *john* in quotes within the file. The command displays any lines containing one or more of the strings. This command has only two input variables, *string* and *filename*, so is combinatorial testing really useful here?

In fact, combinatorial methods can be highly effective for this common type of testing problem. To check the “find” command, testers will want to ensure that it handles inputs correctly. The input variables in this case are *string* and *filename*, but it is common to refer to such variables as *parameters*. We will distinguish between the two here, but follow conventional practice where the distinction is clear. The *test parameters* or *abstract parameters* identify characteristics of the command input variables. So the *test parameters* are in this case different from the two *input parameters*, *string* and *filename*. For example, the *string* input has characteristics such as length and presence of embedded blanks. Clearly, there are many ways to select test parameters, so engineering judgment must be used to determine what are most important. One selection could be the following, where *file\_length* is the length in characters of the file being searched:

String length: {0, 1, 1..*file\_length*, >*file\_length*}

Quotes: {yes, no, improperly formatted quotes}

Blanks: {0, 1, >1}

Embedded quotes: {0, 1, 1 escaped, 1 not escaped}

Filename: {valid, invalid}

Strings in command line: {0, 1, >1}

String presence in file: {0, 1, >1}

For these seven test parameters, we have 213442= 2,592 possible combinations of test parameter values. If we choose to test all 2-way interactions we need only 19 tests. For 3 and 4-way combinations, we need only 67 and 218 tests respectively. Because the number of tests grows only as log *n* for *n* parameters, we can do very thorough testing at relatively low cost for problems like this. That is, we can include a large number of characteristics to be used as test parameters without significantly increasing the test burden. In the problem above, if we used only the first four of the test parameters, instead of all seven, the number of tests required for *t* = 2, 3, and 4 respectively are 16, 54, and 144. Using all seven characteristics means much more thorough testing with relatively little increase in test set size.

When testing combinations of input characteristics as above, we must be careful that the test set captures enough important cases. For the find command, testing 3-way or 4-way combinations of the seven characteristics should be an excellent sample of test cases that can detect problems. That is, the tests will include both valid and invalid strings. In some cases, there may be a need to ensure the presence of test cases with a number of specific characteristics. For example, passwords may be required to (1) exceed a certain length, (2) contain numerics, and (3) contain special characters. A 2-way covering array might not include any valid cases, because it contains all pairs but three characteristics must be true to constitute a valid test case. We may need to supplement the covering array with some additional tests in this case. Sect. 5.4.3 discusses this situation in more detail, along with ways to deal with it. A good case study that illustrates the use of some of these techniques can be found in Borazjany et al. [21].