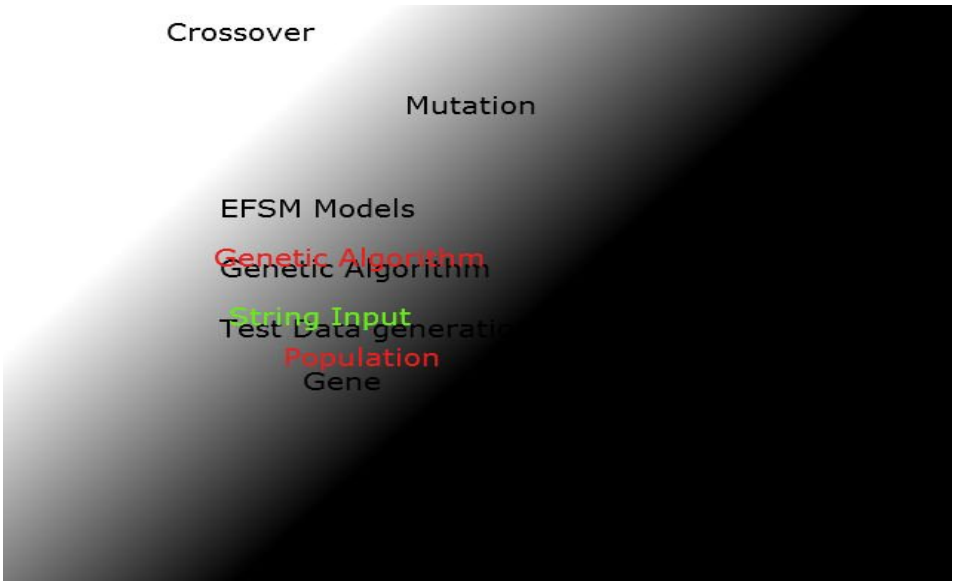
UNIVERSITY OF GOTHENBURG



An evolutionary testing approach for test data generation from EFSM model with string data input

Master of Science Thesis in the Programme of Computer Science

Diponkar paul

University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, October 2009

An evolutionary testing approach for test data generation from EFSM model with string data input

Diponkar Paul

Examiner: K.V.S Prasad
Supervisor: Prof. Robert Hierons[1]

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden October 2009

[1] Professor of Computing, School of Information Systems, Computing and Mathematics, Brunel University, Uxbridge, Middlesex, UK.

Abstract:

The thesis has aimed to test data generation from EFSM model with string data input. In testing area a very few work is done to generate test data with string data input. So this topic is interesting to the testing arena. To reach the goal a genetic algorithm (GA) tool is developed. A study was carried out to choose the best fitness function for string data input; resulting modified edit distance algorithm was used as a fitness function. Firstly, string and alphanumeric values with different lengths were passed through the GA tool and evaluated the result. Then three EFSM models were designed and deployed to the GA tool where most of cases the whole path is passed. This work was limited to string equality and there is a scope to work with string ordering in future.

Keywords: Evaluation algorithm, test data generation, fitness function, Extended Finite State Machine (EFSM).

Acknowledgment

First, I would like to thanks to my supervisor Professor Robert Hierons, Professor of Computing at Brunel University. His eagerness and continual time to time support throughout the thesis made possible to finish my thesis on time. I am grateful for his valuable supervision and friendly approach.

I would like express my gratitude to Associate Professor KVS Prasad. He is my examiner from Chalmers University of Technology. He gave his valuable guidance and supported me before starting the project and during the project work.

Furthermore, I would like to thanks to Abdulsalam Kalaji, for his valuable supports and comments during the development of the system and writing the report.

I would like to spread my thanks to my family and friends who helped me in a different way to accomplish the task successfully.

Table of Contents

1. Introduction:

Investigation shows that more than 50 % of a total software cost is spent for the purpose of testing. Since manual testing is time consuming and expensive, it is important that the practice of testing being automated. Being attached to the cost, automating the process of testing is a business concern now-a-days [1-4]

Incase of any system implementation; testing is necessary to check whether the implementation is match with the system specification. This can be achieved through conformance testing where a tester can find out the differences in the behavior between the implemented system and its specification.

According to KVS Prasad [5], “A specification of a (sequential) system is an explicit or implicit listing of input/output pairs. For any given input, it says what the output should be. An example is "given a positive real number y, find a positive real number x such that x*x = y." This says what property the output should have (it is the square root of the input), but it does not tell you how to find the square root. Indeed the strength of this kind of specification is that it can be satisfied by many programs. Another example of a specification is a set of use cases. For any specification, but particularly for use-cases, it is reasonable to ask if the specification is complete (does it say what to do for any possible input?) and consistent (it should not specify more than one output for any given input), and indeed such validation of the specification is part of rigorous software development.

Conformance testing is to see to what extent it meets its specification: for each of a set of test inputs, see what that the implementation does, and compare it with what the specification says the system should do. Since the number of possible inputs to the system is usually very large, we can only choose some inputs to test. Which ones? Ideally we would like to pick the tests that give the most information about the system - i.e., that cover, in some sense, as much of the input spsce as possible.

Now for a specification given as a logic relation between input and output, it is very hard to say which tests give the most information.about the program under test. To do so would need some kind of theory about the space of inputs.An alternative approach is to use a program or system design as a specification. This is common in the hardware industry, where the specification can be a working circuit, called the reference system. This is a system taken to be obviously correct but unusable as an implementation because it is perhaps too slow or uses too much power. The implemented system is typically fast and power efficient but too complicated to be taken as correct. Testing then compares the implementation with the reference system. So the specification is treated as a white box and the implementation as a black one (its designis not visible to the tester, only its input/output behaviour). Similar approaches are possible with software, with the specification being in effect a very high level program.”

In order to apply the conformance testing, it is useful to represent a given system specification in terms of a model. This model is used then to derive test sequences that can be applied to the implementation under that test. This thesis studies the problem of generating input test sequences from systems that have strings inputs and modelled as extended finite state machine (EFSM).

In this project three EFSM models are designed and passed to the GA tool to generate the test data for each model. The EFSM models those are implemented in the project as follows: 1) Typical log on system for internet banking 2) retrieve forgotten password of internet application 3) report student's grade to the Ladok[2].

2. Problem Area:

Though there are many ways to generate automatic test data but little attention is paid to generate test data for program that include character string predicates [6].To meet the challenges for generating test data for string data needs to consider the following:

- Choose an optimization algorithm for automatic test data generation.
- Define a fitness function that suits for string predicates.
- With the help of cost function; generate test data that can satisfy a set of predicates.

Generally test data generation is limited to the programs in which predicate compare numbers, e.g.

```
If (z ==20) {
//target
}
```

For the above predicate, a typical cost function is the absolute difference between *z* and 20. This cost function is appropriate for a single numerical value. On the other hand, for string equality such cost function is worthless. There are some string matching algorithm e.g. edit distance, binary hamming distance which may be useful for defining cost function for string data[7].

3. Methodology

This section describes the steps of the design and implementation of the proposed approach.

3.1. Pre-study:

The literature review step helps me to understand the problem domain deeply. Different research papers those are relate to search based software engineering more specifically search based software test data generation helped me to realize the problem area in a broader scope. Firstly, gained the knowledge of test data generation then came to search based software test data generation. In test data generation area there are other approaches beside search based; for instance model driven software test data generation. Most of the study covered within search based test data generation area. Generally metaheuristic technique such as hill climbing, Simulated Annealing and evaluation algorithm are uses in search based software test data generation [8]. I found evolutionary algorithm is better than other techniques. Mainly genetic algorithm is used in the project as an evolutionary algorithm. Then, study of genetic algorithm produced concrete idea to develop my own tool to generate automatic test data.

[2] Ladok is a study administrative system used by Swedish universities and colleges for documentation purposes

Beside, every meeting with my supervisor in the period of the project helps me to understand the topic more clearly. While reading a lot of paper on test data generation area aware me what is done in this domain and what scope is waiting for the future?

3.2. Design:

This phase aimed to result a design of the prototype that was later to be implemented.
After brain storming session a prototype is designed for the project where choosing best algorithm was critical job and thereafter a tool is designed from chosen algorithm. Next step was to choose optimize cost function that is fit for the problem domain. And finally to choose EFSM model that will passed to the developed tool and carried out the expected output.

3.3. Implementation:

In this phase implementation is done according to the prototype. It is needed to check that prototype is designed correctly and feasible for the project. During the implementation of prototype several meeting held with my supervisor resulting small changes of the prototype. The project is not divided into small parts against the time slots because of simply the project is not so big.

3.4. Evaluation:

The goal of evaluation is how the prototype supports the problem definition and gives the desired output. And then conclusion is drawn and future guide line is provided after the evaluation. The cost function is reviewed and modified to get better result.

4. Literature Review:

4.1. Genetic Algorithm:

A genetic algorithm (GA) is an optimization technique used in computing to find exact or approximate solutions to search problems. In early 1970s genetic algorithm (GA) is introduced by John H. Holland where he implemented the natural selection theory to present a powerful, simple and sturdy method that can be used in dealing with optimization problems (Bayer and Liu,1991). The idea of using genetic algorithms has emerged from the observation of life and evaluation. Life reproduces and evolves by exchanging DNA information to produce mixture of character. Biologically, there are few terms that are used for GA:

- Genome: A sequence of values (could be characters, floats, integer etc).
- Chromosome: A collection of gene e.g. string “HELLO” is collection of characters.
- Population: A group of chromosome in the generation.

There are a set of operators those are used by GA such as

- Fitness: A value that describes how ‘good’ a given chromosome is.
- Crossover: An operator that a GA uses for simulating ‘reproduction’.
- Mutation: When a piece of data in a chromosome is altered randomly.

Genetic search process is iterative: evaluating, selecting, and recombining strings in the population is occurred in the each iteration (generation) until reaching some termination condition. The following steps come sequentially in genetic algorithm operation.

a) Evaluation
b) Selection
c) Crossover
d) Mutation

a) Evaluation:
The first step of GA is evaluation. This step evaluates the fitness of each chromosome.

b) Selection:
Selection is used to find two individuals that will be mated to contribute for the next generation. Two individuals are selected randomly, but each individual's probability of being chosen is proportional to its fitness. This is known as roulette wheel selection. So the selection is done on the basis of qualified fitness [4].

c) Crossover:
The third important step is crossover where two qualified individual is exchanged their genes. Pairs of strings are picked at random from the population to be subjected to crossover. There many ways to conduct crossover such as single point crossover and multipoint crossover. Single point crossover is very simple where a crossover point is randomly selected and then the selected two strings are divided at the crossover point and then do crossover and produce two new offspring like as below:

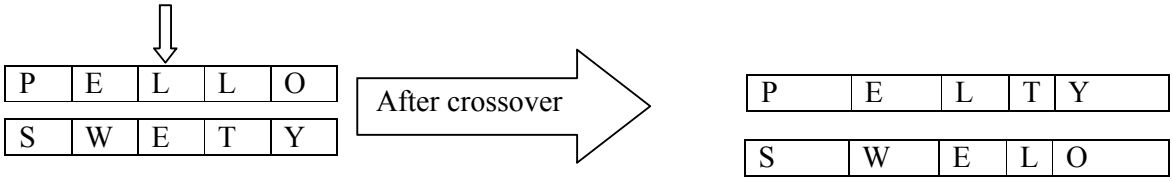


Fig: Single point crossover

d) Mutation:
After crossover, strings are subjected to mutation, mutation mean flip one bit; e.g. changing 0 to 1 or vice versa. Not every gene in individual will be flipped; the gene will be flipped when mutation rate, give the probability that allow to do so. The GA treats mutation only as a secondary operator with the role of restoring lost generic material. For example, let all the strings in a population converged to 0 at a given position then crossover can not give you the value 1; only mutation can give the different value than 0 [9].

4.2. Finite State Machine (FSM) Fundamentals:

A finite state machine is one that has a limited or finite number of possible states. For example in a model of computation consisting of a set of states; a start state, an input alphabet and a transition function that maps input symbols and current state to the next state [11]. Bran Selic & Garth Gullekson in the book *Real-time Object-oriented Modeling*, view a state machine as:

- A set of input events
- A set of output events
- A set of states
- A function that maps states and input to output
- A function that maps states and inputs to states (which is called a state transition function)
- A description of the initial state

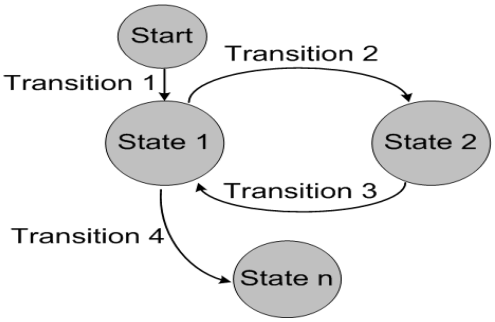


Figure: A typical example of FSM.

4.3. Extended Finite State Machine (EFSM) Fundamentals:

According to Network Dictionary[3], "Extended Finite State Machine Model (EFSM) is an enhanced model based on the traditional finite state machine (FSM), which is a model of behavior composed of states, transitions and actions."[11]

An FSM model can successfully represent control part of a system but if the system has control and data part then it needs EFSM model to represent successfully. For example

The EFSM model is a 6-tuple [4] (S, $s0$, V, I, O, T) where:

- S is the finite set of logical states
- $s0 \in S$ is the initial state
- V is the finite set of internal variables
- I is the set of input declarations
- is the set of output declarations
- T is the finite set of transitions

The transition $t \in T$ is represented by the 5-tuple (s_s, i, g, op, s_e) in which:

[3] http://www.networkdictionary.com/

- s_s is the start state of t
- i is the input where $i \in I \cup \{\text{Nil}\}$
- g is the guard and is either Nil or is represented as a set of logical expressions given in terms of variables in V' where $\Phi \neq V' \subseteq V$
- op is the sequential operation which consists of simple statements such as output statements and assignment statements

- s_e is the end state of t.

EFSM model consist of set of variables e.g. state variables and context variables. The state variable is used to represent the state of a finite state machine such as idle, wait for connection, connection opened and so on. On the other hand, port number, sequencing numbers, data to transfer etc are stored in context variables. Each state transition is associated with a head state, a tail state, and a label that represents a transition. The execution of a transition is atomic. A transition consists of two parts: the condition part and the action part. The condition part can contain 1) a input event, 2) predicate, which is a Boolean expression, 3) a time clause, which is represented as delay and 4) a priority clause, which is represented as **"**priority scale." The action part can contain output events and a number of statements that operate on variables. The time clause identifies when the transition can be executed if the transition is executable.

4.4. Cost/fitness function for String equality

The most used string matching algorithm in respect of finding the distance value between two strings are Binary hamming distance, Character distance and Edit distance[8]. Different application needs different type of cost function to satisfy their goal, for instance spell checker adopted algorithm that can gives output after comparing with user given words and database stored dictionary's words.

4.4.1. Binary Hamming distance

Hamming distance is defined as "the minimum number of bits that must be changed in order to convert one bit string into another". There are many cases where genetic algorithm uses the binary hamming distance as a cost function. Character strings vary in different length so that hamming distance must be considered the unequal length strings. Comparison of unequal length of string needs consideration of the cost of inserting or deleting characters. Strings are left aligned that can produce extra cost for comparing two strings, For example, the HD ("KHELLO","HELLO") where only one character 'L' is matched consequently cost is high. When two strings are equal then hamming distance can give better performance e.g. HD ("HELLP","HELLO") where cost is low i.e. 1.

4.4.2. Edit Distance Algorithm

For information retrieval and biological matching use different string comparison metrics whereas vast majority string comparison metrics is derived from edit distance. The edit distance is resulting explicitly from consideration of three operators that execute character

insertion, character deletion and character substitution. The edit distance between two strings is the minimum distance of two strings where the lengths difference between two strings never misguides to carry out the exact result. Generally, minimum number of deletion, insertion or substitution is required to transform one string into other.
For example, ED (TEAM, TEB) =2 because one insertion (or one deletion) and one substitution is necessary to match the two strings.

Considering the suitability of the edit distance as a cost function, the size of the range is important. The range of edit distance value is equal to the maximum length of the two strings compared. [8]

5. The proposed approach:

The proposed approach is divided into some smaller parts:

1) Choose Genetic algorithm (GA) as a serach based test data generator.
2) Develop GA tool.
3) Define a cost function.
4) Generate test data for string input by using the developed GA tool.
5) Pass EFSM model to the GA tool.

6. Approach implementation and experiment:

6.1. Development and deployment of GA tool:

Design, coding, testing and implementation part of GA tool is described under this section.

6.1.1. Design:

This phase aimed to result a design of the prototype that was later to be implemented.
Basic software design principles are followed, such as ease of maintenance, low coupling and easy extensibility. Firstly, control flow graph is designed for the whole system then fragmented it into smaller parts. Secondly, class diagram is design for the genetic algorithm. Basic structure of GA is described below then initial population, selection, crossover and mutation process is shown in flow diagram.

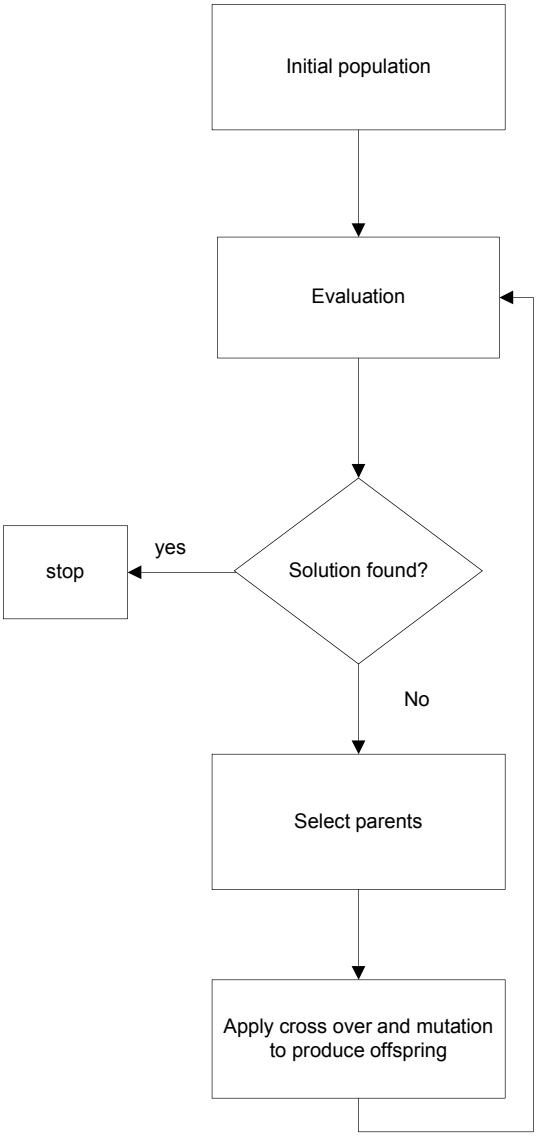


Figure 1: Flowchart of genetic algorithm

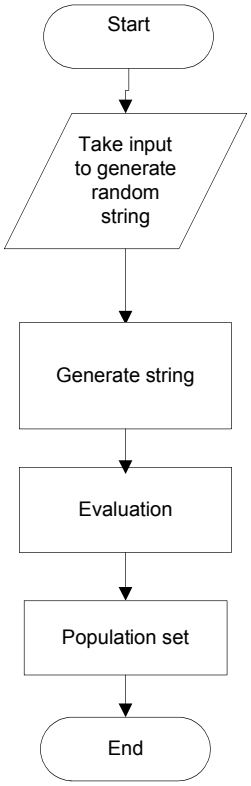


Figure 2:
Initial Population

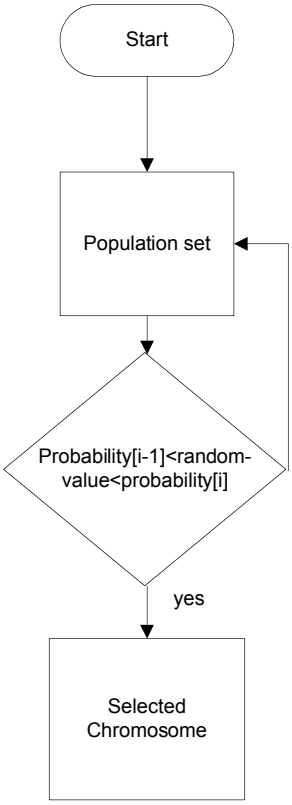


Figure 3:
Selection

Initial population is generated according to user needs, for example if user wants to get the strings containing five characters, one hundred population and thousand generations then user needs to give the inputs set to produce desire population. As figure 1 describe the flow diagram of initial population where at first user gives all necessary input then string is generated and then evaluation takes place to evaluate every single string and set of population is arranged in ascending order according to their fitness. Selection is the second step of GA where linear ranking selection method is chosen to select the strings. As figure 2 shows the steps for selection of chromosomes, here probability means linear ranking value. The linear value is calculated by using the linear ranking formula as follows: P (T) = (2-SP/M)+(2* i *(sp-1)/M(M-1)).
Where:
SP: is the selective pressure and ranges from [1.00-2.00]
M: is population size
i: is the position of chromosome on the rank where the least fit chromosome has i=1 and the fittest chromosome has i=M.

The most important part in GA is crossover and mutation; where in crossover pairs of strings are chosen and cross over them at randomly chosen cross over point. Mutation is treated in GA as an operator that can restore the lost genetic material. For example, suppose

all strings in a population converged to 0 at a given position but optimum solution is 1 at that position. Then crossover cannot regenerate a 1 at that position, while a mutation could.

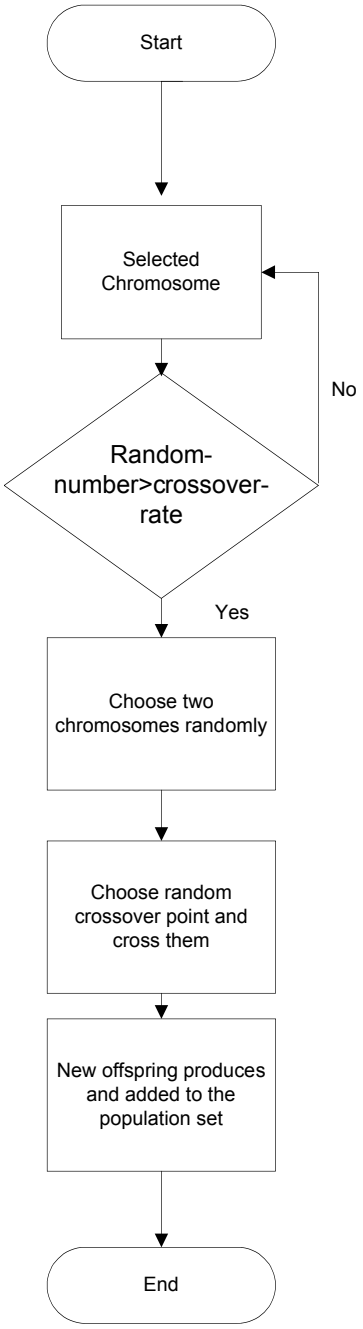


Figure 4: Crossover

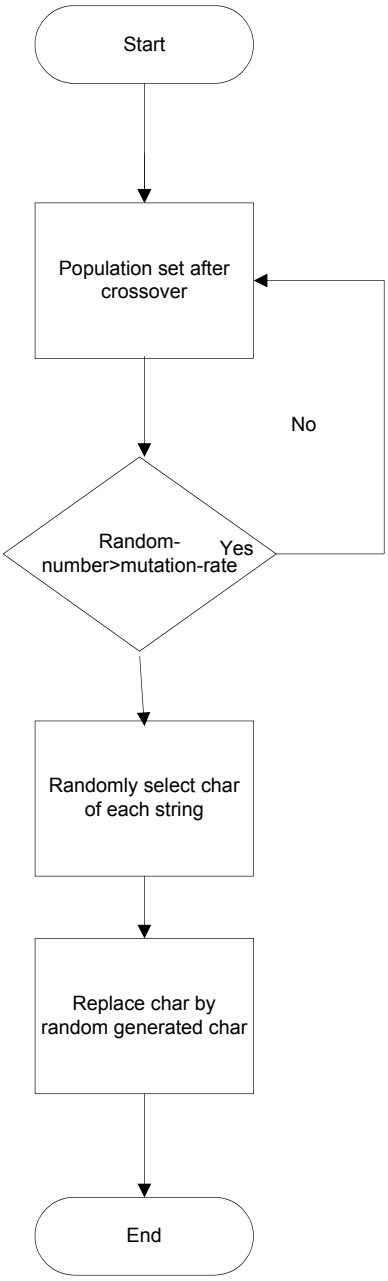


Fig 5: Mutation

Figure 4, shows all steps to complete the crossover operation and figure 5 shows the steps of mutation operation.

Three EFSM models are designed to pass through GA tool and evaluate to see the performance. First EFSM model is designed from customer's log in part of typical internet banking application. For example, typical online banking application's log on page required

user id or pass code, birth date and memorable word to successfully log on to the system. Second EFSM model is chosen from the idea of password retrieval procedure of web application. As an example; to retrieve password you requires email id, user id and memorable words. Third EFSM model is for reporting student's grade to the Ladok[4]. When student's grade is reported to the Ladok the reporter needs to check the student name, personal number and the grade that student earned. The three EFSM models are depicted in Figure 6, 7 and 8 below:

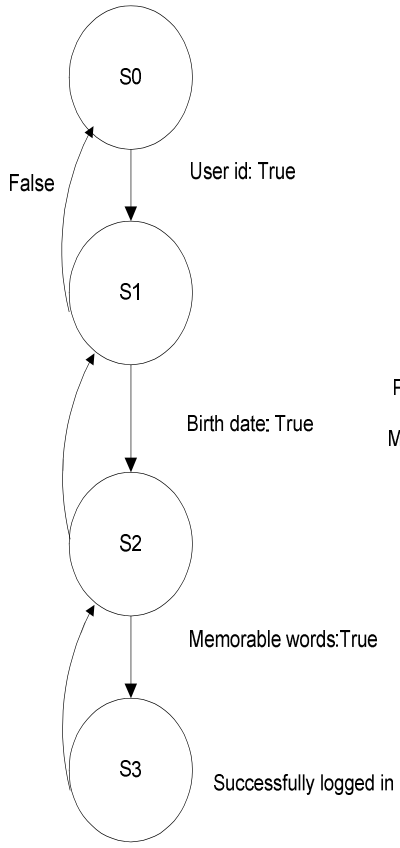


Fig 6: EFSM model for typical log on system for internet banking

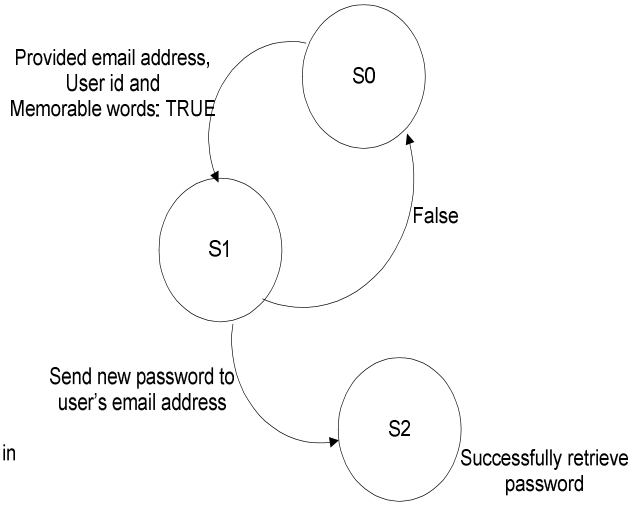


Fig 7: EFSM model for retrieving password

[4] Ladok is a study administrative system used by Swedish universities and colleges for documentation purposes

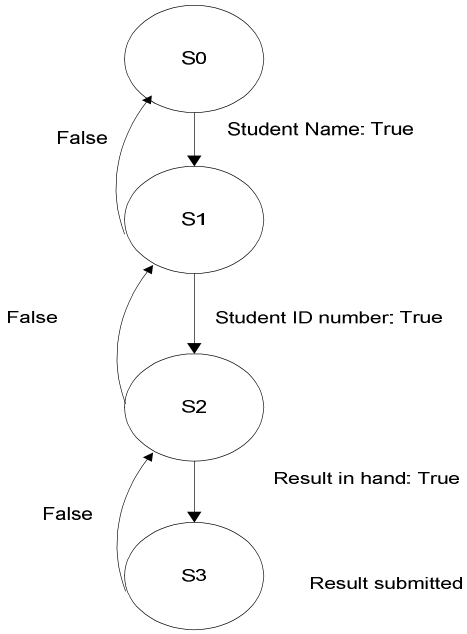


Fig 8: EFSM model for submitting student result to the LADOK*

*Ladok is a study administrative system used by Swedish universities and colleges for documentation purposes.

6.1.2. Coding and Testing:

According to UML class diagram of GA coding part is accomplished with the help of C# language in an integrated development environment named Microsoft visual studio. Class diagram is not included in the report. As a cost function Edit distance function is used after modification. Edit distance function is coded according to the following algorithm:

Step	Description
1	Set n to be the length of s. Set m to be the length of t. If n = 0, return m and exit. If m = 0, return n and exit. Construct a matrix containing 0..m rows and 0..n columns.
2	Initialize the first row to 0..n. Initialize the first column to 0..m.
3	Examine each character of s (i from 1 to n).
4	Examine each character of t (j from 1 to m).|
5	If s[i] equals t[j], the cost is 0. If s[i] doesn't equal t[j], the cost is 1.
6	Set cell d [i, j] of the matrix equal to the minimum of: a. The cell immediately above plus 1: d[i-1,j] + 1. b. The cell immediately to the left plus 1: d [i, j-1] + 1. c. The cell diagonally above and to the left plus the cost: d [i-1, j-1] + cost.

Code is tested before implementation. Conventional test approach is taken for white box and black box testing. If found any bug in code then went back to design and think again about the design and if necessary changes design and code again according to new design. The Rapid Action Development (RAD) model is considered to develop the tool.

6.1.3. Implementation:

In the first phase, different lengths of strings are passed to the GA tool and evaluate the result. Details result is described in the experiment section. In the second phase, three EFSM models are passed through the GA tool and evaluates the result as well as the result is described in experiment section.

6.2. Experiment:

Experiment section is divided into two parts : 1) Experiment with strings 2) Experiment with EFMS models. The result of experiment is contains in the table. The experiment is done with the help of evaluation version of Matlab GA tool named GEATbx [12] with the modified cost function. Since the Matlab GA tool was performed better than developed GA tool so that Matlab GA tool is used.

6.2.1. Experiment with strings:

Five strings are chosen with differents lengths of characters. Chosen characters are 1) Hello 2) Password 3) MYPASSWORD 4) password1990 and 5) Password£$%09-08. For each execution assumed that Population size is 100 and generation is 1000. Each table is constructed with the result of twenty (20) executions. The tables contain execution number, generations, best fitness value, time (in minutes) and status of matches. A summary table is constructed from the result of five tables.

The following table is constructed with the output of string value ‘Hello’

Number of execution	Generations	Best fitness value	Time(m)	Match
1	744	0	0.38	OK
2	392	0	0.20	OK
	160.	0	0.8	OK
4	373	0	0.19	OK
5	1000	1	0.57	NO

6	219	0	0.13	OK
7	803	0	0.47	OK
8	393	0	0.23	OK
9	658	0	0.35	OK
10	164	0	0.09	OK
11	588	0	0.34	OK
12	137	0	0.08	OK
13	225	0	0.14	OK
14	928	0	0.53	OK
15	69	0	0.04	OK
16	212	0	0.13	OK
17	304	0	0.16	OK
18	1000	1	0.51	NO
19	233	0	0.12	OK
20	168	0	0.09	OK

Table 1: result with string 'Hello'

From the above table it is clearly shown that the string 'hello' is matched almost every in each execution.

The following table is consrtuted with the output of string ‘Password’.

Number of execution	Generations	Best fitness value	Time(m)	Match
1	19	0	0.03	Ok
2	27	0	0.04	OK
3	528	0	0.81	OK
4	94	0	0.13	OK
5	57	0	0.9	OK
6	115	0	0.18	OK
7	33	0	0.06	OK
8	23	0	0.04	OK
9	48	0	0.08	OK
10	129	0	0.20	OK
11	173	1.01	0.24	OK
12	69	0	0.11	OK
13	24	0	0.03	OK
14	69	0	0.09	OK
15	57	0	0.08	OK
16	41	0	0.06	OK

17	1000	2	1.51	NO
18	120	0	0.16	OK
19	23	0	0.03	Ok
20	29	0	0.04	OK

Table 2: result with string 'password

Same like the string 'Hello', the string 'password' is also matched almost in each execution.

The following table is matches with string 'MYPASSWORD'

Number of execution	Generations	Best Fitness value	Time(M)	Match
1	120	0	0.25	OK
2	32	0	0.06	OK
3	33	0	0.07	OK
4	1000	2	2.09	NO
5	1000	2	2.08	NO
6	205	0	0.42	OK
7	152	0	0.32	OK
8	76	0	0.16	OK
9	1000	2	1.94	NO
10	81	0	0.15	NO
11	57	0	0.11	NO

12	1000	2	2.00	NO
13	158	0	0.32	OK
14	1000	2	1.81	NO
15	1000	2	1.81	NO
16	179	0	0.32	OK
17	95	0	0.17	OK
18	195	0	0.35	OK
19	279	0	0.50	OK
20	19	0	0.04	OK

Table 3: result with string 'MYPASSWORD'

From the result table it is clearly shown that the string 'MYPASSWORD' is not matched in each execution like the string 'Hello' and 'Password' matched. The reason behind this is 'MYPASSWORD'contains all capital characters and it is longer in length.

The following table contains the result of matches with string 'password1990'

Number of execution	Generations	Best Fitness value	Time(m)	Match
1	166	0	0.43	OK
2	339	0	0.97	OK
3	166	0	0.48	OK
4	274	0	0.80	OK
5	224	0	0.61	OK

6	133	0	0.34	OK
7	1000	2	2.55	NO
8	78	0	0.23	OK
9	121	0	0.36	OK
10	153	0	0.39	OK
11	28	0	0.07	OK
12	60	0	0.15	OK
13	1000	2	2.63	NO
14	1000	2	2.58	NO
15	64	0	0.16	OK
16	222	0	0.59	OK
17	155	0	0.51	OK
18	208	2	0.64	NO
19	203	0	0.50	OK
20	511	0	1.46	OK

Table 4: result with string 'password1990'

The alphanumeric value 'password1990' is matched most of the cases. Though it is not matched in every execution like 'Hello' and 'password'. But the success rate is lied on expectation level.

The following table contains the result of matches with string 'Password£$%09-08'

Number of execution	Generations	Best fitness value	CPU time	Match
1	1000	1.01	4.50	NO
2	1000	1.01	4.78	NO
3	1000	1.01	5.15	NO
4	1000	3.01	5.06	NO
5	1000	2	4.89	NO
6	1000	1.01	3.95	NO
7	1000	3.01	3.95	NO
8	1000	5.01	3.94	NO
9	1000	1.01	3.96	NO
10	1000	2	3.96	NO
11	1000	1.01	4.34	NO
12	1000	3.02	4.68	NO
13	1000	1.01	3.61	NO
14	1000	2	4.87	NO
15	1000	1.01	4.57	NO
16	1000	3.01	4.96	NO

17	1000	2	4.62	NO
18	1000	2	4.97	NO
19	1000	1.01	4.95	NO
20	1000	1.01	5.01	NO

Table 5: result with string 'Password£$%09-08'

As the result table shown that there is no matched in any single execution, it is assumed that the result is held because of long length string and alphanumeric values.

The summary table with all strings; result is given below:

String	Average generation	Average Time(m)	Match
Hello	420.6	0.211	Yes
Password	133.9	0.241	Yes
password1990	305.25	0.8225	Yes
MYPASSWORD	731.5238	0.7489	Yes
Password£$%09-08	1000	3.95	No

Table 6: Summary of the experiments

6.2.2. Experiment with EFSM models:

Three EFSM models such as 1) Typical log on system for internet banking 2) retrieve forgotten password for internet application 3) report student's grade to the Ladok are considered to pass into the GA tool and evaluate the result. For each execution generation is fixed to 2000 and population size is 100. After each execution result table is produced where generations, best fitness value, execution time and status of matches are included. Each table contains the result of ten(10) executions.

The following table is produced after passing the EFSM model of typical log on system for internet banking. When the whole path of the model is sucessfuly passed to the GA then best fitness value became 0.

Number of execution	Generations	Best fitness value	Time(m)	Match
1	2000	0.092971	1.47	NO
2	2000	0.092971	1.44	NO
3	447	0	0.43	OK
4	2000	0.092971	2.57	NO

5	2000	2.093	2.39	NO
6	680	0	0.62	OK
7	2000	0.092971	2.49	NO
8	644	0	0.67	OK
9	811	0	0.86	OK
10	475	0	0.45	OK

Table 7: result of the EFSM model 'a typical log on system for internet banking'

According to the result table, from ten executions five execution is sucessfully passed and rest five executions did not pass sucessfully. So there is 50% success rate when passing the EFSM model to the GA tool.

The following table is produced after passing the EFSM model for retrieving forgotten password of a typical internet application. When the whole path of the model is sucessfuly passed to the GA then best fitness value become 0.

Number of execution	Generations	Best fitness value	Time(m)	Match
1	2000	0.047619	3.11	NO
2	2000	0.13616	2.95	NO
3	2000	0.13616	3.25	NO
4	2000	0.47619	2.92	NO
5	1703	0	2.45	OK
6	2000	1.0481	3.22	NO
7	2000	1.093	3.29	NO
8	1200	0	1.94	OK

9	811	0	1.02	OK
10	2000	0.092971	3.34	NO

Table 8: result of the EFSM model 'retrieve forgotten password for internet application'

From the result table 8, it is shown that the success rate of the EFSM model is poor; and that is 33%. In the model; user name, email address and memorable words are included. The email address is bigger in length and contains alphanumeric value those are the mainly reason to produce the poor result.

The following table is produced after passing the EFSM model for reporting student's grade to the Ladok. When the whole path of the model is sucessfuly passed to the GA then best fitness value become 0.

Number of execution	Generations	Best fitness value	Time(m)	Match
1	2000	0.092971	3.23	NO
2	510	0	0.80	OK
3	1131	0	1.81	OK
4	2000	0.092971	3.03	NO
5	2000	2.093	3.11	NO
6	2000	0.092971	3.29	NO
7	709	0	1.18	OK
8	2000	1.093	3.20	NO
9	2000	1.093	2.90	NO
10	1089	0	1.74	OK

Table 9: result of the EFSM model 'report student's grade to the LADOK'

The result reported in table 9 is shown that, success rate of passing the model to the GA tool is 40% . The model is included student name, person number[5] and grade, the path has combination of string and numeric value.

The following table is summary of the result of EFSM models:

EFSM model	Average generation	Average Time(m)	Match
Sign in operation for Online banking	1305.7	1.339	Yes
Password retrieval process for internet application	1771.4	2.749	Yes
Report to the LADOK system	1543.9	2.429	Yes

Table 10: Summary of the experiments of EFSM models

From the summary table it is shown that every EFSM model is passed to the GA tool and average time is depend on the model's path length. And the more generation is taken if the model is complex where the simple model needs less generation to get the sucessful result.

[5] Swedish civic registration number

7. Conclusion and future work:

The evaluationary approach for test data generation is sucessfully implemented for string predicates and EFSM models. Though each execution did not give the sucessful result but result is graded as satisfactory level. Smaller length of string and shortest path of EFSM model gives better result than bigger string and larger path of EFSM model. However, there is a scope to future enhancement of the project. The propose future plan are following:

a) Extending the search space.
b) String ordering and regular expression could be included.
c) Improvement of the cost function.

a) extending the search space:
The search space is limited to characters ordinal value between 0 and 127 for this project. Since over 99% of characters are between the range of 0 and 127 so 16 bit character (0 to 255) is avoided in this work. But there is a scope to work in future with extending the search space. The search space could be considered with characters which has ordinal value between 0 to 255.

b) String ordering and regular expression could be included:
The present work is limited to the string equality; where string ordering and regular expression could be included when future cost function is defined.

c) Improvement of the cost function.
There is a scope to future improvement of cost function. After analyis of the result of experiments it is shown that longer length of string did not give good result compare with the shortest length of string. For example string ‘Hello’ matched almost in every execution, on the other hand, big length string ‘Password£$%09-08’ did not match any single time from the twenty executions. So there is a scope to modify the cost function so that present limitation can be overcome.

8. References:

[1]. Korel, B., *Automated software test data generation.* Software Engineering, IEEE Transactions on, 1990. **16**(8): p. 870-879.
[2]. McMinn, P., *Search-based software test data generation: a survey: Research Articles.* Software Testing, Verification & Reliability, 2004. **14**(2): p. 105-156.
[3]. Harman, M. *Automated Test Data Generation using Search Based Software Engineering.* in *Automation of Software Test , 2007. AST '07. Second International Workshop on.* 2007.
[4]. Michael, C.C., G. McGraw, and M.A. Schatz, *Generating software test data by evolution.* Software Engineering, IEEE Transactions on, 2001. **27**(12): p. 1085-1110.
[5]. K.V.S. Prasad, Associate Professor at the Department of Computing Science at the Chalmers University of Technology.
[6]. Kalaji, A., Hierons, R. M., and Swift, S. (2008), *Automatic Generation of Test Sequences form EFSM Models using Evolutionary Algorithms.* Technical report, School of Information Systems,Computing and Mathematics, Brunel University.
[7]. Ruilian Zhao, Michael R. Lyu, *Character String Predicate Based Automatic Software Test Data Generation*, qsic, pp.255, Third International Conference On Quality Software, 2003
[8]. Mohammad Alshraideh and Leonnardo Bottaci, *Search-based software test data generation for string data using program-specific search operators.* Software Testing, Verification & Reliability, 2006. **16**(2): p. 175-203.
[9] Srinivas, M. and Patnaik, L. M. 1994. Genetic algorithms: A Survey. *Computer* 27, 6 (Jun. 1994), 17-26.
[10].http://www.itl.nist.gov/div897/sqg/dads/HTML/finiteStateMachine.html (accessed on7th July, 2009)
[11].http://www.networkdictionary.com/software/e.php?PHPSESSID=9926acc9b2e4f8f05ced05a620abcfe9 (accessed on 15th July, 2009)
[12]. http://www.geatbx.com (accessed on 20th July 2009)