Test reactive systems with Büchi automata: acceptance condition coverage criteria and performance evaluation

Bolong Zeng and Li Tan
School of Electrical Engineering and Computer Science
Washington State University
Richland, WA 99354
{*bzeng, litan*}*@wsu.edu*

***Abstract*—Büchi automata have been used to specify and reason linear temporal requirements of reactive systems. A reactive system interacts with its environment constantly, and its executions may be modeled as infinite words. A key question in testing a reactive system is how to make testing *relevant* to the system's requirement, that is, to focus testing on the required behaviors in terms of infinite words. We propose a specification-based testing technique that tests a reactive system with respect to its requirement in Büchi automaton. At the core of our approach is a metric measuring how well a test suite covers the acceptance condition of a Büchi automaton. Acceptance condition is a key element of Büchi automaton defining infinite words. We propose weak and strong variants of coverage metrics and their related test criteria for applications requiring tests of different strengths. We propose a model-checking-assisted algorithm to automate test-case generation for proposed criteria. In addition to debugging a system, our proposed approach may be used for revealing problems with system requirements. We propose an algorithm for refining a requirement encoded in Büchi automaton. Finally, we evaluate the performance of the proposed test criteria using cross-coverage comparison. The result indicates that our approach improves the effectiveness of testing with more efficient test cases targeted for system requirements.**

***Keywords*-reactive systems; specification-based testing; Büchi automaton; model checking.**

I. Introduction

Reactive systems refer to systems constantly interacting with their environments. Many of highly dependable systems are reactive systems: they are expected to interact with their environment (e.g. users, physical objects, etc.) even under adversary condition. Examples of such systems include air traffic control systems and nuclear reactor control systems. To ensure correct functioning of these systems, engineers often deploy a mix of Verification and Validation (V&V) techniques. Two of the most frequently used V&V techniques for reactive systems are testing and formal verification.

Testing examines a system by checking its behavior under a given set of stimuli. Based on the "trial and error" ideology, testing has been an essential part of many V&V processes. In comparison, formal verification refers to a variety of techniques that establish the correctness of a system against its formal requirement, through a mathematically rigid proof. Formal verification techniques such as model checking have received much attention from research community, and they are increasingly adopted in industry as a powerful tool to verify designs of safety-critical systems.

Testing and formal verification are two complementary V&V techniques, each of which has its own set of pros and cons. Testing in general has a better scalability, and it may be applied to both specification and implementation, for example, in the case of model-based testing. Nevertheless, a major drawback of testing, as notably noted by Dijkstra is that testing only shows the existance of bugs, but unable to proof otherwise [1]. In contrast, formal verification techniques such as model checking build a mathematically sound proof for the correctness of a design. Nevertheless, it does not scale nearly as well as testing, and it is often applied to a design or a model extracted from an implemented system, instead of the actual implementation.

A research theme in the V&V field is how to harness the synergy between testing and formal verification. Techniques such as model-checking-based test case generation [2] have been proposed to utilize such a synergy. In this paper we are interested in specification-based testing with temporal requirements. A consequence of the proliferation of formal verification techniques, particularly model checking, is the increasing availability of formal requirements. We want to extend the use of formal requirements into testing. Our objective is two-fold: (1) improve the effectiveness of testing by centering it around formal requirements; and (2) improve the efficiency of testing by developing a model-checking-assisted test case generator for our proposed test criteria.

In this paper we study specification-based test for reactive systems with a formal requirement in Büchi automaton. We choose reactive systems as our subject because of their practical importance in developing safety-critical systems. The essence of a reactive system is its ability of performing infinite executions. A challenge in testing a reactive system is how to test these infinite behaviors. Many of previous works on testing reactive systems (cf. [3], [4]), including those of ours [5], have been focusing on testing finite prefixes of infinite executions. In comparison we are developing a testing technique focusing on features of infinite executions

themselves, that is, temporal patterns exhibited by an infinite execution of a reactive system. Particularly, we consider requirements encoded in Büchi automaton. Büchi automaton, a type of ω-automata accepting infinite words, has been widely used for specifying linear temporal behaviors of a reactive system. It is also instrumental in developing linear temporal model checkers: other formalisms such as Linear Temporal Logic (LTL) are often first translated into Büchi automaton, before used in model checking.

The first-order question in specification-based testing, or any software testing for that matter, is to measure the adequacy of a test suite. We define two coverage metrics measuring how a test suite cover a requirement in Büchi automaton. The metrics define how *thoroughly* a test suite covers the acceptance condition of a Büchi automaton. Test criteria derived from these metrics may guide test-case generation and execution. A Büchi automaton differs from a finite automaton in its acceptance condition, which enables the Büchi automaton to accept infinite words. By focusing on its acceptance condition, our approach centralizes testing efforts on infinite executions of a reactive system.

To improve the efficiency of test generation, we also propose model-checking-assisted test case generation algorithms for proposed test criteria. By utilizing the counterexample producing capability of an off-the-shelf model checker, these algorithms automate the test case generation for reactive systems with Büchi automaton.

By deriving test cases from a reactive system model and its formal requirement, our approach also provides a powerful tool to detect the discrepancy between a model and its requirement. In addition to debugging a system, our approach may also be used for detecting the deficiency of a requirement. The latter provides an opportunity for refining the requirement, and thus reducing the gap between the semantics of a requirement and a system implementation. We propose a technique that automates the property-refinement process, by reusing the information from model-checking-assisted test generation algorithms.

To evaluate the performance of our approach, we conduct a computational study to compare the effectiveness of the proposed acceptance condition coverage criteria against other existing criteria, including traditional test criteria such as branch coverage, as well as the other LTL or Büchi automaton based test criteria introduced in [5], [6]. We select samples from a diversified range of applications, including software engineering (GIOP protocol for constructing middleware), security (Needham public key protocol), and automobile (a fuel system example). The results validate the effectiveness of our approach.

The rest of the paper is organized as follows: Section II prepares the notations used in the rest of the paper; Section III introduces two variants of accepting state combination coverage metrics and criteria for Büchi automata; Section IV describes the model-checking-assisted test case generation algorithms for the proposed criteria; Section V discusses the requirement refinement using the feedback from the model-checking-assisted test case generation; Section VI discusses the result of our computational study on the performance comparison between the new criteria and other existing test criteria; and finally Section VII concludes the paper.

Related Works An important component of our approach is a model-checking-assisted algorithm that utilizes the counterexample mechanism of a model checker to generate test cases. Model checkers are able to generate counterexamples of a model that violates a temporal formula that describes a desired property. Taking advantage of such ability of model checkers to assist test generation has received a significant amount of attention in recent years. One of the core problem in model-checking-assisted test generation is how to shape the test objectives into temporal properties that can be fed to model checkers. Both [7] and [8] have discussed the usage of formal specification in software testing. Gaudel further provided an overview for the conjunction area of testing and model checking, encouraging a more clear and uniformed field for the "industrial actors" [9].

Various works have shown that traditional structural test criteria can be used as the core standard for test generation via model checkers. For instance, Fraser *et al.* show that Modified Condition/Decision Coverage (MC/DC) can be encoded in Computational Tree Logic (CTL) [10], and then be used by a model checker such as NuSMV for generating tests. The authors also compare an array of different test criteria such as logic expression coverage criteria and dataflow criteria [10]. In [2], Hong *et al.* expressed the dataflow criteria in CTL. All these works presented their methods of translating one or more existing structure-based test objectives into a temporal property, such as CTL or LTL.

A key feature of our formal-specification-based testing technique is that it is built upon Büchi automaton, which enables syntax-based as well as semantic-oriented test criteria for linear temporal properties. Our work may be seen as an extension of [11], which proposed test criteria based on syntactical mutations of LTL formulae. To support semantics-oriented testing, we used Büchi automaton as the underlying formalism in [5] and [6], which explore state and transition coverage of Büchi automaton respectively. Büchi automata have the same expressibility as LTL formulae, but they can be minimized to reduce syntax variant while maintaining the same semantics. This work extends [5] and [6] with test criteria on the acceptance condition of a Büchi automaton, which is its key element for modeling infinite executions of a reactive system.

II. Preliminaries

A. Kripke Structures, Traces, and Tests

We model systems as *Kripke structures*. A Kripke structure is a finite transition system in which each state is

labeled with a set of atomic propositions. Semantically atomic propositions represent primitive properties held at a state. Definition II.1 formally defines Kripke structures.

Definition II.1 (Kripke Structures)**.** *Given a set of atomic proposition $\mathcal{A}$, a Kripke structure is a tuple $\langle V, v_0, \rightarrow, \mathcal{V}\rangle$, where V is the set of states, $v_0 \in V$ is the start state, $\rightarrow \subseteq V \times V$ is the transition relation, and $\mathcal{V} : V \rightarrow 2^{\mathcal{A}}$ labels each state with a set of atomic propositions.*

We write $v \rightarrow v'$ in lieu of $\langle v, v'\rangle \in \rightarrow$. We let $a, b, \cdots$ range over $\mathcal{A}$. We denote $\mathcal{A}_\neg$ for the set of negated atomic propositions. Together, $P = \mathcal{A} \cup \mathcal{A}_\neg$ defines the set of *literals*. We let $l_1, l_2, \cdots$ and $L_1, L_2, \cdots$ range over P and 2^P, respectively.

We use the following notations for sequences: let $\beta = v_0 v_1 \cdots$ be a sequence, we denote $\beta[i] = v_i$ for the $i+1$-th element of β, $\beta[i, j]$ for the subsequence $v_i \cdots v_j$, and $\beta^{(i)} = v_i \cdots$ for the i-th suffix of β. A *trace* τ of the Kripke structure $\langle V, v_0, \rightarrow, \mathcal{V}\rangle$ is defined as a maximal sequence of states starting with v_0 and respecting the transition relation $\rightarrow$, i.e., $\tau[0] = v_0$ and $\tau[i-1] \rightarrow \tau[i]$ for every $i < |\tau|$. We also extend the labeling function $\mathcal{V}$ to traces: $\mathcal{V}(\tau) = \mathcal{V}(\tau[0])\mathcal{V}(\tau[1]) \cdots$.

Definition II.2 (Lasso-Shaped Sequences)**.** *A sequence τ is* lasso-shaped *if it has the form $\alpha(\beta)^\omega$, where α and β are finite sequences. $|\beta|$ is the* repetition factor *of τ. The length of τ is a tuple $\langle |\alpha|, |\beta|\rangle$.*

Definition II.3 (Test and Test Suite)**.** *A test is a word on $2^{\mathcal{A}}$, where $\mathcal{A}$ is a set of atomic propositions. A test suite ts is a finite set of test cases. A Kripke structure $K = \langle V, v_0, \rightarrow, \mathcal{V}\rangle$ passes a test case t if K has a trace τ such that $\mathcal{V}(\tau) = t$. K passes a test suite ts iff. it passes every test in ts.*

B. Generalized Büchi Automata

Definition II.4. *A generalized Büchi automaton is a tuple $\langle S, S_0, \Delta, \mathcal{F}\rangle$, in which S is a set of states, $S_0 \subseteq S$ is the set of start states, $\Delta \subseteq S \times S$ is a set of transitions, and the acceptance condition $\mathcal{F} \subseteq 2^S$ is a set of sets of states.*

We write $s \rightarrow s'$ in lieu of $\langle s, s'\rangle \in \Delta$. A generalized Büchi automaton is an ω-automaton, which can accept the infinite version of regular languages. A run of a generalized Büchi automaton $B = \langle S, S_0, \Delta, \mathcal{F}\rangle$ is an infinite sequence $\rho = s_0 s_1 \cdots$ such that $s_0 \in S_0$ and $s_i \rightarrow s_{i+1}$ for every $i \geq 0$. We denote $\inf(\rho)$ for a set of states that appear for infinite times on ρ. A successful run of B is a run of B such that for every $F \in \mathcal{F}$, $\inf(\rho) \cap F \neq \emptyset$.

In this work, we extend Definition II.4 using state labeling approach in [12] with one modification: we label the state with a set of literals, instead of with a set of sets of atomic propositions in [12]. A set of literals is a succinct representation of a set of sets of atomic propositions: let L be a set of literals labeling state s, then semantically s is labeled with a set of sets of atomic propositions $\Lambda(L)$, where $\Lambda(L) = \{A \subseteq \mathcal{A} | (A \supseteq (L \cap \mathcal{A})) \wedge (A \cap (L \cap \mathcal{A}_\neg) = \emptyset)\}$, that is, every set of atomic propositions in $\Lambda(L)$ must contain all the atomic propositions in L but none of its negated atomic propositions. In the rest of the paper, we use Definition II.5 for (labeled) generalized Büchi automata (GBA).

Definition II.5. *A labeled generalized Büchi automaton is a tuple $\langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F}\rangle$, in which $\langle S, S_0, \Delta, \mathcal{F}\rangle$ is a generalized Büchi automaton, P is a set of literals, and the label function $\mathcal{L} : S \rightarrow 2^P$ maps each state to a set of literals.*

A GBA $B = \langle \mathcal{A} \cup \mathcal{A}_\neg, S, S_0, \Delta, \mathcal{L}, \mathcal{F}\rangle$ accepts infinite words over the alphabet $2^{\mathcal{A}}$. Let α be a word on $2^{\mathcal{A}}$, B has a run ρ induced by α, written as $\alpha \vdash \rho$, if and only if for every $i < |\alpha|$, $\alpha[i] \in \Lambda(\mathcal{L}(\rho[i]))$. B accepts α, written as $\alpha \models B$ if and only if B has a successful run ρ such that $\alpha \vdash \rho$.

GBAs are of special interests to the model checking community. Because a GBA is an ω-automaton, it can be used to describe temporal properties of a finite-state reactive system, whose executions are infinite words of an ω-language. Formally, a GBA accepts a Kripke structure $K = \langle V, v_0, \rightarrow, \mathcal{V}\rangle$, denoted as $K \models B$, if for every trace τ of K, $\mathcal{V}(\tau) \models B$. Efficient Büchi-automaton-based algorithms have been developed for linear temporal model checking. The process of linear temporal model checking generally consists of translating the negation of a linear temporal logic property ϕ to a GBA $B_{\neg\phi}$, and then checking the emptiness of the product of $B_{\neg\phi}$ and K. If the product automaton is not empty, then a model checker usually produces an accepting trace of the product automaton, which serves as a counterexample to $K \models \phi$.

III. Accepting State Combination Coverage Criteria

A Büchi automaton differs from a finite automaton in its acceptance condition, which enables a Büchi automaton to accept infinite words. Since we are interested in testing a reactive system, and particularly the temporal patterns of its infinite executions, we focus on covering the acceptance condition of a Büchi automaton. In what follows, we denote $\bigcup \mathcal{F} = F_0 \cup \cdots \cup F_{n-1}$, where $\mathcal{F} = \{F_0, \cdots, F_{n-1}\}$.

Definition III.1 (Accepting State Combination)**.** *Given a Büchi automaton $B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F}\rangle$, an accepting state combination (ASC) C is a minimal set of states such that (i)$C \subseteq \bigcup \mathcal{F}$; (ii)$\forall F \in \mathcal{F}$, $F \cap C \neq \emptyset$.*

We denote $\mathcal{C}(B)$ as the set of the ASCs of B. The coverage metrics are thus about covering these ASCs.

Definition III.2 (Covered Accepting State Combinations)**.** *Given a GBA $B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F}\rangle$, let C be one of B's ASCs,*

1) *A run ρ of B covers C if ρ visits every state of C infinitely often;*
2) *A test t* strongly *covers C on B if t satisfies B and every successful run induced by t on B covers C;*
3) *A test t* weakly *covers C on B if at least one run induced by t on B covers C.*

Intuitively, an ASC is a basic unit for the sets of acceptance states covered by a successful run. That is, any successful run must visit every state of some ASC infinitely often, as stated in Lemma III.3.

Lemma III.3. *Given a Büchi automaton $B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F}\rangle$, ρ is a successful run of B if and only if ρ covers some ASC of B.*

Definition III.2 presents two different ways to cover an ASC, due to the non-deterministic nature of a GBA. In the strong variant, every successful run induced by a successful test is required to visit the ASC infinitely often; and in the weak variant, only one successful run induced by the test is required to visit the ASC infinitely often. By Lemma III.4 the strong coverage criterion subsumes the weak one. Users may pick and choose the type of coverage, depending on the desired strength of testing set forth for an application.

Lemma III.4. *An ASC C of a Büchi automaton B is weakly covered by a test t if C is strongly covered by t.*

Definition III.5 (Strong/Weak ASC Coverage Metric and Criterion)**.** *Given a generalized Büchi automaton $B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F}\rangle$, let $\mathcal{C}(B)$ be the set of B's ASCs, the strong (or weak) ASC coverage metric for a test suite T on B is $\frac{|\delta'|}{|\delta|}$, where $\delta' = \{C \mid t$ strongly (or weakly) covers $C\}$, and $\delta = \mathcal{C}(B)$. T strongly (or weakly) covers δ if and only if $\delta' = \delta$.*

It shall be noted that the number of all the ASCs is significantly smaller than the number of all the possible combinations of acceptance states. The first is bounded by $O(m^n)$ and the latter is bounded by 2^m, where $n = |\mathcal{F}|$ is the cardinality of $\mathcal{F}$ and $m = |\bigcup \mathcal{F}|$ is the number of acceptance states. By focusing on covering ASCs instead of all the combinations of acceptance states, we significantly reduce the complexity of computing our coverage metrics.

IV. Model-Checking-Assisted Test Generation for Accepting States Combination Coverage

To improve the efficiency of test case generation, we develop a model-checking-assisted algorithm for generating test cases under proposed criteria. The algorithm uses the counterexample capability of an off-the-shelf linear temporal model checker to generate test cases. One of the fundamental questions in model-checking-assisted test generation is how to specify test objectives in a formalism acceptable by a model checker. The properties specifying test objectives are often referred to as "trap properties" in the context of model-checking-assisted test generation. In our case, "trap properties" are defined in the form of Büchi automaton. We synthesize a set of "trap (Büchi) automata" from the original Büchi automaton, using graphic transformation techniques.

Definition IV.1 (ASC Excluding Automaton)**.** *Given a Büchi automaton $B = \langle P, S, S_0, \Delta, \mathcal{L}, F \equiv \{F_0, \cdots, F_{n-1}\}\rangle$ and an ASC C, an ASC excluding (ASC-E) GBA is $B_{\overline{C}} = \langle P, S^e, S_0^e \equiv S_0 \times \{\bot\}, \Delta^e, \mathcal{L}^e, \mathcal{F}^e \equiv \{F_0^e, \cdots, F_{n-1}^e\}\rangle$, where,*

1) $S^e = (S \times (C \cup \{\bot\})) - \bigcup_{s \in C}\{\langle s, s\rangle\}$
2) $F_i^e = \{\langle s, u\rangle \mid s \in F_i \wedge u \neq \bot\}$.
3) $\Delta^e = \{(\langle s, u\rangle \rightarrow \langle s', u'\rangle) \mid (s \rightarrow s') \in \Delta \wedge (u = u' \vee (u = \bot))\}$
4) $\mathcal{L}^e(\langle s, u\rangle) = \mathcal{L}(s)$

Intuitively speaking, for a Büchi automaton B and an ASC C, its ASC-E-GBA $B_{\overline{C}}$ accepts precisely B's successful runs, except for those visiting C infinitely often. $B_{\overline{C}}$ does so by extending B with additional copies. To distinguish these copies, the states of the original copy (denoted as $B_\bot$) is indexed by the symbol $\bot$, whereas the states of each additional copy (denoted as $B_{\neg s}$) are indexed by a state $s \in C$. $B_{\neg s}$ inherits all the states from B except for s, the very state indexing B (i.e. $\langle s, s\rangle$ in Definition IV.1.(1)). Intuitively, $B_{\neg s}$ accepts all the successful runs of B, except for those visiting the indexing state s. Each copy $B_{\neg s}$ retains the transitions from B (except for, of course, the ones associating with the indexing state s, which is not in $B_{\neg s}$). In addition, for each transition of the original copy $B_\bot$, say $\langle s, \bot\rangle \rightarrow \langle s', \bot\rangle$, we create $|C|$ copies of that transition, each of which replaces the destination node $\langle s', \bot\rangle$ with its counterpart in a copy $B_{\neg t}$ indexed by a state $t \in C$ (Definition IV.1.(3)). Formally, for each transition $\langle s, \bot\rangle \rightarrow \langle s', \bot\rangle$, we add more transitions $\delta = \bigcup_{t \in C}\{\langle s, \bot\rangle \rightarrow \langle s', t\rangle\}$. We refer to these new transitions as "bridging" transitions. Note that these bridging transitions go one-way only, that is, they jump from the original copy $B_\bot$ to a copy $B_{\neg t}$, where $t \in C$. There are no transitions linking $B_{\neg t}$ back to $B_\bot$. As a final touch, only the copies indexed by the states of C, not the original one, retain the acceptance condition. Since every additional copy $B_{\neg s}$ misses its indexing state s, it implies that the indexing state is not part of the acceptance condition of $B_{\neg s}$.

It follows from the construction of $B_{\overline{C}}$ that a successful run ρ of $B_{\overline{C}}$ must satisfy the following conditions: (1) it starts at $B_\bot$ (i.e. the start states S_0^e in Definition IV.1) and can spend only a finite number of steps in $B_\bot$, since $B_\bot$ does not have an acceptance state (Definition IV.1.(2)); (2) at some point, ρ will make a non-deterministic choice to take a bridging transition to one of the copies indexed by a state $s \in C$, say $B_{\neg s}$, and satisfy $B_{\neg s}$'s acceptance condition. Clearly ρ is also a successful run of the original GBA B,

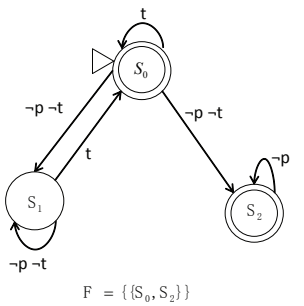

Figure 1. A general Büchi automaton representing the LTL property $\mathbf{G}(\neg t \Rightarrow ((\neg p \ \mathbf{U} \ t) \vee \mathbf{G}\neg p))$.

since each state on ρ may be mapped back to a state of B, and the acceptance condition of the copy of B accepting ρ is a subset of the acceptance condition of B. In addition, because $B_{\neg s}$ does not have s (Definition IV.1.(1)), ρ cannot visit s infinitely often. Furthermore, no matter which copy the ρ jumps to, there is no way that ρ can visit every state of C infinitely often, since there is one state of C being missed in that copy, i.e., its indexing state. It follows that a successful run ρ' of B becomes a successful run of its ASC-E-GBA only if ρ' does not visit every state of C infinitely often. Interested users may refer to our full paper ([13]) for the proof of Theorem IV.2.

Theorem IV.2. *Given a GBA $B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F}\rangle$ and an ASC C, $B_{\overline{C}} = \langle P, S^e, S_0^e, \Delta^e, \mathcal{L}^e, \mathcal{F}^e\rangle$ be the ASC-E-GBA for B and C, then a test t satisfies $B_{\overline{C}}$ if and only if B has a successful run ρ such that $t \vdash \rho$ and $\inf(\rho) \not\supseteq C$.*

As an example, consider a GBA in Figure 1. The GBA represents LTL property $\phi = \mathbf{G}(\neg t \Rightarrow ((\neg p \ \mathbf{U} \ t) \vee \mathbf{G}\neg p))$, a temporal requirement used with the GIOP model [14] in our experimental study. ϕ's semantics is explained in Section VI. Since its acceptance condition $\{\{s_0, s_2\}\}$ contains only one set of states, its ACSs are the singleton set of each acceptance state, that is, $\{s_0\}$ and $\{s_2\}$. Figure 2 gives an ASC excluding automaton $B_{\overline{\{s_0\}}}$ with respect to the ASC $\{s_0\}$. $B_{\overline{\{s_0\}}}$ has two copies of B: the original copy $B_\perp$ and the copy indexed by s_0, the only state in C. Note that the indexing state itself s_0 (i.e. $\langle s_0, s_0\rangle$) and its transitions are removed from the copy $B_{\overline{\{s_0\}}}$. These are represented by the dashed circle and lines in Figure 2. The highlighted solid links represent bridging transitions linking from the original copy to the copy indexed by s_0. Since the only acceptance state, $\langle s_2, s_0\rangle$ exists in the copy $B_{\neg s_0}$, a successful run of $B_{\overline{\{s_0\}}}$ must visit s_2 (in the form of $\langle s_2, s_0\rangle$), not s_0 (in the form of $\langle s_0, s_0\rangle$), infinitely often.

Algorithm 1 generates a test suite strongly covering all the ASCs of a Büchi automaton B. It makes use of ASC excluding automata. For each of B's ASCs, Algorithm 1

Algorithm 1 TestGen_SC($B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F}\rangle$, $K_m = \langle \mathcal{S}, s_0, \rightarrow, \mathcal{V}\rangle$)

Require: B is a GBA, K_m is a system model, and K_m satisfies B

Ensure: Return a test suite ts such that ts strongly covers every ASC of B and K_m passes ts. Return $\emptyset$ if such a test suite is not found;

```
1: C(B) = ASC_Gen(B);
2: for every ASC C ∈ C(B) do
3:    B_C̄ = ⟨P, S × C_s ∪ ∅, S_0 × ∅, Δ_e, L_e, F_e⟩;
4:    τ = MC_isEmpty(¬B_C̄, K_m));
5:    if |τ| ≠ 0 then
6:       ts = ts ∪ {V(τ)}
7:    else
8:       return ∅;
9:    end if
10: end for
11: return ts;
```

constructs an ASC excluding GBA $B_{\overline{C}}$ with respect to C. ASC_Gen is a sub-routine computing all the ASCs for a GBA. We skip the details of ASC_Gen due to the space limit. Interested users may refer to our full paper for the details [13]. The algorithm uses a model checker to search for a successful run τ on the production of $\neg B_{\overline{C}}$ and K_m, and τ is a successful run of $\neg B_{\overline{C}}$ accepting K_m. $MC\_isEmpty$ refers to the emptyness checking algorithm in an off-the-shelf linear temporal model checker. If a run exists, it returns with a test set containing $t = \mathcal{V}(\tau)$, which is a word accepted by $\neg B_{\overline{C}}$. Consequently, t cannot be accepted by $B_{\overline{C}}$. Note that τ is a successful run of the production of B and K_m, therefore based on Theorem IV.2, for every successful run ρ that $t \vdash \rho$, $\inf(\rho) \supseteq C$. Based on Definition III.2, ts is a test suite that strongly covers C.

Theorem IV.3. *If the test suite ts returned by Algorithm 1 is not empty, then (i) K_m passes ts and (ii) ts strongly covers all the ASCs of B.*

Compared with constructing an ASC excluding automaton, constructing a Büchi automaton accepting only the runs that weakly covers an ASC is relatively straightforward: the new automaton may be obtained by removing from the acceptance condition the states *not* in the ASC, that is, replacing the acceptance condition with C. Definition IV.4 describes the process.

Definition IV.4 (ASC Marking Automaton)**.** *Given a GBA $B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F}\rangle$ and an ASC C, $B_C = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F}_C\rangle$ is the ASC-Marking (ASC-M) Büchi automaton for B with respect to C, in which $\mathcal{F}_C = \{F \cap C | F \in \mathcal{F}\}$.*

Clearly $L(B_C) \subseteq L(B)$, since the acceptance condition

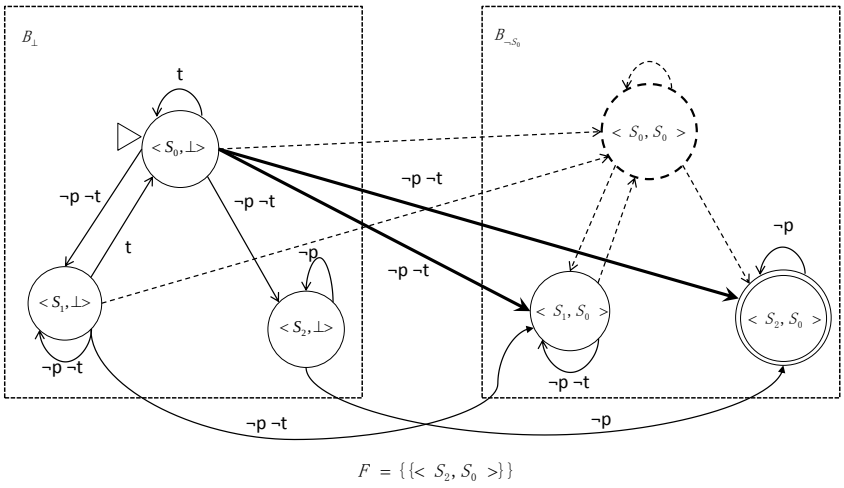

Figure 2. An ASC excluding general Büchi automaton for the ASC $\{s_0\}$ of the GBA in Figure 1.

of B_C is a refinement of that of B, that is, $\forall F \in \mathcal{F}, \exists F' \in \mathcal{F}_C, (F' \subseteq \mathcal{F}')$ and $\forall F' \in \mathcal{F}_C, \exists F \in \mathcal{F}, (F' \subseteq F)$. Note that, $\mathcal{F}_C$ in Definition IV.4 is a subset of $2^C - \emptyset$. By Definition III.1, C has to be a minimal set of states that $\forall F \in \mathcal{F}$, $F \cap C \neq \emptyset$. Combine these two conditions, it is straightforward $\bigcup \mathcal{F}_C = C$. Based on Definition II.4, this means that for a run to be successful on B_C, all states in C must be visited infinitely often. Therefore we rewrite the acceptance condition for B_C as $\mathcal{F}_C = \{\{s\} \mid s \in C\}$, i.e., a set of singleton sets of states in C. For the rest of the paper, we consider ASC-M-GBA to be defined with the rewritten acceptance condition.

Algorithm 2 generates tests that weakly cover the ASC C. We construct an ASC-M-GBA B_C in Algorithm 2, and then search for a successful run on the product of B_C and the system model K_m. If such run τ exists, the test case $t = \mathcal{V}(\tau)$ is then added to ts and return as the singleton test suite. Since $t \in L(B_C)$ and $L(B_C) \subseteq L(B)$, $t \in L(B)$. By Definition IV.4 and Definition III.2, since τ is a successful run of B_C that weakly covers C on B, therefore t is a test case that weakly covers C on B.

Theorem IV.5. *If the test suite ts returned by Algorithm 2 is not empty, then (i) K_m passes ts and (ii) ts weakly covers all the ASCs of B.*

Due to the space limit, we include our proof for Theorem IV.3 and Theorem IV.5 in our full paper ([13]).

Algorithm 2 TestGen_WC($B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F} \rangle$, $K_m = \langle \mathcal{S}, s_0, \rightarrow, \mathcal{V} \rangle$)

Require: B is a GBA, K_m is a system model, and K_m satisifies B.

Ensure: Return a test suite ts such that ts weakly covers every ASC in $\mathcal{C}(B)$ and K_m passes ts. Return $\emptyset$ if such a test suite is not found;

1: $\mathcal{C}_{\perp} = ASC\_Gen(B)$;
2: **for** every ASC $C \in \mathcal{C}_{\perp}$ **do**
3: $B_C = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F}_C \rangle$, where $\mathcal{F}_C = \{\{s\} \mid s \in C\}$;
4: $\tau = MC\_isEmpty(B_C, K_m))$;
5: **if** $|\tau| \neq 0$ **then**
6: $ts = ts \cup \{\mathcal{V}(\tau)\}$;
7: **else**
8: **return** $\emptyset$;
9: **end if**
10: **end for**
11: **return** ts;

V. ASC-Induced Property Refinement

ASC coverage metrics measure the conformance of a design against a formal requirement in Büchi automaton. Lacking of ASC coverage may be contributed either by bugs in the design, or by the deficiency of the requirement, or sometimes by both. We develop an algorithm that identifies the deficiency of the requirement and refines the requirement, using the information collected from test case

generations (Algorithm 2).

We consider the refinement in terms of language inclusion, that is, if the language of an automaton B' is a subset of that of B, we refer to B' as a refinement of B. Given a Kripke structure K representing a system with its requirement as GBA B, we develop an algorithm to refine B if not every ASC of B can be weakly covered w.r.t. K.

Given a Kripke structure K_m as a system model and a GBA $B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F} \rangle$ as its requirement, the basic steps of refining B w.r.t. B are described below:

1) Identify the set of ASCs $\mathcal{C} = \{C_0, \cdots, C_n\}$ of B that are weakly covered w.r.t. K_m. $\mathcal{C}$ may be identified by Algorithm 2 during the test case generation;
2) Produce an automaton $B_{\mathcal{C}} = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F}_{\mathcal{C}} \rangle$, where $\mathcal{F}_{\mathcal{C}} = \{F \cap (\bigcup \mathcal{C}) \mid F \in \mathcal{F}\}$

Algorithm 3 TestGen_RefineWC($B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F} \rangle$, $K_m = \langle \mathcal{S}, s_0, \rightarrow, \mathcal{V} \rangle$)

[TestGen_RefineWC()]

Require: B is a GBA, K_m is a system model, and K_m satisfies B.

Ensure: Return a test suite ts such that ts weakly covers all ASCs that can be covered in $\mathcal{C}_\perp$ and K_m passes ts. Also returns a Büchi automata with a refined acceptance condition;

1: $\mathcal{C}_\perp = ASC\_Gen(B)$;
2: **for** every ASC $C \in \mathcal{C}_\perp$ **do**
3: $B_C = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F}_C \rangle$, where $\mathcal{F}_C = \{\{s\} \mid s \in C\}$;
4: $\tau = MC\_isEmpty(B_C, K_m))$;
5: **if** $|\tau| \neq 0$ **then**
6: $ts = ts \cup \{\mathcal{V}(\tau)\}$
7: $\mathcal{C} = \mathcal{C} \cup \{C\}$
8: **end if**
9: **end for**
10: $B_{\mathcal{C}} = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F}_{\mathcal{C}} \equiv \{F \cap (\bigcup \mathcal{C}) \mid F \in \mathcal{F}\} \rangle$
11: **return** ts and $B_{\mathcal{C}}$;

For example, consider a Büchi automaton B with an acceptance condition $\mathcal{F} = \{F_0, F_1, F_2\}$, in which $F_0 = \{s_1, s_2, s_4\}$, $F_1 = \{s_2, s_3, s_4\}$, $F_2 = \{s_1, s_3, s_4\}$. The ASCs for B would be $C_0 = \{s_1, s_2\}, C_1 = \{s_2, s_3\}, C_2 = \{s_1, s_3\}$ and $C_3 = \{s_4\}$. Assume that only C_0 and C_2 can be weakly covered w.r.t. a system K, we can then refine B to $B_{\mathcal{C}}$, where $B_{\mathcal{C}}$'s acceptance condition is $\{\{s_1, s_2\}, \{s_2, s_3\}, \{s_1, s_3\}\}$.

The entire process of property refinement may be automated by extending test generation algorithm, as described in Algorithm 3. Clearly $L(B_{\mathcal{C}}) \subseteq L(B)$, since the acceptance condition of $B_{\mathcal{C}}$ is a refinement of the acceptance condition of B. Moreover, by the construction of $\mathcal{F}_{\mathcal{C}}$, $B_{\mathcal{C}}$ contains all the ASCs of B that can be weakly covered by some tests passed by K_m. Theorem V.1 states that the refined automaton, $B_{\mathcal{C}}$, is still satisfied by K_m. In other words, by refining B to $B_{\mathcal{C}}$, we obtain a "restricted" version of the property that more closely specifies a requirement for K_m. Due to the space limit, we omit the proof of Theorem V.1. Interested readers may refer to the full length of the paper for the details [13].

Theorem V.1. *Given a GBA B and a Kripke structure K_m such that $K_m \models B$, let $B_{\mathcal{C}}$ be the GBA returned by $TestGen\_RefineWC(B, K_m)$, then, (i) $L(B_{\mathcal{C}}) \subseteq L(B)$ and (ii) $K_m \models B_{\mathcal{C}}$.*

VI. Experiments

To assess the performance of the proposed criteria, we perform a computational study using cross-coverage measurement. Cross-coverage measurement compares the effectiveness of test criteria with respect to each other. It measures how well a test suite generated for one criterion A covers another criterion B. A higher percentage for A indicates that A would be more effective in "covering" B, and hence may induce a more effective test suite than B. To obtain a close-to-reality measurement, we select testing subjects from a diversified range of applications. The first subject is a model of the general Inter-ORB Protocol (GIOP) from the area of software engineering. GIOP is a key component of the Object Management Group (OMG)'s Common Object Request Broker Architecture (CORBA) specification [14]. The second model is a model of the Needham-Schroeder public key protocol from the area of computer security. The Needham-Schroeder public key protocol intends to authenticate two parties involving with a communication channel. Finally, our third subject is a model of a fuel system from the area of control system. The model is translated by Sabina Joseph [15] from a classic Simulink demo model[16].

Each model comes with one or more linear temporal properties that specify its behavioral requirements. We selected a representative property for each model to be used in our computational study. For the GIOP model, the property models the behaviors of a recipient during communication. The LTL property for the Needham-Shroeder public-key protocol is a liveness property requiring that an initiator can only send messages after a responder is up and running. Finally, the properties for the fuel system checks that under abnormal conditions, the system's fault tolerant mechanism functions properly. Interested readers may refer to the full paper [13] for the details of these models and properties.

For performance comparison, we select several traditional as well as newer formal-specification-based test criteria. Based on the coverage for outcomes of a logic expression (c.f. [17]), branch coverage (BC) is one of the most commonly-used structural test criteria. We include both transition and state variations of strong coverage criteria (SC/strong, TC/strong) and weak coverage criteria

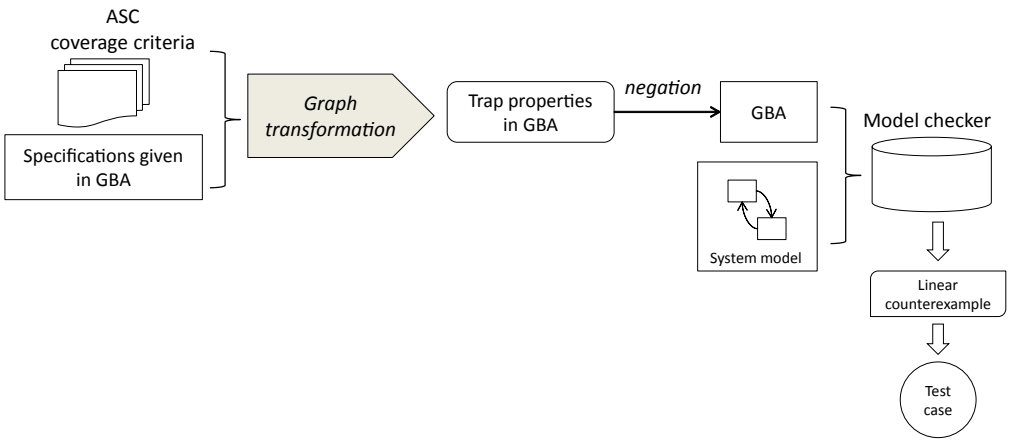

Figure 3. The workflow of model-checking-assisted test case generation under ASC coverage criteria for Büchi automaton.

(SC/weak, TC/weak) for Büchi automaton[5], [6]. We also include a property-coverage criterion (PC) for Linear Temporal Logic (LTL) [11]. In our experiment, the performances of these criteria, and two ASC coverage criteria (ACC/stong and ACC/weak) are compared with each other.

In [18] we developed a tool to compare the effectiveness of test criteria in context of model-checking-assisted test case generation. We extend the tool to support the proposed ASC criteria. We use GOAL [19] to perform graph transformation that constructs ASC-E-GBAs and ASC-M-GBAs. We use SPIN [20] as the underlying model checker. Figure 3 shows the workflow of generating tests with model checking under ASC coverage criteria for Büchi automaton. Details of this procedure and our previous computational study can be found in [21], with a different set of subjects and criteria.

Table I shows the cross-coverage measurement. The number in each cell indicates the precentile coverage of a test suite generated for the criterion on the row with respect to the criterion on the column. Numbers on diagonal cells (marked with parentheses) represent the coverage of a test suite generated for the same criterion. A less-than perfect coverage on these diagonal cells indicates potential deficiency of a model and/or a requirement. For instance, the test suite of the fuel system model for ACC/weak may only reach 67% coverage upon all the ASCs because the property has an ASC that cannot be covered.

The results show that our proposed ASC coverage criteria, especially the strong variant, exhibits a competent performance. It is on par with the other GBA based criteria, and falls only barely behind the branch coverage criterion. It shall be noted that the test suite generated for the branch coverage criterion is much larger than those generated for the property-based test criteria, including our ASC criteria, indicating that the property-based criteria can potentially make testing more effective by producing smaller and more focused test suites. Interested readers may refer to the full paper [13] for the comparision on the measurements of generated test cases, including sizes, trace lengths and time.

The test suites for ASC coverage criteria, while competent, did not achieve full branch coverage. This is because we only use one temporal property for each model, and the property does not cover all the functional aspects of the models. For instance, the property for the GIOP model specifies the recipient's behavior at "waiting" or "receiving" modes, it does not concern other modes of operations. Therefore, the generated test suite skips some code segments, which leads to a less-than perfect branch coverage.

This observation leads to an important feature of property-based test criteria, including our ASC coverage criteria. That is, the performance of these criteria are heavily influenced by the quality of underlying requirement. A thorough requirement touching more aspects of a model may result in a test suite with better quality. In Section V we capitalized this observation via our ASC-induced property refinement. Alternatively, a more complete set of temporal properties that address multiple aspects of a model could also greatly improve the performance on this part.

Finally, the results above also establish that ASC coverage criteria correlate well with the Büchi automaton-based state and transition coverage criteria. The strong variant performs similarly as the state and transition coverage criteria, while the weak variant exhibits the result that are somewhat in between. Intuitively, a path covering an ASC also covers the states and transitions on it. Although these coverage criteria cover some elements of Büchi automata, ASC criteria are not merely an extension of Büchi automaton-based state and transition coverage criteria. ASC criteria concern temporal patterns of infinite words, whereas state and transition coverage criteria can only test temporal patterns of finite prefix of infinite words. The essence of Büchi automaton is its ability of modeling temporal patterns of infinite words, not just their finite prefixes. Therefore, ASC criteria lend engineers

Table I
CROSS-COVERAGE COMPARISON RESULTS

	GIOP							
	BC	PC	SC/strong	SC/weak	TC/strong	TC/weak	ACC/strong	ACC/weak
BC	(77%)	75%	100%	100%	100%	100%	100%	100%
PC	66%	(75%)	100%	100%	100%	100%	100%	100%
SC/strong	66%	75%	(100%)	100%	100%	100%	100%	100%
SC/weak	66%	75%	100%	(100%)	100%	100%	100%	100%
TC/strong	66%	75%	100%	100%	(100%)	100%	100%	100%
TC/weak	66%	75%	100%	100%	100%	(100%)	100%	100%
ACC/strong	66%	75%	100%	100%	100%	100%	(100%)	100%
ACC/weak	66%	75%	100%	100%	100%	100%	100%	(50%)

	Needham Protocol							
	BC	PC	SC/strong	SC/weak	TC/strong	TC/weak	ACC/strong	ACC/weak
BC	(86%)	100%	100%	100%	100%	100%	100%	100%
PC	47%	(100%)	100%	100%	100%	100%	100%	100%
SC/strong	47%	100%	(100%)	100%	100%	100%	100%	100%
SC/weak	28%	0%	0%	(100%)	0%	100%	0%	100%
TC/strong	47%	100%	100%	100%	(100%)	100%	100%	100%
TC/weak	40%	0%	0%	100%	0%	(100%)	0%	100%
ACC/strong	47%	100%	100%	100%	100%	100%	(100%)	100%
ACC/weak	30%	0%	0%	100%	0%	100%	0%	(100%)

	Fuel System							
	BC	PC	SC/strong	SC/weak	TC/strong	TC/weak	ACC/strong	ACC/weak
BC	(82%)	25%	75%	50%	86%	33%	67%	67%
PC	78%	(100%)	50%	50%	29%	33%	67%	67%
SC/strong	75%	100%	(50%)	50%	29%	33%	67%	67%
SC/weak	64%	25%	75%	(50%)	86%	33%	67%	67%
TC/strong	75%	100%	100%	50%	(29%)	33%	67%	67%
TC/weak	67%	25%	75%	50%	86%	(33%)	67%	67%
ACC/strong	75%	100%	50%	50%	29%	33%	(67%)	67%
ACC/weak	55%	25%	75%	50%	86%	33%	67%	(67%)

the ability of testing the essence of formal requirements in Büchi automaton. It shall also be noted that the ASC criteria do not subsume their counterparts in state and transition coverage. For example, in the fuel system model, the test suite for ASC/weak criterion only achieves 55% of coverage over the branch coverage, lower than both SC/weak and TC/weak. These criteria are complementary to each other, and may be used in combination to achieve desired coverage on requirements in Büchi automaton.

VII. CONCLUSIONS

We proposed a specification-based approach for testing reactive systems with requirement expressed in Büchi automata. At the core of our approach are two variants of accepting state combination coverage metrics measuring how well a test suite covers the acceptance condition of a Büchi automaton. By measuring the coverage of the acceptance condition of a Büchi automaton encoding a system requirement, these metrics and criteria enable testing temporal patterns of infinite executions of a reactive system. To facilitate the practical use of the proposed test criteria, we developed test-case generation algorithms for these proposed criteria. The algorithms utilize the counterexample generation capability of an off-the-shelf model checker to automate the process of test-case generation for the proposed criteria.

It shall be noted that, although specification-based testing with automata has been studied before (c.f.[22]), the specification concerned in most of these previous works is a system design modeled in a finite automaton. In comparison, we focus on a behavioral requirement modeled in Büchi automaton. Moreover, existing approaches for specification-based testing for reactive systems focus on the finite prefixes of its infinite executions. In contrast, our approach works with temporal patterns of its infinite executions. All of these make our approach more effective in testing the temporal patterns of a reactive system.

Our approach tests the conformance of a reactive system to its requirement in Büchi automaton. It may be used for revealing the deficiency of the system as well as its requirement. We discussed how our approach may be used to debug and even refine the requirement, using the information from the model-checking-assisted test case generation. We proposed a property-refinement algorithm that automated the process of property refinement.

To assess the effectiveness of our approach, we carried out an extended computational study using cross-coverage measurement as a tool. We measure how well a test suite produced under one test criterion covers another test criterion. We also select subjects from a diversified range of applications. The experimental results indicate that our criteria exhibit competent performance over existing test criteria. They are particularly effective at reducing the size of test suites, making testing more targeted and efficient. For the future work, we want to extend our approach to more complex requirements, such as those in μ-calculus.

REFERENCES

[1] O.-J. Dahl, E. W. Dijkstra, and C. Hoare, *Structured Programming*, ser. A.P.I.C. Studies in Data Processing. Academic Press, 1972, vol. 8.

[2] H. S. Hong, I. Lee, O. Sokolsky, and H. Ural, "A temporal logic based theory of test coverage and generation," in *TACAS'02*, 2002.

[3] J. Tretmans, "Model based testing with labelled transition systems," in *Formal methods and testing*. Springer, 2008, pp. 1–38.

[4] K. Meinke and M. A. Sindhu, "Incremental learning-based testing for reactive systems," in *Tests and Proofs*. Springer, 2011, pp. 134–151.

[5] L. Tan, "State Coverage Metrics for Specification-Based Testing with Büchi Automata," in *5th International Conference on Tests and Proofs*, ser. Lecture Notes in Computer Science. Zurich, Switzerland: Springer Verlag, 2011.

[6] L. Tan and B. Zeng, "Specification-Based Testing with Buchi Automata: Transition Coverage Criteria and Property Refinement," in *International Conference on Information Reuse and Integration*. IEEE, August 2014.

[7] R. M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Lüttgen, A. J. H. Simons, S. Vilkomir, M. R. Woodward, and H. Zedan, "Using formal specifications to support testing," *ACM Comput. Surv.*, vol. 41, no. 2, pp. 9:1–9:76, Feb. 2009. [Online]. Available: http://doi.acm.org/10.1145/1459352.1459354

[8] M.-C. Gaudel, "Software testing based on formal specification," in *Testing Techniques in Software Engineering, Second Pernambuco Summer School on Software Engineering, PSSE 2007, December 3-7, 2007, Revised Lectures*, ser. Lecture Notes in Computer Science, P. Borba, A. Cavalcanti, A. Sampaio, and J. Woodcock, Eds., vol. 6153. Springer, 2010, pp. 215–242.

[9] ——, "Checking models, proving programs, and testing systems," in *TAP 2011 proceedings*, ser. Lecture Notes in Computer Science, M. Gogolla and B. Wolff, Eds., vol. 6706. Springer, 2011, pp. 1–13.

[10] G. Fraser and A. Gargantini, "An evaluation of model checkers for specification based test case generation," in *ICST '09: Proceedings of the 2009 International Conference on Software Testing Verification and Validation*. Washington, DC, USA: IEEE Computer Society, 2009.

[11] L. Tan, O. Sokolsky, and I. Lee, "Specification-based Testing with Linear Temporal Logic," in *the proceedings of IEEE Internation Conference on Information Reuse and Integration (IRI'04)*. IEEE society, 2004.

[12] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper, "Simple on-the-fly automatic verification of linear temporal logic," in *Protocol Specification Testing and Verification*. Chapman & Hall, 1995.

[13] B. Zeng and L. Tan, "Test reactive systems with Büchi automata: acceptance condition coverage critera and performance evaluation," 2015, (The full version with proofs). [Online]. Available: http://www.tricity.wsu.edu/~litan/papers/acccovfull.pdf

[14] M. Kamel and S. Leue, "Formalization and validation of the General Inter-ORB Protocol (GIOP) using PROMELA and SPIN," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 2, no. 4, Mar. 2000.

[15] S. Joseph, "Fault-Injection through Model Checking via Naive Assumptions about State Machine Synchrony Semantics," Master's thesis, West Virginia University, Morgantown, West Virginia, 1998.

[16] SIMULINK, "Dynamic system simulation for matlab, the mathworks," January 1997.

[17] P. C. Jorgensen, *Software Testing: A Craftsman's Approach*, 1st ed. Boca Raton, FL, USA: CRC Press, Inc., 1995.

[18] B. Zeng and L. Tan, "Test Criteria for Model-Checking-Assisted Test Case Generation: A Computational Study," in *International Conference on Information Reuse and Integration*. IEEE, 2012.

[19] Y.-k. Tsay, Y.-f. Chen, M.-h. Tsai, K.-n. Wu, and W.-c. Chan, "GOAL : A Graphical Tool for Manipulating B uchi Automata and Temporal Formulae," in *13th Tools and Algorithms for the Construction and Analysis of Systems*, vol. 02. Springer, 2007, pp. 466–471.

[20] G. J. Holzmann, "The model checker SPIN," *IEEE Transactions on Software Engineering*, vol. 23, May 1997.

[21] B. Zeng and L. Tan, "A unified framework for evaluating test criteria in model-checking-assisted test case generation," *Information Systems Frontiers*, April 2013. [Online]. Available: http://dx.doi.org/10.1007/s10796-013-9424-y

[22] S. Fujiwara, G. von Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test selection based on finite state models," *IEEE Trans. Softw. Eng.*, vol. 17, no. 6, Jun. 1991.