Random Cluster Sampling on X-Machines Test Cases

Yasir Imtiaz Khan
Laboratory of Advanced Software Systems
University of Luxembourg
6, rue R. Coudenhove-Kalergi, Luxembourg
Email: khialian@hotmail.com

Sadia Kausar
Department of Computer Science
The University of Lahore
1-km Rai Wind Road Lahore Pakistan
Email: sadiakausar1@gmail.com

***Abstract*—Software testing is considered one of the most expensive and critical phases of the software development. Formal testing approaches are extensively used for verifying the conformance of implementations to a given specification. These formal approaches usually generate a large amount of input test data which is costly in terms of time and effort. Techniques for reducing test input data are thus of the utmost importance. The contribution of this paper is to propose a framework for the reduction of test input data generated by a formal testing approach based on X-Machines.**

To achieve these objectives we have applied a well known statistical approach called Random Cluster Sampling on the test case set generated by a formal approach X-Machines. To exemplify our technique we have generated a test set for an X-Machine Microwave oven specification and then drew a sample from the test set by using the Random Cluster sampling technique. Based on the tolerated fault rate we have extracted conclusion about the accuracy of implementation.

I. Introduction

Software testing is considered one of the most costly and critical phases of software development [1]. The National Institute of Standards and Technology (NIST) reported recently that software defects cost the U.S. economy *59.9* billion dollars per year. The report estimates that *22* billion dollars could be saved through the early application of more effective defect detection methodologies. According to this report software error identification and correction cost is *80%* of the overall cost of software development [2]. There exists thus a compelling need to produce efficient testing approaches in order to reduce the software development cost. Formal testing approaches are extensively used for checking the conformance of implementations regarding specifications. These formal approaches typically generate a large amount of input test data for which reduction techniques are currently being studied by the community [3].

The work we present in this paper consists of reducing test sets generated for X-Machines models. X-Machines were pioneered by Eilenberg [4] [5] in the seventies and are considered as an abstract computation model useful for writing software specifications that, given their operational flavor, can be effectively used of Model-Based Testing.

Sampling is a statistical technique which is used in almost every discipline in order to collect information about a whole population. On its basis, inference about the characteristics of the whole population is made [8]. It is a popular method that provides sufficient information about the characteristics of a population without examining every unit of whole population [9]. Statistical sampling methods are being currently being adopted by the software industry for reliability testing, fault detection, verification and filtering of execution profiles for test set reduction [10]. Sampling saves cost and time as it is much cheaper to collect the desired information from a small sample than from the whole population. Additionally, sampling provides information that is almost as accurate as that obtained from whole population.

In particular, we are using a technique called Random Cluster Sampling. Fundamentally, categorizing objects into one group that are dissimilar and heterogeneous from objects in another group is called clustering. Here we consider X-Machines test cases as clusters of function elements. The dissimilarity and heterogeneity that exists between X-Machines' test cases is due to some unique properties of those test cases: overall *behavior* of the test case, *size* of the test case and *order* of the function elements that compose a test case.

For example, a test case may represent the intended behavior of system, and some test cases may represent the unintended behavior of the system. Each individual test case is independent from any other test case in terms of execution, given it may execute independently. Each test case of X-Machines may be different in size regarding other test cases, because it may consist of different number of functions elements. X-Machines test cases consist of a sequence of function elements and inherently there exists a particular order between those elements that form test cases. These functions are input complete and output distinguishable.

In this paper, we present an approach that supports reducing the test sets generated from X-Machines. We extract sample test sets from the whole test cases of X-Machines using Random Cluster Sampling. First of all the sample size is determined and then we pick test cases randomly from the whole test cases of X-Machines. Thus, according to the theory, the accuracy of the sample test set in terms of error discovery should be equivalent to that of the whole test set.

The rest of paper is organized as follow: In Section II detailed related work is discussed. Introduction to Random Cluster sampling is given in section III. An introduction to X-

Machines is described in section IV. An X-Machine specification for a Micro wave oven is given in section V. In section VI an algorithm for the proposed test case selection frame work and its description is described. Section VII consists of testing X-Machines. In section VIII experimental results and an evaluation of our approach is given. Discussion about the proposed methodology and its pictorial representation is given in section IX. In the last section conclusion and future work is provided.

II. Related Work

The objective of software testing is to uncover errors and reduce the overall cost of software. Several software testing methodologies, tools, test case reduction techniques have been proposed over the past few decades that aim at better software quality. Exhaustive testing of any software system is impractical due to its laborious and costly nature. In [11] a technique for test set reduction based on a genetic algorithm has been proposed and provides full fault coverage regarding the original test set. The study in [12] also showed that the clustering technique helps reducing test set size, its overall execution cost and time is reduced for regression testing [12].

The authors of [13] integrated random testing, program invariants (e.g. data coverage properties of program) and a genetic algorithm to achieve a reduced test case set. In [14] the coverall algorithm for test case reduction using simple algebraic conditions is described. By using algebraic conditions the authors assign fixed values to variables for test case reduction.

Software testing approaches based on statistical methods motivate the software industry to scientifically test software reliability (probability of proper operation of software). Statistical testing techniques as described in [15] are based on probability distributions and inference procedures to ensure reliability. The paper [16] described the use of sampling techniques for selection of test sets from software input domain and proposes a search based approach to derive an appropriate distribution for software verification through statistical testing.

The authors of [17] suggested cluster analysis techniques to improve the estimation of software reliability. Cluster filtering techniques select a subset of test cases from test set and also indicate that this technique is more efficient then simple random sampling. Cluster filtering techniques have been applied effectively to test set reduction [17], [18] – the aim is to sample from each cluster consisting of test cases, based on using an execution profile. The method described in [19] involves an iteration process to select a sample of test cases from the clusters during each iteration. Each test case is chosen to have a maximum possibility to provoque a failure, based on information provided by last test case chosen from same cluster.

III. Random Cluster Sampling

Random Cluster sampling is a statistical technique that helps in test selection by reducing tester's effort by reducing test case suite size. In Random Cluster sampling, clusters are heterogeneous in nature. As they are heterogeneous in nature, they have ability to capture all the variability of population. Random cluster sampling is done by many ways, e.g. 1-stage cluster sampling, 2-stage-cluster sampling and *n*-stage cluster sampling. Here we consider the single stage cluster sampling method also known as random cluster sampling. It has the following properties.

The population is divided into *N* groups, called clusters. For example, researcher selects n clusters at random.

Sample Size Determination:

Factors that are largely involve in determination of appropriate sample size [20] [21]:

The estimated prevalence of the variable of interest accuracy of X-Machines in this instance, (*p=0.985*).

The desired level of confidence (*t= 0.95*) i.e. *1.96* standard value.

The acceptable margin of error, (*m=0.05*).

For Random Cluster sampling design based on a simple random sample, the sample size required can be calculated according to the following formula [8].

$$n = (t^2 * p(1-p))/m^2 \quad (1)$$

X-Machines test cases contain sequence of functions that are distinguishable, independent and have certain unique properties in terms of order, number and size. For example, some test case may represent the intended behavior of system, and some test cases may represent the unintended behavior of the system. The unintended behavior is always unique from the intended behavior of test case. Each test case may consist of the different number of functions of X-Machines. For example one test case may consist of two, three or four functions. Each individual test case is independent from the other test case (in terms of execution or it may execute independently).

Each test case of X-Machines is consisting on group of functions. Because clusters are group of functions and in case of X-Machines test cases, grouping of function in a single test case is naturally evident and hence considered as cluster. In this study, scope of cluster is from context of cluster sampling domain. A single test case of X-Machines where grouping of function is naturally evident is considered here as cluster. And the scope of term cluster is neither taken nor refers to cluster of cluster analysis theory here.

IV. Introduction To X-Machines

Model based testing approaches are extensively used for the conformance between the formal model of the system and its implementation. To model the behavior a system by using model based testing, label transition system, UML model, finite state machines, extended finite state machines (EFSM), Markov chain model (MCM), State charts (SC) and X-Machines etc. are most commonly used techniques in literature. Model based testing techniques are the blend of formal methods and testing. Model based testing techniques have ability to automatically generate test cases. The scope of this work is limited to X-Machine only. The following section contains introduction of some aforementioned model based testing methodologies and introduction of X-Machines in detail.

X-Machines is a formal method which is used to specify a system. This computation machine was introduced by Eilenburg and extended by Holcombe [4] [5]. X-Machines captures both the control behavior and functional behavior. X-Machines are used in static and dynamic analysis also. X-Machines follow the diagrammatic approach and is an extension to the finite state machines with two noteworthy differences

1) Simple input labels on the transition are replaced by functions
2) Infinite Memory is attached to machine

Actually X-Machines are integration of control and functional behavior. A particular class of X-machines is stream X-machines (SXM), which is defined as a construct as follows

$(X, \Sigma, \Gamma, \phi, Q, M, F, I, T)$

X representing fundamental data set that machine operates.

Σ, Γ are input and output sets;

Q is the set of finite states;

ϕ is of type SXM and a finite set of partial functions ϕ that map an input and a memory state to an output and a new memory state $\phi : \Sigma \times M \rightarrow \Gamma \times M$;

F is the next state partial function that given a state and a function from the type ϕ, provides the next state,

$F : Q \times \phi \rightarrow \mathbb{P} Q$

I, T are initial and final states respectively which are of type Q .

There are also non deterministic and deterministic X Machines. X-Machines in which all data are triples consisting of stream of input,output symbols and memory values are called stream X-Machines. X-Machines testing methods also tests the equivalence of the behavior of specification and implementation. A function modeled by X-Machines itself can also be tested by X-Machines testing method. The time taken by X-Machines testing methods depends on the system to be tested. X-Machines testing methodology due to imposed constraints is easy [6] [7].

V. X-Machines for Microwave Oven

X-Machine for Microwave oven is defined as

$(X, \Sigma, \Gamma, \phi, Q, M, F, I, T)$

Input Set $\Sigma = \{open, close, t.set, run, t.out, interrupt\}$;

Output Set $\Gamma = \{D.Open, D.Close, Clock, Work\}$;

Set of States$Q = \{IDLE, D.OPEN, , COOKING\}$;

$Max - Time, Current - Time, Set - Time : N$;

Memory M = $Current - Time, Set - Time$;

Initial Memory$m_0 = (0, 0)$;

Initial State $q0 = IDLE$;

Next State Function$F : Q \times \phi \rightarrow Q$;

Type ϕ;

OPEN(open,$(Current - Time, Set - Time)) = [true]$
$(D.Open, (Current - Time = 0, Set - Time = 0))$.

CLOSE(close,$(Current - Time, Set - Time)) = [true]$
$(D.Close, (Current - Time, Set - Time))$.

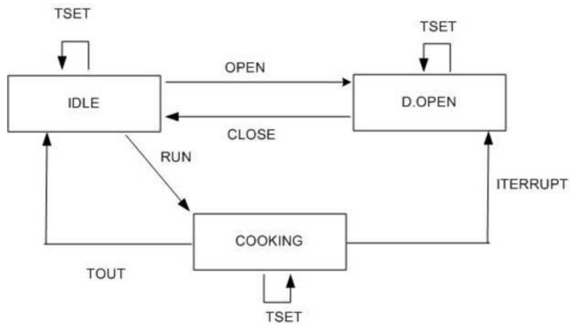


Fig. 1. X-Machine for Microwave oven

T.SET(t.set,$(Current - Time, Set - Time)) = [Set - Time < Max - Time]$
$(Clock, (Current - Time + Set - Time))$.

RUN(run,$(Current - Time, Set - Time)) = [Set - Time < Max - Time]$
$(Work, (Current - Time, Set - Time))$.

T.OUT(t.out,$(Current - Time, Set - Time)) = [Set - Time = Current - Time]$
$(D.Close, (Current - Time = 0, Set - Time = 0))$.

INTERRUPT(interrupt,$(Current - Time, Set - Time)) = [true]$
$(D.Open, (Current - Time = 0, Set - Time = 0))$.

In this section, X-Machines specifications of microwave oven is described. Initial state of microwave is idle and its temperature initially is set to zero, we have taken an assumption that maximum temperature can be up to *100* minutes.

Open function will change its idle state to door open state while time set function will first ensure that temperature must be less than maximum temperature and will update current temperature. Time out operation first ensures that microwave reaches on the set temperature value.

VI. Testing from X-Machines

It is necessary to test and prove the specification of the developed model. X-Machines are used for specifying system and producing test set. Test set generated from X-Machines is finite set of functions. These functions are input complete and output distinguishable. It is possible to pass special test inputs to the functions of X-Machines. There are two parts of test generation:

1) Generating a set Y_{m-n} following W-method, where m is upper bound on the number of states in an implementation and n is the number of states in the specification.
2) Computation of $t_n Y_{m-n}$ using a test function.

X_{m-n} = tY_{m-n} where t is function of Z,

Y_{m-n} = P ($\phi^{\ m-n} \cup ... \cup \{\epsilon\}$)(W $\cup$ $\{\epsilon\}$),

P is transition cover of A_z and W is characterization set of A_z.

Testing feature of X-Machine makes it more prominent in the family of formal specification of systems. Here we have generated test case set.

$W = \{OPEN, CLOSE, RUN, TOUT, INTERRUPT\}$
$P = \{\lambda, TSET, OPEN, RUN, OPENCLOSE,$
$OPENTSET, RUNTOUT, RUNTSET, RUNINTRRUPT\}$
$S = \{\lambda, RUN, RUNINTRRUPT\}$
$R = P - Q = \{TSET, OPEN, OPENCLOSE, OPENTSET,$
$RUNTOUT, RUNTSET\}$

The derived test set X for example k = 0

$X = \{<(tset, Clock)>, <(open, DOpen)>, <(run, Work)>, <(run, Work), (tout, DClose)>, <(run, Work), <(interrupt, DOpen)>, <(tset, Clock), (open, DOpen)>, <(tset, Clock), (run, Work)>, <(open, DOpen), (close, DClose)>, <(open, DOpen), (close, DClose), (open, DOpen)>, <(run, Work)>, <(open, DOpen), (tset, Clock), (close, DClose)>, <(run, Work), (t.out, DClose), (open, DOpen)>, <(run, Work), (tout, DClose), (run, Work)>, <(run, Work), (tset, Clock), (interrupt, DOpen)>, ...\}$

VII. Algorithm

In this section, we present an algorithm for the proposed framework. Informal description of algorithm is that it takes cluster *Ci* consisting on X-Machines test cases as input and its output is *n*, a reduce test case set that are sufficient to determine the faulty and non-faulty implementation.

First of all, unique numbers are assigned to each Cluster *Ci* (Clusters are X-Machine test cases) where *i = 1, 2, 3,..., n*. Then a sample size *n* is calculated by $n = (t^2 * p(1-p))/m^2$.

An iteration is performed as long as the sample size i.e. *n*, and in its each iteration, it draws a single Cluster at random with replacement from complete *Ci*. Set of Cluster obtained is tested using X-Machines testing method. During this fault rate is observed. Algorithm return implementation is faulty if the observed fault rate is greater than tolerated rate *m* else it returns implementation is according to the specification.

```
Input: Cluster consisting on X-Machine test cases;
Output: Faulty or non faulty implementation;
Perform Cluster Sampling on Clusters (X-Machine Test
Cases);
Assign unique number to each cluster $C_i$ where
$i = 1, 2, 3, \ldots, n$;
Calculate sample size $n = (n = (t^2 * p(1-p))/m^2)$;
for $i = 0$; $i \leq n$; $i++$ do
    Randomly draw a single Cluster from $C_i$;
    $C_i = C_i -$ Previously drawn Cluster;
    $n = n +$ Previously drawn Cluster;
end
Test n Cluster by using X-Machine testing strategy;
if (Observed fault rate in n test cases $\leq$ tolerated fault
rate) then
    Implementation is faulty;
end
else
    Implementation is according to the specification;
end
Return.
```

VIII. Experimental Results

X-Machines testing method generates *45* test cases for the Microwave oven case study. Every test case is representing 2-3 functionalities (shown in the figure 2).

Sample size determination require sample size *n*, we have assumed the confidence level at *t= 95*percent(standard value of *1.96*), estimated prevalence of accuracy in the X-Machines *p=0.985*, margin of error at *m=5*percent (standard value of *0.05*). Applying these values to equation (1) we get *n= 22.703* $\approx$ *23* a required sample size.

As our sample size is *n=23*, First we assign the codes to the clusters from *00 - 44*.

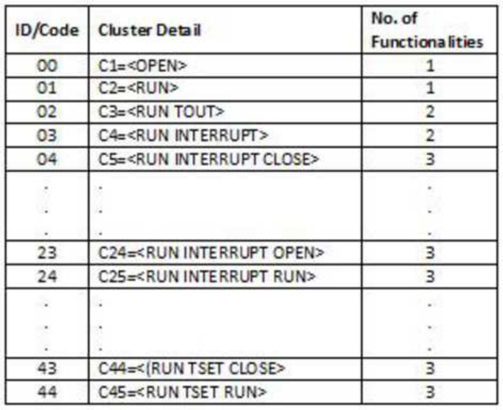
By drawing the random numbers from the random number table [22] (this table has inherent ability to draw unique number in each turn),for sample size *n=23*

cluster *07,12,37,03,19,29,17,01,13,09,31,15,21,11,04,10 ,43,06,27,41, 25,36 and 44* is selected to ease the prescribed software testing procedures.

The final reduced test case suite by using random cluster sampling is.

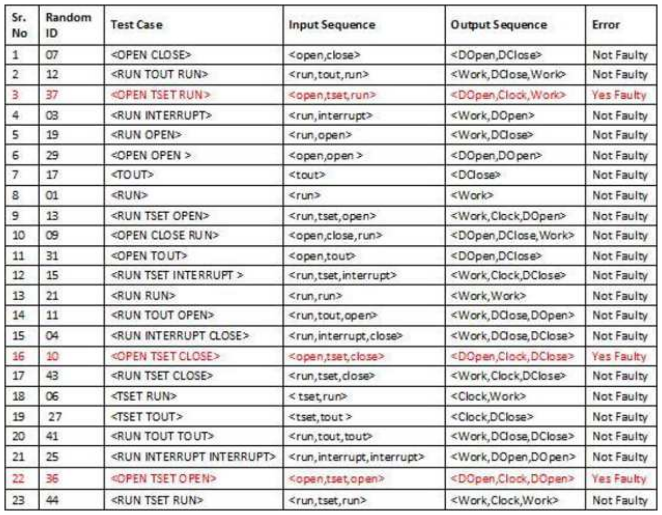
$n = \{<(open, DOpen), (close, DClose)>, <(run, Work), (tout, DClose), (run, Work)>, <(open, DOpen), (tset, Clock), (run, Work)>, <(run, Work), (interrupt, DOpen)>, <(run, Work), (open, DOpen)>, <(run, Work), (open, DOpen)>, <(tset, Clock)>, <(run, Work), (tset, Clock), (interrupt, DOpen)>, <(open, DOpen), (close, DClose), (run, Work)>, <(open, DOpen), (interrupt, DOpen)>, <(open, DOpen), (open, DOpen)>, <(tout, DClose)>, <(run, Work)>, <(run, Work), (tset, Clock), (open, DOpen)>, <(open, DOpen), (close, DClose), (run, Work)>, <(open, DOpen), (tout, DClose)>, <(run, Work), (tset, Clock)>, <(interrupt, DOpen)>, <(run, Work), (run, Work)>, <(run, Work), (tout, DClose), (open, DOpen)>, <(run, Work), (interrupt, DOpen), (close, DClose)>, <(open, DOpen), (tset, Clock), (close, DClose)>, <(run, Work), (tset, Clock), (close, DClose)>, <(tset, Clock), (run, Work)>, <(tset, Clock), (tout, DClose)>, <(run, Work), (tout, DClose), <(tout, DClose)>, <(run, Work), (interrupt, DOpen), (interrupt, DOpen)>, <(open, DOpen), (tset, Clock), (open, DOpen)>, <(run, Work), (tset, Clock), (run, Work)>\}$. To automate the testing process above test set '*n*' exhibiting sequences of operations is converted into sequences of inputs. This conversion is done according to the fundamental test function described in [23].

In order to perform testing on the implemenatation of Microwave oven System, first of all sample of *n=23* test cases are converted into the sequence of inputs. Secondly, it is observed that for every output sequence corresponding to every input sequence of the micro wave implementation, must be identical to one that is expected by the Micro Wave oven Model or its specification. The result of this observation is listed in the fifth column of the table (Figure 3). The percentage of faulty implementation in our resultant output set is *0.13* which is greater than the acceptable margin of



ID/Code	Cluster Detail	No. of Functionalities
00	C1=<OPEN>	1
01	C2=<RUN>	1
02	C3=<RUN TOUT>	2
03	C4=<RUN INTERRUPT>	2
04	C5=<RUN INTERRUPT CLOSE>	3
. . .	. . .	. . .
23	C24=<RUN INTERRUPT OPEN>	3
24	C25=<RUN INTERRUPT RUN>	3
. . .	. . .	. . .
43	C44=<(RUN TSET CLOSE>	3
44	C45=<RUN TSET RUN>	3

Fig. 2. Table for complete test Cases



Sr. No	Random ID	Test Case	Input Sequence	Output Sequence	Error
1	07	<OPEN CLOSE>	<open,close>	<DOpen,DClose>	Not Faulty
2	12	<RUN TOUT RUN>	<run,tout,run>	<Work,DClose,Work>	Not Faulty
3	37	<OPEN TSET RUN>	<open,tset,run>	<DOpen,Clock,Work>	Yes Faulty
4	03	<RUN INTERRUPT>	<run,interrupt>	<Work,DOpen>	Not Faulty
5	19	<RUN OPEN>	<run,open>	<Work,DClose>	Not Faulty
6	29	<OPEN OPEN >	<open,open >	<DOpen,DOpen>	Not Faulty
7	17	<TOUT>	<tout>	<DClose>	Not Faulty
8	01	<RUN>	<run>	<Work>	Not Faulty
9	13	<RUN TSET OPEN>	<run,tset,open>	<Work,Clock,DOpen>	Not Faulty
10	09	<OPEN CLOSE RUN>	<open,close,run>	<DOpen,DClose,Work>	Not Faulty
11	31	<OPEN TOUT>	<open,tout>	<DOpen,DClose>	Not Faulty
12	15	<RUN TSET INTERRUPT >	<run,tset,interrupt>	<Work,Clock,DClose>	Not Faulty
13	21	<RUN RUN>	<run,run>	<Work,Work>	Not Faulty
14	11	<RUN TOUT OPEN>	<run,tout,open>	<Work,DClose,DOpen>	Not Faulty
15	04	<RUN INTERRUPT CLOSE>	<run,interrupt,close>	<Work,DClose,DClose>	Not Faulty
16	10	<OPEN TSET CLOSE>	<open,tset,close>	<DOpen,Clock,DClose>	Yes Faulty
17	43	<RUN TSET CLOSE>	<run,tset,close>	<Work,Clock,DClose>	Not Faulty
18	06	<TSET RUN>	< tset,run>	<Clock,Work>	Not Faulty
19	27	<TSET TOUT>	<tset,tout >	<Clock,DClose>	Not Faulty
20	41	<RUN TOUT TOUT>	<run,tout,tout>	<Work,DClose,DClose>	Not Faulty
21	25	<RUN INTERRUPT INTERRUPT>	<run,interrupt,interrupt>	<Work,DOpen,DOpen>	Not Faulty
22	36	<OPEN TSET OPEN>	<open,tset,open>	<DOpen,Clock,DOpen>	Yes Faulty
23	44	<RUN TSET RUN>	<run,tset,run>	<Work,Clock,Work>	Not Faulty

Fig. 3. Table for fault detection (23test cases)

error *m=0.05*. If the percentage of this observed fault rate is greater than the tolerated parameter *m* then we conclude that the implementation of Microwave oven model is faulty or not equal to specification.

In the table (Figure 4) testing of the complete test cases is shown. The result of the observation is shown in the fifth column of the table. We compared the results obtained from our approach to the results obtained by testing the complete test cases.

Experiment(figure 3) (with *n=23*)

Total test cases=*23*

Faulty implementations=*3*

Percentage of Faulty Implementations= *(3/23)*100=0.1304*100=13.04 %*

Accuracy level=*100-13.04=86.95=87%*

Experiment(figure 7) (with *n=28*)

Total test cases=*28*

Faulty implementations=*3*

Percentage of Faulty Implementations= *(3/28)*100=0.1071*100=10.71%*

Accuracy level=*100-10.71=89.28=89%*

Experiment with Complete test case(figure 4)

Total test case=*45*

Faulty implementation=*5*

Percentage of Faulty Implementations= *(5/45)*100=0.1111*100=11.11%*

Accuracy level=*100-11.11=88.88=89%*

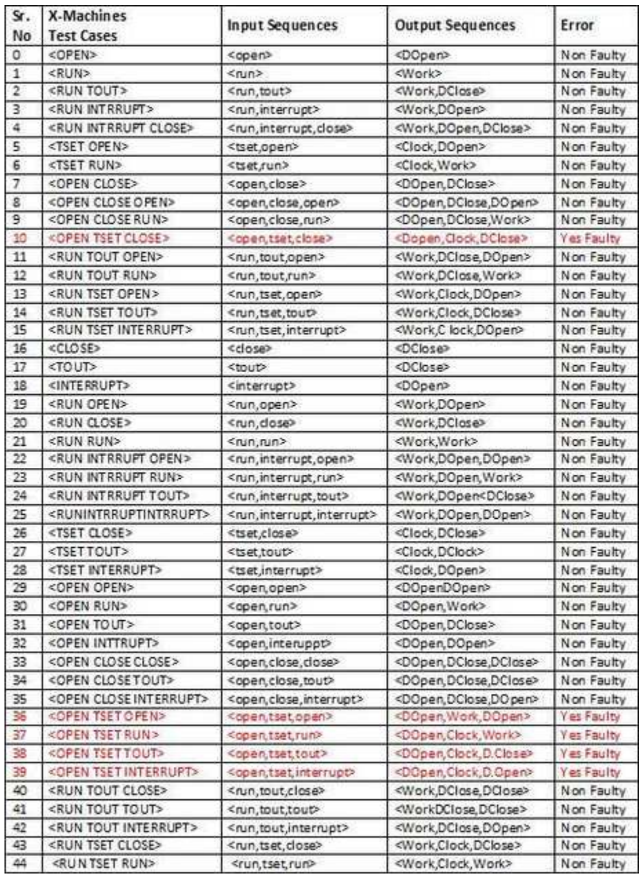
From these calculations, we are in much confidence that testing of the reduced test cases is approximately equal to the testing of complete test cases. We can determine easily about the percentage of faulty implementations and the level of accuracy by using our approach instead doing tiresome work to test all the test cases.

Table shown in (Figure 5) contains a comparative analysis of the proposed with existing mthodology. During experiment artificially an error (transfer error) is introduced in the implementation of case study. Analysis shows that by testing only *23* test cases instead of *45* test cases we can infer the same result.

IX. Discussion

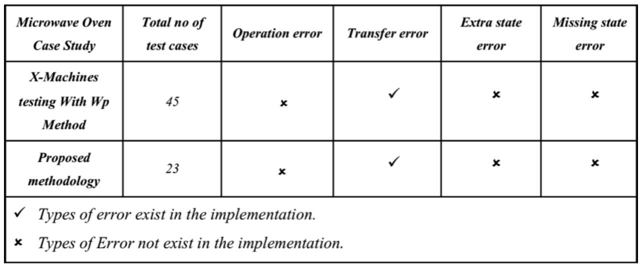
We have given a practical exhibition and explanation of our approach for verifying the conformance of implementation to given specification. Methodology for the proposed work is shown in (Figure 6). Our approach is based on the two well known approaches i.e. X-Machines and Random Cluster Sampling. Each of the test case generated by X-Machines contains sequence of functions inside. These functions are input complete and output distinguishable. Noticeably these functions are distinguishable, independent from each other and therefore provides natural grouping and forms clusters. For this reason Random Cluster Sampling can be applied to the test cases generate by X-Machines. We assumed the certain values such as confidence level, estimated prevalence of accuracy in X-Machines and margin of error to determine the sample size. The user has flexibility to set these values according to his own criteria. In order to check the probability that there may happen a case where the randomly picked test cases are not representing the accurate results(in terms of marginal error). We performed several experiments by setting different parameters and observed the results as shown in (Figure 3 and 7). Here we can clearly see the consistency in the results. Out of *10* experiments one is not showing exact behavior. To avoid this inconsistency we believe that it is sufficient to iterate the process for three times. For example if we have case study consisting*500* test cases and we pick sample of *23* test cases and test it by using our approach and we repeat this process even for three times. Even in this case the effort of testing would be less than the half as compared to testing of complete test cases.

An important aspect to notice is that we have performed several experiments on different case studies and compared their results. We are in much confidence that our approach is



Sr. No	X-Machines Test Cases	Input Sequences	Output Sequences	Error
0	<OPEN>	<open>	<DOpen>	Non Faulty
1	<RUN>	<run>	<Work>	Non Faulty
2	<RUN TOUT>	<run,tout>	<Work,DClose>	Non Faulty
3	<RUN INTRRUPT>	<run,interrupt>	<Work,DOpen>	Non Faulty
4	<RUN INTRRUPT CLOSE>	<run,interrupt,close>	<Work,DOpen,DClose>	Non Faulty
5	<TSET OPEN>	<tset,open>	<Clock,DOpen>	Non Faulty
6	<TSET RUN>	<tset,run>	<Clock,Work>	Non Faulty
7	<OPEN CLOSE>	<open,close>	<DOpen,DClose>	Non Faulty
8	<OPEN CLOSE OPEN>	<open,close,open>	<DOpen,DClose,DOpen>	Non Faulty
9	<OPEN CLOSE RUN>	<open,close,run>	<DOpen,DClose,Work>	Non Faulty
10	<OPEN TSET CLOSE>	<open,tset,close>	<Dopen,Clock,DClose>	Yes Faulty
11	<RUN TOUT OPEN>	<run,tout,open>	<Work,DClose,DOpen>	Non Faulty
12	<RUN TOUT RUN>	<run,tout,run>	<Work,DClose,Work>	Non Faulty
13	<RUN TSET OPEN>	<run,tset,open>	<Work,Clock,DOpen>	Non Faulty
14	<RUN TSET TOUT>	<run,tset,tout>	<Work,Clock,DClose>	Non Faulty
15	<RUN TSET INTERRUPT>	<run,tset,interrupt>	<Work,C lock,DOpen>	Non Faulty
16	<CLOSE>	<close>	<DClose>	Non Faulty
17	<TOUT>	<tout>	<DClose>	Non Faulty
18	<INTERRUPT>	<interrupt>	<DOpen>	Non Faulty
19	<RUN OPEN>	<run,open>	<Work,DOpen>	Non Faulty
20	<RUN CLOSE>	<run,close>	<Work,DClose>	Non Faulty
21	<RUN RUN>	<run,run>	<Work,Work>	Non Faulty
22	<RUN INTRRUPT OPEN>	<run,interrupt,open>	<Work,DOpen,DOpen>	Non Faulty
23	<RUN INTRRUPT RUN>	<run,interrupt,run>	<Work,DOpen,Work>	Non Faulty
24	<RUN INTRRUPT TOUT>	<run,interrupt,tout>	<Work,DOpen<DClose>	Non Faulty
25	<RUNINTRRUPTINTRRUPT>	<run,interrupt,interrupt>	<Work,DOpen,DOpen>	Non Faulty
26	<TSET CLOSE>	<tset,close>	<Clock,DClose>	Non Faulty
27	<TSET TOUT>	<tset,tout>	<Clock,DClock>	Non Faulty
28	<TSET INTERRUPT>	<tset,interrupt>	<Clock,DOpen>	Non Faulty
29	<OPEN OPEN>	<open,open>	<DOpenDOpen>	Non Faulty
30	<OPEN RUN>	<open,run>	<DOpen,Work>	Non Faulty
31	<OPEN TOUT>	<open,tout>	<DOpen,DClose>	Non Faulty
32	<OPEN INTTRUPT>	<open,interuppt>	<DOpen,DOpen>	Non Faulty
33	<OPEN CLOSE CLOSE>	<open,close,close>	<DOpen,DClose,DClose>	Non Faulty
34	<OPEN CLOSE TOUT>	<open,close,tout>	<DOpen,DClose,DClose>	Non Faulty
35	<OPEN CLOSE INTERRUPT>	<open,close,interrupt>	<DOpen,DClose,DOpen>	Non Faulty
36	<OPEN TSET OPEN>	<open,tset,open>	<DOpen,Work,DOpen>	Yes Faulty
37	<OPEN TSET RUN>	<open,tset,run>	<DOpen,Clock,Work>	Yes Faulty
38	<OPEN TSET TOUT>	<open,tset,tout>	<DOpen,Clock,D.Close>	Yes Faulty
39	<OPEN TSET INTERRUPT>	<open,tset,interrupt>	<DOpen,Clock,D.Open>	Yes Faulty
40	<RUN TOUT CLOSE>	<run,tout,close>	<Work,DClose,DClose>	Non Faulty
41	<RUN TOUT TOUT>	<run,tout,tout>	<WorkDClose,DClose>	Non Faulty
42	<RUN TOUT INTERRUPT>	<run,tout,interrupt>	<Work,DClose,DOpen>	Non Faulty
43	<RUN TSET CLOSE>	<run,tset,close>	<Work,Clock,DClose>	Non Faulty
44	<RUN TSET RUN>	<run,tset,run>	<Work,Clock,Work>	Non Faulty

Fig. 4. Table for fault detection (complete test cases)



Microwave Oven Case Study	*Total no of test cases*	*Operation error*	*Transfer error*	*Extra state error*	*Missing state error*
X-Machines testing With Wp Method	*45*	×	✓	×	×
Proposed methodology	*23*	×	✓	×	×
✓ *Types of error exist in the implementation.*					
× *Types of Error not exist in the implementation.*					

Fig. 5. A comparison table with transfer error

consistent with the results. The detailed technical work can be found [24].

It is important to note that our approach is limited to X-Machines test cases only. It is possible to apply this frame work to other testing approaches by designing a criteria to treat test cases as clusters.

X. Conclusion and Future Work

In this paper we have presented a framework for the reduction of test case set generated by formal testing approach X-Machines. This proposed work is based on the statistical approach Random Cluster Sampling. Through the running case study of Microwave oven it is shown that input data is

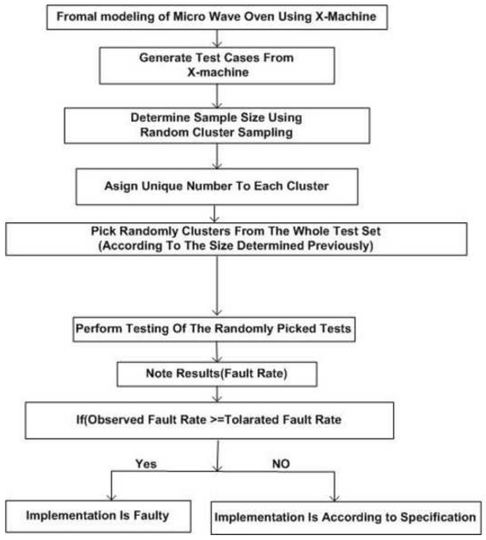

Fromal modeling of Micro Wave Oven Using X-Machine
Generate Test Cases From X-machine
Determine Sample Size Using Random Cluster Sampling
Asign Unique Number To Each Cluster
Pick Randomly Clusters From The Whole Test Set (According To The Size Determined Previously)
Perform Testing Of The Randomly Picked Tests
Note Results(Fault Rate)
If(Observed Fault Rate >=Tolarated Fault Rate
Yes
NO
Implementation Is Faulty
Implementation Is According to Specification

Fig. 6. Methodology for proposed work

successfully reduced by applying this frame work. We have shown that testing of the sample taken from X-Machines test case set represents the accuracy and reliability of whole population (complete test case set). Ultimately it reduces efforts, save time and cost to test complete test case set generated by X-Machines testing method. In future, we aim to apply statistical approaches to other model based testing techniques available in the literature.

References

[1] S.Patwa, A.K.Malviya, "Testability of software systems". International Journal of Research and Reviews in Applied Sciences, Vol: 5, 2010.

[2] Research Triangle Institute, NIST Planning Report02-3: "The Economic Impacts of Inadequate Infrastructure for Software Testing", March 5, 2003.

[3] H. Cichos, T. Heinz, "Effecient Reduction of Model-Based Generated Test Suites Through Test Case Pair Prioritization" Model-Driven Engineering, Verification, and Validation (MoDeVVa),2010.

[4] S. Eilenberg, "Automata, Machines and Languages". Vol. A. Academic Press, 1974.

[5] M. Holcombe, "X machine as a basis for dynamic system specification". Software Engineering Journal, Vol.3.No.2, pp. 69-76, 1988.

[6] K. Bogdanov, "Testing from X-Machines Specifcations". Formal Methods and Testing, Springer-Verlag Berlin Heidelberg. Pg:184-208, 2008.

[7] K. Bogdanov, M. Holcombe, F. Ipate, L. Seed, S. Vanak, "Testing Methods for X Machines: a review". Formal Aspects of Computing, pg:3-30, 2006.

[8] W. Chochran, "Sampling Techniques 3rd Edition, ISBN 978-0-471-16240-7, 1977.

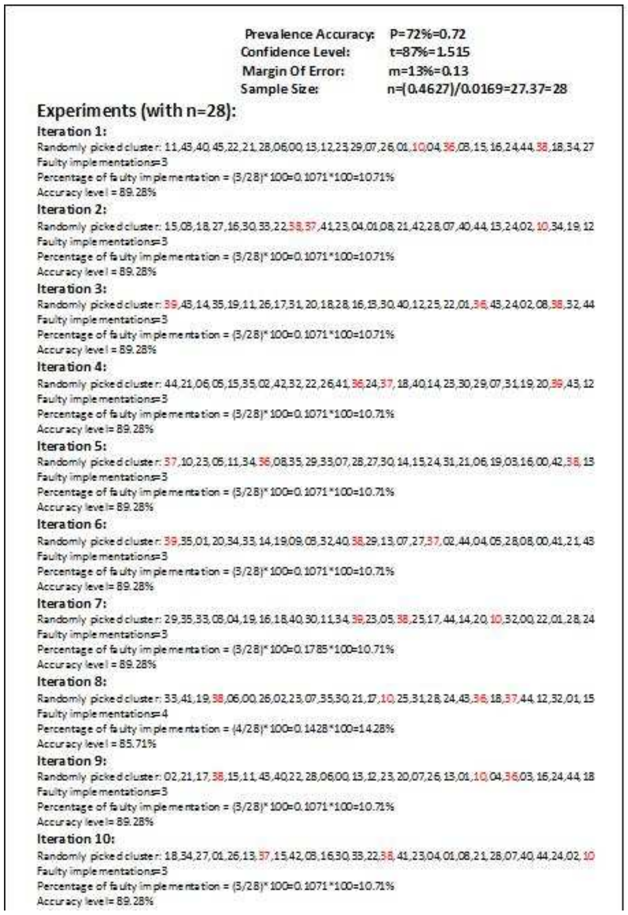
[9] A. Podgurski, W. Masri, Y. McCleese, F. Wolff, C. Yang, "Estimation of software reliability by stratified sampling". ACM Transactions on Software Engineering and Methodology, Pg:263-283, 1999.

[10] A. Podgurski, C. Yang, "Partition testing, stratified sampling, and cluster analysis". Proceedings of the First ACM Symposium on Foundations of Software Engineering, ACM Press, Pg:169-181, 1993.

[11] X. Ma, B. Sheng, C. Ye, "Test-Suite Reduction Using Genetic Algorithm", In Proceedings of APPT, pp.253-262, 2005.

[12] L. Raamesh, "An Efficient Reduction Method for Test Cases", International Journal of Engineering Science and Technology, Vol. 2(11), 6611-6616, 2010.

[13] N. Pan, F. P. Zeng, Y. H. Huang, "Test Case Reduction Based on Program Invariant and Genetic Algorithm", Wireless Communications Networking and Mobile Computing (WiCOM), 6th International Conference September, Chengdu, 2010.



Prevalence Accuracy:	P=72%=0.72
Confidence Level:	t=87%=1.515
Margin Of Error:	m=13%=0.13
Sample Size:	n=(0.4627)/0.0169=27.37=28

Experiments (with n=28):

Iteration 1:
Randomly picked cluster: 11,43,40,45,22,21,28,06,00,13,12,23,29,07,26,01,10,04,36,03,15,16,24,44,38,18,34,27
Faulty implementations=3
Percentage of faulty implementation = (3/28)*100=0.1071*100=10.71%
Accuracy level = 89.28%

Iteration 2:
Randomly picked cluster: 15,03,18,27,16,30,33,22,38,37,41,23,04,01,08,21,42,28,07,40,44,13,24,02,10,34,19,12
Faulty implementations=3
Percentage of faulty implementation = (3/28)*100=0.1071*100=10.71%
Accuracy level = 89.28%

Iteration 3:
Randomly picked cluster: 39,43,14,35,19,11,26,17,31,20,18,28,16,13,30,40,12,25,22,01,36,43,24,02,08,38,32,44
Faulty implementations=3
Percentage of faulty implementation = (3/28)*100=0.1071*100=10.71%
Accuracy level = 89.28%

Iteration 4:
Randomly picked cluster: 44,21,06,05,15,35,02,42,32,22,26,41,36,24,37,18,40,14,23,30,29,07,31,19,20,39,43,12
Faulty implementations=3
Percentage of faulty implementation = (3/28)*100=0.1071*100=10.71%
Accuracy level= 89.28%

Iteration 5:
Randomly picked cluster: 37,10,23,05,11,34,36,08,35,29,33,07,28,27,30,14,15,24,31,21,06,19,03,16,00,42,38,13
Faulty implementations=3
Percentage of faulty implementation = (3/28)*100=0.1071*100=10.71%
Accuracy level= 89.28%

Iteration 6:
Randomly picked cluster: 39,35,01,20,34,33,14,19,09,03,32,40,38,29,13,07,27,37,02,44,04,05,28,08,00,41,21,43
Faulty implementations=3
Percentage of faulty implementation = (3/28)*100=0.1071*100=10.71%
Accuracy level= 89.28%

Iteration 7:
Randomly picked cluster: 29,35,33,03,04,19,16,18,40,30,11,34,39,23,05,38,25,17,44,14,20,10,32,00,22,01,28,24
Faulty implementations=3
Percentage of faulty implementation = (3/28)*100=0.1785*100=10.71%
Accuracy level = 89.28%

Iteration 8:
Randomly picked cluster: 33,41,19,38,06,00,26,02,23,07,35,30,21,17,10,25,31,28,24,43,36,18,37,44,12,32,01,15
Faulty implementations=4
Percentage of faulty implementation = (4/28)*100=0.1428*100=14.28%
Accuracy level = 85.71%

Iteration 9:
Randomly picked cluster: 02,21,17,38,15,11,43,40,22,28,06,00,13,12,23,20,07,26,13,01,10,04,36,03,16,24,44,18
Faulty implementations=3
Percentage of faulty implementation = (3/28)*100=0.1071*100=10.71%
Accuracy level= 89.28%

Iteration 10:
Randomly picked cluster: 18,34,27,01,26,13,37,15,42,03,16,30,33,22,38,41,23,04,01,08,21,28,07,40,44,24,02,10
Faulty implementations=3
Percentage of faulty implementation = (3/28)*100=0.1071*100=10.71%
Accuracy level= 89.28%

Fig. 7. Table for Experiments(28 test cases)

[14] P. Pringsulaka, J. Daengdej, "Coverall algorithm for test case reduction", In Aerospace Conference, 2006.

[15] D. Banks, W. Dashiell, L. Gallagher, C. Hagwood R. Kacker, S. Rosenthal, "Software Testing by Statistical Methods - Preliminary Success Estimates for Approaches Based on Binomial Models, Coverage Designs, Mutation Testing, and Usage Models". NIST Interagency/Internal Report (NISTIR) - 6129, 1998.

[16] S. Poulding, J. Clark, "Efficient Software Verification: Statistical Testing Using Automated Search". IEEE trans. Software Eng., Vol. 36, 2010.

[17] W. Dickinson, D. Leon, A. Podgurski, "Finding failures by cluster analysis of execution profiles". In Proceedings of the 23rd International Conference on Software Engineering, Pg.339-348, 2001.

[18] L. Cesin, D. Zaen, "Profile analysis techniques for observation-based software testing". Ph.D thesis, Case Western Reserve University, 2005.

[19] S. Yan, Z. Chen, Z. Zhao, C. Zhang, Y. Zhou, "A Dynamic Test Cluster Sampling Strategy by Leveraging Execution Spectra Information". Third International Conference on Software Testing, Verification and Validation, Pg:147-154, 2010.

[20] D. Cooper, P. Schindler, "Marketing Research". Special Indian Edition, ISBN 0-07-060091-0, 2006.

[21] W. Athur, "Sampling Statistics", Latest Edition, 2009.

[22] R. Fisher, F. Yates, "Statistical tables for biological, agricultural and medical research". 6th Edition. London: Oliver and Boyd. Pg:26-27, 1963.

[23] M. Holcombe, F. Ipate, "Correct Systems: Building a Business Process Solution". Springer Verlag, London, 1998.

[24] https://sites.google.com/site/softwareengineeringpk/my-forms

ACKNOWLEDGMENT

The authors would like to thank Respected Dr. Kiril Bogdanov Department of Computer Science Sheffield University UK for his kind attitude, support, valuable motivation and guidance. Without his guidance this research was not possible.

This work has been supported by the National Research Fund, Luxembourg, Project MOVERE, ref.C09/IS/02