



550, rue Sherbrooke Ouest, bureau 100
Montréal (Québec) H3A 1B9
Tél. : 514 840-1234; Téléc. : 514 840-1244
Place de la Cité – Tour de la Cité
2600, boul. Laurier, bureau 625
Québec (Québec) G1V 4W1
Tél. : 418 648-8080; téléc. : 418 648-8141
<http://www.crim.ca>

CRIM - Documentation/Communications

Technical Report
Incremental Test Generation Guided by Fault Coverage

Final Version

CRIM-09/03-01

Adenilso Simao

On a leave from Instituto de Ciências Matemáticas e de Computação/USP

Alexandre Petrenko

Lead Researcher and Director, Distributed Systems Analysis

March 2009

Scientific and Technic Collection

ISBN-10 : 2-89522-116-2
ISBN-13 : 978-2-89522-116-6

Pour tout renseignement, communiquer avec:

CRIM Centre de documentation

CRIM

550, rue Sherbrooke Ouest, bureau 100

Montréal (Québec) H3A 1B9

Téléphone : (514) 840-1234

Télécopieur : (514) 840-1244

Tous droits réservés © 2009 CRIM

ISBN-13 : 2-89522-116-2

ISBN-10 : 978-2-89522-116-6

Dépôt légal - Bibliothèque et Archives nationales du Québec, 2009

Dépôt légal - Bibliothèque et Archives Canada, 2009

TABLE OF CONTENTS

LIST OF FIGURES	4
1. INTRODUCTION	5
2. DEFINITIONS	6
3. TEST PROPERTIES	7
4. SUFFICIENT CONDITIONS FOR P-COMPLETENESS	10
5. ALGORITHM FOR GENERATING P-COMPLETE TEST SUITES	11
6. EXAMPLES	19
6.1 Incremental Generation	20
6.2 Confirming <i>P</i> -completeness	22
6.3 Completing User-Defined Test Suites	24
7. CONTRIBUTIONS AND RELATED WORK	26
8. CONCLUSION	27
9. REFERENCES	28

LIST OF FIGURES

<i>Figure 1 - Algorithm for Generating a P-complete Test Suite.....</i>	<i>15</i>
<i>Figure 2 - A complete FSM.</i>	<i>20</i>
<i>Figure 3 - Divergence graphs obtained during the generation of $T = \text{pref}\{\text{aaaba}, \text{baaa}, \text{bbaa}\}$</i>	<i>21</i>
<i>Figure 4 - Divergence graphs obtained during the execution with $T = \text{pref}\{\text{aaa}, \text{abb}, \text{baba}, \text{bbab}\}$</i>	<i>22</i>
<i>Figure 5 - Divergence Graphs obtained during the generation of $T = \text{pref}\{\text{a}, \text{bbabaab}, \text{bbbbaa}\}$</i>	<i>25</i>

Abstract

We consider a classical problem of complete test generation for deterministic FSMs in a more general setting. The first generalization is that the number of states in implementation FSMs can even be smaller than that of the specification FSM. This generalization provides more options to the test designer: when traditional methods trigger test explosion for large specification machines, tests with a lower, but yet guaranteed, fault coverage can still be generated. The second generalization is that tests can be generated starting with a user-defined test suite, by incrementally extending it until the desired fault coverage is achieved. To solve the generalized test derivation problem, we formulate sufficient conditions for test suite completeness weaker than the existing ones and use them to elaborate an algorithm which can be used both for extending user-defined test suites to achieve the desired fault coverage and for test generation.

1. INTRODUCTION

The problem of generating tests with fault coverage guarantee, called n -complete tests, for a specification FSM with n states, aka checking experiments and checking sequences, has traditionally been investigated only for the fault domain containing all implementation FSMs with at most n states or even higher, see, e.g., [9], [21], [2], [6], [13], [17]. An n -complete test suite guarantees to the test designer exhaustive fault coverage with respect to the given upper bound n on the number of states in implementation machines [16]. The length of n -complete tests is proportional to n^3 [21], thus their size can become unacceptably large for complex specifications. The test designer may resort to less exhaustive coverage criteria used in FSM-based testing such as state, transition, and path coverage, see, e.g., [1], [19]. Indeed, tests which satisfy these criteria scale much better than n -complete tests, but they offer no coverage guarantee in terms of the number of states in faulty implementation FSMs.

We believe that the test designer may want to be able to generate tests while retaining a (reduced) fault coverage guarantee similar to that offered by n -complete tests. More specifically, the question is how can one generate a p -complete test suite for $p < n$. A solution to this problem would provide control of the degree of test exhaustiveness by varying a maximal number of states p of faulty state machines whose detection by a p -complete test suite is guaranteed. Methods for building tests providing fault coverage with respect to a number of states in implementation FSMs smaller than that of a specification FSM are thus needed to offer to the test designer a possibility for finding a desirable compromise between the fault coverage and the size of a test suite. Clearly, an n -complete test suite is also p -complete for any $p \leq n$; however, it may well be redundant when $p < n$. Intuitively, the smaller the state number bound for which the fault coverage is guaranteed the shorter the needed tests. We are not aware of any work addressing complete test generation for the case when faulty FSMs do not necessarily have as many states as the specification FSM.

In this paper, we consider a problem of test generation in a more general setting, namely, how a user-defined test suite for a given deterministic FSM with n states which may contain just an empty sequence can be extended until it becomes p -complete, for a given $p \leq n$.

The generalization considering initial user-defined tests has practical motivations. The test designer may start test generation using approaches based on specification coverage criteria [1], use-cases [14], or test purposes [5]. Test generation can then be completed with additional tests to achieve a required level of fault coverage. A naïve approach would just ignore the existing tests and use a generation method which provides the required fault coverage. However, this approach likely results in redundant tests. However, if test generation starts with a given test suite, as proposed in this paper, both, specification and fault coverage driven approaches can in fact be employed together, i.e., it is possible to construct tests which satisfy specification as well as fault coverage criteria.

Solving the generalized test derivation problem, we present sufficient conditions for test suite completeness that are weaker than the ones known in the literature. Based on these conditions, we propose an algorithm which generates a p -complete test suite starting with user-defined initial tests if they are available. The algorithm is also able to determine whether the user-defined test suite satisfies the sufficient conditions; thus, can also be used for test analysis.

This paper is organized as follows. In Section 2, we present the necessary basic definitions. In Section 3, we define p -completeness of test suite and discuss tests convergence and divergence in a set of FSMs. These relationships are the basis for defining sufficient conditions for p -completeness in Section 4. In Section 5, an algorithm for generating p -complete test suites is elaborated and its complexity is analysed. We illustrate the algorithm in various scenarios of usage in Section 6. In Section 7 we discuss the related work. Finally, in Section 8 we present concluding remarks and point to future work.

2. DEFINITIONS

A Finite State Machine is a deterministic Mealy machine, which can be defined as follows.

Definition 1. A Finite State Machine (FSM) M is a 7-tuple $(S, s_0, I, O, D, \delta, \lambda)$, where

- S is a finite set of states with the initial state s_0 ,
- I is a finite set of inputs,
- O is a finite set of outputs,
- $D \subseteq S \times I$ is a specification domain,
- $\delta: D \rightarrow S$ is a transition function, and
- $\lambda: D \rightarrow O$ is an output function.

If $D = S \times I$, then M is a *complete* FSM; otherwise, it is a *partial* FSM. As M is deterministic, a tuple $(s, x) \in D$ determines uniquely a defined *transition* of M . For simplicity we use (s, x) to denote the transition, thus omitting its output and final state. A string $\alpha = x_1 \dots x_k$, $\alpha \in I^*$, is said to be a *defined* input sequence at state $s \in S$, if there exist s_1, \dots, s_{k+1} , where $s_1 = s$, such that $(s_i, x_i) \in D$ and $\delta(s_i, x_i) = s_{i+1}$, for all $1 \leq i \leq k$. We use $\Omega(s)$ to denote the set of all defined input sequences for state s and Ω_M as a shorthand for $\Omega(s_0)$, i.e., for the input sequences defined for the initial state of M and, hence, for M itself.

We extend the transition and output functions from input symbols to defined input sequences, including the empty sequence ϵ , as usual, assuming $\delta(s, \epsilon) = s$ and $\lambda(s, \epsilon) = \epsilon$, for $s \in S$. An FSM M is said to be *initially connected*, if for each state $s \in S$, there exists an input sequence $\alpha \in \Omega_M$, such that $\delta(s_0, \alpha) = s$, called a *transfer* sequence for state s . In this paper, only initially connected machines are considered, since any state that is not reachable from the initial state can be removed without changing the machine's behaviour. A set $C \subseteq \Omega_M$ is a *state cover* for an FSM M if for each state $s \in S$, there exists $\alpha \in C$ such that $\delta(s_0, \alpha) = s$. A state cover is *minimal* if it contains exactly one transfer sequence for each state. The set $C \subseteq \Omega_M$ *covers* a transition (s, x) if, there exists $\alpha \in C$ such that $\delta(s_0, \alpha) = s$ and $\alpha x \in C$. The set C is a *transition cover* (for M) if it covers every defined transition of M . A set of sequences is *initialized*, if it contains the empty sequence.

Two states $s, s' \in S$ are *distinguishable*, if there exists $\gamma \in \Omega(s) \cap \Omega(s')$, such that $\lambda(s, \gamma) \neq \lambda(s', \gamma)$. We say that γ distinguishes s and s' . Given a set $C \subseteq \Omega(s) \cap \Omega(s')$, states s and s' are *C-equivalent*, if $\lambda(s, \gamma) = \lambda(s', \gamma)$ for all $\gamma \in C$. We define distinguishability and C-equivalence of machines as a corresponding relation between their initial states. An FSM M is *reduced*, if all states are pairwise distinguishable. In this paper, all the FSMs are assumed to be reduced.

Given sequences $\alpha, \beta, \gamma \in I^*$, if $\beta = \alpha\gamma$, then α is a *prefix* of β , denoted by $\alpha \leq \beta$, and γ is a *suffix* of β . We also say that a prefix of γ *extends* α (in β) and that β is an *extension* of α . We denote by $pref(\beta)$ the set of prefixes of β , i.e., $pref(\beta) = \{\alpha \mid \alpha \leq \beta\}$. For a set of sequences A , $pref(A)$ is the union of $pref(\beta)$, for all $\beta \in A$. Given a sequence α and $k \geq 0$, we define α^k recursively as: $\alpha^0 = \epsilon$; $\alpha^k = \alpha\alpha^{k-1}$, if $k > 0$. The *common extensions* of two sequences are the sequences obtained by appending a common sequence to them.

3. TEST PROPERTIES

In this section, we discuss various properties of FSM tests used to formulate a test generation method. First, we formalize the notion of test suite completeness with respect to a given fault domain.

Throughout this paper, we assume that $M = (S, s_0, I, O, D, \delta, \lambda)$ and $N = (Q, q_0, I, O', D', \Delta, \Lambda)$ are a specification FSM and an implementation FSM, respectively. Moreover, n is the

number of states of M . We denote by \mathfrak{S} the set of all deterministic FSMs with the same input alphabet as M for which all sequences in Ω_M are defined, i.e., for each $N \in \mathfrak{S}$, it holds that $\Omega_M \subseteq \Omega_N$. The set \mathfrak{S} is called a *fault domain* for M . Given natural $p \leq n$, let \mathfrak{S}_p be the FSMs of \mathfrak{S} with at most p states, i.e., the set \mathfrak{S}_p is the fault domain for M which represents all faults that can occur in an implementation of M with no more than p states. Faults can be detected by tests, which are input sequences defined in the specification FSM M .

Definition 2. A defined input sequence of FSM M is called a *test case* (or simply a *test*) of M . A *test suite* of M is a finite prefix-closed set of tests of M . A given test suite T of FSM M is p -complete, $p \leq n$, if for each FSM $N \in \mathfrak{S}_p$, distinguishable from M , there exists a test in T that distinguishes them.

Since the distinguishability of FSMs is defined as the corresponding relation of their initial states, tests are assumed to be applied in the initial state. For simplicity, we assume that the empty test suite contains the empty test.

A p -completeness of a test suite provides a full *fault coverage* guarantee for the fault domain defined by the input alphabet of the specification FSM and maximal number of states p .

The rest of the paper is devoted to the problem of extending a given test suite until it becomes p -complete for a given natural $p \leq n$. The approach developed in this paper is based on intricate properties of FSM tests, namely their convergence and divergence. Two defined input sequences of an FSM converge or diverge if they take the FSM into the same or different state(s), respectively. We generalize these notions to sets of tests and sets of FSMs. Given a non-empty set of FSMs $\Sigma \subseteq \mathfrak{S}$ and two tests $\alpha, \beta \in \Omega_M$, we say that α and β are Σ -convergent, if they converge in each FSM of the set Σ . Similarly, we say that α and β are Σ -divergent, if they diverge in each FSM of Σ . We slightly abuse the notation and say that two tests are M -convergent (M -divergent) when they are $\{M\}$ -convergent ($\{M\}$ -divergent). Moreover, when it is clear from the context, we will drop the set in which tests are convergent or divergent. A set of tests is convergent (divergent), if each pair of its tests are convergent (divergent).

Test convergence and divergence with respect to a single FSM are complementary, i.e., any two tests are either convergent or divergent. However, when a set of FSMs Σ is considered, some tests are neither Σ -convergent nor Σ -divergent. Notice that the Σ -convergence relation is reflexive, symmetric, and transitive, i.e., it is an equivalence relation over the set of tests. On the other hand, the Σ -divergence relation is irreflexive and symmetric.

Several properties of test convergence and divergence can be established.

Lemma 1. *Given a non-empty set Σ of deterministic reduced FSMs with the same input alphabet, the following properties hold:*

- (i) *Common extensions of Σ -convergent tests are also Σ -convergent;*

- (ii) Tests which have Σ -divergent common extensions are also Σ -divergent; and
- (iii) Given two Σ -divergent tests, any test Σ -convergent with one of them is Σ -divergent with the other.
- (iv) If tests α and $\alpha\phi^k$ are Σ -divergent, for natural $k > 1$, then α and $\alpha\phi$ are also Σ -divergent.
- (v) If tests α and $\alpha\beta\gamma$ are Σ -convergent and tests α and $\alpha\gamma$ are Σ -divergent, then α and $\alpha\beta$ are also Σ -divergent.
- (vi) If tests α and $\alpha\gamma$ are Σ -convergent and tests β and $\beta\gamma$ are Σ -divergent, then α and β are also Σ -divergent.

Proof. Properties (i) and (ii) follow directly from the determinism of the FSMs in Σ , whereas property (iii) comes from the fact that convergence is transitive and divergence is irreflexive. For property (iv), notice that if α and $\alpha\phi$ converge in some FSM of Σ , then so do α and $\alpha\phi^2$, by (i) and the transitivity of convergence. By the same token, α and $\alpha\phi^3$, $\alpha\phi^4$, ..., $\alpha\phi^k$ would converge, a contradiction.

For property (v), suppose α and $\alpha\beta$ converge in some FSM of Σ . Then, due to Lemma 1(ii), $\alpha\gamma$ and $\alpha\beta\gamma$ would also be convergent. Thus, α and $\alpha\gamma$ would converge due to the transitivity of convergence, a contradiction.

For property (vi), suppose that α and β converge in some FSM of Σ . Then, by Lemma 1(i), $\alpha\gamma$ and $\beta\gamma$ would converge. Consequently, β and $\beta\gamma$ would also converge, a contradiction. ♦

Two tests α and β in a given test suite T are *T-distinguishable*, if there exist common extensions $\alpha\gamma, \beta\gamma \in T$, such that $\lambda(\delta(s_0, \alpha), \gamma) \neq \lambda(\delta(s_0, \beta), \gamma)$. An important property of *T-distinguishable* tests is that they are divergent in all FSMs that are *T-equivalent* to M . Given a test suite T , let $\mathfrak{S}(T)$ be the set of all $N \in \mathfrak{S}$, such that N and M are *T-equivalent*.

Lemma 2. *Given a test suite T of an FSM M , T -distinguishable tests are $\mathfrak{S}(T)$ -divergent.*

Proof. Let tests α and β be *T-distinguishable*. Thus, there exist common extensions $\alpha\gamma, \beta\gamma \in T$ and $\lambda(\delta(s_0, \alpha), \gamma) \neq \lambda(\delta(s_0, \beta), \gamma)$. Let N be an FSM *T-equivalent* to M ; thus, we have that $\lambda(\delta(s_0, \alpha), \gamma) = \Lambda(\Delta(q_0, \alpha), \gamma)$ and $\lambda(\delta(s_0, \beta), \gamma) = \Lambda(\Delta(q_0, \beta), \gamma)$. It follows that $\Lambda(\Delta(q_0, \alpha), \gamma) \neq \Lambda(\Delta(q_0, \beta), \gamma)$. Thus, $\Delta(q_0, \alpha) \neq \Delta(q_0, \beta)$. ♦

We now address the problem of demonstrating that tests are $\mathfrak{S}(T)$ -convergent. However, ensuring convergence is more involved than ensuring divergence. Divergence of two tests can be witnessed by different outputs produced by the tests, which are thus divergent in any FSM *T-equivalent* to M , while convergence of two tests cannot be directly ascertained. However, it can be shown that the two tests are $\mathfrak{S}(T)$ -divergent with tests to all but one state

of the FSM M , implying that they must converge in that state. Thus, a maximal number of states of FSMs in the fault domain has to be known. Given a test suite T , let $\mathfrak{S}_n(T) = \mathfrak{S}_n \cap \mathfrak{S}(T)$, i.e., the set of FSMs in \mathfrak{S} which are T -equivalent to M and have at most n states. Below we consider only the $\mathfrak{S}_n(T)$ -convergence, instead of the $\mathfrak{S}(T)$ -convergence. In particular, we show how the $\mathfrak{S}_n(T)$ -convergence of tests can be established based on the existence of an $\mathfrak{S}_n(T)$ -divergent set with n tests. Notice that, while $\mathfrak{S}(T)$ -divergent tests are also $\mathfrak{S}_n(T)$ -divergent, the converse does not hold, i.e., there are $\mathfrak{S}_n(T)$ -divergent tests which are not $\mathfrak{S}(T)$ -divergent. For instance, Lemma 1 can be used to establish the $\mathfrak{S}_n(T)$ -divergence of tests from the $\mathfrak{S}_n(T)$ -divergence and $\mathfrak{S}_n(T)$ -convergence of other tests, but cannot determine their $\mathfrak{S}(T)$ -divergence, which requires that the tests in question are T -distinguishable.

Lemma 3. *Given a test suite T and $\alpha \in T$, let K be an $\mathfrak{S}_n(T)$ -divergent set with n tests and $\beta \in K$ be a test M -convergent with α . If α is $\mathfrak{S}_n(T)$ -divergent with each test in $K \setminus \{\beta\}$, then α and β are $\mathfrak{S}_n(T)$ -convergent.*

Proof. Let $K' = K \setminus \{\beta\}$. The set K' is an $\mathfrak{S}_n(T)$ -divergent set, thus it reaches $n - 1$ states of N . As both α and β are $\mathfrak{S}_n(T)$ -divergent with each test in K' , in any FSM of $\mathfrak{S}_n(T)$, both α and β reach a state which is not reached by the tests in K' . As K' reaches $n - 1$ states and any FSM in $\mathfrak{S}_n(T)$ has at most n states, α and β must reach the same state, i.e., they are $\mathfrak{S}_n(T)$ -convergent. ♦

In the next section, we use test divergence and convergence properties to formulate conditions which ensure p -completeness of test suites.

4. SUFFICIENT CONDITIONS FOR P-COMPLETENESS

In this section, we present sufficient conditions for test completeness with respect to the fault domain \mathfrak{S}_p , where each FSM has at most p states. These conditions are used to elaborate a generation method in the next section.

The conditions for p -completeness of a test suite T can be divided into two cases, depending on whether $p < n$ or $p = n$. If $p < n$, then it is sufficient show that no FSM in \mathfrak{S}_p is T -equivalent to M , i.e., that $\mathfrak{S}_p(T)$ is empty. However, if $p = n$, $M \in \mathfrak{S}_n$ and, thus, $\mathfrak{S}_n(T)$ is by definition not empty. To formulate the conditions for dealing with the case of $p = n$, we introduce the notion of convergence-preserving set, for which the M -convergence implies the $\mathfrak{S}_n(T)$ -convergence.

Definition 3. *Given a test suite T of an FSM M , a set of tests is $\mathfrak{S}_n(T)$ -convergence-preserving (or, simply, convergence-preserving) if all its M -convergent tests are $\mathfrak{S}_n(T)$ -convergent.*

Notice that any M -divergent set is, by definition, convergence-preserving.

Theorem 1. *Let T be a test suite for an FSM M with n states and $p \leq n$. T is a p -complete test suite for M if*

- (i) $p < n$ and T contains a $\mathfrak{S}(T)$ -divergent set with $p + 1$ tests; or
- (ii) $p = n$ and T contains an $\mathfrak{S}_n(T)$ -convergence-preserving initialized transition cover for M .

Proof.

(i) If T contains an $\mathfrak{S}(T)$ -divergent set with $p + 1$ tests, then any FSM T -equivalent to M has $p + 1$ states. As there exists no such FSM in $\mathfrak{S}_p(T)$, it follows that the test suite T is p -complete.

(ii) Assume now that T contains an $\mathfrak{S}_n(T)$ -convergence-preserving initialized transition cover K for M and $p = n$. We prove by contradiction that T is n -complete. Suppose that T is not n -complete. Thus, there exists an FSM $N \in \mathfrak{S}_n(T)$ distinguishable from M . Let φx be a shortest input sequence distinguishing N and M , where x is an input symbol; hence $\lambda(\delta(s_0, \varphi), x) \neq \Lambda(\Delta(q_0, \varphi), x)$. We show, by induction on the length of φ , that there exists a test in K which is N -convergent with φ . In the base case, we have that φ is the empty sequence. As K includes this sequence, the result follows. The inductive hypothesis is that $\varphi = \beta y$, for some input sequence β and input symbol y , such that β is N -convergent with some test π in K . We have that there exists a test υ in K , such that υ and π are M -convergent and $\upsilon y \in K$, since K is a transition cover. As K is $\mathfrak{S}_n(T)$ -convergence-preserving and $\upsilon, \pi \in K$, it follows that υ and π are $\mathfrak{S}_n(T)$ -convergent. As $N \in \mathfrak{S}_n(T)$, we have that υ and π are N -convergent. Thus, so are υ and β . By Lemma 1(i), we have that υy and βy are also N -convergent, and the result follows.

Let χ be a test in K which is N -convergent with φ . As K is a transition cover for M , there exists $\alpha \in K$ such that α and χ are M -convergent and $\alpha x \in K$. As K is $\mathfrak{S}_n(T)$ -convergence-preserving, α and χ are $\mathfrak{S}_n(T)$ -convergent; hence α and χ are N -convergent, since $N \in \mathfrak{S}_n(T)$. It follows that $\lambda(s_0, \alpha x) = \lambda(\delta(s_0, \alpha), x) = \lambda(\delta(s_0, \chi), x) = \lambda(\delta(s_0, \varphi), x) \neq \Lambda(\Delta(q_0, \varphi), x) = \Lambda(\Delta(q_0, \chi), x) = \Lambda(\Delta(q_0, \alpha), x) = \Lambda(q_0, \alpha x)$, i.e., $\lambda(s_0, \alpha x) \neq \Lambda(q_0, \alpha x)$. Thus, αx distinguishes M and N , and, as $\alpha x \in K \subseteq T$, we can conclude that M and N are T -distinguishable, a contradiction. Thus, T is n -complete. ♦

In the next section, the proposed conditions are used to elaborate an algorithm for extending a given (possibly empty) test suite until it becomes p -complete.

5. ALGORITHM FOR GENERATING P-COMPLETE TEST SUITES

In this section, we present an algorithm for generating p -complete test suites based on Theorem 1 and Lemmas 1-3. Given an FSM M , a (possibly empty) test suite T and a natural

$p \leq n$, the algorithm generates a test suite which contains T and satisfies the conditions of Theorem 1, thus the resulting test suite is p -complete. The tests in T are analysed, so that more tests are added only if needed. Depending on the value of p , it is sufficient to do so until the test suite has either an $\mathfrak{S}(T)$ -divergent set with $p + 1$ tests or an $\mathfrak{S}_n(T)$ -convergence-preserving initialized transition cover.

Notice that an $\mathfrak{S}(T)$ -divergent set corresponds to a clique in a graph which represents the $\mathfrak{S}(T)$ -divergence relation. A *divergence* graph on the tests in T , is a graph such that two tests $\alpha, \beta \in T$ are adjacent, if α and β are $\mathfrak{S}(T)$ -divergent. Thus, an $\mathfrak{S}(T)$ -divergent set corresponds a clique in a divergence graph. Notice that a divergence graph is n -partite, since M -convergent tests are not adjacent. As a clique in an n -partite graph has at most n vertices, an $\mathfrak{S}(T)$ -divergent set has no more than n tests. Then, if $p < n$, to obtain a p -complete test suite, it is sufficient to guarantee that the corresponding divergence graph contains a clique of size $p + 1$.

If $p = n$, there exists no $\mathfrak{S}(T)$ -divergent set with $n + 1$ tests. In this case it is required to ensure the existence of an initialized transition cover which is $\mathfrak{S}_n(T)$ -convergence-preserving. Recall that convergence of some tests is implied by divergence and/or convergence of other tests, according to Lemma 1. Thus, the $\mathfrak{S}_n(T)$ -convergence and $\mathfrak{S}_n(T)$ -divergence relations should be determined incrementally. To this end, we define two relations C and D to represent, respectively, the subsets of $\mathfrak{S}_n(T)$ -convergence and $\mathfrak{S}_n(T)$ -divergence relationships which are already identified. Initially, the relation C is the identity relation, representing the fact that initially no $\mathfrak{S}_n(T)$ -convergence relationships are known, except for the trivial reflexive relationships. On the other hand, the relation D is initially the set of all pairs of T -distinguishable tests according to Lemma 2. These relations are iteratively updated by applying a set of rules which infer new relationships from existing relationships, following Lemma 1. The rules are event-driven, in the sense that they are applied when some relationship is added to C or D . More than one rule can be applicable at the same time.

We derive these rules from Lemma 1 as follows.

Rule 1: If (α, β) is added to C , for each $(\alpha, \chi) \in C$, add (β, χ) to C . (Transitiveness)

Rule 2: If (α, β) is added to C , then for all their common extensions $\alpha\phi, \beta\phi \in T$, add $(\alpha\phi, \beta\phi)$ to C . (Lemma 1(i))

Rule 3: If (α, β) is added to D , and they are common extensions of tests α' and β' , add (α', β') to D . (Lemma 1(ii))

Rule 4: If (α, β) is added to C , then for each $\chi \in T$, if $(\alpha, \chi) \in D$, add (β, χ) to D ; if $(\beta, \chi) \in D$ add (α, χ) to D . (Lemma 1(iii))

Rule 5: If (α, β) is added to D , then for each $\chi \in T$, if $(\alpha, \chi) \in C$, add (β, χ) to D ; if $(\beta, \chi) \in C$, add (α, χ) to D . (Lemma 1(iii))

Rule 6: If (α, β) , $\alpha \leq \beta$, is added to D , and there exists sequence φ and a natural $k > 1$, such that $\beta = \alpha\varphi^k$, then, add $(\alpha, \alpha\varphi)$ to D . (Lemma 1(iv))

Rule 7: If $(\alpha, \alpha\beta\gamma)$ is added to C , and $(\alpha, \alpha\gamma) \in D$, then add $(\alpha, \alpha\beta)$ to D . (Lemma 1(v))

Rule 8: If $(\alpha, \alpha\gamma)$ is added to D , then, for each sequence β , such that $(\alpha, \alpha\beta\gamma) \in C$, add $(\alpha, \alpha\beta)$ to D . (Lemma 1(v))

Rule 9: If $(\alpha, \alpha\gamma)$ is added to C , then, for each sequence β , such that $(\beta, \beta\gamma) \in D$, add (α, β) to D . (Lemma 1(vi))

Rule 10: If $(\beta, \beta\gamma)$ is added to D , then, for each sequence α , such that $(\alpha, \alpha\gamma) \in C$, add (α, β) to D . (Lemma 1(vi))

If to achieve p -completeness, more tests are added to T , the relations C and D should also be extended with the new tests. Recall that a test suite is prefix-closed. Thus, adding a test α to T results in the addition of all prefixes of α . If α is added to T , the identity pair (α, α) has to be added to C . Moreover, the test pairs which are T -distinguishable in the extended test suite must be added to D .

Initially, the algorithm finds a largest $\mathfrak{S}(T)$ -divergent set, which corresponds to a clique in the divergence graph. If the determined clique has more than $p + 1$ tests, then the test suite is already p -complete. Recall however that no clique in an n -partite graph has more than n tests. Thus, when the clique has fewer than $\min(p + 1, n)$ tests, a test which is not in the clique may be selected to extend it. It is possible to do so if the test to be added is $\mathfrak{S}(T)$ -divergent with all tests in the clique. Hence, if the test is not $\mathfrak{S}(T)$ -divergent with some test in the clique, it is sufficient to add tests to T , so that the two tests become T -distinguishable. If $p = n$, an n -clique can thus be eventually obtained, but this is not sufficient for ensuring the n -completeness, according to Theorem 1. In this case, it is additionally required to ensure that T contains an initialized transition cover which is $\mathfrak{S}_n(T)$ -convergence-preserving.

We now show how such a transition cover can be obtained from an n -clique. As the relation C is an equivalence relation, it induces a partition on the tests in T . Given a test $\alpha \in T$, let $C(\alpha) = \{\beta \mid (\alpha, \beta) \in C\}$ be the block of the partition induced by C which contains α . Let K be an n -clique. We denote by $C_{\cup}(K)$ the union of the blocks which have a test in K , i.e., $C_{\cup}(K) = \{\beta \mid (\alpha, \beta) \in C, \alpha \in K\}$. Recall that in an $\mathfrak{S}_n(T)$ -divergent set, no tests are M -convergent, i.e., an $\mathfrak{S}_n(T)$ -divergent set is trivially $\mathfrak{S}_n(T)$ -convergence-preserving. Thus, the set of tests $C_{\cup}(K)$ is $\mathfrak{S}_n(T)$ -convergence-preserving. To ensure that $C_{\cup}(K)$ is an initialized transition cover for M , we might need to extend it. We say that a test α is added to $C_{\cup}(K)$, when (α, β) is added to C , where $\beta \in C_{\cup}(K)$ is a test $\mathfrak{S}_n(T)$ -convergent with α . Lemma 3

indicates that a test can be added to $C \cup (K)$ if it is $\mathfrak{S}_n(T)$ -divergent with $n - 1$ tests in K . It is sufficient to show that the test which is not in $C \cup (K)$ is $\mathfrak{S}_n(T)$ -divergent with the $n - 1$ tests of the clique. If the tests form a pair in D , then they are already $\mathfrak{S}_n(T)$ -divergent. Otherwise, tests could be added so that the two tests become T -distinguishable and, thus, $\mathfrak{S}_n(T)$ -divergent. The set $C \cup (K)$ resulting from the addition of (α, β) to C remains $\mathfrak{S}_n(T)$ -convergence-preserving. Therefore, to obtain an n -complete test suite, it is sufficient to add suitable tests to $C \cup (K)$ until it becomes an initialized transition cover for M .

Depending on the tests which are already in the sets $C \cup (K)$ and T , there are three cases to consider. The first case occurs when tests can be added to $C \cup (K)$ without adding tests to T , i.e., when there are tests which already satisfy the condition of Lemma 3. As a result, the number of blocks in the partition induced by C is decreased, since the blocks to which these tests belong are merged in the resulting partition. It is important to note that this case may result in $C \cup (K) = T$. Thus, if T is also a transition cover for M (recall that T is prefix-closed and, thus, initialized), then T is n -complete.

In the remaining cases, adding tests to $C \cup (K)$ requires new tests be first added to T , making Lemma 3 applicable. If the empty sequence is not in $C \cup (K)$, tests are added so that the empty sequence can be added to $C \cup (K)$. Then, $C \cup (K)$ becomes initialized. Finally, if there is a transition not covered by $C \cup (K)$ a test is added to $C \cup (K)$, so that it becomes covered. Thus, $C \cup (K)$ eventually becomes a transition cover. As it is also initialized and $\mathfrak{S}_n(T)$ -convergence-preserving, by Theorem 1, T is p -complete.

The above discussion leads to the algorithm, presented in Figure 1, for extending a test suite until its p -completeness can be guaranteed. Labels on edges connecting steps and conditions, φ and χ , denote the tests defined in a precedent step or condition.

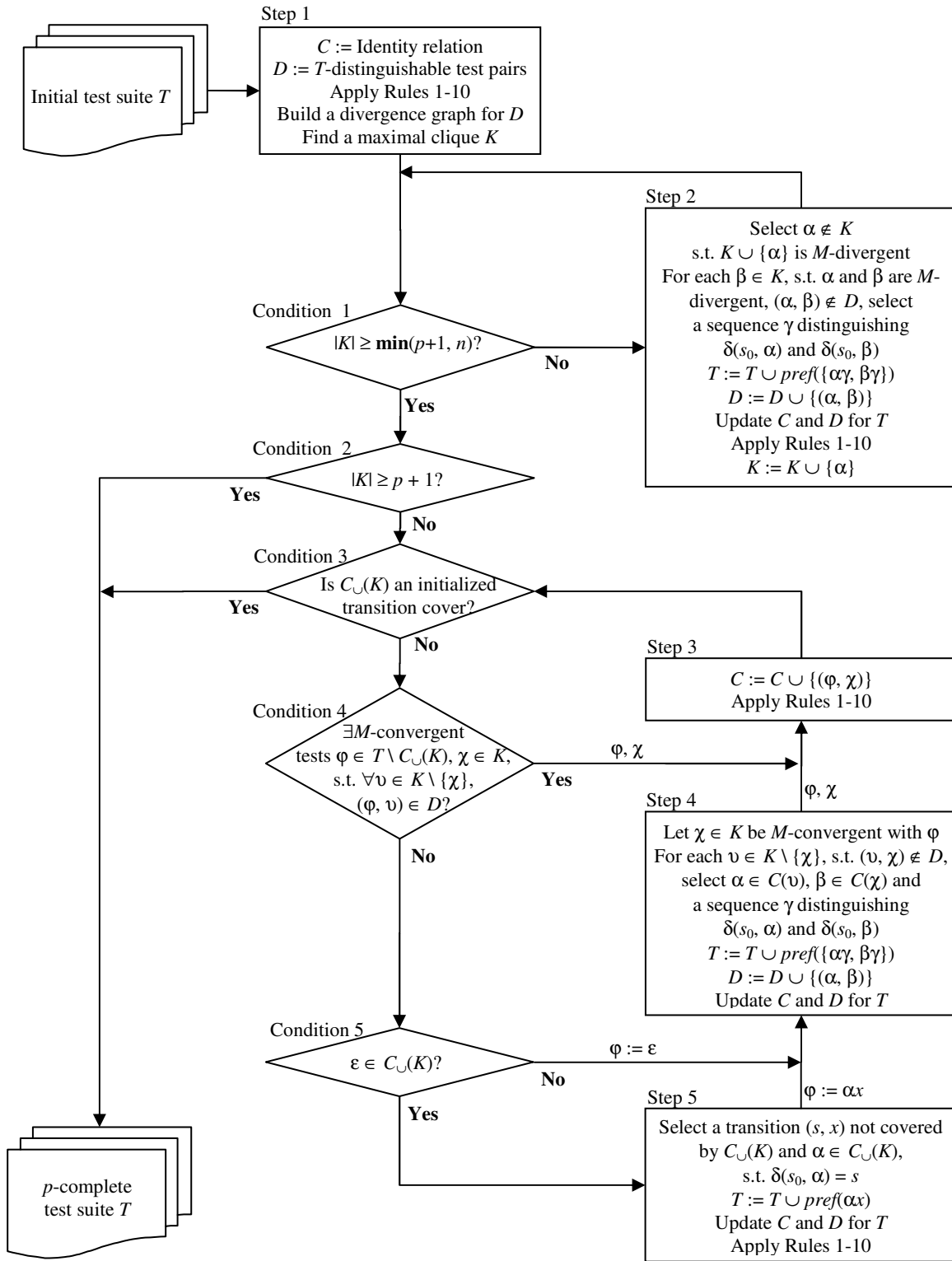


Figure 1 - Algorithm for Generating a P-complete Test Suite

We illustrate the algorithm in Section 6. It is important to note that in several steps of the algorithm we do not restrict the selection of tests with the required properties. Various selection strategies can be used there, for instance, the distinguishing sequences selected in Step 4 can be obtained from identification sets obtained a priori, as in the methods W [21], [2] and Wp [6], or *on-the-fly*, as in H-method [3]. Moreover, the sequences needed to reach a state (in Step 2) or to cover a transition (in Step 5) can be selected using different strategies, such as a breadth-first traversal of the FSM or a transition tour. We believe that several alternative selection strategies should become options in a tool implementing the proposed algorithm for constructing complete tests for FSMs.

In the remaining of this section, we prove that the algorithm terminates and the obtained test suite is indeed p -complete. This discussion is independent from the strategies for sequence selection. Then, we show that the algorithm can be executed in polynomial time, if the strategies used to select sequences can also be executed in polynomial time.

Theorem 2. *The algorithm terminates with a p -complete test suite for M .*

Proof. The algorithm contains four cycles. We show that each cycle can be executed a finite number of times and, thus, the algorithm indeed terminates. Then, we prove that the resulting test suite is p -complete.

In the cycle which contains Step 2, the size of the clique is increased in each iteration, until the required size is reached. Thus, this cycle can only be executed a finite number of times. The other cycles correspond to the three cases discussed above. At the end, in the execution of each cycle, (φ, χ) is added to C and Rules 1-10 are applied as long as possible, in Step 3. The steps which precede Step 3 guarantee that φ and χ are $\mathfrak{S}_n(T)$ -convergent. For instance, if necessary, Step 4 adds tests to T so that Lemma 3 can be applied.

Cycle 1 corresponds to the executions where Condition 3 does not hold, but Condition 4 does, i.e., $C \cup (K)$ is not an initialized transition cover and there exists a test which can be added to $C \cup (K)$ without adding new tests to T . As the number of tests in $C \cup (K)$ is increased in each execution of this cycle and the number of tests in T is not changed, after a finite number of executions, the cycle can no longer be executed without involving other cycles. Notice that those cycles add new tests to T , possibly increasing the number of blocks. However, as it will be shown, those cycles also can be executed a finite number of times and, thus, the number of executions of Cycle 1 is bounded.

Cycle 2 corresponds to the executions where Conditions 3, 4 and 5 do not hold, i.e., the empty sequence is not in $C \cup (K)$. Then, the empty sequence is added to $C \cup (K)$, and thus, this cycle can be executed at most once.

Cycle 3 corresponds to the executions where Conditions 3 and 4 do not hold, but Condition 5 does, i.e., $C \cup (K)$ is initialized but is not a transition cover. Step 5 selects a transition (s, x) which is not covered by $C \cup (K)$ and a test $\alpha \in C \cup (K)$, $\delta(s_0, \alpha) = s$. The test αx is added to T . Then, Step 4 adds tests to T so that αx is added to $C \cup (K)$ and, thus, the transition (s, x) be-

comes covered by $C_{\cup}(K)$. Therefore, each execution of this cycle requires that a transition not covered by $C_{\cup}(K)$ exists and it results in covering of at least one transition. This cycle can thus be executed at most as many times as the number of transitions of M .

Therefore, the algorithm actually terminates, since all cycles can be executed only a finite number of times.

We now show that the obtained test suite is p -complete. When the algorithm terminates, either Condition 2 or Condition 3 holds. If Condition 2 holds, the clique has $p + 1$ tests, then the test suite contains an $\mathfrak{S}(T)$ -divergent set with $p + 1$ tests and, thus is p -complete, by Theorem 1. If Condition 3 holds, the set $C_{\cup}(K)$ is an initialized transition cover for M . As $C_{\cup}(K)$ is $\mathfrak{S}_n(T)$ -convergence-preserving, by Theorem 1, the resulting test suite T is p -complete. ♦

We now discuss the worst-case time complexity of the algorithm and the upper bounds of p -complete test suites. In particular, we show that the algorithm terminates in polynomial time.

Initially, the algorithm needs to find a maximal clique in an n -partite graph. This problem is known to be NP-complete and, thus, an optimal solution cannot be found in a reasonable time for some instances. Nonetheless, the algorithm does not rely on the fact that the clique found is a largest one. Indeed, if a suboptimal clique is found, it will be extended to the required size by adding new tests to create T -distinguishability relationships omitted when a sub-clique is chosen. Thus, it is always possible to reduce time needed to find a largest clique at a price of increasing the test suite. In other words, the NP-completeness of the maximal clique problem does not imply that the proposed algorithm does not scale. For instance, polynomial time greedy-based algorithms for finding cliques can be used; see, [11] and [8]; optimization techniques have also been used to solve this problem, which can handle very large graphs in reasonable time [10] [22]. Nevertheless, even in the worst case when the maximal clique found is smaller than a largest one, a complete test suite can be obtained, though its irreducibility might be hard to claim.

In Steps 2 and 5, the algorithm requires that tests are added to obtain divergence relationships. Specifically, given two tests, it is necessary to select a sequence which distinguishes the states reached by them. This problem can be solved by a breadth-first search in a product machine, as defined in [17], in $O(v + w)$, where v and w are the numbers of vertices and edges in the graph, respectively. The product machine has at most n^2 vertices and kn^2 edges, where n is the number of states and k is the number of inputs of M . Thus, the time required to find shortest distinguishing sequences is $O(n^2 + kn^2) = O(kn^2)$ [13].

The application of Rules 1-10 is event driven, in the sense that the rules are applied when new relationships are added to C or D . Thus, they are applied at most once for each pair of tests. Let l be the number of tests in the test suite obtained by the algorithm. Thus, there are $O(l^2)$ pairs of tests. As relation C is an equivalence relation, it can be represented by the

partition it induces. Using a *union-find* algorithm, the time for performing the operations on the partition is $O(\text{Ack}^{-1}(l, l))$, where $\text{Ack}^{-1}(l, l)$ is the inverse of the extremely quickly-growing Ackermann function. For any reasonable value of l , $\text{Ack}^{-1}(l, l)$ is less than five, i.e., the running time of the operations on C is effectively a small constant [7].

As the execution of the algorithm advances, the size of the relation D approximates $l(l - 1)/2$. Thus, we represent this relation in a symmetrical matrix, so that the operations on D can be performed in constant time, at the price of using $O(l^2)$ space.

We now discuss the complexity of executing each rule. Based on the discussion above, we assume that the operation of verifying whether a pair is in C or in D can be completed in constant time. Thus, we show that all the rules can be executed in $O(l)$ time. Rule 1 enforces the transitivity of relation C , which, in the worst case, requires analysing all tests. This can be done in $O(l)$. Rule 2 requires identification of common extensions of two tests, which can be achieved, in the worst case, by inspecting all tests, i.e., in $O(l)$. Similarly, Rule 3 can be executed by checking whether the tests in a pair added to D are common extensions of their prefixes. In the worst case, the time required for Rule 3 is the number of prefixes of the longest test, which is $O(l)$ when the test suite contains a single test. Both Rules 4 and 5 require the inspection of all tests, i.e., in $O(l)$. Rules 6-10 are applicable if one test is a prefix of the other, which can be checked in $O(l)$. For Rules 6 and 7, it is sufficient to check if the suffix is in an appropriate form. Rules 8-10 require inspecting all the tests; thus, $O(l)$ time.

As there are $O(l^2)$ pairs and each may need operations with $O(l)$ time, the worst case complexity time for applying Rules 1-10 is $O(l^3)$. It is important to note that, although the rules are applied in various steps executed several times, each pair is analysed at most once, when it is added to the respective relation.

The algorithm contains four cycles. In the cycle which contains Step 2, the size of the clique is increased in each iteration, until the required size is reached. In the worst case, the cycle can be executed n times. As Step 2 requires finding $n - 1$ distinguishing sequences, in the worst case, the execution time of this cycle is $O(n^2)O(kn^2) = O(kn^4)$.

The cycle where Condition 4 holds can be executed at most $l - n$ times, i.e., $O(l)$. Condition 4 requires inspecting n tests in K and at most $l - n$ tests in $T \setminus C_{\cup}(K)$, totalling at most $O(nl)$ pairs. For each pair, the other $n - 1$ tests in K are analysed. Thus, the execution time of Condition 4 is $O(n^2l)$, and the cycle which contains it requires time $O(n^2l^2)$.

The other cycles require the execution of Step 4, which finds $n - 1$ distinguishing sequences; thus, its execution time is $O(n)O(kn^2) = O(kn^3)$. This step is involved in two cycles, which can be executed at most as many times as the number of defined transitions, i.e., $O(kn)$. Thus, these cycles are executed in $O(kn)O(kn^3) = O(k^2n^4)$.

The cost of the algorithm is thus $O(CLIQUE) + O(n^2l^2 + l^3 + k^2n^4)$, where $O(CLIQUE)$ is the time required by the algorithm chosen for finding a maximal clique. Thus, the algorithm runs in polynomial time, after a maximal clique has been found.

The algorithm proposed in this paper can generate p -complete test suites even when $p < n$. The authors are not aware of other methods with such property. It allows the test designer to find a compromise between the cost of complete tests and the size of the fault domain where the completeness is guaranteed. For instance, if n -complete test suites are too expensive to be used, the test designer may choose using, say, $(n/2)$ -complete test suites, which nonetheless ensures that if any faulty implementation has at most $n/2$ it will be caught by the test.

Finally, we discuss the upper bounds of p -complete test suites when $p < n$. For $p = n$, it is known that such there is an n -complete test suite with at most $O(kn^3)$ inputs, for complete FSMs [21] or $O(kn^4)$ inputs, for reduced partial FSMs [13]. For $p < n$, in the worst case, a p -complete test suite contains $p + 1$ tests, reaching $p + 1$ distinct states, and each pair of states requires a distinct distinguishing sequence. Thus, a p -complete test suite has at most $p(p + 1)$ tests. Any state in an initially connected FSM can be reached by a test no longer than $n - 1$. In a complete FSM, any pair of states can be distinguished by a sequence of at most $n - 1$ inputs. Thus, there is a p -complete test suite for a complete FSM with at most $p(p + 1)2(n - 1)$ inputs, i.e., $O(p^2n)$. In a partial FSM, any pair of distinguishable states can be distinguished by a sequence no longer than $n(n - 1)/2$ [17]. There is thus a p -complete test suite for a partial reduced FSM with at most $p(p + 1)(n - 1 + n(n - 1)/2) = p(p + 1)((n + 2)(n - 1)/2)$ inputs, i.e., $O(p^2n^2)$. Therefore, when $p < n$, the upper bounds for p -complete test suites are lower than those for n -complete test suites by a factor of $O((p/n)^2)$. It is important to note that reasonable choices have to be made when sequences are selected for the algorithm to obtain a test suite not exceeding these bounds. For example, a longer test suite would be obtained if distinguishing sequences selected in Step 2 are not the shortest ones (e.g., longer than $n - 1$).

6. EXAMPLES

In this section we present examples of the execution of the algorithm. In the first example, the algorithm generates a series of test suites with the increasing fault coverage for various values of p . In the second example, the algorithm is given a test suite which already has the desired fault coverage. As expected, the algorithm terminates without adding new tests, even though the test suite does not satisfy the existing sufficient conditions. This demonstrates the fact that the proposed conditions are weaker and the algorithm improves the state-of-the art in test coverage analysis. Finally, in the last example, we illustrate the algorithm can be used to extend a given test suite until complete fault coverage is achieved.

In the examples, we consider the FSM M , whose initial state is depicted in bold in Figure 2 [3].

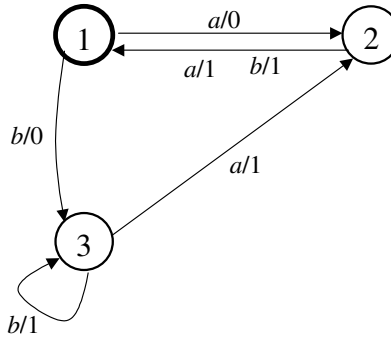


Figure 2 - A complete FSM.

6.1 Incremental Generation

We consecutively execute the algorithm to obtain p -complete test suites T_p , $p = 1, 2, 3$. Initially a test suite contains only the empty sequence, $T_0 = \{\epsilon\}$.

Case $p = 1$. As the divergence graph has just one vertex, it has a 1-clique $K = \{\epsilon\}$. We have that Condition 1 does not hold. Then, in Step 2, a test $\alpha = a$ is added to T , which reaches a new state and, we select the sequence $\gamma = a$ to distinguish $\delta(s_0, a)$ and $\delta(s_0, \epsilon)$. We then add aa to T , resulting in a 2-clique $K = \{\epsilon, a\}$. The resulting test suite $T_1 = \text{pref}(aa)$ is 1-complete

Case $p = 2$. We now want to obtain a 2-complete test suite, starting the algorithm with T_1 . The algorithm finds the 2-clique $K = \{\epsilon, a\}$. We have that Condition 1 does not hold. A test $\alpha = b$ which reaches a new state is added in Step 2. Then, tests are added to T so that b and ϵ , as well as b and a , are T -distinguishable. For b and ϵ , we add ba to T . For b and a we add aaa and baa . Then, the clique is extended with b , resulting in a 3-clique $K = \{\epsilon, a, b\}$. The final test suite $T_2 = \text{pref}(\{aaa, baa\})$ is 2-complete.

Case $p = 3$. Finally, we execute the algorithm to obtain a 3-complete test suite. The test suite T_2 is used to initialize the algorithm and the clique $K = \{\epsilon, a, b\}$ is found. We have that Condition 1 holds, but Condition 2 does not. Then, it is necessary to add tests to T to obtain an $\mathfrak{S}_n(T)$ -convergence-preserving initialized transition cover. The relation D is represented in the divergence graph in Figure 3(a). We use $\Upsilon(C)$ to represent the partition induced by the relation C . In this case, $\Upsilon(C) = \{\{\epsilon\}, \{a\}, \{aa\}, \{aaa\}, \{b\}, \{ba\}, \{baa\}\}$.

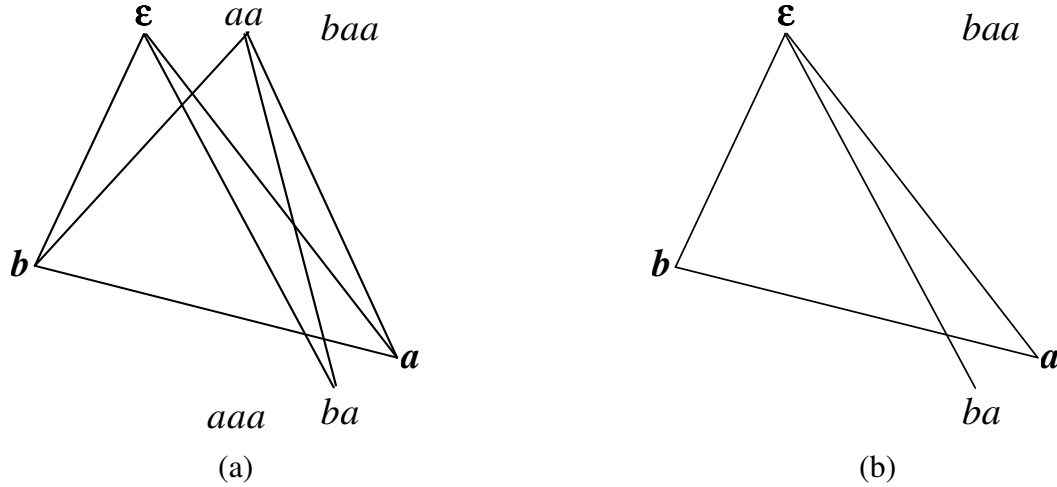


Figure 3 - Divergence graphs obtained during the generation of $T = \text{pref}(\{aaaba, baaa, bbaa\})$

Condition 3 holds for $\varphi = aa$ and $\chi = \varepsilon$. Then, after execution of Step 3, we obtain $\Upsilon(C) = \{\{\varepsilon, aa\}, \{a, aaa\}, \{b\}, \{ba\}, \{baa\}\}$ and the divergence graph in Figure 3(b). For simplicity, only one test per block is shown in this and the following divergence graphs, since the relationships of the omitted tests can be inferred.

Condition 4 does not hold, but Condition 5 does. Then, we select $\alpha = b$ and $x = a$ and execute Steps 3, 4 and 5 for $\varphi = ba$ and $\chi = a$. For $v = \varepsilon$, no additional tests are necessary. However, for $v = b$, we add the test $baaa$ to T , so that b and ba are T distinguishable. Then, after adding $(ba$ and $a)$ to C and applying Rules 1-10, we obtain $\Upsilon(C) = \{\{\varepsilon, aa, baa\}, \{a, aaa, ba, baaa\}, \{b\}\}$.

As $C_{\cup}(K)$ is not a transition cover, Step 5 is executed, extending T to cover a yet uncovered transition. We select the transition $(2, b)$, the tests $\alpha = aaa$, $x = b$ and execute Steps 3 and 4 for $\varphi = aaab$ and $\chi = \varepsilon$. For $v = a$, we add the test $aaaba$ to T . For $v = b$, it is not necessary to add new tests, since b and $aaab$ are already T -distinguishable. The application of Rules 1-10 results in the partition $\Upsilon(C) = \{\{\varepsilon, aa, baa, aaab\}, \{a, aaa, ba, baaa, aaaba\}, \{b\}\}$.

The algorithm continues to cover the transition $(3, b)$. In Step 5, we select the test bb to be added to T . Then, Steps 4 and 5 are executed for $\varphi = bb$ and $\chi = b$. The test $bbaa$ is added to T . The resulting partition is $\Upsilon(C) = \{\{\varepsilon, aa, baa, aaab, bbba\}, \{a, aaa, ba, baaa, aaaba, bba\}, \{b, bb\}\}$. As $C_{\cup}(K)$ an initialized transition cover, T is 3-complete. The resulting test suite is $T = \text{pref}(\{aaaba, baaa, bbaa\})$, of length 16, among them three resets.

The example shows that the algorithm allows generating tests which require fewer resets than the existing methods. In particular, the Wp-metod [6] and H-method [3] generate the same test suite $T_H = \text{pref}(\{aaa, aba, baaa, bbaa\})$ of length 18, which requires four resets.

6.2 Confirming P -completeness

We illustrate the execution of Algorithm 1 with the FSM in Figure 2, initial test suite $T = \text{pref}(\{aaa, abb, baba, bbab\})$ and $p = n = 3$. We show that T is indeed an n -complete test suite for M , without adding more tests. Notice that the n -completeness of T cannot be established using the conditions proposed in [3] and [18].

Initially, C is the identity relation. After populating D with the T -distinguishable tests (Rules 1-10 are not applicable), we obtain the divergence graph in Figure 4(a).

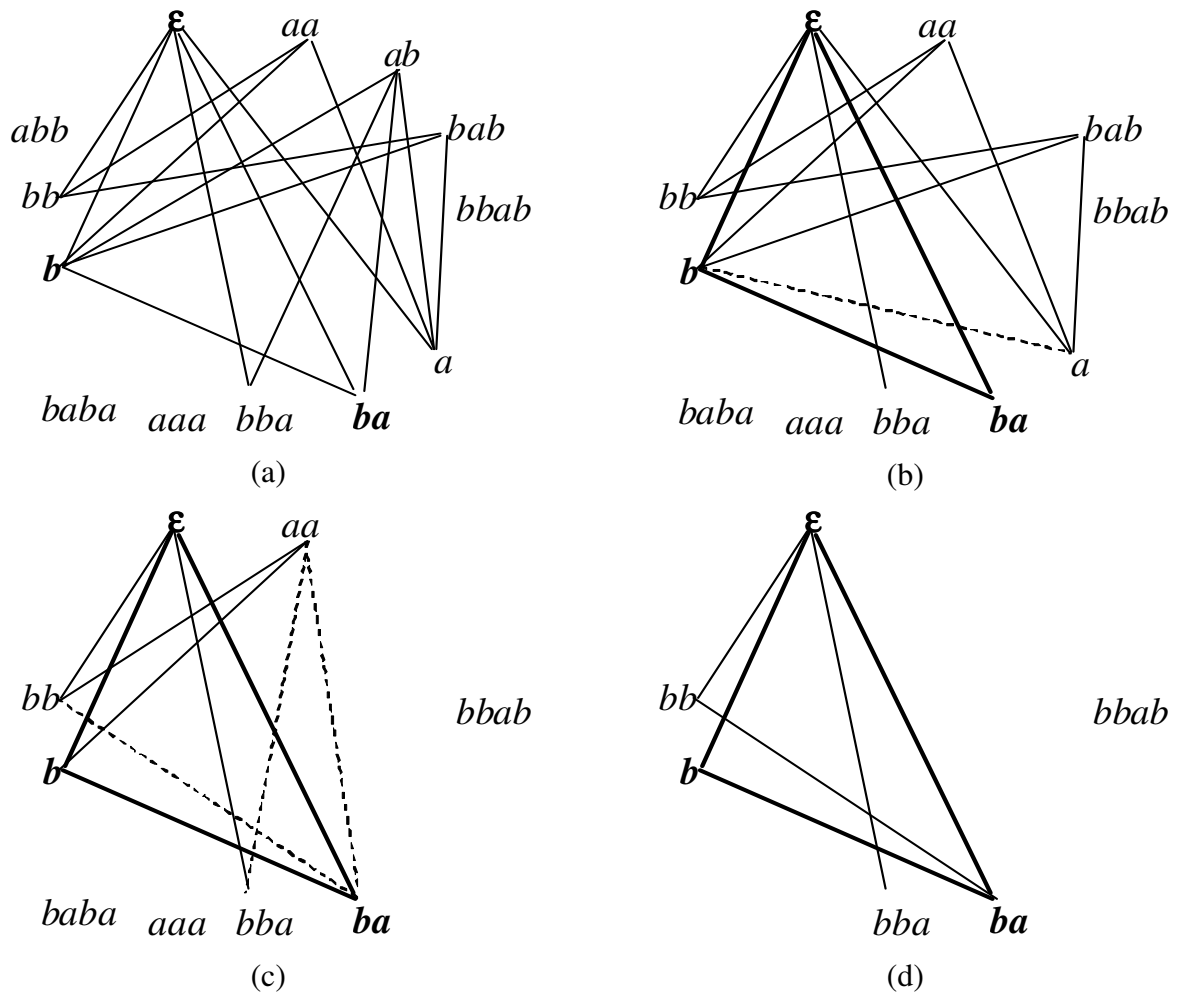


Figure 4 - Divergence graphs obtained during the execution with $T = \text{pref}(\{aaa, abb, baba, bbab\})$

As Condition 4 holds for $\chi = \varepsilon$ and $\varphi = ab$, Step 3 is executed, adding ab to $C \cup (K)$. After applying Rules 1-10, the following relationships are determined:

- (b, abb) is added to C (Rule 2)
- (bb, ab) is added to D (Rule 4)
- (b, a) is added to D (Rule 3)
- $(abb, \varepsilon), (abb, aa), (abb, ab), (abb, bab), (abb, a), (abb, ba)$ are added to D (Rule 4).

We have that $\Upsilon(C) = \{\{\varepsilon, ab\}, \{a\}, \{aa\}, \{aaa\}, \{abb, b\}, \{ba\}, \{bab\}, \{baba\}, \{bb\}, \{bba\}, \{bbab\}\}$. Figure 4(b) presents the updated divergence graph. In bold we represent the tests in $C \cup (K)$ and the edges added to the graph are dashed.

As Condition 4 holds for $\chi = ba$ and $\varphi = a$, Step 4 is executed. Then, (ba, a) is added to C and Rules 1-10 are applied, resulting in the updated graph in Figure 4(b). The following relationships are determined:

- (ab, bab) is added to C (Rule 2)
- (ε, bab) is added to C (Rule 1)
- $(ba, bab), (ba, aa), (bba, aa), (bab, bba)$ are added to D (Rule 4)
- (bb, a) is added to D (Rule 3)
- (bb, ba) is added to D (Rule 5)

In Figure 4(c), we present the updated divergence graph. We have that $\Upsilon(C) = \{\{\varepsilon, ab, bab\}, \{a, ba\}, \{aa\}, \{aaa\}, \{abb, b\}, \{baba\}, \{bb\}, \{bba\}, \{bbab\}\}$.

Now, Condition 4 holds for $\chi = \varepsilon$ and $\varphi = aa$. The execution of Step 4 adds (ε, aa) to C and Rules 1-10 are applied, resulting in the following additional relationships:

- $(ab, aa), (bab, aa)$ are added to C (Rule 1)
- $(a, aaa), (aaa, baba)$ are added to C (Rule 2)
- $(ba, baba), (a, baba)$ are added to C (Rule 1)
- $(aaa, bab), (aaa, ab), (aaa, aa), (aaa, \varepsilon), (aaa, abb), (aaa, bb), (aaa, b), (baba, bab), (baba, ab), (baba, aa), (baba, \varepsilon), (baba, abb), (baba, bb), (baba, b)$ are added to D (Rule 4)

In Figure 4(d), we present the updated divergence graph. We have that $\Upsilon(C) = \{\{\varepsilon, aa, ab, bab\}, \{a, aaa, ba, baba\}, \{abb, b\}, \{bb\}, \{bba\}, \{bbab\}\}$.

Condition 4 holds once more, selecting $\chi = b$ and $\varphi = bb$. Then, (b, bb) is added to C and Rules 1-10 are applied. Now, we have that $C \cup (K)$ is an initialized transition cover for M ; thus T is indeed 3-complete.

The example demonstrates that the proposed sufficient conditions are weaker than the existing ones, as the later cannot establish the test suite completeness.

6.3 Completing User-Defined Test Suites

We now illustrate how the algorithm can be used to extend a user-defined test suite, obtaining a p -complete test suite. Consider again the FSM M in Figure 2 and $p = 3$. In this example, we use a test suite $T = \text{pref}(bbabaa)$, which is derived from a transition tour for M . A simple approach to obtain a 3-complete test suite which contains T is to add to it a 3-complete test suite, generated by an existing known method, thus ignoring the tests already in T . For instance, as mentioned before, the Wp- and H-methods generate the test suite $T_H = \text{pref}(\{aaa, aba, baaa, bbaa\})$. Therefore, the resulting test suite would be $T \cup T_H$, whose length is 25. A smaller test suite of length 16 is obtained by the proposed algorithm, as we show below.

Figure 5(a) presents the divergence graphs constructed during the execution of the algorithm.

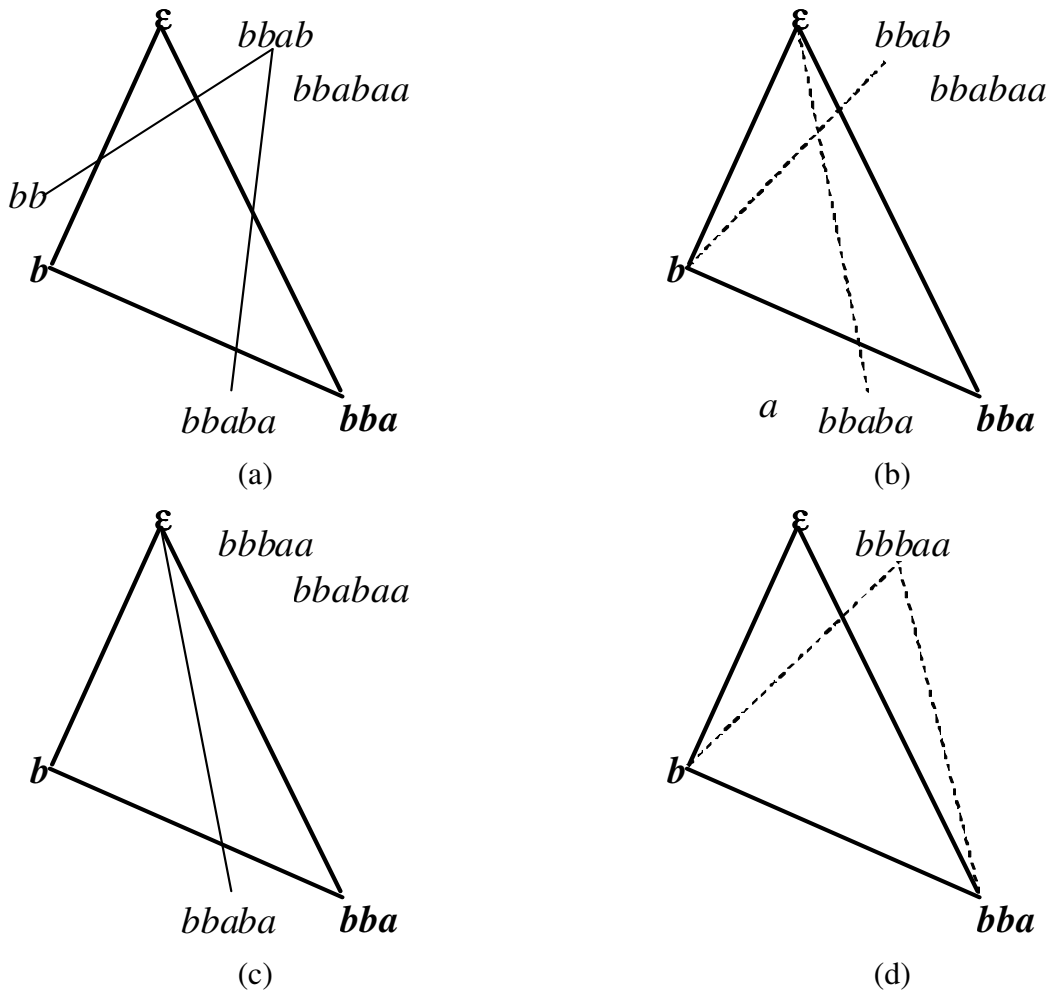


Figure 5 - Divergence Graphs obtained during the generation of $T = \text{pref}(\{a, \text{bbabaab}, \text{bbbaa}\})$

The $K = \{\varepsilon, b, bba\}$ is the only maximal 3-clique in the divergence graph. Step 5 is executed, selecting the transition $(3, b)$ and $\alpha = b$. Then, Step 4 is executed for $\varphi = bb$ and $\chi = b$. For $v = \varepsilon$, we add the test a to T . For $v = bba$, we add $bbba$ to T . After applying Rules 1-10, we obtain the following partition $\Upsilon(C) = \{\{\varepsilon\}, \{a\}, \{b, bb, bbb\}, \{bba, bbba\}, \{bbab\}, \{bbaba\}, \{bbabaa\}\}$. The resulting divergence graph is presented in Figure 5(b) (one test per block).

As $C_{\cup}(K)$ is initialized but is not a transition cover, Step 5 is executed. We select the transition $(2, b)$ and the test $\alpha = bba$. Then, we execute Steps 3 and 4 for $\varphi = bbab$ and $\chi = \varepsilon$. For $v = bba$, we may add test $bbbaa$, so that bba and $bbab$ become T -distinguishable. However, as $(bba, bbba) \in C$, we instead add $bbbaa$, which ensure the T -distinguishability of $bbbaa$ and $bbab$. After applying Rules 1-10 (specifically, Rule 5), $(bba, bbab)$ is added to D , as required. This choice is motivated by the fact that adding $bbbaa$ instead of $bbba$ would not

require an additional reset. After applying Rules 1-10, we obtain $\Upsilon(C) = \{\{\varepsilon, bbab\}, \{b, bb, bbb\}, \{bba, bbba\}, \{a, bbaba\}, \{bbabaa\}, \{bbbaa\}\}$. Figure 5(c) presents the resulting graph.

As $C \cup (K)$ is not a transition cover, we execute Step 5 for the transition $(1, a)$ and $\alpha = bbab$. Steps 3 and 4 are executed for $\varphi = bbaba$ and $\chi = bba$. For $\upsilon = \varepsilon$, it is not necessary to add tests to T . For $\upsilon = b$, we add $bbabaab$ to T . After applying Rules 1-10, we obtain $\Upsilon(C) = \{\{\varepsilon, bbab\}, \{b, bb, bbb\}, \{a, bba, bbba, bbaba\}, \{bbabaa, bbbaa\}\}$. Figure 5(d) presents the resulting graph.

We have that, with $\varphi = bbbaa$ and $\chi = \varepsilon$, (φ, χ) can be added to C without adding new tests. Now we have that $C \cup (K) = T = \text{pref}(\{a, bbabaab, bbbaa\})$ is a transition cover, therefore, is 3-complete, whose length is 16. The example shows how a user-defined test suite (derived using a specification coverage criterion in this particular example) can be extended using the proposed algorithm to achieve the desired fault coverage.

7. CONTRIBUTIONS AND RELATED WORK

In this section, we summarize the contributions of this paper comparing them with the related work in four research directions.

First, test generation for a fault domain containing only implementation FSMs with fewer states than the specification FSM is investigated, addressing the concern of the scalability of complete tests for sizeable specifications. Note that all the existing methods for complete test suite generation provide fault coverage guarantee only for fault domains which necessarily include FSMs with at least as many states as in a specification FSM. As a result, they offer no means to avoid test explosion, while the proposed approach allows the test designer to find a compromise between the fault coverage and the size of a test suite which provides a fault coverage guarantee.

Second, the proposed approach allows incremental test generation; it may start not with an empty test suite as all the existing methods, but with some tests already conceived by the test designer. The problem of test extension has in fact been considered in previous work, namely, [4] and [15]. However, these methods assume that an existing test suite is n -complete for a given specification FSM M which is modified into another FSM. Thus, tests have to be added to the test suite until a test suite complete for the modified machine is obtained. The method of [4] assumes further that the parts of the implementation that correspond to the unmodified parts of the specification have not been changed. The approach of [15] relies on the knowledge of not only a method which produced the initial test suite, but also state identification sequences used in it. In the setting assumed in this paper, no such assumptions are needed.

Third, the proposed test generation method improves the existing methods which start with an empty test suite and terminate with an n -complete test suite (aka checking experiments). These methods rely on “centralized” state identification, in the sense that all sequences which distinguish a state in question from all the other states are applied after a transfer sequence chosen to reach the state (the reader is referred to several surveys available, e.g., [13], and [16]). This is achieved without using the reset input, when there exists a preset distinguishing sequence, as in [20], or an adaptive one, as in [12] and [13]. However, when preset or adaptive distinguishing sequences cannot be found, characterization sets, i.e., state identifiers containing several sequences, and the reset input are usually used to ensure that all of them extend the same transfer sequence from a chosen state cover. Differently from these methods, the proposed method allows state identification in a “distributed” way, meaning that sequences in a state identifier, distinguishing a given state from all the other states, can in fact extend not necessarily the same but various convergent transfer sequences for this state. As a result, not only state identifiers, as in [3], but also transfer sequences can be chosen *on-the-fly*, while an n -complete test suite is constructed. Thus the method exploits new possibilities for overlapping subsequences in a complete test suite and reducing its length. Moreover, since the reset input is not necessarily used each time a given state is to be reached, the number of tests in a test suite, i.e., the number of reset inputs, can thus become a subject for optimization. Examples in Section 6 illustrate a potential save which the proposed method can achieve.

Fourth, the proposed algorithm improves the state-of-the-art in fault coverage analysis. The sufficient conditions for the p -completeness proposed in this paper generalize the existing ones, such as [16], [20], [3], [18], by allowing $p < n$ and further relax them for the case $p = n$. In our recent work [18], we elaborated sufficient conditions for the case of $p = n$ and showed that they are weaker than the sufficient conditions in [3] for checking experiments and those in [20] for checking sequences. Besides being applicable when $p < n$, the conditions proposed in this paper require the existence of an initialized convergence-preserving transition cover, while in [18] not only the convergence, but also divergence is considered for the tests in the initialized transition cover. Moreover, the conditions rely on new possibilities for determining divergence and convergence of tests which are not used in the previous work. Thus, the formulated sufficient conditions are weaker than the existing one.

8. CONCLUSION

In this paper, we considered a problem of incrementally generating tests until the desired level of fault coverage is reached. Solving this problem, we presented sufficient conditions for test suite completeness that are weaker than the ones known in the literature. Based on these conditions, we proposed an algorithm which generates a test suite with complete fault coverage starting with a given set of initial tests, if it is available. The algorithm determines whether the initial test suite already satisfies the sufficient conditions; thus, can also be used for test suite analysis. The possibility of augmenting the fault coverage of test suites also demonstrates the fact that the algorithm allows one to generate tests using specification

coverage as well as fault coverage criteria. Note that these two criteria are often considered as alternatives, where specification coverage criteria are presumed to be more practical.

As forthcoming steps in this work, it is interesting to investigate how the results in this paper can be extended to other fault domains, e.g., to deal with cases where the implementation may have more states than the specification. It is also interesting to experimentally evaluate the scalability of p -complete test suite compared to other test coverage criteria focusing on the specification.

9. REFERENCES

- [1] Binder, R. (2000). *Testing Object-Oriented Systems*. Addison-Wesley, Inc
- [2] Chow, T. S. (1978) Testing software design modeled by finite-state machines. *IEEE Trans. Softw. Eng.* **4**(3), 178-187.
- [3] Dorofeeva, R., El-Fakih, K. and Yevtushenko, N. (2005) An improved conformance testing method. *Formal Techniques for Networked and Distributed Systems*, LNCS 3731, Taipei, Taiwan, 204–218.
- [4] El-Fakih, K., Yevtushenko, N. and Bochmann, G. v. (2004) FSM-based incremental conformance testing methods. *IEEE Transactions on Computers*, **30**(7), 425-436.
- [5] Fraser, G., Weiglhofer, M., and Wotawa, F. (2008) Coverage based testing with test purposes. *Proceedings of the 2008 the Eighth international Conference on Quality Software*, 199-208.
- [6] Fujiwara, S., von Bochmann, G., Khendek, F., Amalou, M., and Ghedamsi, A. (1991) Test selection based on finite state models. *IEEE Trans. Softw. Eng.* **17**(6), 591-603.
- [7] Galil, Z. and Italiano, G. F. (1991) Data structures and algorithms for disjoint set union problems. *ACM Comput. Surv.* **23**(3), 319-344.
- [8] Griggs. J. R. (1983), Lower bounds on the independence number in terms of the degrees. *Journal of Combinatorial Theory, Series B*, **34**, 22-39.
- [9] Hennie, F.C. (1965) Fault-detecting experiments for sequential circuits. *Proceedings of Fifth Annual Symposium on Circuit Theory and Logical Design*. 95–110.
- [10] Jagota, A. and Sanchis L. A. (2001) Adaptive, restart, randomized greedy heuristics for maximum clique. *Journal of Heuristics*, **7**(6), 565-585.

- [11] Karp. R. M. (1976) The probabilistic analysis of some combinatorial search algorithms. *Algorithms and Complexity: New Directions and Recent Results*, 1-19. Academic Press, New York.
- [12] Kohavi, Z., Rivierre, J. A. and Kohavi I. (1973) Machine distinguishing experiments. *The Computer Journal*, **16**, 141-147.
- [13] Lee, D. and Yannakakis, M. (1996) Principles and methods of testing finite state machines - a survey, *Proceedings of the IEEE* , **84**(8), 1090-1123.
- [14] Nebut, C., Fleurey, F., Traon Y. and Jezequel, J. (2006) Automatic test generation: a use case driven approach. *IEEE Trans. Softw. Eng.* **32**(3), 140-155.
- [15] Pap, Z., Subramaniam, M. and Kovacs, G. and Nemeth, G. A. A bounded incremental test generation algorithm for finite state machines. *TestCom/FATES 2007*, LNCS 4581, Tallinn, Estonia, 244–259.
- [16] Petrenko, A., Bochmann, G. v. and Yao, M. (1996) On fault coverage of tests for finite state specifications, *Computer Networks and ISDN Systems*, **29**, 81-106.
- [17] Petrenko, A. and Yevtushenko, N. (2005) Testing from partial deterministic FSM specifications. *IEEE Transactions on Computers*, **54**(9), 1154-1165.
- [18] Simão, A. and Petrenko, A. (2007). *Checking FSM test completeness based on sufficient conditions*. CRIM-07/10-20, Montreal, Quebec, Canada.
- [19] Simão, A. and Petrenko, A. (2009). Comparing FSM test coverage criteria. *IET Software*, (to appear).
- [20] Ural, H., Wu, X. and Zhang, F. (1997) On minimizing the lengths of checking sequences, *IEEE Transactions on Computers*, **46**(1), 93-99.
- [21] Vasilevskii, M. P. (1973) Failure diagnosis of automata. *Cybernetics*, **4**, 653-665.
- [22] Wang, R. L., Tang Z. and Cao, Q. P. (2003) An efficient approximation algorithm for finding a maximum clique using hopfield network learning, *Neural Computation*. **15**, 1605-1619.