Practical Adoptions of T-Way Strategies for Interaction Testing

Kamal Z. Zamli[1], Rozmie R. Othman[2], Mohammed I. Younis[1], and Mohd Hazli Mohamed Zabil[2]

[1] School of Electrical Engineering, Universiti Sains Malaysia, Engineering Campus, Nibong Tebal 14300 Penang, Malaysia
[2] School of Computer and Communication Universiti Malaysia Perlis (UniMAP) PO Box 77, d/a Pejabat Pos Besar 01007 Kangar, Perlis, Malaysia
eekamal@eng.usm.my, rozmie.razif.othman@gmail.com, younismi@gmail.com, mhmz11_eee078@student.usm.my

Abstract. This paper discusses the practical adoption of t-way strategies (also termed interaction testing) for interaction testing. Unlike earlier work, this paper also highlights and unifies the different possible use of t-way strategies including uniform interaction, variable strength interaction, and input-output based relations. In order to help engineers make informed decision on the different use of t-way strategies, this paper discusses the main issues and shortcomings to be considered as well as demonstrates some practical results with a-step-by-step example. In doing so, this paper also analyzes the related works highlighting the current state-of-the-arts and capabilities of some of the existing t-way strategy implementations.

Keywords: software testing, interaction testing, t-way strategies, multi-way testing, combinatorial testing.

1 Introduction

The demand for multi-functional software has grown drastically over the years. To cater this demand, software engineers are forced to develop complex software with increasing number of input parameters. As a result, more and more dependencies between input parameters are to be expected, opening more possibilities of faults due to interactions. Although traditional static and dynamic testing strategies (e.g. boundary value analysis, cause and effect analysis and equivalent partitioning) are useful in fault detection and prevention [1], however they are not designed to detect faults due to interaction. As a result, many researchers nowadays are focusing on sampling strategy that based on interaction testing (termed t-way testing) [2].

As far as t-way testing is concerned, 3 types of interaction can be associated with interaction testing (i.e. uniform strength interaction, variable strength interaction and input-output based relations). Although one interaction type has advantages over the others in certain cases, however, no single interaction type can claim to be the ultimate solution for all interaction testing problems. Motivated by this challenge, this paper unifies and highlights the different possible use of t-way strategies. In order to help engineers make informed decision on the different use of t-way strategies, this paper also discusses the main issues and shortcomings to be considered as well as demonstrates some practical results with a-step-by-step example. In doing so, this paper also analyzes the related works highlighting the current state-of-the-arts and capabilities of some of the existing t-way strategy implementations.

For the purpose of presentation, the rest of this paper is organized as follows. Section 2 discusses fundamental of t-way strategies. Section 3 demonstrates the running example. Section 4 highlights our observations and issues. Section 5 analyses the related works. In Section 6, we present the digest of our analysis. Finally, section 7 summarizes our conclusion.

2 Fundamental of T-Way Strategies

Mathematically, t-way strategies can be abstracted to a covering array. Throughout this paper, the symbols p, v, and t are used to refer to number of parameters (or factor), values (or levels) and interaction strength for the covering array respectively. Referring to Table 1, the parameters are A, B, C, and D whilst the values are (a1, a2, b1, b2, c1, c2).

Earlier works suggested three definitions for describing the covering array. The first definition is based on whether or not the numbers of values for each parameter are equal. If the number of values is equal (i.e. uniformly distributed), then the test suite is called Coverage Array (CA). Now, if the number of values in non-uniform, then the test suite is called Mixed Coverage Array (MCA) [3, 4]. Finally, Variable Strength Covering array (VCA) refers to case when a smaller subset of covering arrays (i.e. CA or MCA) constitutes a larger covering array.

Although useful, the aforementioned definitions do not cater for the fact that there could be consideration of input and output (IO) based relations in order to construct CA, MCA, and VCA. As will be seen later, building from VCA notation for covering array, we have introduced a workable notation for IO based relations.

Normally, the CA takes parameters of N, t, p, and v respectively (i.e. CA(N,t,p,v)). For example, CA (9, 2, 4, 3) represents a test suite consisting of 9x4 arrays (i.e. the rows represent the size of test cases (N), and the column represents the parameter (p)). Here, the test suite also covers 2-way interaction for a system with 4 3 valued parameter.

Alternatively, MCA takes parameters of N, t, and Configuration (C) (i.e. MCA (N,t, C)). In this case, N and t carries the same meaning as in CA. Here, C captures the parameters and values of each configuration in the following format: $v_1^{\ p1}\ v_2^{\ p2}$, ….. $v_n^{\ pn}$ indicating that there are p1 parameters with v1 values, p2 parameters with v2 values, and so on. For example, MCA (1265, 4, $10^2 4^1 3^2 2^7$) indicates the test size of 1265 which covers 4-way interaction. Here, the configuration takes 12 parameters: 2 10 valued parameter, 1 4 valued parameter, 2 3 valued parameter and 7 2 valued parameter. Such notation can also be applicable to CA (e.g. CA (9,2,4,3) can be rewritten as CA (9,2,3^4)).

In the case of VCA, the parameter consists of N, t, C, and Set (S) (i.e. VCA (N,t,C,S)). Similar to MCA, N,t, and C carry the same meaning. Set S consists of a multi-set of disjoint covering array with strength larger t. For example, VCA (12, 2, $3^2 2^2$, {CA (3,$3^2 2^2$)}) indicates the test size of 12 for pairwise interaction (with 2 3 valued parameter and 2 2 valued parameter) and 3-way interaction (with 1 3 valued parameter and 2 2 valued parameter). As a special case of VCA, we can also consider cumulative combination of interaction. Using the same example, we can have VCA (14, {CA (2, $3^2 2^2$)}, {CA (3, $3^2 2^2$)}). Here, we have the test size of 14 for both pairwise and 3-way interaction (with with 2 3 valued parameter and 2 2 valued parameter).

In order to expand the scope of covering arrays for IO based relations, there is a need for a more compact notation. Here, building from CA, MCA, and VCA notation, we can express IO base relations (IOR) as IOR (N, C, R). Here, N and C take the same meaning given earlier whilst R represents a multi set of parameter relationship definition contributing towards the outputs. For example, for a 4 parameters system with 2 values and each parameter will be assigned a number 0, 1, 2 and 3 respectively. Assume two input-output relationships involve in the outputs (i.e. the first and the last parameter for the first output and the second and third parameter for the second output). Here, the relationship is written as R = {{0, 3}, {1, 2}}. Assuming the test size is 12, the complete notation for can be expressed as IOR (12, 4^2, {{0,3}, {1,2}}).

3 Running Example

In order to aid the discussion, consider the following software system example in Fig. 1.

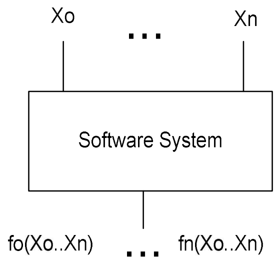


Fig. 1. Model of a Typical Software System

Assume that the input set X = {x_0.... x_n} significantly affects the output, noted as fo (x_0.... x_n) to fn (x_0.... x_n). If X is known to take a set of data values: D(x_0), D(x_1)... D(x_n), then the system must be tested against the set of all possible combinations of D. Here, the result is an ordered n-tuples {d0,d1...dn} where each d_i is an element of D(x_i). The size of the test suite would be the product size of all D(x):

$$T_{suite} = \{ D(x_0) \times D(x_1) \times \ldots D(X_n)\} \quad (1)$$

Obviously, the test suite T_{suite} can grow exponentially with the increase size of data element in the set D(x_0), D(x_1)... D(x_n). As far as the actual test data of T_{suite} is concerned, one can consider the interaction between all n variables x0, x21, x2...xn, termed, *exhaustive test*. Optionally, one can also consider the interaction of any t-way interactions of variables. Here, the value of t can take the minimum of 2 and the maximum of n-1. As a running example, let us assume that the starting test case for X, termed *base test case*, has been identified in Table 1. Here, symbolic values (e.g. a1, a2, b1, b2, c1, c2) are used in place of real data values to facilitate discussion.

Here, at full strength of interaction (i.e. t=4), we can get all exhaustive combination. In this case, the exhaustive combinations would be $2^4 = 16$.

As highlighted earlier, considering all exhaustive interaction is infeasible for large number of parameters and values. The next sub-sections demonstrate the fact that by adopting the t-way strategies (i.e. relaxing the interaction strength), the test data for testing can be systematically reduced. In this case, a step-by-step example will be demonstrated to illustrate the possible use of t-way strategies including uniform interaction, cumulative interaction, variable strength interaction, and input output relation based interaction.

Table 1. Base Data Values

	Input Variables			
	A	**B**	**C**	**D**
Base Values	a1	b1	c1	d1
	a2	b2	c2	d2

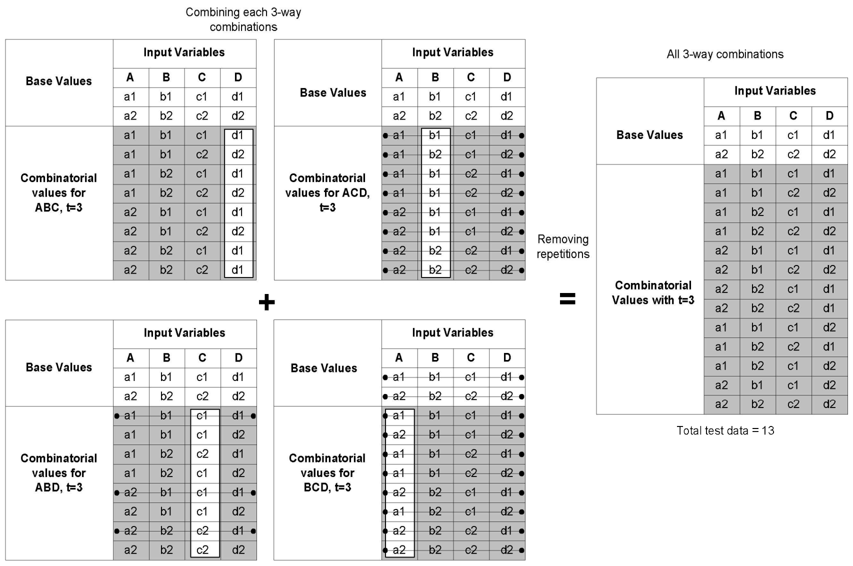
Table 2. Exhaustive Combination

	Input Variables			
	A	**B**	**C**	**D**
Base Values	a1	b1	c1	d1
	a2	b2	c2	d2
All Combinatorial Values	a1	b1	c1	d1
	a1	b1	c1	d2
	a1	b1	c2	d1
	a1	b1	c2	d2
	a1	b2	c1	d1
	a1	b2	c1	d2
	a1	b2	c2	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b1	c1	d2
	a2	b1	c2	d1
	a2	b1	c2	d2
	a2	b2	c1	d1
	a2	b2	c1	d2
	a2	b2	c2	d1
	a2	b2	c2	d2

3.1 Uniform Strength T-Way Interaction

Here, it is assumed that the interaction of variable is uniform throughout. Revisiting Table 1, and considering t=3, Fig. 2 highlights how the reduction is achieved. Firstly, the interaction is broken down between parameters ABC, ABD, ACD, and BCD.

Here, when parameters ABC are considered, the values for parameter D are don't cares (i.e. any random valid values for parameter D suffice). Similarly, when parameters ABD are considered, values for parameter C are don't cares. When parameters ACD are considered, values for parameter B are don't care. Finally, when parameters BCD are considered, values for parameter A are don't cares. Combining these results, we note that there are some repetitions of values between some entries for ABC, ABD, ACD and BCD. If these repetition is removed, we can get all the combinations at t=3.


Combining each 3-way combinations
+
=
Removing repetitions
All 3-way combinations
Total test data = 13

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for ABC, t=3	a1	b1	c1	d1
	a1	b1	c2	d2
	a1	b2	c1	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b1	c2	d2
	a2	b2	c1	d1
	a2	b2	c2	d1

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for ACD, t=3	● ~~a1~~	~~b1~~	~~c1~~	~~d1~~ ●
	● ~~a1~~	~~b2~~	~~c1~~	~~d2~~ ●
	● ~~a1~~	~~b1~~	~~c2~~	~~d1~~ ●
	● ~~a1~~	~~b1~~	~~c2~~	~~d2~~ ●
	● ~~a2~~	~~b1~~	~~c1~~	~~d1~~ ●
	● ~~a2~~	~~b1~~	~~c1~~	~~d2~~ ●
	● ~~a2~~	~~b2~~	~~c2~~	~~d1~~ ●
	● ~~a2~~	~~b2~~	~~c2~~	~~d2~~ ●

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for ABD, t=3	● ~~a1~~	~~b1~~	~~c1~~	~~d1~~ ●
	a1	b1	c1	d2
	a1	b2	c2	d1
	a1	b2	c1	d2
	● ~~a2~~	~~b1~~	~~c1~~	~~d1~~ ●
	a2	b1	c1	d2
	● ~~a2~~	~~b2~~	~~c2~~	~~d1~~ ●
	a2	b2	c2	d2

Base Values	Input Variables			
	A	B	C	D
	● ~~a1~~	~~b1~~	~~c1~~	~~d1~~ ●
	● ~~a2~~	~~b2~~	~~c2~~	~~d2~~ ●
Combinatorial values for BCD, t=3	● ~~a1~~	~~b1~~	~~c1~~	~~d1~~ ●
	● ~~a2~~	~~b1~~	~~c1~~	~~d2~~ ●
	● ~~a1~~	~~b1~~	~~c2~~	~~d1~~ ●
	● ~~a1~~	~~b1~~	~~c2~~	~~d2~~ ●
	● ~~a2~~	~~b2~~	~~c1~~	~~d1~~ ●
	● ~~a1~~	~~b2~~	~~c1~~	~~d2~~ ●
	● ~~a2~~	~~b2~~	~~c2~~	~~d1~~ ●
	● ~~a2~~	~~b2~~	~~c2~~	~~d2~~ ●

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial Values with t=3	a1	b1	c1	d1
	a1	b1	c2	d2
	a1	b2	c1	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b1	c2	d2
	a2	b2	c1	d1
	a2	b2	c2	d1
	a1	b1	c1	d2
	a1	b2	c2	d1
	a1	b2	c1	d2
	a2	b1	c1	d2
	a2	b2	c2	d2

Fig. 2. Uniform t-way Interaction Results (t=3), CA $(13,3,2^4)$

Here, we note that the test suite has been reduced from 16 (for exhaustive combination) to 13 (for t=3), a saving of 18.75 percent. Using the notation discussed earlier, we can write this test suite as $T_{suite} = CA\ (13,3,2^4)$.

3.2 Variable Strength T-Way Interaction

In many real applications, interaction may not be uniform for all parameters. Here, a particular subset of variables can have a higher interaction dependency than other variables (indicating failures due to the interaction of that subset may have more significant impact to the overall system). For example, consider a subset of components that control a safety-critical hardware interface. We want to use stronger coverage in that area (i.e. t=3). However, the rest of our components may be sufficiently tested with t=2. In this case, we can assign variable coverage strength to each subset of components as well as to the whole system.

To illustrate variable strength t-way interaction, we adopt the same example as Table 1. Now, we assume that all interaction is uniform at t=2 for all parameters (i.e. based on our result in Fig. 3). Then, we consider t=3, only for parameters B,C,D. Combining both interactions yield result shown in Fig. 3. Here, the test suite has been reduced from 16 (for exhaustive case) to 13, a saving of 18.75 percent. Using the notation describe earlier, we can write this reduction as T $_{suite}$ = VCA (13,2,2^4, {CA(3,2^3)}).

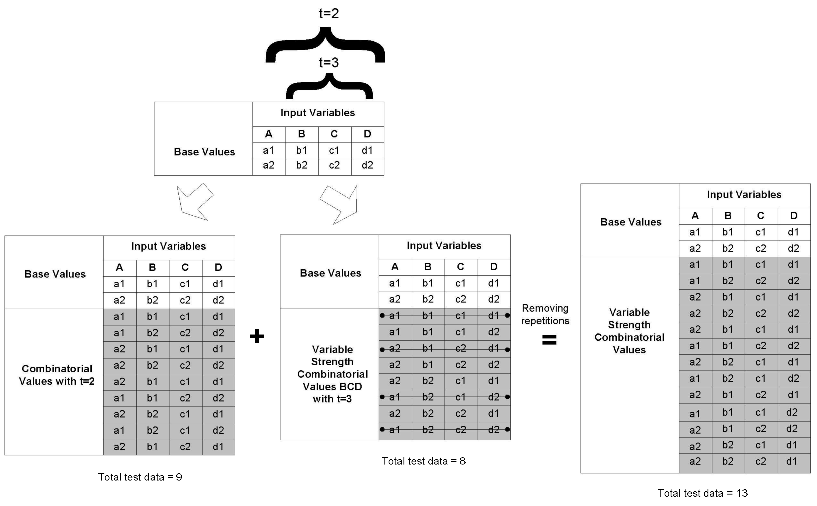

t=2
t=3
Input Variables
Base Values
Combinatorial Values with t=2
Variable Strength Combinatorial Values BCD with t=3
Removing repetitions
Variable Strength Combinatorial Values
Total test data = 9
Total test data = 8
Total test data = 13

Fig. 3. Variable Strength Interaction, VCA (13,2,2^4, {CA(3,2^3)})

As a special case for VCA, we can also consider cumulative strength, t=3 and t=2. Revisiting Table 1, we can derive the test suite for t=2 using the same technique as t=3 (see Fig. 4).

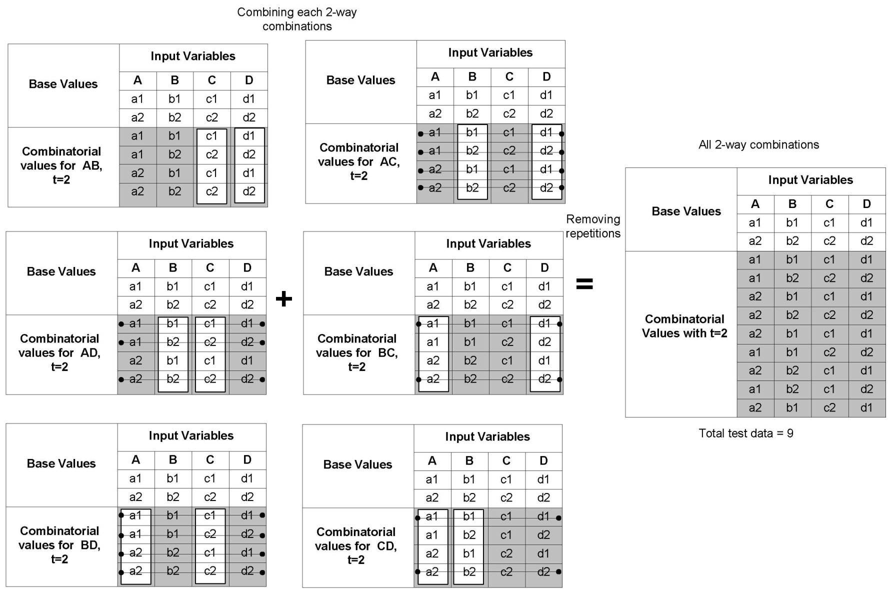
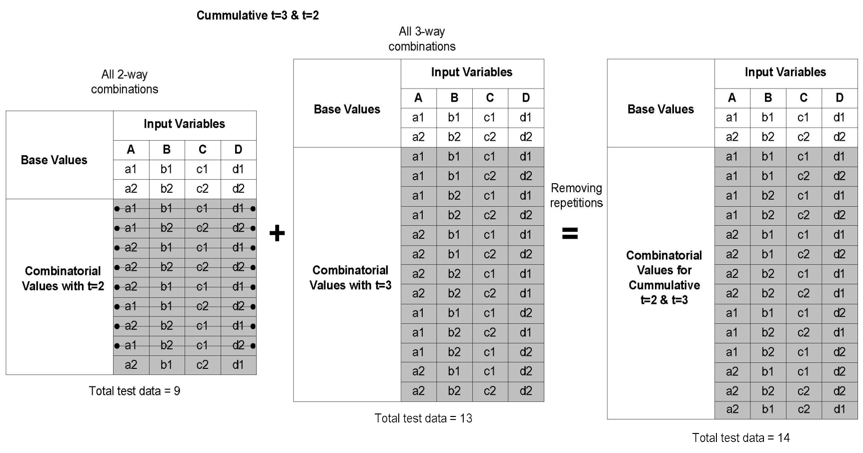

Combining each 2-way combinations
Input Variables
Base Values
Combinatorial values for AB, t=2
Combinatorial values for AC, t=2
Combinatorial values for AD, t=2
Combinatorial values for BC, t=2
Combinatorial values for BD, t=2
Combinatorial values for CD, t=2
Removing repetitions
All 2-way combinations
Combinatorial Values with t=2
Total test data = 9

Fig. 4. Uniform t-way Interaction Results (t=2), CA (9,2,2^4)



Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial Values with t=2	● ~~a1~~	~~b1~~	~~c1~~	~~d1~~ ●
	● ~~a1~~	~~b2~~	~~c2~~	~~d2~~ ●
	● ~~a2~~	~~b1~~	~~c1~~	~~d1~~ ●
	● ~~a2~~	~~b2~~	~~c2~~	~~d2~~ ●
	● ~~a2~~	~~b1~~	~~c1~~	~~d1~~ ●
	● ~~a1~~	~~b1~~	~~c2~~	~~d2~~ ●
	● ~~a2~~	~~b2~~	~~c1~~	~~d1~~ ●
	● ~~a1~~	~~b2~~	~~c1~~	~~d2~~ ●
	a2	b1	c2	d1

Total test data = 9

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial Values with t=3	a1	b1	c1	d1
	a1	b1	c2	d2
	a1	b2	c1	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b1	c2	d2
	a2	b2	c1	d1
	a2	b2	c2	d1
	a1	b1	c1	d2
	a1	b2	c2	d1
	a1	b2	c1	d2
	a2	b1	c1	d2
	a2	b2	c2	d2

Total test data = 13

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial Values for Cummulative t=2 & t=3	a1	b1	c1	d1
	a1	b1	c2	d2
	a1	b2	c1	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b1	c2	d2
	a2	b2	c1	d1
	a2	b2	c2	d1
	a1	b1	c1	d2
	a1	b2	c2	d1
	a1	b2	c1	d2
	a2	b1	c1	d2
	a2	b2	c2	d2
	a2	b1	c2	d1

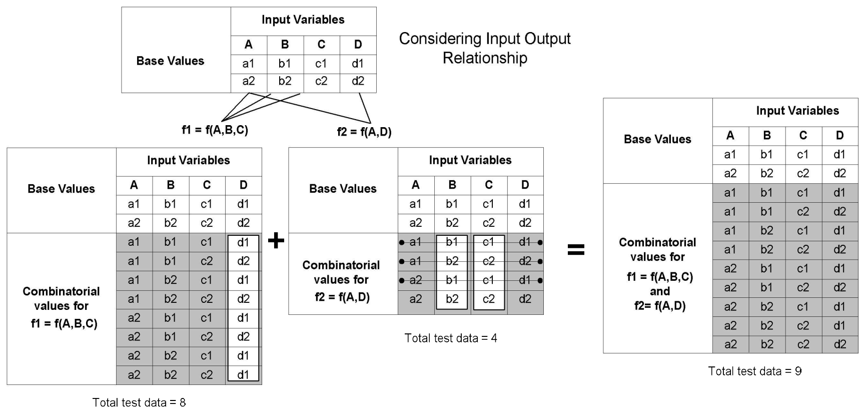
Total test data = 14

Fig. 5. Cummulative t=2 & t=3 Results, CA (14, {CA (9,2, 2^4)}, {CA (13, 3, 2^4)})

Combining the test suite with t=3, yields the following result (see Fig. 5). Here, we note that T_{suite} for t=2 is not necessarily a subset of T_{suite} for t=3. In this case, the test suite has been reduced from 16 (for exhaustive case) to 14, a saving of 12.5 percent. Using the notation described earlier, we can write this reduction as T_{suite} = CA(9,2,2^4) + CA (13,3,2^4) or simply T_{suite} = CA (14, {CA (9,2, 2^4)}, {CA (13, 3, 2^4)}).

3.3 Input Output Relation Based Interaction

Similar to variable strength t-way interaction, input output relation based interaction does not deal with uniform interaction. Also, unlike other interaction possibilities discussed earlier, the reduction is performed by considering the knowledge on the input and output relationship amongst the parameter values involved. Normally, this relationship can be derived based on some statistical analysis such as Design of Experiments (DOE).



Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for f1 = f(A,B,C)	a1	b1	c1	d1
	a1	b1	c2	d2
	a1	b2	c1	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b1	c2	d2
	a2	b2	c1	d1
	a2	b2	c2	d1

Total test data = 8

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for f2 = f(A,D)	● ~~a1~~	~~b1~~	~~c1~~	~~d1~~ ●
	● ~~a1~~	~~b2~~	~~c2~~	~~d2~~ ●
	● ~~a2~~	~~b1~~	~~c1~~	~~d1~~ ●
	a2	b2	c2	d2

Total test data = 4

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for f1 = f(A,B,C) and f2= f(A,D)	a1	b1	c1	d1
	a1	b1	c2	d2
	a1	b2	c1	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b1	c2	d2
	a2	b2	c1	d1
	a2	b2	c2	d1
	a2	b2	c2	d2

Total test data = 9

Fig. 6. Input Output Based Interaction, T_{suite} = IOR (9, 2^4, {{0,1,2},{0,3}})

To illustrate the input output based interaction, we revisit Table 1 with the following input output relationship.

i. Only two outputs are considered, f1 an f2.
ii. f1 is a function of A,B,C, that is, f1=f(A,B,C).
iii. f2 is a function of A,D, that is, f2=f(A,D).

Ideally, these input output relationship are not to be assumed as they come from experimental results. Upon establishing these assumptions, we can derive the test suite accordingly. Fig. 6 illustrates the complete results. Here, the test suite has been reduced from 16 (for exhaustive case) to 9, a saving of 43.75 percent. Using the notation described earlier, we can write R ={{0,1,2}, {0,3}} or the overall reduction as T_{suite} = IOR (9, 2^4, {{0,1,2}, {0,3}}).

4 Observation and Issues

The main observation here is the fact that by relaxing the interaction, we can systematically reduce the test data for consideration significantly. Other subtle observations can also be discussed further here.

- The final results for all cases discussed earlier (i.e. for uniform interaction, cumulative interaction, variable strength interaction, and input output relation based interaction) are not the most optimum, that is, all the interaction elements appear more than once. In order be optimum, more efficient algorithms are required (see section 5 on related work).
- Uniform strength strategies are useful when there is little knowledge on the system under test. Thus, the interaction is assumed to be uniform throughout through judicious (e.g. based on experience on similar system) selection of interaction strength (t).
- Variable strength interaction strategies are applicable when a particular subset of input parameters has a higher interaction dependency than other parameters (indicating failures due to the interaction of that subset may have more significant impact to the overall system). As such, at least two interaction strengths can be assigned accordingly with one being stronger than the other.
- Input output relation based strategies are useful when the relationship amongst inputs and outputs are known (e.g. often through statistical method such as Design of Experiments) in advanced. Thus, the interaction can be properly established.

5 Related Works

The main aim of any t-way strategies is to ensure that the generated test suite covers the interaction tuples of interest for a certain type of interaction at least one whilst reducing the test data into manageable ones. However, there is no unique solution to this problem (i.e. NP-hard problem [5, 6]). In fact, it is unlikely that a unique strategy exists that can always generate the most optimum number of test case in every configuration.

A number of useful strategies have been developed from the last decade. A significant number of work have focused on pairwise (t=2) strategies (e.g. OATS (Orthogonal Array Test System) [7], IRPS [8], AllPairs [9], IPO [10], TCG (Test Case Generator) [11], OATSGen [12], ReduceArray2 [13], DDA (Deterministic Density Algorithm) [14], CTE-XL [15], rdExpert [16],and SmartTest [17]). As interaction is limited to t=2, pairwise strategies often yield the most minimum test set as compared other interaction. Although useful in some class of system, pairwise testing is known be ineffective for system with highly interacting variables [18-20]. For this reason, rather than dwelling on pairwise strategies, we are more interested on a general strategy for t-way test generation including that of variable strength, and input output based relations. The survey of each of these strategies is discussed next.

Klaib and Zamli developed a deterministic t-way strategy called GTWay [21, 22]. The main feature of GTWay is that it supports both test data generation and automated execution. This strategy heavily depends on its pair generation and backtracking algorithm. Once all pairs are generated, the backtracking algorithm will iteratively traverses all pairs in order to combine pairs with common parameter values in order to complete a test suite. To ensure the optimality of test data generated, combination of pairs can only be accepted if its cover the most uncovered pairs. In case of pairs that cannot be combined, the algorithm falls back to the first defined value.

Hartman et al. developed a t-way strategy, called IBM's Intelligent Test Case Handler (WHITCH), as Eclipse Java plug-in tool [23]. WHITCH uses the sophisticated combinatorial algorithms based on exhaustive search to construct test suites for t-way testing. Although useful as part of IBM's automated test plan generation, WHITCH results appear to be not optimized as far as the number of generated test cases is concerned. Furthermore, due to its exhaustive search algorithm, WHITCH execution times typically take a long time.

Jenkins developed a deterministic t-way generation strategy, called Jenny [24]. Jenny adopts a greedy algorithm to produce a test suite in one-test-at-a time fashion. In Jenny, each feature has its own list of t-way interaction. It starts out with 1-way interaction (just the feature itself). When there are no further 1-way interaction left to cover, Jenny goes to 2-way interactions (this feature with one other feature) and so on. Hence, during generation instance, there could have one feature still covering 2-way interaction while another feature is already working on 3-way interactions. This process goes on until all interactions are covered.

Cohen et al developed the first commercialized t-way strategy, called AETG [25].. AETG starts with the generation of all possible parameter interactions. Based all the possible parameter interactions, AETG then decides the combination of values to maximize the interaction coverage so that it can build an efficient test set. This selection process is performed "one-test-at-a-time" until all the parameter interactions are covered. To enhance its capability (e.g. for better test size), a number of variant AETG implementations have been implemented such as that of mAETG [3], TCG [11] and mTCG [3].

Lei et al developed IPOG [26] based a novel "one-test-at-a-time" approach. In IPOG, the interaction parameters will be generated first as the partial test suite based on the number of parameters and interaction value. The test suite is then extended with the values of the next parameters by using horizontal and vertical extension mechanism. Here, horizontal extension extends the partial test suite with values of the next parameter to cover the most interaction. Upon completion of horizontal extension, vertical extension may be summoned to generate additional test cases that cover all uncovered interactions. More recently, a number of variants have been developed to improve the IPOG's performance (i.e. IPOG-D [27], IPOF and IPOF2 [28]).

Younis and Zamli proposed another variant for IPOG named MIPOG [29, 30]. Addressing the dependency issue arising in IPOG strategy (i.e. generation of a test data can be unstable in IPOG due to the possibility of changing values during the vertical extension especially for test cases that include "don't care" value), MIPOG introduces two new algorithms for both horizontal and vertical extension. Here, the both algorithms remove the inherent dependency between subsequently generated test data (as occurred in IPOG family). Furthermore, MIPOG strategy also further optimizes the don't care value during vertical extension making this strategy in most cases outperform IPOG in term of test size. As the data dependency issue is removed, Younis and Zamli implemented the parallel MIPOG strategy in the Multi-core platform (called MC-MIPOG (MultiCore MIPOG)) [31] as well as in the Grid platform (called G-MIPOG (Grid MIPOG)) [32]. By implementing the strategy in multi-core and grid environment, the time taken to produce the final test suite for MIPOG strategy is reduced.

Arshem developed a freeware Java based t-way testing tool called Test Vector Generator (TVG) [33] based on extension of AETG strategy to support t-way testing . Similar efforts are also undertaken by Bryce and Colbourn [34, 35] to enhance AETG for t-way testing. Nie et al. [36] proposed a generalization for IPO with genetic algorithm (GA), called IPO_N, and GA_N respectively for t=3. Here, IPO_N performed better than GA_N in terms of test size as well as execution time [36].

As far as variable strength t-way strategies are concerned, Cohen et al. implemented the first model t-way strategy with variable strength based capability based on simulated annealing (SA) [37]. Although generating optimal test suites, this approach is very time consuming because all interaction elements needs to be analyzed exhaustively using binary search strategy.

Wang et al. extended the model proposed by Cohen et al [37] and proposed a more general strategy relying on two greedy algorithms. The first algorithm is based on one-test-at-a-time strategy while the other algorithm is based on in-parameter-order strategy [38]. Although useful as far as addressing the limitation of the Cohen's model in terms of the need for the interaction strength (t) involved to be disjoint, Wang et al. approach appears to produce non-optimized set for mixed parameter values.

Chen et. al. proposed a variant algorithm based on ant colony approach (named Ant Colony Strategy (ACS)) in order to support variable strength capability [39]. Similar to Cohen et al [37], this approach is also time consuming and supports low interaction strength $2 \leq t \leq 3$. Apart from these approaches, new version of TVG [33] also addresses the variable strength capabilities but with non-optimized set.

Concerning the strategies that address the support for input output based relations much work has started to appear. The input output based strategies can be considered as the general case for t-way strategies as they can be customized to behave as such to support all interaction possibilities (i.e. uniform strength and variable strength interactions). However, if the parameters are large, setting up for uniform and variable strength interaction can be cumbersome as there is a need to define all the relations for each interaction.

Schroeder and Korel developed an input output relations based strategy called Union [40, 41]. In the case of Union, the strategy generates the test suite for each output variable that cover all associated input interaction and then assign random value for all the 'don't care'. Then, the strategy finds the union of all test suites in order to reduce the number of generated test data.

Building from the Union Strategy, Schroeder et al developed the Greedy strategy [41, 42]. Similar to Union, the Greedy strategy also generates the initial test suite that covered all associated input interaction by randomly selecting values for all don't care parameters. Nonetheless, unlike the Union strategy, the Greedy strategy picks only the unselected test case from the initial test suite which covers the most uncovered interactions as the final test suite. In this manner, the Greedy strategy often generates a more optimal test size than that of the Union Strategy.

Wang et al developed two strategies to support input based relations called ParaOrder and ReqOrder [43]. ParaOrder strategy implements horizontal and vertical extension for generating the final test cases, much like the uniform strength IPOG implementation [26]. The main difference between ParaOrder with IPOG is the fact that the initial test case for the former is generated based on the first defined input output relationships while the initial test case for the latter is generated in-defined-order-of-parameter found. In the case of ReqOrder, the selection of initial test case does not necessarily follow the first defined input output relationships rather the selection is done based on the highest input output relationship coverage.

6 Analysis of Related Works

In order to help test engineers make inform decision on the adoption of a particular t-way strategy implementation, Table 3 provides the digest information regarding the supported interactions by all strategies discussed earlier. It should be noted that an "√*" indicates that the strategy implementation of interest partially supports the said interaction (i.e. only supports pairwise interaction) whereas an "√" indicates that the strategy implementation of interest provides full support. An "X" indicates the missing support.

Table 3. Analysis of Related Works

Strategy	Uniform Strength	Variable Strength	I/O Relations	Strategy	Uniform Strength	Variable Strength	I/O Relations
OATS	√*	X	X	IPOG	√	X	X
IRPS	√*	X	X	IPOG-D	√	X	X
AllPairs	√*	X	X	IPOF	√	X	X
IPO	√*	X	X	IPOF2	√	X	X
TCG	√*	X	X	MIPOG	√	X	X
OATSGen	√*	X	X	MC-MIPOG	√	X	X
ReduceArray2	√*	X	X	G-MIPOG	√	X	X
DDA	√*	X	X	TVG	√	√	√
CTE-XL	√*	X	X	IPO_N	√	X	X
rdExpert	√*	X	X	GA_N	√	X	X
SmartTest	√*	X	X	SA	√	√	X
GTWay	√	X	X	ACS	√	√	X
WHITCH	√	X	X	Union	√	√	√
Jenny	√	X	X	Greedy	√	√	√
AETG	√	X	X	ParaOrder	√	√	√
mAETG	√	X	X	ReqOrder	√	√	√
mTCG	√	X	X				

Referring to Table 3, most strategy implementations merely support uniform strength interactions. Here, only Union, TVG,ParaOrder, and ReqOrder to support all possible type of interactions. With such knowledge, test engineers can adopt the tool implementation accordingly (i.e. based on the interaction requirements).

7 Conclusion

Summing up, this paper has presented the different possible use of t-way strategies including uniform interaction, variable strength interaction, and input-output based relations. Additionally, this paper has also analyzed the related works by highlighting the capabilities of some of the existing t-way strategy implementations. It hoped that such an analysis can help test engineers choose the interaction type of interest as well as the tool support required.

Finally, while much useful research work has been done in the last decade (i.e. as evident by the large number of developed strategy implementations), the adoption of interaction testing for studying and testing real life systems has not been widespread [44]. In order to address this issue, more research into the algorithms and techniques are required to facilitate its adoption in the main stream of software engineering.

Acknowledgement

This research is partially funded by the generous MOHE fundamental grants – "Investigating T-Way Test Data Reduction Strategy Using Particle Swarm Optimization Technique" and USM RU grants – "Development of Variable-Strength Interaction Testing Strategy for T-Way Test Data Generation".

References

[1] Zamli, K.Z., Younis, M.I., Abdullah, S.A.C., Soh, Z.H.C.: Software Testing, 1st edn. Open University, Malaysia KL (2008)

[2] Kuhn, D.R., Lei, Y., Kacker, R.: Practical Combinatorial Testing: Beyond Pairwise. IEEE IT Professional 10(3), 19–23 (2008)

[3] Cohen, M. B.: Designing Test Suites For Software Interaction Testing. PhD Thesis, School of Computer Science, University of Auckland (2004)

[4] Zekaoui, L.: Mixed Covering Arrays On Graphs And Tabu Search Algorithms. Msc Thesis, Ottawa-Carleton Institute for Computer Science, University of Ottawa, Ottawa, Canada (2006)

[5] Shiba, T., Tsuchiya, T., Kikuno, T.: Using Artificial Life Techniques To Generate Test Cases For Combinatorial Testing. In: Proceedings of the 28th Annual Intl. Computer Software and Applications Conf. (COMPSAC 2004), Hong Kong, pp. 72-77 (2004)

[6] Younis, M.I., Zamli, K.Z., Klaib, M.F.J., Soh, Z.H.C., Abdullah, S.A.C., Isa, N.A.M.: Assessing IRPS As An Efficient Pairwise Test Data Generation Strategy. International Journal of Advanced Intelligence Paradigms 2(3), 90–104 (2010)

[7] Krishnan, R., Krishna, S.M., Nandhan, P.S.: Combinatorial Testing: Learnings From Our Experience. ACM SIGSOFT Software Engineering Notes 32(3), 1–8 (2007)

[8] Younis, M.I., Zamli, K.Z., Isa, N.A.M.: IRPS: An Efficient Test Data Generation Strategy For Pairwise Testing. In: Proceedings of the 12th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, Part I, pp. 493–500. Springer, Zagreb, Croatia (2008)

[9] Allpairs Test Case Generation Tool, `http://www.satisfice.com/tools.shtml`

[10] Lei, Y., Tai, K.C.: In-Parameter-Order: A Test Generation Strategy For Pairwise Testing. In: Proceedings of 3rd IEEE International Conference on High Assurance Systems Engineering Symposium, Washington DC, USA, pp.254–261 (1998)

[11] Tung, Y.W., Aldiwan, W.S.: Automatic Test Case Generation For The New Generation Mission Software System. In: Proceedings of IEEE Aerospace Conference, pp. 431–437. Big Sky, MT, USA (March 2000)

[12] Harrell, J.M.: Orthogonal Array Testing Strategy (OATS) Technique: Seilevel, Inc. (2001)

[13] Daich, G.T.: Testing Combinations Of Parameters Made Easy (Software Testing). In: Proceedings of IEEE Systems Readiness Technology Conference (AUTOTESTCON 2003), pp. 379–384 (2003)

[14] Colbourn, C.J., Cohen, M.B., Turban, R.C.: A Deterministic Density Algorithm For Pairwise Interaction Coverage. In: Proceedings. of the Intl. Conference on Software Engineering (IASTED 2004), pp. 345–352 (2004)

[15] Lehmann, E., Wegener, J.: Test Case Design By Means Of The CTE-XL. In: Proceedings of the 8th European International Conference on Software Testing, Analysis & Review (EuroSTAR 2000), Copenhagen, Denmark (2000)

[16] Copeland, L.: A Practitioner's Guide To Software Test Design. Massachusetts, STQE Publishing, USA (2004)

[17] SmartTest - Pairwise Testing, `http://www.smartwaretechnologies.com/smarttestprod.htm`

[18] Kuhn, D.R., Wallace, D.R., Gallo, A.M.: Software Fault Interaction And Implication For Software Testing. IEEE Transaction on Software Engineering. 30(6), 418–421 (2004)

[19] Younis, M.I., Zamli, K.Z.: Assessing Combinatorial Interaction Strategy For Reverse Engineering Of Combinational Circuits. In: Proceedings of the IEEE Symposium on Industrial Electronics and Applications (ISIEA 2009), Kuala Lumpur, Malaysia (2009)

[20] Younis, M.I., Zamli, K.Z.: A Strategy for Automatic Quality Signing And Verification Processes For Hardware And Software Testing. Advances in Software Engineering 1–7 (2010)

[21] Zamli, K.Z., Klaib, M.F.J., Younis, M.I., Isa, N.A.M., Abdullah, R.: Design And Implementation Of A T-Way Test Data Generation Strategy With Automated Execution Tool Support. Information Sciences 181(9), 1741–1758 (2011)

[22] Klaib, M. F. J.: Development Of An Automated Test Data Generation And Execution Strategy Using Combinatorial Approach. PhD. Thesis, School of Electrical And Electronics, Universiti Sains Malaysia (2009)

[23] IBM Intelligent Test Case Handler, `http://www.alphaworks.ibm.com/tech/whitch`

[24] Jenny Test Tool, `http://www.burtleburtle.net/bob/math/jenny.html`

[25] Cohen, D.M., Dalal, S.R., Fredman, M.L., Patton, G.C.: The AETG System: An Approach To Testing Based On Combinatorial Design. IEEE Transactions on Software Engineering 23(7), 437–444 (1997)

[26] Lei, Y., Kacker, R., Kuhn, D.R., Okun, V., Lawrence, J.: IPOG: A General Strategy For T-Way Software Testing. In: Proceedings of the 14th Annual IEEE International Conference and Workshops on The Engineering of Computer-Based Systems, Tucson, AZ, pp. 549–556 (2007)

[27] Lei, Y., Kacker, R., Kuhn, R., Okun, V., Lawrence, J.: IPOG/IPOG-D: Efficient Test Generation For Multi-way Combinatorial Testing. Journal of Software Testing, Verification and Reliability 18(3), 125–148 (2008)

[28] Forbes, M., Lawrence, J., Lei, Y., Kacker, R., Kuhn, D.R.: Refining The In-Paramater-Order Strategy For Constructing Covering Arrays. Journal of Research of the National Institute of Standards and Technology. 113(5), 287–297 (2008)
[29] Younis, M.I., Zamli, K.Z., Isa, N.A.M.: MIPOG - Modification Of The IPOG Strategy For T-Way Software Testing. In: Proceeding of The Distributed Frameworks and Applications (DFmA), Penang, Malaysia (2008)
[30] Younis, M. I.: MIPOG: A Parallel T-Way Minimization Strategy For Combinatorial Testing. PhD. Thesis, School of Electrical And Electronics, Universiti Sains Malaysia (2010)
[31] Younis, M.I., Zamli, K.Z.: MC-MIPOG: A Parallel T-Way Test Generation Strategy For Multicore Systems. ETRI Journal 32(1), 73–83 (2010)
[32] Younis, M.I., Zamli, K.Z., Isa, N.A.M.: A Strategy For Grid Based T-Way Test Data Generation. In: Proceedings the 1st IEEE International Conference on Distributed Frameworks and Application (DFmA 2008), Penang, Malaysia, pp. 73–78 (2008)
[33] TVG, `http://sourceforge.net/projects/tvg`
[34] Bryce, R.C., Colbourn, C.J.: A Density-Based Greedy Algorithm For Higher Strength Covering Arrays. Software Testing, Verification and Reliability 19(1), 37–53 (2009)
[35] Bryce, R.C., Colbourn, C.J.: The Density Algorithm For Pairwise Interaction Testing. Software Testing, Verification and Reliability. 17(3), 159–182 (2007)
[36] Nie, C., Xu, B., Shi, L., Dong, G.: Automatic Test Generation For N-Way Combinatorial Testing. In: Reussner, R., Mayer, J., Stafford, J.A., Overhage, S., Becker, S., Schroeder, P.J. (eds.) QoSA 2005 and SOQUA 2005. LNCS, vol. 3712, pp. 203–211. Springer, Heidelberg (2005)
[37] Cohen, M.B., Gibbons, P.B., Mugridge, W.B., Colbourn, C.J., Collofello, J.S.: Variable Strength Interaction Testing Of Components. In: Proceedings of 27th Annual International Computer Software and Applications Conference, Dallas, USA pp. 413–418 (2003)
[38] Wang, Z., Xu, B., Nie, C.: Greedy Heuristic Algorithms To Generate Variable Strength Combinatorial Test Suite. In: Proceedings of the 8th International Conference on Quality Software, Oxford, UK, pp. 155–160 (2008)
[39] Chen, X., Gu, Q., Li, A., Chen, D.: Variable Strength Interaction Testing With An Ant Colony System Approach. In: Proceedings of 16th Asia-Pacific Software Engineering Conference, Penang, Malaysia, pp. 160–167 (2009)
[40] Schroeder, P.J., Korel, B.: Black-Box Test Reduction Using Input-Output Analysis. SIGSOFT Software Engineering Notes 25(5), 173–177 (2000)
[41] Schroeder, P. J.: Black-Box Test Reduction Using Input-Output Analysis. PhD Thesis, Department of Computer Science, Illinois Institute of Technology, Chicago, IL,USA (2001)
[42] Schroeder, P.J., Faherty, P., Korel, B.: Generating Expected Results For Automated Black-Box Testing. In: Proceedings of 17th IEEE International Conference on Automated Software Engineering (ASE 2002), Edinburgh, Scotland, UK, pp. 139–148 (2002)
[43] Wang, Z., Nie, C., Xu, B.: Generating Combinatorial Test Suite For Interaction Relationship. In: Proceedings of 4th International Workshop on Software Quality Assurance (SOQUA 2007), Dubrovnik, Croatia, pp. 55–61 (2007)
[44] Czerwonka, J.: Pairwise Testing In Real World. In: Proceedings of 24th Pacific Northwest Software Quality Conference, Portland, Oregon, USA, pp. 419–430 (2006)

On the Relationship between Proof Writing and Programming: Some Conclusions for Teaching Future Software Developers

Michael Hartwig

Multimedia University, Malaysia
`Michael.Jua.Hartwig@gmail.com`

Abstract. The analogy between proving theorems and writing computer programs has been debated for a long time. In a recent paper, Calude and others [5] argue that - albeit mentioned analogy seems to exist - the role of proof in mathematical modeling (and hence programming) is very small. Describing the act of proving and the act of writing a computational program with the help of the SECI model, a model used widely to describe knowledge creation processes, it can be argued that the thought processes needed for both activities complement each other. This fact can then be used to justify a sound and rigorous training in proof writing for the programmer and future software developer.

Keywords: proof writing, programming, software development, computer science education.

1 Motivation

The similarity (or difference) between the act of proving a theorem and the act of writing a computer program has caused many discussions in the computer science community [1,2]. Some authors like Friske [3] focussing on the immanent use of abstract objects in both activities find a strong similarity: "There is a close analogy between the thought processes used in computer programming and the thought processes for proof writing." Daly [4] described similar experiences in teaching concluding: "Constructing a mathematical proof is isomorphic with writing a computer program. Validating a proof is as difficult as validating a computer program."

Others like Calude [5] stressed the importance of the outcome of the thought processes involved. In their view, computer programs correspond to models and proofs correspond to algorithms. Henceforward, computer programs are subject to adequacy tests, while theorems and proofs are subject to a correctness test. This difference is often justified by the (de facto) non existence of correctness proofs in the programming world. But it also allows for a different view of programming where programming is seen as an art that is centred around human machine interaction. It could then be easily interpreted to minimize or relegate the role of mathematics and abstract thinking taught in computer science classes