A Proposition for a Theory of Testing: An Abstract Approach to the Testing Process

Luc Bougé

The author is currently visiting Aarhus University, Denmark, supported by an INRIA grant. He will be working for CNRS after October 1983.

Mailing address: LITP, Université Paris 6 & 7,
2, place Jussieu
F-75221 Paris Cedex 05
France

Abstract

We describe an abstract model for the program testing process, based on first-order egalitary logic. We define the notion of a battery of tests for a given testing context. Its properties are studied: reliability, validity, bias and acceptability. A preorder is defined and studied, which leads to an equivalence relation among batteries of tests. This equivalence turns out to be of great interest, both theoretical and practical.

We show the application of this model to some classical questions: effective (automatic) test generation, test optimization, quality assessment of the testing process, and relationships between program proving and program testing.

Program testing, as GOODENOUGH & GERHART stated some years ago [Good 75], is certainly the most widely used way of assessing program correctness. It is also certainly one of the least studied from a theoretical viewpoint.

Program testing is a part of the program validation process. To validate a program, or better, a given implementation of this program, is to decide, with a certain level of confidence, whether this implementation is correct or incorrect with respect to a given specification. We are here mostly interested in functional specifications (a compiler specification for example). We have in an earlier work analysed the many problems arising in a rigorous approach to program validation [Boug 82a].

It has been written that program testing and program proving are essentially two complementary activities, both supporting program correctness assessment [Gerh 79]. Program proving has received for a long time some very strong and efficient theoretical interest, whereas program testing has been regarded as an empirical and less interesting problem. Some formalization attempts may be found in the scientific literature, but, to our mind, they do not seem to have tackled the problem at an abstract enough level.

The aim of this work is to present an abstract formalization for program testing notions, based on a rigorous utilization of first-order predicate logic as an underlying mathematical tool (see [Shoe 67] for a complete introduction to mathematical logic). We have designed this formalization to meet two requirements

- to be as faithful as possible to the common intuitive understanding of program testing (for a discussion about this point, see [Boug 82 b]);
- to be as faithful as possible to the internal requirements of our underlying mathematical tool, remaining at an

abstraction level enabling some powerful references to testing theory, putting into evidence their complementarity.

The first section sums up briefly some earlier work in this area of research and states the basic framework of this work: the Testing Process Diagram.

The second section recalls some basic facts about first-order egalitary logic, our basic mathematical tool. It then states our main definitions: Testing Context, and Battery of Tests for a given Testing Context. Many examples illustrate these abstract notions.

We study the basic properties of a battery of tests in the third section: Projective Reliability, Asymptotic Validity, Lack of Bias, Acceptability. Testability of specifications is discussed according to their syntactical form.

The fourth section is devoted to (pre-)ordering testing contexts and batteries of tests for a given context according to their "quality". We introduce the notions of Conservative Context Restriction and Asymptotic Sharpness. The equivalence relation deduced from the latter is extensively studied, and leads to the Fundamental Theorem of Testing: two comparable acceptable batteries of tests are in fact equivalent.

In the fifth section, all these theoretical and abstract developments are applied to formalize precisely some essential practical problems about testing: Effective Test Construction Method, Test Optimization, Testing Process Quality Assessment.

The conclusion of this work tries to assess the expressive power gained by using a sound but rather heavy mathematical tool. It makes more precise the better understanding we have got about the relationships between testing and proving. Some further research goals are described.

1. BACKGROUND: TESTING PROCESS DIAGRAM

We sum up here briefly some of the main attempts one can find in the literature to give a precise formalization of what it is to test a program. Rather than describing their very technical details, we shall emphasize the intuition, the philosophies which underlie them, in order to pick out the main ideas to build a general model for the testing process: that is the Testing Process Diagram. This modelling process can be mainly characterized by a Formal Approach and an Asymptotic Approach to the notion of testing.

1.1 GOODENOUGH & GERHART's Proposal

GOODENOUGH & GERHART [Good 75] were some of the first to proclaim the need for a testing theory bearing comparison to proving theory. As a first attempt to solve this problem, they described the following notions.

Consider a given implementation of a program, any F, working over an input domain D. Suppose that, for each data d of D, one is able to decide whether the implementation behaves correctly or not with respect to the given specification. This induces a (calculable) predicate over D, say OK, OK(d) means that the implementation behaves 'correctly' for $d \in D$. The correctness of the implementation with respect to the specification is thus expressed by $\forall d \in D$ OK(d).

In this framework, a test can be viewed as a subset of D, say T (the authors, curiously, allow an infinite subset of D to be a test in their sense). The elements of T can be called test cases, or elementary experiments.

One of the main ideas of their paper is that "test cases are chosen typically to satisfy some data selection criterion, C, where C denotes a predicate over subsets of D". T is a test if and only if C(T). Thus, a key to testing theory is to focus on the property of a criterion with respect to a certain predicate OK over D instead of considering the properties of a test.

The authors are led to define two fundamental properties for a criterion C. Reliability "refers to the consistency with which results are produced, regardless of whether the results are meaningful". It can be expressed by

$$\begin{aligned}&\text{RELIABLE}(C) = \\ &(\forall T_1, T_2 \subseteq D)\ [(C(T_1) \wedge C(T_2)) \rightarrow \\ &\qquad (\text{SUCCESSFUL}(T_1) \leftrightarrow \text{SUCCESSFUL}(T_2))]\end{aligned}$$

where $\text{SUCCESSFUL}(T) = \forall t \in T\ \text{OK}(t)$

On the other hand, Validity "refers to the ability to produce meaningful results, regardless of how consistently such results are produced". It can be expressed by

$$\begin{aligned}&\text{VALID}(C) = \\ &\quad (\forall d \in D)\ [\neg \text{OK}(d) \rightarrow (\exists T \subseteq D)\ (C(T) \wedge \neg \text{SUCCESSFUL}(T))]\end{aligned}$$

Obviously a "good" criterion (but not a good test!) should be both reliable and valid. Such a criterion is said to be IDEAL. The successful execution of any test satisfying such a criterion demonstrates the correctness of the implementation with respect to the specification.

We would like to point out some features of the above proposal. Note first that it is based on the functional behaviour of the implementation being tested (we use the word "implementation" where the authors used the word "program" to underline this fact). Correctness has nothing to do with the internal structure of the program or of the machine executing it. Then, a first-order logic-like language arises naturally as the best tool for dealing with those objects, though the reasons for this choice are not stated by the authors. Note that a sentence like $\forall d \in D\ \text{OK}(d)$ can be more clearly expressed by $\mathcal{D} \models \forall d\ \text{OK}(d)$, where $\mathcal{D}$ is the obvious structure with universe D.

The most important feature of the proposal is, to our mind, the notion of criterion and the properties which are defined about it.

Notice that building any test satisfying an ideal criterion demonstrates the correctness. So testing is viewed as a special case of proving with certain restrictions on the kind of proof that is used. It must in fact be split into a transcendental part (proving ideality of the criterion and adequacy of the test) and a calculable part (running of the test cases).

Lastly, it should be noticed that such a formalization is "anistrope" with respect to correctness. It only deals with successful execution of test cases, and does not say anything about failure. In fact, any failure demonstrates, in this case, the uncorrectness of the implementation. But we feel that this "anistropy" is a general feature in this area. The famous DIJKSTRA's statement "Program testing can be used to show the presence of bugs, but never to show their absence!" should be understood in this context. Any testing theory should focus rather on success of the test than on failure, because only the former is actually informative.

1.2 Criticisms and Improvements

Several authors, following the above work, have described related notions. WEYUKER [Weyu 80a] has shown the extreme importance of the socalled Oracle Problem captured above by the predicate OK. One must be able, in a sensible way, to decide whether a given test has been passed or not. It follows that a test should contain at most a finite number of test cases, the successful execution of each being decidable. This will lead us to the notion of an experiment.

WEYUKER & OSTRAND [Weyu 80b] point out the central problem in the testing area, that is to infer an infinitary conclusion ($\forall d \in D$ OK(d)) from some finite knowledge ($\forall d \in T$ OK(t)), where T is a finite subset of D. We thus need some a priori infinitary hypothesis. They claim the need to part the domain D into subdomains that are uniform with respect to correctness. Whenever any data in such a subdomain is correctly handled, so are all of

them. A uniform subdomain D_i of D is thus such that $[(\exists d \in D_i\ OK(d)) \rightarrow (\forall d \in D_i\ OK(d))]$ or, more clearly, $\mathcal{D} \models [\exists d\ (d \in D_i \wedge OK(d))] \rightarrow [\forall d\ (d \in D_i \rightarrow OK(d))]$. We will call such hypothesis a uniformity hypothesis. Notice that it is only concerned with the specification (the predicate OK) and the given implementation (the structure $\mathcal{D}$), but not with the selection criterion, it is only a postulate about the testing context independant of the criterion being used.

BUDD, LIPTON, SAYWARD & De MILLO, followed by HOWDEN have been developing a genuine approach to testing that is called Mutation Testing [Budd 81]. Its most important feature is, to our mind, consideration not only of the implementation to be tested, but instead a "neighbourhood of potential implementations". This formalizes the idea that we do not know perfectly the implementation we are testing, but only to some extent. We thus have to deal with a set of potential implementations, which the actual one is known to belong to, without being able to pick it out precisely. Any definition we state should only depend on this set rather than on the actual implementation. It must be uniform with respect to this set of objects.

This set itself is, in practice, only known through certain hypotheses about the implementation to be tested (among them are, for example, some uniformity hypotheses). Thus, the properties of a selection criterion must depend on those formal hypotheses, rather than on the actual implementation being run. We will refer to this principle as the Formal Approach to the notion of testing.

The advantages of this approach are twofold. Though we are only considering implementations as functions, the description of the set of potential implementations will take account of the syntactical features of the given program. It will in fact most likely contain the syntactical mutants of this program, as described by the above authors.

On the other hand, this makes the properties of a criterion independent of the syntactical evolutions of the program (design improvements), which is highly valuable.

1.3 Reliability, Validity and Bias

Let us now focus on the fundamental properties defined by GOODENOUGH & GERHART. Firstly, reliability means that all the tests selected by a reliable criterion are equivalent with respect to the correctness assessment of the implementation. Such a criterion should be viewed as "flat", just as a partial order is. It is obviously more interesting to consider some more complex criteria, where test powers (and costs) may vary. It suffices, in fact, to find one test for each level of power (or cost). We are thus led to consider, rather than a flat ordering, a total ordering, for example the N-ordering. Reliability expresses in this case that a test which is "higher" than another is better for correctness assessment. We will call this notion Projective Reliability (see §3.1).

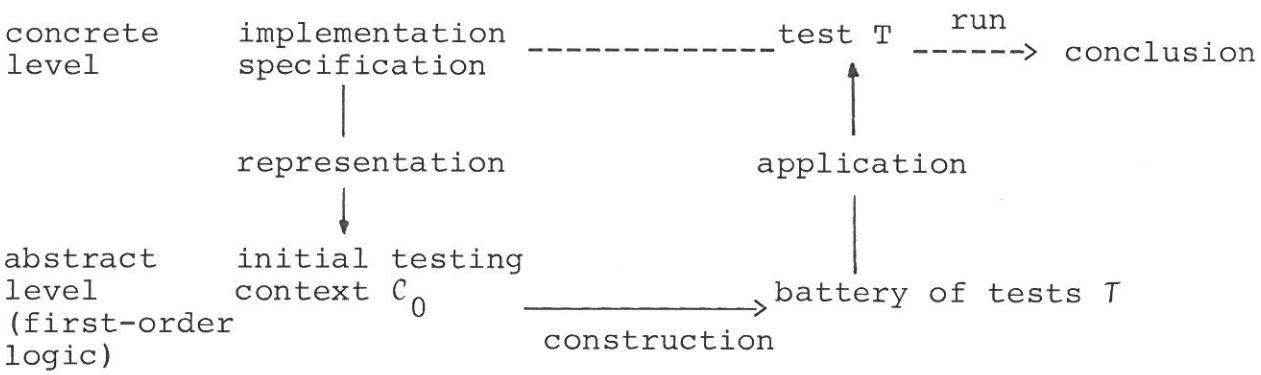
Validity of a criterion C expresses that, for each "error" in implementation behaviour, one can find in C an adequate test. But we are now dealing with a chain, and we can consider the "limit" of the tests of C, which is more powerful than any of them, and which can be "approximated" as closely as wanted if a sufficient cost is allowed. A quite sensible condition is thus to require this limit test to be adequate for each "error". This means that the above adequate test may be eventually rejected to the infinity. This is the main feature of what we call the Asymptotic Approach to the notion of testing. We will thus speak of Asymptotic Validity (see §3.2).

However, our feeling is that those two notions are not able to give a precise account of the intuitive notion of testing. Statistical test theory is based on the notion of Bias. A test is said to be unbiased if it is passed by an implementation more probably than failed if and only if this implementation is correct. This should be described as a coherence property, just as reliability is a consistency property. This property has been left

implicit by GOODENOUGH & GERHART since all the tests they consider are actually unbiased, because $\forall d \in D$ OK(d) implies $\forall t \in T$ OK(T). In other words, whenever the implementation is correct, it will pass any test. In our proposal, which deals with a more abstract testing notion, the notion of bias arises naturally as a highly valuable property. It is discussed in more detail later (see §3.3).

1.4 Testing Process Diagram

We are now in a position to draft the Testing Process Diagram. It should model the sequence of operations that take place from the definition of the property to be tested, up to the decision of whether the implementation is correct or not with respect to the above property (specification).



From the problem to be solved: "Is this implementation correct with respect to this specification?", we build, in the representation phase, an abstract problem, (intuitively) equivalent to the concrete one, expressed in the first-order egalitary logic language. This defines an initial testing context C_0 (see §2.2).

We now construct for this testing context an acceptable battery of tests (see §2.3 and §3.4), exactly as GOODENOUGH & GERHART were looking for an ideal criterion. However, this will not be possible in general, because at this point we do not have enough knowledge about the implementation to create any precise enough test.

We thus have at first to postulate some new hypothesis about the implementation (uniformity hypothesis, typically). We thus restrict the initial context C_0 to a new context C_1 (see §4.1), for which we can effectively construct an acceptable battery of tests T. Of course, this new hypothesis must be coherent with the property to be tested; the restriction must be conservative. Note that the construction phase is entirely abstract, and deals only with first-order logical objects.

Having now obtained an abstract battery of tests, we can pick some test from it, according to our quality/cost requirements, and apply it to the given problem, leading to the conclusion.

The Testing Process Diagram is the basic motivation for the precise definitions that we shall now study. It has been briefly sketched here to give the reader some insight into our general viewpoint. However it will be discussed extensively in section 5.

1.5 Conclusion

This section is intended to give some background for the theory of testing we are now going to develop. Our approach is mainly based on GOODENOUGH & GERHART's earlier proposal [Good 75]. Following them, we use first-order logic as an underlying mathematical tool, and we focus on the properties of a battery of tests (a selection criterion) rather than on those of a test.

We have emphasized several weaknesses and inadequacies in the work of GOODENOUGH & GERHART, many of which have already been pointed out in the literature. This has led us to define some precise guidelines for our modellling. By the Formal Approach, we mean that the construction of a battery of tests must be founded on some formal properties, deduced or postulated, about the functional behaviour of the given implementation. By the Asymptotic Approach, we mean that we must deal with a "chain-like" criterion rather than a "flat" criterion, tests being ordered by their positive quality (ability to be passed rather than to be failed). We can then study the property of the limit test of a criterion.

As we shall see later, these two approaches allow us to give an account of the main usefulness of testing with respect to proving: the "likely feature". Proving only leads us to conclude whether a program is correct or not. Testing allows us to conclude that an implementation is probably correct, where the probability is as high as possible for the cost of this assessment (see §5.3).

2. BASIC NOTIONS: TESTING CONTEXT AND BATTERY OF TESTS

We now focus on the precise and abstract mathematical definitions of the entities involved in the Testing Process Diagram above. These are mainly Testing Context and Battery of Tests. Their properties and the relationships between them will be extensively studied in the following sections.

2.1 Mathematical Tools

Most of the work in the area of program testing theory has been carried out in a logic-like mathematical framework [Good 75], [Whit 81], [Weyu 80b], [Budd 81] etc. This rather surprising unanimity leads us to choose also the first order egalitary logic formalism as a basis for our work. Yet we must take care of the results we get. They must be considered as direct consequences of this choice. Another one, statistical test formalism for example, would probably have led us to a quite different approach to the notion of program testing.

Having chosen a mathematical tool, we will try to use, as much as possible, its powerful features. That is we will have to keep in mind the "spirit" of our tool. We feel that one of the main features of logic is the duality between semantic and axiomatic approaches. Most of our statements will thus be split into a logical part and a formal part. The logical part tries to stick as closely as possible to the intuition we have in mind. The formal part expresses this intuition in a maybe more restrictive way, but allows handy and fruitful mathematical treatments. As a rule, the formal statement implies the logical one. This duality disciplin will lead us to somewhat cumbersome sentences, but it will be shown to be a very fair bridge between practice and theory, basic intuitions and mathematical requirements. We will refer to it as the Duality Principle.

Most of the notions that are used in our work are extensively described in the book "Mathematical Logic" of SHOENFIELD [Shoe 67]. We will only state here the main features we use, and introduce some notation.

Language, Structure

The languages we consider are first-order egalitary languages, denoted L. We identify such a language and its set of non-logical symbols. $L = \{P\}$ means that L is the (first-order egalitary) language whose only non-logical (predicate) symbol is P. If S denotes a set, L(S) is the language obtained from L by adding the elements of S as constant symbols. If L and L' are two languages, L' is called an extension of L if it contains all the (non-logical) symbols of L: $L \subseteq L'$. Then, L'(S) is an extension of L(S).

Given a set S and a language L, a structure $\mathcal{S}$ is obtained by associating with each functional symbol of L a function and with each predicate symbol a predicate. They are called interpretations or meanings of those symbols. $\mathcal{S}$ is said to be an L-structure with universe S. It can be canonically extended to an L(S)-structure by giving to the elements of S their obvious meanings. We ambiguously denote this structure $\mathcal{S}$. By addressing an L(S)-structure we always mean an L-structure extended in this way.

Formulas, Theories, Logical Validity

Given a language L, we can construct L-formulas. They look like $\exists x(P(x) \vee \forall y\ Q(y) \vee R(z))$. We are only interested in closed formulas, in which each variable x is in the scope of some $\exists x$ or $\forall x$. Now, given an L-structure $\mathcal{S}$, we can associate with each closed formula Φ a truth-value T or F. If it is T, Φ is said to be valid in $\mathcal{S}$. Then $\mathcal{S}$ is said to validate Φ. We write $\mathcal{S} \models \Phi$. Note that $\mathcal{S}$ validates Φ if and only if the canonically extended L(S)-structure does so.

A theory T on a language L is a set of L-formulas. They are often called axioms. They are divided into two parts. The logical axioms only depend on L, and are in any L-theory. They mainly express that the equality symbol "=" acts as a congruence. They are valid

in any L-structure. The other axioms are said to be non-logical. We identify a theory with its set of non-logical axioms. If T and T' are two L-theories, $T \sqcup T'$ is the L-theory whose non-logical axioms are those of T joined with those of T'.

An L-structure S validates an L-theory T if it validates all its axioms. It is obviously sufficient that it validates the non-logical ones. This is written $S \models T$. We say that S is a model for T. Obviously, $S \models T \sqcup T$ whenever $S \models T$ and $S \models T'$.

Formal Validity

Given an L-theory T and an L-formula Φ, we say that Φ is formally valid in T, or provable in T if Φ can be deduced from the axioms of T by the usual inference rules for first-order calculus. These are briefly Propositional Calculus rules and Quantifier Introduction rules: if x is not free in Ψ, infer $\exists x\ \Phi \rightarrow \Psi$ from $\Phi \rightarrow \Psi$, and dually with $\forall x$. If Φ is formally valid in T, we write $T \vdash \Phi$.

More generally, if T and T' are two L-theories, $T \vdash T'$ means that any axiom of T' can be proved in T. It is obviously sufficient that any non-logical axiom of T' can be proved in T. Provability relation " $\vdash$ " is reflexive and transitive. Obviously, $T \vdash T' \sqcup T''$ whenever $T \vdash T'$ and $T \vdash T''$.

If $T \vdash T'$ and $T' \vdash T$, T and T' are said to be formally equivalent. We then write $T \dashv\vdash T'$.

Let T be an L-theory, and Φ be an L-formula, such that $T \vdash \Phi$. The proof of Φ in T can only use a finite number of axioms. Thus, there exists a subtheory T_0 of T, with only a finite number of non-logical axioms (finite subtheory), such that $T_0 \vdash \Phi$. This result is known as the Compactness Theorem for First-Order Egalitary Logic.

Let T be an L-theory, S an L-structure and ϕ an L-for
that S is a model for T: $S \models T$ and that ϕ is provable
Then $S \models \phi$. This result is known as the Validity Theore
First Order Logic. It will ensure that our formal stateme
our logical ones, following our Duality Principle.

2.2 Testing Context

We are now going to make more precise the components of the Te
Process Diagram (cf §1.4). Let us first turn to the Testing Cont
notion. This notion should model the problem to be solved (i.e.
"is the implementation correct for the given specification?")
independently of the battery of tests we will build to solve it.

Remember that we are interested in the validation of large pro-
grams, for example compilers. They are considered by (most) users
as "black boxes". Users are only concerned with input/output beha-
viour, and generally not with their internal structure. An abstract
data type-like model thus arises naturally. An implementation will
be so modelled by its functional behaviour. In the first-order
logic formalism we use it will be thought of as a structure
(abstract data types hackers could however think of it as a
Σ-algebra).

We thus take a universe S, usually the domain of the program, and
a language L, containing usually at least the functional symbols
occurring in the program, plus a symbol representing the functional
behaviour. The model of our implementation will be an $L(S)$-structure
S. But, as we argued before, we must not only deal with one imple-
mentation, but with a family of potential implementations, among
hich belongs the one under test (cf. §1.3). We thus deal with a
nily (S) of $L(S)$-structures.

ation to be
er formulas.
the actual st

L

Let T be an L-theory, S an L-structure and Φ an L-formula. Suppose that S is a model for T: $S \models T$ and that Φ is provable in T: $T \vdash \Phi$. Then $S \models \Phi$. This result is known as the Validity Theorem for First Order Logic. It will ensure that our formal statements imply our logical ones, following our Duality Principle.

2.2 Testing Context

We are now going to make more precise the components of the Testing Process Diagram (cf §1.4). Let us first turn to the Testing Context notion. This notion should model the problem to be solved (i.e. "is the implementation correct for the given specification?") independently of the battery of tests we will build to solve it.

Remember that we are interested in the validation of large programs, for example compilers. They are considered by (most) users as "black boxes". Users are only concerned with input/output behaviour, and generally not with their internal structure. An abstract data type-like model thus arises naturally. An implementation will be so modelled by its functional behaviour. In the first-order logic formalism we use it will be thought of as a structure (abstract data types hackers could however think of it as a Σ-algebra).

We thus take a universe S, usually the domain of the program, and a language L, containing usually at least the functional symbols occurring in the program, plus a symbol representing the functional behaviour. The model of our implementation will be an L(S)-structure S. But, as we argued before, we must not only deal with one implementation, but with a family of potential implementations, among which belongs the one under test (cf. §1.3). We thus deal with a family (S) of L(S)-structures.

set of first-order formulas. The implementation is then correct if and only if the actual structure abstracting it (remember that

we do not know which one it is) validates those formulas. It is not restrictive to consider a (first-order) theory A whose non-logical axioms are those formulas.

Definition: Testing Context

A testing context $\mathcal{C}$ is a 4-uple $\langle L,S,(S),A\rangle$ where

* L is a first-order equalitary language
* S is a set
* (S) is a family of L(S)-structures
* A is an L(S)-theory.

Example

Consider, as an example, that we are testing a square-root program P, which should output the square-root of any natural number given as input. Then a sensible testing context $\mathcal{C} = \langle L,S,(S),A\rangle$ would be the following:

* $L = \{F;\ \_^2;\ \_ \leq \_ < \_\ ;\ \_ + \_\ \}$
  the symbol F represents the functional behaviour of the implementation;
* $S = \mathbb{N}$;
* (S) could be the family of the L(S)-structures such that the symbols of L different from "F" have their standard meanings, and such that F has a calculable meaning;
* $A = \{\forall x[F(x)^2 \leq x < (F(x)+1)^2]\}$

Notice that many first-order formulas are validated by all the structures of the family (S): for example $0+1=1$ or $\forall n[n\leq n+1<n+2]$. These formulas can be viewed as some formal hypothesis about the implementation being tested. Again, it is not restrictive to structure them in an L(S)-theory H.

This theory H captures the formal knowledge we a priori postulate about the implementation. We may, for example, postulate that the implementation is not "too" incorrect: take $H = \{\forall x[0\leq F(x)^2<(x+10)^2]\}$. Then we need only to consider those structures which actually validate H.

Definition: H-context

Let $\mathcal{C}$ = <L,S,(S),A> be a testing context, and let H be an L(S)-theory.

$\mathcal{C}$ is said to be an H-context if every structure of (S) validates H.

Notice that though the family (S) is a part of the testing context, H is not. Even if the actual conditions of the testing process do not change, the formal knowledge we make postulates about may vary (in most cases increase) during the testing process.

2.3 Battery of Tests

We can now define what a battery of tests for a given context is.

As we saw in the first section, a test for a given problem (a given testing context) is a finite set of experiments. An experiment is a question about implementation whose answer can be decided finitarily. We are thus led to the following definitions:

Definition: Experiment

Let $\mathcal{C}$ = <L,S,(S),A> be a testing context.

An experiment E for $\mathcal{C}$ is a (closed) L(S)-formula without quantifier, such that

for any non-logical symbol p of E

for any structure $\mathcal{S}$ of (S)

the meaning of p in $\mathcal{S}$ is calculable.

We do not want to make more precise what "calculable" means; in most cases, S = $\mathbb{N}$ and we will use the classical definition of a calculable function (predicate). We just note that a (theoretically) calculable, but extremely complex function should not, in practice, be considered to be a "humanly" calculable one. This is the so-called Oracle Problem [Weyu 80 a].

Definition: Test

Let $\mathcal{C} = \langle L, S, (S), A \rangle$ be a testing context.
A test T for $\mathcal{C}$ is an L(S)-theory with only a finite number of non-logical axioms (finite theory), each of them being an experiment of $\mathcal{C}$.

Note that the property that T is a test does not depend on A (the property to be tested), but only on L, S and (S) (the postulated features of the implementation).

The distinction between experiment and test is only a matter of convenience, as a test can always be identified with the conjunction of the experiments it is made up of.

Let us now turn to the definition of a battery of tests for a given context. It must cover the formal approach and the asymptotic approach we have outlined. A battery of tests should be a family of tests, ordered by a quality criterion that depends only on some formal hypothesis about the implementation. Two batteries of tests built on distinct formal hypotheses must, in our approach, be considered to be distinct.

Definition: Battery of Tests

Let $\mathcal{C} = \langle L, S, (S), A \rangle$ be a given testing context.
A battery of tests $\mathcal{T}$ for $\mathcal{C}$ is a pair $\langle H, (T_n)_{n \in \mathbb{N}} \rangle$ where

* H is an L(S)-theory such that $\mathcal{C}$ is an H-context (valid formal hypothesis)
* $(T_n)_{n \in \mathbb{N}}$ is a family of tests for $\mathcal{C}$ such that for all $n \in \mathbb{N}$ $T_{n+1} \sqcup H \vdash T_n$ (formal projective reliability, see §3.1)

The second condition is discussed more extensively in §3.1. We note here that, for any structure $\mathcal{S}$ of (S), and any $n \in \mathbb{N}$, if $\mathcal{S} \models T_{n+1}$, then $\mathcal{S} \models T_n$: the tests T_n are more and more precise, uniformly with respect to (S). These properties thus express the essence of the asymptotic approach.

We sometimes write (T_n) instead of $\langle H,(T_n)_{n\in\mathbb{N}}\rangle$ for the sake of conciseness.

2.4 Examples

In this section we give some basic examples that (hopefully) will help the reader to understand the different aspects of the above statements. We use freely the abbreviations and notation described above, as no confusion should arise.

We consider $\mathcal{C} = \langle L,S,(\mathcal{S}),A\rangle$ with $L = \{P\}$, $S = \mathbb{N}$, $A = \{\forall x\ P(x)\}$, and $(\mathcal{S})$ being the family of the $L(S)$-structures such that P has a calculable meaning.

$\mathcal{C}$ is a $\emptyset$-testing context.

(1) Take $\mathcal{T} = \langle H,(T_n)_{n\in\mathbb{N}}\rangle$, with $H = \emptyset$ and $T_n = \emptyset$ $\forall n\in\mathbb{N}$. $\mathcal{T}$ is actually a battery of tests for $\mathcal{C}$. It is called the empty battery of tests for $\mathcal{C}$.

(2) Take $\mathcal{T} = \langle H,(T_n)_{n\in\mathbb{N}}$ with $H = \emptyset$ and $T_n = \{P(0)\wedge\ldots\wedge P(n)\}$. $\mathcal{T}$ is also a battery of tests for C. It corresponds to the exhaustive testing strategy for A; if a structure $\mathcal{S}$ validates all the T_n, then $\mathcal{S}$ validates A. But notice that $H \bigsqcup_{n\in\mathbb{N}} T_n$ does not prove A! Keep on reading for more details.

(3) idem with $T_n = \{P(n)\wedge\ldots\wedge P(0)\}$. This <u>new</u> battery of tests should obviously be equivalent to the previous one for any sensible notion of equivalence (and fortunately it will, see §4.3).

(4) idem with $T_n = \{P(0)\wedge P(2)\wedge\ldots\wedge P(2n),P(1)\wedge P(3)\wedge\ldots\wedge P(i_n)\}$ with $i_n = E(\sqrt[3]{n}/10)$ (why not?). Same conclusion as above. One can easily generalize these examples with an enumeration (possibly not monic) of $\mathbb{N}$.

⑤ Consider $T = \langle H, (T_n)_{n\in\mathbb{N}} \rangle$ and $T' = \langle H', (T'_n)_{n\in\mathbb{N}} \rangle$, two batteries of tests for a testing context C. Then it is easy to prove that $T \sqcup T' = \langle H \sqcup H', (T_n \sqcup T'_n)_{n\in\mathbb{N}} \rangle$ is also a battery of tests for C. Note that the empty battery of tests is a neutral element for union.

⑥ Consider now another testing context $C = \langle L, S, (S), A \rangle$, with L, S and (S) as before, and $A = \{P(0), P(1), \ldots\} (= \{P(i),\ i\in\mathbb{N}\})$. Notice that for any $L(S)$-structure S,

$$S \models \{\forall x\ P(x)\} \text{ if and only if } S \models \{P(0), P(1), \ldots\},$$

but that the latter theory cannot prove the former. Look back at the example ②. It is still a battery of tests for this new context, but we now have the property $H \sqcup \underset{n\in\mathbb{N}}{} T_n \vdash A$. Yet, for any $k\in\mathbb{N}$, $H \sqcup \underset{n\leq k}{} T_n \not\vdash A$.

Let us now consider a more restricted (and realistic) family (S) of potential implementations. Consider for example, a $k\in\mathbb{N}$ being fixed, the theory H is

$$H = \{P(0)\wedge\ldots\wedge P(k) \rightarrow \forall x\ P(x)\}.$$

This hypothesis means that, for any potential implementation, if that implementation works correctly for any data of "complexity" less than k, then it works correctly for any data. This is a basic (but always hidden) hypothesis in path analysis testing methods where implicitly the "complexity" of data is the "complexity" of its path in the program.

Consider now the H-context $C = \langle L, S, (S), A \rangle$, with $L = \{P\}$, $S = \mathbb{N}$, $A = \{\forall x\ P(x)\}$ and (S) being the family of all those $L(S)$-structures S such that $S \models H$ and the meaning of P for S is calculable.

Then $T = \langle H, (T_n)_{n\in\mathbb{N}} \rangle$, with $T_n = \{P(0)\wedge\ldots\wedge P(n)\}$ is a battery of tests for C, with the property $H \sqcup \underset{n\in\mathbb{N}}{} T_n \vdash A$ (in fact, $H \sqcup T_k \vdash A$, and this remark is general, see §3.2).

This kind of hypothesis is called a regularity hypothesis. Another classical (and hidden) hypothesis in path analysis testing strategy is a uniformity hypothesis. Two data items which follow the same path in the program are equivalent with respect to correctness: if one is correctly handled by the implementation, the other will be as well.

We can give an account of this fact by postulating a hypothesis $H = \{\exists x\ P(x) \rightarrow \forall x\ P(x)\}$. These two kinds of hypotheses play an essential role in this work. In particular we can see that a regularity hypothesis is minimal in some sense in order to get a "good" battery of tests, and that using different constants leads to equivalent batteries of tests (virtual constants).

2.4 Conclusion

In this section we have stated the basic definitions for our work. We have tried to give an account of the principles outlined in the previous section, namely the Formal Approach and the Asymptotic Approach. A battery of tests is thus a family of tests, ordered by a certain quality criterion, together with a set of formal hypotheses about the implementation to be tested. Different hypotheses lead to different batteries of tests.

We should emphasize the perhaps hidden but fundamental choice which underlies this work: The definitions we deal with are induced by our intuitive understanding of the notion to be modelled; but they are at the same time strongly influenced by the mathematical tool we use, namely the first-order egalitary logic. This leads us to produce somewhat unelegant and cumbersome formulations and notations.

Our claim is that this effort,i.e. rigour and faithfulness to our underlying mathematical tool, is globally quite fruitful. We will try to assess it more precisely at the end of this work (see §6.1).

3. FUNDAMENTAL PROPERTIES OF BATTERIES OF TESTS

We have, up to now, got mathematical and precise definitions for the basic notions we are dealing with: a testing context and a battery of tests for a given testing context. Those definitions are strongly related to GOODENOUGH & GERHART's ones; however, they take care of the so-called Asymptotic Approach to the notion of testing, and of the fact that the implementation is only partially known, if not partially specified.

Following GOODENOUGH & GERHART's work, we now define by improving and adapting their propositions, the notion of Reliability and Validity for a battery of tests in a given testing context. Furthermore, we claim that by working at a perhaps too low level of abstraction, those authors left implicit the notion of Bias.

From this background, we finally define what is a "good" battery of tests for a given problem (a testing context). Such a battery of tests will be called acceptable.

As we noticed before, we are trying to stick as closely as possible to the mathematical tool we are using; mainly first-order logic. This formalism is largely concerned with the duality between formal statements (expressed with "$\vdash$"), and logical ones (expressed by "$\models$"). Henceforward, we must take account of this duality in our definitions. These will be split into two dual parts: logical and formal. In all cases, the formal statement will imply the logical one, the converse being generally false (because of the non-categoricity of the first-order logic, see [Shoe 67] §5.3).

3.1 Reliability

The reliability property expresses the internal consistency of the battery of tests which is being considered [Good 75],[Whit 81] From our viewpoint, consistency means that a test with an upper index is more powerful, with respect to success, than a test with a lower index.

In fact, for technical reasons, this property was already required when we defined a battery of tests in a given testing context (see §2.3). But because of its importance we feel that it should be stated here for itself.

Definition: logical and formal projective reliability.

Let $\mathcal{C} = \langle L, S, (S), A \rangle$ be an H-context, and $(T_n)_{n \in \mathbb{N}}$ a countable family of tests for this context.

* $(T_n)_{n \in \mathbb{N}}$ is said to be logically projectively reliable (log. proj. reliable) if

  for every structure S of (S)

  for every $n \in \mathbb{N}$

  if $S \models T_{n+1}$, then $S \models T_n$

* $(T_n)_{n \in \mathbb{N}}$ is said to be formally projectively reliable if for every $n \in \mathbb{N}$

  $T_{n+1} \sqcup H \vdash T_n$

Obviously, since $\mathcal{C}$ is an H-context, the second statement implies the first one.

As we stated before, a battery of tests for a given context (more precisely, its family of tests) is formally projectively reliable (and thus, logically too). We will only deal with such families of tests: in practice, it would be strange to consider a collection of tests (remember that a test is a set of experiments) such that an expensive test is not necessarily better than a cheaper one!

The work "projective" was chosen because of the analogy between the formula

$$T_{n+1} \sqcup H \vdash T_n$$

and the definition of a projective sequence of sets, for example.

We will sometimes abbreviate logical and formal projective reliability by reliability, when the addressed property is evident.

3.2 Validity

We now turn to validity. Reliability was concerned with the consistency of a family of tests. Validity is concerned with its power and its usefulness. GOODENOUGH & GERHART gave a strong definition of this property: a collection (criterion) is valid if, when the program is incorrect, it fails at least some test (see §1.1). Our formalism allows us to give a slightly more general statement: the failed test may be incidentally rejected to infinity, i.e. may be (virtually) T_∞.

Definition: logical and formal asymptotic validity.

Let $C = \langle L, S, (S), A \rangle$ be a testing context, and $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ a battery of tests in this context.

* $\mathcal{T}$ is said to be logically asymptotic valid (log. as. valid) if

  for every structure S of (S),
  if $S \models T_n$ for every $n \in \mathbb{N}$,
  then $S \models A$

* $\mathcal{T}$ is said to be formally asymptotically valid (form. as. valid) if

$$\bigsqcup_{n \in \mathbb{N}} T_n \sqcup H \vdash A$$

Again, the second statement implies the first one, because every S of (S) is such that $S \models H$. Notice that by (formal projective) reliability, the first statement is equivalent to the following one:

> for every $S \in (S)$, there exists $n_S \in \mathbb{N}$ such that if $S \models T_n \quad \forall n \geq n_S$, then $S \models A$.

In the same way, the second one can be written

$$\exists n_0 \in \mathbb{N} \qquad \bigsqcup_{n \geq n_0} T_n \sqcup H \vdash A$$

These facts justify the informal description stated above.

As before, we will sometimes say validity for logical or formal asymptotic validity when the addressed property is evident.

Examples

Let us look at some examples.

① Let C be a context, and $T = \langle H, (T_n)_{n\in\mathbb{N}} \rangle$, with $H=\emptyset$, $T_n=\emptyset$ for all $n\in\mathbb{N}$.
T is formally valid (for C) iff $A=\emptyset$.

② Let $C = \langle L,S,(S),A \rangle$ be a context and $T = \langle A, (T_n)_{n\in\mathbb{N}} \rangle$ a battery of tests for C. This implies that every structure S of (S) is a model of A, i.e. the property being tested is verified by all the potential implementations being considered.
T is formally valid for C.

③ Let $C = \langle L,S,(S),A \rangle$ be a context, with $L = \{P\}$, $S = \mathbb{N}$, $A = \{\forall x\ P(x)\}$, and (S) being the family of all the $L(S)$-structures such that P has a calculable meaning.
Take $T_n = \{P(0)\wedge\ldots\wedge P(n)\}$. Then $T = \langle \emptyset, (T_n)_{n\in\mathbb{N}} \rangle$ is a logically asymptotically valid battery of tests for C, but <u>not</u> a formally valid one, because $\{P(n),\ n\in\mathbb{N}\} \nvdash \forall x\ P(x)$.

④ Take C as in ③, but restrict (S) to those structures such that $S \models H$, with $H = \{P(0)\wedge\ldots\wedge P(k) \rightarrow \forall x\ P(x)\}$.
Then T is formally valid. Notice that this does not depend on the value of k. k is called a virtual constant. In the example ③, we had virtually $k=\infty$. H is called a <u>regularity hypothesis</u>.

⑤ As ④, but now with $H = \{\exists x\ P(x) \rightarrow \forall x\ P(x)\}$ the same conclusions hold. Following WEYUKER & OSTRAND [Weyu 80b], H is called a <u>uniformity hypothesis</u>.

Notice that in ④, T is in fact valid in the "classical" sense, since $H \sqcup T_k \vdash A$. This fact is general indeed. More precisely, we define the finite validity, that matches the classical validity.

Definition: (formal) finite validity.

Let $\mathcal{C} = \langle L, S, (S), A\rangle$ be a testing context, and $\mathcal{T} = \langle H, (T_n)_{n\in\mathbb{N}}\rangle$ a battery of tests for $\mathcal{C}$.

$\mathcal{T}$ is said to be formally finitely valid if

$$\exists n_0 \in \mathbb{N} \quad T_{n0} \sqcup H \vdash A$$

Note that if $\mathcal{T}$ is finitely valid, then $\forall n \geq n_0 \quad T_n \sqcup H \vdash A$, so $\mathcal{T}$ is formally asymptotically valid.

Remember that a finite theory is a theory having only a finite set of non-logical axioms. A straightforward consequence of the compactness theorem of the first order egalitary logic (see [Shoe 67] §5.1) is the

Compactness theorem

Let $\mathcal{C} = \langle L, S, (S), A\rangle$ be a testing context, with A a finite theory, and $\mathcal{T}$ a battery of tests for $\mathcal{C}$. $\mathcal{T}$ is formally asymptotically valid iff it is formally finitely valid.

Proof

If $\mathcal{T}$ is asymptotically valid, $H \bigsqcup_n T_n \vdash A$. But each non-logical axiom of A can be proved by using only finitely many axioms of $H \bigsqcup_n T_n$. Because of the finiteness of A, we can find $i_1, \ldots, i_p$ such that $H \sqcup T_{i_1} \sqcup \ldots \sqcup T_{i_p} \vdash A$.

Take $n_0 = \sup(i_1, \ldots, i_p)$. By reliability,

$$H \sqcup T_{n_0} \vdash H \sqcup T_{i_k} \quad \text{for } k = 1, \ldots, p.$$

So $H \sqcup T_{n_0} \vdash A$. ∎

Thus, if A is finite, our definition is nothing more than a restatement of the previous one. If A is not finite, this is not true. Take $H = \emptyset$, $A = \{P(n),\ n \in \mathbb{N}\}$; $T_n = \{P(0) \wedge \ldots \wedge P(n)\}$; is formally asymptotically valid, but not finitely valid.

Example (3) shows that no logical dual of this theorem may be expected. In this case, and in many others, formal properties are more fruitful in their theoretical consequences than the logical ones. They allow us to understand better the relationships between the many objects we are dealing with. On the other hand, they are sometimes too strong with respect to our intuitive perception of the testing process, which leads us to use logical definitions. Our approach, taking account of duality, is a possible mathematical answer to bridge the gap between theory and practice (see §6.1).

3.3 Bias

The bias concept is fundamental in statistics. A (statistical) test is said to be unbiased if it leads to the right conclusion with a higher probability than to the wrong one. Because we do not have any probability concept in our formalism, we will simply state that a battery of tests is unbiased if every correct implementation passes all the tests. So, if the tested implementation fails a test, we can conclude that it is certainly incorrect with respect to the considered specification. This is therefore a basic requirement for a battery of tests to be unbiased.

Definition: logical and formal lack of bias.

Let $\mathcal{C} = \langle L, S, (S), A \rangle$ be a testing context, and let $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ be a battery of tests in this context.

* $\mathcal{T}$ is said to be logically unbiased if for every structure S of (S) if $S \models A$, then $S \models T_n$ for all $n \in \mathbb{N}$
* $\mathcal{T}$ is said to be formally unbiased if

$$H \sqcup A \vdash T_n \quad \forall n \in \mathbb{N}$$

Once again, the second statement implies the first one. Both are the converses of the corresponding asymptotic validity definitions (cf. §3.2).

As far as we know, no bias-like property has yet been stated about program testing. In fact, it was implicit in most of the previous work on this subject.

Let us consider for example GOODENOUGH & GERHART's theory (see §1.1). A test T is a (finite) subset of the input domain D of P. The implicit theory associated with it is

$$T = \{\forall t \in T \ OK(t)\}$$

The implicit theory A to be tested is

$$A = \{\forall d \in D \ OK(d)\}$$

i.e. the program is correct for every input value. But, of course, $A \vdash T$, so the criterion C = (T) is formally unbiased.

Our formalism is more general than their one, and allows us to point out this property, left implicit so far.

Examples

Let us now look at some examples.

① Let $\mathcal{C}$ be a context, and $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$, where $H=\emptyset$, $T_n=\emptyset$ for all $n \in \mathbb{N}$. $\mathcal{T}$ is formally unbiased.

② Take now $\mathcal{T} = \langle A, (T_n)_{n \in \mathbb{N}} \rangle$; it is unbiased too.

③ Take $S=\mathbb{N}$, $H=\emptyset$, $A = \{\forall x(\neg(x=0) \rightarrow P(x))\}$; let $(\mathcal{S})$ be the family of the L(S)-structures such that P has a calculable meaning.
Then $T_n = \{P(0) \wedge \ldots \wedge P(n)\}$ is biased (not even logically unbiased). But $T'_n = \{P(1) \wedge \ldots \wedge P(n)\}$ for $n \geq 1$ with $T_0=\emptyset$ is unbiased.

④ non-calculable properties
Consider now the case where a property about a non-calculable symbol is being tested. The natural way to test it is to look for a stronger calculable property.
Take $S=\mathbb{N}$; $H = \{\forall x(Q(x) \rightarrow P(x))\}$, $A = \{P(0)\}$, with $(\mathcal{S})$ the family of the L(S)-structures such that Q (and not necessarily P) has a calculable meaning. The natural battery of tests

is then $T_n = \{Q(0)\}$. It is valid, but biased, unless we have $Q(0) \leftrightarrow P(0)$ for all the considered structures.

⑤ existential properties

Take $S=\mathbb{N}$, $L = \{P\}$, $A = \{\exists x\ P(x)\}$, and let (S) be the family of the L(S)-structures such that P has a calculable meaning. Take $H=\emptyset$. It can be shown that the only $T = \langle H, (T_n)_{n\in\mathbb{N}} \rangle$ unbiased in the context $C = \langle L,S,(S),A \rangle$ is such that $T_n=\emptyset$ for all $n\in\mathbb{N}$.

If we can now restrict (S) to those structures that validate $H = \{\exists x\ P(x) \rightarrow P(0) \vee \ldots \vee P(k)\}$, with k a (virtual) constant, then $T_n = \{P(0) \vee \ldots \vee P(k)\}$ is unbiased (and valid), but depends on k.

3.4 Acceptability

We can now define what a "good" battery of tests for a given problem is. We do not want to reject a correct program, so we need the lack of bias property. We obviously cannot demand conversely, that a program which passes some test is necessarily correct. Yet we can require that any incorrect program fails some test, namely asymptotic validity.

Definition: logical and formal acceptability.

Let C be a testing context, and T a battery of tests in this context.

* T is said to be logically acceptable if it is logically asymptotically valid and logically unbiased.
* T is said to be formally acceptable if it is formally asymptotically valid and formally unbiased.

From the previous remarks, it is obvious that formal acceptability implies the logical acceptability. With the now-usual notation, logical acceptability means for every structure S of (S)

$$S \vdash T_n \text{ for all } n\in\mathbb{N} \text{ iff } S \vdash A.$$

Formal acceptability means simply

$$H \sqcup \bigsqcup_{n \in \mathbb{N}} T_n \vdash H \sqcup A,$$

i.e. with respect to H (the formal hypothesis about the program), A (the property being tested) is equivalent to $\bigsqcup_{n \in \mathbb{N}} T_n$ (which can be viewed as T_∞).

We define a "good" battery of tests for a given context as a formally acceptable one. We will sometimes just say acceptable, when no confusion is threatened.

As we pointed out before, a natural choice might rather have been logical acceptability. But in this case, the acceptability of a given battery of tests would have been very strongly dependent on the underlying testing context (more precisely, on (S)). The context can be viewed as the way of using the battery of tests, and, in practice, it is not a well-defined entity. Specifically, it is very difficult to define precisely the family (S) of all the potential implementations of the program (including incorrect implementations).

Therefore, we choose a more intrinsic notion: namely formal acceptability. This property can be states as an entirely abstract relation between abstract entities A, H, and $(T_n)_{n \in \mathbb{N}}$. It therefore depends on the way of using the battery of tests only through H, that is the formal hypotheses about the implementation being considered. The more precise they are, the closer is this notion to logical acceptability.

3.5 Examples

(1) Let $\mathcal{C} = \langle L, S, (S), A \rangle$ be an A-context. Then $\mathcal{T} = \langle A, (T_n)_{n \in \mathbb{N}} \rangle$ with $T_n = \emptyset$, is (formally) acceptable. The problem is therefore to describe an acceptable battery of tests with a hypothesis H as weak as possible.

(2) Take L = {P}, S=$\mathbb{N}$, H=$\emptyset$, A = {P(n), n$\in\mathbb{N}$}. Let (S) be a family of L(S)-structures such that P has a calculable meaning in each of them; then T_n = {P(0)$\wedge$...$\wedge$P(n)} is acceptable.

(3) The same with A = {$\forall$x P(x)}. Notice that, with respect to (S), {P(n), n$\in\mathbb{N}$} and {$\forall$x P(x)} are logically equivalent. But now, $(T_n)_{n\in\mathbb{N}}$ is <u>not</u> acceptable.

(4) Let us restrict (S) to those structures which validate the <u>regularity hypothesis</u> H = {P(0)$\wedge$...$\wedge$P(k) $\rightarrow$ $\forall$x P(x)}, k being a virtual constant. Then $(T_n)_{n\in\mathbb{N}}$ is acceptable: it is nothing more than our old friend the exhaustive testing strategy.

(5) Instead of a regularity hypothesis, let us consider a <u>uniformity hypothesis</u> H = {$\exists$x P(x) $\rightarrow$ $\forall$x P(x)}. Add to the language, if there is not one already, a special constant symbol a (see §4.1). Then T_n = {P(a)} is acceptable. This is our dear friend the random stampling testing strategy.

(6) <u>existential properties</u>

In §3.3, we examined the case of not purely universal properties, for example $\exists$x P(x). We can construct an acceptable battery of tests by assuming the hypothesis H = {$\exists$x P(x) $\rightarrow$ P(0)$\vee$...$\vee$P(k)}. Note that this can be written $\neg$P(0)$\wedge$...$\wedge\neg$P(k) $\rightarrow$ $\forall$x $\neg$P(x), i.e. as a regularity hypothesis for $\neg$P.

Note also that the natural family of tests is then T_n = {P(0)$\vee$...$\vee$P(k)}, <u>which depends on k</u>. In fact, it is <u>not</u> possible to build an acceptable family $(T_n)_{n\in\mathbb{N}}$ that does not depend on k.

This is why we state informally that existential (more precisely not purely universal) are <u>not testable</u>. Another good reason will be stated in §4.1.

actual implementation being tested; this is a very valuable property. Furthermore, it gives us more confidence when extrapolating, in the conclusion of the testing process, the success of one test say T_p, to the success of all the tests T_n, $n \in \mathbb{N}$ (see §5.3).

⑦ non-calculable properties

We already noticed that non-calculable properties "naturally" lead to a biased battery of tests.

⑧ mutation testing strategy

This strategy (including testing-quality measure) has been described by LIPTON et al. (see [Budd 81] for details and references). It can easily be expressed in our framework.

Let $C = \langle L,S,(S),A\rangle$ be a testing context, and $T = \langle H,(T_n)_{n\in\mathbb{N}}\rangle$ a battery of tests for C.
Suppose there exists at least one S_0 in (S) such that $S_0 \models A$.
Suppose that $(T_n)_{n\in\mathbb{N}}$ discriminates between the structures of (S): for every pair of distinct structures S,S' of (S), there exists $n\in\mathbb{N}$ such that $\neg[(S \models T_n) \wedge (S' \models T_n)]$.
Suppose that T is (logically) unbiased (this hypothesis is in most cases left implicit).
Then T is logically acceptable for C. It suffices to show that it is logically asymptotically valid. Note that, by lack of bias, $S_0 \models T_n \ \forall n\in\mathbb{N}$. Consider now any structure $S \in (S)$ such that $S \models T_n \ \forall n\in\mathbb{N}$. Because $(T_n)_n$ discriminates between the structures of (S), we can conclude that S actually is S_0, so that $S \models A$.

3.6 Conclusion

This section was devoted to the fundamental properties of batteries of tests for a context. Following GOODENOUGH & GERHART, and taking account of the asymptotic approach which we outlined previously, we defined the properties of projective reliability asymptotic validity. We pointed out the (lack of) bias properties, left implicit by those authors.

We then defined a "good" battery of tests to be a formally valid and unbiased one. By this choice, we got a quite intrinsic notion, which depends on the actual utilisation conditions only through the formal hypothesis H. This means that the acceptability property is very stable with respect to the evolution of the

4. ORDERING CONTEXTS AND BATTERIES

Up to now, we have only been restating the main ideas of GOODENOUGH & GERHART in a more general framework, translating informal descriptions into precise and rigorous mathematical definitions, and sugaring the whole thing with asymptotic and formal approaches and duality.

We now turn to some new ideas. In the previous examples (cf. §3.5) we sometimes found it necessary to restrict the family of structures (S) in order to construct an acceptable battery of tests. That meant that, in some way, we had obtained more knowledge about the program being tested. The new context was then more precise than the old one. This remark leads us to a preorder on testing contexts.

On the other hand, once a context has been given, we would like to be able to compare its batteries of tests. Thus, we want to be able to describe precisely what it means to improve, to optimize, a testing strategy. We want to be able to describe, as well, what two equivalent testing strategies are. This leads us straightforward to a preorder on batteries of tests for a given testing context. The equivalence induced turns out to be of great interest.

4.1 Conservative Restriction of a Testing Context

In most cases, there does not exist any acceptable battery of tests for a given context $\mathcal{C} = \langle L,S,(S),A\rangle$. One reason could be that L does not allow us to express precise enough properties, so we would want to be able to extend L. But this reason is obviously rather technical.

Another more crucial reason could be that (S) is too large, too heterogeneous. We would like to restrict it to a more homogeneous family, but without eliminating any correct potential implementations of the program being tested. The restriction must be conservative.

Let L be a language, and L' an extension of L. Let $\mathcal{S}'$ be an L'-structure. We denote by $\mathcal{S}'_{|L}$ the L-structure obtained by removing the symbols of L'-L. Let A be an L-theory. We denote by $A_{|L'}$ the L'-theory having the same non-logical axioms as A (but considered as L'-formulas). We can now state the definition.

Definition: conservative restriction

Let $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ and $\mathcal{C}' = \langle L', S', (\mathcal{S}'), A' \rangle$ be two testing contexts. $\mathcal{C}'$ is said to be a conservative restriction of $\mathcal{C}$ if

1) L' is an extension of L
2) $S' = S$
3) $(\mathcal{S}')_{|L(S)} \subseteq (\mathcal{S})$
4) $A_{|L'(S)} = A'$
5) conservative condition

   for every structure $\mathcal{S}$ of $(\mathcal{S})$ such that $\mathcal{S} \models A$, there exists a structure $\mathcal{S}'$ of $(\mathcal{S}')$ such that

$$\mathcal{S}'_{|L(S)} = \mathcal{S}$$

Note that if L' is an extension of L, then L'(S) is an extension of L(S). Note, too, that for any L'(S)-structure $\mathcal{S}'$, $\mathcal{S}' \models A_{|L'(S)}$ iff $\mathcal{S}'_{|L(S)} \models A$, because we are dealing with the same universe S in both cases.

It can easily be shown that the relation $\subseteq$ is a preorder on the class (but not the set!) of testing contexts. Therefore, by transitivity, we may deal only with elementary conservative restrictions, putting them together to get the desired result.

The following theorem shows that a (conservative) restriction cannot, in any case, discard any acceptable battery of tests. It is therefore a safe tool.

(Conservative) Restriction Theorem

Let $\mathcal{C} = \langle L, S, (\mathcal{S}), A \rangle$ be a testing context, and let $\mathcal{C}' = \langle L', S', (\mathcal{S}'), A' \rangle$ be a conservative restriction of $\mathcal{C}$.
Let $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ be a battery of tests for $\mathcal{C}$.
Let $\mathcal{T}' = \langle H_{|L'(S)}, (T_{n|L'(S)})_{n \in \mathbb{N}} \rangle$.
If $\mathcal{T}$ is logically acceptable for $\mathcal{C}$, so is $\mathcal{T}'$ for $\mathcal{C}'$.
If $\mathcal{T}$ is formally acceptable for $\mathcal{C}$, so is $\mathcal{T}'$ for $\mathcal{C}'$.

Lemma

Let U and V be two L-theories and L' be an extension of L.
Then $U \vdash V$ iff $U_{|L'} \vdash V_{|L'}$.

Proof

By the completeness theorem of the first-order egalitary logic (see [Schoe 67] §4.2.)
Suppose $U \vdash V$. Let S' be any L'-structure that validates $U_{|L'}$. Then $S'_{|L} \models U$, $S'_{|L} \models V$ and $S' \models V_{|L'}$. Thus $U_{|L'} \vdash V_{|L'}$. The converse is proved in the same way.

□

Proof of the theorem

* Let us show that $\mathcal{T}'$ is actually a battery of tests for $\mathcal{C}'$.
Let $S' \in (\mathcal{S}')$; then $S'_{|L(S)} \in (\mathcal{S})$, $S'_{|L(S)} \models H$ and $S' \models H_{|L'(S)}$. Because $(\mathcal{S}')_{|L(S)} \subseteq (\mathcal{S})$, the calculability of the meanings of the non-logical symbols of T_n is preserved. $T_{n|L'(S)}$ is thus actually a test for $\mathcal{C}'$. By applying the lemma with L(S) and L'(S), reliability is obvious.

* Suppose $\mathcal{T}$ is logically acceptable for $\mathcal{C}$. For any $S' \in (\mathcal{S}')$ $S' \models A_{|L'(S)}$ iff $S'_{|L(S)} \models A$ iff $S'_{|L(S)} \models T_n$ for any $n \in N$ (because $S'_{|L(S)} \in (\mathcal{S})$) iff $S' \models T_{n|L'(S)}$ for any $n \in N$.
Thus $\mathcal{T}'$ is logically acceptable for $\mathcal{C}'$.

* Suppose $\mathcal{T}$ is formally acceptable for $\mathcal{C}$. By the lemma, the relation $H \bigsqcup_n T \vdash H \sqcup A$ implies

$$H_{|L'(S)} \bigsqcup_n T_{n|L'(S)} \vdash H_{|L'(S)} \sqcup A_{|L'(S)},$$

which expresses formal acceptability of $\mathcal{T}'$ for $\mathcal{C}'$.

□

Note that the converse of the above theorem is true in the formal case: a battery of tests that was not formally acceptable cannot become acceptable as if by (conservative) magic! This means that formal acceptability is a rather intrinsic property which depends little on the concrete application conditions. Dependence only occurs through the formal hypothesis H , which is, as we saw earlier, a valuable feature.

Yet, theorem converse is not true in the logical case: Logical acceptability may depend on external conditions. This feature agrees with practical intuition, but leads to conceptual difficulties.

Note that the conservative condition is not used in the above proof. It is yet needed by the intuitive meaning of our definition, and will be of great interest later (see §5.3).

Conservative restrictions of testing contexts will be used in two main ways in practice. On the one hand, we will add some new symbols to L, extending the family of structures by giving to those new symbols some specific meanings. For example, in case (5) of §3.5 the new constant symbol "a" models an arbitrary element of the domain. To choose a structure S' of the restricted context means to pick up randomly a value in S. We thus give a precise model of random stampling strategy.

On the other hand, we will use some new hypothesis H_1 about the potential implementation, for example the regularity or uniformity hypothesis. We then restrict (S) to those structures that validate H_1 (see example (5) of §3.3). It can be easily shown that a sufficient condition for the restriction to be conservative is $A \sqcup H \vdash H_1$. The restricted context is then an $H \sqcup H_1$-context.

As a special case, we can take $H_1 = A$. We then get an A-context. A trivial acceptable battery of tests is $T = \langle A, (T_n)_{n \in \mathbb{N}} \rangle$ with $T_n = \emptyset$ for all $n \in \mathbb{N}$. If we consider the property to be tested as a hypothesis, testing becomes trivial!

This is why we must only consider some likely hypothesis H_1. A must obviously not be considered as a likely hypothesis. The regularity or uniformity hypothesis may, in most cases. This "likely" feature will get a crucial place for quality assessment (see §5.3).

For purely universal properties, the regularity or uniformity hypothesis leads actually to conservative restrictions. For the regularity hypothesis,

$$[P(0) \wedge \ldots \wedge P(k) \rightarrow \forall x\ P(x)] \rightarrow [\forall x\ P(x)]$$

holds for any k. But this is not true in general. For example, in example (5) of §3.3, consider the structure S such that $P_S(x) = (x = k+1)$ then $S \models A = \{\exists x\ P(x)\}$, but S does not validate $H_1 = \{\exists x\ P(x) \rightarrow P(0) \vee \ldots \vee P(k)\}$. The restriction is not conservative, we implicitly eliminate some correct potential implementations of our program. That is why properties which are not purely universal are not testable in general.

4.2 Asymptotic Sharpness

We now consider a given testing context, and look at its set of batteries of tests. Informally, we wish to be able to compare two of them with respect to the quality of information we can gain. But we consider only positive information, i.e. information about correctness of the implementation, and not information about its possible incorrectness. As with conservative restriction, we are interested rather in correct implementation than in incorrect ones.

A battery of tests is said to be sharper than another if the success of its tests implies the success of the tests of the other, irrespective of indexes.

This is the intuitive, logical definition. For the formal one, we have to deal with formal hypotheses H and H'. It is, of course, a highly valuable property for a battery of tests to involve weak hypotheses, because we do not know if they are actually valid for the concrete implementation. Our definition must reflect this viewpoint.

Definition logical and formal asymptotic sharpness

Let $C = \langle L,S,(S),A\rangle$ be a testing context.

Let $\mathcal{T} = \langle H,(T_n)_{n\in\mathbb{N}}\rangle$ and $\mathcal{T}' = \langle H',(T'_n)_{n\in\mathbb{N}}\rangle$ be two batteries of tests for C.

* $\mathcal{T}$ is said to be logically asymptotically sharper than $\mathcal{T}'$ if

for every $i\in\mathbb{N}$
there exists $j\in\mathbb{N}$, such that
for every structure S of (S)
if $S \models T_j$, then $S \models T'_i$

* $\mathcal{T}$ is said to be formally asymptotically sharper than $\mathcal{T}'$ if

1) $\forall i\in\mathbb{N}\ \exists j\in\mathbb{N}\quad H \sqcup T_j \vdash H' \sqcup T'_i$

2) $H' \sqcup A \vdash H \sqcup A$

As with previous definitions formal sharpness implies logical sharpness. Both of them define partial orders on the set of batteries of tests for the context C. However, the logical order is more "complete" than the formal one. For example, two batteries of tests have a logical least upper bound, but not necessarily a formal one.

We will sometimes write sharpness instead of logical or formal asymptotic sharpness, when no confusion will result. The sharpness orderings will be denoted $\leqq$. The equivalence relations canonically associated with them will be denoted $\sim$.

Examples

① Let $C = \langle L,S,(S),A\rangle$ be a context, and $\mathcal{T} = \langle H,(T_n)_{n\in\mathbb{N}}\rangle$ be a battery of tests for C such that $A \vdash H$ (this is in practice often true). Then $\mathcal{T}' = \langle \emptyset,(\emptyset)_{n\in\mathbb{N}}\rangle$ is a battery of tests for C, and $\mathcal{T}' \leqq \mathcal{T}$ strictly.

② Let $C = \langle L,S,(S),A\rangle$ be a context, and let $\mathcal{T} = \langle H,(T_n)_{n\in\mathbb{N}}\rangle$ and $\mathcal{T}' = \langle H',(T'_n)_{n\in\mathbb{N}}\rangle$ be two batteries of tests for C. Then $\mathcal{T} \sqcup \mathcal{T}' = \langle H \sqcup H',(T_n \sqcup T'_n)_{n\in\mathbb{N}}\rangle$ is a battery of tests for C, and it is the least upper bound of $\mathcal{T}$ and $\mathcal{T}'$ for logical sharpness.

If $A \sqcup H \vdash A \sqcup H'$, then this is so for formal sharpness as well.

③ Let C be a context, and $T = \langle H, (T_n)_{n\in\mathbb{N}} \rangle$ be a battery of tests for C. Let $T' = \langle H, (T'_n)_{n\in\mathbb{N}} \rangle$ and $T'' = \langle H, (T''_n)_{n\in\mathbb{N}} \rangle$ with $T'_n = T_{2n}$ and $T''_n = T_{E[\frac{n}{2}]}$. Then T' and T'' are two batteries of tests for C, formally equivalent to T.

④ Take $L = \{P,Q\}, S = \mathbb{N}, H = H' = \{\forall x\ (Q(x) \rightarrow P(x))\}$. $A = \{\forall x\ P(x)\}$. Consider $T = \langle H, (T_n)_{n\in\mathbb{N}} \rangle$ and $T' = \langle H', (T'_n)_{n\in\mathbb{N}} \rangle$ where

$$T_n = \{P(0)\wedge\ldots\wedge P(n)\}$$
$$T'_n = \{Q(0)\wedge\ldots\wedge Q(n)\}$$

Then $T' \geqq T$ strictly.

⑤ Take $L = \{P\}$, $S = \mathbb{N}$, $A = \{P(n),\ n\in\mathbb{N}\}$, $T_n = \{P(0)\wedge\ldots\wedge P(n)\}$

$$H = \{P(n),\ n\leq k,\ n \text{ odd}\}$$
$$H' = \{P(n),\ n\leq k,\ n \text{ even}\}$$

Consider $T = \langle H, (T_n)_{n\in\mathbb{N}} \rangle$ and $T' = \langle H', (T'_n)_{n\in\mathbb{N}} \rangle$. Then $T \sim T'$, but $H \not\vdash H'$, and $H' \not\vdash H$.

Properties of sharpness equivalence will be studied more precisely in the following paragraph (see §4.3). The following theorem shows that sharpness ordering is relevant with respect to validity and bias. Informally, validity is an increasing property, and lack of bias a decreasing one.

Monotony theorem

Let C be a testing context, and let T and T' be two batteries of tests for C.

Suppose T is logically (resp. formally) asymptotically sharper than T'.

If T' is logically (resp. formally) asymptotically valid, so is T.

If T is logically (resp. formally) unbiased, so is T'.

Proof

We consider only the formal case (the logical one can be proved in a very similar way).

Take $C = \langle L,S,(S),A\rangle$, $T = \langle H,(T_n)_{n\in\mathbb{N}}\rangle$, $T' = \langle H',(T'_n)_{n\in\mathbb{N}}\rangle$.

* Assume now that T' is valid, and examine $H \bigsqcup_{n\in\mathbb{N}} T_n$;

$H \bigsqcup_{n\in\mathbb{N}} T_n \vdash H' \bigsqcup T'_i$ and this is true for all $i\in\mathbb{N}$;

thus $H \bigsqcup_{n\in\mathbb{N}} T_n \vdash H' \bigsqcup_{n\in\mathbb{N}} T'_n$, and $H' \bigsqcup_{n\in\mathbb{N}} T'_n \vdash A$ by hypothesis.

Thus $H \bigsqcup_{n\in\mathbb{N}} T_n \vdash A$.

* Assume now T is unbiased and examine $H' \bigsqcup A$. Given any $n\in\mathbb{N}$; $H' \bigsqcup A \vdash H \bigsqcup A$, and $H \bigsqcup A \vdash T_n$ by hypothesis. So $H' \bigsqcup A \vdash T_n$; but $H' \bigsqcup A \vdash H$ obviously, and thus $H' \bigsqcup A \vdash H \bigsqcup T_n$, and this is true for all $n\in\mathbb{N}$. Thus $H' \bigsqcup A \vdash H \bigsqcup_{n\in\mathbb{N}} T_n$, and, as we have shown previously, $H \bigsqcup_{n\in\mathbb{N}} T_n \vdash H' \bigsqcup_{n\in\mathbb{N}} T'_n$. Thus $H' \bigsqcup A \vdash H' \bigsqcup_{n\in\mathbb{N}} T'_n$, and, as a consequence $H' \bigsqcup A \vdash T'_n$ for all $n\in\mathbb{N}$. □

(Counter-) examples

(1) Consider example (4) above; $T' \geq T$, T is unbiased but T' is biased.

(2) Take $L = \{P\}$, $S = N$, $A = \{\forall x\ P(x)\}$, $H = \emptyset$, $H' = \{P(0)\wedge\ldots\wedge P(k) \rightarrow \forall x\ P(x)\}$ and $T_n = \{P(0)\wedge\ldots\wedge P(n)\}$. Consider $T = \langle H,(T_n)_{n\in\mathbb{N}}\rangle$ and $T' = \langle H',(T_n)_{n\in\mathbb{N}}\rangle$. Then $T' \geq T$, T' is valid and T is not.

4.3 Asymptotic equivalence

We shall now study more closely the properties of the equivalences induced by the asymptotic sharpness (pre) orders. In fact, it should be noticed that those pre-orders, especially the formal one, are rather poor, and are mainly valuable because of their induced equivalences.

The following theorem makes precise the relationships between sharpness equivalence and acceptability.

Stability theorem

Let C be a testing context, and T, T' be two batteries of tests for this context.
If T is logically (resp. formally) acceptable,
and T' is logically (resp. formally) asymptotically equivalent to T,
then T' is logically (resp. formally) acceptable.

In other words, acceptability is preserved by (asymptotic) equivalence.

A very surprising fact (at least for the author) is that, in practical cases, two acceptable batteries of tests are actually equivalent.

Equivalence theorem

Let $C = \langle L, S, (S), A\rangle$ be a testing context, and $T = \langle H, (T_n)_{n\in\mathbb{N}}\rangle$ and $T' = \langle H', (T'_n)_{n\in\mathbb{N}}\rangle$ be two batteries of tests for C.
Suppose that there exists a finite sub-theory A_0 of A such that $A_0 \sqcup H \vdash A_0 \sqcup H'$ (finiteness condition).
Then, if T and T' are formally acceptable, they are formally asymptotically equivalent.

Proof

Notice at first that the finiteness condition implies that
$A \sqcup H \vdash A_0 \sqcup H \vdash A_0 \sqcup H' \vdash A \sqcup H'$.
By acceptability,

$$H \bigsqcup_n T_n \vdash H \sqcup A, \quad H' \bigsqcup_n T'_n \vdash H' \sqcup A$$

(we write $\bigsqcup_n$ instead of $\bigsqcup_{n\in\mathbb{N}}$ when no confusion is threatened).
Then we have

$$H \bigsqcup_n T_n \vdash H \sqcup A \vdash H \sqcup A_0 \vdash A_0.$$

By the same trick we used for the compactness theorem (cf. §3.2), we get j such that

$$H \sqcup T_j \vdash A_0.$$

Then $H \sqcup T_j \vdash H \sqcup A_0 \vdash H' \sqcup A_0 \vdash H' \bigsqcup_n T'_n$.
Thus for all $i \in \mathbb{N}$, there exists $j \in \mathbb{N}$ such that

$$H \sqcup T_j \vdash H' \sqcup T'_i$$

From the first remark, $H' \sqcup A \vdash H \sqcup A$.
Thus $\mathcal{T} \geq \mathcal{T}'$, and, by symmetry, $\mathcal{T} \sim \mathcal{T}'$. □

Intuitively, the finiteness condition means that $\mathcal{T}$ and $\mathcal{T}'$ are not too different from one another. The theorem then states that two acceptable batteries of tests are either equivalent, or <u>very</u> different.

Consider such an example:
Take $L = \{P\}$, $S = \mathbb{N}$, $A = \{P(n), n \in \mathbb{N}\}$ (A must of course be infinite!).
Let $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ with

$$H = \emptyset, \; T_n = \{P(0) \wedge \ldots \wedge P(n)\},$$

and $\mathcal{T}' = \langle H', (T'_n)_{n \in \mathbb{N}} \rangle$, with

$$H' = A, \; T'_n = \emptyset$$

Then $\mathcal{T}$ and $\mathcal{T}'$ are formally acceptable, $\mathcal{T}' \geq \mathcal{T}$ but not $\mathcal{T} \geq \mathcal{T}'$.
Notice that $H \sqcup A \nvdash H' \sqcup A'$. It can be seen that a <u>finite</u> $A_0 \subseteq A$ such that $H \sqcup A_0 \nvdash H' \sqcup A_0$ does <u>not</u> exist.

Another example is the following one: take $S = \mathbb{N}$, $L = \{P\}$, $A = \{P(n), n \in \mathbb{N}\}$, $T_n = T'_n = \{P(0) \wedge \ldots \wedge P(n)\}$; take

$$H = \{P(0) \wedge \ldots \wedge P(k) \rightarrow \forall x \; P(x)\}$$
$$H' = \{P(0) \wedge \ldots \wedge P(k) \rightarrow P(n), n \in \mathbb{N}\}$$

However, such examples are not very common, and, in most practical cases, the finiteness condition holds.

Let $\mathcal{C}$ be a testing context, and $\mathcal{T}$ a battery of tests for $\mathcal{C}$. We say that $\mathcal{T}$ is (formally) finitely acceptable if $\mathcal{T}$ is formally acceptable and (formally) finitely valid (cf. §3.2).

Fundamental lemma

Let $\mathcal{C}$ be a testing context, and let $\mathcal{T}$ and $\mathcal{T}'$ be two batteries of tests for $\mathcal{C}$.
Suppose that $\mathcal{T}$ and $\mathcal{T}'$ are (formally) finitely acceptable and are comparable with the formal asymptotic sharpness preorder.
Then the finiteness condition holds.

Proof

We use the above notation.
Let us examine $\mathcal{T}$. By finite acceptability we get k such that $H \sqcup T_k \vdash A$. By acceptability then,

$$H \bigsqcup_n T_n \dashv H \sqcup A \dashv H \sqcup T_k$$

By the compactness trick, we get a finite subtheory of A, say B, such that

$$H \bigsqcup_n T_n \dashv H \sqcup A \dashv H \sqcup T_k \dashv H \sqcup B.$$

With $\mathcal{T}'$, we get in the same way k' and B'.
Suppose now $\mathcal{T} \geqq \mathcal{T}'$ for example.
From the first condition, it can be easily shown that

$$H \bigsqcup_n T_n \vdash H' \bigsqcup_n T'_n$$

so

$$H \sqcup B \vdash H' \sqcup B'$$

From the second one we get

$$H' \sqcup B' \vdash H \sqcup B.$$

Take now $A_0 = B \sqcup B'$, which is actually a finite subtheory of A: $H \sqcup A_0 \dashv H' \sqcup A_0$. □

We can now state our so-called fundamental theorem of testing.

Fundamental theorem of testing

Let $\mathcal{T}$ and $\mathcal{T}'$ be two (formally) finitely acceptable batteries of tests for a testing context $\mathcal{C}$.
If $\mathcal{T}$ and $\mathcal{T}'$ are comparable with the formal asymptotic sharpness preorder, then they are equivalent for this preorder.

In practice, the theory being tested is often finite, and by the compactness theorem, the above theorem applies.

Let us conclude this paragraph by showing that the potential dual statement of the above theorem is not true in general. For this purpose, take $S = \mathbb{N}$, $L = \{P,Q\}$, $A = \{\forall x\ P(x)\}$; let $(\mathcal{S})$ be the family of those L(S)-structures that validate $H = \{(\forall x\ Q(x)) \leftrightarrow (\forall x\ P(x))\}$, and such that P and Q have calculable meanings; take

$$T_n = \{P(0) \wedge \ldots \wedge P(n)\},\ T'_n = \{Q(0) \wedge \ldots \wedge Q(n)\}.$$

Then $\mathcal{T} = \langle H, (T_n)_{n\in\mathbb{N}} \rangle$ and $\mathcal{T}' = \langle H', (T'_n)_{n\in\mathbb{N}} \rangle$ are both logically acceptable, but not comparable.

4.4 Conclusion

In this section, we have described some preorders for the class of testing contexts, and for the set of batteries for a given context.

Context (conservative) restriction models increasing knowledge about the implementation being tested during the testing process. One is (in most cases implicitly) led to postulate new hypotheses in order to be able to infer correctness from the success of the tests. The main result is that context restriction cannot eliminate any potential acceptable batteries of tests.

Sharpness formalizes the relative quality of a battery of tests. We have shown that our definition is consistent with the fundamental properties: they have been proved to be monotonic. Thus, those properties are preserved by equivalence, and we can deal with acceptable equivalence classes. Furthermore, the Fundamental Theorem of Testing asserts that, in practical cases, each of those classes is maximal for asymptotic sharpness.

Notice that these nice results are not true in general with logical definitions. Once again, formal definitions turn out to be more fruitful, though less intuitive, than logical ones. Duality prevents us from stating any definition unfitted for the concrete reality of testing.

5. CONSTRUCTION, OPTIMIZATION AND QUALITY ASSESSMENT

The earlier sections were devoted to studying, from a theoretical viewpoint, the properties of testing contexts and batteries of tests. They have shown that the behaviour of those abstract objects may be quite directly related to the common intuition about program testing.

This section describes the application of this theoretical model to three important practical problems arising in this field: the effective construction of a suitable battery of tests for a given problem, the optimization of a given battery of tests according to some given criterion, and the global quality assessment of a testing process.

5.1 Towards an Effective Construction Method

As shown by the Testing Process Diagram (see §1.4) the construction phase consists of, starting with a testing context $\mathcal{C}_0$, defining a new testing context $\mathcal{C}_1$, which is a conservative restriction of $\mathcal{C}_0$, together with a formally acceptable battery of tests $\mathcal{T}$ for $\mathcal{C}_1$

$$\mathcal{C}_0 \xrightarrow[\text{construction}]{} \quad \mathcal{T} \text{ acceptable for } \mathcal{C}_1, \quad \mathcal{C}_1 \leq \mathcal{C}_0.$$

In practice, we must, of course, only deal with "likely" conservative restrictions. This means that the new hypothesis we postulate (regularity or uniformity hypothesis) must be sensible with respect to the concrete problem).

Consider the following example. Let $\mathcal{C}_0 = \langle L_0, S, (S_0), A \rangle$ be an initial context with formal hypothesis H_0. $\mathcal{C}_0$ is an H_0-context. Suppose that A is $\{\forall x\ \phi(x)\}$ where ϕ is an $L_0(S)$-formula, without any quantifier, and without any variable but x, such that any non-logical symbol of ϕ has a calculable meaning for every structure of (S). Suppose that S is countable, say S = N.
In order to construct $\mathcal{T}$ and $\mathcal{C}_1$, we may use two kinds of hypotheses, depending on the practical context.

(1) Take $H_1 = \{\phi(0) \wedge \ldots \wedge \phi(k) \rightarrow \forall x\ \phi(x)\}$, restrict C_0 with H_1 to get an $H_0 \sqcup H_1$-context C_1 (notice that $H_0 \sqcup A \vdash H_1$ for any choice of k!).
Then $T = \langle H_0 \sqcup H_1, (T_n)_{n \in \mathbb{N}} \rangle$ with $T_n = \{\phi(0) \ldots \phi(n)\}$ solves the problem. Note that the choice of k does not need to be explicit (exhaustive testing strategy).

(2) Add a new constant a to L_0, and consider $H_1 = \{\phi(a) \rightarrow \forall x\ \phi(x)\}$; restrict C_0 to get C_1 as above. Again, $H_0 \sqcup A \vdash H_1$. Now $T_n = \{\phi(a)\}$ solves the problem (random sampling testing).

We thus know how to effectively construct T and C_1 for the basic case $A = \{\forall x\ \phi(x)\}$ in many practical problems (other special sensible hypotheses could however be used). We now show that the syntactical form of A may be modified.

Construction lemma

Let $C = \langle L, S, (S), A \rangle$ and $T' = \langle L, S, (S), A' \rangle$ be two testing contexts with the same family of L(S)-structures (S).
Let $T = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ be a battery of tests for C.
Suppose that $A \sqcup H \dashv A' \sqcup H$.
Then T is an acceptable battery of tests for C' if and only if it is for C.

Proof

T is obviously a battery of tests for C' (this does not depend on A).
T is valid for C': $H \underset{n}{\sqcup} T_n \vdash A$ (validity for C), $H \underset{n}{\sqcup} T_n \vdash H$, $H \underset{n}{\sqcup} T_n \vdash H \sqcup A \vdash H \sqcup A'$ and thus $H \underset{n}{\sqcup} T_n \vdash A'$.
T is unbiased for C': for any n, $H \sqcup A \vdash T_n$ (lack of bias for C), and $H \sqcup A' \vdash H \sqcup A \vdash T_n$.
The converse is proved by symmetry. □

Now, the Decomposition Theorem transforms the problem of constructing a battery of tests for a complex theory A into (probably) infinitely many basic problems by splitting A into pieces.

Decomposition theorem

Let $\mathcal{C} = \langle L, S, (S), A \rangle$ be a testing context, with A a countable union $A = \bigsqcup_{i \in \mathbb{N}} A^{(i)}$.
For each $i \in \mathbb{N}$, let $\mathcal{C}^{(i)}$ be the testing context $\langle L, S, (S), A^{(i)} \rangle$, and let $\mathcal{T}^{(i)} = \langle H^{(i)}, (T_n^{(i)})_{n \in \mathbb{N}} \rangle$ be a battery of tests for $\mathcal{C}^{(i)}$.
Let $H = \bigsqcup_{i \in \mathbb{N}} H^{(i)}$

$$T_n = \bigsqcup_{i \leq n} T_n^{(i)} = T_n^{(0)} \sqcup \ldots \sqcup T_n^{(n)}$$

Then $\mathcal{T} = \langle H, (T_n)_{n \in \mathbb{N}} \rangle$ is a battery of tests for $\mathcal{C}$.
If each $\mathcal{T}^{(i)}$ is logically acceptable for $\mathcal{C}^{(i)}$, so is $\mathcal{T}$ for $\mathcal{C}$.
If each $\mathcal{T}^{(i)}$ is formally acceptable for $\mathcal{C}$, so is $\mathcal{T}$ for $\mathcal{C}$.

Proof

① $\mathcal{T}$ is a battery of tests for $\mathcal{C}$.

Fix n. Each $T_n^{(i)}$ being a test for $\mathcal{C}^{(i)}$ and T_n being the union of finitely many $T_n^{(i)}$, T_n is actually a test for $\mathcal{C}$ ($\mathcal{C}$ and $\mathcal{C}^{(i)}$ are built on the same (S)!).

Each $\mathcal{C}^{(i)}$ is a testing context. For each $S \in (S)$, $S \vDash H^{(i)}$, for each index i; so $S \vDash \bigsqcup_i H^{(i)}$ and $S \vDash H$.
For the same reason, for each index i, $H^{(i)} \sqcup T_{n+1}^{(i)} \vdash T_n^{(i)}$ for each n. Thus

$$\bigsqcup_{i \leq n+1} \left(H^{(i)} \sqcup T_{n+1}^{(i)} \right) \vdash \bigsqcup_{i \leq n+1} T_n^{(i)}$$

This can be rewritten

$$\left(\left(\bigsqcup_{i \leq n+1} H^{(i)} \right) \sqcup T_{n+1} \right) \vdash \left(T_n^{(n+1)} \sqcup T_n \right).$$

This shows

$$\left(H \sqcup T_{n+1}\right) \vdash T_n^{(n+1)} \sqcup T_n$$

and as a special case

$$H \sqcup T_{n+1} \vdash T_n, \text{ which holds for all } n.$$

We now focus on the formal case; the logical one can be handled in the same way.

② *T* is valid for *C*.

Consider $H \bigsqcup_n T_n$. This can be rewritten

$$\bigsqcup_i H^{(i)} \bigsqcup_n \left(\bigsqcup_{i \leqq n} T_n^{(i)} \right), \text{ that is}$$

$$\bigsqcup_i \left(H^{(i)} \bigsqcup_{n \geqq i} T_n^{(i)} \right) \qquad (*)$$

But, by projective reliability, we have

$$H^{(i)} \sqcup T_i^{(i)} \vdash \bigsqcup_{n \leqq i} T_n^{(i)} \quad \text{for any } i$$

So (*) proves actually

$$\bigsqcup_i (H^{(i)} \bigsqcup_n T_n^{(i)}),$$

and by the validity of $T^{(i)}$ for $C^{(i)}$, that proves

$$\bigsqcup_i A^{(i)}, \text{ that is A.}$$

③ *T* is unbiased for *C*.

Consider $H \sqcup A$. This may be rewritten

$$\bigsqcup_i (H^{(i)} \sqcup A^{(i)}).$$

By lack of bias of $T^{(i)}$ for $C^{(i)}$, this proves, for any n, $\bigsqcup_i T_n^{(i)}$ and, as a special case, $\bigsqcup_{i \leqq n} T_n^{(i)}$, that is T_n. □

It should be emphasized that the "diagonalization method" used to build T_n does not work if there are uncountably many $A^{(i)}$.

Given the Construction Lemma, and the Decomposition Theorem, we can now describe an effective and general method to achieve the construction phase.

Remember firstly that, following the discussion in section 3 (see mainly §3.5), we only have to deal with some purely universal "calculable" formulas. Consider therefore an initial context $\mathcal{C} = \langle L, S, (S), A \rangle$, with $A = \{\forall x\ \forall y\ \phi(x,y)\}$ with ϕ a formula as previously. Suppose $S = \mathbb{N}$.

① Firstly, restrict $\mathcal{C}$ with a uniformity hypothesis H_1 (or some other suitable likely hypothesis) for the formula $\forall x[\forall y\ \phi(x,y)]$. We then obtain $\mathcal{C}_1 = \langle L, S, (S_1), A \rangle$
$H_1 = \{\forall y\ \phi(0,y) \wedge \ldots \wedge \forall y\ \phi(k,y) \rightarrow \forall x\ \forall y\ \phi(x,y)\}$.
Under H_1, A is now equivalent to

$$A_1 = \{\forall y\ \phi(i,y),\ i \in \mathbb{N}\}.$$

② By the Construction Lemma, it suffices now to consider the context $\mathcal{C}_2$ obtained by replacing A with A_1 in $\mathcal{C}_1$.
$\mathcal{C}_2 = \langle L, S, (S_1), A_1 \rangle$.

③ The Decomposition Theorem may now be applied. It suffices to consider the (infinitely many) contexts
$\mathcal{C}_2^{(i)} = \langle L, S, (S_1), A_1^{(i)} \rangle$, with

$$A_1^{(i)} = \{\forall y\ \phi(i,y)\}.$$

④ The example described earlier may now be applied.

Notice that the Decomposition Theorem can be applied at point ③ precisely because S is countable. With an uncountable S, we might not have been able to conclude anything.

The method can obviously be extended to a more complex A. The following definition states some sufficient practical conditions on the testing contexts that can be handled in this way.

Definition Testable Testing Context

Let C = <L,S,(S),A> be a testing context.

C is said to be testable if

* S is at most countable
* A is a theory with at most countably many non-logical axioms (countable theory), each of them being a purely universal formula each of whose non-logical symbols has a calculable meaning, whatever structure of (S) is considered.

Notice that if L has at most countably many non-logical symbols (countable language), one can always (by the Construction Lemma) reduce A to a countable theory (L(S) being countable).

5.2 Construction vs. Optimization

We are now able to construct effectively an acceptable battery of tests for a (possibly restricted) given testing context. However, it is often the case that this battery of tests is not satisfactory, for some practical and/or theoretical reasons. We are thus naturally led to the optimization problem with some given criterion.

Consider firstly the formal sharpness criterion, this is, to our mind, a quite natural criterion in our formalism. Given a context C and a battery of tests T, which is acceptable for C, we try to optimize T to get a strictly sharper acceptable battery of tests. The Fundamental Theorem of testing applies (at least if the theory to be tested is finite - that is actually often the case). It states that any sharper acceptable battery of tests is in fact equivalent to T, and thus can be easily characterized.

We must therefore turn to some more precise criterion which results from some external considerations. Those considerations are not, in general, expressable in our formalism. They often deal with testing cost or profitability. Nevertheless, we can, to a certain extent, provide a method for assessing their legitimacy.

The idea is to look at the likelihood, the legitimacy, of the hypotheses which are involved in sharpness equivalence. Consider the H_0-initial context $\mathcal{C}_0 = \langle L,S,(S_0),A\rangle$. H_0 can be considered as the set of fundamental, initial hypotheses about the implementation, whose validity may not be in doubt.

Now, suppose we have already constructed (probably using the above method) an $H_0 \sqcup H_1$-context $\mathcal{C}_1 = \langle L,S,(S_1),A\rangle$, which is a conservative restriction of $\mathcal{C}_0$ (in most cases, $H_0 \sqcup A \vdash H_1$), and an acceptable battery of tests $\mathcal{T} = \langle H_0 \sqcup H_1, (T_n)_{n\in\mathbb{N}}\rangle$ for $\mathcal{C}_1$. H_1 is a set of technical construction hypotheses, probably uniformity and regularity hypotheses, which have been postulated about the implementation for theoretical reasons. They should have been carefully chosen as likely hypotheses, but in any case they may be considered as less legitimate than the initial hypotheses.

Consider now a battery of tests $\mathcal{T}' = \langle H_0 \sqcup H_1, (T'_n)_{n\in\mathbb{N}}\rangle$, which is an optimization of $\mathcal{T}$ according to some external and inexpressible extra criterion. $\mathcal{T}'$ is equivalent to $\mathcal{T}$, and we thus have

$$H_0 \sqcup H_1 \bigsqcup_n T_n \quad H \quad H_0 \sqcup H_1 \bigsqcup_n T'_n \qquad (*)$$

This means that, given the initial hypotheses and the construction hypotheses, $\bigsqcup_n T_n$ and $\bigsqcup_n T'_n$ are equivalent. We say that this optimization is of level 0 if equivalence still holds under the initial hypotheses. Otherwise, it is said to be of level 1.

A level 0 optimization can be considered as perfectly safe and legitimate. It can be seen that a typical optimization of level 0 is to increase the redundancy of the battery of tests, by adding some new experiments, already implied by the old ones, to the tests. Such an optimization will probably increase the cost of the testing process, but also the quality of the conclusion (see §5.3).

A level 1 optimization must be considered not to be as safe as a level 0 one. A typical example is to decrease the redundancy of the battery of tests, and thus the cost of the testing process. Consider for example a theory $A = \{\forall x\ \phi(x)\}$ to be tested under a construction hypothesis $H_1 = \{\exists x\ \phi(x) \rightarrow \forall x\ \phi(x)\}$; a test $\{\phi(a), \phi(b)\}$ will be, at level 1, optimized into $\{\phi(a)\}$.

These two kinds of optimization are quite usual in practice and are often left implicit. In fact, the word "optimization" tends to be used when some extra hypotheses are involved, leading to some simplifications. In our formalism, we can express this by restricting the context once more with a new hypothesis theory H_2.

$$H_0\text{-initial context} \xrightarrow[\text{construction } H_1]{} H_0 \sqcup H_1 \text{ context } C_1 \xrightarrow[\text{optimization } H_2]{} H_0 \sqcup H_1 \sqcup H_2 \text{ context } C_2$$

(remember that conservative restriction is transitive, see §4.1). H_2 is called the optimization hypothesis theory. Such an optimization is said to be of level 2.

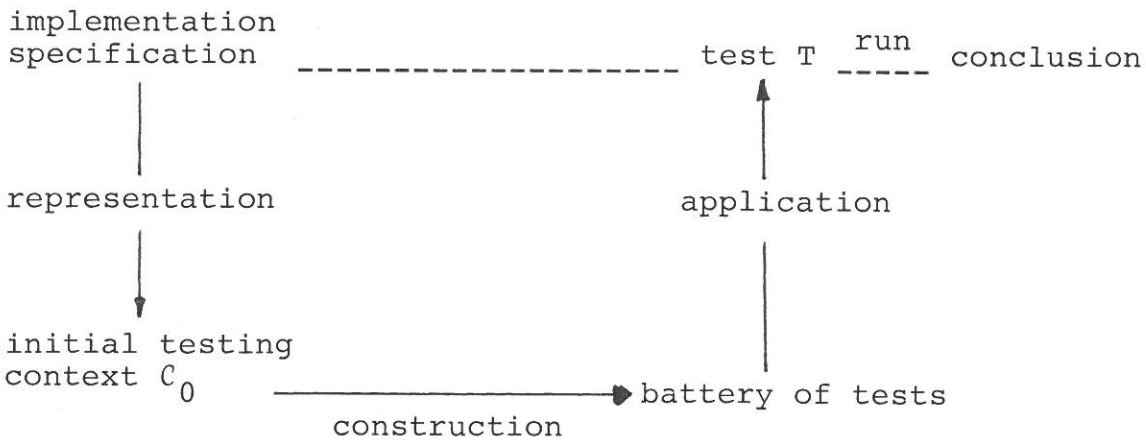
Such extra hypotheses must be considered a priori to be illegitimate and not useful. Their validity should be examined very carefully, as should the profitability of such an optimization.

An optimization of level 2 leads in general to some considerable alteration of the battery of tests.

5.3 Quality assessment of a testing process

As we are reaching the end of this work, we cannot avoid any longer the question of assessing the quality of a testing process. Even though we are not able at present to answer this question in a satisfying way, we will give here some insight into this very difficult but fundamental problem.

We can now proceed with the discussion of the Testing Process Diagram first sketched at §1.4.



From the implementation features and the specification to be tested, we get an abstract version of the problem as an H_0-testing context $C_0 = \langle L_0, S, (S_0), A \rangle$. We then construct an acceptable battery of tests $T = \langle H_0 \sqcup H_1, (T_n)_{n \in \mathbb{N}} \rangle$ for a (possibly) conservatively restricted $H_0 \sqcup H_1$-testing context $C_1 = \langle L_1, S, (S_1), A \rangle$. In the application phase, we choose a test T_p from $(T_n)_{n \in \mathbb{N}}$ and run it. If T_p is passed, the implementation is declared to be correct with respect to the given specification; if T_p fails, it is declared to be incorrect.

We suppose that the representation phase is coherent in that the concrete problem is intuitively "correctly" translated into an abstract formulation. As a special case, we suppose that there is an (unknown) structure S_r which actually represents the implementation functional behaviour.

Suppose now that the selected test T_p fails. Suppose that the implementation is nevertheless correct. Then $S_r \models A$ (coherency). Because of the conservative restriction, there exists $S'_r \in (S_1)$ such that $S'_{r|L_0(S)} = S_r$. Then by an earlier remark, $S'_r \models A_{|L_1(S)}$ and, by hypothesis, $S'_r \models (H_0 \sqcup H_1)$. But, by lack of bias, $A_{|L_1(S)} \sqcup H_0 \sqcup H_1 \vdash T_p$ (in fact for any p!). So $S'_r \models T_p$ and we get a contradiction. We then can conclude that the implementation is certainly incorrect, whatever the index p chosen. This should be related to the discussion of §1.1 about the testing "anisotropy" property.

The above proof shows that if the implementation is correct, then all the tests T_n will be passed. Conversely suppose that the selected test T_p is passed, and that the implementation is not correct. In general we cannot conclude anything. We need some more hypotheses, called coupling hypotheses.

First coupling hypothesis

> Everything happens exactly as if S_r is conserved by the restriction: one can find an $S'_r \in (S_1)$ such that $S'_{r|L_0(s)}$ behaves like S_r.

Second coupling hypothesis

> Everything happens exactly as if all the tests T_n, $n \in \mathbb{N}$ were passed - instead of T_p only.

Assume these coupling hypotheses. Because of the validity of T, $H_0 \sqcup H_1 \sqcup_n T_n \vdash A$. Thus $S'_r \models A$ (in fact $A_{|L_1(s)}$) and, by an earlier remark, $S_r \models A$, which leads to a contradiction.

We have shown that the quality of the conclusion which our testing process leads to is actually related to (and only to) these two hypotheses. We may thus identify them and define the quality of a testing process to be the validity level of the hypotheses.

It should be emphasized that these two hypotheses are essentially of the same nature. (S_1) may be viewed as a sampling of (S_0), the fairness of which is determined by the likelihood of the postulated construction hypothesis H_1. On the other hand, if T_p is passed, then, by reliability, all T_n with $n \leq p$ would have been passed. $(T_n)_{n \leq p}$ may also be viewed as a sampling of $(T_n)_{n \in \mathbb{N}}$, the fairness of which is determined by the index p. Thus, in both cases, we state that some results which are formally true on a fair sampling of a domain may be considered to be true for any object of the domain. This is no more than a special kind of regularity or uniformity hypothesis! Such kinds of coupling hypotheses have been used extensively in Mutation Testing [Budd 81].

Thus we can now (re-)define testing process quality as the fairness of those samplings. Remember that restricting the context from C_0 to C_1 may be regarded as increasing one's knowledge (in fact by postulating some likely knowledge) about the implementation. On the other hand, the criterion for choosing T_p out of $(T_n)_{n \in \mathbb{N}}$ is mainly running cost. Testing process quality is thus directly related, through fairness, to

(1) the quality of information about the tested implementation

(2) test running costs.

It must be verified that perfect information and infinite cost actually lead to a perfectly safe testing process conclusion. It is sufficient to look at an incorrect implementation. The initial context C_0 looks like $\langle L, S, \{S_r\}, A \rangle$: the family of structures has collapsed to a singleton. Suppose, for the sake of clarity, that $A = \{\forall x\ P(x)\}$ and $S = \mathbb{N}$. There exists an integer k such that $S_r \not\models P(k)$, so that $S_r \models \{P(0) \wedge \ldots \wedge P(k) \rightarrow \forall x\ P(x)\}$ (perfect information). We restrict the context C_0 with this hypothesis to C_1 (that is not to do anything), and take $T_n = \{P(0) \wedge \ldots \wedge P(n)\}$. We get an acceptable battery of tests for C_1, and we may pick out T_k (infinite cost), which obviously fails.

The above discussion shows that, under perfect conditions, testing an implementation may <u>prove</u> its correctness. Proving appears here as a special case of testing, namely as an extrapolation of

testing to infinity. In practice we are thus led to consider program testing as a finite but highly valuable approximation of program proving, which is itself, in most cases, quite untractable. This is the very specificity of testing to give a continuous graduation of correctness assessments instead of the correct/incorrect (often humanly undecidable) alternatives of proving.

5.4 Conclusion

Our formalism, by the rigorous application of mathematical tools, allows us to give a precise account of several important notions.

We have described a general framework for constructing a suitable battery of tests for a given initial context. We have defined some precise conditions about the considered context that allows the method to be applied in a safe way. This method leads one to postulate some new construction hypotheses about the implementation, the likelihood of which must be examined in each practical case.

We have also given a precise meaning for the word optimization. Using the sharpness criterion, the Fundamental Theorem of Testing shows that any optimized battery of tests is in fact equivalent to the original one. We have shown that extra external criteria may, to a certain extent, be characterized. Level 0 optimization is quite safe and typically increases redundancy. Level 1 optimization is legitimate; its profitability must be assessed; it typically decreases redundancy. And lastly, level 2 optimization is in principle illegitimate, but may be useful in some practical cases; it alters the structure of the battery of tests considerably by considering some new hypotheses about the implementation.

Lastly, we have given some insight into testing process quality assessment. We have shown that the quality of the decision about implementation correctness is only related to the quality of the information about implementation, and to the testing cost. Perfect knowledge and infinite cost lead to a perfectly safe conclusion i.e. a proof of implementation correctness.

6. CONCLUSION

We have presented an abstract formalization for the notion of program testing, based on a rigorous utilization of first-order logic as an underlying mathematical tool.

6.1 On Formalization Power

We feel that the wideness of application of our model is its main powerful feature. Earlier papers have concentrated on the notion of a test (or a test criterion), rather than on the testing process. The authors were thus unable to give a precise account of the specific "likely" feature of testing process conclusion.

By splitting the testing process into three phases, we can precisely localize this "likely" feature and model the other actions as rigorous, formal manipulations of logical theories. The likelihood of the testing process may thus be related to the precise intuitions of knowledge quality and testing cost (see §5.3). However it should noticed that all of this is only concerned with the abstract level of the Testing Process Diagram. In practice, one must also carefully assess that the abstract problem represents in a "satisfying" way the concrete problem to be solved. Such an assessment can of course only be empirical and is inherent in any modelling process.

Let us now focus on our underlying mathematical tool. We have chosen first-order egalitary logic for several reasons. As most of the earlier papers in the field (implicitly) used it, it is quite a natural tool in this area. It allows us to express directly and easily the common intuitions about program testing. On the other hand, it is a well-known mathematical tool, and we may get benefits from its classical notation and methods: its "spiritual tradition". Yet the main reason for this choice (instead of statistical theories, for example) is our conviction about the complementary natures of testing and proving programs.

We therefore chose some common mathematical tool in order to enable further comparisons (see next paragraph).

Having chosen our tool, we had to respect its own nature, and requirements. This has led us to somewhat complex definitions and expressions ("formally projectively reliable"!). Yet, to our mind, this effort is quite fruitful in the long term. We have got some precise theorems which describe and justify precisely some common intuitions. Discriminating formal properties from logical ones allows us to make precise the relationships between the concrete (logical) level and the abstract (formal) level in a surprisingly efficient way.

6.2 Further research goals

Testing vs. Proving

This work allows us to understand better the relationships between testing and proving. On the one hand, testing can be viewed as a special case of proving (see §1.1). Building an acceptable battery of tests for a given testing context is nothing more than building a proof of the property to be tested, according to some precise conditions (decidability of experiments, for example) about the form of this proof. On the other hand, proving can also be viewed as a special case of testing (see §5.3). We have shown that extrapolating testing to infinity produces proving.

It should certainly be very fruitful to study those relationships in a deeper way, leading perhaps to unified theory of program correctness.

Quantity Quality Assessment

In the last section we have discussed quality assessment of the testing process (see §5.3). In fact, we have not been able to give a precise and quantitative assessment method. Yet such a method would be highly valuable; it should, at first glance, deal with some probabilistic tools, but our mathematical tool is unfortunately not very well suited to these kinds of concepts. Some further research, involving perhaps some advanced concepts of mathematical logic, are needed here.

Towards Automatical Test Generation

In an earlier work [Boug 82b] we have extensively described the application of our model to some concrete programs.

The first example deals with a quite common sorting program. It can be noticed that the very heart of the testing process, the construction phase, is somewhat tedious because it is so repetitive. The same holds also for the optimization phase. In fact, the central problem is to choose some likely hypothesis according to one's own intuition about implementation behaviour.

The testing process may be considerably simplified if we do not consider any given implementation and any given specification, but their abstract versions, that is a Σ-algebra and a Σ-data type, Σ being a signature (see [Gaud 80] for an introduction and a bibliography about abstract data types). In most cases some standard hypothesis may be used, and the testing process may be easily handled automatically. [Boug 82b] describes an experimental test generation system we have implemented according to this method.

There seem to exist some promising applications for our testing theory with abstract data type specifications, leading perhaps to powerful automatical test generation methods.

6.3 All is well that ends

We cannot find any better conclusion than GOODENOUGH & GERHART's one in their earlier paper [Good 75]:

> We know less about the theory of testing
> that we do often
> than about the theory of proving
> that we do seldom.
> This paper is a step toward redressing
> this imbalance.

Acknowledgement

This work is mainly based on the author's doctoral thesis [Boug 82b], which was submitted to the following jury:

B. Robinet, chairman
M.C. Gaudel, M. Nivat, J.P. Jouannaud, H. Gallaire, examiners
J.C. Rault, K. Culik II, guests.

I would like to thank all of them for their interest in this work and their encouragement. I owe a special debt to M.C. Gaudel for her help during the last years.

Special thanks to K. Møller who typed this paper, and to N. Derrett who corrected patiently many language errors, in English as in Danish.

Bibliography

[Boug 82a] Bougé, L.
Validation de programmes par test: Théorie et pratique.
LITP Report 82-18, Université Paris 6 & 7, Paris, France, April 1982.

[Boug 82b] Bougé, L.
Modélisation de la notion de test de programmes; Application a la production de jeux de tests.
Thèse de 3ème cycle, Université Paris 6, Paris, France, October 1982.

[Budd 81] Budd, T.A.
"Mutation Analysis: Ideas, Examples, Problems and Prospects."
Computer Program Testing, B. Chandrasekaran, S. Radicchi (Eds.). North-Holland Publishing Company, 1981, pp. 129-148.

[Gaud 80] Gaudel, M.C.
Génération et preuve de compilateurs basés sur une semantique formelle des langages de programmation.
Thèse d'état, Institut National Polytechnique de Lorraine, Nancy, France, May 1980.

[Gerh 79] Gerhart, S.L.
"Program Validation".
Computing System Reliability, T. Anderson, B. Randers (Eds.). Cambridge, England: Cambridge, 1979, pp. 66-108.

[Good 75] Goodenough, J.B., Gerhart, S.L.
"Toward a Theory of Test Data Selection".
SIGPLAN Notices, 10 (6), June 1975, pp. 493-510.

Another version may also be found in
"Toward a Theory of Testing: Data Selection Criteria".
Current trends in programming methodology, Vol. IV,
R.T. Yeh (Ed.) Englewood Cliffs, N.J.: Prentice-Hall,
Inc., 1977, pp. 44-79.

[Shoe 67] Shoenfield, J.R.
Mathematical Logic.
Addison-Wesley, Inc., 1967.

[Weyu 80a] Weyuker, E.J.
"The Oracle Assumption of Program Testing".
Proceedings of the 13th Hawaii International Conference on system sciences, Vol. I, 1980, pp. 44-49.

[Weyu 80b] Weyuker, E.J., Ostrand, T.J.
"Theories of Program Testing and the Application of Revealing Subdomains".
IEEE, Vol. SE-6 (3), May 1980, pp. 236-246.

[Whit 81] White, L.J., Cohen, E.I., Zeil, S.J.
"A Domain Strategy for Computer Program Testing".
Computer Program Testing, B. Chandrasekaran,
S. Raddicchi (Eds.). North-Holland Publishing
Company, 1981, pp. 103-113.

Appendix: Testing a sorting program

This appendix is aimed to apply the theory developed above to test a realistic example: a sorting program. This testing process is presented along the lines described in section 5.

Consider a system P running the following program (inspired by J.C. REYNOLDS). This program takes as input an array X[1..n] of integers and sorts it. n is an integer greater than 1.

```
var m,j,k,t: integer;
begin
  m:=n;
  while 1<m do
    begin
      j:=1; k:=1;
      while k<m do
        begin
          k:=k+1;
          if X[k] > X[j] then j:=k end
        end
      t:=X[j]; X[j]:=X[m]; X[m]:=t;
      m:=m-1
    end
end
```

The input assertion is

"X is an integer array of size n≥1"

The output assertion is

"X' is an integer array of size $n \geq 1$, which is obtained by a permutation from X, and which is sorted w.r.t. $\leq$",

where X' denotes the final state of X.

The property to be tested is

{Input assertion} P {Output assertion}

It can be expressed in our logical framework by

$$A = \forall n \; \forall X \; [\text{Input}(X,n) \rightarrow \text{Output}(X',X,n)].$$

1. Representation

From the object P (the system running the sorting program) and the property to be tested, A, we have to build an initial testing context $C_0 = \langle L_0, S, (S_0), A \rangle$ (cf. §2.2), and a set H_0 of initial hypotheses validated by all the potential behaviours S_0 of the system.

Universe and language

We have to deal with lengths of arrays, elements of arrays and arrays. Lengths vary over $N \setminus \{0\}$, elements over Z, and arrays over Z*. We thus choose

$$S = N \setminus \{0\} \sqcup Z \sqcup Z^* \sqcup \{\perp\} \qquad \text{(disjoint sum)}$$

Cook the language L_0 in the following way. First take predicate symbols to distinguish between the (disjoint) components of S: Natural, Element, Array, $\perp$.

Then add the symbols used in the program under test:

$_{-}<_{N}{-}$, $_{-}\leq_{Z}{-}$, $_{-}<_{Z}{-}$, $_{-}+_{Z}{-}$, $_{-}-_{Z}{-}$, 1_{Z} (we will drop indices).

Now, one has to express the formula A to be tested. Thus add the function symbols Access which returns an element of an array, and Length which returns the length of an array. Add also the predicate symbols Permutation which means that two arrays are permutations one from the other, and Sorted which indicates that an array is sorted w.r.t. $\leq_{Z}$. Lastly, choose a new functional symbol F to express the external behaviour of the system under test: if x is an array, F(x) is the value output by P with input x.

Notice that S and L_0 are countable.

Structures and hypothesis

The family (S_0) of $L_0(S)$-structures models the potential (correct or incorrect) behaviours of the system. We postulate, following [Budd 81] that whatever this behaviour is, it is not "too incorrect" (competent programmer assumption). All happens exactly as if all symbols but F were correctly implemented.

(S_0) is thus the family of all the $L_0(S)$-structures such that:

* all the symbols of L_0 but F have their ordinary meanings on their ordinary domains, and are extended by $\perp$ for function and False for predicates elsewhere.
* F is a calculable function on Z*, extended by $\perp$ elsewhere, such that for any array x, F(x) is a permutation of x.

By the way, we implicitly postulate that P halts and outputs a value for any array given as input, and that this value is a permutation of this array. Notice that all the symbols of L_0 have a calculable meaning for any structure S_0. H_0 is namely the set of all the $L_0(S)$-formulas validated by all the structures of (S_0).

Property under test

The property to be tested is

$$\{Input(X,n)\}\ P\ \{Output(X',X,n)\}$$

This intuition can be caught by the following theory

$$\begin{aligned} A = \{&\forall n\ [Natural\ (n) \rightarrow \\ &\forall x\ [Array\ (x) \wedge Length\ (x) = n \rightarrow \\ &\quad Length\ (F(x)) = n \wedge \\ &\quad Permutation\ (x,F(x)) \wedge \\ &\quad Sorted\ (F(x))]]\} \end{aligned}$$

This choice and the previous remarks show that the H_0-context $C_0 = \langle L_0, S, (S_0), A\rangle$ is a testable testing context (cf. §5.1). The general method described earlier may thus be applied.

2. Construction

C_0 may not admit any acceptable battery of tests. We thus have to restrict it conservatively (cf. §4.1) to a new context C_1 by enriching L_0 with some new symbols and adding some new construction hypotheses H_1. Because of transitivity, we can proceed incrementally.

We start with $L_1 := L_0$

$H_1 := \emptyset$ (construction hypotheses)

$(S_1) := (S_0)$

First reduction

H_0 contains the following formulas

$\forall x$ [Array (x) → Permutation (x,F(x))]

$\forall x$ [Array (x) → Length (x) = Length (F(x))].

By the Construction Lemma (cf. §5.1), noticing that we are looking for an acceptable battery of tests of the form $\langle H_0 \sqcup H_1, (T_n)_n \rangle$, we can replace A by the theory B

B = {$\forall n$ [Natural (n) →

$\forall x$ [Array (x) ∧ Length (x) = n → Sorted (F(x))]]}

Discarding $\forall n$

Following the method presented in §5.1, we now have to discard the first quantifier. We now describe this elimination in some detail.

Let us add a new functional symbol b to L_0 and its defining axiom to H_1.

$$\forall n\ [b(n) \leftrightarrow \forall x\ [\text{Array}\ (x) \wedge \text{Length}\ (x) = n \rightarrow \text{Sorted}\ (F(x))]] \qquad (1)$$

$L_1 := L_1 \sqcup \{b\}$

$H_1 := H_1 \sqcup \{(1)\}$

We extend the structures (S_1) along this definition: this restriction is conservative (and in fact purely notational).

By the Construction Lemma, we replace B by

$$\{\forall n\ [\text{Natural}\ (n) \rightarrow b(n)]\} \qquad (2)$$

We would prefer of course $\{b(1), b(2), \ldots\}$. Following the method, we are led to add the regularity hypothesis

$$b(1) \wedge \ldots \wedge b(k) \rightarrow \forall n\ [\text{Natural}\ (n) \rightarrow b(n)] \qquad (3)$$

Notice that $A \sqcup H_0 \vdash (3)$, so that the associated restriction is actually conservative.

Intuitively, (3) means that if P behaves "correctly" for any array of length less than k, then it does so for any array. There always exists such a k. For if P is "correct", take k=1, and otherwise take k the length of some array incorrectly handled. This regularity hypothesis is thus legitimate.

We therefore make the associated conservative restriction $H_1 := H_1 \sqcup \{(3)\}$, and we replace B by $\{b(1), b(2), \ldots\}$. By the Decomposition Theorem (cf. §5.1), we are led to test the theory $\{b(n)\}$ where n is a "generic" constant such that $n \geq 1$.

Thus we have eliminated the first quantifier.

Discarding ∀x: first stratification

By the Construction Lemma, we can expand b to its original form. We are therefore now looking for an acceptable battery

of tests for the $H_0 \sqcup H_1$-testing context $\langle L_1, S, (S_1), C \rangle$, with

$$C = \{\forall x\ [\text{Array}\ (x) \wedge \text{Length}\ (x) = n \rightarrow \text{Sorted}\ (F(x))]\}$$

(remember that n is here a "generic" constant).

Because there is no natural complexity order for the set of the arrays of integers of Length n (n given), we cannot apply the previous method. We have to use a uniformity hypothesis.

As a first attempt, we postulate that two arrays, whose elements are in the same relative order, are equivalent with respect to the property under test, C. That is to say the sub-domains are described by a family of predicates U_p, where p is a permutation of {1..n}, with

$$U_p\ (x) = \text{Length}\ (x) = n \wedge x[p(1)] \leqq \ldots \leqq x[p(n)]$$

Let us restrict conservatively our current context by adding the n! symbols U_p to L_1, and the n! corresponding defining axioms to H_1 of the form

$$\forall x\ [U_p\ (x) \leftrightarrow [\text{Array}\ (x) \wedge \text{Length}\ (x) = n \wedge \text{Access}\ (x,p(1)) \leqq \text{Access}\ (x,p(2)) \wedge \ \ldots \ \text{Access}\ (x,p(n-1)) \leqq \text{Access}\ (x,p(n))]] \qquad (4)$$

Then, any structure of (S_1) validates

$$\forall x\ [\text{Array}\ (x) \wedge \text{Length}\ (x) = n \leftrightarrow \bigvee_p U_p\ (x)] \qquad (5)$$

Therefore, we can add those formulas to H_1 without modifying our current context

$$H_1 := H_1 \sqcup \{(5)\}$$

Then, applying the Construction Lemma, we are led to replace C by the following (finite) theory

$$\{\forall x\ [U_p\ (x) \rightarrow \text{Sorted}\ (F(x))], p \text{ permutation of n elements}\}$$

Applying the Decomposition Theorem, we only have to deal with the simpler theory

$$\{\forall x\ [U_p\ (x) \rightarrow \text{Sorted}\ (F(x))]\}$$

with p generic.

We have thus restricted the quantifier range to a "more uniform" subdomain than the primitive one.

Discarding ∀x: second stratification

We are thus looking for an acceptable battery of tests for the context $\langle L_1, S, (S_1), D\rangle$ with

$D = \{\forall x\ [U_p(x) \rightarrow \text{Sorted}\ (F(x))]\}$ p generic

Following the general method we have to assess U_p uniformity. For example, looking at the text of the program, is it likely that system behaviour correctness for the array (0 0 0) and the array (-3872 -683 -22) are related? It is definitely not the case, and we have to stratify once again our sub-domains to get some "more uniform" ones.

The problem is namely with the borders of the U_p's. A good thing to do is to cut them out. We only sketch the (maybe tedious) method as it is the same as above.

In each U_p we distinguish between strict inequalities and equalities. We therefore get 2^{n-1} sub-domains V_q. The defining axioms of the V_q look like, for example,

$$\forall x\ [V_q\ (x) \leftrightarrow [U_p\ (x) \wedge \text{Access}\ (x,p(1)) = \text{Access}\ (x,p(2))$$
$$\wedge\ \neg\ \text{Access}\ (x,p(2)) = \text{Access}\ (x,p(3))$$
$$\ldots \wedge\ \text{Access}\ (x,p(n-1)) = \text{Access}\ (x,p(n))]]$$

Lastly, we have only to deal with the simpler theory

$$\{\forall x\ [V_q\ (x) \rightarrow \text{Sorted}\ (F(x))]\}$$

with q generic.

Discarding ∀x: uniformity hypothesis

We now feel that each of the $n! \times 2^{n-1}$ sub-domains V_q is very likely uniform with respect to the property under test. One of the main arguments is that all the data of a V_q follow the same path in the program.

We can now apply the general method to the context $\langle L_1, S, (S_1), E \rangle$ with

$$E = \{\forall x \; [V_q(x) \rightarrow \text{Sorted}(F(x))]\} \qquad q \text{ generic.}$$

We add a new constant t to L_1. We extend each structure S_1 by assigning to this new constant successively all the values of V_q. The restriction is conservative (there is at least such a value). We can add to H_1 the axiom $V_q(t)$.

Then we restrict our context with the uniformity formula

$$\text{Sorted}(F(t)) \rightarrow \forall x \; [V_q(x) \rightarrow \text{Sorted}(F(x))]$$

Again, the restriction is conservative, and we get a $H_0 \sqcup H_1$-testing context C_1.

It admits the acceptable battery of tests $\mathcal{T}$

$$\mathcal{T} = \langle H_0 \sqcup H_1, (T_n)_n \rangle$$

with $T_n = \{\text{Sorted}(F(t))\}$ which is actually a test by calculability of F.

3. Application and optimization

We first note that, for each n, we have only finitely many (n!) sub-domains U_p, and for each U_p finitely many (2^{n-1}) V_q. In such a case, we can prove a slightly different version of the Decomposition Theorem (cf. §5.1).

Decomposition Theorem (finite case)

Let $C = \langle L,S,(S),A\rangle$ be a testing context, with A a finite union $A = A^{(0)} \sqcup \ldots \sqcup A^{(k)}$.

For i = 0,k let $C^{(i)}$ be the testing context $\langle L,S,(S),A^{(i)}\rangle$, and let $\mathcal{T}^{(i)} = \langle H^{(i)},(T_n^{(i)})_n\rangle$ be a battery of tests for $C^{(i)}$.

Let $H = H^{(0)} \sqcup \ldots \sqcup H^{(k)}$ and

$$T_n = T_n^{(0)} \sqcup \ldots \sqcup T_n^{(k)}$$

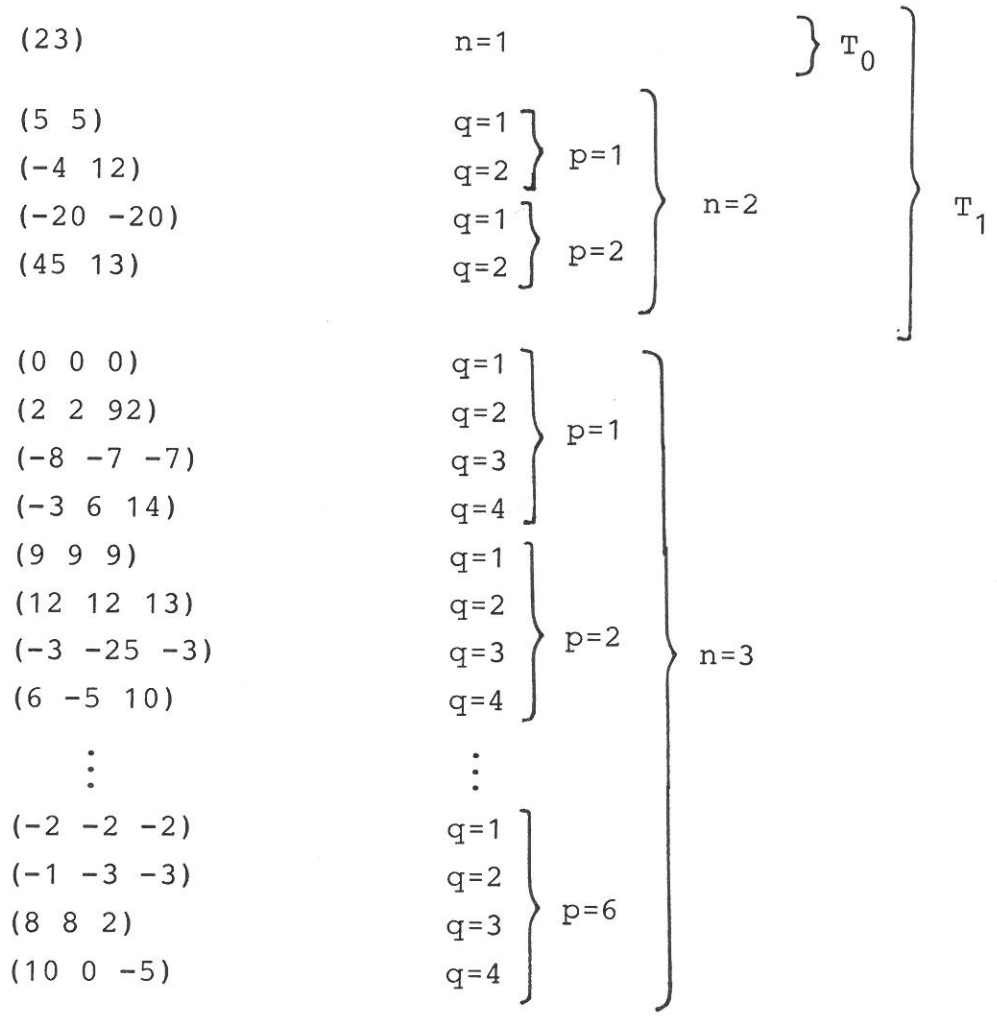
Then $\mathcal{T} = \langle H,(T_n)_n\rangle$ is a battery of tests for $C = \langle L,S,(S),A\rangle$.

If each $\mathcal{T}^{(i)}$ is logically acceptable for $C^{(i)}$, so is $\mathcal{T}$ for C.

If each $\mathcal{T}^{(i)}$ is formally acceptable for $C^{(i)}$, so is $\mathcal{T}$ for C.

Putting now all our pieces of batteries of tests together with the Decomposition Theorem (general case for n, finite case for U_p and V_q), we get our final battery of tests $\mathcal{T} = \langle H_0 \sqcup H_1,(T_n)_n\rangle$, acceptable for the testing context $C_1 = \langle L_1,S,(S_1),A\rangle$.

We then pick out one of its tests according to our quality/cost requirements, and apply it to the system P. Figure 1 shows what applying T_2 looks like. Remember that the concrete object associated with a uniformity constant t for V_q is nothing more that a randomly picked out array. We are therefore typically using a random sampling strategy.



To test P at level 2, run P on those data and for each run, check that the output is Sorted.

Figure 1: Application of T_2

We now look at some optimizations of *T*.

Level 0 optimization

At this level, assuming H_0 (the initial hypothesis) is enough to prove equivalence of the battery of tests and its optimized version.

We can for example pack our battery of tests by letting $T'_n = T_{2n}$, and keeping the same hypotheses. The condition $H_0 \underline{\sqcup}_n T_n \vdash H_0 \underline{\sqcup}_n T'_n$ obviously holds. We can also draw it out by letting $T'_n = T_{E(\frac{n}{2})}$. More complex manipulations may be designed, of course.

A more radical level 0 optimization consists in increasing redundancy. Assume for example, that you suddenly get your doubts about the behaviour of P. Then you may replace, in the tests T_n, Sorted (F(t)) by

... by Array(F(t))∧Permutation(t,F(t))∧Sorted(F(t))

Your battery of tests becomes redundant and more expensive, but less dependent on the hypotheses H_0 about P.

Level 1 optimization

At this level, equivalence proof may use H_1 (construction hypotheses, mainly regularity and uniformity hypotheses) as well. You may increase redundancy. Assume for example that, for personal reasons, you consider 0 as a very crucial value. You would like to add the arrays filled with 0 as experiments. At level 1 you can optimize $(T_n)_n$ into (T'_n) with

$$T'_n = T_n \sqcup \{\text{Sorted } (F(0^k)),\ 1 \leq k \leq n+1\}$$

where 0^k is the array of length k filled with k 0.

You may also decrease redundancy, and, by the way, running costs. Assuming $H_1 \sqcup H_0$, one can prove for example that the constants associated with (0 0 0) and (9 9 9) (cf. Figure 1) belong to the same V_q. By uniformity hypothesis one of them can be cancelled. Using such arguments, the size of T_n may be considerably reduced.

Level 2 optimization

By making some extra postulates H_2 (optimization hypotheses) the sizes of our tests may be even more reduced. For example, we could state that "if P works for an array of the form (a a a), then it works for any array of the form (a a ... a) of length greater than 3". Then, in the T_n, $n \geq 2$, we can cancel all experiments of the above form but one.

Many other kinds of extra postulates may of course be designed according to the knowledge and the confidence one has about the system under test.

4. Conclusion

A realistic application of our theory of testing has been described in some detail. Several of its aspects should be stressed.

The test we generate after optimization is roughly one that any experienced programmer would have designed for this sorting program. This is also typically one obtained by the method described in [Good 75] etc. (stepwise specification refinement). Our theory is thus fairly faithful to the primitive intuition.

In contrast to previous attempts, the core of our test generation consists only in abstract manipulations of theories following formal deduction rules, and is therefore correct, exactly as proving is, whatever you prove. We are able to relate the quality of the whole process to precise criteria (cf. §5.3). Notice particularly that this discipline enforces explicitation of up-to-now implicit hypotheses.

It should be noted that the test generation process depends very little on the program high-level text. It is so because of our "black box" modelization of the system under test (cf. §2.2). In contrast, program text is the best criterion one generally has to assess hypotheses quality.

The generated test is very closely related to the syntactical form of the property to be tested. In fact the whole generation process can be easily handled by a machine using standard construction hypotheses. Of course, no assurance is given about the quality of the result, and manual assessment is needed. In restricted cases such as abstract data types specification testing, syntactical sufficient conditions for quality can be described. In those cases, automatic test generation is fully justified and has indeed already been experimented by the author [Boug 82b].