

Finding all the faults

In the penultimate article of his six-part series on Requirements-based Testing That Finds All Faults, Professor Mike Holcombe states the generalized machine model theorem formally, and makes a claim about the power of the test set it generates

We have looked at how a generalized machine model can be captured from a set of requirements which provides a mechanism for the generation of functional test sets. We have looked at what might be done to design a system so that it is more easily tested, the design for test conditions. In this article, we will make a claim about the power of the test set so created.

Test set generation

Once we have created our stream X-Machine model, our specification, which satisfies the design for test conditions of controllability (being able to drive the system to trigger any function from any state) and observability (being able to detect which function has been triggered at any state) we have to consider a number of other issues.

The main issue revolves around the basic test process. We are trying to create a number of tests that will be applied to the implementation under test. The philosophy is the following, we regard the system under test as a stream X-Machine, we do not know whether it is the one we want - that is equivalent to our specification machine - this is what we want to test. Our tests are designed to establish whether the implementation behaves exactly as the specification machine does (figure 1).

We will use the the fundamental theorem of testing to justify the claims.

Recall that we constructed a test set in part 3 specifically given by

$$Z = \Phi^k W \cup \Phi^{k-1} W \cup \dots \cup W$$

and formed the set of input strings $t(\mathbf{TZ})$, the function t allows us to convert function sequences to input sequences with the same effect.

The value of k is chosen to represent the difference between the known state size of the specification and the (unknown) state size of the implementation. In practice this is not usually large; for especially sensitive applications one can make very pessimistic assumptions about k at the cost of a large test set.

We must make the following further assumptions:

- 1 The specification is a deterministic stream X-Machine;
- 2 The set of basic functions Φ is output-distinguishable and complete (design for test);
- 3 The associated automata are minimal;
- 4 The implementation is a deterministic stream X-Machine with the same set of basic functions F .

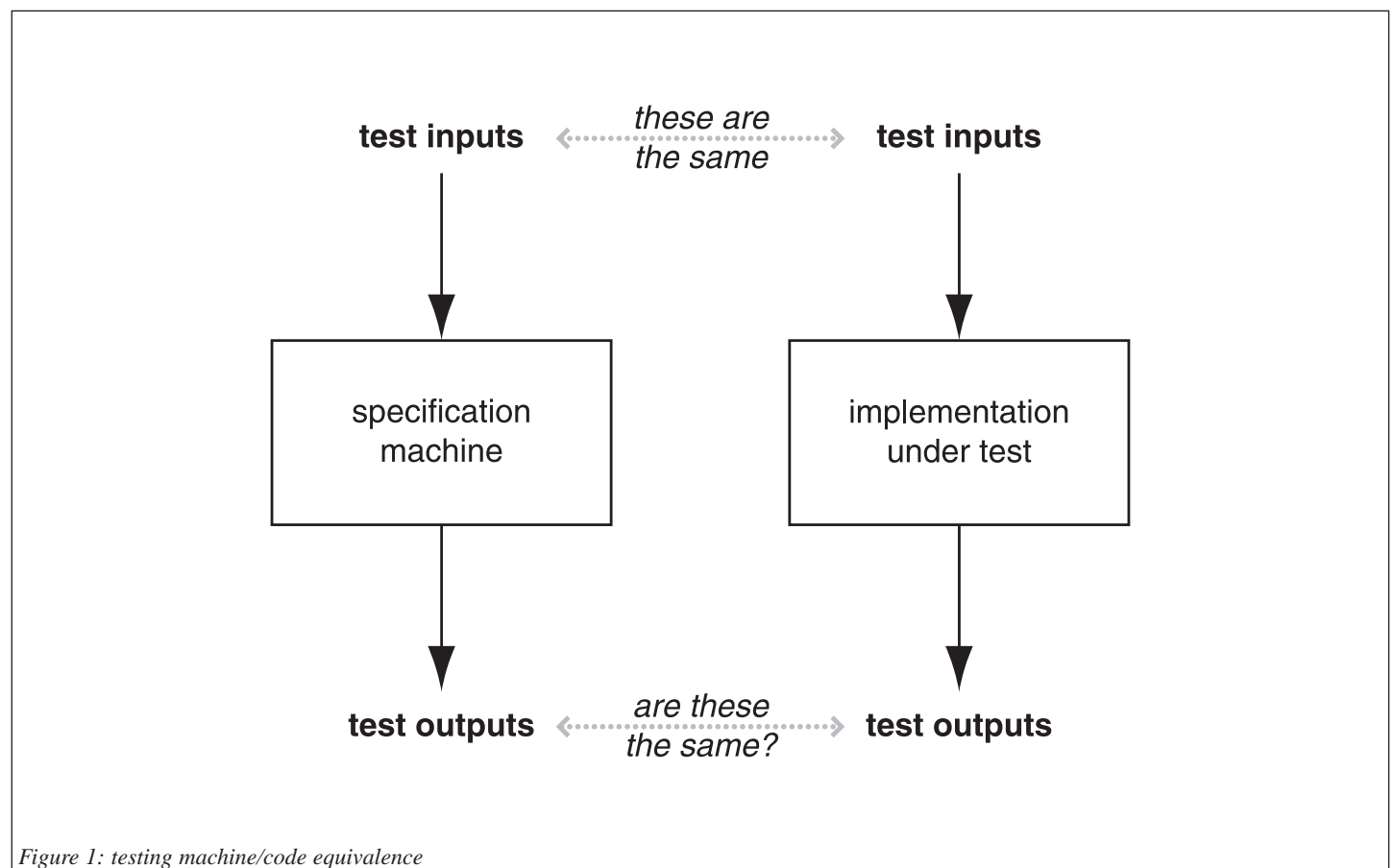
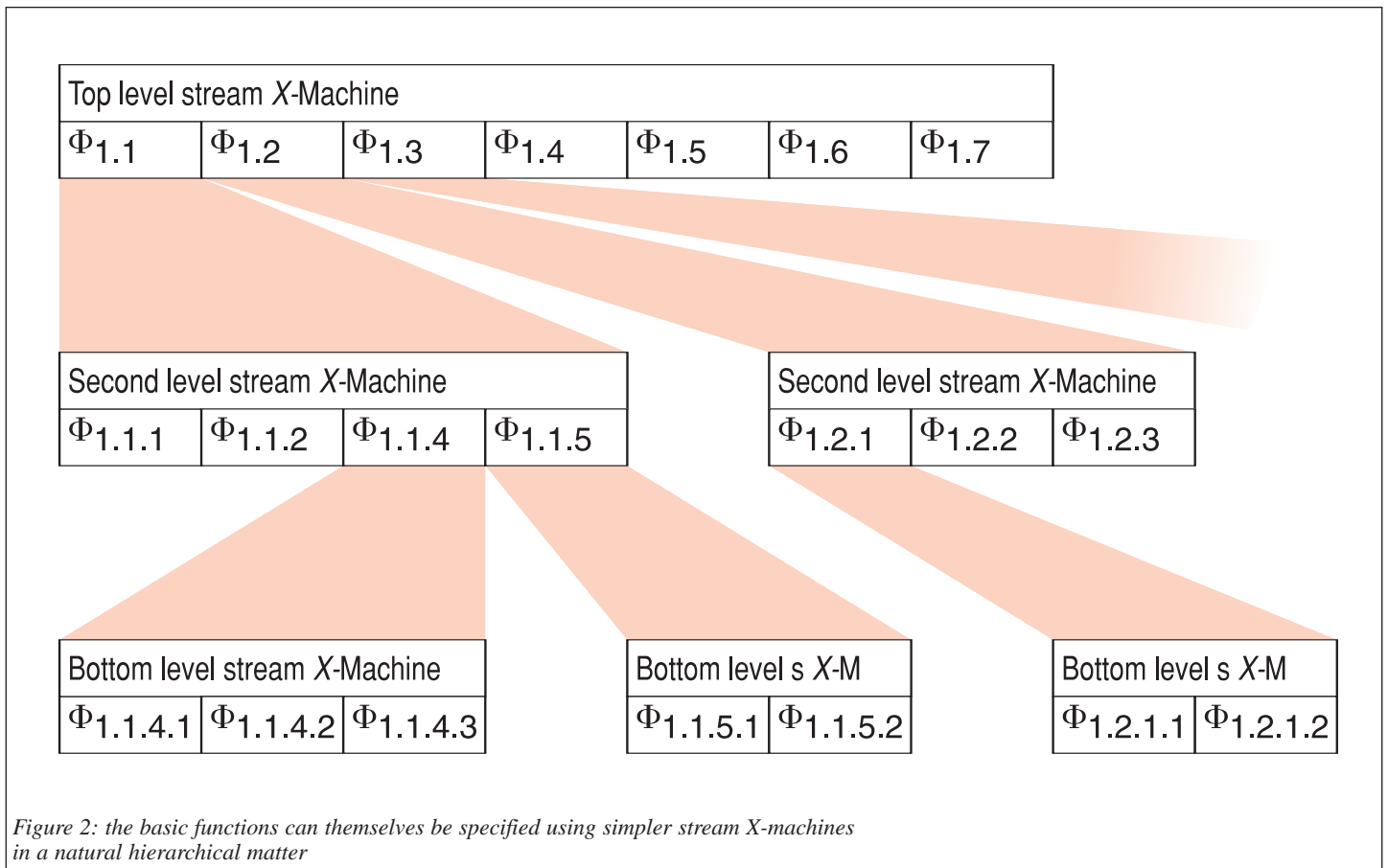


Figure 1: testing machine/code equivalence



Of these assumptions the first three lie within the capability of the designer. An algorithm for ensuring that a stream X-Machine satisfies condition 2 is given in Ipate & Holcombe [1]. Any stream X-Machine can be replaced by a machine satisfying condition 2. We have called these conditions “design for test” conditions. Without them it is going to be difficult to test a system properly; there may be hidden behavioural faults in the implementation which cannot be exposed. However, even without them the test sets produced are extremely powerful as the case study below shows. The procedures are quite straightforward and intuitive.

Condition 3 requires some comment. It is clear that the designer can arrange for the associated automata of the specification X-Machine to be minimal, standard techniques from finite state machine theory are available. The problem remains with the requirement that the implementation’s associated automata is minimal. Since we do not have an explicit description of the implementation as an X-Machine we cannot analyse its associated automata to see if it is minimal. We do know, however, that there is a minimal automata with the same behaviour as the automata of the implementation. It is this that will feature in the application of the fundamental theorem. Thus we have a test set that determines whether the behaviour, that is the function computed, by the specification equals the function computed by the implementation - providing that both implementation

X-Machine and the specification X-Machine have the same basic function set Φ .

The final condition is the most problematical. Establishing that the set of basic functions, Φ , for the implementation is the same as the specification machine’s has to be resolved, however. In practice this will be done with a separate testing process, either an application of the method explained above since the basic processing functions are computable and thus expressible as the computations of other, presumably much simpler, X-Machines or by using some other testing method for testing simple functions - perhaps the category-partition method or a variant. If the basic processing functions are tried and tested with a long history of successful use - perhaps they are standard procedures, modules or objects from a library - then their individual testing could perhaps be assumed done. If we assume that the Φ s are implemented correctly this carries with it the consequence that the implementation is a stream X-Machine since these functions are the processing functions of a stream X-Machine by construction.

In likely applications of the method we will successively apply the test method to the hierarchy of stream X-Machines that are created when we consider the basic functions Φ at each level (figure 2). Thus, testing a specific function ϕ will involve considering it as the computation defined by a simpler stream X-Machine and so on.

Ultimately, at the bottom level we need to test the basic functions in some suitable way - or assume that they are implemented correctly.

In many of the case studies that we have looked at the basic functions that need to be used are typically very straightforward ones that carry out simple tasks on simple data structures, inserting and removing items from registers, stacks etc, carrying out simple arithmetic operations on simple types and processing character strings in well understood ways. There is probably little point in devoting a great deal of testing effort to this aspect of the problem; it may seem to be slightly dangerous to say so but we know how to implement such simple operations correctly.

The benefits that accrue if the method is applied are that the entire control structure of the system is tested and *all* faults detected *modulo* the correct implementation of the basic functions.

There will clearly be a need to develop the method to deal with large scale systems and a refinement process which allows the development of components and their linking together will be needed. This will be the subject of the final article in this series.

An industrial case study

A colleague, Salim Vanak, [2], spent several months in a well known company that design systems for mass market consumer products. They have a sophisticated approach to testing since the cost of recall due to serious

Mutation scores			
Mutation operator	Mutations	X-machine	in-house
IF Branch Mutation	49	0.861	0.556
CASE Label Mutation	14	0.818	0.364
IF Condition Mutation	38	0.842	0.684

faults is colossal. The main technique used for test generation is a combination of structural and random testing. We worked alongside their engineers building X-Machine test sets for their latest systems. The first task was to construct X-Machines that described their systems. This was straightforward, we used their source code together with some 'rough' sketches which were based on state machines, as it happened. The systems were complex with considerable concurrency involved in their operation.

Having designed the X-Machines the test sets were generated automatically. These test sets were then compared with the in-house test sets in the following way. Individual mutation faults were injected into the source code and the two test sets were then applied to these mutated programs. The performance of the two test sets was compared and a mutation score based on error seeding calculated.

Error seeding is a metric derived from mutation testing. The process involves taking the code under test, introducing errors or mutations at various points and then executing the test set on this mutated code. If the test set finds the introduced fault the mutation is considered "dead". This seeding of errors is continued for a number of iterations. The result is a ratio of the number of dead mutants to total mutants.

The mutation score is usually shown as a percentage and is given by the following formula.

$$\text{Score } S = (D - E_d) / (M - E_m)$$

where D is the number of dead mutants, E_d is the number of equivalent mutants that are also dead, M is the total number of mutants and E_m is the number of equivalent mutants.

The in-house test set is made up of two parts: a structured test set and a random test set. The structured test set takes 1-2 hours to execute. The random test set is generated by a C program that constructs test cases by randomly sending different types of values through the system. The X-Machine test set does not require the complete internal details of the system. As a result, the design time is significantly smaller than for the in-house test

set. The X-Machine test set is designed to explore the state space and transition structure of the system under test resulting in a test set that executes much faster than the complete in-house test set, taking only minutes as opposed to the overnight run the in-house test set requires.

Coverage results for the X-Machine test set are encouraging. Statement and Branch coverage are all over 94%. The basic sub-condition coverage is also over 90%.

Note that the full X-Machine test method could not be applied because the design for test conditions were not met. The method was applied to existing code and this was not designed to satisfy the design for test conditions. Despite this the results were significantly better than the inhouse approach.

“The mutation scores give some evidence, of an industrial-strength nature, that the X-machine approach is both powerful and practical”

Mutation scores

The mutation scores for the X-Machine test set are higher than the mutation scores for in-house test set. All the faults missed by the X-Machine test set were also missed by the in-house test set. The table shows a breakdown of the mutation scores for each mutation operator. There is little anomalous behaviour in these results and every thing is as predicted. X-Machine mutation scores for all three mutation operators are high. The in-house test set shows a very low score for case label mutations. However this is only 7 of the 28 live

mutants left by the in-house test set that were killed by the X-Machine test set. Increasing the size of the random test set had no effect on the mutation scores for the in-house test set.

This gives some evidence, of an industrial strength nature, to the claim that the X-Machine approach is both powerful and practical. There were some faults that were not picked up by the X-Machine test set but that is because the design for test conditions were not met.

References

- [1]F. Ipate and M. Holcombe, "A method for refining and testing generalized machine specifications." Int. Jour. Comp. Math 68, 197-219, 1998.
- [2]S.Vanak, Complete Functional Testing of Hardware Descriptions, PhD thesis, University of Sheffield, 2002.

Appendix: formal statement of the main theorem

The fundamental theorem of testing (see [1]): Let M and M' be two deterministic stream X-Machines with Φ output-distinguishable and complete which compute f and f' respectively and $t : \Phi^* \rightarrow \Sigma^*$ be a fundamental test function of M .

Let \mathbf{T} and \mathbf{W} be a transition cover and a characterisation set, respectively, of the associated automaton A of M and put

$$\mathbf{Z} = \Phi^k \mathbf{W} \cup \Phi^{k-1} \mathbf{W} \cup \dots \cup \mathbf{W}.$$

If A and A' are minimal, $\text{card}(Q') - \text{card}(Q) \leq k$ and $f(s) = f'(s) \forall s \in t(\mathbf{TZ})$,

then the associated automata A and A' are isomorphic.

So, $t(\mathbf{TZ})$ is the test set we are seeking.

The value of k is chosen to represent the difference between the known state size of the specification and the (unknown) state size of the implementation.

In practice this is not usually large, for especially sensitive applications one can make very pessimistic assumptions about k at the cost of a large test set.

We must make the following further assumptions for the method to work and for the conclusion - that ALL faults are detected:

- 1 The specification is a deterministic stream X-Machine;
- 2 The set of basic functions Φ is output-distinguishable and complete;
- 3 The associated automata are minimal;
- 4 The implementation is a deterministic stream X-Machine with the same set of basic functions Φ .

The final article in this series will appear in the next issue