CTWeb
User's manual
May 21, 2012

Macario Polo Usaola, Beatriz Pérez Lamancha, Pedro Reales Mateo
Alarcos Research Group
Institute of Technologies and Information Systems
University of Castilla-La Mancha
Paseo de la Universidad, 4
13071-Ciudad Real (Spain)
http://alarcos.esi.uclm.es
Contact person: macario.polo@uclm.es

1 Introduction

CTWeb is a web application for generating test cases. It includes two tools:

- A combinatorial tool, that gets test cases by applying several combinatorial strategies.
- A state machine tool, that generates test cases from textual specifications of state machines.

Currently, the use of the application is completely open and free, although (for some functionalities) we plan to include a pay-per-use for companies, leaving it free for students and researchers.

2 The combinatorial tool

Figure 1 shows the main screen of the combinatorial tool: on the left side it lists the algorithms implemented by the tool (by clicking on the algorithm's name, the user gets an explanation of it); the tester uses the right-hand side to specify the parameters and values of the system or functionality under test.

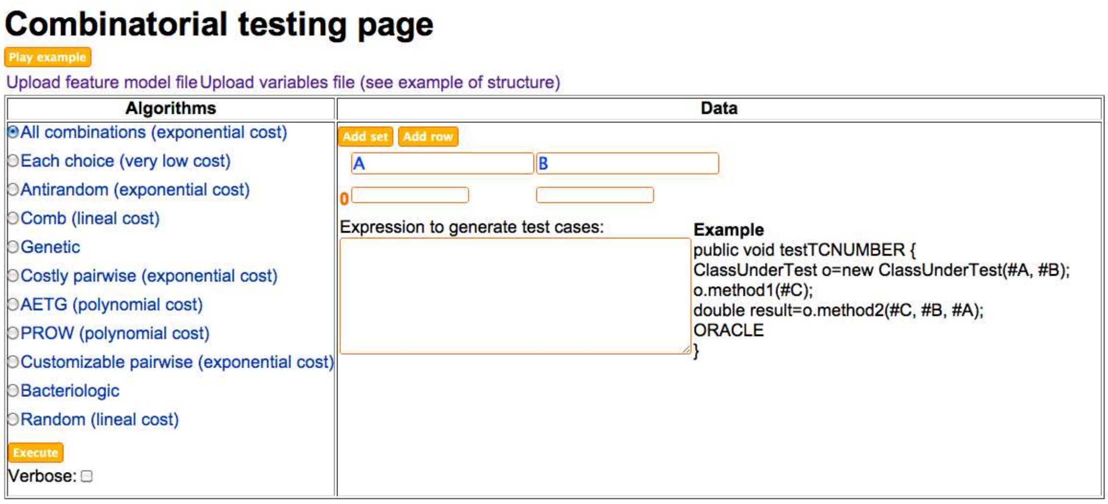


Figure 1

2.1 A simple example

Let us suppose we want to test a function that converts temperature measures whose signature is:

convert(sourceUnit : String, targetUnit : String, magnitude : double) : double

It translates the numeric *magnitude* passed as third parameter from the *source unit* to the *target unit*, respectively passed as first and second parameters. The conversion functions from Celsius to Kelvin and Fahrenheit are:

$$K = C + 273 \qquad F = \frac{9}{5} \times C + 32$$

Supposing *c* is an instance of the container class (let it be *Converter*), some possible calls to the function under test could be:

c.convert("C", "K", 0); c.convert("K", "C", 0); c.convert("K", "F", -200);

If you remind, the minimum possible temperature is the *absolute zero,* which corresponds to: *0ºK=-273ºC=-459.4ºF.* Thus, some test data to test this simple function could be those in Table 1:

Source units	Target units	Magnitude
C	C	0
F	F	-273
K	K	-273.01
Another	Another	-459.4
		-459.41
		100

Table 1

With CTWeb it is very easy to generate data combinations to test this function:

1) First of all, as we have three parameters, we press the *Add set* button to add a new set of test data values. Then, the tool inserts a new column in the right:

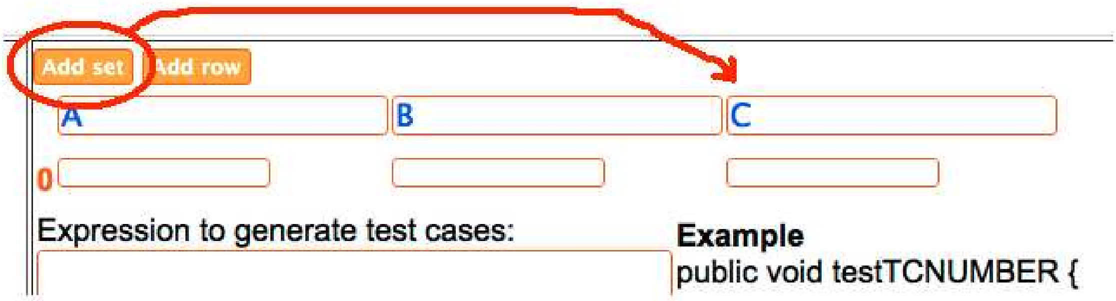


Figure 2

2) We continue filling-in the row with the test data. As we need to add rows, we press the *Add row* button in Figure 2 so many times as we need. The screen will look as in Figure 3.

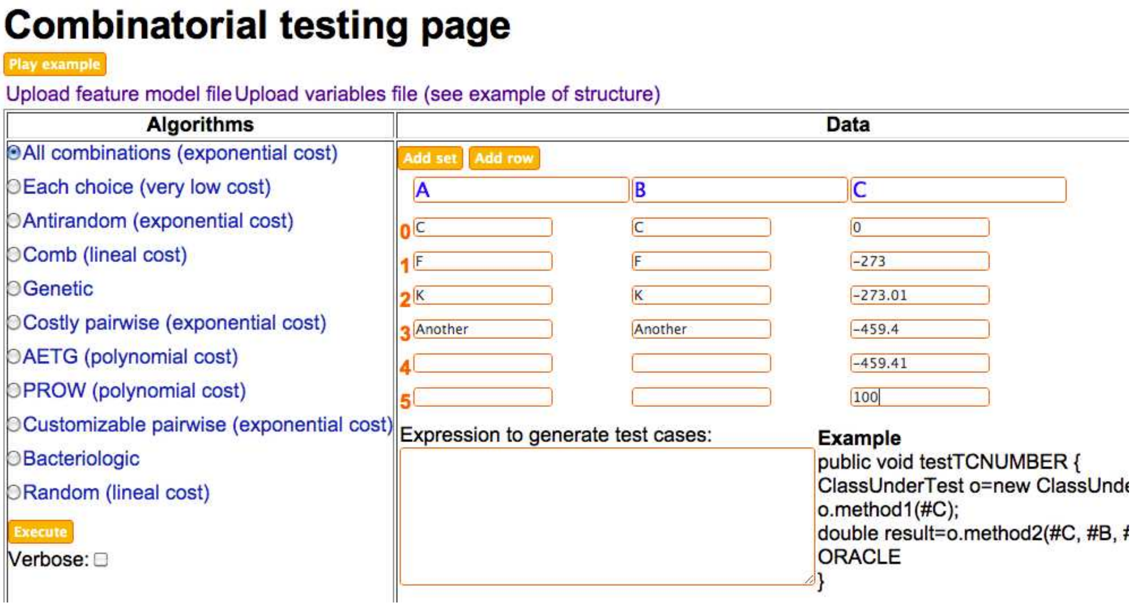
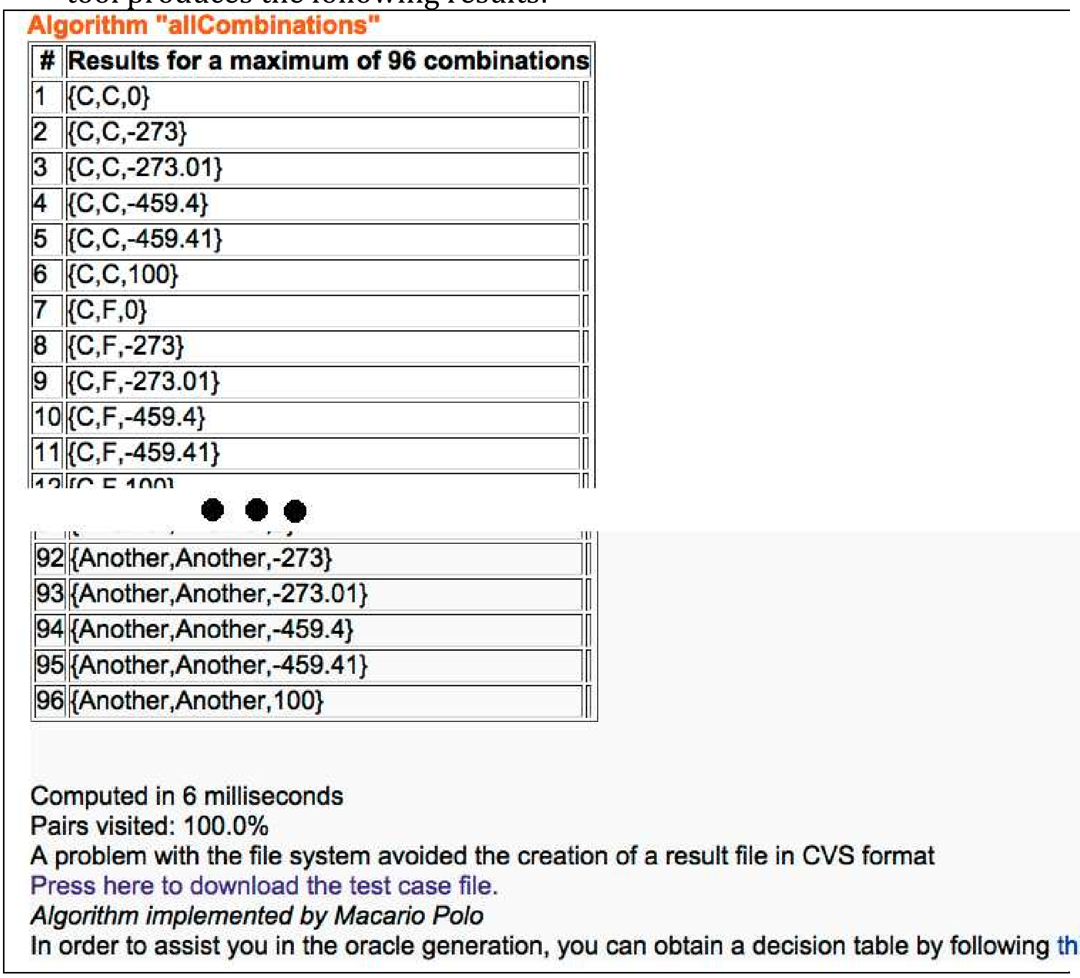


Figure 3

3) Now, can generate the test data combinations by selecting the desired algorithm (left side) and pressing the *Execute* button, beneath the left side. If we leave selected the *All combinations* algorithm and press *Execute*, the tool produces the following results:



Algorithm "allCombinations"

#	Results for a maximum of 96 combinations
1	{C,C,0}
2	{C,C,-273}
3	{C,C,-273.01}
4	{C,C,-459.4}
5	{C,C,-459.41}
6	{C,C,100}
7	{C,F,0}
8	{C,F,-273}
9	{C,F,-273.01}
10	{C,F,-459.4}
11	{C,F,-459.41}
12	{C,F,100}
● ● ●	
92	{Another,Another,-273}
93	{Another,Another,-273.01}
94	{Another,Another,-459.4}
95	{Another,Another,-459.41}
96	{Another,Another,100}

Computed in 6 milliseconds
Pairs visited: 100.0%
A problem with the file system avoided the creation of a result file in CVS format
Press here to download the test case file.
Algorithm implemented by Macario Polo
In order to assist you in the oracle generation, you can obtain a decision table by following thi

Figure 4

As there are 4, 4 and 6 values in the three sets, the *All combinations* algorithms produces 4·4·6=96 test data combinations. After the results table, we get some information regarding the computation time and the percentage of pairs of data valuesvisited by test cases.

One of the problems of *All combinations* is, on the one side, its high computational cost (exponential) and, on the other side, the high number of test cases it produces. To deal with this, we can use any of the other algorithms provided by CTWeb: AETG, for example, has a polynomial cost and produces a test suite visiting all pairs, but whose size is much more small (Figure 5): 25 test cases in this example.

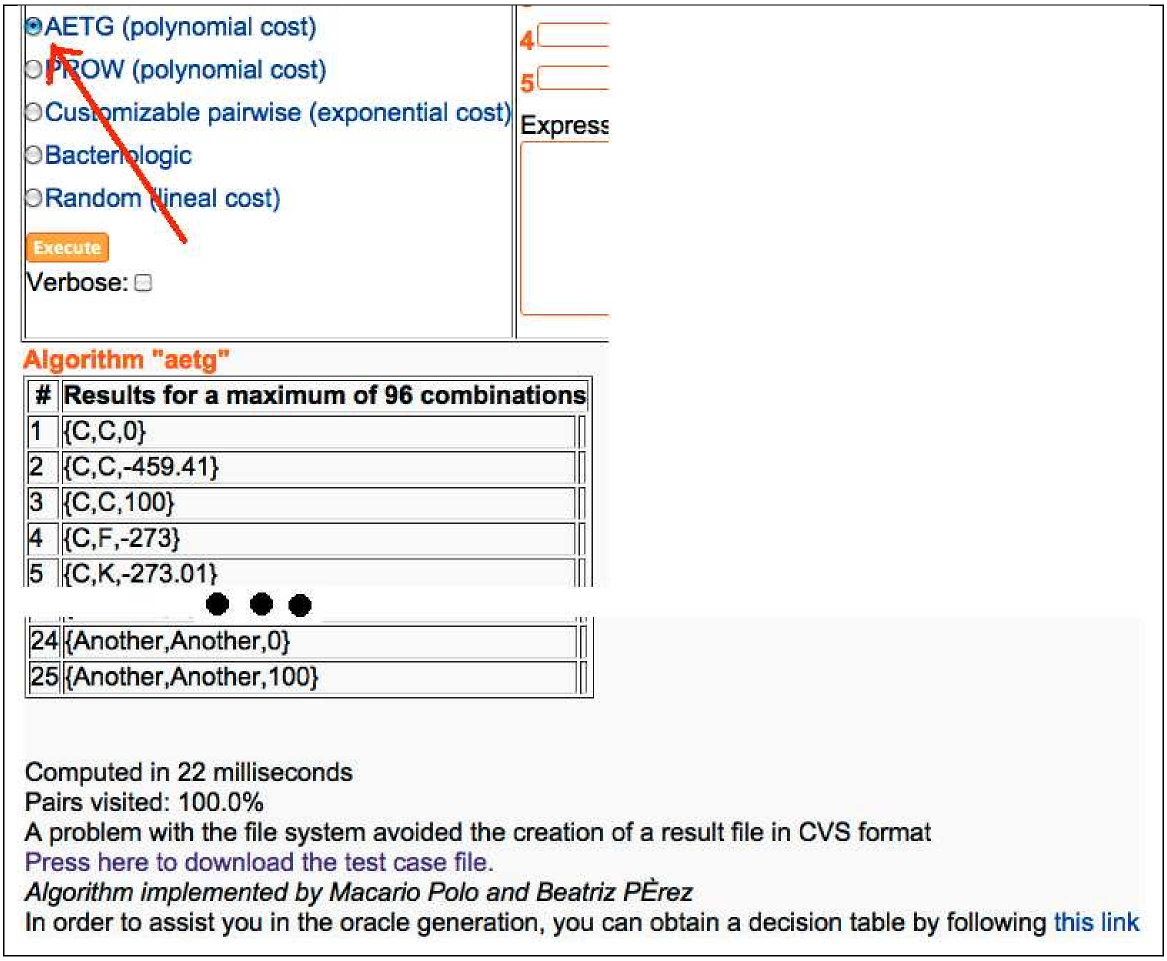


Figure 5

Suppose now that we want to use the test data combinations generated in a set of test cases as those we have written as example:

c.convert(“C”, “K”, 0); c.convert(“K”, “C”, 0); c.convert(“K”, “F”, -200);

For this, we can write a template at the text arealabeled *Expression to generate test cases.* Suppose we want that our test cases have this aspect:

```
public void test1() {
    Converter c=new Converter();
    c.convert(“C”, “K”, 100);
}
```

Figure 6

To generate test cases like that in Figure 6, we can write, in the afore mentioned test area, an expression like this one:

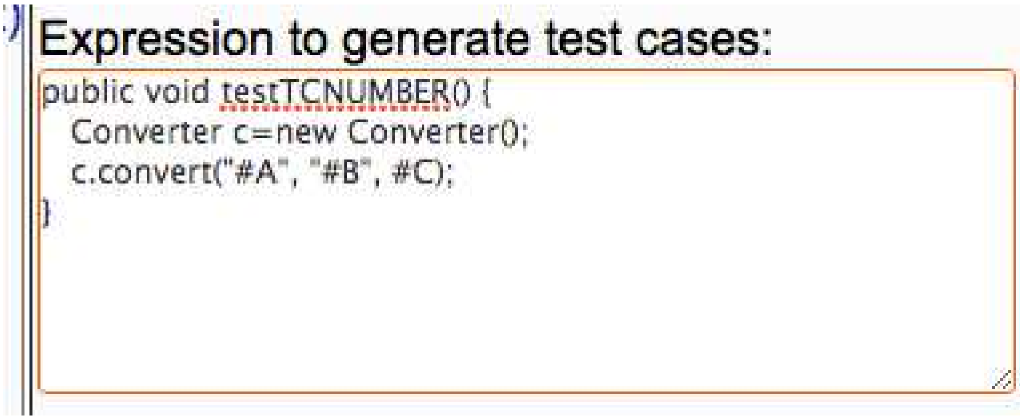


Figure 7

Now, when CTWeb generates the test data combinations, will substitute the *TCNUMBER* token by the actual index of the combination, and the tokens *#A, #B* and *#C* by the values of the first, second and third parameters in the current combination. In other words, the results table will look such as that in Figure 8: note that, now, the code corresponding to the translation of the combination values has been added into the third column.

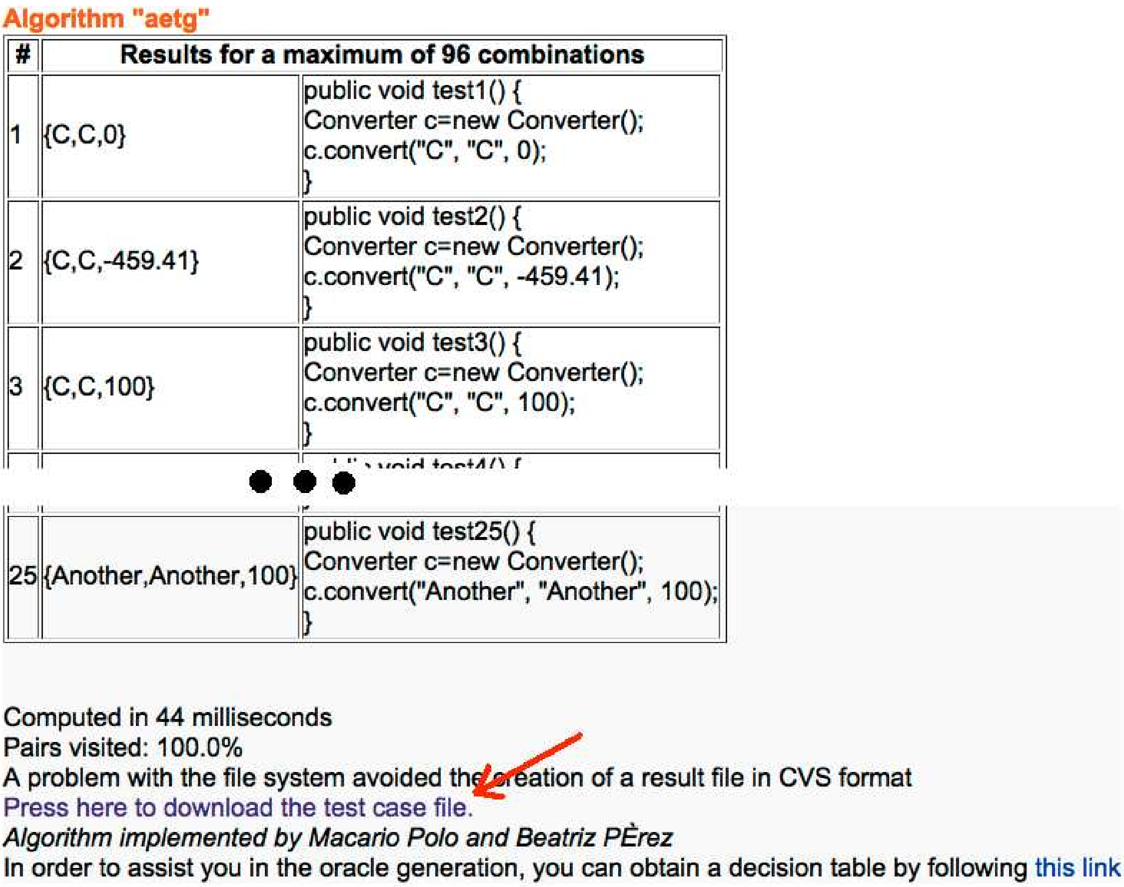


Figure 8

See also, in Figure 8, the link highlighted with a red arrow: if you press it, a new window with all the test cases generated is open: you can copy and paste it to work with it:

```
public void test1() {
Converter c=new Converter();
c.convert("C", "C", 0);
}


public void test2() {
Converter c=new Converter();
c.convert("C", "C", -459.41);
}


public void test3() {
Converter c=new Converter();
```

Figure 9

2.2 Uploading the data from a test file

Instead of filling-in by hand the data area, we can upload a variables file that, moreover, can be enriched with more information to generate the tests. The following (Figure 10) is a possible text file to test the *convert* function. Note it has several sections:

1) In the **%Sets** section we add a line for each variable or set. After the variable's name there is a tab, and also a tab after each variable value.
2) In the **%Includes** section we add those combinations that we want to include always in the test suite, writing their values with a comma between each two values. To exemplify, we have added the test cases *C, K, -273.01* and *F, K, -459.41.*
3) Then, there are several **%Oracle** sections. Each oracle may have a description (tab-separated from the **%Oracle** keyword). In the following lines we specify (also tab-separated), the values of the variables for which the oracle expression (which appears in the last line) must be included in the test case.
    a. In the first **%Oracle**(described as *// Celsius or Kelvin under absolute zero*) there appear two values (C and K) for the *SOURCE* variable and three values (-273.01 and -459.41) for the *MAGNITUDE* variable. This means that this oracle is applicable to all those test cases whose *SOURCE* variable is *C* or *K* and whose *MAGNITUDE* is *-273.01 and -459.41:* this is, this oracle will be included in all the test cases that try to convert *-273.01ºC, -273.01ºK, -459.41ºC* or *-459.41ºK*. Moreover, the oracle expression for these test cases is that appearing after a tab after the **oracle** keyword: *assertTrue(result==Integer.MIN_VALUE);.* The *convert* function returns -∞ when the conversion is erroneous, value that is represented as *Integer.MIN_VALUE.*Note that, in the **oracle** line, references to variables values are preceded by a # symbol.

b. The second **%Oracle** is slightly different: it involves the three sets (*SOURCE, TARGET* and *MAGNITUDE*) but, moreover, its last line has the keyword **conditionalOracle** in its last line. Conditional oracles have two parameters tab-separated: the first one is a condition (expressed in Java language, because this is the programming language in which CTWeb is implemented) that, when it is true, says the tool that the expression included as second parameter must be added to the test case. In this example, the condition says that, when the *SOURCE* variable is equals to the *TARGET* (note, moreover, that the values are restricted to *C, F* and *K),* the oracle expression (*assertTrue(result==#MAGNITUDE);*) must be added to the test case (note also that the values of *MAGNITUDE* are 0, -273, -459.4 and 100).Note here that, in the **conditionalOracle** line, references to variables values are also preceded by a # symbol.

4) In the **% Test template** section, the tester writes the template used to generate the test cases. Note this section finishes with **%%,** and note also the presence of the keyword **ORACLE**: in test case generation time, the tool will substitute this token by the corresponding oracle or oracles. A special detail of this section is the use of the first letters of the alphabet to do reference to the first, second, third… sets, according to the order they appear in the **%Sets** section.

```
%Sets
SOURCE	C	F	K	Another
TARGET	C	F	K	Another
MAGNITUDE	0	-273	-273.01	-459.4	-459.41	100

%Includes
C, K, -273.01
F, K, -459.41

%Oracle		// Celsius or Kelvin under absolute zero.
SOURCE	C	K
MAGNITUDE	-273.01	-459.41
oracle	assertTrue(result==Integer.MIN_VALUE);

%Oracle // Transformation between same units
SOURCE	C	F	K
TARGET	C	F	K
MAGNITUDE	0	-273	-459.4	100
conditionalOracle #SOURCE == #TARGET	assertTrue(result==#MAGNITUDE);

%Oracle		// Transformations FROM invalid units
SOURCE	Another
oracle	assertTrue(result==Integer.MIN_VALUE);

%Oracle		// Transformations TO invalid units
TARGET	Another
oracle	assertTrue(result==Integer.MIN_VALUE);

%Test template
public void testTCNUMBER() {
		Converter c = new Converter();
		double result = c.convert("#A", "#B", #C);
		ORACLE
}
%%
```

Figure 10

Figure 11shows some of the test cases generated with this text file:

1) Test case 1 corresponds to the conditional oracle, since it is a conversion from 0º Celsius to Celsius.
2) Test case 2 is also a conversion from Celsius to Celsius, but the value of MAGNITUDE does not match with the values in the MAGNITUDE values of the conditional oracle. This test case is a conversion from -459.41 Celsius degrees, which fits with the first **%Oracle** section and, therefore, its expressions is added.
3) The test data of test case number 4 doesn't fit with any oracle: the tool adds a comment line explaining this situation.
4) The test data in test case 20 fit with two oracles: a conversion to invalid units and a conversion under the absolute zero. Both oracle expressions are added to the test case, although also this situation is added in a comment line.

Algorithm "aetg"

#	Results for a maximum of 96 combinations	

1	{C,C,0}	public void test1() {
		Converter c = new Converter();
		double result = c.convert("C", "C", 0);
		
		// Transformation between same units
		assertTrue(result==0);
		}

2	{C,C,-459.41}	public void test2() {
		Converter c = new Converter();
		double result = c.convert("C", "C", -459.41);
		
		// Celsius or Kelvin under absolute zero.
		assertTrue(result==Integer.MIN_VALUE);
		}

4	{C,F,-273}	public void test4() {
		Converter c = new Converter();
		double result = c.convert("C", "F", -273);
		
		// Warning: this test case has no oracle assigned
		}

20	{K,Another,-273.01}	public void test20() {
		Converter c = new Converter();
		double result = c.convert("K", "Another", -273.01);
		
		// Celsius or Kelvin under absolute zero.
		assertTrue(result==Integer.MIN_VALUE);
		// Transformations TO invalid units
		assertTrue(result==Integer.MIN_VALUE);
		// Warning: more than one oracle for this test case
		}

Figure 11

2.3 A less simple example

Suppose now a new version of the *convert* function with the same signature than the previous one, but that is now capable of making more types of conversions: it may translate temperatures (Celsius, Fahrenheit and Kelvin: C, F, K), lengths (Meters, Yards, Inches, Kilometers and Miles: M, Y, I, KM, ML) and weights (Kilograms, Pounds and Ounces: K, P, O).

For testing this new version of the function, we should take into account the appropriate equations for conversions, as well as the invalid values for the function's parameters. Considering that one cannot convert between different types of units (from temperatures to lengths, for example), the different values of the absolute zero we have seen and that there are no negative lengths or weights, the following table shows a set of possible equivalence classes for these parameters:

	Temperature	Length	Weight
Equivalence classes	**From ºC**: (-∞, -273) [-273ºC, +∞) **From ºF**: (-∞, -459.4) [-459.4, +∞) **From ºK:** (-∞, 0) [0, +∞,)	(-∞, 0) [0, +∞)	(-∞, 0) [0, +∞)

Table 2

From the equivalence classes of Table 2, the tester must propose a set of test data, which could be those in Table 3.

Temperature		Length and weight	
Value	**Type of expected result**	**Value**	**Type of expected result**
From ºC: -300 -273.01 -273	 Error (value out of range) Error (value out of range) Error (value out of range)	-10 -0.1	Error (value out of range) Error (value out of range)
From ºF: -459.41 -459.4	 Error (value out of range) Error (value out of range)	0 10	Valid conversion Valid conversion
From ºK: -0.01	 Error (value out of range)		
From ºC, ºF, ºK: 0 100	 Valid conversion Valid conversion		

Table 3

Figure 12shows a text file for this new version of *convert*. Besides having more values in the variables definition and much more oracles, it also has two new sections:

1) We can write several **%Excludes** sections. Each one starts with the names of a pair of sets and, then, some lines with pairs of values of these sets that the tester does not desire to include in the test cases. In this example, we are saying CTWeb that we don't want test cases with conversions from Celsius to Kilograms, Pounds or Ounces.

```
%Sets
MAGNITUDE            -300      -273.01   -273      -459.41   -459.4    -0.01     0         100
SOURCE    C          F         K         M         Y         I         KM        ML        KG
          P          O
TARGET    C          F         K         M         Y         I         KM        ML        KG
          P          O

%Includes
-273, C, K
-459.4, F, K
0, K, C
0, K, F

%Excludes
SOURCE, TARGET
C, KG
C, P
C, O

%Weights
SOURCE, TARGET
C, F, 1
F, C, 1
KM, ML, 1
ML, KM, 1
KG, P, 1
P, KG, 1

%Oracle   // Celsius or Kelvin under absolute zero or negative length or weight.
MAGNITUDE            -300      -273.01   -459.41   -459.4
SOURCE    ANY
oracle    assertTrue(result==Integer.MIN_VALUE);

%Oracle // Transformation between same units
MAGNITUDE            0         100
SOURCE    ANY
TARGET    ANY
conditionalOracle    #SOURCE == #TARGET assertTrue(result==#MAGNITUDE);

%Oracle // Celsius, Kelvin or Fahrenheit under absolute zero or negative length or weight. The value is a F
temperature just below 0ºK
MAGNITUDE            -459.41
SOURCE    ANY
oracle    assertTrue(result==Integer.MIN_VALUE);

%Oracle              // A Kelvin temperature just under 0
MAGNITUDE            -0.01
SOURCE    K
oracle    assertTrue(result==Integer.MIN_VALUE);

%Oracle              // Conversions from temperatures to other units
SOURCE    C          F         K
TARGET    M          Y         I         KM        ML        KG        P         O
oracle    assertTrue(result==Integer.MIN_VALUE);

%Oracle              // Conversions from lengths to other units
SOURCE    M          Y         I         KM        ML
TARGET    C          F         K         KG        P         O
oracle    assertTrue(result==Integer.MIN_VALUE);

%Oracle              // Conversions from weights to other units
SOURCE    KG         P         O
TARGET    C          F         K         M         Y         I         KM        ML
oracle    assertTrue(result==Integer.MIN_VALUE);

%Oracle              // C to K
MAGNITUDE            0         100       -273
SOURCE    C
TARGET    K
oracle    assertTrue(result==#MAGNITUDE+273);

%Oracle              // From Celsius to Celsius
MAGNITUDE            0         100       -273
SOURCE    C
TARGET    C
oracle    assertTrue(result==#MAGNITUDE);


%Oracle   // Transformations from Km to Miles
MAGNITUDE            0         100
SOURCE    KM
TARGET    ML
oracle assertTrue(result==#MAGNITUDE/1609);

%Oracle   // Transformations from Miles to Km
MAGNITUDE            0         100
SOURCE    ML
TARGET    KM
oracle assertTrue(result==#MAGNITUDE*1609);

%Oracle   // Negative length or weights
MAGNITUDE            -300      -273.01   -273      -459.41   -459.4    -0.01
SOURCE    M          Y         I         KM        ML        KG        P         O
oracle assertTrue(result==Integer.MIN_VALUE);

%Test template
public void testTCNUMBER() {
          Converter c = new Converter();
          double result = c.convert("#B", "#C", #A);
          ORACLE
}
%%
```

Figure 12

2) The **%Weights** section is used to assign an importance to certain pairs of values. As you know, pairwise algorithms (such as AETG) generate test cases until all the pairs of values between any two parameters have been included in at least one test case. By assigning weights to pairs, the tester expresses that, if two different pairs have the same chance of being included in a test case, CTWeb should include that with a higher weight. By default, all pairs have 0 as weight.

Actually, the **%Excludes** and the **%Weights** sections are used only by the PROW algorithm (Pairwise with Restrictions, Order and Weight), which will be described in the next section.

Note that several oracles use the reserved word **ANY**, what makes reference to any value of the referenced variable. The first oracle, for example, says that, always that a conversion of -300, -273.01, -459.41 or -459.4 is to be made, independently of the source unit, the result should be -∞.

2.4 Execution with PROW

If we upload the text file in Figure 12 to CTWeb and press *Execute,* the tool shows, in the first time, the pairs tables corresponding to the three parameters: this is, it shows the pairs table for *(MAGNITUDE, SOURCE), (MAGNITUDE, TARGET)* and *(SOURCE, TARGET).* Since, in the **%Excludes** section, we have said that, for *(SOURCE, TARGET),* the pairs *(C, KG), (C, P)* and *(C, O)* must not be included in any test case, these three pairs appear checked (Figure 13). The figure also shows the weights assigned to those pairs appearing in the **%Weights** section of the text file.

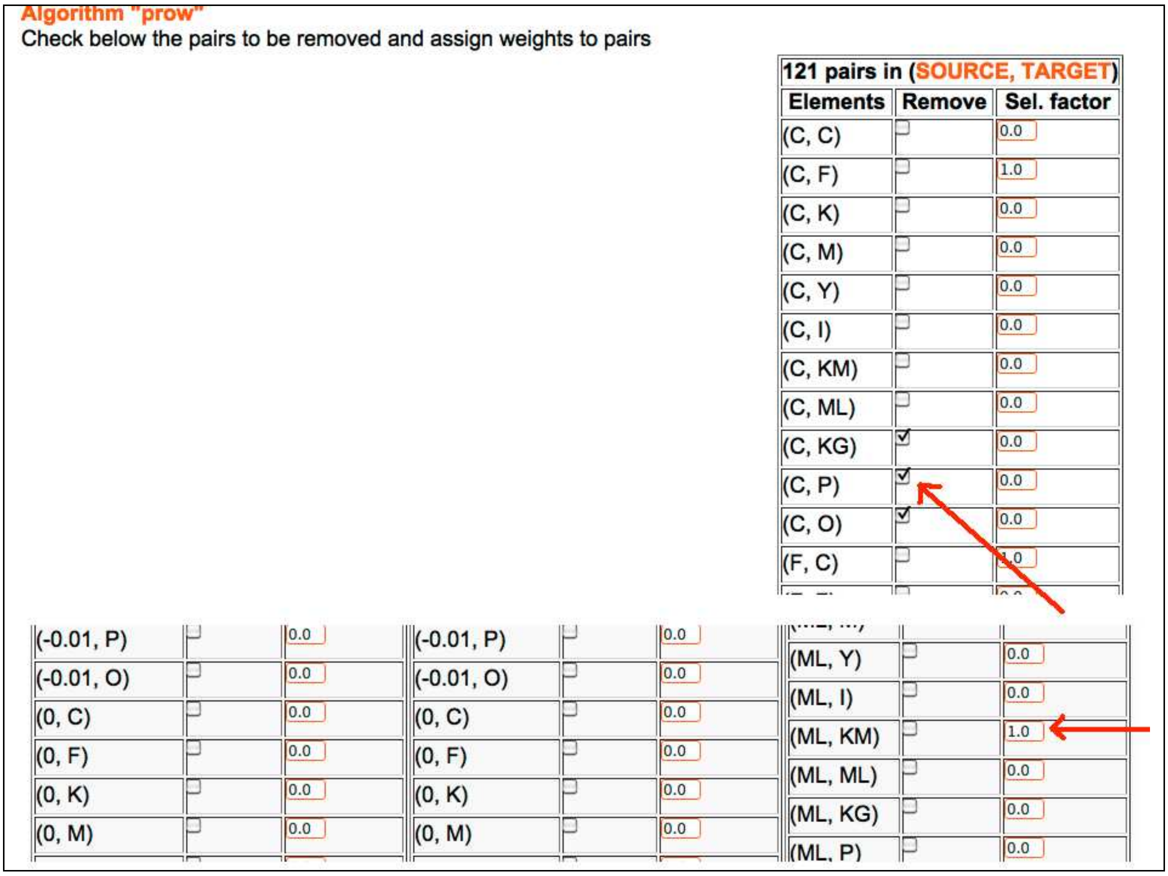


Figure 13

If we agree with this execution configuration, we can press again *Execute* and the tool gives us the set of test cases: all the desired pairs (i.e., those which have not been excluded) are visited at least once; if it has been possible, those pairs with more weight will have been included more often than those with less; furthermore, the test suite is ordered according to the sum of the weights of the pairs included in the test case.

2.5 A grammar of the variables file

A brief grammar for variables files is the following:

file = sets [includes]? [excludes]* [weights]* [oracle]* [testTemplate]?
sets = **%Sets** \n [variableDefinition \n]+
variableDefinition = variableName \t value [\t value]*
includes = **%Includes** \n [combination]+
combination = value **,** value **,** value ... \n
excludes = **%Excludes** \n variableName **,** variableName \n [pair \n]+
pair = value **,** value
weights = **%Weights** \n variableName **,** variableName \n [pair **,** number\n]+
oracle = **%Oracle** [*freeText*]? \n [variableValues \n]+ oracleLine \n
variableValues = variableName \t [value [\t value]*] **|** **ANY** \n
oracleLine = simpleOracle **|** conditionOracle **|** otherwiseOracle
simpleOracle = **oracle** \t *freeText*
conditionalOracle = **conditionalOracle** \t condition \t *freeText*
otherwiseOracle = **otherwise** \t *freeText*
testTemplate = **%Test template** \n *freeText* \n **%%**

Note that:

1) **\t** and **\n** respectively denote a tab and a carriage return.
2) The *freeText* in the **%Oracle** section can contain variable names, with the # prefix.
3) The *freeText* in the **%Test template** section may also contain references to the variables, but in this case using #A, #B, #C, #D, etc. to do reference to the first, second, third, fourth, etc. variable.
4) There may exist an **otherwhise** oracle, which is an expression that is added to all test cases whose test data do not fit to any other oracle. See an example in the next section.

2.6 Use of numeric variables in conditional oracles

A famous problem in software testing is the determination of the type of a triangle according to three values that represent the lengths of its three sides. These values may correspond to an equilateral, isosceles or scalene triangle or, maybe, not to a triangle (negative sides, sum of two sides greater or equals to the third one).

As a last text file example, the following one can be used to exercise the problem of determining the type of a triangle: in this example, we have boldfaced the last oracle (an **otherwise** oracle), which corresponds to triangles of the scalene type. This oracle will be added to all those test cases whose values do not match with any of the other oracles.

```
%Sets
N_I     -1      0       1       2       3       4       5       6
N_J     -1      0       1       2       3       4       5       6
N_K     -1      0       1       2       3       4       5       6

%Oracle // EQUILATERAL
N_I     1       2       3       4       5       6
N_J     1       2       3       4       5       6
N_K     1       2       3       4       5       6
conditionalOracle #N_I==#N_J && #N_J==#N_K      assertTrue(result==Triangle.EQUILATERAL);

%Oracle // A line or negative sides(s)
N_I     ANY
N_J     ANY
N_K     ANY
conditionalOracle #N_I+#N_J==#N_K || #N_I+#N_K==#N_J || #N_J+#N_K==#N_I || #N_I<=0 || #N_J<=0 ||
#N_K<=0         assertTrue(result==Triangle.NO_TRIANGLE);

%Oracle // Isosceles
N_I     1       2       3       4       5       6
N_J     1       2       3       4       5       6
N_K     1       2       3       4       5       6
conditionalOracle (#N_I==#N_J && #N_I!=#N_K) || (#N_I==#N_K && #N_I!=#N_J) || (#N_J==#N_K &&
#N_J!=#N_I)     assertTrue(result==Triangle.ISOSCELES);

%Oracle // Sides do not fit
N_I     1       2       3       4       5       6
N_J     1       2       3       4       5       6
N_K     1       2       3       4       5       6
conditionalOracle (#N_I>#N_J+#N_K) || (#N_J>#N_I+#N_K) || (#N_K>#N_I+N_J)
        assertTrue(result==Triangle.NO_TRIANGLE);

%Oracle         // Default oracle
otherwise       assertTrue(result==Triangle.SCALENE);


%Test template
public void testTCNUMBER() {
        Triangle t=new Triangle();
        t.setI(#A);
        t.setJ(#B);
        t.setK(#C);
        t.calculateType();
        int result=t.getType();
        ORACLE
}
%%
```

Figure 14

As our tool is implemented in Java, the conditional expressions of the conditional oracles are processed and evaluated as Java expressions. In order to give a suitable processing to conditions that involve numeric variables, remember to include the prefix **N_** to those numeric variables which will appear in some condition. Due to this, in the example of the previous figure we called *N_I, N_J*and *N_K* to the three variables used.

In general, it is a good idea to name all numeric variables with the prefix**N_.** In the text filesof the *convert* function used in the previous pages, a good name for the *MAGNITUDE* variable had been *N_MAGNITUDE*, even though it does not appear in any condition of any **conditionalOracle**.

3 The state machine tool

State machines have been widely used as models to generate test cases, and there exist several coverage criteria to assess the quality of the test suite T:

1) State coverage. A test suite T satisfies state coverage if each state is covered by one or more test sequences in T.
2) Transition. A test suite T satisfies this criterion if each transition is traversed by one or more test sequences in T.
3) Full predicate. For each predicate P on each transition and each test clause ci in P, T must include tests that cause each clause ci in P to determine the value of P, where ci has both the values true and false. A predicate is a boolean expression whose value may determine the triggering of a transition.
4) Transition pair. For each pair of adjacent transitions (Si, Sj) and (Sj, Sk), T must contain a test that traverses each transition of the pair in sequence.

Consider for example the state machine in Figure 15, that models the behavior of a supposed banking account.

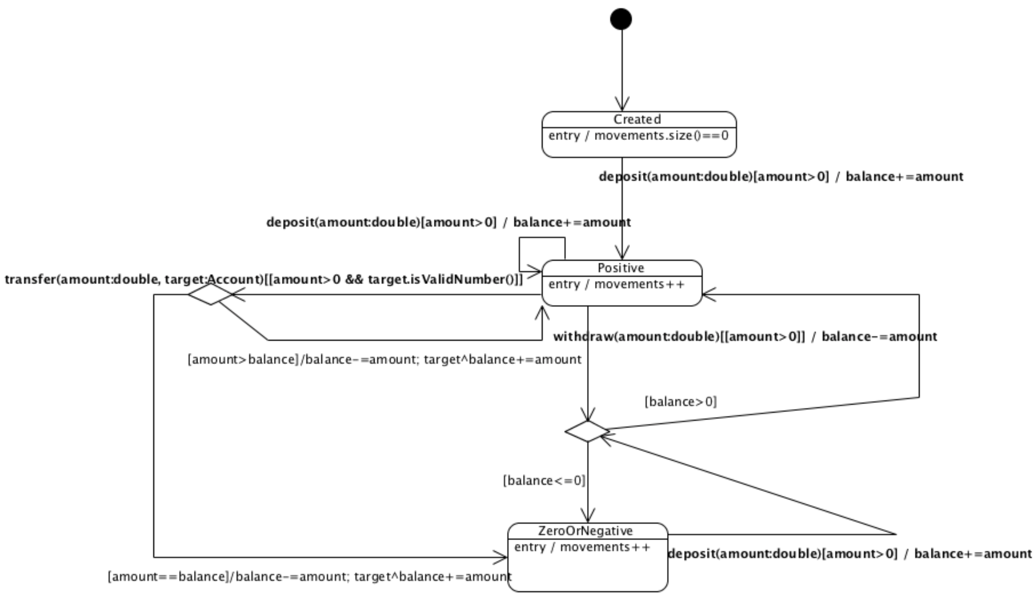


Figure 15

In order to get, for example, states coverage, a possible test case could be:

create·deposit(100)·withdraw(200)

Obviously, states coverage leaves (or may leave) many uncovered transitions, and that's the reason of using stricter coverage criteria.

3.1 Description of state machines with text files

CTWeb may process state machines described as simple text files. The following figure shows a text representation of the state machine in Figure 15:

```
% This is a small example of a state machine description file

Initial node
Created

% Transitions have: source state TAB symbol of the alphabet and target state TAB all of them comma-
separated (TAB is a tabulator)
Transitions
Created	deposit	Positive
Positive	deposit	Positive
Positive	withdrawAndBalanceGreaterThanZero	Positive
Positive	withdrawAndBalanceLessOrEqualThanZero	Negative
Negative	depositAndBalanceGreaterThanZero	Positive
Negative	depositAndBalanceLessOrEqualThanZero	Negative
Positive	transferAndBalanceGreaterThanZero	Positive
Positive	transferAndBalanceLessOrEqualThanZero	Negative

% Symbols can be mapped to method calls of the system using: symbol TAB method.
Symbol aliases
deposit	deposit(amount);
withdrawAndBalanceGreaterThanZero	withdraw(amount);
withdrawAndBalanceLessOrEqualThanZero	withdraw(amount);
depositAndBalanceGreaterThanZero	deposit(amount);
depositAndBalanceLessOrEqualThanZero	deposit(amount);
transferAndBalanceGreaterThanZero	transfer(amount, targetAccount);
transferAndBalanceLessOrEqualThanZero	transfer(amount, targetAccount);

% States can also be used to the further creation of action oracles.
% The syntax is State TAB expression and the label is State aliases. For example:
State aliases
Created	// Check the account has a balance =0 and has no movements
Positive	// Check the account has a balance >=0
Negative	// Check the account has a balance <0
```

Figure 16

As you see, there are four sections in the file, each highlighted in the figure with boldfaced labels:

1) **Initial node** points to the initial node of the state machine. In the example, this one is stated called *Created.*
2) With **Transitions** we represent the transitions in the state machine:
   a. The first transition goes from the *Zero* to the *Positive* state by means of the a call to the *deposit* operation.
   b. The second one corresponds to a *deposit* call from *Positive* to *Positive.*
   c. Then, the *withdraw* operation can be called from *Positive* and may go to two different states: to *Positive* (if the balance remains >=0) or to *Negative* (if the balance remains <0). In this case we represent these two possibilities with two different transitions:
      i. *withdrawAndBalanceGreaterThanZero*, that goes from *Positive* to *Positive.*
      ii. withdrawAndBalanceLessOrEqualThanZero, that from *Positive* to *Negative.*
   d. The next two transitions correspond to calls to *deposit* from the *Negative* state, that may put the machine in *Positive* or *Negative.*
   e. Finally, the two calls to *transfer* from *Positive* are represented in the last two lines of this section.
3) In the **Symbol aliases** section, the tester assigns messages or triggers to the transitions enumerated in the **Transitions** section. For example, it is said

that *deposit* (used in the transitons *Zero deposit Positive*) actually corresponds to a call to *deposit(amount);* that *withdrawAndBalanceGreaterThanZero* and *withdrawAndBalanceLessOrEqualThanZero*are really calls to *withdraw(amount)*, etc. The tester may assign here actual parameters or, as in this example, just leave the parameter names and assign values later… although, actually, she/he may assign any test.

4) As each state represents an invariant condition that the system must fulfill with it is in that state, the **State aliases** section is useful to add the test cases the condition that must be checked when the state is reached. For example, when the machine is in *Created*, it should be tested that has a balance of zero and that it has no movements.

3.2 Processing state machines text files

In Figure 17, the web form for uploading state machines files appears.

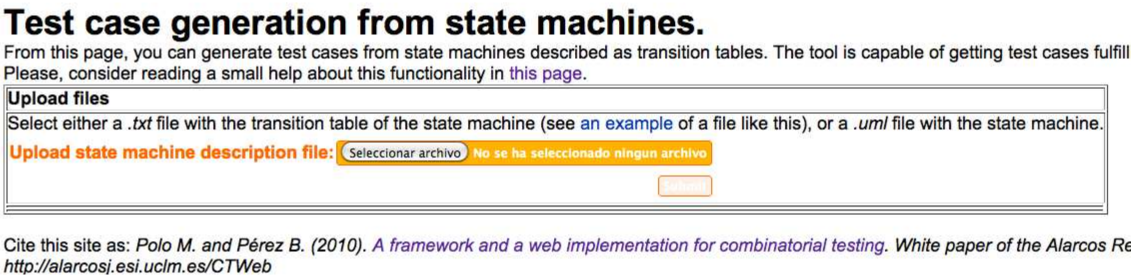


Figure 17

After uploading the file in Figure 16, the tool shows the transitions table, a piece of which is shown in Figure 18.

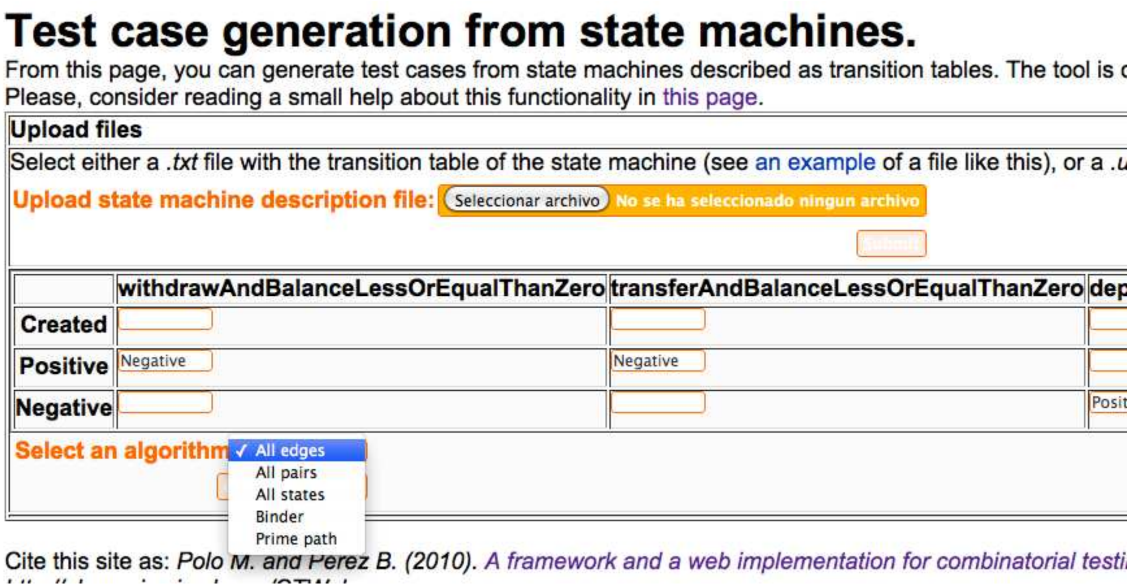


Figure 18

Note the list box with the label "Select an algorithm": depending on the desired coverage criterion, the tester will select one of the provided algorithms:

1) **All edges** produces a test suite that visits all the transitions in the state machine at least once.
2) **All pairs** produces a test suite that, for each state, visits all the pairs of input and output transitions at least once.
3) The test suite generated by **All states** visits all the states in the machine at least once.
4) **Binder** generates test cases according to the Binder's algorithm.
5) **Prime path** produces test cases according to the Prime path algorithm.

If we select, for example, *All transitions*, and press the button labeled *Create test cases,* CTWeb produces the output shown in Figure 19.

Results

Path: [1] [deposit, deposit]
Path: [2] [deposit, withdrawAndBalanceGreaterThanZero]
Path: [3] [deposit, withdrawAndBalanceLessOrEqualThanZero, depositAndBalanceGreaterThanZero]
Path: [4] [deposit, withdrawAndBalanceLessOrEqualThanZero, depositAndBalanceLessOrEqualThanZero]
Path: [5] [deposit, transferAndBalanceGreaterThanZero]
Path: [6] [deposit, transferAndBalanceLessOrEqualThanZero, depositAndBalanceGreaterThanZero]

List of test cases

(May be empty if there are no symbol aliases in the state machine file)

Test case: [1] [deposit, deposit]:
// Check the account has a balance =0 and has no movements
deposit(amount);
// Check the account has a balance >=0
deposit(amount);
// Check the account has a balance >=0

Test case: [2] [deposit, withdrawAndBalanceGreaterThanZero]:
// Check the account has a balance =0 and has no movements
deposit(amount);
// Check the account has a balance >=0
withdraw(amount);
// Check the account has a balance >=0

Test case: [3] [deposit, withdrawAndBalanceLessOrEqualThanZero, depositAndBalanceGreaterThanZero]:
// Check the account has a balance =0 and has no movements
deposit(amount);
// Check the account has a balance >=0
withdraw(amount);
// Check the account has a balance <0
deposit(amount);
// Check the account has a balance >=0

Test case: [4] [deposit, withdrawAndBalanceLessOrEqualThanZero, depositAndBalanceLessOrEqualThanZero]:
// Check the account has a balance =0 and has no movements
deposit(amount);
// Check the account has a balance >=0
withdraw(amount);
// Check the account has a balance <0
deposit(amount);
// Check the account has a balance <0

Test case: [5] [deposit, transferAndBalanceGreaterThanZero]:
// Check the account has a balance =0 and has no movements
deposit(amount);
// Check the account has a balance >=0
transfer(amount, targetAccount);
// Check the account has a balance >=0

Test case: [6] [deposit, transferAndBalanceLessOrEqualThanZero, depositAndBalanceGreaterThanZero]:
// Check the account has a balance =0 and has no movements
deposit(amount);
// Check the account has a balance >=0
transfer(amount, targetAccount);
// Check the account has a balance <0
deposit(amount);
// Check the account has a balance >=0

Figure 19

The figure shows, in the first time, the six paths the tool has generated to go through all transitions; then, for each path, it includes the set of calls required to exercise each transition included in the path, as well as the alias of each state.

3.3 One more example

The following state machine represents a *Manager* that controls the light flow of two semaphores: when there are no pedestrians, the manager changes the light of both semaphores (*a* and *b*) sending them the *change* event every a fixed number of seconds (60, 63, 66, 83, and 86). However, a pedestrian may request the red light in any of the semaphores: if the semaphore where red is requested is in yellow or red, nothing happens; if it is in green and the semaphore is *a,* then the managers changes *a* to yellow either 20 seconds after the request or, if less than 20 seconds remain, in this time. If the red light is requested on *b*, then the request is passed to *a*.

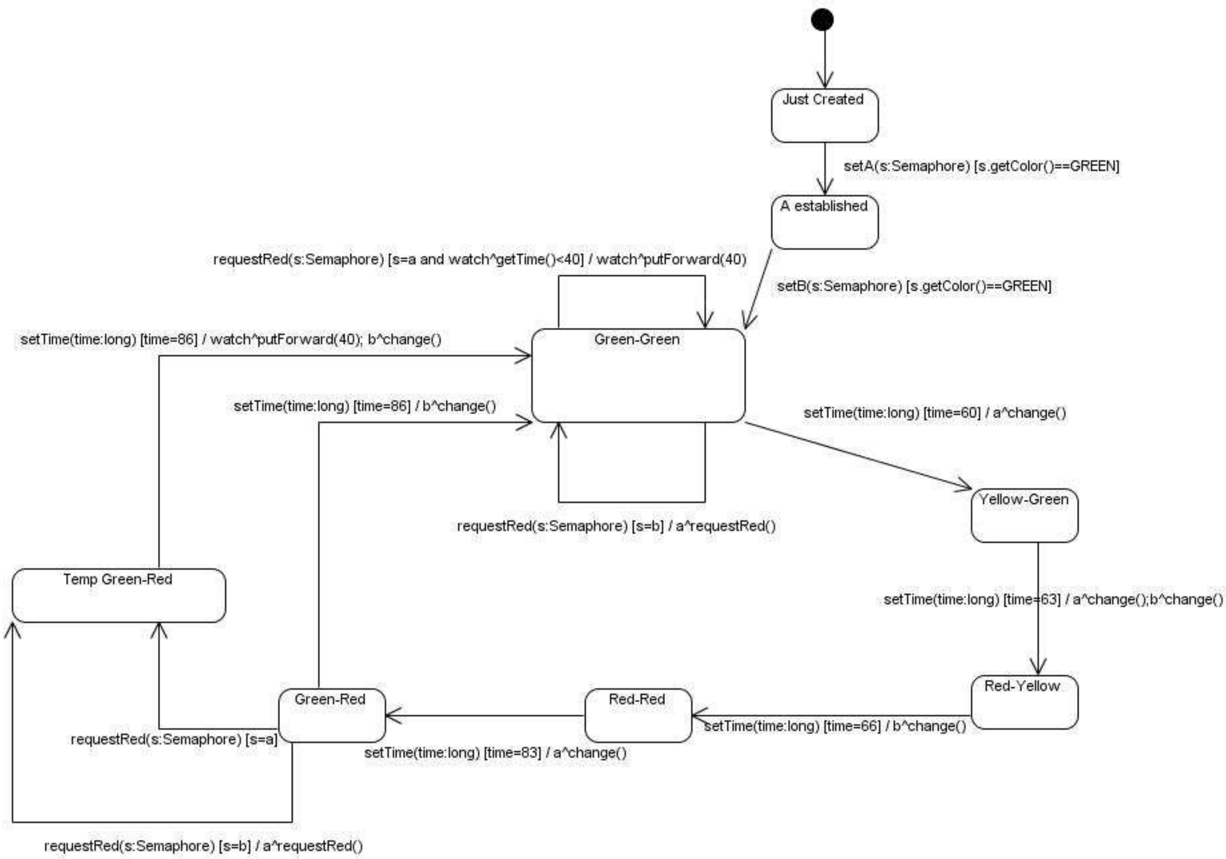


Figure 20

The system is implemented as a single Java desktop application (Figure 21) whose structure is shown in

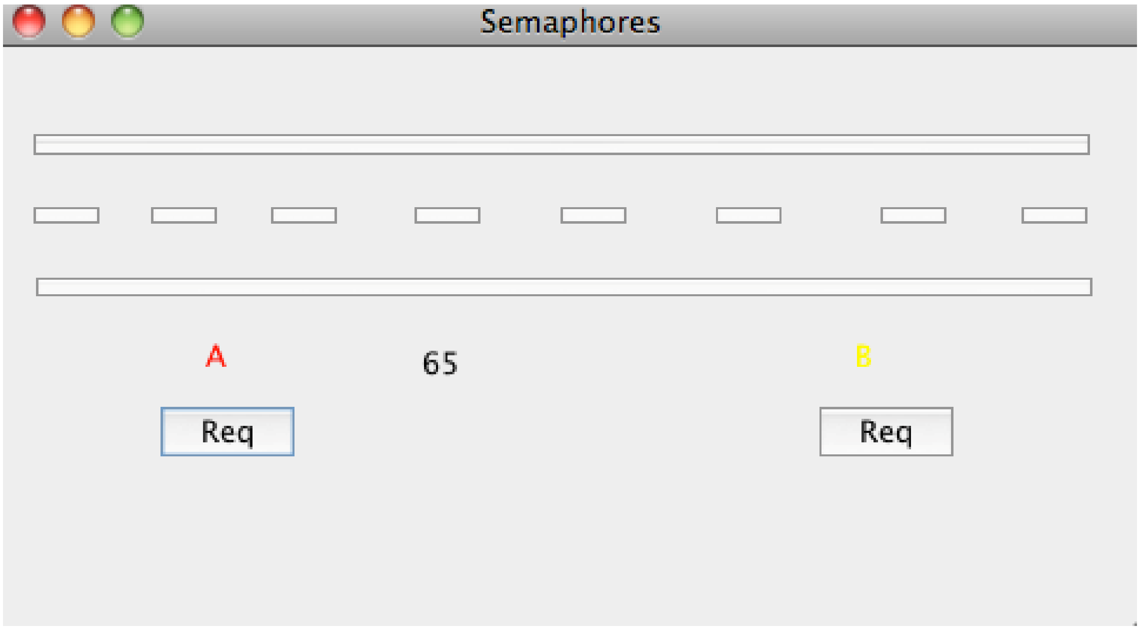


Figure 21

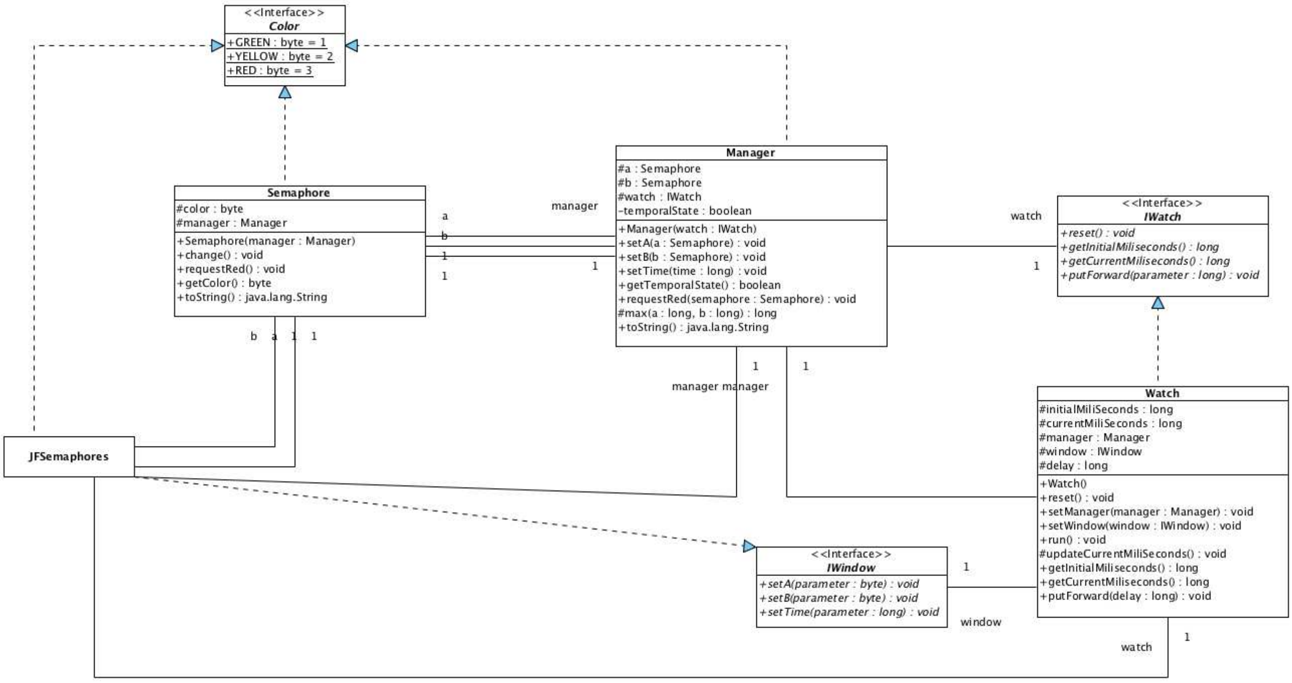


Figure 22

The text representation of the state machine is the following:

% This is a small example of a state machine description file

Initial node
JC

% Transitions have: source state TAB symbol of the alphabet TAB target state
Transitions

JC	setA	A established
A established	setB	GG

```
GG	requestRedOnA		GG
GG	requestRedOnB		GG
GG	setTime60	YG
YG	setTime63	RY
RY	setTime66	RR
RR	setTime83	GR
GR	requestRedOnA		TGR
GR	requestRedOnB		TGR
TGR	setTime86	GG
GR	setTime86	GG

% Symbols can be mapped to method calls of the system using: symbol TAB method.
Symbol aliases
setA	Manager manager=new Manager(this); Semaphore a=new Semaphore(manager);
manager.setA(a);
setB	Semaphore b=new Semaphore(manager); manager.setB(b);
requestRedOnA		manager.requestRed(a);
requestRedOnB		manager.requestRed(b);
setTime60	manager.setTime(60);
setTime63	manager.setTime(63);
setTime66	manager.setTime(66);
setTime83	manager.setTime(83);
setTime86	manager.setTime(86);

% States can also be used to the further creation of action oracles.
% The syntax is State TAB expression and the label is State aliases. For example:
State aliases
A established	assertTrue(a.toString().equals("GREEN"));
GG	assertTrue(manager.toString().equals("GREEN,GREEN,false"));
YG	assertTrue(manager.toString().equals("YELLOW,GREEN,false"));
RY	assertTrue(manager.toString().equals("RED,YELLOW,false"));
RR	assertTrue(manager.toString().equals("RED,RED,false"));
GR	assertTrue(manager.toString().equals("GREEN,RED,false"));
TGR	assertTrue(manager.toString().equals("GREEN,RED,true"));
```

Figure 23

If we upload this file and generate a test suite covering *All pairs,* we get a set of test cases that can copy and paste on our IDE. Two of these test cases are:

```
   publicvoid test1() {
      Manager manager=new Manager(this); Semaphore a=new Semaphore(manager);
manager.setA(a);
      assertTrue(a.toString().equals("GREEN"));
      assertTrue(a.toString().equals("GREEN"));
      Semaphore b=new Semaphore(manager); manager.setB(b);
      assertTrue(manager.toString().equals("GREEN,GREEN,false"));
      manager.requestRed(a);
      assertTrue(manager.toString().equals("GREEN,GREEN,false"));
      manager.requestRed(a);
      assertTrue(manager.toString().equals("GREEN,GREEN,false"));
      manager.requestRed(b);
      assertTrue(manager.toString().equals("GREEN,GREEN,false"));
      manager.requestRed(a);
      assertTrue(manager.toString().equals("GREEN,GREEN,false"));
      manager.setTime(60);
      assertTrue(manager.toString().equals("YELLOW,GREEN,false"));
      manager.setTime(63);
      assertTrue(manager.toString().equals("RED,YELLOW,false"));
      manager.setTime(66);
      assertTrue(manager.toString().equals("RED,RED,false"));
      manager.setTime(83);
```

```
      assertTrue(manager.toString().equals("GREEN,RED,false"));
      manager.requestRed(a);
      assertTrue(manager.toString().equals("GREEN,RED,true"));
      manager.setTime(86);
      assertTrue(manager.toString().equals("GREEN,GREEN,false"));
      manager.requestRed(a);
      assertTrue(manager.toString().equals("GREEN,GREEN,false"));
   }

   publicvoid test2() {
      Manager manager=new Manager(this); Semaphore a=new Semaphore(manager);
manager.setA(a);
      assertTrue(a.toString().equals("GREEN"));
      assertTrue(a.toString().equals("GREEN"));
      Semaphore b=new Semaphore(manager); manager.setB(b);
      assertTrue(manager.toString().equals("GREEN,GREEN,false"));
      manager.requestRed(b);
      assertTrue(manager.toString().equals("GREEN,GREEN,false"));
      manager.requestRed(b);
      assertTrue(manager.toString().equals("GREEN,GREEN,false"));
      manager.setTime(60);
      assertTrue(manager.toString().equals("YELLOW,GREEN,false"));
   }
```

Figure 24