An Analytical Approach to Communication Protocol Testing

Son T. Vuong, Jadranka Alilovic-Curgus
UBC Department of Computer Science

CICSR-TR92-010
An Analytical Approach to Communication Protocol Testing

Technical Report
Son T. Vuong
Jadranka Alilovic-Curgus

March 18, 1992

Computer Science Department and CICSR
University of British Columbia
Vancouver, B.C. Canada V6T 1W5
1 Introduction

Building trusted software systems is quickly becoming an important issue in both state-of-the-art and practice of software engineering. It addresses the issue of building software which can be relied upon to safely perform in a wide range of high performance and stress situations and control life-critical systems in its most critical extreme. Important concepts in software engineering which are emerging as crucial in the 1990's address this issue, among other, with formal methods for software specification, verification and testing.

Although the potential advantages of formal methods have been widely reported, their use in industry has been so far very limited. Industry still reports the (very informal) use of outdated methods for assuring the quality of the systems built, most notably in the area of software testing. The methods used are the ones that had been developed for testing sequential, low performance software systems where safety was not a critical issue. As reported in [NS90], the perceived problem in industry related to the use of formal methods, is the ease of applicability of these methods and the lack of realistic examples to demonstrate their usefulness.

We address the problem of building trusted systems by focusing on the problem of assuring the reliability of these systems through specification based testing. We concentrate on communication protocols, for the following two reasons. These systems satisfy the most stringent requirements of current complex software systems: they are reactive, concurrent and nondeterministic processes, with recent protocols which fall into category of high performance systems as well. Secondly, there is a well researched body of work related to the formal specification, verification and testing of these systems.

In our research we start with formal description of communication protocols,
and make use of the available theory, to develop a new approach to the testing of communication protocols. We are guided by the requirements of the industry that the approach be immediately applicable to realistic examples and easy to use, and at the same time we impose the theoretical requirement that the method should be mathematically sound. It turns out that the analytical approach to communication protocols testing is very promising in this respect, which should not be surprising. Formally defined objects are amenable to objective measurement as observed in [NS90], and test selection and evaluation of the thoroughness of the testing process is indeed best done through objective set of criteria. Some other independent results, e.g. [Ham87], further justify the necessity of a completely new approach along the lines such as we report here, by seriously disputing the trustworthiness of the traditional software testing methods.

In this report, we address the following interesting fundamental problem encountered in conformance testing: how to measure the quality of a set of test cases and how to generate or select test suites with some good (objective) coverage measure.

Formal work on test coverage metrics for protocols has been long overdue, but to our knowledge, none has been reported in the literature. There have been some research efforts on fault models [vBAD+91] and related test sequence generation based on FSM model of protocols, which addressed the problems of test selection and evaluation as well. However, in general, lacking a suitable objective test coverage metrics, existing formal methods for protocol test generation suffer from two problems: (i) they don’t systematically cover protocol behaviour beyond its relatively fundamental parts, and (ii) they allow little or no analytical insight with respect to recursion and parallelism. For modern, complex protocols, where almost all of the activities occur beyond the basic scope and involve recursion and parallelism, neglecting these aspects of the protocol behaviour in conformance testing can produce very limited test suites which convey unrealistic picture of the quality of implementations, specially when they are executed in various high load conditions in real-time, high-speed environment.

Our proposed approach is based on two fundamental notions used in recent research in protocol engineering: 1) standardized formal specifications of protocol behaviour, and 2) the testing theory for process-algebra based languages for description of concurrent non-deterministic processes. In this approach, we introduce a metric space on top of the space of all execution sequences inferred by the formal semantics of the specification language. Basically, a protocol can be viewed as having a control part and a data part. The control part of a protocol can be unfolded into (possibly infinite) execution tree of externally observable
events, whereas the data part of the protocol constitutes the specification of individual data parameters for every observable event (protocol data unit, PDU, or service primitive, SP). In this report we confine ourselves to investigating analytically the control part of the protocol behaviour.

For the control part, we introduced a framework which captures the notions of approximations of different patterns of behaviors (different temporal ordering of externally observable events) using well defined metrics which are not only mathematically tractable, but also applicable in practical settings. It is clear that metrics based on operational semantics of process algebras are not practically tractable. The metric we introduce is based on specific notions of distance of execution sequences which may be finite or infinite (representing recursive behavior). Thus, we called it a testing distance $dt$. This metric is applicable to concurrent systems represented by (closed) expressions in the LOTOS FDT without value passing (i.e. basic LOTOS).

In this report, we present our preliminary findings on test metrics for the control part of protocols and its application for test coverage evaluation and test selection. In Section 2, after establishing the background on the metric function and metric model, we introduce the definition of distance $dt$. This definition forms the basis of the control coverage metrics defined in Section 3. Section 4 presents an effective metric-based test selection method for communication protocols. Together with some further results which link the intuitive and traditional software testing criteria with the results obtainable within our analytical model, this section wraps up our theoretical investigation into the advantages and possibilities of the analytical approach to the protocol engineering activities. Finally, in the conclusion section we present open research issues related to the testing metrics as well as further work within this model.

2 Background and Metric $dt$

Our analytical approach to testing activities within communication protocols engineering can be summarized as follows. We define a coverage metric to solve the following fundamental problems in conformance testing: (i) how to generate a generic (super) test suite, $S$, to "fully" cover a given protocol specification within the space and time resources available, (ii) how to select a subset of test cases, $T$, from a given generic test suite, $S$, so as to maximize the coverage, and (iii) how to determine the coverage of a given test suite, $T$ w.r.t. a given protocol specification or its derived generic (super) test suite, $S$. We address these problems, based on the concepts of distance and control coverage metrics. In this section,
we first present some background on metric model, then define the distance $d_t$ which leads to the definition of control coverage metrics.

### 2.1 Background on Metric Model

Essentially, we build a metric space on top of the space of all execution sequences inferred by the formal semantics of the specification language. An execution sequence is considered as a point in the space, $D$, of all execution sequences. Our metric is derived based on the notion of distance between two execution sequences, i.e. between two points in $D$. This distance is designed to support our intuitive notion of testing. Therefore, we call it a testing distance $d_t$. In addition, as we shall see in the next subsection, the parameters in the definition of metric $d_t$ are tunable to facilitate the application of the metric to specific protocols. Nonetheless, these parameters are subject to some constraints which are carefully laid out so that the resulting metric always yields a metric completion of the initial space of all specified execution sequences according to the formal specification of the protocol. Also, all the information required for the computation of this metric is contained within the execution sequences alone. This property distinguishes this metric from other well known software metrics, which require relatively complex information on the structure of (program) specifications.

By defining a distance which measures how far a test sequence is from its closest potential neighbor (another test sequence) in a set of execution sequences of the specification or in a test suite, a number of fully automated algorithms can be developed, that facilitate the problems of test design, generation/selection and evaluation, such as those defined previously in this section. Intuively, the smaller the distance, the "closer" and more similar a test sequence is to some specified or already selected test sequence. Based on this distance, the average or, alternatively, maximum distance can be computed to measure how well a test suite covers the original specification. These control coverage metrics are defined in the Section 3. Alternatively, if a threshold distance is given such that all selected test sequences are no farther than that distance apart from an execution sequence in the original specification, we are able to say with some degree of certainty that the behaviour of protocol implementation is well exercised.

In defining our distance $d_t$ as given in the next subsection, we have kept in mind the following characteristics of the metric.

1) Our distance is such that, for any converging sequence of execution sequences, its limit is in the same space of execution sequences as formally specified. Therefore, we can claim that we are indeed approximating only those behaviors
which are in the original execution sequence space.

2) Our distance is such that, all (including any infinite) execution sequence can be approximated by some converging sequence in the specification. This allows us to claim that under suitable conditions of the algorithm, we will be approximating all execution sequences which can be found in the original specification.

3) If a pseudo-exhaustive testing strategy is applied with some threshold distance specified, we will in effect produce a set of test sequences which can be viewed as a set of spherical regions or "balls" approximating the set of all execution sequences of the specification. Each test sequence corresponds to a ball which represents an equivalence class in which all execution sequences are similar ("close") enough that they can be approximated by the test sequence, with a degree of accuracy dictated by the radius of that ball, i.e. the given threshold distance.

2.2 Testing Distance \( dt \)

From now on we assume we are working with a CCS-like process algebra or LOTOS without value passing (the interprocess communication is thus reduced to pure synchronization). The omission of the data part is for the sake of simplicity and should represent no restrictions on the overall results.

Let \( m \) range over a set of actions \( \text{Act} \), and \( E \) range over a suitable set of process expressions \( P \). Then, a derivation is a finite or infinite sequence of the form \( E_0 \rightarrow m_1 \rightarrow E_1 \rightarrow m_2 \rightarrow E_2 \ldots \) and a computation is a non-extendable derivation.

If we take the initial term to be always \( E_0 \), then a generic derivation \( Y \) of length \( n \) can be represented as a sequence of pairs as follows:

\[
Y = (m_1, E_1), \ldots, (m_n, E_n) \text{ such that } \forall i \in [1, n]. E_{i-1} \rightarrow m_i \rightarrow E_i.
\]

When \( n=\infty \) \( Y \) becomes an infinite derivation. In what follows, we assume that the initial term is \( E_0 \) and \( D \) represents the set of all (finite and infinite) derivations from \( E_0 \). Furthermore, the terms "derivation" and "execution sequence" are used interchangably.

Let \( Y \) be a derivation in \( D \), and \( Y(i) \) denote the \( i \)-th element, i.e. \( (m_i, E_i) \), of \( Y \). We give the following definition of the mapping \( \xi \), which forms the basis of our distance \( dt \).
Definition 2.1 The mapping $\xi$ is defined as a mapping from a generic derivation $Y = \{(m_i, E_i), 0 \leq i \leq n\}$ into an abstract derivation $Z = \{(a_i, \alpha_i), 0 \leq i \leq k\}$, as follows:

$$
\begin{align*}
a_1 &= Y(1) = \ldots = Y(\alpha_1) \neq Y(\alpha_1 + 1) \\
a_2 &= Y(\alpha_1 + 1) = \ldots = Y(\alpha_1 + \alpha_2) \neq Y(\alpha_1 + \alpha_2 + 1) \\
&\vdots \\
a_k &= Y(\sum_{j=1}^{k-1} \alpha_j + 1) = \ldots = Y(\sum_{j=1}^{k} \alpha_j) \neq Y(\sum_{j=1}^{k} \alpha_j + 1), k = 3, 4, \ldots,
\end{align*}
$$

where $a_k = (m_k, E_k), m_k \in \text{Act}, E_k \in \text{P}, \alpha_k \in \mathbb{N}$.

We make the following exception:

If for some $k$,

$$
Y(\sum_{j=1}^{k} \alpha_j + 1) = Y(i) \text{ for all } i \geq \sum_{j=1}^{k} \alpha_j + 1,
$$

then we set

$$
\begin{align*}
a_n &= Y(\sum_{j=1}^{k} \alpha_j + 1), \text{ and } a_n = 1 \text{ for all } n \geq k + 1.
\end{align*}
$$

This mapping $\xi$ of the space of generic derivations onto the space of abstract derivations allows a more concise representation of derivations. For simplicity, we shall write an abstract derivation in short form as $Z = \{(m_i, \alpha_i), 0 \leq i \leq k\}$. Furthermore, without confusion, we shall refer to $D$ as the space of both generic and abstract derivations.

In this mapping, individual derivations are viewed in a two-dimensional space, $(a_k, \alpha_k)$: one dimension, $a_k$, is represented by an element of the sequence $Y$ which we shall refer to as a (recursive) call to a process, and the remaining one, $\alpha_k$, the level of recursion for each of the (recursive) calls that constitute a particular behaviour. We take that a single (nonrecursive) call to a process has a level of recursion equal to 1. These pairs are then ordered in time by their relative position in the derivation; $\infty$ is used to characterize an infinite derivation. This definition forms the basis of our distance $\text{dt}$ defined in the next subsection, and sets the basic characteristics of the test coverage expressible within our metrics.

Consider the three execution trees in Figure 1 which are extracted from a protocol specification given in Appendix 1. Each execution tree of Figures 1b and 1c shows three execution sequences A, B and C.
Figure 1: Execution sequences of a communication protocol given in Appendix 3

Suppose that (A,B,C) in each of Figures 1b and 1c are all valid test sequences generated by some test generation system. If two out of these three test cases are to be selected for execution, a random (probabilistic) test selection strategy would have an equally good chance of selecting (A,B), (A,C) or (B,C). This is obviously not an acceptable approach. On the other hand, applying some metric models researched earlier for process algebraic spaces ([dBJ82, Cos84]), one could be led to always give preference to test sequences of the type (A,B), which is clearly a poor selection from the testing point of view. For our metric dt, we would like to define it in such a way as to distinguish the relative value of test sequences, which would allow the above choice if desired, as well as preferring (A,C) or (B,C) over (A,B) in (A,B,C) (see Appendix 2 for the calculations of dt(A,B), dt(A,C) and dt(B,C)). Since the test sequences A and B are very close (similar), the selection of (A,C) or (B,C) instead of (A,B) would be a natural choice according to most test selection criteria (e.g. distinct-paths criterion as well as switch-cover paradigm).

We now define $\delta_k(A, B)$, the measure of the difference in the level of recursion between two derivations $A$ and $B$ of length $K$ and $L$, respectively.
Definition 2.2

\[ \delta_k(A, B) = \begin{cases} 
0 & \text{if } a_k = b_k \text{ and } \alpha_k = \beta_k \\
|\alpha_k - \beta_k| & \text{if } a_k = b_k \text{ and } \alpha_k \neq \beta_k \\
\infty & \text{if } a_k \neq b_k 
\end{cases} \]

Here, \( k = 1, 2, \ldots, \min\{K, L\} \), and we take \( \delta_k(A, B) = \infty \), for \( \min\{K, L\} < k \leq \max\{K, L\} \).

Clearly,

\[ \delta_k(A, B) \in \mathbb{N} \cup \{\infty\}. \]

**Definition 2.3** Let \( A \) and \( B \) be two (finite or infinite) derivations in \( D \). In abstract representation, \( A = \{(a_k, \alpha_k)\}_{k=1}^{L}, \) and \( B = \{(b_k, \beta_k)\}_{k=1}^{M}, L, M \in \mathbb{N} \cup \{\infty\}. \) The testing distance \( d_t \) between any two derivations \( A, B \) in \( D \) is defined as follows:

\[ d_t(A, B) = \max\{K, L\} \sum_{k=1}^{\max\{K, L\}} p_k r(\delta_k(A, B)) \]

where \( p_k \) represents the weight of the individual calls reflecting the weight of the computational pattern and \( r(\delta_k(A, B)) \) represents the weight of the level of recursion (the number of consecutive invocations of the same process) of a (recursive) call within the derivations.

In order for \( d_t \) to be a distance in \( D \), we introduce the following constraints for \( p_k \) and \( r(\delta_k(A, B)) \).

**Constraint 2.1** \( \{p_k\}_{k=1}^{\infty} \) is a sequence of positive numbers such that

\[ \sum_{k=1}^{\infty} p_k \]

converges.

**Example 2.1**

\[ p_k = \frac{1}{2^k} \]

If the sequence \( \{p_k\}_{k=1}^{\infty} \) is defined in this manner, the weight of the individual calls decreases exponentially with respect to their distance from the initial term \( E_0 \).
Example 2.2

\[ p_k = \begin{cases} 
1 & \text{for } k = 1..m \\
\frac{1}{2^{k-m}} & \text{for } k = m..\infty
\end{cases} \]

In this case, all individual calls are weighted equally up to the m-th call, after which the weight of each consecutive call decreases exponentially.

Constraint 2.2 \( \{r(k)\}_{k=0}^{\infty} \) is a nondecreasing sequence in \([0,1]\) such that \( r(0) = 0 \) and \( \lim_{k \to \infty} r(j) = 1(= r_{\infty}) \). Furthermore, the sequence \( \{\frac{r(k)}{k}\}_{k=1}^{\infty} \) is nonincreasing.

The last constraint gives rise to the following lemma which will be used in Theorem 2.1.

**Lemma 2.1** The sequence \( \{r(k)\}_{k=0}^{\infty} \) has the following important properties:

\[ r(j+i) \leq r(i) + r(j), i,j=1,2,\ldots \]

**Proof.**

Suppose

\[ 0 \leq i \leq j \]

Then

\[ r(i) \leq r(j) \]

and

\[ \frac{r(i)}{i} \geq \frac{r(j)}{j} \]

and

\[ \frac{r(i+j)}{i+j} \leq \frac{r(j)}{j} \]

Therefore

\[ r(i+j) \leq \frac{i+j}{j} r(j) = r(j) + \frac{i}{j} r(j) \leq r(j) + \frac{i r(i)}{i} = r(j) + r(i) \]

Example 2.3

\[ r(k) = \frac{ck}{ck+1}, c > 0 \]
Figure 2: Some representative examples of the function $r(k)$

The graph of this function for the cases where $c=1$, $c=.1$ and some values in between these extreme points, is sketched in Figure 2.

This is one example of a suitable sequence satisfying Constraint 2.2, which allows us a great deal of flexibility when testing. The larger $c$ is, the more weight is given to each recursion. With this $r(k)$, we can express the marginal benefit obtained by testing one more recursive call of the same event versus testing a call to a different process, by suitably setting the constant $c$.

**Theorem 2.1**

$dt$ defined in previous definition is a distance in $D$.

**Proof.**

1. $dt(A,B)$ is a non-negative real number for each $A,B$.

   Indeed, since
   
   $$0 \leq r(\delta_k(A,B)) \leq 1$$
   
   for all $k=1,2,\ldots,\max\{K,L\}$,

   we have
   
   $$0 \leq \sum_{k=1}^{n} p_k r(\delta_k(A,B)) \leq \sum_{k=1}^{n} p_k \leq \sum_{k=1}^{\infty} p_k$$
   
   for all $n \in N$.

   Thus,
   
   $$\sum_{k=1}^{\infty} p_k r(\delta_k(A,B))$$
converges and
\[ dt(A, B) = \sum_{k=1}^{\infty} p_k r(\delta_k(A, B)) \geq 0 \]

2. \( dt(A, B) = 0 \) if and only if \( A = B \).

Indeed, \( dt(A, B) = 0 \) iff \( \delta_k(A, B) = 0 \) for all \( k=1, 2, \ldots, \max\{K, L\} \). Clearly, \( \delta_k(A, B) = 0 \) for all \( k=1, 2, \ldots, \max\{K, L\} \) is equivalent to \( K=L \) and \( a_k = b_k \) and \( \alpha_k = \beta_k \) for all \( k=1, 2, \ldots, K=L \), what gives \( A=B \).

3. \( dt(A, B) = dt(B, A) \)

It follows from the definition of \( \delta_k(A, B) \), \( k=1, 2, \ldots, \max\{K, L\} \), that
\[ \delta_k(A, B) = \delta_k(B, A), \quad k = 1, 2, \ldots, \max\{K, L\}. \]
Therefore,
\[ r(\delta_k(A, B)) = r(\delta_k(B, A)) \]
and consequently
\[ dt(A, B) = \sum_{k=1}^{\infty} p_k r(\delta_k(A, B)) = \sum_{k=1}^{\infty} p_k r(\delta_k(B, A)) = dt(B, A) \]

4. The only non-trivial part in the proof of this theorem is to show that the triangular inequality holds for \( dt \).

Let \( A, B, C \) be three (finite or infinite) derivations in \( D \). Then
\[ \delta_k(A, C) + \delta_k(C, B) \geq \delta_k(A, B), \quad k=1, 2, \ldots, \max\{K, L, M\}. \]

Since the sequence \( \{r(j)\}_{j=0}^{\infty} \) is non-decreasing, we have
\[ r(\delta_k(A, C) + \delta_k(C, B)) \geq r(\delta_k(A, B)), \quad k=1, 2, \ldots, \max\{K, L, M\}. \]

It follows from the Lemma 2.1 that
\[ r(\delta_k(A, C)) + r(\delta_k(C, B)) \geq r(\delta_k(A, C) + \delta_k(C, B)), \quad k=1, 2, \ldots, M. \]
Therefore,
\[ r(\delta_k(A, C)) + r(\delta_k(C, B)) \geq r(\delta_k(A, B)), \quad k=1, 2, \ldots, \max\{K, L, M\}. \]
Consequently,

\[
\begin{align*}
\text{dt}(A, C) + \text{dt}(C, B) &= \sum_{k=1}^{\max(K,L)} p_k \Delta_r(\delta_k(A, C)) + \sum_{k=1}^{\max(L,M)} p_k \Delta_r(\delta_k(C, B)) \\
&\geq \sum_{k=1}^{\max(L,M)} p_k \Delta_r(\delta_k(C, B)) = \text{dt}(A, B)
\end{align*}
\]

Hence

\[
\text{dt}(A, C) + \text{dt}(C, B) \geq \text{dt}(A, B)
\]

and the triangular inequality indeed holds.

The metrics defined in the previous section gives freedom to test designers to tune the "parameters" of the metric \( \delta_k(A, B) \) and \( p_k \), essentially weighing the impacts of recursive execution of identical patterns of the primitives of communication versus the occurrence of new patterns of communication behaviour. Going back to the previous example, it is easily verified that by defining suitable sequences \( p_k \) and \( r(\delta_k) \), we obtain distances between these potential test cases, which would favour the testing strategy we choose.

Additionally, the following theorem expresses a further characteristic of this metrics which is both mathematically pleasing as well as supporting our intuitive view from the testing standpoint.

**Definition 2.4** The metric space is complete if every Cauchy sequence in it is convergent.

**Theorem 2.2** The metric space \((D, \text{dt})\) is complete.

**Proof of the Completeness Theorem**

Let \( A_n = \{a_j^{(n)}\}_{j=1}^\infty \), be a Cauchy sequence of execution sequences in the space \( E \) of a specification of a protocol behaviour. Then \( \forall \varepsilon > 0 \exists N_\varepsilon \) such that

\[
\forall n,m \in \mathbb{N}, n,m > N_\varepsilon \implies \text{dt}(A_n, A_m) < \varepsilon.
\]

Let \( k \) be arbitrary and choose \( \varepsilon \) such that \( 0 < \varepsilon < p_k \Delta_r(1) \). Then for all \( n,m > N_\varepsilon \) we have

\[
p_k \Delta_r(\delta_k(A_n, A_m)) \leq \text{dt}(A_n, A_m) < \varepsilon < p_k \Delta_r(1).
\]

It follows from the last relation that \( r(\delta_k(A_n, A_m)) < r(1) \forall n,m > N_\varepsilon \). Since \( r \) is increasing the last relation implies that \( \delta_k(A_n, A_m) = 0 \) for all \( n,m > N_\varepsilon \), or
equivalently,

\[ a_k^{(n)} = a_k^{(m)}; \text{ and } \alpha_k^{(n)} = \alpha_k^{(m)} \]

for all \( n, m > N_\varepsilon \). Thus each of the sequences \( \{a_k^{(n)}\}_{n=1}^\infty \), \( \{\alpha_k^{(n)}\}_{n=1}^\infty \), is eventually constant. Denote by \( a_k \) and \( \alpha_k \) the corresponding constants. Since \( k \in \mathbb{N} \) was arbitrary, in this way we get the sequence \( A = \{(a_k, \alpha_k)\}_{k=1}^\infty \). We now prove that \( \{A_n\}_{n=1}^\infty \) converges to \( A \) with respect to \( dt \).

At this point we must assume that the sequence \( \{p_k\}_{k=1}^\infty \) is non-increasing. Since \( p_j \leq p_k \) for all \( j \leq k \), the same reasoning yields \( a_j^{(n)} = a_j^{(m)} \) and \( \alpha_j^{(n)} = \alpha_j^{(m)} \) for all \( m, n > N_\varepsilon \) and all \( j \in \mathbb{N}, 1 \leq j \leq k \). It now is implied by the definition of \( A \) that for all \( n > N_\varepsilon \)

\[ d(A, A_n) = \sum_{i=k+1}^{\infty} p_i r(\delta_i(A, A_n)) \leq \sum_{i=k+1}^{\infty} p_i < \varepsilon \]

Furthermore, it is obvious from the construction of the limiting sequence, that all its subsequences are in the original space since they are subsequences of some other finite execution sequence in that space. From the rules of the process algebra and the above proof it then follows that each limiting sequence is in the space of execution sequences of the original specification.

This completes the proof that every Cauchy sequence in the space \((E, dt)\) is convergent with the limit in that same space. Therefore the space of all execution sequences of a protocol specification \( E \) supplied with the metric conforming with the constraints of the metric \( dt \) is complete.

**Example 2.4** Figure 3 shows two execution trees of a LOTOS specification of the ISO transport service given in Appendix 3. The example is an excerpt from [Sco87].

The two execution trees shown represent a TConnection specification. In this example, we will allow parallelism, up to 4 concurrent TConnections, and recursion up to four instantiations of processes TCEPDataTransfer, as shown in Figure 3. The complexity of representing such an example dictates that we freely use the functions Subsort and boolean functions on Transport Service Primitives, e.g. IsTDT or IsTDIS, as specified in the document, instead of labelling the branches of the trees with actual events. When appropriate mappings are performed the branches become labelled with appropriate events. Notice that the complexity of execution in this unrealistically simple case yields an order of \( 2^{12} \) execution se-
Figure 3: Two processes from LOTOS specification of a Transport Service
quences, taking into account only the "inner body" of the interleaved tree. Now, we turn our attention to some extreme cases (A, B, C, D, E, F), as well as a typical case of execution sequences (G).

1. $A = \{(\text{TCONreq},4), (\text{IsTDT},11), (\text{IsTDIS},4)\}$
2. $B = \{(\text{TCONreq},2), (\text{TCONind},2), (\text{IsTDT},11), (\text{IsTDIS},4)\}$
3. $C = \{(\text{TCONreq},1), (\text{TCONind},3), (\text{IsTDT},11), (\text{IsTDIS},4)\}$
4. $D = \{(\text{TCONreq},1), (\text{IsTDT},3), (\text{IsTDIS},1)\}$
5. $E = \{(\text{TCONind},1), (\text{IsTDT},3), (\text{IsTDIS},1)\}$
6. $F = \{(\text{TCONreq},1), (\text{IsTDIS},1)\}$
7. $G = \{(\text{TCONreq},2), (\text{IsTDT},3), (\text{IsTDIS},1), (\text{IsTDIS},1), (\text{TCONind},1), (\text{IsTDT},3), (\text{IsTDIS},2)\}$

We evaluate some distances within this execution sequence space using the metric $dt$ defined in Definition 2.3 with the following parameters:

$$p_k = \begin{cases} 4 & \text{for } k = 1, \ldots, 7 \\ 1/2^{k-5} & \text{for } k = 8, \ldots, \infty \end{cases}$$

and

$$r(k) = ck/c(k+1), c = 1$$
Note that this metric treats the difference in 1 pattern as being of distance 4 and the difference in 1 level of recursion as being of distance 2 (for the first 7 patterns). Thus, the following sequences are considered as being at the same distance:

1. those sequences (e.g. D,E) which differ in 1 complete pattern and
2. those (e.g. B,C) which differ in 2 levels of recursion in total, over same pattern of behaviour.

Some representative results are summarized in Figure 4.

<table>
<thead>
<tr>
<th>Computation x</th>
<th>Computation y</th>
<th>Difference characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>in patterns</td>
<td>in recursion</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
<td>3</td>
</tr>
<tr>
<td>F</td>
<td>G</td>
<td>6</td>
</tr>
<tr>
<td>D</td>
<td>G</td>
<td>4</td>
</tr>
</tbody>
</table>

* inf denotes the difference of recursion when the respective patterns are different

Figure 4: Some results of application of metric dt on given execution sequences

3 Control Coverage Metrics

Our definition of the control coverage measure is based on the distance between a subset (i.e. a test suite) and a set of execution sequences (i.e. the super test suite derived from the specification). To define the distance of a subset from a set, we must first define the distance of a point (i.e. an execution sequence) from a set (of execution sequences).
Definition 3.1 Let \( L \) be a subset of a metric space of all execution sequences \( E \). Distance of a point \( x \in E \) from \( L \) is defined as

\[
d(x, L) = \inf \{d(x, y) | y \in L\}
\]

3.1 A Coverage Metrics for Pseudo-Exhaustive Testing Strategies

General acceptance testing, as well as final system integration testing, and some kinds of availability testing strategies, will produce a final verdict to convey the quality of the implementation system in terms of how well it performs (and conforms) according to the entire specification. This is of course impossible to judge precisely since the domain of execution sequences is infinite in general. In terms of an objective measure, however, the following estimate can be determined through the metric \( dt \).

To measure the quality of coverage of a test suite \( T \) with respect to a (super) set of execution sequences of a specification \( S \), we take the supremum of the average distances from \( T \) to the complement of \( T \) with respect to \( S \) denoted as \( S \setminus T \).

Definition 3.2 Let \( \{(a_k, \alpha_k)\}_{k=1}^L \) be an execution sequence in \( S \), and let \( n \) be a natural number. A truncation of length \( n \) of this sequence is defined as follows: If \( L > n \), then the truncation is the sequence \( \{(a_k, \alpha_k)\}_{k=1}^L \), and if \( L \geq n \), then the truncation is the sequence itself, \( \{(a_k, \alpha_k)\}_{k=1}^L \). Denote the set of all truncations of length \( n \) by \( S_n \). Note that \( S_n \) is taken to be finite. Also note that it is not necessarily true that \( S_n \subset S_{n+1} \). (Figure 5)

Let \( T \) represent the set of test sequences. We assume that for any \( T \) there exists a natural number \( n_T \) such that \( T \subset S_{n_T} \), (i.e. we assume that there always exists the longest test sequence).

Let \( k_n \) be the number of elements of \( S_n \setminus T \). For a natural number \( n, n \geq n_T \), we define \( v_n \) to be an average distance from \( T \) to \( S_n \setminus T \), i.e.

\[
v_n = \frac{\sum\{dt(x, T), x \in S_n \setminus T\}}{k_n}
\]

As a measure of how well \( T \) approximates \( S \), we will take the normalized supremum of \( v_n \).
Definition 3.3 \( \text{CovAve}(T) = 1 - c(T) \).

Remark 3.1 1. If \( T_1 \subseteq T_2 \) then \( \text{CovAve}(T_1) \leq \text{CovAve}(T_2) \).
2. \( 0 \leq \text{CovAve}(T) \leq 1 \)
3. \( \text{CovAve}(S_n) \to 1 \) as \( n \to \infty \). In other words, as the test suite represents a larger subset of the original specification, its coverage approaches 1.
4. Coverage of an empty suite is 0 by definition.
3.2 A Coverage Metrics for Strategies With Pre-defined Target Test Sequences

If a certain subset of a specification is pre-selected as a focal point of the testing campaign (as in safety-critical testing, test-purpose driven testing, estimated-usage profile testing, etc.), then the knowledge of specific execution sequences that need to be approximated in the original specification may be available. In these cases, we are interested in knowing how well a given test suite performs in the worst case.

Therefore, to measure the quality of coverage of a test suite \( T \) with respect to a set of execution sequences of a specification \( S \), we use the maximum distance from \( T \) to \( S \setminus T \), the complement of \( T \) with respect to \( S \) according to the following definition.

**Definition 3.4** The normalized maximum distance from a given test suite \( T \) to \( S \), the set of all execution sequences of a specification, is defined as

\[
m(T) = \frac{\sup \{ d(x, T) : x \in S \setminus T \} }{\sum_{k=1}^{\infty} p_k}
\]

**Definition 3.5** The coverage of \( T \) w.r.t. \( S \) is defined as \( \text{CovMax}(T) = 1 - m(T) \).

**Remark 3.2** The properties of \( \text{CovAve} \) listed in Remark 3.1 in the previous subsection also apply to \( \text{CovMax} \).

4 Metric-Based Test Selection for Communication Protocols

Test selection is a recognized problem within protocol testing community. There have been some research efforts on fault models [vBAD+91] and on test selection based on external knowledge [BTV92], as well as many activities on test sequence generation based on FSM model of protocols [SL89]. However, the approach based on fault models and external knowledge is so far limited by the lack of empirical understanding (or expert knowledge) of faults and their relationships to test cases. In [BTV92], Brinksma et al. propose an interesting framework for test selection. This framework however works from the principle that test selection must be made on the basis of external knowledge, such as experience and heuristics, that cannot be reflected in the formal specification of protocols.
Our approach is different. We focus on making use, as much as possible, of the generic knowledge derived from the formal specification and existing theory in introducing a sound, effective (metric based) test selection method. This test selection method has a number of (metric definition) parameters which may be "tuned" according to external knowledge. But this should be viewed as a flexibility rather than a necessity since the test selection method possesses some desired properties (addressing the problems mentioned above) irrespective of the parameter tuning.

Our metrics as introduced earlier, due to the well defined constraints, gives rise to a convergent test selection process where the more test sequences are selected the closer the selected set tends to the original set, i.e. there are no relevant, peculiar ("singular") test cases or groups of test cases that would be completely missed out in the selection process (this is a critical, common error in the manual ad hoc test selection process). The ability to handle infinite sets of infinite test sequences is a unique feature of our metric based test selection method since other approaches deal mainly with the finite cases [BTV92, vBAD+91]. Furthermore, the metric based test selection method can be automated and run simultaneously with any random or exhaustive test generation process to eliminate the need for storing and accessing a large number of test cases generated.

4.1 General Metric Based Test Selection Algorithm

We first introduce a number of basic concepts which are essential for the construction of the selection algorithm.

Let \((E, d_E)\) be a metric space and \(A \subseteq E\).

**Definition 4.1** A family \(F\) of subsets of \(E\) is said to be a covering for \(A\) if \(A \subseteq \bigcup\{F : F \in F\}\), i.e. if \(A\) is contained in a union of all sets in \(F\).

**Definition 4.2** Let \(S \subseteq E\) and \(\epsilon > 0\). A set \(S\) is said to be \(\epsilon\)-dense in \(A\) if for every \(x \in A\) there is \(y \in S\) such that \(d_E(x,y) < \epsilon\).

In other words, \(S\) is \(\epsilon\)-dense in \(A\) if the family of all open balls with centers at the points in \(S\) and with radius \(\epsilon\) forms a covering for \(A\).

We now present the general metric based test selection algorithm. This algorithm is a multi-pass process where each pass is realized by the selection function `SELECT(T_0, G, \epsilon, C)` which returns a selected set \(T\), being an \(\epsilon\) cover of the original set \(S\) of test cases, such that the cost of \(T\) (which includes the initially
To) is less than some given threshold cost $C$. Figure 6 illustrates an $\epsilon$-dense cover which is generated after each pass of the algorithm.

![Figure 6: $\epsilon$-dense covering of the original set $G$](image)

The cost function of a test case may be defined so as to represent resources (time and space) needed to execute that test case, e.g. its length. The cost of a set of test cases may be defined simply as the sum of the cost of individual test cases in the set.

**Metric Based Test Selection Algorithm**

S: 1. Initially, the selected set $T$ is empty, $G$ is the given (generated) set of test cases, $\epsilon$ is the initial target distance and $C$ is the given cost threshold for the selected set

Step 2. While $\text{Cost}(T) < C$ do $T = \text{SELECT}(T, G, \epsilon = \epsilon/k, C)$ for some scaling factor $k$ of $\epsilon$, applied at each iteration (i.e. each pass)

Step 3. Stop. No further pass is possible since any further test case added to the set $T$ of test cases selected so far would violate the cost constraint
where the function \( \text{SELECT}(T, G, \epsilon, C) \) (i.e. each pass in Step 2 of the above general algorithm) can be specified as follows:

Step 2.1 Let \( X = G \setminus T \), i.e. \( G \) excluding \( T \).

Step 2.2 If \( X \) is empty then return \( T \) and exit

Else remove (randomly) some test case \( t \) from \( X \).

Step 2.3 If \( d(t,T) < \epsilon \) goto Step 2.2

Step 2.4 If \( \text{Cost}(T \cup t) < C \) then add \( t \) to the selected set \( T \)

Step 2.5 Goto Step 2.2

Since the target distance \( \epsilon \) is decreasing with each pass in the algorithm, the multipass algorithm generates Cauchy sequences, each being formed by test cases selected one by one (one in each pass) over the successive passes. This process of Cauchy sequence generation is illustrated by Figure 8 for the simple example in the next subsection. These Cauchy sequences converge to limits which are themselves in the original set of test cases. Furthermore, any infinite test case in the original set can also be approximated by some converging sequence of selected test cases. (These interesting properties are proven in the next section). Thus, the algorithm performs test selection as a convergent approximation process, which ensures there are no test cases or regions of test cases which are simply overlooked.

It is also worth noting that in this algorithm we use a greedy heuristic in selecting the next test case \( t \) (Step 2.2). In fact, we simply select the first randomly encountered test case which is sufficiently far (greater than \( \epsilon \)) from all already selected test cases (Step 2.3), and which satisfies the cost constraint (Step 2.4). Ideally, we should compute the distances between each of the possible test case candidates (in \( S \setminus T \)) and all selected test cases (in \( T \)), and based on these distances select the ”best” test case with minimum cost. Obviously, such an optimal algorithm would be np-complete and the problem of finding a suboptimal, computationally tractable heuristic is a nontrivial research question.

Another issue is how to select the initial target distance \( \epsilon \) and the scaling factor \( k \) which may take different values for different passes. Inappropriate selection of these values can lead to either excessive amount of computing (with many passes) (if \( \epsilon \) is too large and \( k \) is too small, e.g. close to 1), or too coarse a selection (if \( \epsilon \) and \( k \) are too large). The proper values of \( \epsilon \) and \( k \) would depend on the metric.
definition parameters and the protocol specification (which gives the distribution of the original test cases in the metric space) as well as the cost function and the cost threshold defined.

This multipass algorithm eventually terminates if and only if the initial set of test sequences $G$ is finite. $G$ can be generated from a random or exhaustive test generator, successively filling the execution space of a protocol behaviour more and more densely, thus in the limit approaching the space of all execution sequences of the protocol specification (by the Completeness Theorem 10.2 and Theorem 12.1). In fact, the test selection can be carried simultaneously with the test generation process so as to eliminate the need for storing a large number of test cases generated. In this case, every test case generated would be considered as a candidate for selection and is selected only if the distance and cost constraints are satisfied.

After the algorithm terminates, the coverage, CovMax, of the selected test set $T_S$ w.r.t. to $G$ can be calculated. It can be related to the achieved target density $\epsilon$, by observing that

$$\sup\{d(x, T_S) : x \in S \setminus T_S\} \leq \epsilon$$

Therefore, according to the Definition 11.5,

$$m(T_S) = \frac{\sup\{d(x, T_S) : x \in S \setminus T_S\}}{\sum_{k=1}^{\infty} p_k} < \frac{\epsilon}{\sum_{k=1}^{\infty} p_k}$$

Thus,

$$CovMax(T_S) = 1 - m(T_S) > \frac{\epsilon}{\sum_{k=1}^{\infty} p_k}$$

where $p_k$ is the appropriate parameter for the distance $d_t$ used in the selection process.

Since the cost function and distance are defined based on the current test sequence and test sequences already included in the selected set, no additional bulk of data needs to be carried around or evaluated during the selection process. Also, all decisions made during the selection process are based on the protocol control behavior inherent in execution sequences and are independent of the internal structure of the formal specification. This distinguishes this metric and its applications from most other metric based methods in which this relatively complex information is required. We should also note that if cost (for storing and executing test suites) was not a constraint, the selection problem would become a one-pass process where Step 2.4 in the algorithm would be skipped and the goal would be to select test cases so as to achieve certain specified target distance (coverage).
4.2 Example of $\epsilon$-Dense Test Selection

In this subsection, we present a simple example to illustrate the idea of the $\epsilon$-dense test selection algorithm.

Example 4.1 Let a sequence

\[ s_s = \{(\text{ConReq}, \alpha_p), (\text{ConConf}, \alpha_q), (\text{ConReq}, \alpha_s), (\text{AnyOtherPrimitive}, \alpha_r)\} \]

denote the excerpt specification of execution sequences of a connection establishment phase of a typical communication protocol. Assuming the number of possible simultaneous connections is 128, we will consider in this example only those sequences which have the specified pattern of behaviour, and which are bound by the following constraints:

\[
0 < \alpha_p \leq 128, 0 \leq \alpha_q \leq 128, 0 \leq \alpha_s \leq 128 \]

\[
\alpha_q \leq \alpha_p, \alpha_s = 128 - \alpha_p \]

Let the metric $d_t$ be defined by the following parameters:

\[
p_k = 1, k = 1, 2, 3, p_k = 0.01/2^k, k > 3 \]

\[
r(\delta_k) = \frac{c\delta_k}{c\delta_k + 1}, c = 0.01 \]

The graph of $r(\delta_k)$ is shown in Figure 7.

Assume the cost associated with every test sequence is linear as follows:

\[
c(t_s) = \alpha_p + \alpha_q + \alpha_s \]

Let the threshold cost be $C=1300$.

Assuming test cases of the pattern specified are generated by a random test generator for the selection process, the following test selection may be made after three passes with the different target distances shown:

Pass 1:

\[
\epsilon_1 = 0.55 * 3 \]

Selected test sequences:

\[
t_{s1} = \{(\text{ConReq}, 1), (\text{ConConf}, 1), (\text{ConReq}, 127), (\text{AnyOtherPrimitive}, \alpha_r)\} \]
Pass 2:

\( \epsilon_2 = 0.55 \times 2 = 1.10 \)

Selected test sequences:

\[ t_{s2} = \{(\text{ConReq}, 127), (\text{ConConf}, 1), (\text{ConReq}, 1), (\text{AnyOtherPrimitive}, \alpha_r)\} \]

\[ t_{s3} = \{(\text{ConReq}, 70), (\text{ConConf}, 61), (\text{ConReq}, 58), (\text{AnyOtherPrimitive}, \alpha_r)\} \]

Note 1: These test sequences are of distance greater than \( \epsilon_2 \) from the test sequence selected in the first pass. In fact,

\[ d_t(t_{s1}, t_{s2}) = 1 \times r(69) + 1 \times r(60) + 1 \times r(69) = 2 \times 0.408 + 0.375 = 1.191. \]

\[ d_t(t_{s1}, t_{s2}) = 1 \times r(126) + 0 \times r(126) = 2 \times 0.557 = 1.114. \]

Note 2: Together with the previously selected test sequence \( t_{s1} \), these test sequences form an \( \epsilon_2 \)-dense cover of the entire given execution sequence set. Since the cost of all selected test sequences is \( 129 + 129 + 189 = 447 \), which is less than the threshold cost \( C = 1300 \), we go to pass 3.

Pass 3:

\( \epsilon_3 = 0.50 \)
Selected test sequences:

\[ t_{a4} = \{(\text{ConReq},127),(\text{ConConf},127),(\text{ConReq},1),(\text{AnyOtherPrimitive},a_r)\} \]

\[ t_{a5} = \{(\text{ConReq},114),(\text{ConConf},87),(\text{ConReq},14),(\text{AnyOtherPrimitive},a_r)\} \]

\[ t_{a6} = \{(\text{ConReq},100),(\text{ConConf},47),(\text{ConReq},28),(\text{AnyOtherPrimitive},a_r)\} \]

\[ t_{a7} = \{(\text{ConReq},57),(\text{ConConf},21),(\text{ConReq},71),(\text{AnyOtherPrimitive},a_r)\} \]

Note 1: The cost of all seven selected test sequences can be easily calculated to be 1241. Since any further test sequence selected (costing at least 128) would violate the cost constraint, the algorithm terminates.

Note 2: The coverage of the selected set of test sequence can be estimated via the target density figures \( \epsilon_3 \) as follows:

\[
m(T_S) < \frac{\epsilon_3}{\sum_{k=1}^{\infty} p_k} = 0.166
\]

and so the estimated coverage of this test suite is:

\[
\text{CovMax}(T_S) \geq 1-0.166 = 0.834.
\]

This shows we have performed quite a good selection within the cost constraint.

Note 3: The Cauchy sequences generated over three passes can be given by:

\[ \{t_1, (t_2 \text{ or } t_3), (t_4 \text{ or } t_5 \text{ or } t_6 \text{ or } t_7)\} \]

We can easily verify these Cauchy sequences by noting that their elements (test cases) get progressively closer over the three successive passes.

We note in passing that in an actual implementation, it is not necessary to perform the distance evaluation with floating point numbers. By scaling the value 1.0 to a large integer, all computations can be done in the integer mode. We also note that the test selection algorithm works in general with any distance definitions, and it is not just restricted to the \( \text{dt} \) distance heuristic. We have suggested this heuristic due to its intuitiveness and simplicity and its relationship with other well researched topics in testing theory, as discussed in the rest of this section.

4.3 Test Selection as a Convergent Approximating Process

In this subsection, we present and prove a number of interesting properties of our test selection method which is based on testing distance \( \text{dt} \). These properties
are mathematically pleasing as well as supporting our intuitive understanding on testing.

**Proposition 4.1** Let $\epsilon$ denote the target density (distance) of the test selection algorithm, and assume the cost threshold is unlimited, i.e. cost is not a constraint. If the input set, $G$, of test cases (which may be generated by a random test generator) for this algorithm is $\epsilon_G$-dense in the original specification, then the algorithm will produce an $\epsilon_{sel}$-dense cover of the original specification, where

$$\epsilon_{sel} = \epsilon_G + \epsilon.$$

**Proof.** The proof follows immediately from the selection algorithm and the triangular inequality property of a distance (in Theorem 10.1).

**Consequence 12.1.** The algorithm is able to select a test suite $T_S$ whose density ($\epsilon_{sel}$) in the original specification can be made arbitrarily close to the density ($\epsilon_G$) of the input (generated) set $G$ of test cases with respect to the original specification. (For example, if $\epsilon = 1/n\epsilon_G$, then $\epsilon_{sel} = \epsilon_G + 1/n\epsilon_G$, and so, $\epsilon_{sel} \rightarrow \epsilon_G$ as $n \rightarrow \infty$).
The relevance of the above proposition and consequence w.r.t. the test selection algorithm is twofold: (i) if \( G \) is a finite, but very large set of test sequences, then the test selection algorithm will serve as a test suite reduction strategy, performing "uniform" reductions of \( G \) within its complete execution space, to match various predefined cost thresholds and target distances (densities), and (ii) if \( G \) is infinite, the test selection algorithm may be used to produce a finite test suite \( T_S \) to match any predefined approximation of the starting set \( G \) of test cases which may be generated by some random or exhaustive test generator. In this case, we must adopt some different termination criteria for each pass (Step 2.2) of the algorithm, e.g. via some time threshold.

**Theorem 4.1** Every (infinite) sequence in the space of execution sequences of a protocol is a limiting sequence of some Cauchy sequence in the same space, when this space is viewed as a metric space \((E, dt)\).

**Proof:** Let \( A=\{(a_k, \alpha_k)\}_{k=1}^{\infty} \) be an infinite sequence. For \( n \in \mathbb{N} \), by \( A_n^* \) we denote the finite sequence \( \{(a_k, \alpha_k)\}_{k=1}^{n} \) (i.e. a truncation of A). This sequence is a Cauchy sequence (by the distance definition, Definition 10.6), and is convergent (by the Completeness Theorem, Theorem 10.2). Moreover, we can prove the sequence \( \{A_n^*\}_{n=1}^{\infty} \) converges to \( A \). In fact, we have \( dt(A_n, A) = \sum_{k=n+1}^{\infty} p_k \). Since the series \( \sum_{k=n+1}^{\infty} p_k \) converges, for every \( \epsilon > 0 \), \( \exists N_\epsilon \) such that \( \sum_{k=n+1}^{\infty} p_k < \epsilon \) for all \( n > N_\epsilon \). Thus \( \forall \epsilon > 0 \), \( \exists N_\epsilon \) such that \( dt(A_n, A) < \epsilon \) for all \( n > N_\epsilon \). This proves that \( \{A_n^*\} \) converges to \( A \).

**Consequence 12.2.** The test selection algorithm (through successive passes) generates Cauchy sequences of the test sequences in \( T_s \) with the following important properties:

1) The generated Cauchy sequences of \( T_S \) are convergent by the Completeness Theorem (Theorem 10.2)

2) Limits of Cauchy sequences of \( T_S \) are themselves in the original space of execution sequences of the protocol specification

3) Every (infinite) sequence in the original (input) set of execution sequences is approximated by a converging sequence of test sequences \( t_\star \) in \( T_S \) (Theorem 12.1). The degree of approximation is limited only by the amount of testing resources available.

The first two properties imply that our test selection is a convergent approximation process such that the more test cases selected (through successive passes), the closer the selected set comes to the original set, with no relevant test cases.
or regions of test cases being missed out by mere overlook. This is a critical, common problem in the manual test selection process. The last property implies that in the approximation process, our selected set contains only finite test sequences even though the original set may contain infinite test sequences. As observed in [Led92], an exhaustive tester like a canonical tester can be viewed as a theoretical upper bound on the testing process. Our selection algorithm based on the metric $dt$ behaves as a convergent approximating process for this theoretical upper bound. The degree of approximation depends on the characteristics of the parameters of the metrics selected by the test designer, but the convergence always exists if the constraints of the metric definition are satisfied (Constraints 10.1 and 10.2).

4.4 Test Selection and Trace Equivalence

The connection between our proposed metric and the testing theory can be easily established via the trace equivalence which is the proper equivalence we are assuming in our theoretical setting for this metric.

To analyse the relationship between the selected set of tests $T_S$ and the original set of execution sequences, we need to define the distance between two sets. One such definition was given earlier in Definition 11.2 and used in our definition of coverage metric. We now introduce another definition of the distance between two sets, which will be referred to as the Hausdorf distance [HS65].

**Definition 4.3** Let $S, T$ be two sets equipped with some appropriate metric $dt$. The distance between these metric spaces is defined as follows:

$$dt_H(S, T) = \max\{\sup\{dt(s, T) : s \in S\}, \sup\{dt(t, S) : t \in T\}\}$$

Let $T_S$ be a test suite, viewed as a set (space) of test sequences equipped with the distance $dt$, and for simplicity let $T_S$ denote the resulting metric space $(T_S, dt)$.

Let $S$ represent a specification of a communication system, $\text{Traces}(S)$ represent the set of traces and $\text{Comp}(S)$ represent the set of all computations of $S$. Assume that the same distance $dt$ is defined on the sets $\text{Traces}(S)$ and $\text{Comp}(S)$, and let $\text{Traces}(S)$ and $\text{Comp}(S)$ denote the resulting metric spaces.

The following facts about these metric spaces and their (Hausdorf) distance are easy to establish. They are stated here without proofs.

**Proposition 4.2** Let $T_S$ and $\text{Comp}(S)$ be sets of any (finite or infinite) sequences (where the set of infinite sequences must be closed). Then the following is true:
\[ T_S = \text{Comp}(S) \iff d_{tH}(T_S, \text{Comp}(S)) = 0 \]

i.e., if the (Hausdorff) distance between a test suite and the set of all computations of a specification (and therefore also the trace-set of the specification) is 0 then they are the same. More specifically, it is implied by this proposition that in this case that the test suite is trace-equivalent to the specification, and vice versa.

In view of the above proposition, let \( \epsilon_k \) denote the decreasing sequence of target distances for the successive passes of the test selection algorithm. Let \( T_{S_k} \) denote the appropriate \( \epsilon_k \)-dense subset of \( G(S) \), assuming \( G(S) \) is the starting set in the selection process. Since \( \epsilon_k \to 0 \) as \( k \to \infty \), clearly \( \sup\{d_t(s, T) : s \in G(S)\} \), as well as \( \sup\{d_t(t, G(S)) : t \in T\} \) will tend to 0. Therefore the (Hausdorff) distance between the selected test sequence set and \( G(S) \) will also tend to 0, as the multipass test selection process tends to satisfy its limiting conditions. Thus, we can claim that, in the limit, the selected test sequence set produced by the algorithm is trace-equivalent to the original specification, provided the algorithm starts with the trace set of the specification.

**Proposition 4.3** If \( \text{Comp}(S) \) is closed then the following relation holds:

\[
\sup\{d_t(x, \text{Comp}(S)) : x \in T_S\} = 0 \Rightarrow T_S \subset \text{Comp}(S)
\]

This is obviously true.

Furthermore, notice that if the set \( \text{Comp}(S) \) is reduced to traces of the specification \( S \) only, then the above proposition gives us a metric characterization of the may preorder which plays an important role in the testing equivalence [dN87]. Other associated testing theories [Bri88, Led92] also use this relation in characterizing their respective preorders.

**Proposition 4.4** Let \( T_S \) and \( \text{Comp}(S) \) be sets of any (finite or infinite) sequences (the sets need not be closed). If

\[ d_{tH}(T_S, \text{Comp}(S)) = 0 \]

then the sets of all truncations in \( T_S \) and \( \text{Comp}(S) \) are the same.

In this case, if the Hausdorff distance between a test suite and a computation set of the specification is 0, this proposition characterizes the concept of trace (string) equivalence between the associated concurrent systems.
4.5 Some Covers and Metric Parameter Tuning

As shown earlier in this section, the test selection algorithm based on metric $dt$ is guaranteed to be a convergent approximating process. The parameters of the metric definition ($r(\delta_k(A,B))$ and $p_k$) and of the test selection algorithm (the target distances $\epsilon_k$) can be "tuned" according to expert knowledge of fault models and of the specific protocols to improve the degree of approximation and the speed of convergence of the test selection process to the original specification, subjected to some cost constraint. Empirical studies on fault models would be helpful in tuning these parameters for specific protocols. In this report, we present our analytical observations on some parameter settings which would result in some well known test covers in software engineering: the branch, switch and boundary-interior covers. Even though these covers are defined for finite state machine models whereas our theoretical framework is based on process algebras, we think it is still useful to interpret some of our results in other related models. Interested readers are referred to [Kar88] for an excellent discussion on the relationship between process algebras and finite state machine models.

We first present the brief definitions of those covers and then discuss the appropriate parameter settings in our metric based selection which would result in those covers.

**Definition 4.4 Branch Cover**

The number of executable paths in a design with loops may be infinite. A common practice is to require that the test sequences form a "branch cover", i.e. every branch in the design is traversed by at least one test sequence.

**Definition 4.5 Switch Cover**

A more stringent test coverage is called a "switch cover". A switch is a branch-to-branch pair. A "switch cover" consists of test sequences such that every branch-to-branch pair in the design graph is traversed by some test sequence.

**Definition 4.6 Boundary-Interior Cover**

Another test coverage, called "boundary-interior test cover" is also used frequently in practice. Boundary test paths refer to test paths that enter a loop without causing the loop to be iterated.

In the following Propositions, let $T_S$ and $\text{Tr}(S)$ denote a selected set of test sequences and the set of traces of a protocol specification $S$, respectively. Furthermore, let $t_s = \{(a_{s_k}, \alpha_{s_k})\}$ denote an element of the set $T_S$. Likewise, let $s_t = \{(a_{t_k}, \alpha_{t_k})\}$ be an element of the set $\text{Tr}(S)$. Let $p_k$ be a summable, decreasing sequence of numbers according to the metric definition.

**Proposition 4.5** If $dt_H(T_S, S) < \epsilon < p_k$, for some $0 \leq k_1 \leq \infty$,
then each trace \( s_a \) of the specification \( S \) must have been approximated by at least one test sequence \( t \), from the set \( T_S \), with the following characteristic: all \( a_{t_k} \) in the sequence \( t \) are the same as all \( a_{s_k} \) in the sequence \( s_a \), for \( k=1,...,k_t \).

This selection criteria effectively amounts to the branch cover criteria (up to the level \( k = k_t \)) in general software testing.

**Proposition 4.6** If \( d_{tH}(T_S, S) < \epsilon < \rho \alpha p_k \) for some \( 0 \leq k_t \leq \infty \), then each trace \( s_a \) of the specification \( S \) must have been approximated by at least one test sequence \( t \), from the set \( T_S \), with the following characteristic: all \( a_{t_k} \) in the sequence \( t \) are the same as all \( a_{s_k} \) in the sequence \( s_a \). Moreover, for all \( s_a \), the difference in the level of recursion of any (recursive) call which occurs up to the level \( k_t \), within sequence \( s_a \) and its closest test sequence (i.e. \( |a_{s_k} - a_{t_k}| \), \( 0 \leq k \leq k_t \)) is at most \( \alpha \), where \( \rho \alpha = r(\delta_\alpha) \).

This selection criteria amounts to boundary-interior cover in the general software testing theory.

**Proposition 4.7** Let the trace set \( Tr(S) \) be closed. We take \( p_k \) to satisfy the basic constraints as given in the definition of the metric. If \( d_{tH}(T_S, S) = 0 \) then every sequence in \( Tr(S) \) must have been completely covered by at least one test sequences in \( T_S \).

The testing strategy using this \( T_S \) satisfies the all switch sets criteria in the general software testing theory.

### 5 Further Research

In this report, we define a metric \( dt \) and introduce a metric based model for test coverage and test selection algorithm.

Since our metric based selection method is theoretically sound and practically promising, we plan to implement and incorporate it into TESTGEN, a flexible environment for test generation and selection being developed at the University of British Columbia. Our experimentation with this metric and the test selection algorithm for various protocols would help to verify their practicality and effectiveness. It may also give us more insight to fault models and the fine tuning of parameters of the metric and of the test selection algorithm, as well as possibly leading to improvement in their definitions.
A natural extension of the work on metrics is to relate it to aspects of reliability in the context of determining how well an implementation performs under prolonged rigorous testing circumstances. In a sense, this will generalize the notion of conformance testing to encompass performance and reliability testing. This is a particularly important issue currently being addressed in software engineering within the topic of building trusted real-time systems.

An interesting issue is to determine the degree of interplay between the control and the data parts of protocols which may affect the applicability of our coverage metrics. Other interesting research issues include the problem of executability of the set of test cases selected, and the possible limitations of the labelled transition systems assumed as the underlying model in our metric definition.

References


6 Appendices

A 1 Theoretical Setting

Definition A.1 A Labelled Transition System is a quadruple \((Q,A,-\mu \rightarrow, q_0)\) where \(Q\) is a countable set of states, \(A\) is a countable set of elementary actions, \(-\mu \rightarrow\), where \(\mu \in A \cup \{\tau\}\), is a set of binary relations on \(Q\) and \(q_0 \in Q\) is the initial state.

After Milner [Mil80], the special symbol \(\tau\) is used to denote an internal action.

A transition system can obviously be "unfolded" into a tree, with the initial state at the root of the tree and arcs labelled with elements from \(A\cup\tau\). The following notations will helpful in expressing formulas and proofs in the paper.

\(A\) denotes the set of visible actions, ranged over by \(a, b, c, \ldots, m, \ldots\)
\(A^*\) denotes the set of strings over \(A\), ranged over by \(s, s', \ldots\)
\(A_\tau\) denotes \(A\cup\{\tau\}\), ranged over by \(\mu, \mu_1, \ldots\).

Let \(p, q, \ldots\) denote states of the transition system, then
\(p\rightarrow q\) stands for \(p \rightarrow \tau \rightarrow q\)
\(p \cdot \mu_1 \mu_2 \ldots \mu_n \rightarrow q\) will denote that there exists \(p_1, \ldots, p_n\), with \(0 < i < n\) such that
\(p = p_0 \rightarrow \mu_1 \rightarrow p_1 \rightarrow \mu_2 \rightarrow \ldots \rightarrow p_{n-1} \rightarrow \mu_n \rightarrow p_n = q;\)
\(p \cdot \mu_1 \mu_2 \ldots \mu_n \rightarrow\) denotes that there exists a \(q\) such that \(p \cdot \mu_1 \mu_2 \ldots \mu_n \rightarrow q.\)
\(p= \rightarrow q\) stands for \(p \cdot \tau^n \rightarrow q\) with \(n \geq 0\).
p=μ ⇒ q denotes that there exists p₁ and p₂ such that p=⇒ p₁ - μ → p₂ =⇒ q
p=a₁a₂...aₙ ⇒ q denotes that there exists p₁, ..., pₙ, with 0 < i < n such that
p = p₀ = a₁ ⇒ p₁ = a₂ ⇒ ... ⇒ pₙ₋₁ = aₙ ⇒ pₙ = q
p=s⇒ will denote that there exists q such that p=s⇒q

Definition A.2 The trace set of a specification (system) q is defined as

Traces(q)=\{s ∈ A*|q = s⇒\}

The most straightforward view of equivalence of two systems is to consider
as equivalent those systems which can perform exactly the same sequences of
visible actions.

Definition A.3 If T₁ = (P, A, -μ →₁, p₀) and T₂ = (Q, A, -μ →₂, q₀) are two
transition systems, then:

T₁ ~s T₂ iff for all s ∈ A* p₀ = s⇒ if and only if q₀ = s⇒.

It immediately follows from the definition that T₁ ~s T₂ iff Traces(q₀) =
Traces(p₀).

It is obvious that ~s is an equivalence relation, and it is referred to as trace
or string equivalence.

A 2 Examples of Calculation of Distance dt

Example 1

Let the following description represent a simple protocol:
Start = ConReq; ConConf; DataPhase
DataPhase = (fix X). DataReq; X + DisReq; stop + DataInd; X + DisInd; stop

For computing the distance dt, the pattern weight and the weight of the level
of recursion (the sequences \{p_k\} and \{τ_k\}) will be defined as in the Example 1 and
Example 3 of Section 2.

Consider then the following three computations in the space D of this speci-
fication:

A = E₀ - ConReq - > E₁ - ConConf - > E₂ - DataReq - > E₃ - DataReq - > E₄ - DataReq - > E₅ - DataReq - > E₆ - DisReq - > Stop
The mapping $\xi$ maps these computations into the following sequences of pairs:

$\xi(A) = \{(\text{ConReq,1}), (\text{ConConf,1}), (\text{DataReq,4}), (\text{DisReq,1})\}$
$\xi(B) = \{(\text{ConReq,1}), (\text{ConConf,1}), (\text{DataReq,5}), (\text{DisReq,1})\}$
$\xi(C) = \{(\text{ConReq,1}), (\text{ConConf,1}), (\text{DataReq,4}), (\text{DataInd,1}), (\text{DataReq,1}), (\text{DisInd,1})\}$

All three computations are equidistant with respect to the distance $d_a$:

Indeed,

$$d_a(A,B) = d_a(A,C) = 1/2^6.$$ 

On the other hand, with respect to the testing distance, the following holds:

$$d_t(A,B) < d_t(A,C)$$

since

$$d_t(A,B) = 1/2^4$$
and $$d_t(A,C) = 1/2^4 + 1/2^5 + 1/2^6$$

Example 2

Assume that the protocol description is given by the same specification as in the example above. Assume further that the sequences $\{p_k\}$ and $\{r_k\}$ are also given as in the example above.

Consider the following three computations in $D$:

$A = E_0 - \text{ConReq} - \rightarrow E_1 - \text{ConConf} - \rightarrow E_2 - \text{DataReq} - \rightarrow E_3 - \text{DataReq} - \rightarrow E_4 - \text{DataReq} - \rightarrow E_5 - \text{DataReq} - \rightarrow E_6 - \text{DisReq} - \rightarrow \text{Stop}$

$B = E_0 - \text{ConReq} - \rightarrow E_1 - \text{ConConf} - \rightarrow E_2 - \text{DataReq} - \rightarrow E_3 - \text{DataReq} - \rightarrow E_4 - \text{DataReq} - \rightarrow E_5 - \text{DisReq} - \rightarrow \text{Stop}$

$C = E_0 - \text{ConReq} - \rightarrow E_1 - \text{ConConf} - \rightarrow E_2 - \text{DataReq} - \rightarrow E_3 - \text{DataReq} - \rightarrow E_4 - \text{DataReq} - \rightarrow E_5 - \text{DataReq} - \rightarrow E_6 - \text{DataInd} - \rightarrow E_7 - \text{DataInd} - \rightarrow E_8 - \text{DataReq} - \rightarrow E(9) - \text{DisInd} - \rightarrow \text{Stop}$
Supposing the distance \( da \), the following holds:

\[ da(A,C) < da(A,B) \]. Indeed,

\[ da(A,C) = \frac{1}{2^6} \text{ and } da(A,B) = \frac{1}{2^8}. \]

With respect to the testing distance, the following holds:

\[ dt(A,B) < dt(A,C) \] since:

\[ dt(A,B) = \frac{1}{2^4} \text{ and } dt(A,C) = \frac{1}{2^4} + \frac{1}{2^5} + \frac{1}{2^6}. \]
A 3 Formal Description of ISO 8072 in LOTOS

behaviour
TConnections [t]
||
TCIdentification [t]
||
TCAcceptance [t]
||
TBackpressure [t]
where
process TConnections [t]: noexit
:=
TConnection [t]
||
TConnections [t]
endproc (* TConnections *)

process TConnection [t]: exit
:=
(TCEP [t] (CallingRole) ||| TCEP [t] (CalledRole))
||
(TCEPAssociation [t] -> exit)
endproc (* TConnection *)

process TCEP [t] (role:TSUserRole): exit
:=
TCEPAddress [t]
||
TCEPIdentification [t]
||
TCEPSPOrdering [t] (role)
endproc (* TCEP *)

process TCEPSPOrdering [t] (role:TSUserRole)
:=
TCEPConnect1 (role) >> accept tsp:TSP in
(( TCEPConnect2 [t] (tsp) >> accept x:TEXOption
    TCEPDataTransfer [t] (x)
    ) [> TCEPRelease [t]
    )

[role = CalledRole] -> exit
endproc (* TCEPSPOrdering *)

process TCEPConnect1 [t] (role:TSUserRole)
:=
[role = CallingRole] -> t?ta:TAddress ?tcei:TCEI ?tc
    [IsTCONreq(tcr) and (ta lsCallingOI tcr)];
exit (tcr)

    [IsTCONind(tci) and (ta lsCalledOf tci)];
exit (tci)
endproc (* TCEPConnect1 *)

process TCEPConnect2 [t] (tc1:TSP) : exit
:=
    [tc2 IsValidTCON2For tc1];
    ( choice x:TEXOption [] [x lsTEXOptionOf tc2]
endproc (* TCEPConnect2 *)
process TCEPDataTransfer [t] (x:TEXOption) : noexit

:=
TCEPNormalDataTransfer [t]

[x = UseTEX] → TCEPExpeditedDataTransfer [t]
endproc (* TCEPDataTransfer *)

process TCEPNormalDataTransfer [t]: noexit

:=
  TCEPNormalDataTransfer [t]
endproc (* TCEPNormalDataTransfer *)

process TCEPRelease [t]: exit

:=
endproc (* TCEPRelease *)