# 8

# Basing test coverage on a formalization of test hypotheses

*O. Charles, R. Groz*
*FRANCE TÉLÉCOM - CNET,*
*DTL/MSV, 2 avenue Pierre Marzin, F-22307 LANNION Cedex,*
*FRANCE, Tel: +33 2 96 05 37 90, Fax: +33 2 96 05 39 45,*
*E-mail: (charleso, groz)@lannion.cnet.fr*

**Abstract**

This paper defines a characterization of protocol conformance test coverage based on test hypotheses. All test selection methods and coverage computation make use of test hypotheses in one way or another. Test hypotheses are assumptions made on the implementation, which justify the verdict of conformity provided by testing; thus they are an important part of the coverage. We propose a model of these hypotheses based on functions on automata, enabling a definition of coverage based on test hypotheses, which we call TH-based coverage.

**Keywords**

Conformance testing, coverage, test hypothesis

## 1 INTRODUCTION

Many test coverage measures have been defined in the area of software testing and especially protocol testing. The aim of those definitions is to measure the quality of the test suite. The notion of quality of a test suite for conformance testing can be defined in two complementary ways. On the one hand, it is the ability of the test suite to prove that the implementation conforms to its specification. On the other hand, it is the ability of the test suite to find faults. As a result, we can split coverage definitions into two families. We call the first one specification coverage and the

Testing of Communicating Systems, M. Kim, S. Kang & K. Hong (Eds)
Published by Chapman & Hall 

second one fault coverage.

Specification coverage reflects how the test suite probes the specification. These measures are usually expressed as a percentage of elements of the specification that are exercised by a test suite. For example in the case of traditional software testing the usual coverage measure are branch coverage, path coverage and more generally the different criteria typically defined by (Weyuker, 1984), (Rapps, 1985). For protocol testing, (Ural, 1991) defines similar criteria. The metric based theory of coverage (Vuong, 1991) also expresses to what extent the test suite explores the specification. Specification coverage is practical since the test suite is compared to the specification (Groz, 1996) but gives little information on residual faults.

Fault coverage gives the number and the types of faults that can be discovered by a test suite. The types of faults that may have been introduced while implementing are supposed to be known. They are listed in a fault model and the fault coverage is evaluated with respect to this model. The computation is performed either by mutation analysis (Bochmann, 1991) (Dubuc, 1991) (Motteler, 1993) (Sidhu, 1989), by numbering mutants (Yao, 1994), or generating indistinguishable implementations (Zhu, 1994). The test coverage defined in (ISO, 1995) is fault coverage.

In fact, both types of coverage measures rely on test hypotheses (Gaudel, 1992) to reduce the infinite set of possible implementations to a smaller one in which the coverage will actually be computable. For example the number of states of the implementation or the probable faults are supposed to be known, but many other properties may be assumed on the implementation.

In practice, test designers also use (albeit implicitly) test hypotheses when designing test suites. Indeed, if they were to develop test suites discovering all errors, those test suites would be infinite. They also assume that the implementations have good properties to limit the field of their investigation.

Since fault coverage, specification coverage, and test selection all resort to some sort of test hypotheses, we propose a new coverage definition based on hypotheses. The TH-based coverage (coverage based on test hypotheses) of a test suite is defined as the series of hypotheses that must be made on the implementation such that the test suite is perfect. The main advantage of this coverage is that it provides a unifying concept for the other notions of coverage. Furthermore, it provides richer information than the usual percentage computations, and test hypotheses are meaningful for test designers. However, one problem remains: how can we formalise test hypotheses? We have already presented the first ideas of this work in (Charles, 1996). In this paper we present a more elaborate formalism based on functions on automata whereas in the previous paper the hypotheses were given as trace sets. Thus this model gives a better level of abstraction, closer to testing practice.

In section 2 we give a formalism of test hypotheses and test hypotheses coverage. Our goal is not to study systems in order to extract hypotheses (this work has already been done (Bernot, 1991)(Gaudel, 1992)(Phalippou, 1994)) but to catch this concept in a formal and suitable way for TH-based coverage. In section 3 we recall the IOSM model defined by (Phalippou, 1994). In section 4 it is shown that hypotheses can be viewed as trace sets. In section 5 we introduce a data structure — the partial automata — to store the information on the implementation gathered either by testing or by hypotheses. In section 6 we study on a few examples how hypotheses can be viewed as functions on partial automata. This is formalised in section 7. Finally we show in section 8 that our model embodies the natural concept of strength of hypotheses.

# 2 PRINCIPLES OF TH-BASED COVERAGE

## 2.1 Basic test suites properties

In this section, we recall the basic formal notions that underlie testing (ISO, 1995). The goal of a test suite is to assess the correctness of an implementation with respect to its specification. The correctness is defined by means of a conformance relation **imp** on $Imp \times Spec$; an implementation $I \in Imp$ conforms to its specification $S \in Spec$ if and only if $\mathbf{imp}(I, S)$.

Let $K$ be the set of all possible tests and $T \subseteq K$ the test suite; we denote $pass(I, T)$ the fact that an implementation $I$ executes successfully a test suite $T$ (an instance of this relation will be defined in section 4.1). (ISO, 1995) defines three properties on test suites:

*definition 1* Let $T \subseteq K$ be a test suite and $S \in Spec$ a specification:

- $T$ is exhaustive $=_{def} (\forall I \in Imp)\ (pass(I, T) \Rightarrow \mathbf{imp}(I, S))$
- $T$ is sound $=_{def} (\forall I \in Imp)\ (\neg pass(I, T) \Rightarrow \neg \mathbf{imp}(I, S))$
- $T$ is complete $=_{def} T$ is sound and exhaustive.

## 2.2 Test hypotheses

Test suites for real systems are seldom exhaustive and therefore are usually incomplete. Those systems are indeed very complex and an exhaustive test suite, assuming one exists, would be infinite. Test designers resort to test hypotheses to build a smaller test suite that keeps the same properties under these hypotheses.

Test hypotheses have been proposed for test selection. (Bernot, 1991) proposes suitable hypotheses to generate finite test sets for implementation that would request infinite ones otherwise. In (Phalippou, 1994) hypotheses for the IOSM model are proposed. In this paper we shall use uniformity hypotheses to illustrate our discussion (see 6.1). Such hypotheses claim that if the implementation behaves correctly for some elements of a given domain, then it behaves correctly on the whole domain. We can also mention regularity, reliable reset, independence or fairness hypotheses. Since our goal is not to study these hypotheses but to give a suitable representation of them in the aim of defining test coverage, we shall not detail any further. However we can mention that test hypotheses are always defined in the same way: «if the implementation is correct for some behaviours, then it is also correct for a larger set». As a result, by a successive composition of the hypotheses, the valid behaviour domain grows.

The idea of TH-based coverage is based on that practical use of hypotheses: the formulation of test hypotheses is an iterative process starting from the test set (containing the only behaviours known to be correct) and ending in the exhaustive test set. According to its iterative nature, the process of formulating test hypotheses cannot take into account all the hypotheses at the same time, but one after the other. In most cases the order in which hypotheses are formulated is significant. This non-commutative property will be reflected in our model.

In other words, looking for the coverage of a test suite comes down to asking: «What is the series of assumptions that must be made such that the test suite is exhaustive for the set of implementations satisfying those assumptions?»

## 2.3 TH-based coverage

For the moment, let us regard an hypothesis $H$ as a predicate on the set of implementations $Imp$. We shall see further that in fact $H$ can be the result of the composition of a series of hypotheses $\{H_i\}_{i \in N}$.

*definition 2* Let $T \subseteq K$ be a test suite, $S \in Spec$ a specification and $H$ an hypothesis:

- $T$ is exhaustive under hypothesis $H$
  $=_{def} (\forall I \in Imp)\ (pass(I, T) \wedge H(I) \Rightarrow \mathbf{imp}(I, S))$.
- $T$ is complete under hypothesis $H$
  $=_{def} (\forall I \in Imp)\ (pass(I, T) \wedge H(I) \Leftrightarrow \mathbf{imp}(I, S))$

The hypothesis $H$ expresses to what extend $T$ is exhaustive. This is the basis of what we have called TH-based coverage. Instead of considering the coverage to be a percentage of fired transitions, executed branch conditions, or killed mutants, we propose to define the test coverage as the assumption that must be made on the implementation under test such that the test suite is exhaustive under.

*definition 3* Let $H$ be an hypothesis, $T \subseteq K$ a test suite and $S \in Spec$ a specification. $H$ is a coverage of $T =_{def} T$ is exhaustive under hypothesis $H$.

# 3 MODEL DEFINITION

In order to see how our general framework for TH-based coverage can be instantiated in the case of protocols and communicating systems, we shall base the rest of this paper on a model suitable for this domain.

## 3.1 Input-Output State Machines

Input-Output State Machines (IOSM) have been presented in (ISO Annex A, 1995) as a more fundamental model of systems than the standardised syntactic languages such as Estelle or SDL. Despite there exists a lot of models for communicating systems we choose this one because the notion of test hypothesis has been studied in this framework (Phalippou, 1994).

*definition 4* An Input-Output State Machine is a 4-tuple $\langle S, L, T, s_0 \rangle$ where:

- $S$ is a finite non-empty set of states;
- $L$ is a finite non-empty set of interactions;
- $T \subseteq S \times ((\{?, !\} \times L) \cup \{\tau\}) \times S$ is the transition relation. Each element from $T$ is a transition, from an origin state to a destination state. This transition is associated either to an observable action (input ?a or output !a), or to the internal action $\tau$.
- $s_0$ is the initial state of the automaton.

We give also the following definitions and notations.

*definition 5* Let $S = \langle S_s, L_s, T_s, s_{0s} \rangle$ and $(\sigma = \mu_1 \ldots \mu_n) \in (\{!, ?\} \times L_s)^*$.

- $(s_0, \sigma, s_n)$ iff $(\exists (s_i)_{1 \le i < n} \in S_s^n)\ (\forall i, 1 \le i \le n)\ ((s_{i-1}, \mu_i, s_i) \in T_s)$
- $(S, \sigma, s_n)$ iff $(s_0, \sigma, s_n)$
- $(s_0, \varepsilon, s_1)$ iff $s_0 = s_1$ or $(\exists n \ge 1)\ (s_0, \tau^n, s_1)$
- $(s_0, \hat{\mu}, s_1)$ iff $(\exists s_2, s_3 \in S_s)\ ((s_0, \varepsilon, s_2) \wedge (s_2, \mu, s_3) \wedge (s_3, \varepsilon, s_1))$
- $(s_0, \hat{\sigma}, s_n)$ iff $(\exists (s_i)_{1 \le i < n} \in S_s^n)\ (\forall i, 1 \le i \le n)\ ((s_{i-1}, \hat{\mu}_i, s_i) \in T_s)$

*definition 6* A trace of S is a sequence of observable actions $\sigma \in (\{!, ?\} \times L_s)^*$ such that $(\exists s_n \in S_s)\ (s_{0s}, \hat{\sigma}, s_n)$. The set of all the traces is denoted $Tr(S)$.

The set *Spec* is chosen to be the set of all IOSM. The test hypothesis (ISO, 1995) allows us to choose also the set of all IOSM as the set *Imp*.

### 3.2 Conformance relations

Many conformance relations have been defined on $IOSM \times IOSM$ to suit testing practice. We list here three of them that are suitable for **imp**.

- $I \ge_{tr} S =_{def} Tr(S) \subseteq Tr(I)$
- $I \le_{tr} S =_{def} Tr(I) \ldots Tr(S)$
- $R_5(I, S) =_{def} (\forall \sigma \in T_i\ S))\ (\sigma \in Tr(I) \wedge (O(\sigma, I) = O(\sigma, S)))$ where $O(\sigma, I) = \{a \in L | \sigma ! a \in Tr(I)\}$ is the set of outputs of $I$ after $\sigma$. This relation is used in our automatic test generation tool TVEDA (Clatin, 1995).

## 4 TEST CONCEPTS

### 4.1 Test verdict and tests as traces

(ISO, 1995) tells us that during the execution of a test case all that can be observed are the interactions exchanged between the implementation and the tester. A test verdict is assigned thanks to the observed execution trace. But from the coverage point of view the exchanged interactions are more informative than the verdict itself. In other words, the observed trace gives more information about the implementation than the verdict assigned to this trace.

That is the reason why in this paper we shall adopt the view that one successful test case execution reveals exactly one trace of the implementation. Moreover we shall merge test cases and test executions by choosing the set of traces $(\{!, ?\} \times L_S)^*$ as the set of tests $K$ where $L_S$ is the set of interactions declared in the specification. Now we can define successful test case executions and successful test suite executions.

*definition 7* Let $t \in (\{!, ?\} \times L_S)^*$ be a test case and let $T \subseteq (\{!, ?\} \times L_S)^*$ be a test suite

- $pass(I, t) =_{def} t \in Tr(I)$

- $pass(I, T) =_{def} T \subseteq Tr(I)$

Let us remark that because of this definition, the inputs and outputs ($\{!, ?\}$) of test cases are defined from the implementation point of view and no longer from the tester. For example the test case $t = ?a!b$ means that an interaction $a$ is sent by the tester to the implementation and $b$ is received by the tester from the implementation.

## 4.2 Viewing test hypotheses as trace sets

Remember that an hypothesis $H$ is a coverage of a test suite $T$ iff $(\forall I \in Imp)\ (pass(I, T) \wedge H(I) \Rightarrow \mathbf{imp}(I, S))$. According to definition 7, this is equivalent to: $(\forall I \in Imp)\ (T \subseteq Tr(I) \wedge H(I) \Rightarrow \mathbf{imp}(I, S))$.

Let us instantiate **imp** with the conformance relations listed in 3.2. We can see that they are based in a way or another on a comparison between $Tr(I)$ and $Tr(S)$. Thus, making an hypothesis $H$ covering a test suite $T$ comes down to assuming that some traces are in the implementation and some others are not. For example let us instantiate **imp** with :

- $I \geq_{tr} S$

  $H$ is a coverage of a test suite $T$ iff
  $(\forall I \in Imp)\ (T \subseteq Tr(I) \wedge H(I) \Rightarrow Tr(S) \subseteq Tr(I))$.
  In that case we get : $H$ is a coverage of a test suite $T$ iff
  $(\forall I \in Imp)\ (H(I) \Rightarrow Tr(S) - T \subseteq Tr(I))$.
  Thus, for the conformance relation $I \geq_{tr} S$, making an hypothesis covering a test suite comes down to assuming that the traces of $S$ that are not tested are also in the implementation.
  Proof:
  *if*: $(\forall I \in Imp)\ (H(I) \Rightarrow Tr(S) - T \subseteq Tr(I))$ implies
  $(\forall I \in Imp)\ (T \subseteq Tr(I) \wedge H(I) \Rightarrow T \subseteq Tr(I) \wedge Tr(S) - T \subseteq Tr(I))$ implies
  $(\forall I \in Imp)\ (T \subseteq Tr(I) \wedge H(I) \Rightarrow Tr(S) \subseteq Tr(I))$. Thus $H$ is a coverage of $T$.
  *only if*: Assume there exists $I$ such that $H(I) \wedge \quad Tr(S) - T \not\subset Tr(I)$. That implies $\quad Tr(S) \not\subset Tr(I) \quad$. Thus $I$ is not conformant and consequently $H$ is not a coverage of $T$.

- $I \leq_{tr} S$

  $H$ is a coverage of a test suite $T$ iff
  $(\forall I \in Imp)\ (T \subseteq Tr(I) \wedge H(I) \Rightarrow Tr(I) \subseteq Tr(S))$.
  In that case we get :
  $H$ is a coverage of a test suite $T$ iff $(\forall I \in Imp)\ (H(I) \Rightarrow Tr(I) \subseteq Tr(S))$.
  Thus for this conformance relation, making an hypothesis covering a test suite implies that the traces of $I$ are included in the traces of $S$, or in other words,

$((\{!, ?\} \times L_S)^* - Tr(S)) \cap Tr(I) = \emptyset$.

Proof:

*if*: trivial

*only if*: Assume there exists $I$ such that $H(I) \wedge Tr(I) \not\subset Tr(S)$. That implies that I is not conformant and thus $H$ is not a coverage $T$

We mention that we obtain a similar result with the conformance relations $R_5$ and $R_1$ ($R_1(I, S) \Leftrightarrow (\forall \sigma \in Tr(S))(\sigma \in Tr(I) \Rightarrow O(\sigma, I) \subseteq O(\sigma, S))$). $R_1$ is very close to the relation **ioco** defined on the IOLTS model (Tretmans, 1996). It shows that the results presented in this paper hold for other models than the IOSM.

To sum up this case study, we can say that for the main conformance relations used for testing, formulating a useful test hypothesis comes down to assuming that there exists a set of traces in the trace set of the implementation and another one in the complementary set of the trace set of the implementation. We shall see in further sections that this result will enable us to give a formal representation of hypotheses.

# 5 PARTIAL AUTOMATA

## 5.1 Definition

Remember that the goal of test coverage is to make up one's mind about the correctness of an implementation under test knowing that the implementation can execute correctly a test suite $T$ and that it verifies some hypotheses.

Here we define a new data structure to store the information that could be gathered from the implementation either by test or by hypotheses. Moreover this structure must allow us to determine when we have enough information to give a verdict about the correctness of the implementation.

We have seen in the previous section that making hypothesis implies that some traces are included in the implementation and some others are not. We have assumed that both the specification and the implementations can be modelled by automata. Consistently, we shall consider only hypotheses corresponding to regular sets of traces. Therefore, we shall represent these traces on an automaton.

However, this automaton should be able to represent some traces that are known to belong to the implementation and some others that are known not to belong to the implementation. The first point is easy: is it sufficient that some traces of this automaton (what we call from now on partial automaton since it gives a partial view of the implementation under test) are exactly the traces which have been identified as belonging to the IUT. For the second point, the traces «outside» the implementation are also given as traces of the partial automaton but are distinguished from the others by finishing in a particular state of the partial automaton, denoted *out*. Moreover, we know that if $\alpha \in (\{!, ?\} \times L_s)^*$ is not a trace of $I$ then any extension of $\alpha$ is not a trace of $I$ as well. As a result, no transition can go from *out* to another state and there is a loop-transition on *out* labelled by each observable interaction. We also know that an implementation modeled as an IOSM can never refuse an input (Phalippou, 1994)(Tretmans, 1996), that is to say it is always possible to send something to the implementation. It means that $\alpha \in Tr(I)$ implies

$(\forall a \in L_s)\ (\alpha?a \in Tr(I))$ . So the transitions reaching *out* from another state are necessary labelled by an output.

*definition 8* A partial automaton $Ip$ is an IOSM $\langle S_{Ip}, L_{Ip}, T_{Ip}, s_{0Ip}\rangle$ with a particular state denoted *out* and verifying

- $(\forall \mu \in \{!, ?\} \times L_{Ip})\ (\forall s_j \in S_{Ip})\ ((out, \mu, s_j) \in T_{Ip} \Rightarrow s_j = out)$
- $(\forall \mu \in \{!, ?\} \times L_{Ip})\ ((out, \mu, out) \in T_{Ip})$
- $(\forall \mu \in \{!, ?\} \times L_{Ip})\ ((\exists s \in S_{Ip})\ ((s, \mu, out) \in T_{Ip}) \Rightarrow \mu \in \{!\} \times L_{Ip})$

*definition 9* We denote *IOSMp* the set of partial automata.

Now we have to define which are the implementations that verify the information stored in a partial automaton *Ip*. We call these implementations the candidate implementations because they stand as candidate for being the implementation under test with respect to the constraints stored in the partial automaton. According to the above discussion, each candidate must have the traces of *Ip* that do not end in *out* and must not have the others.

*definition 10* An implementation $I$ is a candidate w.r.t. the partial automaton $Ip$, and we note $cand(Ip, I) =_{def} (\forall \sigma \in Tr(Ip))$

$$([(Ip, \sigma, out) \Rightarrow \sigma \notin Tr(I)] \wedge [(Ip, \sigma, s) \wedge s \neq out \Rightarrow \sigma \in Tr(I)])\ .$$

## 5.2 Basic ideas on how partial automata work

Before giving some new definitions we give some informal clues on how we shall use partial automata.

The partial automaton embodies test cases and test hypotheses in the shape of traces. Assume the hypotheses to be formulated one after the other, then each hypothesis is a function that adds new traces to the partial automaton. If one of these functions adds one new trace that fall in the *out* state, it corresponds to an hypothesis that assumes that this trace is not in the IUT. The other added traces correspond to an hypothesis that assumes that these traces are in the implementation.

Thus, at the beginning of the process the partial automaton contains only the traces of the tests, and is enriched as soon as the hypotheses are applied.

## 5.3 Initial partial automaton

The initial partial automaton is a partial automaton storing nothing but the test set. Since practical test sets are finite, the initial partial automaton has neither loop-transition nor cycles, but there exists many corresponding partial automaton. We choose the minimal tree given by the following algorithm.

*definition 11* Let $T = \{\mu_{11}\ldots\mu_{1n_1}, \ldots, \mu_{i1}\ldots\mu_{ij}\ldots\mu_{in_i}, \ldots, \mu_{m1}\ldots\mu_{mn_m}\}$ be a test set. The initial partial automaton of $T$ is the IOSMp $Ip_0 = \langle S_{Ip}, L_{Ip}, T_{Ip}, s_{0Ip}\rangle$ given by the following algorithm:

1. $S_{Ip} := \{s_{0IP}, out\}$, $L_{Ip} := L$,
   $T_{Ip} := \{(out, \mu, out) \mid \mu \in \{!, ?\} \times L\}$

```
2. for i from 1 to m
   . current-state:= s0Ip
   . for j from 1 to ni
     if (∃s ∈ S_Ip) ((current − state, μ_ij, s) ∈ T_Ip)
     then
       . current-state := s
     else
       . S_Ip := S_Ip ∪ {s_ij}
       . T_Ip := T_Ip ∪ {(current − state, μ_ij, s_ij)}
       . current-state := sij
     end if
```

For example, the initial partial automaton of the test *T={?a!b?c!d, ?a!b?e!f, ?a!d, ?e!b}* is given on figure 1.

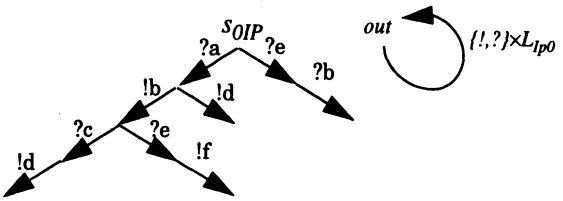



**Figure 1** Initial partial automaton of tests suite {?a!b?c!d, ?a!b?e!f, ?a!d, ?e!b}.

## 6 EXAMPLES OF HYPOTHESES VIEWED AS FUNCTIONS

Along this example and before generalization, we shall use the conformance relation $I \geq_{tr} S$. Thus all significant hypotheses will be formulated as a trace additions (see 4.2 1.). As a result we shall not need the *out* state (see 5.1). So it will not appear on the figure of this example but it must be kept in mind that it exists.

Let us consider the specification *S* given in figure 2. A conforming implementation of *I* is a system that must at least respond *!x* after receiving *?a*, *?b* or *?c* and then going back to its initial state after receiving *?r* (*?r* for reset).

Testing an implementation with *T={?b!x?r}* is far from being sufficient to check that *I* conforms to *S* w.r.t. the conformance relation $I \geq_{tr} S$. All that can be said of *I* is that *I* passes *T* and then the elements of *T* are traces of *I*. Than can be summed up in the initial partial automaton $Ip_0$ (see figure 2).

Let us consider both following hypotheses:

- The implementation is uniform on the set of interactions $\{?a, ?b, ?c\}$ that is to say its behaviour is the same for these three interactions ((Bernot, 1991)(Phalippou, 1994)).
- The reset is correctly implemented, that is to said it actually brings back the implementation to its initial state.

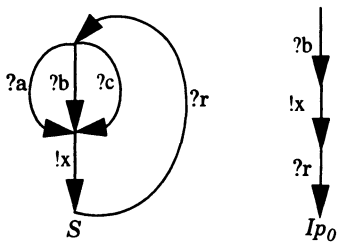



**Figure 2** Specification S and initial partial automaton of T={?b!x?r}.

## 6.1 Uniformity hypothesis

From the trace point of view, the uniformity hypothesis on $\{?a, ?b, ?c\}$ tells us that if there exist $\alpha, \beta \in (\{!, ?\} \times L_s)^*$ and $\mu \in \{?a, ?b, ?c\}$ such that $\alpha\mu\beta \in Tr(I)$ , then $(\forall \mu' \in \{?a, ?b, ?c\})\ (\alpha\mu'\beta \in Tr(I))$ .

In our example we know that $?b!x?r \in Tr(I)$ . Thanks to this hypothesis, we can assume that we also have $?a!x?r \in Tr(I)$ and $?c!x?r \in Tr(I)$ . These two new traces must be added to the partial automaton since they are new pieces of information. $Ip_1$ (see figure 3 a) ) is the resulting partial automaton. We can notice that the gap between $Ip_1$ and $Ip_0$ is bridged by adding two new transitions labelled respectively *?a* and *?c*.

It is easy to imagine that for other examples the mechanism of transformation of a partial automaton into another partial automaton that takes the uniformity hypothesis into account will always duplicate the missing transitions. Thus we can represent this by a function on partial automata that performs the addition of transitions. For example, the uniformity hypothesis on *{?a,? b, ?c}* can be given as a function $U^{[abc]} : IOSMp \rightarrow IOSMp$ defined by $Ip' = U^{[abc]}(Ip)$ where:

$S_{Ip'} = S_{Ip}$, $L_{Ip'} = L_{Ip}$, $s_{0_{Ip'}} = s_{0_{Ip}}$ and

$$T_{Ip'} = T_{Ip} \cup \{ (s_i, ?a, s_j), (s_i, ?b, s_j), (s_i, ?c, s_j) \mid (s_i, ?a, s_j) \in T_{Ip} \vee (s_i, ?b, s_j) \in T_{Ip} \vee (s_i, ?c, s_j) \in T_{Ip} \}$$

## 1.1 Reliable reset hypothesis

The reliable reset hypothesis assumes that each time a *?r* interaction is sent to the implementation, it normally goes back to its initial state. Actually on our example that implies that the test set *T={?b!x?r}* is equivalent to *T'={?b!x?r, ?b!x?r?b!x?r, ?b!x?r?b!x?r?b!x?r..}*. The partial automaton $Ip_1'$ of this test set is given on figure 3 b).

As we have already noticed for the uniformity hypothesis, the reliable reset hypothesis can be viewed as a transformation of partial automata. Here we can see on figure 3 b) that we can go from $Ip_0$ to $Ip_1'$ by looping the transition labelled by

*?r*. It easy to convince oneself that the well implemented reset hypothesis will always corresponds to this transformation, no matter what the partial automaton is.

As a result we are able to define this hypothesis as a function $Ri: IOSMp \rightarrow IOSMp$ defined by $Ip' = Ri(Ip)$ where

$$S_{Ip'} = S_{Ip} - \{s_j \mid (\exists s_i \in S_{Ip})\ ((s_i, ?r, s_j) \in T_{Ip})\}$$

$$L_{Ip'} = L_{Ip}$$

$$s_{0_{Ip'}} = s_{0_{Ip}}$$

$$T_{Ip'} = T_{Ip} \cup \{(s_i, ?r, s_{0_{Ip'}}) \mid (\exists s_j \in S_{Ip})\ ((s_i, ?r, s_j) \in T_{Ip})\}$$

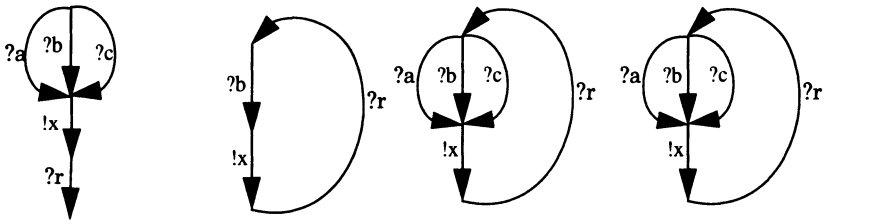
$$\cup \{(s_{0_{Ip'}}, \mu, s_k) \mid (\exists s_i, s_j, s_k \in S_{Ip})\ ((s_i, ?r, s_j) \in T_{Ip} \wedge (s_j, \mu, s_k) \in T_{Ip})\}$$

$$\cup \{(s_j, \mu, s_{0_{Ip'}}) \mid (\exists s_i, s_j, s_k \in S_{Ip})\ ((s_i, ?r, s_k) \in T_{Ip} \wedge (s_j, \mu, s_k) \in T_{Ip})\}$$

$$-(\{(s_i, ?r, s_j) \in T_{Ip}\} \cup$$

$$\{(s_j, \mu, s_k) \in T_{Ip} \mid (\exists s_i, s_j, s_k \in S_{Ip})\ ((s_i, ?r, s_j) \in T_{Ip} \wedge (s_j, \mu, s_k) \in T_{Ip})\} \cup$$

$$\{(s_j, \mu, s_k) \mid (\exists s_i, s_j, s_k \in S_{Ip})\ ((s_i, ?r, s_k) \in T_{Ip} \wedge (s_j, \mu, s_k) \in T_{Ip})\})$$




$Ip_1 = U^{[abc]}(Ip_0)$ $Ip'_1 = Ri(Ip_0)$ $Ip_2 = Ri(U^{[abc]}(Ip_0))$ $Ip'_2 = U^{[abc]}(Ri(Ip_0))$

a) b) c) d)

**Figure 3** Partial automata.

## 6.2 Combination of uniformity and reliable reset hypotheses

It's now possible to give the partial automaton which gives (according to the *cand* relation) the set of the implementations that: a) pass the test set *T*, b) satisfy the uniformity hypothesis on *{?a,?b,?c}* c) satisfy the reset hypothesis. Figures 4 c) and d) show $Ip_2 = Ri(U^{[abc]}(Ip_0))$ and $Ip'_2 = U^{[abc]}(Ri(Ip_0))$. We notice that these automata are equal, but it is not a general rule.

The implementations *i* represented by $Ip_2$ w. r. t. the *cand* relation verifies $(\forall \sigma \in Tr(Ip2))\ (\sigma \in Tr(i))$. On account on their being isomorphic, $Ip_2$ and *S* have the same trace set. As a result, $(\forall \sigma \in Tr(S))\ (\sigma \in Tr(i))$ and then all the candidate implementations conforms to *S*. We conclude that *T={?b!x?r}* is exhaustive under the uniformity hypothesis on *{?a,?b,?c}* and well implemented reset hypothesis.

# 7 FORMALIZATION

## 7.1 Hypotheses and specification

It is important that the specification may be taken into account in the functional definition of test hypotheses. For example, we define uniformity hypothesis on a fixed set *{?a,?b,?c}*. We might wish to define this hypothesis on any set; but at the same time, we would like to apply the hypothesis only in contexts (trace prefixes) where it matches some sort of uniformity within the specification. The point is that the function has to identify the uniform sets and then apply a similar algorithm to $U^{[abc]}$. We can write this function $U: IOSM \times IOSMp \rightarrow IOSMp$ where $Ip' = U(S, Ip)$ if and only if $(S_{Ip'} = S_{Ip}$, $L_{Ip'} = L_{Ip}$, $s_{0_{Ip'}} = s_{0_{Ip}}$ and $T_{Ip'}$ is constructed by the following algorithm

1. $T_{Ip'} = T_{Ip}$
2. For each $s_i \in S_{Ip}$

   If there exist $\alpha \in Tr(Ip)$, $\mu \in L_{Ip}$, $s_j \in S_{Ip}$ such that $(Ip, \alpha, s_i)(s_i, \mu, s_j)$ then,

   (since there exist $s_{s_i}, s_{s_j} \in S_S$ such that $(S, \alpha, s_{S_i})(s_{S_i}, \mu, s_{S_j})$ )

   for each $\lambda \neq \mu \in L_S$

   if $(s_{S_i}, \lambda, s_{S_j}) \in T_S$ then add $(s_i, \lambda, s_j)$ to $T_{Ip}$

We now have sufficient material to define a functional hypothesis.

*definition 12* A functional hypothesis is a function
$F: IOSM \times IOSMp \rightarrow IOSMp$ such that
$(\forall S \in IOSM)\ (\forall Ip \in IOSMp)\ (Tr(Ip) \subseteq Tr(F(S, Ip)))$.

We need to fixe $S$ in $F(S, Ip)$, therefore we note $F_S(Ip)$ for $F(S, Ip)$.

The application of a functional hypothesis to a partial automaton *Ip* reduces the set of candidate implementations. The set of eliminated candidates is
$\{I \in IOSM \mid cand(Ip, I) \wedge \neg cand(F(Ip), I)\}$

## 7.2 Stop condition

It is high time to sum up all we have seen till now. First we have seen that for the most frequently used conformance relations, making a test hypothesis comes down to assuming that some traces are in the implementation and some others are not. Secondly we introduced an original data structure – the partial automata – to store test sets and hypotheses in the shape of automata. Thirdly, we have defined test hypotheses as functions that enrich partial automata. At the beginning of the process, the partial automaton contains only the test cases (we have called it the initial partial automaton). Then, as the hypotheses are formulated, the corresponding functions are applied to the partial automaton. Now the question is: how can we make sure than we have applied enough hypotheses such that exhaustivity (under these hypotheses) is reached?

Remember that the partial automaton represents the set of candidate implemen-

tations, that is to said the set of implementations over which the implementation under test ranges provided it passes the test set and verifies the test hypothesis stored in the partial automaton. As a result, if all the candidate implementations conform, the implementation under test necessarily conforms.

Formally, let T be a test set. At first we compute the corresponding initial partial automaton $Ip_0$. Let us assume that we have a set of functional hypotheses $\{f_S^{h1}, ..., f_S^{hn}\}$. We apply these functional hypotheses one after the other on $Ip_0$. The resulting partial automaton is $f_S^{hn} \circ ... \circ f_S^{h1}(Ip_0)$. The candidate implementations all conform if and only if $(\forall I \in IOSM)\ (cand(f_S^{hn} \circ ... \circ f_S^{h1}(Ip_0), I) \Rightarrow \mathbf{imp}(I, S))$. According to the above discussion we define the exhaustivity condition in the following manner.

*definition 13* Let T be a test set and $Ip_0$ the corresponding initial partial automaton, $f_S^{h1}, ..., f_S^{hn}$ a sequence of functional hypotheses.

$(\forall I \in IOSM)\ (cand(f_S^{hn} \circ ... \circ f_S^{h1}(Ip_0), I) \Rightarrow \mathbf{imp}(I, S))$ implies T is exhaustive under the functional hypotheses $f_S^{h1}, ..., f_S^{hn}$.

Considering a sequence rather than a set of hypotheses reflects the reality since the impact (and even the expression) of new hypotheses may depend on previous ones. As a result, exhaustivity cannot be computed in an «environment» of hypotheses but for an ordered list of hypotheses. This is perfectly reflected by the non-commutative composition of functional hypotheses (as opposed to a purely logical view that would use conjunction of predicates).

This definition is not operational since it is based on the candidate implementations. It can be rewritten by using the definition of *cand* and **imp** in order to obtain a «syntactic» coverage condition (the proofs are omitted but are quite obvious).

- For the relation $I \geq_{tr} S$, $(\forall \sigma \in Tr(S))\ ((Ipn, \sigma, s) \wedge s \neq out)$ implies T is exhaustive under the functional hypotheses $f_S^{h1}, ..., f_S^{hn}$.
- For the relation $I \leq_{tr} S$ the condition is $(\forall \sigma \in L^* - Tr(S))\ (Ipn, \sigma, out)$.
- Finally for $R_5(I, S)$ we get

  $(\forall \sigma \in Tr(S))\ (\forall \mu \in O(\sigma, S))\ (\forall \mu' \in L - O(\sigma, S))$

  $(Ipn, \sigma, s) \wedge s \neq out \wedge (Ipn, \sigma\mu, s') \wedge s' \neq out \wedge (Ipn, \sigma\mu', out)$

# 8 STRENGTH OF FUNCTIONAL HYPOTHESES

## 8.1 partial order on hypotheses

It is well known that we can be more or less confident in an hypothesis. It depends on many subjective factors and particularly the knowledge we have on the implementation. We dealt we this problem in another paper (Charles, 1996). Here we pro-

pose to build a partial order on functional hypotheses based on their distinguishing power. We have seen that applying a functional hypothesis to a partial automaton comes down to reducing the set of candidate implementations. It is obvious that the more candidate implementations are eliminated, the stronger the hypothesis is. For example, imagine the hypothesis assuming the implementation conform whatever the test set. In other words, this hypothesis eliminates all non-conforming implementation from the set of candidates. It is hard to believe without testing that an implementation conform, so it is a very strong hypothesis. Conversely imagine an hypothesis that never reduce the set of candidates : it is the weakest hypothesis because it assumes no further good properties on the implementation.

*definition 14* Let $F$ and $F'$ be two functional hypotheses. $F$ is stronger than $F'$ $=_{def} (\forall Ip \in IOSMp)\ (\forall I \in IOSM)\ (cand(F(Ip), I) \Rightarrow cand(F'(Ip), I))$

If we look at the algorithms that define the functional hypotheses, we can see that this definition has a practical meaning. Consider function $U^{[abc]}$ defined in section 6.1. The underlying significance of this hypothesis is : if there exists $\alpha, \beta \in (\{!, ?\} \times L_s)^*$ and $\mu \in \{?a, ?b, ?c\}$ such that $\alpha\mu\beta \in Tr(I)$ then it can be assumed that $(\forall \mu' \in \{?a, ?b, ?c\})\ (\alpha\mu'\beta \in Tr(I))$. That is to say, testing $I$ with one the interactions of the set $\{?a, ?b, ?c\}$ is equivalent to testing with all three. But one may say that this hypothesis is too hazardous and should be better verified before being applied. For example it could be demand before assuming the whole set $\{?a, ?b, ?c\}$ that:

- two interactions of $\{?a, ?b, ?c\}$ are tested or,
- one interaction of $\{?a, ?b, ?c\}$ are tested with at least a two interaction long preamble ($\beta$ in the discussion above) or,
- two interactions of $\{?a, ?b, ?c\}$ are tested with a two interaction long preamble each, etc...

That defines three new functions on $IOSM \times IOSMp \rightarrow IOSMp$ (resp. $U_1^{[abc]}$, $U_2^{[abc]}$, $U_3^{[abc]}$). Their algorithms are not given here since they are light variations of $U^{[abc]}$.

Now it can be proved that $U^{[abc]}$ is stronger than $U_1^{[abc]}$ and $U_2^{[abc]}$, which are stronger than $U_3^{[abc]}$. Thus this relation reflects perfectly the intuition felt in the examples.

Of course, this is only a partial order. We cannot order hypotheses of different types, for example we cannot establish which of the reliable reset hypothesis or the uniformity hypothesis is the stronger. However it is the first step towards a total order and weight assignment that would reflect the strength of hypotheses as used in (Charles, 1996).

## 8.2 Possible enhancement of TH-based coverage with weight assignment

As we show in (Charles, 1996), ordering and assigning weights to hypotheses according to their strength can improve significantly the TH-based coverage definition. Indeed, for a given test suite $T$ there might exist many series of hypothesis that cover $T$. But if we know the weight (ie the strength) of each hypothesis, we can compute a global weight of a series of hypotheses that reflects how we can be confident in that series. As a result we can redefine TH-based coverage of a test suite by restricting to minimal weight series of hypotheses.

We also show in (Charles, 1996) that with a suitable weight assignment to hypotheses and with a suitable way to compute the global weight of series of hypotheses our model embodies the metric based theory of coverage of (Vuong, 1991) and (Curgus, 1993).

# 9 CONCLUSION

In this paper we have introduced the concept of TH-based coverage. We have focused on the major role of test hypotheses in usual coverage definition and test practice. This has led us to define an original data structure – the partial automata – that represents the set of the implementations – the candidate implementations – passing a test suite and verifying some test hypotheses. Thereafter we have seen that the hypotheses could be viewed as functions on partial automata. This has given us a practical way to handle test hypotheses for TH-based coverage. Finally we have introduced the notion of strength of hypotheses and explained how this could enhance the TH-based coverage definition.

It may seem paradoxical that we propose in this paper a formal and abstract test coverage measure while we claim in (Groz, 1996) that the very poor transition coverage is sufficient. In fact both approaches aims at giving pieces of information understandable and practical to test designers. In (Groz, 1996) we explained how relating the test suites to specifications through a visual tool turns to be more informative than a coverage given as a figure. In this paper we keep on thinking that test coverage must be richer than a percentage and must incorporate the know-how of test designers. Since the know-how of test designers is expressed as test hypotheses, why not mix both approaches in a visual tool that would link a test suite to a specification by means of test hypotheses?

# 10 REFERENCES

Bernot G., Gaudel MC. , Marre B. , *Software testing based on formal specifications: a theory and a tool.* Software Engineering Journal, November 1991.

Bochmann G. v. , Das A. , Dssouli R. , Dubuc M. , Ghedamsi A. , Luo G. , *Fault models in testing*, Proceedings of IWPTS, Leidschendam, October 1991.

Charles O. , Groz R. , *Formalisation d'hypothèses pour l'évaluation de la couverture de test.* Proceedings of CFIP 96, Rabat, Morocco.

Clatin M. , Groz R. , Phalippou M. , Thummel R. , *Two approaches linking a test generation tool with verification techniques*, IWPTS 95, Evry, France.

Curgus J. , Vuong S. T. *A Metric Based Theory of Test Selection and Coverage.* Proceedings of PSTV XIII, Liege, Belgium, 1993.

Dubuc M. , Dssouli R. , Bochmann G. v. , *TESTL: A Tool for Incremental Test Suite Design Based on Finite State Model,* IWPTS, Leidschendam, 1991.

Gaudel MC. , *Test Selection Based on ADT Specifications.* Proceedings of IWPTS 92, Montréal, Canada.

Groz R. , Charles O. , Renévot J. , *Relating Conformance Test Coverage to Formal Specifications.* Proceedings of FORTE 96, Kaiserslautern, Germany.

ISO/IEC/JTC1/SC21/P.54 - ITU-T SG10 Q.8, *Formal Methods in Conformance Testing,* Working Draft, March 1995. To become rec. Z.500 from ITU-T.

Motteler H. , Chung A. , Sidhu D., *Fault Coverage of UIO-based Methods for Protocol testing.* Proceedings of IWTCS 93, Pau, France.

Phalippou M. , *Relations d'implantation et hypothèses de test sur des automates à entrées et sorties,* université de Bordeaux I, 1994.

Rapps S. , Weyuker E. J. . *Selecting Software Test Data Using Data Flow Information.* IEEE Trans. on Software Engineering, vol. SE-11, NO. 4, April 1985.

Sidhu D. , Leung T. , *Formal Methods for Protocol Testing : A Detailed Study,* IEEE transactions on software engineering, vol. 15, n. 4, april 1989.

Tretmans J. , *Test Generation with Inputs, Outputs, and Repetitive Quiescence.* CTIT Technical Report series No. 96-26, ISSN 1381-3625. To be published in Software — Concept and Tools. 1996.

Ural H. , Yang B. , *A Test Sequence Selection Method for Protocol Testing,* IEEE Transactions on Communications, vol. 39, no 4, April 1991, pp. 514-523.

Vuong S. T. , Curgus J. . *On Test coverage Metrics for Communication Protocols.* Proceedings of IWPTS IV, Leinschendam. The Netherlands. 1991.

Weyuker E. J. , *The Complexity of Data Flow Criteria for Test Data Selection,* Information Processing Letters 19 (31 August 1984), pp 103-109.

Yao M. , Petrenko A. , Bochmann G. V. , *A Structural Analysis Approach to the Evaluation of Fault Coverage for Protocol Conformance Testing.* Proceedings of FORTE 94, Berne, Switzerland, 1994.

Zhu J. , Chanson S. T. , *Toward Evaluating Fault Coverage of Protocol Test Sequences.* Proceedings of PSTV XIV, Vancouver, Canada, 1994.

## 11 BIOGRAPHY

Olivier Charles is a Ph.D. student under the joint supervision of Roland Groz, André Schaff and Rachida Dssouli. He graduated from the Ecole Supérieure d'Informatique et Applications de Lorraine, University of Nancy.

Roland Groz received a Ph.D. in computer science from University of Rennes I in 1989. He currently manages a department in France Télécom - CNET devoted to research in the area of validation, specification and prototyping .