Adaptive Testing of Non-Deterministic Communication Protocols

Mohammed Ghriga and Phyllis G. Frankl

Department of Computer Science, Polytechnic University, 6 Metrotech Center
Brooklyn, NY 11201, USA

Abstract

A new approach to testing protocols in which both the specification and the implementation exhibit non-determinism is described. Test sequences are generated by an adaptive tester, which executes in parallel with the implementation under test (IUT). The adaptive tester attempts to determine a posteriori which non-deterministic choice the IUT has made, and uses this information to decide how to proceed. To achieve this, the different alternatives at each non-deterministic state are characterized by Transition Distinguishing Sequence Sets (TDSS), which can be generated in a preprocessing phase.

Keyword Codes: D.2.2; D.2.5
Keywords: Tools and Techniques; Testing and Debugging; Protocol Specification, Testing and Verification

1 Introduction

For Formal Description Techniques(FDTs) [1, 2, 3, 4] to reach a high level of acceptance among the user community, tools that support the specification, design, and analysis of increasingly complex systems have emerged. One important aspect of such system analysis is testing the conformance of implementations with respect to their specifications. Several methods have been proposed for the conformance testing of protocols in which both the specification and the implementation exhibit non-determinism. These methods do not take into account the unpredictability of the Implementation Under Test(IUT) at run time, as test sequences are generated before testing takes place. This paper describes a new approach to testing such non-deterministic protocols. The purpose of the paper is to set the basis for a realistic non-deterministic testing environment that provides *support* for the actual testing process by gradual resolution of non-determinism and test sequence generation at run-time. The current paper focuses on specifications written in a subset of LOTOS. However, the concepts introduced here could be imported to subsets of other formal languages and adapted to the Non-deterministic Finite State Machine Model(NFSM).

In the context of conformance testing implementations specified in LOTOS, canonical testers (possibly of infinite behaviors) have been introduced in [5]. Another method, the so-called CO-OP method [6], provides for the construction of optimized testers by composition of elementary testers, using the semantics of LOTOS operators. Also, some work has been done towards the generation of testers which preserve the hierarchical structure of the specification [7]. Fujiwara and Bochmann have recently proposed a technique for the derivation of test sequences that guarantee fault detection from non-deterministic state machine specifications [8].

These methods provide for the generation of test sequences, but they do not support the actual testing process. An attempt towards dealing with this problem is proposed in [9], in which the IUT is assumed to be deterministic and the Implementation Conformance Statement(ICS) provided, where ICS denotes the knowledge on how non-determinism has been resolved in the IUT. The notion of ICS becomes inapplicable if the IUT is truly non-deterministic.

When non-determinism is present, a test sequence intended to execute a particular trace may result in execution of part of a different trace. Fujiwara and Bochmann [8] address this issue (in the context of non-deterministic finite state machines) by assuming that there is some fairness between different non-deterministic choices in the IUT, so that if a particular test sequence is repeated many times, it will eventually exercise the intended trace. Similarly, in the context of LOTOS specifications, one could offer a given sequence repeatedly, and assume that the desired behavior in the implementation will eventually be executed. In addition to the limitation of assuming fairness of the implementation, these approaches may result in a great deal of wasted effort. If the sequence and the number of times this sequence is to be executed are determined a priori, the sequence may be run numerous times after the required non-deterministic choice has already been taken while trying to exercise another trace. If the question of whether a given sequence exercised the intended behavior is determined a posteriori, "unsuccessful" sequences can be wasted, even though they may actually be exercising other important behaviors. Our approach focuses on monitoring the actual behavior of the system during execution of the test sequence. We describe a technique for determining which non-deterministic choice has been made, based on subsequent behavior of the system. This information can be used both to monitor which behaviors have actually occurred, and to allow adaptive generation of test sequences intended to drive the implementation toward un-executed behaviors.

In this paper, we consider the case where the specification and its implementation are non-deterministic. Moreover, we allow for the presence of internal (non-observable) events. Based on the notion of *n-testers*[7], we describe first a *Generic Adaptive Tester(GAT)* that is *amenable* to implement the following adaptive testing strategy: *each test run is an attempt to cover a new trace*. It is characterized as *generic* since it is not equipped with any strategy on how to make a choice among several possible events at some state(s) of the System Under Test(SUT)[1]. We assume that GAT be prepared to engage in (or offer) one of those alternatives at each stage of its evolution. Thereafter, we illustrate how GAT can be augmented with test selection criteria in order to provide for run-time test sequence generation. GAT with such test selection criteria implements our adaptive testing strategy.

In contrast with state-based identification methods used for test sequence generation

[1]System Under Test refers to both GAT and IUT.

[10, 11, 8], we propose a new transition-based identification method, called *Transition Distinguishing Sequence Sets (TDSSs)*, which provides for a characterization of the different transitions (alternatives) at non-deterministic states during the execution of IUT. The various possible transitions are eventually distinguished by distinct subsequent observable behaviors. These TDSSs set the theoretical basis for gradual resolution of non-determinism and test sequence generation at run-time.

The organization of the paper is as follows. `Section 2` briefly discusses what conformance testing is, and introduces certain assumptions about the interaction of the tester, and GAT in particular, with an IUT. `Section 3` reviews tree model, which is used to denote the functionality of a system on the basis of its specification. `Section 4` proposes a new transition-based method for the identification of nondeterministic transitions taken place during the concurrent execution of a tester process with an IUT. The notion of *Transition Distinguishing Sequence Sets (TDSSs)* will be introduced. `Section 5` shows how TDSSs will be used to provide for the adaptiveness of GAT throughout the testing process. Moreover, we introduce the abstract notion of an *Execution Tree (ET)*, which allows for a formal description of the use of TDSSs during testing. Overall trace coverage will be also discussed. `Section 6` describes the basis for an implementation scheme that provides for an efficient use of TDSSs in our run-time testing environment.

2 Finite Testing

Conformance Testing for communication protocols checks whether implementations conform to their specifications. Given a system specification S and an implementation I of the system, I is a *conforming* implementation of S if the allowed behavior of S is contained in that of I (trace inclusion [5, 6]). Conformance testing is a "black-box" testing. Test sequences are derived from system specifications and IUTs are analyzed in terms of their external observable behaviors.

We view GAT (our tester) as a process running concurrently with, and communicating with the IUT. This abstraction of the implemented tester has been adopted in [7]. Given a system specification S, running GAT concurrently with an IUT will result in testing one single allowed behavior in S, which is usually called a trace of S. A run is considered successful if the tester reaches a terminal state.

To be able to decide on the conformance of an IUT to its specification S according to the definition of *(general) canonical* testers [5], all traces in S must be tested. It follows from [5, 6], the IUT conforms S if, and only if all runs which correspond to the exploration of traces in S are successful. In general, it is hard to decide whether or not an IUT has failed a test. An IUT fails if its response and that of the GAT do not result in a legitimate synchronization, or even worse if it may not accept the event offered and thus never respond.

The testing process is finite, and so must be the test interaction sequences. For this, we adopt the notion of n-testers, which has been introduced in [7]. An n-tester tests all traces whose external behavior is of length less than or equal to n and does not deadlock with the implementation. In this context, GAT is an *n-tester.* It tests for *n-implementation.* To pass an *n-test* an IUT should cooperate on all traces whose external behavior is of length less than or equal to n and then either complete successfully or be ready to continue [7]. Similarly, we may establish whether certain particular aspects of the specification are exhibited by the IUT and testing reduces to the coverage of a pre-determined subset of the set of traces. Note that the relation of *n-testers* to canonical testers has been established in [7].

For large processes, n must be large enough to ensure that the innermost subprocesses were tested. Furthermore, as we shall see in `Section 4`, the choice of n may affect our ability to distinguish between the various subsequent behaviors at non-deterministic states.

3 A Tree Model

A specification S describes the functionality of a system in terms of possible interaction sequences between the system and its environment. Given S, we model the system functionality [9, 12] as a tree $ST(V, E)$, where V is a non-empty set of vertices and E is a set of edges. In $ST(V, E)$, each e in E represents a transition of the system, including internal transitions (also called silent-transitions). An edge e is labeled with an event in LOTOS terminology [1, 13], which is the atomic form of interaction. Moreover, each v in V represents the root of a subtree defining all possible remaining subsequences of events. Otherwise stated, each vertex v at the end of an event μ represents the state of the system after the occurrence of μ. Accordingly, the root of $ST(V, E)$ will represent the initial state of the system.

Let ϵ denote the LOTOS internal event and L the set of labeling events. With respect to $ST(V, E)$, we define a transition function *next* based on *observational behavior* from $V \times L \backslash \{\epsilon\}$ to $\wp(V)$, where $\wp(V)$ is the power set of V: if $v \in V$ and $\mu \in L \backslash \{\epsilon\}$ then $next(v, \mu)$ is the set of states that can be reached by a path originating at v, whose label is of the form $\epsilon^{[m]}\mu$ $(m \geq 0)$.

A path from v to w in $ST(V, E)$ with a label of the form $\epsilon^{[m]}\mu$ $(m \geq 0,\ \mu \in L)$ is referred to as a transition, throughout the paper. Also, we write $t = (v, w)$ to denote such transitions. Therefore, transition t refers to either a *single-step* transition (when $m = 0$) or a *multiple-step* transition (when $m \geq 1$).

A transition(v, w) is said to be a *maximal* transition if v is not reached by an internal event, meaning that the transition from the parent of v to v is not labeled by ϵ. If a transition (v, w) with label μ is not maximal, there exists a unique maximal transition (z, w) labeled $\epsilon^{[m]}\mu$ $(m \geq 1)$ that includes the transition (v, w) as a suffix.

A transition (v, w) with label μ is said to be *non-deterministic* if $\mu = \epsilon$ or if $\mid next(z, \mu) \mid > 1$ for $\mu \neq \epsilon$, where z is the head of the maximal transition that corresponds to (v, w). Note that if v is not reached by an internal event, we have $z = v$. A state v is said to be *non-deterministic* if there is at least one non-deterministic transition originating at v.

Since $ST(V, E)$ may have infinite traces, we define $ST_n(V_n, E_n)$, for some n, as the specification tree where all traces are limited to an observable behavior of length n. Accordingly, $ST(V, E)$ would correspond to $ST_\infty(V_\infty, E_\infty)$.

4 Transition Distinguishing Sequence Sets

The presence of non-determinism in SUT brings about the need to accommodate and adapt to the unpredictability of its behavior during testing, and thus the need for adaptive testing. To resolve non-determinism, it is extremely important to address the question: *"how should the observed behavior of the system be used to resolve non-determinism?"* . In addition to the past observed behavior of SUT, this question is fundamental in the sense that the resulting resolution process gives a systematic insight into what is possible as subsequent behavior of the system.

Using the underlying semantics of specification trees, note that each $v \in V$ represents the root of a subtree defining a set of all possible sequences of events, originating at v. Clearly, there is a corresponding set of sequences that are externally observable, obtained by removing the internal events. Let $SEQ(v)$ be such a set.

Suppose that the SUT is at state $v \in V$ and some external event μ has been *successfully* offered or exchanged with the system environment, resulting in a non-deterministic transition. The set of possible current states of SUT is $next(v, \mu)$. Let $next(v, \mu) = \{w_1, w_2, ..., w_k\}$, where $k > 1$. Thus, each w_i defines a set of observable sequences $SEQ(w_i)$.

Our approach, in essence, is based on the fact that if

1. a subsequence γ has been observed immediately after a non-deterministic transition μ at state v, and
2. there is a unique w_i that exhibits a subsequent behavior with γ as a specified prefix, i.e.
   - $\exists \alpha, \exists i$ such that $\gamma\alpha \in SEQ(w_i)$, and
   - $\forall \beta, \forall j \neq i : \gamma\beta \notin SEQ(w_j)$

then the transition from v to w_i must have occurred.

To identify which transition has taken place at a nondeterministic vertex, we define a unique signature called the Transition Distinguishing Sequence Sets (TDSS) for each non-silent *single-step* transition in the specification tree $ST(V, E)$, if it exists. First, we provide a characterization of the maximal transitions in the specification tree.

Definition 1 *Let $ST(V, E)$ be a specification tree. Let $v \in V$ be a vertex that is not reached by an internal event. Let $\mu \in L \backslash \{\epsilon\}$. Let $next(v, \mu) = \{w_1, w_2, ..., w_k\} \subseteq V$, where $k \geq 1$, and next is the transition function associated with $ST(V, E)$. Let $t_i = (v, w_i)$ denote the maximal transition from v to w_i.*
The Transition Distinguishing Sequence Set for transition t_i, denoted $TDSS(t_i)$, is the set of specified external interaction subsequences originating at state v defined as follows: $TDSS(t_i) = \{\mu\gamma\}$, where γ is a minimal length sequence such that

1. *γ is an observable subsequence originating at w_i, and*
2. *there is no $j \neq i$ such that γ is an observable subsequence originating at w_j.*

$\square$

The $TDSS$ of any maximal transition $t = (v, w)$ with label of the form $\epsilon^{[m]}\mu$ ($m \geq 0$ and $\mu \neq \epsilon$) can be computed, provided that it exists.

When $m \geq 1$, t is a multiple-step transition from v to w. By definition of $TDSS$, $TDSS(t)$ identifies the traversal of the path from v to w, and in turn, the occurrence of any transition from an intermediate vertex(within the path) to w. Hence, we have

Lemma 1 *Let t be a transition from v to w, whose label is of the form $\epsilon^{[m]}\mu$, where $m \geq 1$ and $\mu \in L\backslash\{\epsilon\}$. Let $p(w)$ be the parent of w in $ST(V, E)$.*

$(\forall k)_{1 \leq k \leq m+1} : TDSS(p^{[k]}(w), w) = TDSS(t)$

where,

$$\begin{cases} p^{[0]}(w) = w \\ p^{[k]}(w) = p^{[k-1]}(p(w)) \end{cases}$$

□

Considering the above lemma for $k = 1$, we get $TDSS(p(w), w) = TDSS(t)$. Therefore, `definition 1` and `lemma 1` provide for the computation of the $TDSS$ of any non-silent single-step transition. This computation can be carried out in a preprocessing phase before the testing process takes place. $TDSS(t) = \emptyset$ if $TDSS(t)$ does not exist. If $TDSS(t) \neq \emptyset$ then t is said to be *distinguishable*.

Using the definition, we can easily prove the following theorem that is the basis for transition identification during the testing process.

Let $\gamma_{\rightarrow i}$ denote the prefix of length i of the sequence γ.

Theorem 1 *Let $v \in V$ be a vertex that is not reached by an internal event. Let γ be an observed interaction sequence originating at v. If*

$\exists w \in next(v, \gamma_{\rightarrow 1})$, $\exists i \in \{1, 2, \ldots, |\gamma|\}$ *such that* $\gamma_{\rightarrow i} \in TDSS(v, w)$,

then the occurrence of transition (v, w) is confirmed provided that v has been reached.

□

The results that have been developed for $ST(V, E)$ hold for $ST_n(V_n, E_n)$, for any arbitrary n. Clearly, the choice of n may affect our ability to distinguish between non-deterministic transitions as their respective subsequent behaviors may have large common prefixes. Accordingly, we let $TDSS_n(t)$ denote the transition distinguishing sequence set of transition t in $ST_n(V_n, E_n)$. A transition t is said to be *n-distinguishable* if $TDSS_n(t) \neq \emptyset$.

Example 1 Consider the following specification [5]:

```
S = ConReq_A;
    (   ConInd_B; exit
    []  DisReq_A;
        (   i;  exit
        []  ConInd_B; exit)
    []  i; DisInd_A;
        (   i; exit
        []  ConInd_B; exit)
    )
```

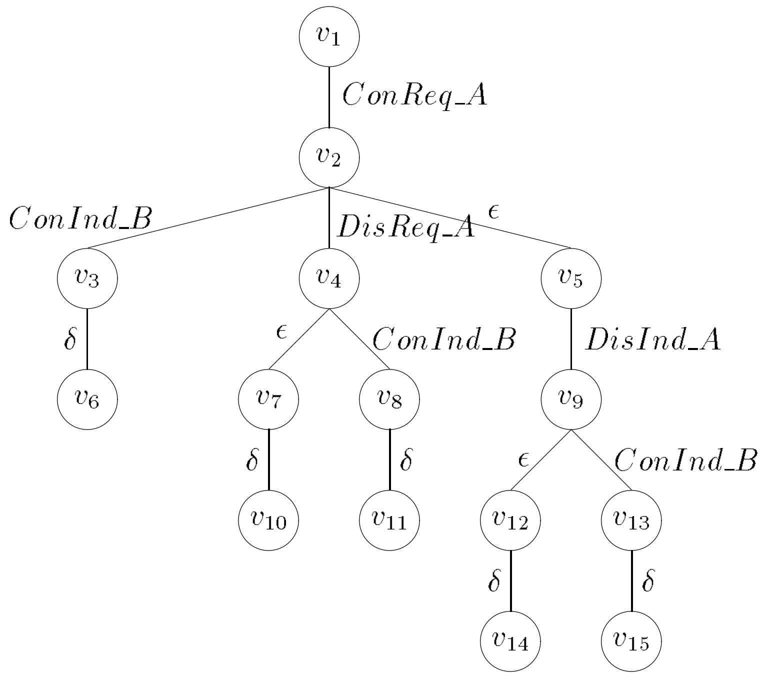

Figure 1: Specification tree $ST(V, E)$ of S

Let δ denote *exit*. The specification tree of S is shown in `Figure 1`. By definition, we have $TDSS(v_1, v_2) = \{ConReq\_A\}$ and $TDSS(v_2, v_9) = \{DisInd\_A\}$. For this latter case, note there are two other transitions originating at v_2, namely transitions (v_2, v_3) and (v_2, v_4), but the subsequence $DisInd\_A$ is not a specified prefix for any of their subsequent behaviors, originating at v_2. Using `lemma 1`, we get $TDSS(v_5, v_9) = \{DisInd\_A\}$. The definition does not provide directly for the computation of $TDSS(v_5, v_9)$ since (v_5, v_9) is not a maximal transition.

5 Testing Using TDSSs

In this section, we describe the use of TDSSs during testing to resolve nondeterminism by specifying the actions carried out by GAT. Based on `theorem 1`, their use enables us to progressively figure out the execution paths that have actually been taken by the IUT. At each atomic interaction, an attempt is made to reduce the set of possible execution paths. This reduction process would result in an effective adaptive test sequence selection as we shall suggest in `Section 5.3`.

A path $p = v_0 v_1 v_2 ... v_k$ is said to be deterministic if $\forall i_{(0 \leq i < k)} : v_i$ is a deterministic state, and it is of length *zero* if $k = 0$. Note that v_k is not necessarily deterministic.

5.1 Informal Presentation

For testing, suppose that SUT takes a deterministic path, possibly of length *zero*, from the initial state to a non-deterministic state s in $ST(V, E)$, result of an interaction subsequence α. Thus, immediately after the interaction subsequence α, SUT is in state s.

Due to its nondeterministic nature, s may present deterministic and nondeterministic transitions. If the next (successful) interaction event $\mu_0 (\neq \epsilon)$ results in a deterministic transition $t = (s, w)$, then SUT moves to state w. Otherwise, SUT might move to any state $w \in next(s, \mu_0)$. If $| next(s, \mu_0) | \geq 2$, such state s is said to have an *unresolved nondeterminism*. However, if $| next(s, \mu_0) | = 1$, the non-determinism at s will be instantly resolved at the occurrence of μ_0. Such a case is exemplified by state v_2 and the external event $DisInd\_A$ in `Figure 1`. Although v_2 is non-deterministic, if $DisInd\_A$ occurs, we know then that SUT must have taken the path $v_2 v_5 v_9$.

As the testing process continues, we may encounter other states with unresolved nondeterminism. Thus, we need to keep track of all these states. In attempt to resolve this nondeterminism, we shall use TDSSs along with the current observed interaction sequence to determine which particular transition has taken place at a state with unresolved nondeterminism, if possible. If this attempt is successful, it may lead to a reduction of the set of possible states of the SUT.

In other words, the GAT attempts to match the observed behavior against the relevant transition-distinguishing sequences. `Section 5.2` describes this idea more formally, and `Section 6` describes how this matching process can be performed efficiently.

5.2 Formal Presentation

Let $CS(s_i, \beta)$ denote the set of all possible current states of SUT given that the system has been initially in s_i and a sequence β,originating at state s_i, has been externally observed.

Let $COIS_\gamma(s_i)$ be the current observed interaction sequence originating at state s_i. These partial interaction sequences are suffixes of the overall observed interaction sequence γ and are associated with states with unresolved nondeterminism as labels. We define an *execution tree* ET_γ to be an annotated tree which includes all possible execution paths whose external behavior is γ, with $COIS_\gamma(s_i)$ as a label for each s_i with unresolved nondeterminism. Note that ET_γ need not be explicitly constructed. Henceforth, it will be characterized by the set of edges that it contains.

Let U be a set of states with unresolved nondeterminism. We assume the availability of the following operations: *CreateSet*, *InsertElement* and *DeleteElement*[2].

Initially, we have

- $CreateSet(U)$
- $ET_\epsilon = \emptyset$
- $CS(s_0, \epsilon) = s_0$ (s_0 is the root of $ST(V, E)$)

and the current observed interaction sequence is obviously ϵ, the empty sequence.

[2]The operation *CreateSet* creates an empty set, while the operations *InsertElement* and *DeleteElement* inserts an element into the set and deletes a given element from the set, respectively.

if U is empty then

1.

$$ET_{\alpha\mu_i} = ET_\alpha \bigcup (\cup_{j=1}^{k} \{(s_l, w_j)\}),$$

where

$$CS(s_0, \alpha) = \{s_l\},$$
$$next(s_l, \mu_i) = \cup_{j=1}^{k} \{w_j\}.$$

2. if $k > 1$ then $--$ Note that $k = \mid next(s_l, \mu_i) \mid$
   - label s_l in $ET_{\alpha\mu_i}$ by $COIS_{\alpha\mu_i}(s_l) = \mu_i$.
   - $InsertElement(s_l, U)$.
3. $CS(s_0, \alpha\mu_i) = next(s_l, \mu_i)$.

Figure 2: Expansion of the ET when U is empty

Inductive Step:

Let us assume that an interaction sequence α has been observed, ET_α represents possible execution paths taken by SUT, and U holds the states with unresolved nondeterminism.

Thus, $CS(s_0, \alpha)$ is the set which consists of the leaves of ET_α.

Suppose that the next interaction event is $\mu_i (\neq \epsilon)$. We distinguish two cases according to whether U is empty or not. The actions carried out by GAT in each case are shown in `Figure 2` and `Figure 3`, respectively.

Case1: `U is empty and` μ_i `is a successful interaction`

Since U is empty, there are no pending points with unresolved nondeterminism. Consequently, the current execution tree consists of only one single path and the set of possible states of SUT is a singleton. If the next successful interaction μ_i results in a nondeterministic transition such that $\mid next(s, \mu_i) \mid \geq 2$, then s is a state with unresolved nondeterminism and therefore should be inserted in U and labeled with μ_i. Furthermore, the execution tree is expanded to reflect the possible transitions of the system from s to the different elements of $next(s, \mu_i)$, which become the current possible states of SUT.

However, if μ_i results in a transition such that $\mid next(s, \mu_i) \mid = 1$, the above described process remains the same except there is no state labeling and U remains unchanged. `Example 2` below illustrates this case.

Case2: `U is not empty and` μ_i `is a successful interaction`

In this case, we update the labels associated with the states kept in U. The update operation consists of concatenating μ_i with the existing labels in the execution tree. Thereafter, we compare those labels against the $TDSS$s of the relevant transitions in attempt to reduce the execution tree, and thus, the set of possible states of SUT.

The identification of a particular transition, say (s, w), at an unresolved nondeterminif U is not empty then

1. Update the labels in the execution tree

$$COIS_{\alpha\mu_i}(s_j) = COIS_{\alpha}(s_j) \mid \mu_i, \forall s_j \in U$$

   where | denotes the concatenation symbol.

2. Reduce ET_α:
   for each element s_j of U do
   if $\exists w \in V : COIS_{\alpha\mu_i}(s_j) \in TDSS(s_j, w)$ then

   - $DeleteElement(s_j, U)$,
   - Remove the label associated with s_j,
   - $ET_\alpha = (ET_\alpha \backslash T_{s_j}) \cup p_w^{s_j} \cup EST_\alpha^w$,

   where T_{s_j} is the subtree rooted at s_j in ET_α, $p_w^{s_j}$ is the path from s_j to w in ET_α, and EST_α^w is the subtree whose root is w in ET_α. Make $CS(s_0, \alpha)$ be the set of leaves of the reduced ET_α.

3. Expand the *Execution Tree*

$$ET_{\alpha\mu_i} = ET_\alpha \bigcup (\cup_{s_l \in CS(s_0,\alpha)} (\cup_{j=1}^{k_l} \{(s_l, w_j^l)\}),$$

   where

$$next(s_l, \mu_i) = \cup_{j=1}^{k_l} \{w_j^l\}, \forall s_l \in CS(s_0, \alpha).$$

4. if $k_l > 1$ then —— Note that $k_l = \mid next(s_l, \mu_i) \mid$

   - label $s_l \in ET_{\alpha\mu_i}$ by $COIS_{\alpha\mu_i}(s_l) = \mu_i$.
   - $InsertElement(s_l, U)$.

5. $CS(s_0, \alpha\mu_i) = \cup_{s_l \in CS(s_0,\alpha)} next(s_l, \mu_i)$.

Figure 3: Expansion of the ET when U is not empty

istic state s may result in a substantial reduction of the set of possible execution paths of the system. This reduction takes place at state s: the subtree rooted at s is reduced to the set of paths that have transition (s, w) as a prefix.

Hence, it follows a reduction of the set of possible states of SUT. This must be accompanied with the removal of s from the set U and that of its label from the execution tree. Finally, the execution tree expansion translates into the expansion of each current state as described in the previous case.

Example 2 Again, let us consider the specification S given in `Example 1`. The testing process starts with an empty *Execution Tree*, an empty set U, and the empty sequence as the current observed interaction sequence. The initial state of SUT is v_1, and if synchronization on event $ConReq\_A$ is successful, SUT takes a deterministic transition (v_1, v_2). Hence, $ET_{ConReq\_A} = \{(v_1, v_2)\}$, $CS(v_1, ConReq\_A) = \{v_2\}$, and U remains *empty*.

At present, GAT makes available either $ConInd\_B$, $DisReq\_A$, or $DisInd\_A$. The occurrence of either event $ConInd\_B$ or $DisReq\_A$ results in a deterministic transition and the *Execution Tree* is expanded accordingly. However, if event $DisInd\_A$ occurs, we conclude that SUT must have taken the *multiple-step* transition (v_2, v_9) or transition (v_5, v_9) must have taken place since $TDSS(v_2, v_9) = TDSS(v_5, v_9) = \{DisInd\_A\}$. In this particular case, we have also $next(v_2, DisInd\_A) = \{v_9\}$. Thus, $ET_{ConReq\_ADisInd\_A} = \{(v_1, v_2), (v_2, v_9)\}$, $CS(v_1, ConReq\_ADisInd\_A) = \{v_9\}$, and U remains unchanged. Note that the *Execution Tree* contains the multiple-step transition (v_2, v_9) instead of the actual individual transitions. But, assuming that each vertex knows its parent, we can easily recover the actual single-step transitions that have taken place.

For this particular case of the specification, there is no labeling of states and U remains empty throughout testing. This is basically due to the fact that all non-deterministic transitions are distinguished at the occurrence of the next external event.

5.3 Trace Coverage

It is extremely important to monitor the path coverage of $ST(V, E)$ since the conformance verdict of an IUT relies on exercising all traces of its specification S. And equally so, if we aim to exercise a pre-determined set of traces. This set could be a collection of traces, satisfying a set of predefined testing criteria. Recall that if γ *is the interaction sequence that has been observed, then the set of possible paths covered is given by* ET_γ. Another important aspect of testing is the ability to detect faults in the IUT.

In general, the overall path coverage of a given system specification and error detection capabilities are functions of test sequence selection criteria. Once these test selection criteria are formally defined, a characterization of faulty behavior of the IUT and error detection capabilities can be addressed.

Next, we illustrate the adaptive nature of our testing approach and give a simple illustration on how TDSSs can be used to select future paths.

The commonly used approach is to generate conformance test sequences without any consideration for the unpredictability of the IUT, and thus, testing translates into exercising one pre-determined test sequence at a time. This approach to testing non-deterministic implementations may result in a *significantly* high number of test run trials, especially

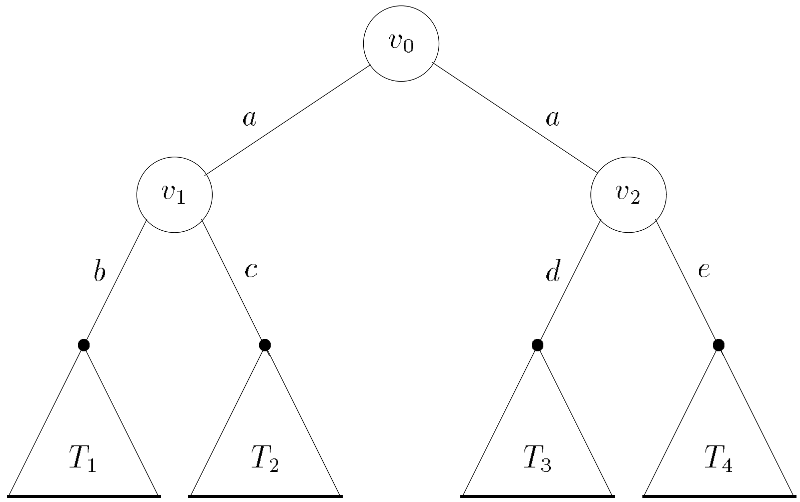

Figure 4: A Schematic Specification Tree

if the SUT has a high degree of non-determinism and if some events corresponding to non-deterministic choices are much likely to occur than others. In contrast, our adaptive testing approach recognizes the uncontrollability of non-determinism and only attempts to enforce the resolution of non-determinism so that new traces are covered. Intuitively, our adaptive approach will do no worse than the previously described approach irrespective of the fairness assumption.

As suggested, TDSSs can be the basis for the definition of test selection criteria that provides for test sequence generation at run-time. The key idea is as follows: knowing what has been covered, the tester tries to enforce the IUT to cooperate on the sequences (of the relevant TDSSs) leading to subtrees that are not completely covered. The relevant TDSSs to be considered will be determined during the expansion of ET. To illustrate this, consider the schematic specification tree shown in `Figure 4`.

Let us suppose that after a number of test runs the paths leading to the leaves of the subtrees T_1 and T_4 have been covered. In this example, we have $TDSS(v_0, v_1) = \{ab, ac\}$ and $TDSS(v_0, v_2) = \{ad, ae\}$. At this stage of the testing process, the tester attempts to resolve non-determinism by trying either ac or ad. This is accomplished by offering the shared prefix which in this case is event a. If there is a successful cooperation on this shared prefix, we have to offer either c or d. In the absence of the *fairness assumption* or any information concerning the likelihood of the non-deterministic events, we may choose to offer c if the coverage percentage of T_2 is higher than that of T_3 where the coverage percentage of a tree T is defined as the proportion of leaves in T that are reached by traces already covered over the total number of leaves in T. *If* c is successfully exchanged, the same technique is applied to T_2 when faced with non-deterministic choices; *otherwise*, event d is tried. If the attempts on c and d fail, the attempt to cover a new trace has failed. Test run trials are considered until we reach the desired path coverage. GAT with such a test selection criterion implements our adaptive testing strategy.

Under the *fairness assumption*, our adaptive testing approach ensures an overall trace coverage of the non-deterministic IUT or any subset of traces that satisfy a pre-defined set of testing criteria in a finite number of test run trials. In practice however, there is no need to make such an assumption, but rather define appropriate test selection criteria (similar to the above criterion) that set the basis for fault detection and give insight to why the current coverage of IUT does not evolve towards the desired coverage. Future research is needed in this area.

6 Comparison of Observed Behavior to TDSSs

In this section, we describe an efficient mechanism for checking the membership of a subsequence into a set of sequences, as the testing process relies heavily on this operation. We use a *deterministic finite automaton (dfa)* for the identification of transitions at a given non-deterministic state. A *dfa* M is defined as a 5-tuple (A, Q, q_1, F, δ), where: A is a set of symbols called alphabet, Q a finite set of states, $F \subseteq Q$ a set of accepting states, $q_1 \in Q$ is the initial state, and δ is a transition function from $Q \times A$ to Q. We write $L(M)$ to denote the language accepted by M.

By *sharing prefixes*, we can easily prove the following theorem that sets the basis for the comparison of the observed behavior to TDSSs in our *run-time* testing environment.

If we consider $ST_n(V_n, E_n)$, we have

Theorem 2 *Let v be a vertex in $ST_n(V_n, E_n)$ that is not reached by an internal event. Let T be the set of all n-distinguishable non-deterministic transitions originating at v of the form $\epsilon^{[m]}\mu$, where $m \geq 0$ and $\mu \in L \backslash \{\epsilon\}$.*

There is a dfa $M^v = (A, Q, q_1, F, \delta)$ with alphabet $A = L \backslash \{\epsilon\}$ such that

1. *$\{P1\}$ $\mid F \mid = \mid T \mid$, and*
2. *$\{P2\}$ $\forall t \in T, \exists ! q_f \in F$ such that*
   $L(M^v \mid_{q_f}) = TDSS_n(t)$, where $M^v \mid_{q_f} = (A, Q, q_1, \{q_f\}, \delta)$.

Proof: immediate by induction on the cardinal of T.

□

Thus, the testing process described in `Section 5` becomes equivalent to actions on the collection of `dfas` $\cup_{v \in U}\{M^v\}$, where U is the set of states with *unresolved non-determinism*: the event labeling of ET at v reduces to transitions in M^v, and checking for membership to checking whether or not a *final* state has been reached. A *dfa* M^v is activated whenever state v is encountered during the testing process, and disactivated whenever a final state is reached or an unspecified transition occurs. Such *dfa*s with properties $P1$ and $P2$ can be constructed during the preprocessing phase.

7 Conclusion

Adaptive testing is essential when dealing with non-deterministic implementations. In this context, we have proposed a new transition-based identification method that provides a characterization of the different alternatives at non-deterministic states of a specification tree. Transition Distinguishing Sequence Sets (TDSSs) support testing in a non-deterministic environment as they provide a theoretical basis for a systematic resolution of non-determinism at run-time and the definition of test selection criteria for test sequence generation at run-time.

There are many directions for future research, foremost among them being to develop various adaptive test sequence selection criteria based on the intuitive notion of TDSSs for fault detection and diagnostic purposes. In addition, we shall investigate how the choice of n for *n-testers* would affect the existence of TDSSs, and how to relax the constraints on the class of specifications considered.

Finally, we plan to integrate this work into a realistic non-deterministic testing environment by providing complementary modules, and also undertake a case study, applying this method to a complete OSI protocol.

Acknowledgement

This research was supported by NSF Grant CCR-9206910 and by the New York State Science and Technology Foundation Center for Advanced Technology program.

References

[1] ISO/8807, "LOTOS- A Formal Description Technique Based on the Temporal Ordering of Observational Behavior."

[2] Michel Diaz, Chris A. Vissers, Jean-Pierre Ansart, "SEDOS, Software Environment for the Design of Open Distributed Systems," in *The Formal Description Technique Estelle*, (North-Holland), Elsevier Science Publishers B. V., 1989, pp. 3-14.

[3] P. Dembinski, S. Budkowski, "Specification Language Estelle," in *The Formal Description Technique Estelle*, (North-Holland), Elsevier Science Publishers B. V., 1989, pp. 35-75.

[4] CCITT, "Functional Specification and Description Language (SDL)," 1984.

[5] Ed Brinksma, "A Theory for the Derivation of Tests," in *Protocol Specification, Testing, and Verification VIII*, (North-Holland), Elsevier Science Publishers B. V., 1988, pp. 63-74.

[6] Clazien D. Wezeman, "The CO-OP Method for Compositional Derivation of Conformance Testers," in *Protocol Specification, Testing, and Verification IX*, (North-Holland), Elsevier Science Publishers B. V., 1990, pp. 145-158.

[7] David H. Pitt, David Freestone, "The derivation of Conformance Tests from LOTOS Specifications," *IEEE Transactions on Software Engineering*, vol. 16, December 1990, pp. 1337-1343.

[8] Susumu Fujiwara, Gregor V. Bochmann, "Testing non-deterministic state machines with fault coverage," in *Protocol Test Systems IV*, (North-Holland), Elsevier Science Publishers B. V., 1992, pp.267-280.

[9] D. Turner, H. Ural, "Modelling of expressed system behavior towards characterization of the conformance testing problem," in *Proceedings of the Seventh Annual International Phoenix Conference on Computers and Communications*, IEEE Computer Soc. Press, 1988.

[10] Tsun S. Chow, "Testing Software Design Modeled by Finite-State Machines," *IEEE Transactions on Software Engineering*, vol. SE-4, May 1978, pp. 178-187.

[11] K. Sabnani, A. Dahbura, "A Protocol Test Generation Procedure," *Computer Networks and ISDN Systems*, vol. 15, 1988, pp. 285-297.

[12] Behcet Sarikaya, "Protocol Test Generation, Trace Analysis and Verification Techniques," in *Proceedings of the Second Workshop on Software Testing, Verification, and Analysis*, IEEE Computer Soc. Press, 1988.

[13] T. Bolognesi, E. Brinksma, "An Introduction to the ISO specification Language LOTOS," *Computer Networks and ISDN Systems*, vol. 14, 1987, pp. 25-59.