Slice Based Testing of CGI Based Web Applications

Madhusmita Sahu and Durga Prasad Mohapatra

Department of CSE, National Institute of Technology, Rourkela-769008
Odisha, India
{513CS8041,durga}@nitrkl.ac.in

Abstract. We propose a slice based testing technique to generate test paths for web applications. Our web application is based on *Common Gateway Interface* (CGI) and we have used *PERL* programming language. Our technique uses slicing criterion for all variables defined and used in the program. Then, it computes the slices for each of these criteria and generates test paths. Finally, we generate test cases using these test paths.

Keywords: Program Slicing, CGI, PERL, Test Case.

1 Introduction

Mark Weiser [1] introduced technique of program slicing. In this paper, we propose an algorithm named Web Slice Testing (WST) Algorithm to generate test paths for web applications using slicing [3]. Rest of paper is organized as follows. In Section 2, slice based testing of CGI programs is discussed. Section 3 concludes paper.

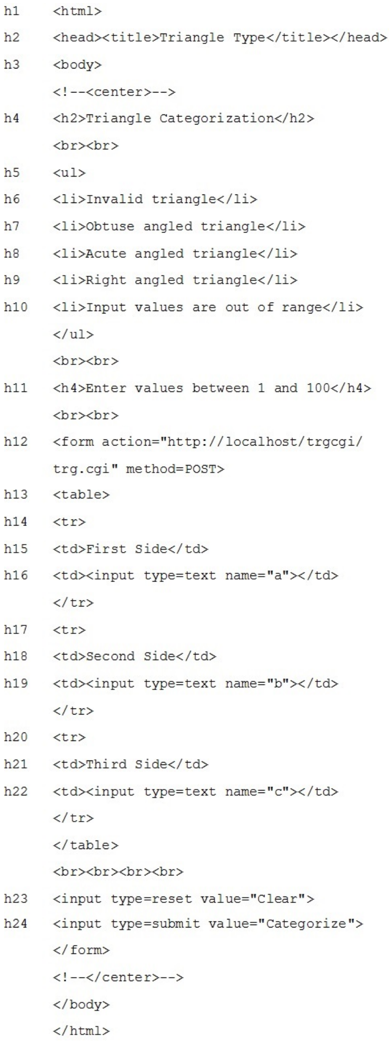
2 Proposed Work

Let u be a node corresponding to statement s in a program P and $slice(u)$ be static slice w.r.t. slicing criterion $< s, v >$ where v is a variable defined or used at s.

Web Slice Testing (WST) Algorithm

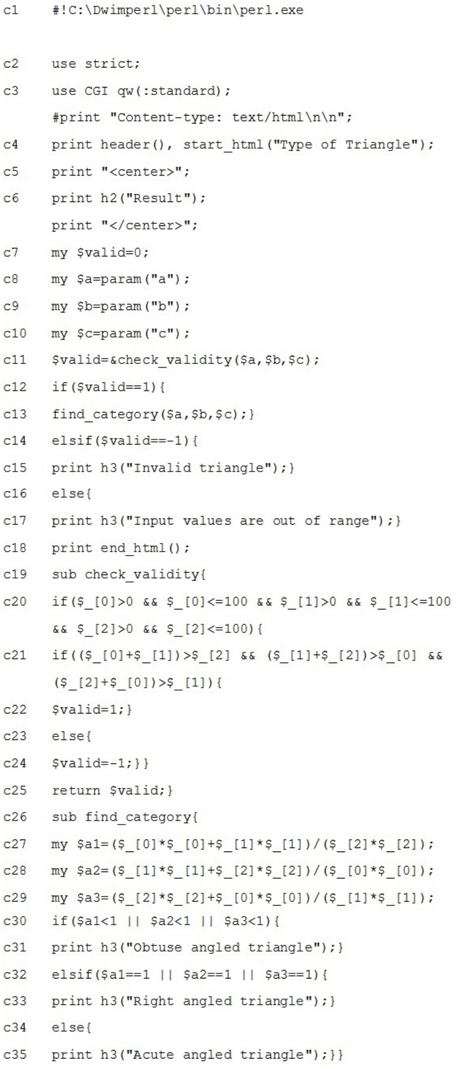
1. Construct the Web Application Dependence Graph (WADG) statically once.
2. Compute static slices with respect to each slicing criterion $< s, v >$ using two-phase algorithm proposed by Horowitz et al. [2].
3. Let $slice(u_1)$ and $slice(u_2)$ be two slices.
   (a) If $slice(u_1) \subset slice(u_2)$, then discard $slice(u_1)$ and retain $slice(u_2)$.
4. Generate the test paths. Let node u correspond to statement s.
   (a) Perform Breadth First Search (BFS) starting from the node present in the $slice(u)$ whose indegree is zero.

R. Natarajan et al. (Eds.): ICDCIT 2015, LNCS 8956, pp. 253–256, 2015.



```
h1    <html>
h2    <head><title>Triangle Type</title></head>
h3    <body>
      <!--<center>-->
h4    <h2>Triangle Categorization</h2>
      <br><br>
h5    <ul>
h6    <li>Invalid triangle</li>
h7    <li>Obtuse angled triangle</li>
h8    <li>Acute angled triangle</li>
h9    <li>Right angled triangle</li>
h10   <li>Input values are out of range</li>
      </ul>
      <br><br>
h11   <h4>Enter values between 1 and 100</h4>
      <br><br>
h12   <form action="http://localhost/trgcgi/
      trg.cgi" method=POST>
h13   <table>
h14   <tr>
h15   <td>First Side</td>
h16   <td><input type=text name="a"></td>
      </tr>
h17   <tr>
h18   <td>Second Side</td>
h19   <td><input type=text name="b"></td>
      </tr>
h20   <tr>
h21   <td>Third Side</td>
h22   <td><input type=text name="c"></td>
      </tr>
      </table>
      <br><br><br><br>
h23   <input type=reset value="Clear">
h24   <input type=submit value="Categorize">
      </form>
      <!--</center>-->
      </body>
      </html>
```

Fig. 1. An example HTML code to input values of 3 sides of a triangle (`trg.html`)



```
c1    #!C:\Dwimperl\perl\bin\perl.exe

c2    use strict;
c3    use CGI qw(:standard);
      #print "Content-type: text/html\n\n";
c4    print header(), start_html("Type of Triangle");
c5    print "<center>";
c6    print h2("Result");
      print "</center>";
c7    my $valid=0;
c8    my $a=param("a");
c9    my $b=param("b");
c10   my $c=param("c");
c11   $valid=&check_validity($a,$b,$c);
c12   if($valid==1){
c13   find_category($a,$b,$c);}
c14   elsif($valid==-1){
c15   print h3("Invalid triangle");}
c16   else{
c17   print h3("Input values are out of range");}
c18   print end_html();
c19   sub check_validity{
c20   if($_[0]>0 && $_[0]<=100 && $_[1]>0 && $_[1]<=100
      && $_[2]>0 && $_[2]<=100){
c21   if(($_[0]+$_[1])>$_[2] && ($_[1]+$_[2])>$_[0] &&
      ($_[2]+$_[0])>$_[1]){
c22   $valid=1;}
c23   else{
c24   $valid=-1;}}
c25   return $valid;}
c26   sub find_category{
c27   my $a1=($_[0]*$_[0]+$_[1]*$_[1])/($_[2]*$_[2]);
c28   my $a2=($_[1]*$_[1]+$_[2]*$_[2])/($_[0]*$_[0]);
c29   my $a3=($_[2]*$_[2]+$_[0]*$_[0])/($_[1]*$_[1]);
c30   if($a1<1 || $a2<1 || $a3<1){
c31   print h3("Obtuse angled triangle");}
c32   elsif($a1==1 || $a2==1 || $a3==1){
c33   print h3("Right angled triangle");}
c34   else{
c35   print h3("Acute angled triangle");}}
```

Fig. 2. CGI code to determine the category of triangle based on the values of 3 sides (`trg.cgi`)

(b) During traversal, perform the followings:
   i. Traverse the ancestor nodes of u whose outdegree is zero.
   ii. Let k be a descendant node of u whose outdegree is zero. Then, discard k if $k \notin slice(u)$.
   iii. Let k be an ancestor node of u whose outdegree is zero. Then, discard k if $k \notin slice(u)$.
   iv. Let k be an ancestor node of u representing the end of the program. Then discard k if $k \notin slice(u)$.

5. Repeat Steps 3 to 4 for all slices computed in Step 2.
6. Design test cases randomly to exercise all test paths obtained in Step 5.

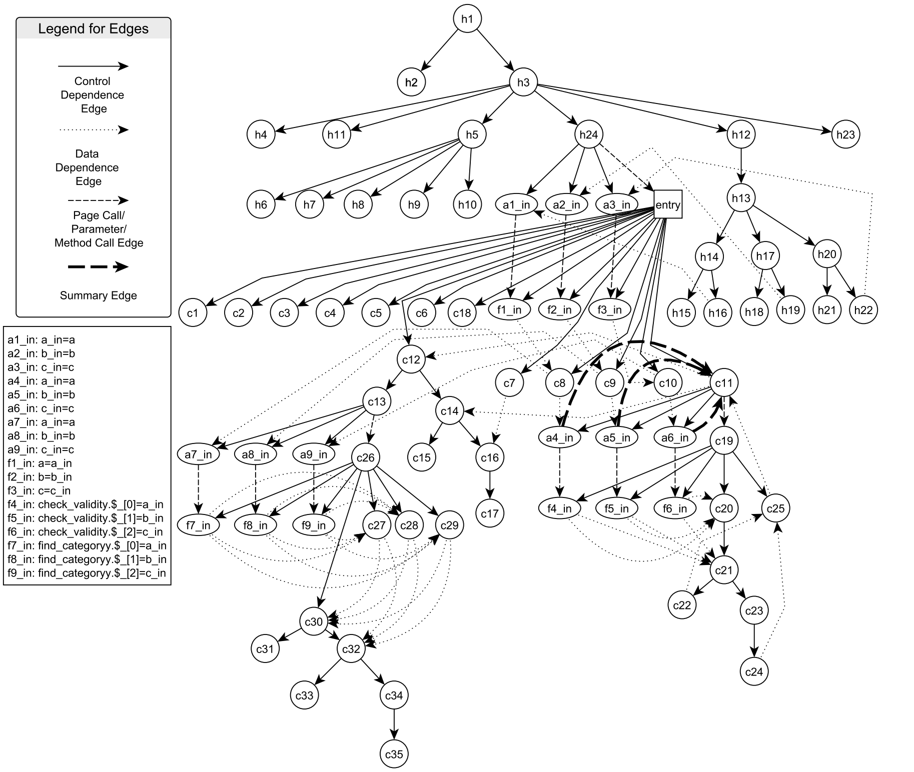


Fig. 3. WADG of the programs given in Fig. 1 and Fig. 2

Working of the Algorithm: Triangle Classification Problem. Consider a program for classification of a triangle. Its input is a triplet of positive integers (say a,b and c) and $1 \leq a \leq 100$, $1 \leq b \leq 100$ and $1 \leq c \leq 100$. The output may have one of the following words: [Acute angled triangle, Obtuse angled triangle, Right angled triangle, Invalid triangle, Invalid Input].

Fig. 1 gives HTML code to input values for 3 sides of the triangle and Fig. 2 shows CGI code to solve the above Triangle Classification Problem. The WADG of programs given in Fig. 1 and Fig. 2 is shown in Fig. 3. We compute static slices for all variables defined and used at some statements and then compare two slices to find whether one slice is subset of another. Then, we perform BFS starting from a node present in retained slice whose indegree is zero. Applying Step 4 of WST algorithm, we find test path for that slice. Table 1 shows test paths to be executed in each slice and test cases for example program given in Fig. 1 and Fig. 2. Symbol p˜q denotes all nodes present on path between nodes p and q.

Table 1. Test cases designed from the slices computed

Test Case ID	Slicing Criterion	Test path to be executed	Test cases			
			Input			Expected Output
			a	b	c	
T1	slice(c15)	h1˜h24,c1˜c11,c19˜c21,c23, c24,c25,c12,c14,c15,c18	30	10	15	Invalid triangle
T2	slice(c17)	h1˜h24,c1˜c11,c19,c20,c25,c12,c14,c16˜c18	10	-1	6	Invalid input
T3	slice(c31)	h1˜h24,c1˜c11,c19˜c22,c25, c12,c13,c26˜c31,c18	30	20	40	Obtuse angled triangle
T4	slice(c33)	h1˜h24,c1˜c11,c19˜c22,c25, c12,c13,c26˜c30,c32,c33,c18	30	40	50	Right angled triangle
T5	slice(c35)	h1˜h24,c1˜c11,c19˜c22,c25, c12,c13,c26˜c30,c32,c34,c35,c18	50	60	40	Acute angled triangle

3 Conclusion

We presented a technique to test web applications based on *Common Gateway Interface* (CGI) using program slicing and generated test cases. Our technique computed slices for all variables defined and used in program. We had performed breadth first search on intermediate representation WADG using only nodes present in those slices. This traversal yielded required test paths for those slices. Finally, we designed test cases using these test paths.

References

1. Weiser, M.: Program Slicing. IEEE Transactions on Software Engineering 10(4), 352–257 (1984)
2. Horwitz, S., Reps, T., Binkley, D.: Inter-Procedural Slicing Using Dependence Graphs. ACM Transactions on Programming Languages and Systems 12(1), 26–60 (1990)
3. Li, X., Wang, F., Wang, T., Wang, M.: A Novel Model for Automatic Test Data Generation Based on Predicate Slice. In: Proceedings of 2nd Int. Conf. on Artificial Intelligence, Management Science and Electronic Commerce, pp. 1803–1805 (2011)