Combinatorial Coverage Measurement Tool

User Guide

December 6, 2012

Text derived from NIST SP 800-142, Practical Combinatorial Testing, Oct. 2010.

Contact: Rick Kuhn; kuhn@nist.gov

Combinatorial Coverage Measurement

Since it is nearly always intractable to test all possible combinations, combinatorial testing is a reasonable alternative. For some value of *t*, testing all *t*-way interactions among *n* parameters will detect nearly all errors. It is possible that *t* = *n*, but reviewing the empirical data on failures, we would expect *t* to be relatively small. Determining the level of input or configuration state space coverage can help in understanding the degree of risk that remains after testing. If 90% - 100% of the state space has been covered, then presumably the risk is small, but if coverage is much smaller, then the risk may be substantial. This tutorial describes some measures of combinatorial coverage that can be helpful in estimating this risk.

Even in the absence of knowledge about a program's inner structure, we can apply combinatorial methods to produce precise and useful measures. In this case, we measure the state space of inputs. Suppose we have a program that accepts two inputs, *x* and *y*, with 10 values each. Then the input state space consists of the 10^2 = 100 pairs of *x* and *y* values, which can be pictured as a checkerboard square of 10 rows by 10 columns. With three inputs, *x*, *y*, and *z*, we would have a cube with 10^3 = 1,000 points in its input state space. Extending the example to *n* inputs we would have a (hard to visualize) hypercube of *n* dimensions with 10^n points. Exhaustive testing would require inputs of all 10^n combinations, but combinatorial testing could be used to reduce the size of the test set.

How should state space coverage be measured? Looking closely at the nature of combinatorial testing leads to several measures that are useful. We begin by introducing what will be called a *variable-value configuration*.

Definition. For a set of *t* variables, a variable-value configuration is a set of *t* valid values, one for each of the variables.

Example. Given four binary variables, *a, b, c*, and *d*, *a*=0, *c*=1, *d*=0 is a variable-value configuration, and *a*=1, *c*=1, *d*=0 is a different variable-value configuration for the same three variables *a, c*, and *d*.

Simple *t* -way combination coverage

Of the total number of *t*-way combinations for a given collection of variables, what percentage will be covered by the test set? If the test set is a covering array, then coverage is 100%, by definition, but many test sets not based on covering arrays may still provide significant *t*-way coverage. If the test set is large, but not designed as a covering array, it is very possible that it provides 2-way coverage or better. For example, random input generation may have been used to produce the tests, and good branch or condition coverage may have been achieved. In addition to the structural coverage figure, for software assurance it would be helpful to know what percentage of 2-way, 3-way, etc. coverage has been obtained.

Definition: For a given test set for *n* variables, simple *t*-way combination coverage is the proportion of *t*-way combinations of *n* variables for which all variable-values configurations are fully covered.

Example. Figure 1 shows an example with four binary variables, *a*, *b*, *c*, and *d*, where each row represents a test. Of the six 2-way combinations, *ab, ac, ad, bc, bd, cd*, only *bd* and *cd* have all four binary values covered, so simple 2-way coverage for the four tests in Figure 1 is 1/3 = 33.3%. There are four 3-way combinations, *abc, abd, acd, bcd*, each with eight possible configurations: 000, 001, 010,

011, 100, 101, 110, 111. Of the four combinations, none has all eight configurations covered, so simple 3-way coverage for this test set is 0%.

a	b	c	d
0	0	0	0
0	1	1	0
1	0	0	1
0	1	1	1

Figure 1. An example test array for a system with four binary components

(*t* + *k*)-way combination coverage

A test set that provides full combinatorial coverage for *t*-way combinations will also provide some degree of coverage for (*t*+1)-way combinations, (*t*+2)-way combinations, etc. This statistic may be useful for comparing two combinatorial test sets. For example, different algorithms may be used to generate 3-way covering arrays. They both achieve 100% 3-way coverage, but if one provides better 4-way and 5-way coverage, then it can be considered to provide more software testing assurance.

Definition. For a given test set for *n* variables, (*t+k*)-way combination coverage is the proportion of (*t+k*)-way combinations of *n* variables for which all variable-values configurations are fully covered. (Note that this measure would normally be applied only to a *t*-way covering array, as a measure of coverage beyond *t*).

Example. If the test set in Figure 1 is extended as shown in Figure 2, we can extend 3-way coverage. For this test set, *bcd* is covered, out of the four 3-way combinations, so 2-way coverage is 100%, and (2+1)-way = 3-way coverage is 25%.

a	b	c	d
0	0	0	0
0	1	1	0
1	0	0	1
0	1	1	1
1	1	0	1
1	0	1	1
1	0	1	0
0	1	0	0

Figure 2. Eight tests for four binary variables.

Variable-Value Configuration coverage

So far we have only considered measures of the proportion of combinations for which all configurations of *t* variables are fully covered. But when *t* variables with *v* values each are considered, each *t*-tuple has v^t configurations. For example, in pairwise (2-way) coverage of binary variables, every 2-way combination has $2^2 = 4$ configurations: 00, 01, 10, 11. We can define two measures with respect to configurations:

Definition. For a given combination of *t* variables, variable-value configuration coverage is the proportion of variable-value configurations that are covered.

Definition. For a given set of *n* variables, (*p, t*)-completeness is the proportion of the C(*n, t*) combinations that have configuration coverage of at least *p*.

Example. For Figure 1 above, there are C(4, 2) = 6 possible variable combinations and $C(4,2)2^2 = 24$ possible variable-value configurations. Of these, 19 variable-value configurations are covered and the only ones missing are *ab*=11, *ac*=11, *ad*=10, *bc*=01, *bc*=10. But only two, *bd* and *cd*, are covered with all 4 value pairs. So for the basic definition of simple *t*-way coverage, we have only 33% (2/6) coverage, but 79% (19/24) for the configuration coverage metric. For a better understanding of this test set, we can compute the configuration coverage for each of the six variable combinations, as shown in Figure 3. So for this test set, one of the combinations (*bc*) is covered at the 50% level, three (*ab*, *ac*, *ad*) are covered at the 75% level, and two (*bd*, *cd*) are covered at the 100% level. And, as noted above, for the whole set of tests, 79% of variable-value configurations are covered. All 2-way combinations have at least 50% configuration coverage, so (.50, 2)-completeness for this set of tests is 100%.

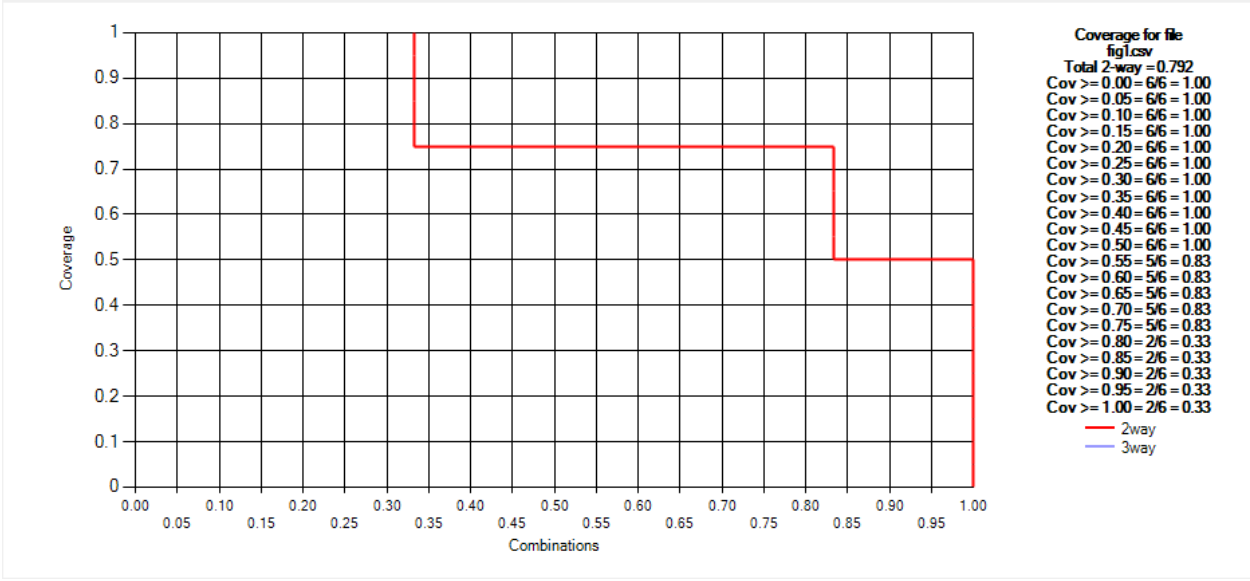
Although the example in Figure 1 uses variables with the same number of values, this is not essential for the measurement. Later sections of this document show how to use the tool for computing coverage of test sets where variables have different numbers of parameters.

Vars	Configurations	Config
a b	00, 01, 10	.75
a c	00, 01, 10	.75
a d	00, 01, 11	.75
b c	00, 11	.50
b d	00, 01, 10, 11	1.0
c d	00, 01, 10, 11	1.0

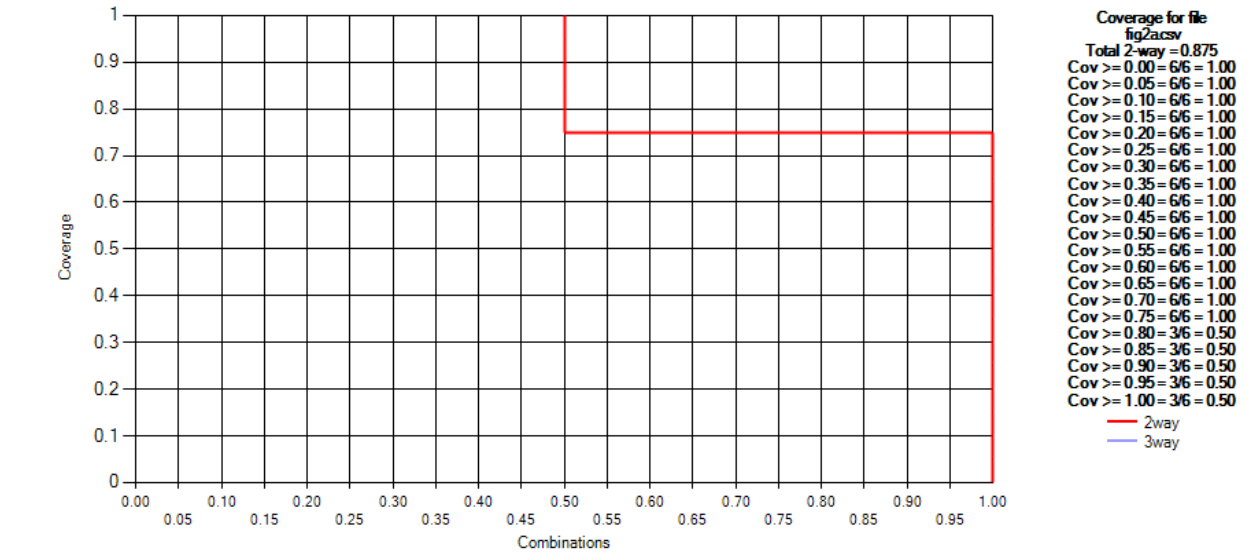
- *total 2-way cove*rage = 19/24 = .79167
- *(.50, 2)-completeness* = 6/6 = 1.0
- *(.75, 2)-completeness* = 5/6 = 0.83333
- *(1.0, 2)-completeness* = 2/6 = 0.33333

Figure 3. The test array covers all possible 2-way combinations of a, b, c, and d to different levels.

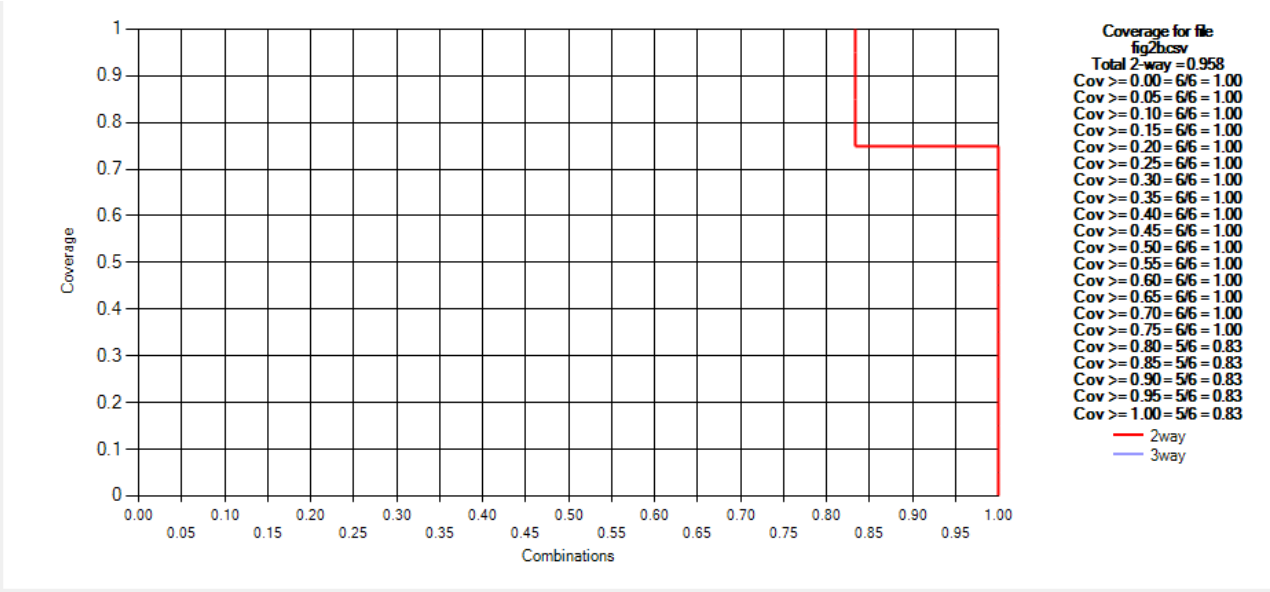
The graph below shows a graphical display of the coverage data for the tests in Figure 1. Coverage is given as the Y axis, with the percentage of combinations reaching a particular coverage level as the X axis. Note from Fig. 1 that 100% of the combinations are covered to at least the .50 level, 83% are covered to the .75 level or higher, and a third covered 100%. Thus the rightmost horizontal line on the graph corresponds to the smallest coverage value from the test set, in this case 50%.



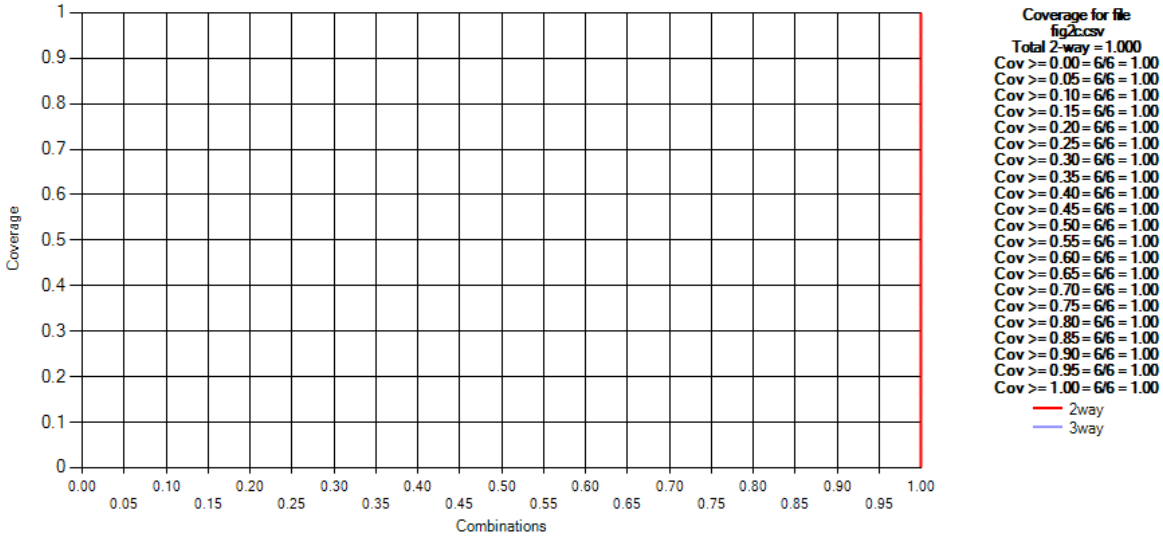
Additional tests can be added to provide greater coverage. Suppose an additonal test [1,1,0,1] is added. Coverage then increases as shown below. Now we can see immediately that all combinations are covered to at least the 75% level, as shown by the rightmost horizontal line (in this case there is only one horizontal line). The lefmost vertical line reaches the 1.0 level of coverage, showing that 50% of combinations are covered to the 100% level.



Adding another test, [1,0,1,1] results in the coverage shown below. -



If a test [1,0,1,0] is then added, 100% coverage for all combinations is reached: -



The graph can thus be thought of as a “coverage strength meter”, where the red indicator line moves to the right as higher coverage levels are reached. A covering array, which by definition covers 100% of variable-value configurations will always produce a figure as above, with full coverage for all combinations.

Using Coverage Measurement Tool

The tool implements the coverage measures documented in Chapter 6 of NIST SP 800-142.

Input: The input file must be a comma-separated values (CSV) format file where each row represents a test and each column represents a parameter. CSV files can be exported from any spreadsheet program. Constraints can be specified at the beginning of the file, each line will be considered a separated constraint. The image below shows an example of an input file with constraints:

	A	B	C	D	E
1	P1 = "Linux" => (P2 != "Explorer")				
2	P5="MySQL"=>(P1="Windows")				
3	Apple OS	Explorer	Ipv4	AMD	MySQL
4	Apple OS	Explorer	Ipv4	AMD	Oracle
5	Apple OS	Explorer	Ipv4	AMD	Sybase
6	Apple OS	Explorer	Ipv4	Intel	MySQL
7	Apple OS	Explorer	Ipv4	Intel	Sybase
8	Apple OS	Explorer	Ipv6	AMD	Oracle
9	Linux	FireFox	Ipv4	Intel	Oracle
10	Linux	FireFox	Ipv4	Intel	Sybase
11	Linux	FireFox	Ipv6	AMD	MySQL
12	Linux	FireFox	Ipv6	AMD	Oracle
13	Linux	FireFox	Ipv6	AMD	Sybase

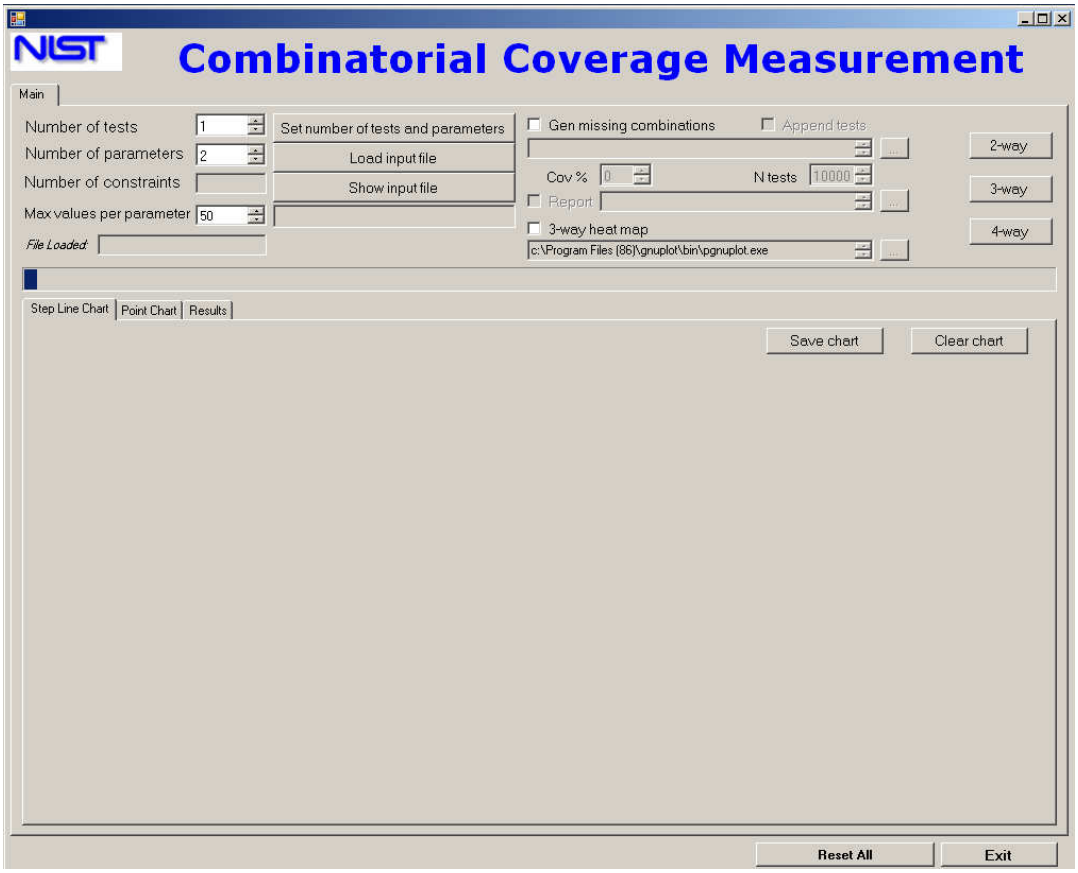
The first two lines are the constraints; and the following are the tests, each column correspond to a parameter.

Charts: Charts display coverage measurements as documented in Chapter 6 of SP 800-142. Charts may be saved to an image file using “Save chart”.

Sections: The tool has these four sections: main, parameters, constraints and Invalid Combinations they are described below.

Main

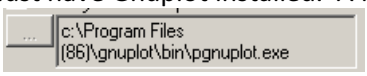
This screen contains the main controls to load the file containing the set test; it will show the result of the measurement and charts.



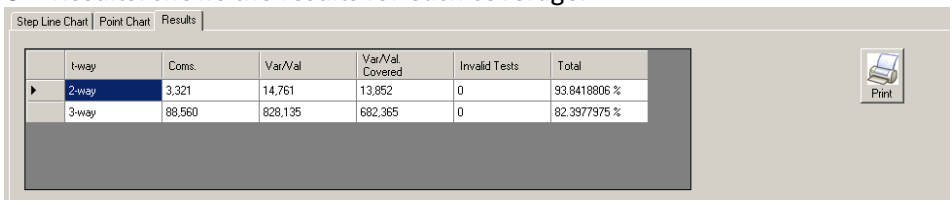
If all tests and parameters in the input file should be loaded, just click on "Load input file" and select the test file to be analyzed. If just some tests or parameters are needed, before loading the input file, the number of tests and parameters should be specified using the numeric fields above on the left and pressing "Set number of tests and parameters". Note that if all tests are to be loaded, it is not necessary to set the number of tests and parameters, provided that parameters have values (no more than as indicated by the "Max values per parameter" field). The tool will read in all tests and discover parameter values that will be used in the measurement process. The tool can also process continuous-valued parameters such as account balances, distances, or others with a large range of possible values. See the "Specifying boundaries" discussion below.
After the file loads, coverage measures may be computed by clicking on the appropriate button.

Features and options

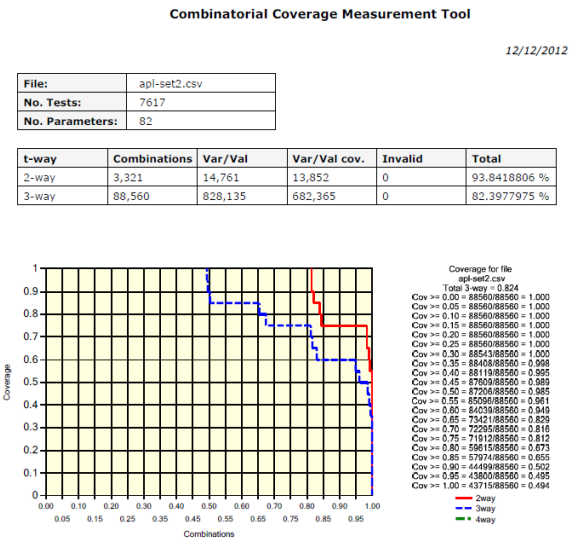
- *Number of tests:* to set number of rows (tests) in input file, if known.
- *Number of parameters*: this must also be set by the user when *Number of tests* is set.
- Number of constraints: Show the number of constraints specified.
- Max values per parameter: to set the max values per parameter.
- File loaded: show the name of the input file.
- *Set number of tests and parameters* – click this button after setting the number of tests and parameters.

- *Load input file*: opens a dialog for the user to select a file; after loading, the number of tests and parameters are displayed in the box below this control.
- *Show input file:* displays the first 10 lines of the file after loading.
- Gen missing combinations: Generates the missing needed combinations to reach the coverage specified in "Cov %". When this option is checked the output name file is requested and it'll be shown in the textbox below.
- Append tests: the output file for missing combinations will contain the already specified combinations besides the missing ones.
- Cov % : coverture to be reached with the missing combinations.
- N Tests: number of missing combinations requested.
- Report: reports, in a file, the missing combinations.
- 3-way heat map: To use this option the system must have Gnuplot installed. A file path browse button is provided to specify the installation path: 

- *Compute* 2-*way*, 3-*way and 4-way coverage*: calculates coverage.
- *Tabs:*
  - Step line chart: shows a step line chart for each coverage calculated.
    - *Clear chart*: removes any currently-displayed graph.
    - *Save chart*: opens a dialog box to allow user to save the displayed graph as an image file.
  - Point chart: shows a point chart for 2-way coverage.
  - Results: shows the results for each coverage.



Contains the option to print out the results, the next image is an example:



Combinatorial Coverage Measurement Tool

12/12/2012

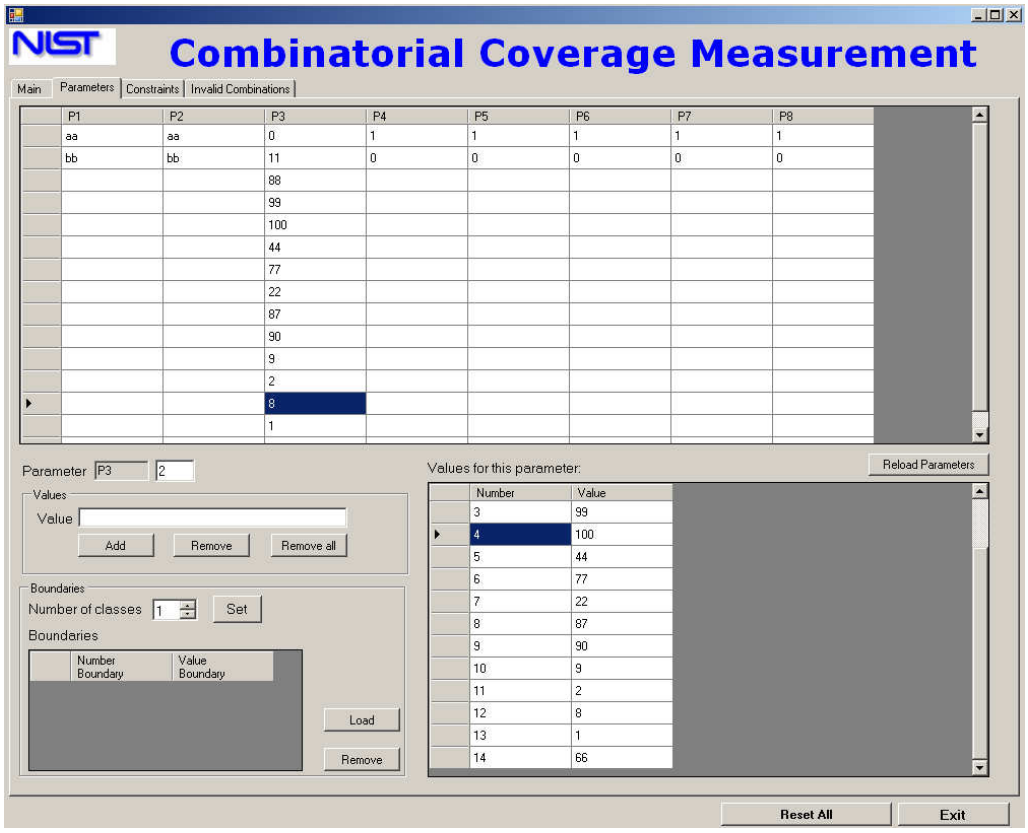
File:	apl-set2.csv
No. Tests:	7617
No. Parameters:	82

t-way	Combinations	Var/Val	Var/Val cov.	Invalid	Total
2-way	3,321	14,761	13,852	0	93.8418806 %
3-way	88,560	828,135	682,365	0	82.3977975 %

- *Reset all*: resets all the features to load a new input file.
- *Exit*: terminates the program.

Parameters

The parameters and their values are shown; they can be modified specifying boundaries, and adding or removing values. Clicking in a column will show the values below in order to be modified.



Features and options

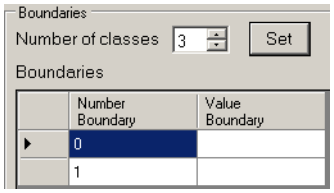
- *Value*: Specified a value to be added.
- *Add*: The value above will be added to the parameter, it is shown in the right.
- *Remove*: Removes a value of the parameter, the value has to be selected in the right
- *Remove* all: remove all the parameter specified for the parameter.

To reverse changes, values added can be removed and vice versa.

Specifying boundaries

For continuous-valued parameters, specify equivalence classes by indicating the number of value classes and boundaries between the classes. Boundaries may include decimal values. Where the boundary between two classes c_1 and c_2 is x, the system places input values $< x$ into c_1 and values $\geq x$ in c_2.

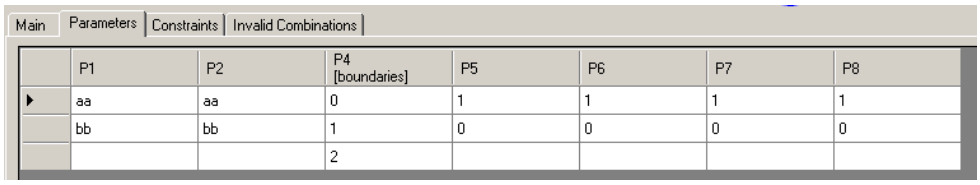
- *Number of classes*: Specify the number of classes, then click “Set” and the equivalence class boundaries will be shown below.



- *Value boundary column*: The value boundary has to be specified.
- *Load*: the value boundaries will be specified to the parameter and they will be shown in the right.

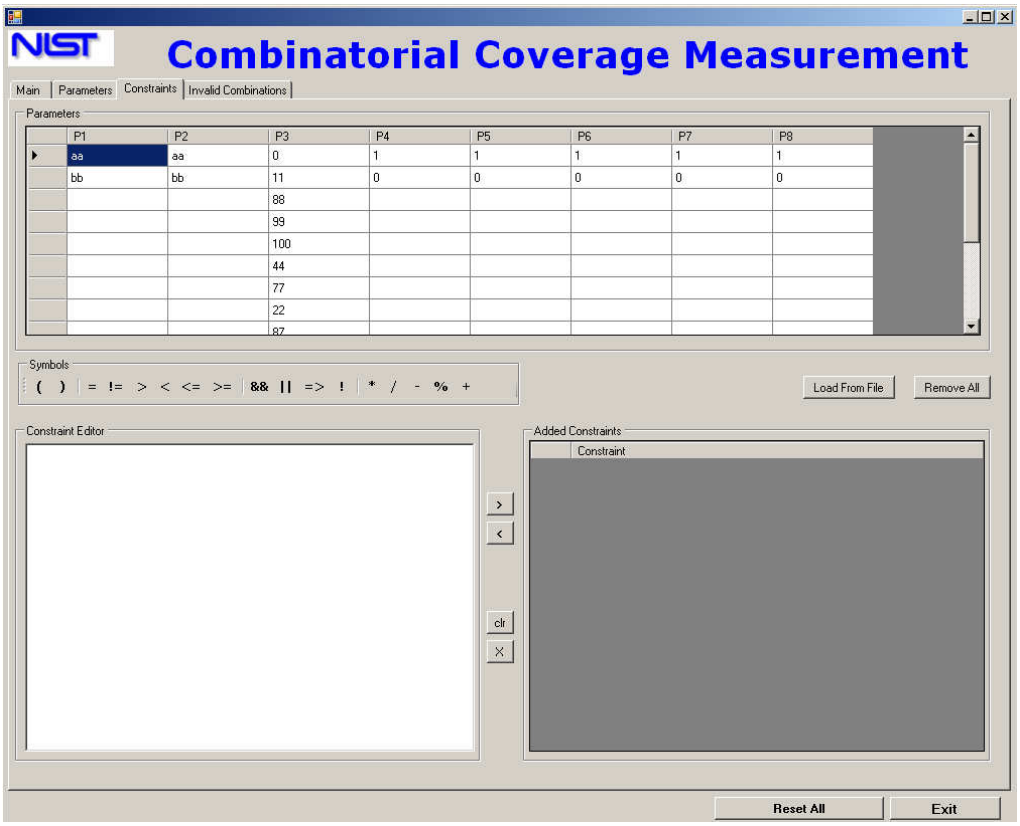
	Number	Value	ValueBoundary	ReferenceBound
	0	0	0	0 <= 25
	1	11	0	11 <= 25
	2	88	2	88 >= 51
	3	99	2	99 >= 51
▶	4	100	2	100 >= 51
	5	44	1	26 <= 44<=50
	6	77	2	77 >= 51
	7	22	0	22 <= 25
	8	87	2	87 >= 51
	9	90	2	90 >= 51
	10	9	0	9 <= 25
	11	2	0	2 <= 25

- *Remove*: remove the boundaries specified, returning the real values to the parameter.
- *Reload Parameters*: The parameters are reloaded in the table above and in the constraints sections in order new values or boundaries are showed.



Constraints

The constraints are shown, if no constraints are specified, they can be added in this section.



Below is the formal syntax of the expressions that can be used to specify a constraint:

<Constraint>::=<Simple_Constraint>|<Constraint><Boolean_Op><Constraint> <Simple_Constraint> ::=
<Term><Relational_Op><Term> <Term>:=<Parameter>|<Parameter><Arithmetic_Op><Parameter>|
<Parameter><Arithmetic_Op> <Value> <Boolean_Op>:="&&" |"||" |"=>"
<Relational_Op> := "=" |"!=" | ">" | "<" | ">=" | "<=" <Arithmetic_Op> := "+" | "-" | "" | "/" | "%"*

There are three types of operators:

1. - Boolean operators (Boolean_Op), including &&, ||, =>
2. - Relational operators (Relational_Op), including =, !=, >, <, >=, <=
3. - Arithmetic operators (Arithmetic_OP), including +, -, *, /, %. Note that arithmetic operators can appear in a term expression (<Term>) only if the parameters involved in the term expression are of type Number or Range. Also, four of the relational operators, namely, >, <, >=, <=, can appear in a simple constraint expression (*Simple_Constraint*) only if both of the terms involved in the simple constraint are evaluated to a parameter value of type Number or Range.

The following are examples of various constraints that can be specified:

Constraint 1: (P1 = “Windows”) => (P2 = “IE” || P2 = “FireFox” || P2 = “Netscape”), where P1 is a parameter for OS and P2 is a parameter for Browser. This constraint specifies that if OS is Windows, then Browser has to be IE, FireFox, or Netscape.

Constraint 2: (P1 > 100) || (P2 > 100), where P1 and P2 are two parameters of type Number or Range. This constraint specifies that P1 or P2 must be greater than 100.

Constraint 3: (P1 > P2) => (P3 > P4), where P1, P2, P3, and P4 are parameters of type Number or Range. This constraint specifies that if P1 is greater than P2, then P3 must be greater than P4.

All the parameters and their values are shown above.

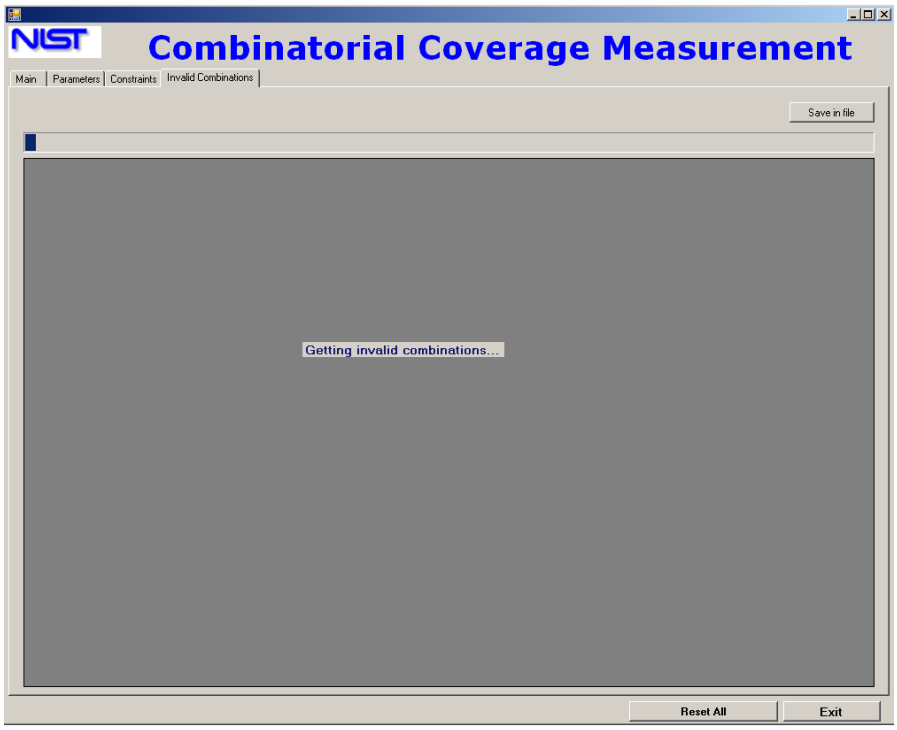
Features and options

- *Set parameter*: Double click on column header.
- *Set value-parameter*: Double click on cell.
- *Set operator*: Click it in the tool bar

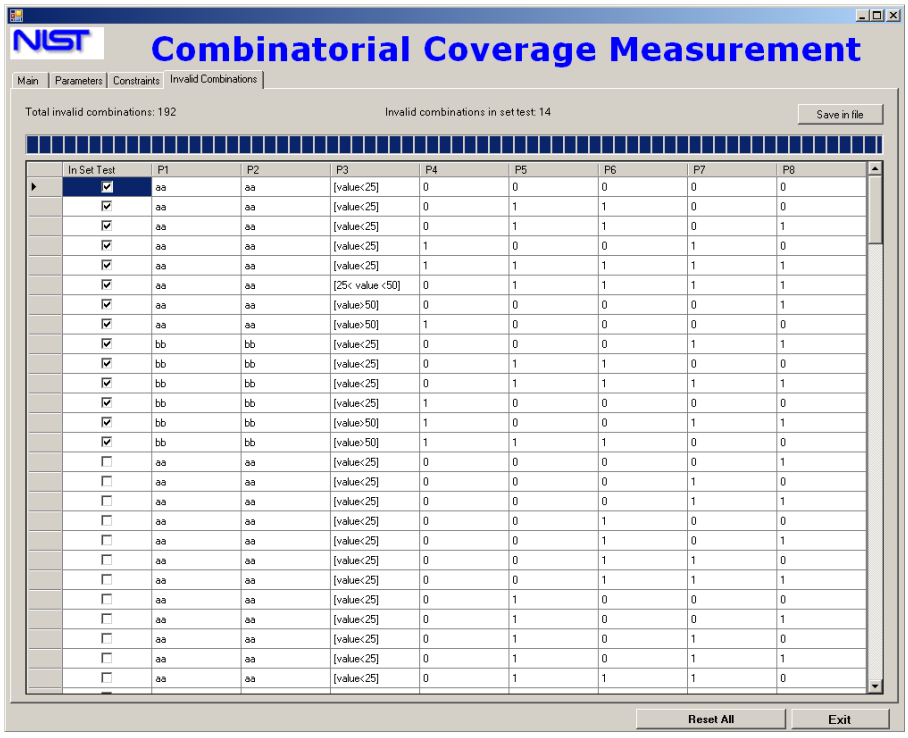
- *Add a constraint*: Click “>” button.
- *Edit a constraint*: Click “<” button. The constraint is showed in the edit box. The constraint is removed until the user press “>” button.
- *Clr*: This option cleans the edit box.
- *X*: Remove the selected constraint.
- *Remove all*: remove all the constraints specified.
- *Load from file*: Load constrains in a file:

Invalid Combinations

The invalid combinations will be shown if there are constraints specified. If any coverage measurement has been specified the invalid combinations will be generated.



When all the invalid combinations have been generated they will be shown -



In Set Test	P1	P2	P3	P4	P5	P6	P7	P8
☑	aa	aa	[value<25]	0	0	0	0	0
☑	aa	aa	[value<25]	0	1	1	0	0
☑	aa	aa	[value<25]	0	1	1	0	1
☑	aa	aa	[value<25]	1	0	0	1	0
☑	aa	aa	[value<25]	1	1	1	1	1
☑	aa	aa	[25< value <50]	0	1	1	1	1
☑	aa	aa	[value>50]	0	0	0	0	1
☑	aa	aa	[value>50]	1	0	0	0	0
☑	bb	bb	[value<25]	0	0	0	1	1
☑	bb	bb	[value<25]	0	1	1	0	0
☑	bb	bb	[value<25]	0	1	1	1	1
☑	bb	bb	[value<25]	1	0	0	0	0
☑	bb	bb	[value>50]	1	0	0	1	1
☑	bb	bb	[value>50]	1	1	1	0	0
☐	aa	aa	[value<25]	0	0	0	0	1
☐	aa	aa	[value<25]	0	0	0	1	0
☐	aa	aa	[value<25]	0	0	0	1	1
☐	aa	aa	[value<25]	0	0	1	0	0
☐	aa	aa	[value<25]	0	0	1	0	1
☐	aa	aa	[value<25]	0	0	1	1	0
☐	aa	aa	[value<25]	0	0	1	1	1
☐	aa	aa	[value<25]	0	1	0	0	0
☐	aa	aa	[value<25]	0	1	0	0	1
☐	aa	aa	[value<25]	0	1	0	1	0
☐	aa	aa	[value<25]	0	1	0	1	1
☐	aa	aa	[value<25]	0	1	1	1	0

The left marked combinations are the ones in the set test.

Features and options

- *Total invalid combinations*: Number of all invalid combinations according to the constraints specified and parameters in the set test.
- *Invalid combinations in set test*: Number of invalid combinations in set test.
- *Save in file*: Save the invalid combinations found in the set test.