Example 2.7 Consider the following program that takes two non-negative integers $x1$ and $y1$ as inputs and computes their greatest common divisor.

```
1     int gcd(int x1, int y1){
2           int x, y;
3           input(x1,y1);
4           x=x1; y=y1;
5           while(x!=y){
6                 if(x>y)
7                       x=x-y;
8                 else
9                       y=y-x;
10          }
11          return x;
12    }
```

The above program has an infinite number of paths one each for a set of pairs of integers. This set constitutes the domain of that path. Following is a list of a few paths through this program. These paths are labeled p_0, p_{11}, and p_{12} where the first of the two indices denotes the number of times the loop is iterated and the second one indicates the variation within the loop due to the condition in the `if` statement.

Loop iterated zero times: $p_0 = (1, 2, 3, 4, 5, 11)$
Loop iterated once:
For $x > y$: p_{11}: (1, 2, 3, 4, 5, 6, 7, 10, 5, 11)
For $x \leq y$: p_{12}: (1, 2, 3, 4, 5, 6, 9, 10, 5, 11)

A path domain can be computed by first finding the path conditions. The set of inputs that satisfy these conditions is the path domain for the path under consideration. For a program with no conditions the path domain is the input domain.

Note that each loop iteration leads to one additional path. This path depends on the code in the loop body. In the above example, the path could correspond to either the true or the false branch of the `if` statement.

To determine the path domains, let us compute the condition associate with each path. Following are the conditions, and their simplified versions, for each of the three paths above.

$p_0 : x1 = y1$
$p_{11} : (x1 \neq y1 \wedge (x1 - y1) = y1) \rightarrow x1 = 2 * y1$
$p_{12} : (x1 \neq y1 \wedge x1 = (y1 - x1)) \rightarrow 2 * x1 = y1$

Note that each path condition is expressed in terms of the program inputs $x1$ and $y1$. Thus, for example, the first time the loop

condition $x \neq y$ is evaluated, the values of program variables x and y are, respectively, $x1$ and $y1$. Hence the loop condition expressed in terms of input variables is $x1 \neq y1$ and the corresponding path condition is $x1 = y1$. Similarly, the interpreted path conditions for p_{11} and p_{12} are, respectively, $x1 = 2 * y1$ and $2 * x1 = y1$.

Expressing a condition along a path in terms of program input variables is known as *interpretation* of that condition. Obviously, the interpretation depends on the values of the program variables at the time the condition is evaluated.

Thus, the domain for path p_0 consists of all inputs for which $x1 = y1$. Similarly, the domains for the remaining two paths consist of all inputs that satisfy the corresponding conditions. Sample inputs in the domain of each of the three paths are given in the following as pairs of input integers.

p_0 : (4, 4), (10, 10)
p_{11} : (4, 2), (8, 4)
p_{12} : (4, 8), (5, 10)

Example 2.8 Consider a program that takes two inputs denoted by integer variables x and y. Now consider the following sequence of three conditions that must evaluate to true in order for a path in this program to be executed.

$y < x + 2$
$y > 1$
$x < 0$

The above conditions define a path domain shown in Figure 2.6. Each of the above three conditions leads to a *border segment*, or simply *border*, shown as a darkened line and marked with an arrow. The path domain is the shaded area bounded by the three borders. All values of x and y that lie inside the domain will cause the path to be executed. Points that lie outside the domain or at the boundary will cause at least one of the above conditions to be false and hence the path will not be executed. For example, point c in the figure corresponds to $(-0.5, 1)$. For this point the condition $y > 1$ is false.

Every path domain has a border along which lie useful tests.

The path conditions in the above examples are linear and hence lead to straight line borders. Also, here we have assumed that the inputs are numbers, integers, or floating point values. In general, path condition dimensions could be arbitrary. Let $relop = \{<, \leq, >, \geq, =, \neq\}$. A general

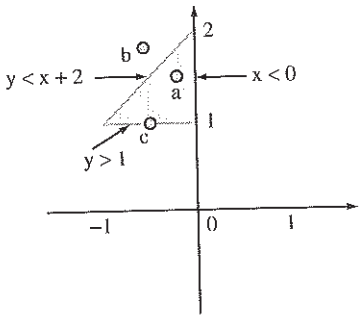

Figure 2.6 Pictorial representation of a path domain corresponding to the condition $y < x + 2 \wedge y > 1 \wedge x < 0$. The shaded triangular area represents the path domain which is a subdomain of the entire program domain. Three sample points labeled *a*, *b*, and *c*, are shown. Point *a* lies inside the path domain, *b* lies outside, and *c* lies on one of the borders.

form of a set of m path conditions in n inputs $x_1, x_2, \ldots, x_n$ can be expressed as follows.

$$\begin{array}{lll} f1(x_1, x_2, \ldots, x_n) & relop & 0 \\ f2(x_1, x_2, \ldots, x_n) & relop & 0 \\ \vdots & & \\ fm(x_1, x_2, \ldots, x_n) & relop & 0 \end{array}$$

In case all functions in the above set of conditions are linear, the path conditions can be written as follows.

$$\begin{array}{ll} a_{11}x_1 + a_{12}x_2, \ldots + a_{1n}x_n + c_1 & relop \quad 0 \\ a_{21}x_1 + a_{22}x_2, \ldots + a_{2n}x_n + c_2 & relop \quad 0 \\ \vdots & \\ a_{m1}x_1 + a_{m2}x_2, \ldots + q_{nm}x_n + c_m & relop \quad 0 \end{array}$$

In the above conditions, a_{ij} denote the $m \times n$ coefficients of the input variables and c_j are m constants. Using the above standard notation, the path conditions in Example 2.8 can be expressed as follows.

$$\begin{array}{lll} -x + y - 2 & < & 0 \\ -y + 1 & < & 0 \\ x & < & 0 \end{array}$$

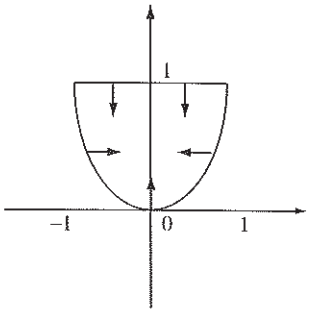

Figure 2.7 Non-linear path domain corresponding to the path condition $y > x^2 \wedge y < 1$. The arrows indicate the closed boundary of the parabolic domain.

In the above set of equations, the coefficients are given below.

$$\begin{array}{lll} a_{11} = -1 & a_{12} = 1 & c_1 = -2 \\ a_{21} = 0 & a_{22} = -1 & c_2 = 1 \\ a_{31} = -1 & a_{32} = 0 & c_3 = 0 \end{array}$$

Path conditions could also be non-linear. They could also contain arrays, strings, and other data types commonly found in programming languages. The next example illustrates a few other types of path conditions and the corresponding path domains.

Non-linear path conditions may lead to complex path domains thereby increasing the difficulty of finding a minimal set of tests.

Example 2.9 Consider the following two path conditions.

$$\begin{array}{lll} y - x^2 & > & 0 \\ y & < & 1 \end{array}$$

Figure 2.7 is a graphical representation of the path domain that corresponds to the above conditions. The domain consists of points inside of a parabola bounded at the top by a line corresponding to the constraint $y < 1$.

Example 2.10 Often a condition in an `if` or a `while` loop contains reference to an array element. For instance, consider the following statements.

```
1   int x, n;
2   input (x, n);
3   inputArray(n, A);
4   ....
```

```
5   if(A[i]>x){
6   ....
```

The input domain of the above program consists of triples (x, n, A). The constraint $A[i] > x$ defines a path domain consisting of all triples (x, n, A'), where A' is an array whose i^{th} element is greater than x. Of course, to define this domain more precisely one needs to know the range of variable i. Suppose that $0 \leq i \leq 10$. In that case the path domain consists of all arrays of length 0 through 10 at least one of whose elements satisfies the path condition.

Just as in the case of arrays, it is often challenging to define path domains for abstract data types that occur in OO languages such as Java and C++. Consider, for example, a stack as used in the following program segment.

```
1   Stack s;
2    if (full(s))
3        output("Stack is full");
4   ....
```

The path condition in the above program segment is *full(s)*. Given stacks of arbitrary sizes, the path domain obviously consists of only stacks that are full. In situations like this, the program domain and the path domain are generally restricted for the purpose of testing by limiting stack size. Once done, one could consider stacks that are empty, full, or neither. Doing so partitions the program domain with respect to a stack into three subdomains one of which is a path domain.

That brings us to the end of this section to illustrate path conditions and path domains. Later in Chapter 4, we will see how to use path conditions to generate tests using a technique known as *domain testing*. (Also see Exercise 2.2.)

A domain error is said to exist in a program if for some input an incorrect path leads to an incorrect output.

A computation error is said to exist in a program if for some input a correct path leads to an incorrect output.

2.2.6 Domain and computation errors

Domain and computation errors are two error types based on the manner in which a program input leads to incorrect behavior. Specifically, a program P is said to contain a *domain* error if for a given input it follows a wrong path that leads to an incorrect output. P is said to contain a *computation* error if an input forces it to follow the correct path resulting in an incorrect output. An error in a predicate used in a conditional statement such as an `if` or a `while` leads to a domain error. An error in an

SUMMARY

In this chapter, we have presented some basic mathematical concepts likely to be encountered by any tester. Control flow graphs represent the flow of control in a program. These graphs are used in program analysis. Many test tools transform a program into its control flow graph for the purpose of analysis. Section 2.2 explains what a control flow graph is and how to construct one. Basis paths are introduced here. Later in Chapter 4, basis paths are used to derive test cases.

Dominators and program dependence graphs are useful in static analysis of programs. Chapter 9 shows how to use dominators for test minimization.

An finite state machine recognizes strings that can be generated using regular expressions. Section 2.6 covers strings and languages useful in understanding the model based testing concepts presented in Chapter 5.

Exercises

2.1 (a) Calculate the length of the path traversed when P2.2 is executed on an input sequence containing N integers. (b) Suppose that the statement on line 8 of P2.2 is replaced by the following.

```
8    if(num>0)product=product*num;
```

Calculate the number of distinct paths in the modified P2.2 if the length of the input sequence can be 0, 1, or 2. (c) For an input sequence of length N, what is the maximum length of the path traversed by the modified P2.2?

2.2 (a) List the different paths in the program in Example 2.7 when the loop body is traversed twice. (b) For each path list the path conditions and a few sample values in the corresponding input domain.

2.3 Find the basis set and the corresponding path vectors for the flow graph in Figure 2.2. Construct a path p through this flow graph that is not a basis path. Find the path vector V_p for p. Express V_p as a linear combination of the path vectors for the basis paths.

2.4 Solve Exercise 2.7 but for the program in Example 2.7.

2.5 Let P denote a program that contains a total of n simple conditions in `if` and `while` statements. What is the maximum number of elements in the basis set for P? What is the largest path vector in P?

2.6 The basis set for a program is constructed using its CFG. Now suppose that the program contains compound predicates. Do you think the basis set for this program will depend on how compound predicates are represented in the flow graph? Explain your answer.